



Ca' Foscari  
University  
of Venice

Master's Degree programme

Computer Science and Information Technology - CM90

**CronFrame:  
A Macro Annotation Cron Job Framework  
with Web Server and CLI Tool written in Rust**

**Supervisor**

Ch. Prof. Pietro Ferrara

**Assistant supervisor**

Ch. Prof. Gianluca Caiazza

**Graduand**

Antonio Cimino

Matriculation Number 897613

**Academic Year**

2023 / 2024

# Contents

<b>1. Introduction .....</b>	<b>5</b>
<b>2. Scheduling of Cron Jobs .....</b>	<b>7</b>
2.1. Definition of a Cron Job .....	7
2.1.1. The Schedule: Cron Expression .....	7
2.1.2. The Job: Commands and Functions .....	9
2.2. Common Issues of Cron Jobs Usage .....	9
2.3. Execution of a Cron Job .....	10
<b>3. The Rust Language .....</b>	<b>12</b>
3.1. The Macro System .....	13
3.1.1. Declarative Macros .....	13
3.1.2. Procedural Macros .....	14
3.1.2.1. Function-Like Macros .....	14
3.1.2.2. Derive Macros .....	14
3.1.2.3. Attribute Macros .....	15
3.2. Developing Macros .....	15
3.3. The Syn Crate .....	15
3.4. The Quote Crate .....	17
<b>4. The Framework .....</b>	<b>19</b>
4.1. Framework Instance .....	20
4.2. Annotation Macros .....	21
4.2.1. Macro: cron .....	21
4.2.2. Macro(s) : cron_obj and cron_impl .....	22
4.2.3. Macro: fn_job .....	24
4.2.4. Macro: mt_job .....	25
4.3. The JobBuilder Type .....	26
4.4. The CronJob Type .....	27
4.5. The CronFrame Type .....	29
4.6. The Web Server: Rocket .....	30
4.7. The Logger: log4rs .....	33
4.8. The Configuration File .....	34
4.9. Macro Expansions .....	35
4.9.1. Macro Expansion: cron .....	36
4.9.2. Macro Expansion: cron_obj, cron_impl, fn_job, mt_job .....	37
<b>5. The CLI Tool .....</b>	<b>48</b>
5.1. Commands .....	48
5.2. Tool Information .....	49
5.3. CLI Jobs .....	50
<b>6. Documentation and Tutorial .....</b>	<b>51</b>
6.1. Framework Tutorial .....	51
6.1.1. Setting Up CronFrame .....	51
6.1.1.1. Set up from Crates.io .....	51
6.1.1.2. Set up from GitHub Repo .....	52
6.1.2. Using CronFrame .....	52
6.2. Running Examples .....	59

6.3. Weather Alert Scenario - Example .....	60
6.4. CLI Tool Tutorial .....	64
6.4.1. Installing CronFrame .....	64
6.4.1.1. Installing from Crates.io .....	64
6.4.1.2. Installing from GitHub Repo .....	64
6.4.2. Using CronFrame CLI .....	64
6.4.2.1. start command .....	66
6.4.2.2. run command .....	66
6.4.2.3. add command .....	66
6.4.2.4. load command .....	66
6.4.2.5. scheduler command .....	66
6.4.2.6. shutdown command .....	67
<b>7. Testing .....</b>	<b>68</b>
7.1. Available Test Modules .....	69
7.1.1. Tests in the Global Module .....	69
7.1.2. Tests in the Function Module .....	69
7.1.3. Tests in the Method Module .....	70
7.2. Testing Peculiarities .....	70
<b>8. Project Reception .....</b>	<b>71</b>
<b>9. Development Summary .....</b>	<b>73</b>
9.1. Notes .....	73
9.2. Further Development .....	73
<b>Bibliography .....</b>	<b>75</b>

## Appendix Contents

<b>A - Code Listing for cronframe crate .....</b>	<b>76</b>
A-1 - src/lib.rs .....	76
A-2 - src/job_builder.rs .....	77
A-3 - src/cronjob.rs .....	82
A-4 - src/cronframe.rs .....	89
A-5 - src/cronframe_expr.rs .....	98
A-6 - src/web_server.rs .....	100
A-7 - src/logger.rs .....	144
A-8 - src/config.rs .....	147
A-9 - src/utils.rs .....	149
A-10 - src/bin.rs .....	150
<b>B - Code Listing for cronframe_macro crate .....</b>	<b>158</b>
B-1 - src/lib.rs .....	158
<b>C - Dependencies List .....</b>	<b>168</b>

# 1. Introduction

This thesis focuses on presenting and describing CronFrame, a framework and CLI tool for the definition and execution of cron jobs developed in the Rust language to be used in Rust projects.

Cron jobs were created to solve the problem of having any kind of task executed at specific times with a simple syntax for the job definition. The first program that addressed this need was aptly named cron, from “Chronos” the Greek word for time, and made it possible to have jobs running with just a string that took care of its scheduling followed by the command to execute. Cron and most other tools of this kind address the execution of commands at the user level, which means a program is run as if it were entered in the terminal by the user. Therefore any command the user has got permission to run can be run. In the context of programming languages, this can be highly restrictive.

If we are dealing with interpreted languages then there is not much of a problem, the command is the call to the interpreter to run the script but if we look at compiled languages we can run their artifact, most of the time an executable binary, only after compilation has completed successfully.

This means that to run a job that executes said binary we require a first job to compile the source code, and a second job to run it.

However, doing so raises other problems:

- How can we be sure that compilation has terminated before it is scheduled to be run by cron?
- How can we check that there hasn't been a compilation error?

We can't if we just use these two commands and need workarounds.

One of these workarounds could be turning the cron job that runs the binary into a job that runs a script that checks compilation is done and only in that case runs the job. Or maybe get rid of the two jobs and have a single job running a script that checks whether compilation has already been performed and if so, run the job. But still, we are relying on a script that is possibly written in another language and interpreted.

This introduces an unnecessary “middleman” and complicates things more than they need to be. It is also tedious for anyone who wants to run jobs since it would require them to set up such scripts in the first place. The main reason behind CronFrame is to render all this nearly identical to simply running a binary in the original cron tool. Such problems in the framework are dealt with at compile time of the job so that it always has everything it needs to run at its disposal when the time comes, and if we need new jobs when the framework is already running we can always add them either from code or with the CLI tool.

CronFrame is primarily a framework for defining and executing cron jobs. It uses macro annotations as the main method of job definition but also supports job definition through function calls. This framework can be easily integrated into any Rust project, allowing a cron job scheduler to work with minimal configuration and setup. Upon initialization of the framework instance, a web server is launched, providing users with GUI functionalities and API routes.

The second aspect of Cronframe is that it serves as a CLI tool that uses the homonymous framework. This means that users can initiate the scheduler using just one command and utilize additional commands for job definition, such as loading them from a txt file containing a job list or adding them one by one in the terminal. The tool launches a global instance of CronFrame,

which can be configured to some extent, offering all the functionalities of the framework along with some extra commands for added convenience.

The motivation behind this project stems from wanting to turn Rust code into a cron job by simply adding an annotation to the original code. This idea is inspired by frameworks like Spring, which utilize annotations to expand the capabilities of the host language, in the case of Spring, Java. Rust conveniently offers a native way to extend itself using macros, and certain types of macros can be employed like annotations in Spring.

CronFrame first started conceptually as a library but soon evolved into a small and compact framework. It can be easily extended and maintained by anyone with a decent familiarity with the Rust language. The CLI tool was a natural addition to the original framework.

Documentation of the framework is provided with the cargo doc tool native to Rust which allows for embedding documentation as comments on top of code.

A website containing step-by-step guided tutorials has been built with docusaurus and configured to provide references to multiple versions of the framework and CLI tool.

The entire code is available on a GitHub repository under either an MIT license or an APACHE 2.0 license.

Chapter 2 provides a general definition of a cronjob and then specializes it to the framework. Chapter 3 quickly delves into Rust and focuses on its macros to convey the reason behind its choice for the project. Chapter 4 presents the framework in all of its components explaining how they work and relate to each other. Chapter 5 is where the CLI tool is presented with a detailed explanation of its commands. Step-by-step guided tutorials for both the framework and the CLI tool are available in Chapter 6. The testing suite written for checking the correct functionality of the framework is described in Chapter 7. The reception both the framework and the CLI tool have received from the Rust community is recorded in Chapter 8. Finally, a development summary with notes and possible future outlooks is available in Chapter 9.

## 2. Scheduling of Cron Jobs

In this chapter, we start from the definition of a cron job. While at first, the definition will certainly be general, it will soon be specialized to the specific case of the project. Particular attention is given to the context of execution the framework operates in since it is not the typical one that might be anticipated by users of other cron job utilities.

### 2.1. Definition of a Cron Job

A cron job is a task of any type scheduled to run without ambiguity at a specific time. Such time of execution is described by a codified expression which allows the user to define schedules. A schedule can have times spanning from a simple one-off occurrence to multiple occurrences, to very frequent time intervals.

There are fields at the user's disposal for schedule definition such as minute, hour, month, and more. The exact number of fields is heavily dependent on the implementation of a specific tool. The values these fields can assume mostly follow those of the first release of the cron [1] CLI utility for Unix systems from 1975, but each cron job utility might differ in what is allowed in an expression, and even for the number of fields the expression is made of. One example is the mentioned cron tool lacking the field for seconds while more recent tools account for it. This is due to the use of different parsers and different needs at the time of writing a cron job tool.

For the entirety of this thesis, the composition of a cron expression will pertain to the specifics of the parser used which is the one implemented in the cron crate available on crates.io.

There are seven different fields at the user's disposal for schedule definition, these are:

- seconds
- minutes
- hours
- day of the month
- month
- day of the week
- year

#### 2.1.1. The Schedule: Cron Expression

A string with at least six and at most seven fields separated by whitespace is known as a cron expression and it is the first step for the definition of a cron job since it takes care of denotating its schedule.

The fields are written from left to right with six of them as mandatory, the exception being the year field which can be omitted to mean any year.

All fields take numeric values in specific ranges. Only two fields, namely day of month and day of week can substitute their numeric values with a 3 letter abbreviation of their English names. These abbreviations are not case-sensitive.

All fields can have special characters that allow for more variation and flexibility when defining a schedule.

Field	Required	Allowed Values	Allowed Special Characters
Seconds	Yes	0-59	, - * /
Minutes	Yes	0-59	, - * /
Hours	Yes	0-23	, - * /
Day of Month	Yes	1-31	, - * / ?
Month	Yes	1-12 or JAN-DEC	, - * /
Day of Week	Yes	1-7 or sun-sat	, - * / ?
Year	No	empty or 1970-2100	, - * /

Table 1: Allowed Values for Cron Expressions Fields

The meaning of the special characters is quite straightforward and intuitive.

Character	Meaning	Effect
,	set of values	Only values separated by a comma are used
-	range of values	Only values in the range (inclusive) are used
*	wildcard / all values	All values the field can assume are considered valid
/	step of values	Start from the values on the left of the / and repeat after the time of the value on the right has elapsed
?	Yes	Does the same as * but can only be used in day of the month and day of the week

Table 2: Special Characters Meaning in Cron Expressions

Examples of job definitions with their worded-out description follow.

Cron Expression	Description
0/5 * * * * *	At every 5th second from 0 through 59.
* 0 0 25 Dec * *	At 00:00 on day-of-month 25 in December.
* 0 22 * * mon-fri *	At 22:00:00 on every day-of-week from Monday through Friday.
* 30 5 * * SUN *	At 05:30 on Sunday.
* 0/10 7-9,16-18 * * Mon-Fri *	At every 10th minute from 0 through 59 past every hour from 7 through 9 and every hour from 16 through 18 on every day-of-week from Monday through Friday.
* 0 0,12 1,5,12 2 * *	At minute 0 past hour 0 and 12 on day-of-month 1, 5, and 12 in February.

Table 3: Examples of Cron Expressions

There are cron job tools that might provide further special characters to add more options. It is wise to never assume that every cron job tool provides the same functionality despite sharing a very similar base.

Some cron expressions define schedules that are of naturally frequent use, so much so that short-hands are available in their stead.

Cron Expression	Shorthand	Description
0 0 0 1 1 * *	@yearly	At midnight every January 1st
0 0 0 1 * * *	@monthly	At midnight every 1st day of every month
0 0 0 * * 1 *	@weekly	At midnight every week on Sunday
0 0 0 * * * *	@daily	At midnight every day
0 0 * * * *	@hourly	At the start of every hour

Table 4: Shorthand of Common Cron Expressions

For shorthands as well, it varies on the tool implementation, some might have additional ones or even more than one alias for the same one.

### 2.1.2. The Job: Commands and Functions

With a cron expression at hand and our timezone of preference, for that will always be UTC, we can define schedules that best suit our needs. This however does not define a cron job in itself. In addition to the schedule a cron job needs, as the name implies, the job component to be executed. This job component is usually a CLI command in most cases which allows the user to define any possible task the computer has the software required to carry out, whether it be simply echoing some text or running a script to gather data or perform complex computations.

In the scope of this thesis however, a cron job is quite literally a function in the environment of the Rust programming language therefore what we can do in a job is dictated by the code the language enables us to write and what permissions we possess on the system.

## 2.2. Common Issues of Cron Jobs Usage

While the use of cron jobs is extremely common and has been trialed and tested for decades, there are still some pitfalls that users might find themselves into when dealing with them:

- **Expression Issues**
  - Entering the wrong scheduling syntax, leading to jobs not running when intended or even entering invalid syntax, leading to a parse error.
- **Permissions Issues**
  - A cron job may not have the necessary access to execute if it is set up with the wrong user permissions.
- **Environment and Path Issues**
  - A cron job might fail due to it not having access to necessary environment variables.
  - It might also fail due to the use of relative paths instead of absolute paths in its code.
- **Overlapping Issues**
  - Without proper management, jobs may overlap, leading to system resource contention.
  - Even worse if jobs are performed in a cloud environment.
- **Error Handling Issues**
  - Without proper logging of output and errors it is troublesome to do any kind of troubleshooting.
  - Due to the jobs spanning different threads any troubleshooting is even more complex.
- **Output Issues**

- ▶ If a cron job sends its output via email to one or multiple addresses it can be an issue if not properly configured, especially if the output volume is high.

- **Recovery Issues**

- ▶ If a cron job fails to execute or complete either because of application or machine failure there is no guarantee it will be recovered in any way.
- ▶ Job recovery is not standard functionality for most cron utilities, if required it must be handled by integrating a separate tool that takes care of it.

Dealing with all of these issues requires, for the most part, the application of best practices and common sense. Knowing the tools one uses in and out is usually the first step to take for mitigating or getting entirely rid of any kind of issue.

### 2.3. Execution of a Cron Job

Taking as example cron, one of the most used tools for cron jobs on Linux here is how to define a job using it:

```
1 * * * * * python3 /home/user/run_every_second.py
```

This definition is stored in what is called a crontab file and it will result in the execution of the Python script “run\_every\_second.py” every second by the cron utility that is automatically running right after the system starts.

Notice how the cron expression here is composed of 5 fields, in order from left to right:

- minute
- hour
- day-of-month
- month
- day-of-week

As stated earlier, cron doesn’t account for the seconds field, and the year field is just omitted so the actual number of fields at a user’s disposal when using cron is 6.

Our job here can be any command we have permission to run on the system:

```
1 * * * * * <command>
```

Since there is a plethora of time-withstanding tools that deal with this kind of cron job, this thesis work veered on making a framework for turning functions defined in the Rust language into cron jobs.

A closer depiction of how we define a cron job in this context would be:

```
1 * * * * * * <function_name>
```

Notice how the fields in our case are 7 (if the year is not omitted).

This is however not the way to define a job because for us it needs context, namely the scope in which it can be defined and what it can access when dealing with struct types. Still, we will see that when using the CLI tool the job definition gets pretty similar to the one above and we can even use a txt file to define a list of jobs.

The CronFrame framework will make extensive use of macro annotations on top of functions for most of its functionality as will be explained in detail in Chapter 4 but before that a look as to why Rust was picked as a language for this project is dealt with in the following chapter.

### 3. The Rust Language

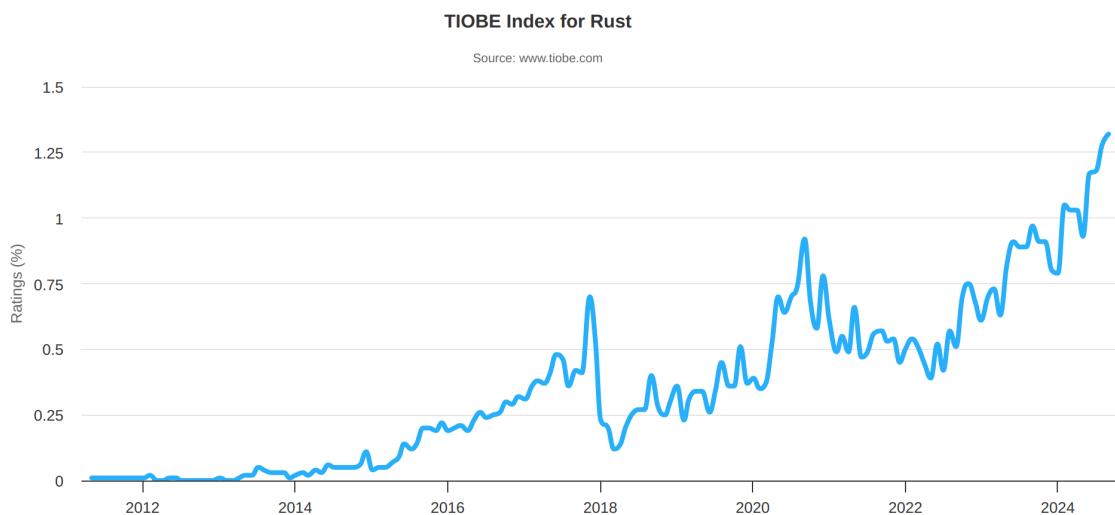
Before we start looking at cron jobs and the framework, this chapter introduces Rust to those who might be unfamiliar with it. Since Rust is a big and complex language, after a brief introduction focus is shifted to macros which play a big role in the framework.

Rust [2] is a general-purpose programming language that focuses on performance, concurrency, and memory safety. Memory safety in particular is enforced at compile time so that all references point to valid memory which is done without using a garbage collector.

All this is possible because of the borrow checker which tracks the object lifetime of all references during compilation helping prevent data races as well. Rust is heavily influenced by functional programming languages and has features such as immutability, higher-order functions, and algebraic data types. The programming paradigm of Rust is neither fully object-oriented nor fully functional but it can be used so to lean towards one of the two according to the user's preference.

It also comes with great tooling and dealing with structurally complex Rust projects is much less of a headache thanks to cargo [3], its native package manager whose counterpart in C and C++ is entirely lacking. Documentation also gets far less tedious to write since it can easily be written with doc comments on top of code with one neat feature of supporting testing code in the documentation so that examples are always sure to compile.

One feature of Rust in particular that the framework developed for this thesis work heavily relies upon is the macro system [4] which is far more useful and powerful than the text substitution-based pre-processor type of macros available in other languages.



The past few years saw a notable increase in Rust's popularity, with companies like Microsoft that have invested millions into it and are actively porting substantial portions of their code bases to Rust. The Defense Advanced Research Projects Agency, also known as DARPA has stated their intention of converting legacy C code to Rust. In research, Rust won the Programming Languages Software Award in 2024 [5] given by ACM SIGPLAN with a citation that mentioned:

“... Born out of many years of research and experimentation, Rust tackles the real-world challenges and issues needed for practical adoption. Rust has been recognized as one of a handful of Safer Languages by NIST, and is increasingly deployed in industry for its safety benefits by large and small companies alike.”

— sigplan.org/Awards/Software/2024\_The\_Rust\_Programming\_Language

In the annual developer survey, [6] carried out by Stackoverflow, Rust was elected the most admired language in 2024, and 83% of developers who use it want to continue using it.

### 3.1. The Macro System

What mainly pushed towards picking Rust as the language for this project is its native macro system which is extremely versatile and far more advanced than the macros one might be familiar with from C-like languages.

There are two types of macros in Rust, each for a specific purpose but in the end, what they all do is get code or tokens as input, perform code manipulation/expansion on it, and get the resulting code as output. Essentially macros allow us to use metaprogramming which is code that generates more code. A big reason for wanting metaprogramming in Rust is that macros are expanded at compile time and can have variadic parameters as well as implement traits which is something that functions cannot do since they get called at run time. The reduced amount of code needed to write when using macros is also an advantage for the maintainability of code bases in the long run.

#### 3.1.1. Declarative Macros

```
1 // macro definition
2 macro_rules! power {
3     ($value:expr, squared) => {{
4         let x: u32 = $value;
5         x.pow(2)
6     }}
7     ($value:expr, cubed) => {{
8         let x: u32 = $value;
9         x.pow(3)
10    }}
11 }
12 // macro utilisation
13 let square_of_2 = power!(2, squared);
14 let cube_of_2 = power!(2, cubed);
```

Declarative macros [7] are invoked very much like a normal function gets invoked but they are marked with the ! symbol at the end of their name so there is no risk of mistaking a macro for a function or vice versa.

One example in addition to the one above is the `println!` macro which is available for the user's convenience and provides a very versatile way for printing to the standard output.

They are defined with `macro_rules!`, which is technically not part of Rust syntax but rather a syntax extension.

There must be at least one rule where each rule looks like the following:

```
1 ($matcher) => {$expansion}
```

These kinds of macros can enormously reduce the amount of code to be written and are very frequently used in Rust. They are a great way to have less verbose code, however, in the scope of this thesis project there was no need for them therefore we shall no longer be concerned with them going forward.

Let us have a look at the other types of macros Rust possesses.

### 3.1.2. Procedural Macros

Procedural macros [8] are so named since they are nothing more than a procedure that gets code as input and outputs code, they are essentially a Rust function that generates other Rust code.

There is a further categorization of this kind of macro into three subtypes but at their core all three work almost the same with differences found in their inputs and outputs.

To write this kind of macros they must be defined in a separate crate of type proc-macro due to internal structure and name resolution reasons. All this complexity is probably going to be relaxed or even lifted in future versions of the language. One peculiarity of this type of crate is that it is enabled to export only macros and nothing else.

#### 3.1.2.1. Function-Like Macros

```
1 // macro definition
2 #[proc_macro]
3 pub fn function_macro(input: TokenStream) -> TokenStream {
4     TokenStream::new()
5 }
6 // macro utilization
7 function_macro!();
```

This first type of procedural macro can be invoked the way a declarative macro can. While at first glance they might seem identical to declarative macros they are not. Function-like macros are much more powerful than declarative macros since their input is not restricted in the same way that declarative macros inputs are, it can be anything and it can be transformed into anything within the bounds of the Rust language.

#### 3.1.2.2. Derive Macros

```
1 // macro definition
2 #[proc_macro_derive(MyDerive)]
3 pub fn derive_macro(annotated_item: TokenStream) -> TokenStream {
4     TokenStream::new()
5 }
6
7 // macro utilisation
8 #[derive(MyDerive)]
9 struct MyStruct{
10     // ...
11 };
```

Derive Macros are also very commonly used in Rust since they are a very quick way of automatically implementing a trait on a struct or enum type. This is useful since Rust possesses quite a few traits such as Display, Debug, Clone, Copy, and Default just to name a few. Derive macros can also be used to automatically derive a user-defined trait which is why they are quite neat and spare programmers the tedious task of writing implementation blocks for each trait they want to have on a struct or enum by hand.

Once again, such macros are not used in this thesis work therefore we shall no longer be concerned with them going forward.

The last type of macro is what we will focus on and what our framework uses extensively.

### 3.1.2.3. Attribute Macros

```
1 // macro definition
2 #[proc_macro_attribute]
3 pub fn derive_macro(input: TokenStream, annotated_item: TokenStream) ->
TokenStream {
4     TokenStream::new()
5 }
6
7 // macro utilization
8 #[derive_macro(attributes)]
9 fn example_function(){
10     // ...
11 }
12
```

Derive macros are essentially used to add functionality to existing code by automatically implementing traits on them, however, they are restricted to structs and enums, most importantly they cannot add fields to an existing structure which is something we require. For these reasons, the type of macros used in cronframe are attribute macros which can be used on any kind of item and can also extensively alter the original code. In our specific use case, we will use attribute macros to parse structs only, since the parsing of enums is far more complex due to the various ways they can be used in Rust. This is however not a restriction because we can put structs inside enum variants and still make use of the framework functionalities.

## 3.2. Developing Macros

Developing macros requires parsing code into tokens and performing modifications or even turning text generated from this initial parsing into new tokens. All of this poses a big problem for anyone wanting to develop a macro: Rust is not a trivial language to parse.

Luckily some libraries make parsing the language much easier, some are also written specifically so that users can extend the language with macros according to their needs.

These libraries are the syn crate and the quote crate.

## 3.3. The Syn Crate

Available from crates.io, Rust's official crate registry:

```
1 $ cargo add syn
```

Syn [9] is a parsing library for parsing a stream of Rust tokens into a syntax tree of Rust source code geared toward use in Rust procedural macros.

Syn provides syntax trees starting from a syntax tree rooted at `syn::File` which represents a full source file of valid Rust source code to different entry points, such as:

- `syn::Item`
- `syn::Expr`
- `syn::Type`

Such entry points are of particular interest for macros.

Macros can have input parameters that need to be parsed themselves and support for that is also provided, for example in the case of Derive macros there is `syn::DeriveInput` which is any of the three legal input items to a derive macro.

Every token parsed by syn is associated with a span that tracks line and column information back to the source of that token, this allows the display of detailed error messages pointing to the causing source in the code.

An example of how to write a derive macro using syn is:

```
1 use proc_macro::TokenStream;
2 use quote::quote;
3 use syn::{parse_macro_input, DeriveInput};
4
5 #[proc_macro_derive(MyMacro)]
6 pub fn my_macro(input: TokenStream) -> TokenStream {
7     // parse the input tokens into a syntax tree
8     let input = parse_macro_input!(input as DeriveInput);
9
10    // build the output
11    let expanded = quote! {
12        // using quasi-quotation in this case
13    };
14
15    // hand the output tokens back to the compiler
16    TokenStream::from(expanded)
17 }
```

As we can see we get the input tokens and parse the input expecting valid syntax for the derive input specifically. Then we go on to generate the expanded code and finally return it as a token stream.

Now, there are a few ways to expand code:

- using the `parse` method implemented by syn on a string of text
- parse the input code and directly modify its syntax tree
- use quasi-quoting

The first one is the simplest by far, you just write code in a string and call the `parse` method but it might be a bother to inject data from the macro input into the string.

The second one is by far the most tedious of the three and probably the most error-prone approach as well since it requires to directly modify the syntax tree.

Finally, the last approach is a new thing the quote [10] crate allows us to do using the quote macro as per the example. With the quasi-quoting approach, you can write code normally inside the macro and inject variables from your macro expansion directly into the code that the macro needs to generate.

It is a bit of a “mix-world” where you use text and tokens at the same time to write code and it is the main approach used for this thesis.

### 3.4. The Quote Crate

The idea behind quasi-quoting is writing code that we treat as data.

Within the quote macro, we can write code that looks like the one we would write normally in a text editor and therefore we get all the benefits of using it such as brace matching, syntax highlighting, and indentation.

Autocompletion and type hinting are not always a given due to the complexity of the context the quote macro operates in but we might get it in some fortunate instances.

What differs from writing normal code to writing code inside the quote macro is that rather than compiling the code into the current crate, it is treated as data, passed around, and mutated until eventually it is handed back to the compiler as tokens to compile into the macro’s crate.

An example of writing macros with the quote macro is:

```
1 extern crate proc_macro;
2
3 use proc_macro::TokenStream;
4 use quote::quote;
5
6 #[proc_macro_derive(MyMacro)]
7 pub fn my_macro(input: TokenStream) -> TokenStream {
8     // parse the input and figure out what to do with it
9     let name = /* ... */;
10    let expr = /* ... */;
11
12    // build the output
13    let expanded = quote! {
14        impl MyTrait for #name {
15            fn my_trait_function(&self) -> usize {
16                #expr
17            }
18        }
19    };
20
21    // hand the output tokens back to the compiler
22    TokenStream::from(expanded)
23 }
```

Here we use the content of the name and expr variables defined inside the body of the my\_macro function called by the macro. What we do is use that content as data inside the quote macro so

that we can implement the MyTrait trait on the content of the name variable which will be a type identifier and inside the function my\_trait\_function we use the content of the expr variable to define what the function should return.

This quasi-quoting approach is extremely useful as essentially it makes writing macros which are inherently complex pieces of code more like writing “normal” code.

## 4. The Framework

After an initial presentation of the framework is dealt with at the start of this chapter, what follows is an in-depth look at each component and how they work together. Finally, there are examples of macro expansions so to provide striking proof as to how much easier they make life for a user.

CronFrame is at its core a Rust framework that allows the definition of cron jobs from functions. An instance of the framework includes a web server that can be used to start and stop the scheduler, pause the scheduling of an individual job, and more. It also provides opt-in logging that is active by default but, should the user prefer to have their own, they can simply opt-out of it in the init function.

The framework allows for the definition of four types of jobs, all of these are functions but their context is different:

- **Global Jobs:** these are the jobs defined from basic Rust functions, those in the “global” scope, that is to say, they are not part of a type as they are standalone functions
- **Function Jobs:** in the context of struct types this first type of job is related to the type itself rather than to a concrete instance of it, meaning there will never be a duplicate of a function job on a type running in the scheduler
- **Method Jobs:** this second type of job that can be defined in struct types is related to an instance of the type as all methods are, at any given moment we can have multiple “versions” of the same job each related to a specific instance of the type they are defined on
- **CLI Jobs:** as the name implies these jobs are defined only through the CLI tool commands, they refer to external Rust code that is compiled and turned into a job

Examples of each job type follow.

```
1 // Global Job Example
2 #[cron(expr = "0/5 * * * * *", timeout = "0")]
3 fn job_function() {
4     println!("I print every 5 seconds...");
```

```
5 }
6
7 // Cron Object Example
8 #[cron_obj]
9 struct Greeting {
10     name: String,
11     expr: CronFrameExpr,
12 }
```

```
13
14 #[cron_impl]
15 impl Greeting {
16     // Function Job Example
17     #[fn_job(expr = "0/5 * * * * *", timeout = "10000")]
18     fn function_job() {
19         println!("Have a good day!");
```

```
20     }
21
22     // Method Job Example
23     #[mt_job(expr = "expr")]
```

```

24     fn method_job(self) {
25         println!("Hello {}", self.name);
26     }
27 }
28
29 // CLI Job Example
30 fn main(){
31     println!("the main function is the job function for CLI jobs!");
32 }
```

Notice the different contexts they are defined in. A CLI job is a peculiar case where the entire Rust code is the job and so main itself is to be considered as the job's function body.

Scheduling times are all UTC to keep macro attributes at a bare minimum, but for each job, a local time is provided on their web page.

Job recovery, much in line with the cron tool, is not a functionality that cronframe provides, at least at the time of writing. That is partly due to intricacies in the implementation of such functionality but also due to how it would impact the native macros-first intended utilization of the framework. How inflated macros would get if we had to mark a job either as recoverable or not, and how it should be recovered if more than one option is provided might too negatively impact user experience.

## 4.1. Framework Instance

A framework instance is required for jobs to be collected and parsed from their definitions and to be sent to the scheduler for execution. Such an instance carries a web server and a log configuration, both running right after completion of the init function while the scheduler will require invoking the start\_scheduler method to get working.

The first thing the user will have to do is an initialization of the framework and then start the scheduler in the main function of their Rust project.

The initialization will also take care of gathering any global jobs and pass them to the scheduler which will set up their thread of execution exactly one second before they are supposed to run according to the cron expression dictated schedule.

For function and method jobs there are a few further steps to perform as it wouldn't make much sense to automatically collect them for two simple reasons:

- it is very subjective how and when a function job should be collected, if there is at least one instance of the type, or if a cron object simply serves as a job container
- method jobs are strictly related to a type instance which is possibly created well after the framework has been initialized

To front these problems, each cron object is injected with functions and methods that allow it to pass its jobs to a cronframe instance in a very straightforward way.

```

1 use cronframe::CronFrame;
2
3 fn main(){
4     // framework initialization
```

```

5  // global job collection happens here as well
6  let cronframe = CronFrame::default();
7
8  // scheduler start with keep main alive
9  cronframe.run();
10 }

```

## 4.2. Annotation Macros

Macros are the main interface with the framework instance from the users' perspective and they have been the addressed focus point from the very start of development.

Job creation without macros is still supported but it is not the intended way of use for cronframe as a framework.

There is a total of five macros, specifically attribute macros as they are known in Rust:

- `cron`
- `cron_obj`
- `cron_impl`
- `fn_job`
- `mt_job`

The first one is a standalone macro to be used with “global scoped” functions to make cron jobs out of them. The other four macros instead are to be used with struct types, if we want to define cron jobs from methods or associated functions within them. Specifically, we need to use the `cron_obj` macro on top of a struct definition and the `cron_impl` macro on top of its implementation block which will only contain jobs for any given struct we want to use for cron job definition. The jobs contained in this implementation block will either be function or method jobs depending on which macro between `fn_job` and `mt_job` is used on top of the single function/method definition.

A struct type capable of hosting cron jobs inside it will be referred to as a **cron object** purely for ease of reference from here onwards.

These few macros allow us to generate code at compilation time therefore making all the required boilerplate setup code a non-burden on the user so that they can turn their attention to the code that defines what the jobs do much sooner than it would take if the macros weren't there in the first place.

A more in-depth look and explanation of the macros follows.

### 4.2.1. Macro: cron

As mentioned before, this macro is used on top of standalone functions in Rust to define what is known in cronframe as “global cron jobs”.

Such functions take no input arguments and return no value.

The macro itself takes two attributes which are:

- **expr**
  - the cron expression defining the scheduling for the job
- **timeout**
  - value in ms at which to stop the job after its first execution, it resets daily

The following code snippet shows how to use the macro:

```
1 #[cron(expr = "0 * * * *", timeout = "0")]
2 fn cron_job(){
3     println!("I run every minute...");
4 }
```

The first thing done in the macro is the parsing of the attributes to check they are correct in name and quantity, expecting to get exactly two attributes named expr and timeout in that order. When this step has a positive outcome, the original function code will be parsed using the syn crate expecting to get an ItemFn. To this code, we will simply append a call to the submit macro of the inventory crate [11] in which we will create an instance of the JobBuilder type for the specific case of a global job. Using the inventory crate for global job collection allows us to collect all global job builders automatically at run time before even entering the main function so that when we init the framework we can easily get them and build the job instances to be used.

#### 4.2.2. Macro(s) : cron\_obj and cron\_impl

These macros are only useful if both are used therefore they will be dealt with as if they were one. What they do is “turn” a struct into a cron object by acting on two different contexts of the struct with cron\_obj acting on the struct definition, and cron\_impl acting on an implementation block that is expected to solely contain job definitions.

Both macros take no attributes as they just set up a lot of framework code behind the scenes which is not dependent in any way on user directives.

A Cron Object is just a struct type that:

- automatically derives the **Clone** trait
- requires its fields to implement the Clone trait
- has a mutex boolean associated with it in the global scope
- has message-passing channels associated with it in the global scope
- has **two arrays of helper functions** returning builder instances associated with it in the global scope
  - one for functions jobs
  - one for method jobs
- can carry fields of type **CronFrameExpr** to be used to schedule cron jobs
- is injected with an additional field name **tx** for message passing between threads
- gets an implementation of the drop trait for dropping method jobs when their instance goes out of scope
- is injected with additional functions
  - **new\_cron\_obj** for creating a new instance of the cron object
  - **cf\_drop\_fn** for dropping function jobs
  - **cf\_gather\_fn** for passing all function jobs it contains to the cronframe instance
- is injected with additional methods
  - **cf\_gather\_mt** for passing all method jobs it contains to the cronframe instance
  - **cf\_gather** for passing all the jobs it contains to the cronframe instance

The user can add as many fields as needed of type CronFrameExpr inside a cron object for job scheduling.

The following code snippet shows how to use the macros:

```
1 #[cron_obj]
2 struct User {
3     expr1: CronFrameExpr,
4     expr2: CronFrameExpr,
5     name: String,
6     age: u8,
7 }
8
9 #[cron_impl]
10 impl User {
11     // define jobs here
12 }
```

Starting with the cron\_obj macro, the first thing it does is inject a field named tx which is an Option<Sender> type from the crossbeam\_channel crate used for message-passing.

Then it builds the definition of the new\_cron\_obj function which is variadic in number of parameters and has exactly as many as those defined by the user in the struct definition in the order they were defined, necessary to deal with the setup of the additional tx field.

To the edited structure it appends the mutex definition and the message-passing channels definition as well as the Drop trait implementation and an implementation block containing the new\_cron\_obj function and the cf\_drop\_fn function.

Last, but not least, it appends the definition of two arrays of helper functions that are annotated with a macro from the linkme [12] crate. This is what makes gathering them before main is executed possible. As soon as we step in the main function we can already instantiate a cron object and pass its jobs to the framework instance.

This macro is the first step for defining a struct type that can carry cron jobs and as mentioned above, it is not of much use by itself which is why after annotating a struct type with it, we need to annotate an implementation block with the cron\_impl macro that will signal that every function or method contained in the block is expected to be a cron job.

The first thing the cron\_impl macro does is fill the arrays defined by the cron\_obj macro with the helper functions so that for each job that is inside the implementation block we add its helper function to the arrays.

The second thing this macro does is the addition of a new implementation block where the gathering functions are defined.

There are three types of gathering functions, all of them:

- build job instances from the JobBuilders returned by the helper functions
- set up the message-passing, and pass the jobs to the cronframe instance

In particular, **cf\_gather\_fn** is an associated function that takes care of function jobs, **cf\_gather\_mt** is a method that takes care of methods jobs and **cf\_gather** is also a method that simply calls the previous two to handle both job types.

As we can see from the code example, inside the implementation block annotated with cron\_impl is where we need to define jobs, which we can do with:

- **fn\_job** macro
  - we use this macro on associated functions, therefore it defines a job for the type rather than one of its instances
- **mt\_job** macro
  - we use this macro on methods, therefore it defines a job for an instance of the type

#### 4.2.3. Macro: fn\_job

This macro is the equivalent of the `cron` macro in the context of a cron object and as the `cron` macro does, it also takes two attributes which are:

- **expr**
  - the cron expression defining the scheduling for the job
- **timeout**
  - value in ms to stop the job after its first execution, it resets daily

This macro must be used in cron objects and only inside an implementation block annotated with the `cron_impl` macro.

It annotates an associated function turning it into a function job in the context of cronframe.

```

1 #[cron_obj]
2 struct User {
3     expr1: CronFrameExpr,
4     expr2: CronFrameExpr,
5     name: String,
6     age: u8,
7 }
8
9 #[cron_impl]
10 impl User {
11     // function job definition
12     #[fn_job(expr = "0 * * * * *", timeout = "0")]
13     fn my_function_job(){
14         // ...
15     }
16 }
```

The first thing this macro needs to do is make certain there is no `self` parameter in the function definition otherwise it would be meaningless to expand it since the context would be that of a method instead of an associated function.

Since the macro is the cron object equivalent of the `cron` macro, it also needs to check the number and the names of the attributes, asserting that there are only two named `expr` and `timeout` respectively. If that is the case it proceeds to add its helper function returning a `JobBuilder` instance for the job after the original function.

The helper function is named: `cron_helper_<name of the original function>`.

A macro allowing dead code is put on top of the original function definition since it might be falsely signaled as dead when it is being used through indirect calls by the cronframe instance. This might be lifted in the future as Rust analyzer or whatever tool in its stead becomes more able to track uses macro generated code, in the meanwhile this spares the user from annoying warnings.

#### 4.2.4. Macro: mt\_job

This macro is also similar to the `cron` macro in the context of a cron object but unlike the `cron` macro, it takes only one attribute:

- **expr**
  - ▶ the name of the `CronFrameExpr` field defined by the user for scheduling

The name of this attribute is the same as it was in the `cron` and `fn_job` macros however, this is not the cron expression in a string format. What this attribute expects is the name of a field of type `CronFrameExpr` defined by the user in the cron object structure.

There needs to be at least one of this type of field.

This macro must be used in cron objects and only inside an implementation block annotated with the `cron_impl` macro.

It annotates a method turning it into a method job in the context of cronframe.

```
1 #[cron_obj]
2 struct User {
3     expr1: CronFrameExpr,
4     expr2: CronFrameExpr,
5     name: String,
6     age: u8,
7 }
8
9 #[cron_impl]
10 impl User {
11     // method job definition
12     #[mt_job(expr = "expr1")]
13     fn my_method_job(self){
14         // ...
15     }
16 }
```

This time around, the first thing the macro needs to do is make certain there is a `self` parameter in its definition otherwise it would be meaningless to expand it since the context would be that of an associated function instead of a method.

If there is a `self` parameter it goes on to check the presence of a single attribute named `expr` and, in that case, it generates:

- a helper function named `cron_helper_<name of the original method>`
- a function named `cronframe_method_<name of the original method>`

Both of these are added after the original definition of the method.

The field chosen for the `expr` attribute is injected into the construction of the helper function when parsing the cron expression. The last added function (`cronframe_method_<...>`) has the same code block as the original method but here any occurrence of `self` has been replaced with the cron object instance at the moment of job gathering.

This instance is then passed as an input parameter to the function as an `Any` type to be then down-cast to the type of the cron object. This means that essentially a copy of the cron object instance will be passed to the scheduler and they will evolve separately from that point forward.

### 4.3. The JobBuilder Type

A cron job gets built when invoking the `build` method from an instance of this type. `JobBuilder` instances are returned by the helper functions generated by the `fn_job` or the `mt_job` macros while with global jobs a `JobBuilder` is submitted to the inventory crate macro so it can be later obtained. Since what gets built can be either a global job, a function job, or a method job, this type is implemented with an enum containing struct variants.

```
1 #[derive(Debug, Clone)]
2 pub enum JobBuilder<'a> {
3     Global {
4         name: &'a str,
5         job: fn(),
6         cron_expr: &'a str,
7         timeout: &'a str,
8     },
9     Method {
10         name: &'a str,
11         job: fn(arg: Arc<Box<dyn Any + Send + Sync>>),
12         cron_expr: String,
13         timeout: String,
14         instance: Arc<Box<dyn Any + Send + Sync>>,
15     },
16     Function {
17         name: &'a str,
18         job: fn(),
19         cron_expr: &'a str,
20         timeout: &'a str,
21     },
22     CLI {
23         name: &'a str,
24         cron_expr: &'a str,
25         timeout: &'a str,
26     },
27 }
```

As we can see, the `Global` and `Function` variants are virtually the same, their separation is only for keeping things in order from a logical standpoint as well as avoiding too much refactoring should their structure need changes in future versions. The `Method` variant is strikingly different in that it takes a more complex function pointer for the `job` field. It has a parameter that allows you to call it with a copy of the instance of the cron object hosted in the `instance` field.

The `CLI` variant is used for a type of job that can be added at run time from the `CLI` tool and will be dealt with in Chapter 5.

The implementation block of this type has three associated functions each returning an instance of one of the three variants, these are the functions that get called in the helper functions generated by the macros for them to return a `JobBuilder` instance.

Once we have an instance we require a way to build it which is exactly what the `build` method does. It first matches the type of job to build and then goes on to parse the cron expression and

timeout. This is where each job gets its unique UUID and status channels. Life channels will be left to the `cf_gather` method to update as described earlier.

Once the build method returns, we end up with an instance of the `CronJob` type as described in the following paragraph.

The `JobBuilder` type is never used directly, any instance of it is automatically generated by macros or function calls on the framework instance.

#### 4.4. The CronJob Type

This type contains what is required for a job to be scheduled to run by the scheduler and identified in the web server.

```
1 #[derive(Debug, Clone)]
2 pub struct CronJob {
3     pub suspended: bool,
4     pub name: String,
5     pub id: Uuid,
6     pub job: CronJobType,
7     pub schedule: Schedule,
8     pub timeout: Option<Duration>,
9     pub timeout_notified: bool,
10    pub status_channels: Option<(Sender<String>, Receiver<String>)>,
11    pub life_channels: Option<(Sender<String>, Receiver<String>)>,
12    pub start_time: Option<DateTime<Utc>>,
13    pub run_id: Option<Uuid>,
14    pub method_instance: Option<Arc<Box<dyn Any + Send + Sync>>>,
15    pub failed: bool,
16 }
```

While fields like `name` and `id` are self-explanatory, others require some explanation.

The `suspended` field is a boolean that tells the scheduler whether the job is currently suspended from being scheduled or not. In case it is it simply gets skipped in the scheduler loop.

The `job` field is where the actual job resides as well as its type information.

The `schedule` field is a type obtained from parsing the cron expression which is used to handle future executions.

`Timeout` is an optional type containing a duration in milliseconds and the `timeout_notified` is a boolean required to log that a job has timed out.

Message-passing channels are used to handle job completion, abortion, and deletion and there are two different types. The `status_channels` handle all things concerning the job's scheduling and execution events such as completion and abortion.

A job's life, i.e. whether it is still alive and needs scheduling or it must be removed is handled by the `life_channels`. Though these only work for function and method jobs.

Removing global jobs is not available but should their execution no longer be required they can simply be suspended from scheduling.

The `start_time` field gets set upon the first schedule of the job and is used as the time base upon which timeout occurrence is checked. On timeout reset, it also gets reset when it will run once again.

Since jobs might fail on a specific run and execute just fine in others there needs to be a further way to identify a specific execution and this is what the `run_id` field is used for. It gets set anew each time a new execution of the job is required as per the `schedule` field.

The `method_instance` field is only set for method jobs and it is where the copy of the cron object to be used for running the job is hosted, without it we wouldn't be able to pass the context of the cron object to the scheduler.

Lastly, the `failed` field is simply a flag used to signal whether there have been any failed instances for a job in its execution history.

There are only methods associated with this type and as seen in previous examples, we never use it directly to build instances of it because that is what the `JobBuilder` type is meant to be used for.

The list of methods associated with the `CronJob` type is:

- `run`
- `run_graceful`
- `get_run_id`
- `type_to_string`
- `status`
- `schedule`
- `try_schedule`
- `set_schedule`
- `check_schedule`
- `set_timeout`
- `check_timeout`
- `timeout_reset`
- `timeout_to_string`
- `upcoming_utc`
- `upcoming_local`

The purpose of the methods is evident in their naming and most of them are only a few lines of code as shown in the appendix.

The core method involved in the scheduling is `try_schedule` which is invoked by the scheduler to see if an upcoming execution of the job is imminent by calling the `check_schedule` method that will return a true boolean value if the job is expected to run within the next second.

If that is the case, the job will be assigned a `run_id` for that specific execution and it will be scheduled for execution by calling the `run` method. Should the `run` method fail by returning any error, then the job will try to schedule for a limited period called graceful period which is by default 250ms but configurable by the user.

The `run` and `run_graceful` methods do nearly the same thing which is spawning two threads:

- control thread
- job thread

The **control thread** spawns the job\_thread and it is required to check whether the job thread it spawned has terminated correctly or it aborted for whatever reason. This is done by waiting for the job thread to finish and checking its handle. If the handle contains an error a “JOB\_ABORT” message is sent to the scheduler via status channels otherwise the job has been executed correctly and the message “JOB\_COMPLETE” is sent to the scheduler.

The **job thread** does the actual execution of the job, it sleeps until the time for execution (which is within a second), and then it matches the type of the job to invoke the job function. If the job is a global job or a function job, the call to the job function is simply done without any input parameters as it doesn’t need any. For method jobs instead, we have to also get the instance of the cron object the job was created from which is hosted in the method\_instance field, and pass it to the job function as input parameter for execution.

The difference between the two versions of the run method lies in the `run_graceful` method checking whether the upcoming execution is yet to come and in that case, waits for it but if that time already went by then it immediately executes the job. It also logs that a job has been executed during graceful time.

## 4.5. The CronFrame Type

An instance of this type represents the core of the framework.

Macros by themselves prepare all that is necessary for the jobs to be runnable but it is the CronFrame type that provides the execution.

```
1 pub struct CronFrame {
2     pub cron_jobs: Mutex<Vec<CronJob>>,
3     job_handles: Mutex<HashMap<String, JoinHandle<()>>>,
4     _logger: Option<log4rs::Handle>,
5     pub web_server_channels: (Sender<Shutdown>, Receiver<Shutdown>),
6     pub filter: Option<CronFilter>,
7     server_handle: Mutex<Option<Shutdown>>,
8     pub quit: Mutex<bool>,
9     pub grace: u32,
10    pub running: Mutex<bool>,
11 }
```

In the type definition, we find a mutex of a vector of CronJobs which is where all the jobs are collected, the job pool at the scheduler’s disposal. For each job that is in a running state, we have a thread join handle stored in a hashmap with the job’s id as the key.

The logger handle is wrapped in an `Option` type since the user might want their own tailored configuration for logging instead of using the default one provided by the framework.

Webserver channels are used for the setup and start-up of the web server when calling the `init` associated function which takes as arguments an enum variant wrapped in an `Option` for the filter and a boolean for whether or not to use the default logger.

The filter allows the scheduler to run only one type of job among global, function, method, or CLI jobs should the user want to do so.

For the user’s convenience, there is also a `default` associated function that takes no arguments and calls in its body the `init` function with no filter and the default logger configuration enabled.

The `grace` field contains either the default value of 250ms for the graceful period or the one set by the user in the configuration file.

The `running` mutex boolean simply signals whether the scheduler is currently running or not. The `quit` mutex is used for terminating the cronframe instance.

The `init` function first sets up the logger, creates an instance of Cronframe with no jobs, and starts looking for global job builders that have been collected with the `cron` macro using the inventory crate, then it calls the `build` method and gets the jobs built and pushes them into the vector of jobs.

The initialized instance of cronframe is now ready but before it can be passed to the server it needs to be wrapped in an `Arc` type since it will be shared by the main thread of the program as well as the web server thread.

The web server thread is started with an `Arc` instance of the cronframe framework and we wait for its thread to pass via message passing the quit handle for the server to the cronframe instance. Once we have it we can return an `Arc` instance of cronframe where the `init` function was invoked.

At this point, everything is set up and the server is running but the scheduler is not.

We just need to call the `start_scheduler` method to start it and bear in mind that this will not keep the main thread alive. To keep it alive a user either implements their program logic or calls the `keep_alive` method which simply runs an infinite loop that does nothing but sleep for 500ms and checks whether a quit request has been filed. In case it has, it breaks the loop terminating the framework instance.

A `run` method that starts the scheduler and keeps it alive is also provided for convenience.

The initialization process only collects global jobs since that is all that is known at compile time before anything in the main function is executed. For adding function jobs and method jobs to the scheduler, the Cronframe type provides an `add_job` method which is used in the `cf_gather`, `cf_gather_fn`, and `cf_gather_mt` functions available in cron objects.

Should the user not want to use macros, there is a `new_job` method that can be used to add jobs to the cronframe instance from either global functions or closures. These jobs will be typed as global jobs.

## 4.6. The Web Server: Rocket

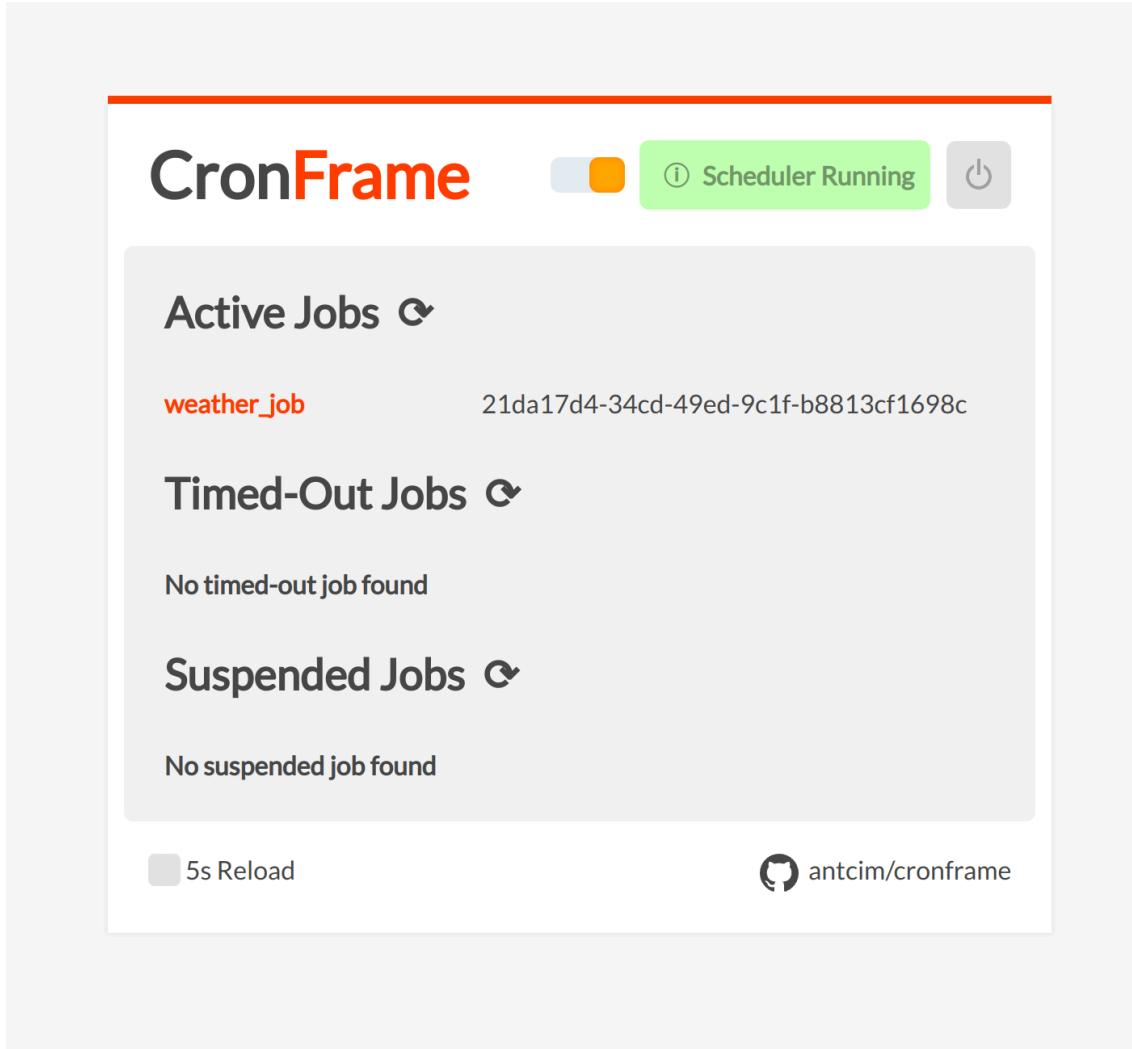


Figure 2: Home Page of the Webserver

The user might want to start and stop the scheduler, see the status of the jobs in the job pool, and check whether they are running, awaiting for schedule or if they have timed out, or maybe make sure the job hasn't failed running instances in the past.

A user might also want to change the scheduling or timeout and they might want to do all this with a GUI when their application is running. For these purposes, a web server is at the user's disposal.

By default, the server is set to run on port 8098 at the address 127.0.0.1 also known as localhost.

The ip and port can be changed with a configuration file as the pertaining paragraph will later show.

The web server uses the Rocket framework [13] since it provides a well-tested multithreaded runtime environment as well as common functionality and a very configurable setup to get a web server up and running without too much boilerplate code.

Upon the first start of the framework inside the current directory, a templates directory will be generated containing the following files:

- base.html.tera

- index.html.tera
- job.html.tera
- tingle.js
- cronframe.js
- styles.css
- tingle.css

The .tera files contain common HTML extended with templating capabilities parsed by the tera engine provided with Rocket. The .js files are tingle.js which is from a modal library and cronframe.js made for the web server functionality. The style files are there to give a somewhat pleasing appearance to the interface with support of light and dark themes.

The templates directory allows for custom configuration of the web server appearance should the user feel the need to.

There is a total of nine routes provided by the server:

- **job list route**
  - ▶ found at `http://127.0.0.1:8098/`
  - ▶ it shows a list of jobs in the current pool with their IDs, clicking on a job name redirects to the job's info route
- **job info route**
  - ▶ `http://127.0.0.1:8098/job/<name>/<id>`
  - ▶ it provides a page with the job's definition and status information
- **schedule set API route**
  - ▶ found at `http://127.0.0.1:8098/job/<name>/<id>/schedset/<expression>`
  - ▶ it provides a way to change the schedule of a job and is used in the job info route
- **timeout set API route**
  - ▶ found at `http://127.0.0.1:8098/job/<name>/<id>/toutset/<value>`
  - ▶ it provides a way to change the timeout of a job and is used in the job info route
- **job suspension toggle API route**
  - ▶ found at `http://127.0.0.1:8098/job/<name>/<id>/suspension_toggle`
  - ▶ it allows to suspend or reprise of the scheduling of a single job
- **scheduler start API route**
  - ▶ found at `http://127.0.0.1:8098/start_scheduler/`
- **scheduler stop API route**
  - ▶ found at `http://127.0.0.1:8098/stop_scheduler/`
- **add CLI job API route**
  - ▶ found at `http://127.0.0.1:8098/add_cli_job/<expr>/<timeout>/<job>`
- **cronframe shutdown API route**
  - ▶ found at `http://127.0.0.1:8098/shutdown`

Each route is checked against a route guard that in addition to providing the requested route also provides type checking for dynamic components in the route, that is to say, the parts enclosed by angle brackets.

All dynamic components in the above routes are regarded to be of type string except for the <value> component of the timeout set route which is checked to be compatible with the i64 type.

Every route carries additional hidden data, this data is the Arc of the cronframe instance that has been passed to the web server thread during the init of the framework.

The screenshot shows the CronFrame web application interface. At the top, there is a red header bar. Below it, the title "CronFrame" is displayed in large red font, followed by a blue button with a play icon and a green button labeled "Scheduler Running". To the right of the title is a grey button with a circular arrow icon.

The main content area has a light grey background and features a card titled "Job Info @weather\_job" with a small edit icon. The card contains the following information:

Name	weather_job
Id	21da17d4-34cd-49ed-9c1f-b8813cf1698c <input type="button" value="Edit"/>
Type	CLI Job
Status	<span>Awaiting Schedule</span> <span>Suspend Scheduling</span>
Fail History	No failed instances recorded
Schedule	0 0/1 * * * * <input type="text" value="enter cron expression"/> <input type="button" value="Update"/>
Timeout	None <input type="text" value="enter timeout in ms"/> <input type="button" value="Update"/>
Upcoming	2024-09-09 10:59:00 UTC 2024-09-09 12:59:00 +02:00 (Local)

At the bottom left is a "5s Reload" button, and at the bottom right is the "antcim/cronframe" logo.

Figure 3: Job Page on the Webserver

#### 4.7. The Logger: log4rs

The logger is implemented with the log4rs [14] crate which is itself affine to the log4j library.

Logging cron jobs can lead to dealing with enormous file sizes depending on the scheduling the jobs have. For this reason, instead of using a logger that always appends to a file, the default logger is set up in a rolling logger configuration with 3 files of archive.

Once the latest.log file reaches 1MB of maximum size it becomes an archive file and a new latest file is written. Archive files are named archive\_i.log where i is a digit that goes from 0 to archive\_files - 1, with 0 being the more recent file of the archives.

The logger is extensively configurable, it can be entirely not used or its parameters can be configured from a configuration file, such as:

- dir
  - the directory where to store log files
- file\_size
  - maximum file size in megabytes of the log files
- archive\_files
  - the number of archive files to generate
- latest\_file\_name
  - the name to be used for the current log file
- archive\_file\_name
  - the name to be used for the archive files
- msg\_pattern
  - the message pattern for what to print when logging
- level\_filter
  - the log message filter for what to write on the log file

## 4.8. The Configuration File

Rust's package manager Cargo makes extensive use of toml files for configuration purposes. Each Rust project has a Cargo.toml where package configuration, dependencies, workspace configuration, development dependencies, and other types of configurations are stored.

This allows for a clear overview and edit of any configuration available in that file which is fast and painless for the user to do.

Other rust crates, for example, Rocket itself, make use of additional toml files to aid user configurability.

For this reason a cronframe.toml configuration file can be used with the framework to set a few things when it comes to the server, the logger, and the scheduler.

The following is an example of a configuration file with all available fields set to the default configuration:

```
1 [webserver]
2 port = 8098
3 ip = "127.0.0.1"
4
5 [logger]
6 dir = "log"
7 file_size = 1 # this is in MB
8 archive_files = 3
9 latest_file_name = "latest"
10 archive_file_name = "archive"
11 msg_pattern = "{d(%Y-%m-%d %H:%M:%S %Z)} {l} {t} - {m}{n}"
12 level_filter = "info"
13
14 [scheduler]
15 grace = 250 # this is in ms
```

As we can see there are three sections marked by square brackets, one for the webserver, one for the logger, and one for the scheduler. Each of these sections is optional and each field in a section is optional as well.

In the web server section we can set the:

- ip
  - the ip to use to run the server
  - 127.0.0.1 (also localhost) by default
- port
  - the port number where the webserver will be available at
  - 8098 by default

In the logger section we can set:

- dir
  - the directory to use to save the log files
  - “log” by default
- file\_size
  - the size in megabytes of a single log file
  - 1MB by default
- archive\_files
  - the number of archive files to retain in addition to the latest file
  - 3 by default
- latest\_file\_name
  - the file name for the latest log file
  - “latest” by default
- archive\_file\_name
  - the file name for the archive files which will end in a number automatically
  - “archive” by default
- msg\_pattern
  - the pattern to use when logging a message on a file
  - “{d(%Y-%m-%d %H:%M:%S %Z)} {l} {t} - {m}\n” by default
  - refer to the log crate on crates.io for message syntax
- level\_filter
  - the level filter to use when logging
  - possible choices are: “error”, “off”, “warn”, “debug”
  - “info” by default

In the scheduler section we can set:

- grace
  - the graceful period to be used for job scheduling
  - 250 milliseconds by default

The full code of all framework components we have seen is available in the Appendix.

## 4.9. Macro Expansions

The rest of this chapter contains examples of the code produced by each macro expansion to show how much the macro themselves prepare the field for the utilization of the framework.

#### 4.9.1. Macro Expansion: cron

The code written by the user is the following:

```
1 #[macro_use]
2 extern crate cronframe;
3
4 use cronframe::{CronFrame, CronFrameExpr};
5
6 #[cron(expr="0 0/30 * * * *", timeout="0")]
7 fn my_global_job(){
8     // job code here...
9 }
10 fn main() {
11     let cronframe = CronFrame::default();
12     cronframe.run();
13 }
```

We have a global job that doesn't do anything but the code inside its body is not relevant to the purpose of showing the expansion of the macro.

At compile time these few lines of code will be expanded in the following according to what has been described earlier:

```
1 #[macro_use]
2 extern crate cronframe;
3 use cronframe::{CronFrame, CronFrameExpr};
4 fn my_global_job() {}
5 #[allow(non_upper_case_globals)]
6 const _: () = {
7     static __INVENTORY: ::inventory::Node = ::inventory::Node {
8         value: &{
9             cronframe::JobBuilder::global_job(
10                 "my_global_job",
11                 my_global_job,
12                 "0 0/30 * * * *",
13                 "0",
14             )
15         },
16         next: ::inventory::core::cell::UnsafeCell::new(
17             ::inventory::core::option::Option::None,
18         ),
19     };
20     #[link_section = ".text.startup"]
21     unsafe extern "C" fn __ctor() {
22         unsafe { ::inventory::ErasedNode::submit(__INVENTORY.value,
23             &__INVENTORY) }
24     }
25     #[used]
26     #[link_section = ".init_array"]
27     static __CTOR: unsafe extern "C" fn() = __ctor;
28 };
29 fn main() {
```

```

29 let cronframe = CronFrame::default();
30 cronframe.run();
31 }

```

Immediately, the amount of work that macros spare us is quite astonishing. We can see that an instance of a JobBuilder for a global job is constructed and collected by the inventory crate right on top of main so that when default is called it will have the JobBuilder at its disposal by calling the inventory crate's functionality.

#### 4.9.2. Macro Expansion: cron\_obj, cron\_impl, fn\_job, mt\_job

This following expansion example is quite big, first of all as we know, the cron\_obj macro and the cron\_impl macro are just one macro split in two due to the context of the operation. Also, neither of these two macros is useful if we do not define a job at the end, therefore we define both types of cron objects jobs in the same implementation block to show what comes out of a full-fledged cron object.

The code written by the user is the following:

```

1 #[macro_use]
2 extern crate cronframe;
3
4 use cronframe::{CronFrame, CronFrameExpr};
5
6 #[cron_obj]
7 struct ExpansionObj{
8     cron_expr: CronFrameExpr,
9 }
10
11 #[cron_impl]
12 impl ExpansionObj{
13     #[fn_job(expr="0 0/30 * * * *", timeout="0")]
14         fn my_function_job(){
15             // my function job code here...
16         }
17
18     #[mt_job(expr="cron_expr")]
19         fn my_method_job(self){
20             // my method job code here...
21         }
22 }
23
24 fn main() {
25     let cronframe = CronFrame::default();
26     cronframe.run();
27 }

```

What these few lines of code will become is the following:

```

1 #[macro_use]
2 extern crate cronframe;

```

```

3
4 use cronframe::{CronFrame, CronFrameExpr};
5
6 #[cron_obj]
7 struct ExpansionObj{
8     cron_expr: CronFrameExpr,
9 }
10
11 #[cron_impl]
12 impl ExpansionObj{
13     #[fn_job(expr="0 0/30 * * * *", timeout="0")]
14     fn my_function_job(){
15         // my function job code here...
16     }
17
18     #[mt_job(expr="cron_expr")]
19     fn my_method_job(self){
20         // my method job code here...
21     }
22 }
23
24 fn main() {
25     let cronframe = CronFrame::default();
26     cronframe.run();
27 }
```

What these few lines of code will become is the following:

```

1 #[macro_use]
2 extern crate cronframe;
3 use cronframe::{CronFrame, CronFrameExpr};
4 struct ExpansionObj {
5     cron_expr: CronFrameExpr,
6     tx: Option<cronframe::Sender<String>>,
7 }
8 #[automatically_derived]
9 impl ::core::clone::Clone for ExpansionObj {
10     #[inline]
11     fn clone(&self) -> ExpansionObj {
12         ExpansionObj {
13             cron_expr: ::core::clone::Clone::clone(&self.cron_expr),
14             tx: ::core::clone::Clone::clone(&self.tx),
15         }
16     }
17 }
18 static CF_FN_JOBS_FLAG_EXPANSIONOBJ: std::sync::Mutex<bool> =
19     std::sync::Mutex::new(
20         false,
21     );
22 static CF_FN_JOBS_CHANNELS_EXPANSIONOBJ: cronframe::Lazy<
23     (cronframe::Sender<String>, cronframe::Receiver<String>),
24 > = cronframe::Lazy::new(|| cronframe::bounded(1));
```

```

24 impl Drop for ExpansionObj {
25     fn drop(&mut self) {
26         if self.tx.is_some() {
27             let _ =
28                 self.tx.as_ref().unwrap().send("JOB_DROP".to_string());
29         }
30     }
31 impl ExpansionObj {
32     fn new_cron_obj(cron_expr: CronFrameExpr) -> ExpansionObj {
33         ExpansionObj {
34             cron_expr,
35             tx: None,
36         }
37     }
38     fn cf_drop(&self) {
39         if *CF_FN_JOBS_FLAG_EXPANSIONOBJ.lock().unwrap() {
40             for func in CRONFRAME_FUNCTION_JOBS_EXPANSIONOBJ {
41                 let _ =
42                     CF_FN_JOBS_CHANNELS_EXPANSIONOBJ.0.send("JOB_DROP".to_string());
43             }
44             *CF_FN_JOBS_FLAG_EXPANSIONOBJ.lock().unwrap() = false;
45         }
46     }
47 static CRONFRAME_METHOD_JOBS_EXPANSIONOBJ: ::linkme::DistributedSlice<
48     [fn(
49         std::sync::Arc<Box<dyn std::any::Any + Send + Sync>>,
50     ) -> cronframe::JobBuilder<'static>],
51 > = {
52     #[cfg(
53         any(
54             target_os = "none",
55             target_os = "linux",
56             target_os = "macos",
57             target_os = "ios",
58             target_os = "tvos",
59             target_os = "android",
60             target_os = "fuchsia",
61             target_os = "illumos",
62             target_os = "freebsd",
63             target_os = "openbsd",
64             target_os = "psp",
65         )
66     )]
67     extern "Rust" {
68         #[link_name =
69             "__start_linkme_CRONFRAME_METHOD_JOBS_EXPANSIONOBJ"]
70         static LINKME_START: <[fn(
71             std::sync::Arc<Box<dyn std::any::Any + Send + Sync>>,
72         ) -> cronframe::JobBuilder<'static>]
73         as ::linkme::__private::Slice::Element;
74         #[link_name = "__stop_linkme_CRONFRAME_METHOD_JOBS_EXPANSIONOBJ"]

```

```

73     static LINKME_STOP: <[fn(
74         std::sync::Arc<Box<dyn std::any::Any + Send + Sync>>,
75     ) -> cronframe::JobBuilder<'static>]
76     as ::linkme::__private::Slice::Element;
77     #[link_name =
78     "__start_linkm2_CRONFRAME_METHOD_JOBS_EXPANSIONOBJ"]
79     static DUPCHECK_START: ::linkme::__private::usize;
80     #[link_name = "__stop_linkm2_CRONFRAME_METHOD_JOBS_EXPANSIONOBJ"]
81     static DUPCHECK_STOP: ::linkme::__private::usize;
82 }
83 #[used]
84 #[cfg(
85     any(
86         target_os = "none",
87         target_os = "linux",
88         target_os = "android",
89         target_os = "fuchsia",
90         target_os = "illumos",
91         target_os = "freebsd",
92         target_os = "openbsd",
93         target_os = "psp",
94     )
95 )]
96 #[link_section = "linkme_CRONFRAME_METHOD_JOBS_EXPANSIONOBJ"]
97 static mut LINKME_PLEASE: [<[fn(
98     std::sync::Arc<Box<dyn std::any::Any + Send + Sync>>,
99     ) -> cronframe::JobBuilder<'static>]
100 as ::linkme::__private::Slice::Element; 0] = [];
101 #[used]
102 #[link_section = "linkm2_CRONFRAME_METHOD_JOBS_EXPANSIONOBJ"]
103 static DUPCHECK: ::linkme::__private::usize = 1;
104 if !(::linkme::__private::mem::size_of::<
105     <[fn(
106         std::sync::Arc<Box<dyn std::any::Any + Send + Sync>>,
107     ) -> cronframe::JobBuilder<'static>]
108 as ::linkme::__private::Slice::Element,
109     >() > 0)
110 {
111     ::core::panicking::panic(
112         "assertion
113 failed: ::linkme::__private::mem::size_of::<<[fn(std::sync::Arc<Box<dyn
114 std::any::Any +\n                           Send + Sync>>) ->
115 cronframe::JobBuilder<\'static>]
116 as\n             ::linkme::__private::Slice::Element>() > 0",
117         )
118     }
119     unsafe {
120         ::linkme::DistributedSlice::__private_new(
121             "CRONFRAME_METHOD_JOBS_EXPANSIONOBJ",
122             &raw const LINKME_START,
123             &raw const LINKME_STOP,
124             &raw const DUPCHECK_START,
125             &raw const DUPCHECK_STOP,

```

```

118         )
119     }
120 };
121 #[doc(hidden)]
122 use _linkme_macro_CRONFRAME_METHOD_JOB_EXPANSIONOBJ as
CRONFRAME_METHOD_JOB_EXPANSIONOBJ;
123 static CRONFRAME_FUNCTION_JOB_EXPANSIONOBJ: ::linkme::DistributedSlice<
124     [fn() -> cronframe::JobBuilder<'static>],
125 > = {
126     #[cfg(
127         any(
128             target_os = "none",
129             target_os = "linux",
130             target_os = "macos",
131             target_os = "ios",
132             target_os = "tvos",
133             target_os = "android",
134             target_os = "fuchsia",
135             target_os = "illumos",
136             target_os = "freebsd",
137             target_os = "openbsd",
138             target_os = "psp",
139         )
140     )]
141     extern "Rust" {
142         #[link_name =
"__start_linkme_CRONFRAME_FUNCTION_JOB_EXPANSIONOBJ"]
143             static LINKME_START: <[fn() -> cronframe::JobBuilder<
144                 'static,
145             >] as ::linkme::__private::Slice>::Element;
146         #[link_name =
"__stop_linkme_CRONFRAME_FUNCTION_JOB_EXPANSIONOBJ"]
147             static LINKME_STOP: <[fn() -> cronframe::JobBuilder<
148                 'static,
149             >] as ::linkme::__private::Slice>::Element;
150         #[link_name =
"__start_linkm2_CRONFRAME_FUNCTION_JOB_EXPANSIONOBJ"]
151             static DUPCHECK_START: ::linkme::__private::usize;
152         #[link_name =
"__stop_linkm2_CRONFRAME_FUNCTION_JOB_EXPANSIONOBJ"]
153             static DUPCHECK_STOP: ::linkme::__private::usize;
154     }
155     #[used]
156     #[cfg(
157         any(
158             target_os = "none",
159             target_os = "linux",
160             target_os = "android",
161             target_os = "fuchsia",
162             target_os = "illumos",
163             target_os = "freebsd",
164             target_os = "openbsd",
165             target_os = "psp",

```

```

166         )
167     ])
168 #[link_section = "linkme_CRONFRAME_FUNCTION_JOBS_EXPANSIONOBJ"]
169 static mut LINKME_PLEASE: [<[fn() -> cronframe::JobBuilder<
170     'static,
171     >] as ::linkme::__private::Slice>::Element; 0] = [];
172 #[used]
173 #[link_section = "linkm2_CRONFRAME_FUNCTION_JOBS_EXPANSIONOBJ"]
174 static DUPCHECK: ::linkme::__private::usize = 1;
175 if !(::linkme::__private::mem::size_of::<
176     <[fn() -> cronframe::JobBuilder<'static>]
177     as ::linkme::__private::Slice>::Element,
178     >() > 0)
179 {
180     ::core::panicking::panic(
181         "assertion
182 failed: ::linkme::__private::mem::size_of::<<[fn() ->
183 cronframe::JobBuilder<\'static>]\n
184 as ::linkme::__private::Slice>::Element>() > 0",
185         )
186 }
187 unsafe {
188     ::linkme::DistributedSlice::private_new(
189         "CRONFRAME_FUNCTION_JOBS_EXPANSIONOBJ",
190         &raw const LINKME_START,
191         &raw const LINKME_STOP,
192         &raw const DUPCHECK_START,
193         &raw const DUPCHECK_STOP,
194         )
195     }
196 };
197 #[doc(hidden)]
198 use _linkme_macro_CRONFRAME_FUNCTION_JOBS_EXPANSIONOBJ as
CRONFRAME_FUNCTION_JOBS_EXPANSIONOBJ;
199 impl ExpansionObj {
200     #[allow(dead_code)]
201     fn my_function_job() {}
202     fn cron_helper_my_function_job() -> cronframe::JobBuilder<'static> {
203         cronframe::JobBuilder::function_job(
204             "my_function_job",
205             Self::my_function_job,
206             "0 0/30 * * * *",
207             "0",
208             )
209     }
210     #[allow(dead_code)]
211     fn my_method_job(self) {}
212     fn cron_method_my_method_job(
213         arg: std::sync::Arc<Box<dyn std::any::Any + Send + Sync>>,
214     ) {
215         let cron_frame_instance = arg.clone();
216         let cronframe_self =
217             (*cron_frame_instance).downcast_ref::<Self>().unwrap();

```

```

213     }
214     fn cron_helper_my_method_job(
215         arg: std::sync::Arc<Box<dyn std::any::Any + Send + Sync>>,
216     ) -> cronframe::JobBuilder<'static> {
217         let instance = arg.clone();
218         let this_obj = (*instance).downcast_ref::<Self>().unwrap();
219         let expr = this_obj.cron_expr.expr();
220         let tout = {
221             let res = ::alloc::fmt::format(
222                 format_args!("{}", this_obj.cron_expr.timeout()),
223             );
224             res
225         };
226         let instance = arg.clone();
227         cronframe::JobBuilder::method_job(
228             "my_method_job",
229             Self::cron_method_my_method_job,
230             expr.clone(),
231             tout,
232             instance,
233         )
234     }
235 }
236 #[used]
237 #[link_section = "linkme_CRONFRAME_FUNCTION_JOBS_EXPANSIONOBJ"]
238 static LINKME_MY_FUNCTION_JOB_0: fn() -> cronframe::JobBuilder<'static>
= {
239     #[allow(clippy::no_effect_underscore_binding)]
240     unsafe fn __typecheck(_: ::linkme::__private::Void) {
241         let __new = || -> fn() -> &'static fn() ->
242         cronframe::JobBuilder<'static> {
243             || &LINKME_MY_FUNCTION_JOB_0
244         };
245         unsafe {
246             ::linkme::DistributedSlice::private_typecheck(
247                 CRONFRAME_FUNCTION_JOBS_EXPANSIONOBJ,
248                 __new(),
249             );
250         }
251         ExpansionObj::cron_helper_my_function_job
252     };
253 #[used]
254 #[link_section = "linkme_CRONFRAME_METHOD_JOBS_EXPANSIONOBJ"]
255 static LINKME_MY_METHOD_JOB_1: fn(
256     _self: std::sync::Arc<Box<dyn std::any::Any + Send + Sync>>,
257 ) -> cronframe::JobBuilder<'static> = {
258     #[allow(clippy::no_effect_underscore_binding)]
259     unsafe fn __typecheck(_: ::linkme::__private::Void) {
260         let __new = || -> fn() -> &'static fn(
261             _self: std::sync::Arc<Box<dyn std::any::Any + Send + Sync>>,
262         ) -> cronframe::JobBuilder<'static> { ||
263             &LINKME_MY_METHOD_JOB_1 };

```

```

263     unsafe {
264         ::linkme::DistributedSlice::private_typecheck(
265             CRONFRAME_METHOD_JOBS_EXPANSIONOBJ,
266             __new__(),
267         );
268     }
269 }
270 ExpansionObj::cron_helper_my_method_job
271 };
272 impl ExpansionObj {
273     pub fn cf_gather_mt(&mut self, frame: std::sync::Arc<CronFrame>) {
274         {
275             let lvl = ::log::Level::Info;
276             if lvl <= ::log::STATIC_MAX_LEVEL && lvl
277             <= ::log::max_level() {
278                 ::log::__private_api::__log(
279                     format_args!("Collecting Method Jobs from {0}",
280 "EXPANSIONOBJ"),
281                     lvl,
282                     &("expansions",
283 "expansions", ::log::__private_api::loc(),
284                     (),
285                     );
286             }
287         }
288         if !CRONFRAME_METHOD_JOBS_EXPANSIONOBJ.is_empty() {
289             let life_channels = cronframe::bounded(1);
290             self.tx = Some(life_channels.0.clone());
291             for method_job in CRONFRAME_METHOD_JOBS_EXPANSIONOBJ {
292                 let job_builder = (method_job)(
293                     std::sync::Arc::new(Box::new(self.clone())));
294                 let mut cron_job = job_builder.build();
295                 cron_job.life_channels = Some(life_channels.clone());
296                 {
297                     let lvl = ::log::Level::Info;
298                     if lvl <= ::log::STATIC_MAX_LEVEL && lvl
299                     <= ::log::max_level() {
300                         ::log::__private_api::__log(
301                             format_args!(
302                                 "Found Method Job \\"{0}\\" from {1}.",
303                                 cron_job.name,
304                                 "EXPANSIONOBJ",
305                                 ),
306                                 lvl,
307                                 &("expansions",
308 "expansions", ::log::__private_api::loc(),
309                                 (),
310                                 );
311                 }
312             }
313             frame.clone().add_job(cron_job);
314         }
315     }

```

```

311             {
312                 let lvl = ::log::Level::Info;
313                 if lvl <= ::log::STATIC_MAX_LEVEL && lvl
314                     <= ::log::max_level() {
315                         ::log::__private_api::log(
316                             format_args!("Method Jobs from {0} Collected.",
317                                         "EXPANSIONOBJ"),
318                                         lvl,
319                                         &("expansions",
320                                         "expansions", ::log::__private_api::loc(),
321                                         (),
322                                         );
323                         );
324                     } else {
325                         {
326                             let lvl = ::log::Level::Info;
327                             if lvl <= ::log::STATIC_MAX_LEVEL && lvl
328                                 <= ::log::max_level() {
329                                     ::log::__private_api::log(
330                                         format_args!(
331                                             "Not Method Jobs from {0} has been found.",
332                                             "EXPANSIONOBJ",
333                                             ),
334                                             lvl,
335                                             &("expansions",
336                                             "expansions", ::log::__private_api::loc(),
337                                             (),
338                                             );
339                         };
340                     }
341                     pub fn cf_gather_fn(frame: std::sync::Arc<CronFrame>) {
342                         {
343                             let lvl = ::log::Level::Info;
344                             if lvl <= ::log::STATIC_MAX_LEVEL && lvl
345                                 <= ::log::max_level() {
346                                     ::log::__private_api::log(
347                                         format_args!("Collecting Function Jobs from {0}",
348                                         "EXPANSIONOBJ"),
349                                         lvl,
350                                         &("expansions",
351                                         "expansions", ::log::__private_api::loc(),
352                                         (),
353                                         );
354                         };
355                         if !CRONFRAME_FUNCTION_JOBS_EXPANSIONOBJ.is_empty() {
356                             let fn_flag = *CF_FN_JOBS_FLAG_EXPANSIONOBJ.lock().unwrap();
357                             if !fn_flag {
358                                 for function_job in CRONFRAME_FUNCTION_JOBS_EXPANSIONOBJ
359                         {

```

```

355         let job_builder = (function_job)();
356         let mut cron_job = job_builder.build();
357         cron_job.life_channels = Some(
358             CF_FN_JOBS_CHANNELS_EXPANSIONOBJ.clone(),
359         );
360         {
361             let lvl = ::log::Level::Info;
362             if lvl <= ::log::STATIC_MAX_LEVEL && lvl
363             <= ::log::max_level() {
364                 ::log::__private_api::log(
365                     format_args!(
366                         "Found Function Job \"{}\" from
367                         {}",
368                         cron_job.name,
369                         "EXPANSIONOBJ",
370                         ),
371                         lvl,
372                         &("expansions",
373                         "expansions", ::log::__private_api::loc(),
374                         (),
375                         );
376                     );
377                 }
378                 let lvl = ::log::Level::Info;
379                 if lvl <= ::log::STATIC_MAX_LEVEL && lvl
380                 <= ::log::max_level() {
381                     ::log::__private_api::log(
382                         format_args!(
383                             "Function Jobs from {} Collected.",
384                             "EXPANSIONOBJ",
385                             ),
386                             lvl,
387                             &("expansions",
388                             "expansions", ::log::__private_api::loc(),
389                             (),
390                             );
391                         *CF_FN_JOBS_FLAG_EXPANSIONOBJ.lock().unwrap() = true;
392                     }
393                 } else {
394                     {
395                         let lvl = ::log::Level::Info;
396                         if lvl <= ::log::STATIC_MAX_LEVEL && lvl
397                         <= ::log::max_level() {
398                             ::log::__private_api::log(
399                                 format_args!(
400                                     "Not Function Jobs from {} has been
401                                     found.",
402                                     "EXPANSIONOBJ",

```

```

401             ),
402             lvl,
403             &("expansions",
404             "expansions", ::log::__private_api::loc()),
405             (),
406             );
407         );
408     );
409 }
410 pub fn cf_gather(&mut self, frame: std::sync::Arc<CronFrame>) {
411     self.cf_gather_mt(frame.clone());
412     Self::cf_gather_fn(frame.clone());
413 }
414 }
```

Macro expansion of a cron object generates a lot more code, even more so due to the expansion of the linkme crate which is necessary to the functioning of cronframe.

If we have a look, amid all that complicated code we will find the arrays of helper functions, the implementation blocks containing the gathering functions, the mutexes and message-passing channels, and all that has been described in the macro paragraphs.

## 5. The CLI Tool

In this chapter, we will deal with how the CLI tool came about, what it allows a user to do, and list all of its commands. There is also a chapter dedicated to general information about the tool to convey how it works and why it does things in a certain way.

The birth of the CLI tool came about to address the need of users who might not want a framework inside their code base and would much rather use a cron job scheduler program with the capability of adding jobs to it with little to no effort.

The CLI tool provides a global instance of cronframe which sets up its configuration and templating files in the .cronframe directory under the user directory when it is run for the first time.

Log files will also be found inside the log directory under the .cronframe directory.

Since Rust allows for the definition of a library and a binary inside the same crate, the same project can work for both the framework and the CLI tool which makes for tighter integration between the framework and tool.

The CLI tool is self-contained in the bin.rs file.

### 5.1. Commands

The tool works with subcommands, each apt at one very specific task:

- **start** command

```
1 cronframe start
```

- ▶ Start the web server and job scheduler in the background.

- **run** command

```
1 cronframe run
```

- ▶ Run the web server and job scheduler in the terminal.

- **add** command

```
1 cronframe add '0 0/30 * * *' 0 my_cli_job.rs
```

- ▶ Adds a new CLI job to a CronFrame instance.
- ▶ The arguments it takes are: expression, timeout, job
- ▶ The job is added to the CLI tool instance of cronframe by default.
- ▶ The –port option can be used to target another cronframe instance.

- **load** command

```
1 cronframe load --file my_job_list.txt
```

- ▶ Load jobs from a job definition file.
- ▶ It will perform the same functionality as the add command for each line of the job definition file.
- ▶ If the –file option is not present it will load the job\_list.txt in the .cronframe directory.

- **scheduler** command

```
1 cronframe scheduler start
```

- ▶ Perform actions on the scheduler like start and stop
- ▶ It acts by default on the cronframe instance of the CLI tool
- ▶ The –port option can be used to change the target of the command.

- **shutdown** command

```
1 cronframe shutdown
```

- ▶ It shuts down the cronframe instance of the CLI tool.

- **help** command

```
1 cronframe help scheduler
```

- ▶ Print the help message for the tool or the help of the given subcommand

## 5.2. Tool Information

By default, the CLI tool instance of cronframe will start at <http://localhost:8098>.

Configuration is available via cronframe.toml inside the .cronframe directory.

The tool itself is written with extensibility in mind.

Each command is self-contained in a function and can take 0 or more arguments upon invocation, this allows for a quick swap of functions to try different versions of the same command should the need arise for it. The modular nature of the commands also allows for splitting the source code of the tool across multiple files should a future version need it because of additional commands or increased complexity of the current ones.

Argument parsing is implemented with the clap library and can be vastly customized to support more subcommands, required arguments, and available options. In addition to keeping things tidy, clap autogenerated a helper utility command that provides the users with information for each registered command as well as their subcommands. A version flag option, -V, is also provided automatically by clap to show the current version of the tool.

OS support for the CLI tool has been tested on both Linux, Ubuntu 22.04 specifically, and on Windows 11. The Rust standard library has a Path type that accounts for the different syntax used by Unix systems and Windows for defining directories in a path.

Much of the work veered on localizing commands specific to a single OS. This is easily done with native Rust macros that allow us to check whether we are compiling the tool for Windows or Linux during installation and therefore supporting more operating systems in the future would simply be a matter of checking for them and providing the os specific code.

While rustc, Rust's compiler, and cargo, Rust's native package manager, used for the compilation of the CLI jobs are commands common to both operating systems, the commands used to copy the final binary of a CLI job inside the .cronframe/cli\_jobs directory, are not.

On Linux we use “cp” for this purpose, while on Windows we use “copy”, they essentially do the same thing in the end but their context of execution is different as well as their name.

The functionality of the CLI tool has been manually tested only on Ubuntu and Windows, however, since MacOs is very much rooted in Unix as Linux is, the tool should function without significant problems. Should it not, making it work on MacOs is not expected to be too much work. This has not been tested due to a lack of Apple hardware at the moment of writing.

Although many of the libraries used within cronframe have support for many other operating systems, the targeted Operating Systems for the library are Linux distros and the latest Windows version.

The only requirement for installing the tool and for it to function properly is to have cargo installed which will be installed by default on any machine that has the Rust toolchain.

Although the CLI tool could be installed without cargo on the system, it relies on it for some of its more interesting functionality, such as adding CLI jobs. For this reason, there is no alternative method of installation to the one using the cargo install command.

### 5.3. CLI Jobs

One feature reserved for the tool is the creation of CLI jobs.

A CLI job simply makes a cron job out of Rust source which can be either a single .rs file or an entire project that needs to be built with cargo.

What this essentially means is that our job is whatever is contained inside the main function.

We have two ways of creating a CLI job.

Use the add command of the tool to add a single CLI job:

```
1 // this uses rustc for compilation
2 cronframe add '0 0/30 * * *' 0 path/to/my_cli_job.rs
```

```
1 // this uses cargo for compilation
2 cronframe add '0 0/30 * * *' 0 path/to/my_cli_job
```

Should we have multiple jobs to add, we can save their definition inside a .txt file and load it with the load command.

```
1 cronframe load --file path/to/my_job_list.txt
```

If the load command does not have the –file option, it will look for a file named “jobs\_list.txt” under the .cronframe directory.

## 6. Documentation and Tutorial

This chapter is mainly focused on tutorials for both the framework and the CLI tool. A link to the official documentation of cronframe is provided, the documentation is not reported here due to it being too cumbersome.

The Documentation for the framework is automatically generated by the Rust doc tool and can be found at <https://docs.rs/cronframe>, which will show the latest version as well as list all available versions.

The rest of this chapter is set as a tutorial for the use of the framework and the CLI tool on Linux. It will show how to set up the framework in a newly created Rust project and define a few different types of jobs to show what can be done with it.

It will also explain how to run the examples available on the GitHub repo.

For setting up Rust on your machine refer to the official Rust website at <https://www.rust-lang.org/tools/install>.

The Rust compiler version used for this tutorial is the 1.80.1.

As of writing, the latest version of Cronframe is the 0.1.3.

While not referenced here Windows is also supported and using cronframe on Windows would be nearly identical to using it on Linux.

### 6.1. Framework Tutorial

#### 6.1.1. Setting Up CronFrame

There are two ways we can set up cronframe:

- using cargo to add it automatically to the project by pulling it from crates.io
- using git to link or clone the repo inside the project

The setup from crates.io, Rust's official crates registry guarantees the use of a working release version of the framework while setting up from git gives the latest updates but no working guarantees as the code might be untested unless the code is the one from a release commit.

##### 6.1.1.1. Set up from Crates.io

The first thing to do is to create a new Rust project with cargo as one normally does with any Rust project.

```
1 $ cargo new hello_cronframe
```

This will generate a binary crate in the hello\_cronframe directory in the current working directory with all the starter code inside it.

To proceed we enter the directory and add the framework:

```
1 $ cargo add cronframe
```

The last step is to add a crate on which cronframe depends on:

```
1 $ cargo add linkme@0.3.26
```

Note that the linkme crate has a specific version to be used. Refer to the readme on the GitHub repo for which version of linkme is required or whether it is no longer needed.

### 6.1.1.2. Set up from GitHub Repo

The setup from the GitHub repo is barely more complicated but requires some care. If you use the latest commit from the master branch then be aware of it possibly not working due to untested code or development of new features.

The first thing to do is to create a new Rust project with cargo as one normally does:

```
1 $ cargo new hello_cronframe
```

This will generate a binary crate in the hello\_cronframe directory in the current working directory with all the starter code inside it.

To proceed we enter the directory and inside the Cargo.toml file we add:

```
1 #[dependencies]
2 cronframe = { git = "https://github.com/antcim/cronframe.git"}
```

Again, the last step is adding a crate cronframe depends on:

```
1 $ cargo add linkme@0.3.26
```

If you want to use the framework from the repo it is recommended to either download the source code from the releases page or clone it from the commit of a release and put it inside your project directory.

To proceed, independently of which way we used to get cronframe, we enter the main directory of our project and inside the Cargo.toml file we add:

```
1 #[dependencies]
2 cronframe = { path = "./cronframe", version = "0.1.3"}
```

Again, the last step is adding a crate cronframe depends on:

```
1 $ cargo add linkme@0.3.26
```

### 6.1.2. Using CronFrame

Once we have the framework set we only need to decide on what job we want to define.

Let us start with a global job that greets the user with a good morning message every morning at 8 am from Monday to Friday only.

What we need to do first is to import the macros into our project by writing the following code on top of our main.rs file:

```
1 #[macro_use]
2 extern crate cronframe;
```

This will bring the cron, cron\_obj, cron\_impl, fn\_job, and mt\_job macros into scope for us to use. Since we are defining a global job, for the moment we shall only be concerned with using the cron macro.

It is now time to define our job by writing a function and annotating it with the cron macro:

```
1 #[cron(expr="0 0 8 * * Mon-Fri *", timeout="0")]
2 fn greeting_job(){
3     println!("Have a good morning!");
4 }
```

Now we have defined the job but we need an instance of the framework so that it can be sent to the scheduler and executed when it is supposed to, so we need to bring into scope the CronFrame type:

```
1 use cronframe::CronFrame;
```

Then we need to init cronframe and for that, there is the default function which inits cronframe with no filter for the job type allowed and sets up the default rolling logger.

```
1 fn main(){
2     let cronframe = CronFrame::default();
3 }
```

At this point, the instance will be initialized and our global job will have been built and added to the job pool. The web server available at the default address <http://localhost:8098> will also be running but the scheduler is yet to run.

We can run the scheduler with either the start\_scheduler method or with the run method. The difference is that start\_scheduler returns and does not keep the main thread alive while run keeps main alive.

```
1 fn main(){
2     let cronframe = CronFrame::default();
3     cronframe.run(); // shorthand for
4     cronframe.start_scheduler().keep_alive();
5 }
```

At this point both the webserver and scheduler will be running and the job will be executed when the scheduled time comes.

To recap, the full code will look like the following:

```
1 #[macro_use]
2 extern crate cronframe;
```

```

3 use cronframe::CronFrame;
4
5 #[cron(expr="0 0 8 * * Mon-Fri *", timeout="0")]
6 fn greeting_job(){
7     println!("Have a good morning!");
8 }
9
10 fn main(){
11     let cronframe = CronFrame::default();
12     cronframe.run(); // shorthand for
13     cronframe.start_scheduler().keep_alive();
14 }
```

Now, say we don't want our jobs defined in global space but rather we would like to have a sort of container for them. That can be done with structs by turning them into cron objects.

The import will be the same as the previous one:

```

1 #[macro_use]
2 extern crate cronframe;
3 use cronframe::CronFrame;
```

We will now make use of the cron\_obj and the cron\_impl macros to define a cron object from a struct named Greeting:

```

1 #[cron_obj]
2 struct Greeting;
3
4 #[cron_impl]
5 impl Greeting{
6     // our job goes here
7 }
```

At this point, we have our container as well as where to put our job. All we need to do is copy and paste the previous global job definition inside the implementation block of Greeting and change to cron macro to the fn\_job macro:

```

1 #[cron_obj]
2 struct Greeting;
3
4 #[cron_impl]
5 impl Greeting{
6     #[fn_job(expr="0 0 8 * * Mon-Fri *", timeout="0")]
7     fn greeting_job(){
8         println!("Have a good morning!");
9     }
10 }
```

At this point, our container with a function job is defined and as expected we need the initialization of the framework which is the same as the previous one:

```
1 fn main(){
2     let cronframe = CronFrame::default();
3 }
```

However, the initialization here does no longer automatically collect our job since it is not a global job anymore. We need to pass the job to the cronframe instance which can be done either before or after we start the scheduler.

Just for the sake of it let us do it after we start the scheduler:

```
1 fn main(){
2     let cronframe = CronFrame::default();
3     cronframe.start_scheduler();
4     Greeting::cf_gather_fn(cronframe.clone());
5 }
```

We use the cf\_gather\_fn function automatically generated by the macros on the cron object to pass all function jobs defined in the Greeting cron object to the cronframe instance. Since it is a function job it doesn't even require instantiating the struct for it to work.

At this point, the scheduler is already started and we just need to keep the main thread alive:

```
1 fn main(){
2     let cronframe = CronFrame::default();
3     cronframe.start_scheduler();
4     Greeting::cf_gather_fn(cronframe.clone());
5     cronframe.keep_alive();
6 }
```

To recap, the full code will look like the following:

```
1 #[macro_use]
2 extern crate cronframe;
3 use cronframe::CronFrame;
4
5 #[cron_obj]
6 struct Greeting;
7
8 #[cron_impl]
9 impl Greeting{
10     #[fn_job(expr="0 0 8 * * Mon-Fri *", timeout="0")]
11     fn greeting_job(){
12         println!("Have a good morning!");
13     }
14 }
15
16 fn main(){
17     let cronframe = CronFrame::default();
18     cronframe.start_scheduler();
19     Greeting::cf_gather_fn(cronframe.clone());
```

```
20     cronframe.keep_alive();
21 }
```

We have a nice greeting job that greets everybody at 8 am UTC. But what if we have employees working remotely from different time zones? And what if we wanted to customize the greeting message by adding the name of the employee to it?

For that, we start by turning our function job into a method job by doing the following:

- importing the CronFrameExpr type
- changing the fn\_job macro to the mt\_job macro
- renaming the job to custom\_greeting\_job
- removing the timeout attribute in the macro
- adding the self parameter to the function definition

```
1 #[macro_use]
2 extern crate cronframe;
3 use cronframe::{CronFrame, CronFrameExpr};
4
5 #[cron_obj]
6 struct Greeting;
7
8 #[cron_impl]
9 impl Greeting{
10     #[mt_job(expr="")]
11     fn custom_greeting_job(self){
12         println!("Hi ..., have a good morning!");
13     }
14 }
```

As you might have noticed, the expression has been set to an empty string.

Why?

Because we need to write the name of the field of type CronFrameExpr we want to use for that job. So now we add that, and since we're at it we also add a String type field for the name of the employee and use it in our message:

```
1 #[cron_obj]
2 struct Greeting{
3     employee: String,
4     my_expr: CronFrameExpr,
5 };
6
7 #[cron_impl]
8 impl Greeting{
9     #[mt_job(expr="my_expr")]
10    fn custom_greeting_job(self){
11        println!("Hi {}, have a good morning!", self.employee);
12    }
13 }
```

At this point our method job definition is complete but since this is a method job and methods require an instance of a type to be used, we know that after the usual initialisation of the cronframe instance we need to create an instance for Greeting as well.

This can be done with the new\_cron\_obj function that is automatically created on the Greeting cron object by the macros. The function takes as arguments the fields that are defined inside the Greeting structure in the exact order they are defined.

```
1 fn main(){
2     let cronframe = CronFrame::default();
3
4     let greeting_john = Greeting::new_cron_obj(
5         "John".into(), CronFrameExpr::new("0", "0", "8*", "*", "*", "Mon-Fri",
6         "*",
7         0));
7 }
```

At this point, we have our cronframe instance with the scheduler not running, and we have our Greeting instance so we need to pass the method job to cronframe either before or after we start the scheduler.

This time around we will pass the job before starting the scheduler:

```
1 fn main(){
2     let cronframe = CronFrame::default();
3     let greeting_john = Greeting::new_cron_obj(
4         "John".into(),
5         CronFrameExpr::new("0", "0", "8*", "*", "*", "Mon-Fri", "*", 0)
6     );
7     greeting_john.cf_gather_mt(cronframe.clone());
8     cronframe.run();
9 }
```

At this point the full code would look like this:

```
1 #[macro_use]
2 extern crate cronframe;
3 use cronframe::{CronFrame, CronFrameExpr};
4
5 #[cron_obj]
6 struct Greeting{
7     employee: String,
8     my_expr: CronFrameExpr,
9 };
10
11 #[cron_impl]
12 impl Greeting{
13     #[mt_job(expr = "my_expr")]
14     fn custom_greeting_job(self) {
15         println!("Hi {}, have a good morning!", self.employee);
16     }
}
```

```

17 }
18
19 fn main(){
20     let cronframe = CronFrame::default();
21     let greeting_john = Greeting::new_cron_obj(
22         "John", CronFrameExpr::new("0", "0", "18*", "*", "*", "Mon-Fri", "*", 0)
23     );
24
25     let greeting_jane = Greeting::new_cron_obj(
26         "Jane", CronFrameExpr::new("0", "0", "16", "*", "*", "Tue-Thu", "*", 0)
27     );
28
29     greeting_john.cf_gather_mt(cronframe.clone());
30     greeting_jane.cf_gather_mt(cronframe.clone());
31
32     cronframe.run();
33 }
```

Notice how we added another greeting job for Jane which has a different cron expression.

What if we wanted to also keep our generic greeting function job?

We put it in the same place it was before and then pass it to the cronframe instance but this time just to keep things brief we are going to use the cf\_gather method which collects all method and function jobs defined on a cron object.

```

1 #[macro_use]
2 extern crate cronframe;
3 use cronframe::{CronFrame, CronFrameExpr};
4
5 #[cron_obj]
6 struct Greeting {
7     employee: String,
8     my_expr: CronFrameExpr,
9 }
10
11 #[cron_impl]
12 impl Greeting {
13     #[fn_job(expr = "0 0 8 * * Mon-Fri *", timeout = "0")]
14     fn general_greeting_job() {
15         println!("Have a good morning!");
16     }
17
18     #[mt_job(expr = "my_expr")]
19     fn custom_greeting_job(self) {
20         println!("Hi {}, have a good morning!", self.employee);
21     }
22 }
23
24 fn main() {
```

```

25 let cronframe = CronFrame::default();
26
27 let mut greeting_john = Greeting::new_cron_obj(
28     "John".into(),
29     "0 0 18 * * Mon-Fri * 0".into()
30 );
31
32 let mut greeting_jane = Greeting::new_cron_obj(
33     "Jane".into(),
34     CronFrameExpr::from("0 0 16 * * Tue-Thu * 0")
35 );
36
37 greeting_john.cf_gather(cronframe.clone());
38 greeting_jane.cf_gather(cronframe.clone());
39
40 cronframe.run();
41 }
42

```

Notice how the main function is virtually the same as the previous one except for replacing cf\_gather\_mt with cf\_gather. Also notice that since CronFrameExpr implements the From trait, we can directly turn a string into it by either using the function from on the CronFrameExpr or the method into on a string. This is useful in case we don't require to set each field of the cron expression from different sources such as a different variable for each field.

Due to function jobs being related to a cron object type rather than one of its instances, calling cf\_gather twice will only gather the function jobs and pass them to the cronframe instance on the first call.

This is a design choice with the philosophy behind it being that if there is an instance of that type then most likely one wants the functions jobs defined in it to be running as well but if one doesn't then they can simply call the cf\_gather\_mt and only method jobs will be passed to cronframe.

## 6.2. Running Examples

If the example is in a single file like “showcase.rs” use the following command:

```
1 $ cargo run --example showcase
```

If the example is in its crate like `1 weather_alert` do the following:

```
1 $ cd examples/weather_alert
2 $ cargo run
```

Let us now have a look at one of the available examples that shows the framework in use for what might be a real world use.

### 6.3. Weather Alert Scenario - Example

Now that we have seen how to use cronframe, let us have a look at a slightly more complex job that might resemble some real use case scenario of the framework in an application.

The job we are looking at is a weather alert job: given the name of a city, we get weather information by making a call to the openweather API and print messages relative to weather conditions.

The job is scheduled to run every ten minutes between 5 am and 7 am and between 2 pm and 4 pm UTC, only from Monday to Friday.

We might want weather alter jobs for different cities so the best way to approach this would be using a method job where we have a cron object with a city field that is used by the job to make the request to the openweather API.

The imports from the library and the cron object definition will look something like the following:

```
1 #[macro_use] extern crate cronframe;
2
3 use cronframe::{CronFrame, CronFrameExpr};
4 use chrono::Local;
5
6 #[cron_obj]
7 struct WeatherAlert {
8     city: String,
9     schedule: CronFrameExpr,
10 }
```

The cron object is very straightforward, we have a city field of type String where the name of our city will be stored and then we have a filed name schedule which is of type CronFrameExpr since we need to provide the schedule for such a job. We added the import of Local from the chrono crate to print the time during each job execution with the Local timezone of the machine that is running the job.

It is now time to define our job.

```
1 #[cron_impl]
2 impl WeatherAlert {
3     #[mt_job(expr = "schedule")]
4     fn weather_alert(self) {
5         println!("Weather alert job for {}", self.city);
6
7         let url_coord = format!(
8             "http://api.openweathermap.org/geo/1.0/direct?q={}&limit={}
9             &appid={}",
10             self.city, 5, API_KEY
11         );
12
13         let resp: serde_json::Value = reqwest::blocking::get(url_coord)
14             .unwrap()
15             .json()
```

```

15     .unwrap();
16
17     let latitude = resp[0]["lat"].clone();
18     let longitude = resp[0]["lon"].clone();
19
20     println!("latitude = {latitude}");
21     println!("longitude = {longitude}");
22
23     let url_weather = format!(
24         "https://api.openweathermap.org/data/2.5/weather?lat={}&lon={}"
25         &appid={}",
26         latitude, longitude, API_KEY
27     );
28
29     let resp: serde_json::Value = reqwest::blocking::get(url_weather)
30         .unwrap()
31         .json()
32         .unwrap();
33
34     let weather_id: i32 = resp["weather"][0]["id"]
35         .clone()
36         .to_string()
37         .parse()
38         .unwrap();
39
40     println!("weather_id = {weather_id}");
41
42     println!("{}: ", Local::now());
43     match weather_id {
44         200..=232 => println!(
45             "!! Thread Carefully: Thunderstorm in {} !!", self.city),
46         500..=531 => println!("! Thread Carefully: Rain in {} !",
47             self.city),
48         600..=622 => println!("! Thread Carefully: Snow in {} !",
49             self.city),
50         781 => println!("!!! Seek Shelter: Tornado in {} !!!", self.city),
51         _ => println!("Nothing to worry about in {}", self.city),
52     }
53 }

```

The code for this is highly dependent on your weather service of choice. For openweather we first need to make an API call to get the coordinates of our city and then use those coordinates to make another call to get the weather data of that city.

Weather conditions are coded within numerical ranges for specific conditions, again this is highly dependent on the service used.

In our case, we alert the user about 4 possible conditions just to make things simple.

These conditions are:

- Thunderstorm
- Rain

- Snow
- Tornado

And just to be nice, we also tell them if there is nothing to worry about the weather so they can be relaxed on their way.

What is left to do is init the framework, create an instance of the cron object with our preferred city, and give the job to the framework instance to run.

```

1 fn main() {
2     let cronframe = CronFrame::default();
3
4     let alert_schedule = CronFrameExpr::new(
5         "0",
6         "0/10",
7         "5-6,14-15",
8         "*",
9         "*",
10        "Mon-Fri",
11        "*",
12        0
13    );
14
15    let mut venice = WeatherAlert::new_cron_obj("Venice".into(),
16        alert_schedule);
17
18    venice.cf_gather(cronframe.clone());
19
20    cronframe.run();
21 }
```

The whole code will look like this:

```

1 #[macro_use] extern crate cronframe;
2
3 use cronframe::{CronFrame, CronFrameExpr};
4 use chrono::Local;
5
6 #[cron_obj]
7 struct WeatherAlert {
8     city: String,
9     schedule: CronFrameExpr,
10 }
11
12 #[cron_impl]
13 impl WeatherAlert {
14     #[mt_job(expr = "schedule")]
15     fn weather_alert(self) {
16         println!("Weather alert job for {}", self.city);
17
18         let url_coord = format!(
19             "http://api.openweathermap.org/geo/1.0/direct?q={}&limit={}appid={}",
20             self.city, 5, API_KEY
21     )
22     }
23 }
```

```

21 );
22
23 let resp: serde_json::Value = reqwest::blocking::get(url_coord)
24 .unwrap()
25 .json()
26 .unwrap();
27
28 let latitude = resp[0]["lat"].clone();
29 let longitude = resp[0]["lon"].clone();
30
31 println!("latitude = {latitude}");
32 println!("longitude = {longitude}");
33
34 let url_weather = format!(
35 "https://api.openweathermap.org/data/2.5/weather?lat={}&lon={}
36 &appid={}",
37 latitude, longitude, API_KEY
38 );
39
40 let resp: serde_json::Value = reqwest::blocking::get(url_weather)
41 .unwrap()
42 .json()
43 .unwrap();
44
45 let weather_id: i32 = resp["weather"][0]["id"]
46 .clone()
47 .to_string()
48 .parse()
49 .unwrap();
50
51 println!("weather_id = {weather_id}");
52
53 println!("{}: ", Local::now());
54 match weather_id {
55 200..=232 => println!(
56 "!! Thread Carefully: Thunderstorm in {} !!", self.city),
57 500..=531 => println!("! Thread Carefully: Rain in {} !", self.city),
58 600..=622 => println!("! Thread Carefully: Snow in {} !", self.city),
59 781 => println!("!!! Seek Shelter: Tornado in {} !!!", self.city),
60 _ => println!("Nothing to worry about in {}", self.city),
61 }
62 }
63
64 fn main() {
65 let cronframe = CronFrame::default();
66
67 let alert_schedule = CronFrameExpr::new(
68 "0",
69 "0/10",
70 "5-6,14-15",
71 "*",
72 "*",

```

```

73     "Mon-Fri",
74     "*",
75     0
76   );
77
78   let mut venice = WeatherAlert::new_cron_obj("Venice".into(),
79   alert_schedule);
80
81   venice.cf_gather(cronframe.clone());
82
83   cronframe.run();
84 }
```

## 6.4. CLI Tool Tutorial

### 6.4.1. Installing CronFrame

There are two ways we can install cronframe:

- install it from crates.io with cargo
- install it from source with cargo

Once again, the recommended way is to install it using crates.io since it will guarantee a working version.

#### 6.4.1.1. Installing from Crates.io

Open your preferred terminal emulator and run the following command:

```
1 $ cargo install cronframe
```

This will compile the binary for cronframe and put it inside .cargo/bin directory in the home folder.

#### 6.4.1.2. Installing from GitHub Repo

Installing from the repo is easy as well.

The first thing to do is to get the code, either by cloning the repo or downloading a release, the latter is recommended.

Once we have the code, we open a terminal instance inside it and run the following command:

```
1 $ cargo install --path .
```

We are essentially using cargo to install from a local source, so the binary ends up in the same place as stated above.

### 6.4.2. Using CronFrame CLI

The CLI tool uses the clap arguments parser library which allows it to have a very user-friendly helper functionality for learning how commands work.

To learn which commands are available we can simply run either of the following:

```
1 $ cronframe
```

```
1 $ cronframe help
```

Both of these will print the same message which is a help message containing a list of available commands with their descriptions which for version 0.1.3 looks like this:

```
1 A Macro Annotation Cron Job Framework with Web Server and CLI Tool.
2
3 Usage: cronframe <COMMAND>
4
5 Commands:
6 start      Start the CronFrame Webserver and Job Scheduler in the
background.
7 run        Run the CronFrame Webserver and Job Scheduler in the terminal.
8 add        Adds a new cli job to a CronFrame instance.
9 load       Load jobs from definition file.
10 scheduler Perform actions on the scheduler like start and stop
11 shutdown   Shutdown the CronFrame Webserver and Job Scheduler.
12 help       Print this message or the help of the given subcommand(s)
13
14 Options:
15 -h, --help    Print help
16 -V, --version Print version
```

We have six commands at our disposal, each is documented further if we use the help command followed by the command we want to know more of:

```
1 $ cronframe help add
2
3 Adds a new cli job to a CronFrame instance.
4
5 Usage: cronframe add [OPTIONS] [EXPR] [TIMEOUT] [JOB]
6
7 Arguments:
8 [EXPR]      The Cron Expression to use for job scheduling.
9 [TIMEOUT]   The value in ms to use for the timeout.
10 [JOB]       The path containing the source code of the job.
11
12 Options:
13 -p, --port <VALUE>
14 -h, --help     Print help
15 -V, --version Print version
```

Learning the tool is something the tool itself takes care of which is nice to avoid having to read some lengthy tutorial or documentation, the following explanation will only further clarify what each command does and why it has been made that way.

#### **6.4.2.1. start command**

This command is used to start a cronframe instance on the machine. Such an instance is run in the background therefore it is completely independent from the terminal instance that launched it. By default, it will be running at <http://127.0.0.1:8098>. The scheduler has already started, but unless we load or add jobs to it, it hasn't got much to do. Under the hood, the start command simply spawns the run command in the background so they are virtually the same. Use start when you want your cronframe instance to keep running even if the terminal closes.

#### **6.4.2.2. run command**

This command is used to run a cronframe instance on the machine. Such an instance is run in the terminal that launched it. By default, it will be running at <http://127.0.0.1:8098>. The scheduler has already started, but unless we load or add jobs to it, it hasn't got much to do. Use run when you want your cronframe instance to quit on terminal closure.

#### **6.4.2.3. add command**

The add command allows us to make use of a type of job that is available with the CLI tool, that is CLI jobs. A CLI job is an executable built from a Rust source that can either be a single source file or an entire crate.

Using the command requires a cron expression, a timeout, and the path to the source as in the following example:

```
1 $ cronframe add "0 0/30 9-10 * * Mon-Wed" 0 path/to/my_project_job
```

**Note:** the cron expression is taken as a single argument so it must be surrounded by double quotes.

#### **6.4.2.4. load command**

The load command reads a txt file containing lines of arguments for multiple add commands to execute automatically. If the -file option is not passed to the command, the used file will be `jobs_list.txt` located inside the `.cronframe` directory.

An example txt file would be:

```
1 "0 0/30 9-10 * * Mon-Wed" 0 path/to/my_project_job
2 "0 0/30 9,12 * * Thu-Fri" 0 path/to/easy_job.rs
3 "0/5 0 14 * * *" 50000 path/to/timeout.rs
```

#### **6.4.2.5. scheduler command**

This command takes an argument that can have only two values:

- start
- stop

There is no need to use it to start the scheduler after either one of the start or the run commands, they take care of it themselves.

A `-port` option is available to target other possible running instances.

#### **6.4.2.6. shutdown command**

This command will provide a graceful shutdown of the instance. It doesn't take a port argument since it is meant to be used only with the cronframe instance created by the CLI tool. Applications that use cronframe as a framework are supposed to handle shutdown on their own.

## 7. Testing

This chapter presents the testing suite that comes with the framework which is only available if the framework is downloaded from GitHub. Downloading cronframe from crates.io does not include the testing suite since it is not vital for projects that use it.

Since Rust comes with a useful native framework for unit and integration testing [15], several tests have been written to check the functionality of the framework.

There are three categories of tests, each apt at checking one peculiarity of execution:

- standard tests check that a job is run according to the scheduler and without failures
- timeout tests check that a job is run according to the scheduler and it timeouts when it should
- failure tests check that a job is run according to the scheduler and it fails

There is a total of 9 tests, 3 per category of test and 1 per category of jobs, that is to say, one per global, function, and method jobs.

Code in the documentation is also tested which is a feature Rust provides to ensure example code compiles, however, due to it being example code it resents code that is not significant for testing purposes.

Including documentation code, the total number of tests jumps up to 17 total.

The duration of tests has different timespans from tens of seconds to tens of minutes and the entire duration of a full testing execution is a bit more than one hour.

Testing can either be done on the full suite of tests or a single test at a time, in case it is done on the full suite it must be done sequentially with 1 thread of execution since it relies on the logger output and the logger is shared in the testing environment.

Running the entire suite of tests requires downloading the source code from the repo and then running the following command

```
1 $ cargo test -- --test-threads=1
```

To show extensive test output run:

```
1 $ cargo test -- --test-threads=1 --show-output
```

To run individual tests we need to first select a submodule of the test suite.

There are 3 of them:

- global
- function
- method

Then we select the test we want to run by its name:

- <job\_type>\_job\_<test\_type>

Examples:

```
1 $ cargo test global::global_job_std
```

```
1 $ cargo test function::function_job_fail
```

```
1 $ cargo test method::method_job_timeout
```

## 7.1. Available Test Modules

For a better organization of the suit, it has been split into modules that separate the tests per type of jobs they target.

### 7.1.1. Tests in the Global Module

The tests available in this module are:

- **global\_job\_std**
  - it runs for a total of 15 seconds
  - the job is scheduled to run every 5 seconds
  - there is no timeout set
  - it shouldn't fail
- **global\_job\_fail**
  - it runs for a total of 15 seconds
  - the job is scheduled to run every 5 seconds
  - there is no timeout set
  - it should fail
- **global\_job\_timeout**
  - it runs for a total of 30 seconds
  - the job is scheduled to run every 5 seconds
  - there is a timeout after 15 seconds from the first job execution
  - it shouldn't fail

### 7.1.2. Tests in the Function Module

The tests available in this module are:

- **function\_job\_std**
  - it runs for a total of 5 minutes
  - the job is scheduled to run every 1 minute
  - there is no timeout set
  - it shouldn't fail
- **function\_job\_fail**
  - it runs for a total of 5 minutes
  - the job is scheduled to run every 1 minute
  - there is no timeout set
  - it should fail
- **function\_job\_timeout**
  - it runs for a total of 5 minutes
  - the job is scheduled to run every 1 minute
  - there is a timeout after 3 minutes from the first job execution
  - it shouldn't fail

### 7.1.3. Tests in the Method Module

The tests available in this module are:

- **method\_job\_std**
  - it runs for a total of 15 minutes
  - the job is scheduled to run every 5 minutes
  - there is no timeout set
  - it shouldn't fail
- **method\_job\_fail**
  - it runs for a total of 15 minutes
  - the job is scheduled to run every 5 minutes
  - there is no timeout set
  - it should fail
- **method\_job\_timeout**
  - it runs for a total of 20 minutes
  - the job is scheduled to run every 5 minutes
  - there is a timeout after 12 minutes from the first job execution
  - it shouldn't fail

## 7.2. Testing Peculiarities

Testing of such a framework was revealed to be less straightforward than expected. The first major problem was the log instance that could be initialized only once in the testing environment. While tests have a separate environment when they execute, the log instance they use is global and shared.

It was necessary to write a function that allowed to init the log only once independently of which test runs first. Since the tests make assertions based on log file timestamps and messages, the log init also had to take care of defining a specific output file for each test by modifying the log configuration.

Rust testing allows for multiple threads of execution to run a testing suite in the least amount of time possible, that is if your tests allow for it. Due to the use of the shared instance of the logger and the tests relying on the output files the logger produces, it is not possible as of the moment of writing to run more than one test at the same time.

Should future logger architecture changes make parallel tests possible, there is the need to account for one more thing. That is having different ports for every running test, otherwise, the initialization of the framework will fail due to a port bind error and only one test will be able to run to completion.

The testing suite could always add more tests, nothing can be tested enough, no matter how extensively tested it is. For this reason, it would perhaps be neat to have some os-dependent tests that target Linux, Windows, and macOS specifically. However since I do not own a Mac, I opted for an os-independent approach to testing which is why there aren't any tests written for the CLI tool. Despite this, I have tested the tool on both Linux and Windows on my machine and the functionality was working as expected.

## 8. Project Reception

This chapter explains what was behind the publishing process of the framework and CLI tool as well as lists all sites related to cronframe, such as the code repository, the website, and community sharing posts.

The entire code base for the project is available on GitHub at <https://github.com/antcim/cronframe> under either one of the MIT or Apache 2.0 licenses.

Most Rust libraries, frameworks, and binaries end up on crates.io, and CronFrame followed suit.

The published crates are:

- cronframe
  - this is where both the framework and the CLI tool are contained
- cronframe\_macro
  - this is where the macros are contained

The reason for having macros inside their crate is owed to restrictions that Rust currently imposes on crates containing macros which might be lifted in the future.

Ideally, the macro crate will not need to stand on its own and it will be turned into a module of the cronframe crate which is exactly what it was before publishing on crates.io.

By publishing on crates.io, both the framework and CLI tool become available via cargo which is quite convenient for quick and easy installation of either one or both.

The homepage of crates.io also provides visibility for the brief period that the published crates remain in the list of the latest publications or the list of the recently updated ones.

This visibility led to a total number of downloads across four versions of cronframe to be over 1000, in particular, the peak of downloads in a single day is 118 for version 0.1.3.

As expected, the greatest number of downloads are observed upon publication of the crate itself or when a new version is published.

To further present the project to Rust developers a post on Rust's subreddit has been posted, available at: <https://www.reddit.com/r/rust/comments/1ekjgh6>

It received, at the time of writing:

- 15 thousand views
- an upvote of 84%
- 6 shares
- 2 comments

The two comments were illuminating on how the project was perceived by the public and one particularly offered interesting suggestions for possible improvements.

Due to the framework being in early stages the reception it had is more than expected and overall positive since Rust developers are known to be a very opinionated bunch, myself being one of them.

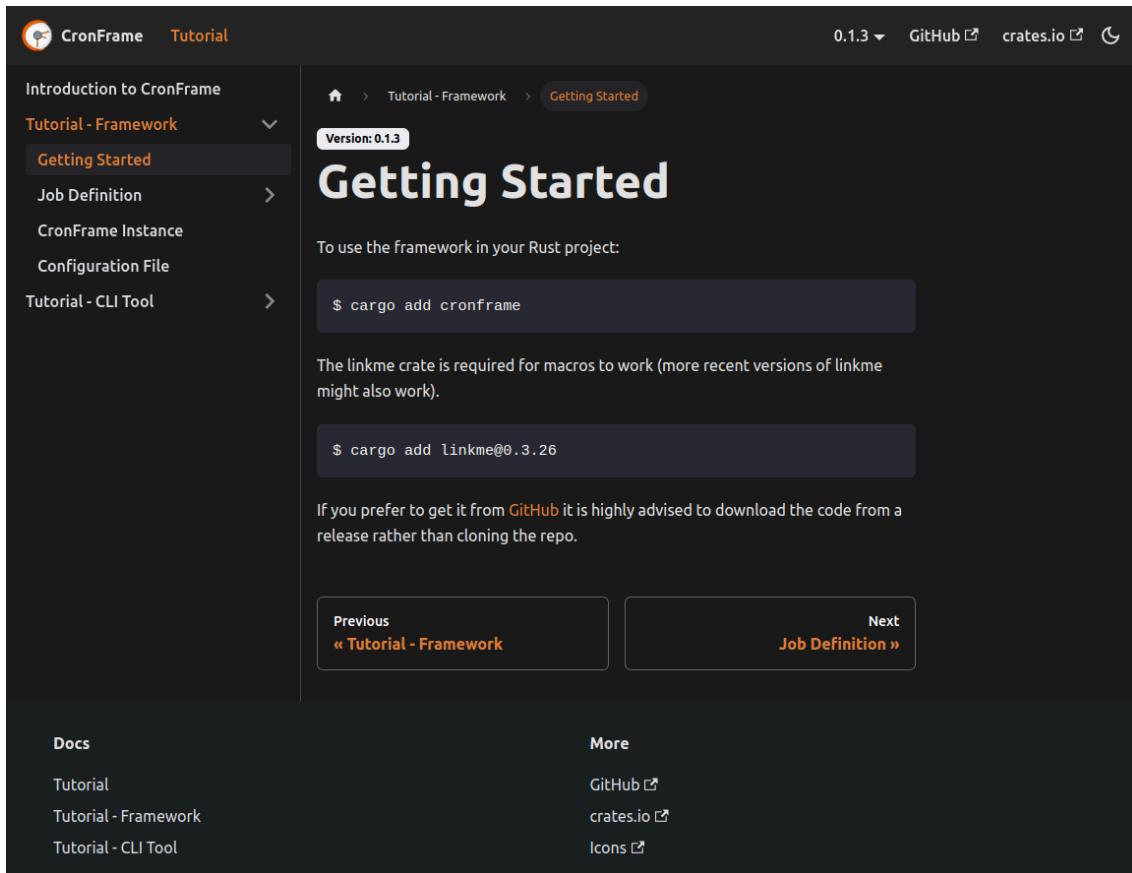


Figure 4: Tutorial Page of the Website

A website providing a step-by-step tutorial has been made using docusaurus [16] and published with GitHub pages, available at [https://antcim.github.io/cronframe\\_site/](https://antcim.github.io/cronframe_site/), it allows for a more direct dive-in into the workings of the framework and most importantly it supports automatic versioning of the tutorials.

## 9. Development Summary

This chapter is left as a footnote to any reader to give insight into the thought process that brought about this work of thesis and also delve into new possible additions that future versions of the framework and CLI tool might see.

### 9.1. Notes

Development started wanting a library that made it possible to use annotation-like syntax on top of functions to define cron jobs but the library base soon evolved into a compact framework quite naturally.

Two main items were the bulk of the work on the framework:

- the annotation macros
- the architecture

Macro syntax has been streamlined as much as possible from early versions that required much more boilerplate code from the user.

The idea for the framework was to make it nearly a plug-and-play dependency which has been achieved as far as:

- small setup for the framework instance
- little code for job definition
- quick configuration

From an architectural perspective, a mixed composition of structs and enums was found to be the better choice, especially for the maintainability of the code in the long run compared to making it full structs or full enums.

No new traits have been defined but Drop and From traits have been implemented for some types for a better integration with the Rust language.

While a cron object's functionality could be thought of as a trait its implementation requires modification of the original struct type which can only be achieved with macros. Injecting all the additional functionality with the macros without a "middleman" trait felt like the right way to do it.

Perhaps a "CronObj" trait would be a neat way to organize what a cron object must contain and have a quick and easy overview of the functionality rather than skimming through macro code but this is much more of a framework developer matter, and most definitely something a user of the framework would overlook.

Supporting only UTC for the job scheduler is a matter of keeping the code more streamlined, it especially makes macros feel as simple as possible both to write and to use.

### 9.2. Further Development

Further development of the framework could focus on two things:

- new functionalities
- better user experience

As to new functionalities what could be done is a lot, starting from supporting different time-zones for scheduling to the recovery of failed jobs either due to app failure or machine failure.

Custom notifications for job completion to be sent via email or other means might also be a feature of interest but the way they would impact macro definitions might be too cumbersome.

Should the framework evolve to offer quite a lot more, it would also be better to turn some functionalities into opt-in features to be enabled in Cargo.toml. This would allow a user to get only the necessary features for their needs.

Last but not least it would be nice to have a function-like macro for cron objects so that they could be defined as follows:

```
1 cron_obj!(
2     struct User{
3         name: String,
4         expr: CronFrameExpr,
5     }
6
7     #[job(expr = "* * * * *", timeout = "0")]
8     fn function_job(){}
9     // do something...
10    }
11
12    #[job(expr = "expr")]
13    fn function_job(self){
14        // do something...
15    }
16 );
```

This would make a cron object even more immediate to define as well as make the code feel more of a cohesive entity. From a functional standpoint, however, it would simply serve as syntactic sugar to the current definition with the cron\_obj and the cron\_impl macros. Still, something that might be worth trying.

Some kind of custom syntax to further simplify macros might also be experimented with, though it being custom might be negatively received by any user who strictly wants to use the native syntax of Rust. For this reason, using something like the function macro above would be the safest option of the two.

## Bibliography

- [1] “Cron CLI.” [Online]. Available: <https://en.wikipedia.org/wiki/Cron>
- [2] “Learn Rust.” [Online]. Available: <https://doc.rust-lang.org/book/>
- [3] “Cargo Package Manager.” [Online]. Available: <https://doc.rust-lang.org/cargo/>
- [4] “Rust Macro System.” [Online]. Available: <https://doc.rust-lang.org/book/ch19-06-macros.html>
- [5] “Programming Languages Software Award 2024.” [Online]. Available: [https://www.sigplan.org/Awards/Software/#2024\\_The\\_Rust\\_Programming\\_Language](https://www.sigplan.org/Awards/Software/#2024_The_Rust_Programming_Language)
- [6] “Developer Survey 2024.” [Online]. Available: <https://survey.stackoverflow.co/2024/>
- [7] “Declarative Macros in Rust.” [Online]. Available: <https://veykril.github.io/tlborm/decl-macros.html>
- [8] “Procedural Macros in Rust.” [Online]. Available: <https://veykril.github.io/tlborm/proc-macros.html>
- [9] “Syn: Rust Parsing Library.” [Online]. Available: <https://docs.rs/syn>
- [10] “Quote: Rust Quasi-quoting Library.” [Online]. Available: <https://docs.rs/quote>
- [11] “Inventory Crate.” [Online]. Available: <https://docs.rs/inventory>
- [12] “Linkme Crate.” [Online]. Available: <https://docs.rs/linkme>
- [13] “Rocket Web Framework.” [Online]. Available: <https://api.rocket.rs/v0.5/rocket/>
- [14] “Logging Library: log4rs.” [Online]. Available: <https://docs.rs/log4rs>
- [15] “Rust Testing.” [Online]. Available: <https://doc.rust-lang.org/rust-by-example/testing.html>
- [16] “Docusaurus.” [Online]. Available: <https://docusaurus.io/docs>

# Appendix

## A - Code Listing for cronframe crate

### A-1 - src/lib.rs

```
1 #[doc(hidden)]
2 #[macro_use] extern crate rocket;
3
4 pub use cronframe_macro::{cron, cron_impl, cron_obj, fn_job, mt_job};
5 #[doc(hidden)]
6 pub use linkme::distributed_slice;
7 #[doc(hidden)]
8 pub use std::{
9     any::{self, Any,TypeId},
10    sync::{Arc, Mutex},
11 };
12
13 #[doc(hidden)]
14 pub use crossbeam_channel::{bounded, unbounded, Receiver, Sender};
15
16 #[doc(hidden)]
17 pub use log::info;
18 #[doc(hidden)]
19 pub use once_cell::sync::Lazy;
20 #[doc(hidden)]
21 pub use std::sync::Once;
22
23 // lib modules
24 pub mod config;
25 pub mod cronframe;
26 pub mod cronframe_expr;
27 pub mod cronjob;
28 pub mod job_builder;
29 pub mod logger;
30 pub mod utils;
31 pub mod web_server;
32
33 // re-export of types
34 pub use cronframe::CronFrame;
35 pub use job_builder::JobBuilder;
36 pub use cronjob::CronJob;
37 pub use cronframe_expr::CronFrameExpr;
38
39 #[doc(hidden)]
40 pub use inventory::{collect, submit};
41
42 // necessary to gather all the global jobs automatically
43 collect!(JobBuilder<'static>);
44
45 /// Used in the init function of the CronJob type to account for the type
46 #[derive(Debug, Clone)]
```

```

47 pub enum CronJobType {
48     Global(fn()),
49     Method(fn(arg: Arc<Box<dyn Any + Send + Sync>>)),
50     Function(fn()),
51     CLI
52 }
53
54 /// Used in the init function of the CronFrame type to filter in a single
55 /// type of job for execution.
56 /**
57 * #[macro_use] extern crate cronframe_macro;
58 * use cronframe::{CronFilter, CronFrame};
59 */
60 /**
61 * fn main(){
62 *     // allow execution of Global Jobs Only
63 *     let cronframe = CronFrame::init(Some(CronFilter::Global), true);
64 *     // no filters for the job type
65 *     //let cronframe = CronFrame::init(None, true);
66 * }
67 */
68 #[derive(PartialEq, Clone, Copy)]
69 pub enum CronFilter {
70     Global,
71     Function,
72     Method,
73     CLI
74 }
```

## A-2 - src/job\_builder.rs

```

1  //! Builder Type for CronJob
2
3  use crate::cronjob::CronJob;
4  use crate::CronJobType;
5  use chrono::Duration;
6  use cron::Schedule;
7  use std::any::Any;
8  use std::str::FromStr;
9  use std::sync::Arc;
10 use uuid::Uuid;
11
12 /// Type returned by the helper functions generated by cron, fn_job and
13 /// mt_job macros.
14 /**
15 * It builds a CronJob instance by calling the build method.
16 */
17 #[derive(Debug, Clone)]
18 pub enum JobBuilder<'a> {
19     Global {
20         name: &'a str,
21         job: fn(),
22         cron_expr: &'a str,
23         timeout: &'a str,
```

```

22     },
23     Method {
24         name: &'a str,
25         job: fn(arg: Arc<Box<dyn Any + Send + Sync>>),
26         cron_expr: String,
27         timeout: String,
28         instance: Arc<Box<dyn Any + Send + Sync>>,
29     },
30     Function {
31         name: &'a str,
32         job: fn(),
33         cron_expr: &'a str,
34         timeout: &'a str,
35     },
36
37     CLI {
38         name: &'a str,
39         cron_expr: &'a str,
40         timeout: &'a str,
41     },
42 }
43
44 impl<'a> JobBuilder<'a> {
45     pub const fn global_job(
46         name: &'a str,
47         job: fn(),
48         cron_expr: &'a str,
49         timeout: &'a str,
50     ) -> Self {
51         JobBuilder::Global {
52             name,
53             job,
54             cron_expr,
55             timeout,
56         }
57     }
58
59     pub const fn method_job(
60         name: &'a str,
61         job: fn(arg: Arc<Box<dyn Any + Send + Sync>>),
62         cron_expr: String,
63         timeout: String,
64         instance: Arc<Box<dyn Any + Send + Sync>>,
65     ) -> Self {
66         JobBuilder::Method {
67             name,
68             job,
69             cron_expr,
70             timeout,
71             instance,
72         }
73     }
74 }
```

```

75      pub const fn function_job(
76          name: &'a str,
77          job: fn(),
78          cron_expr: &'a str,
79          timeout: &'a str,
80      ) -> Self {
81          JobBuilder::Function {
82              name,
83              job,
84              cron_expr,
85              timeout,
86          }
87      }
88
89      pub const fn cli_job(name: &'a str, cron_expr: &'a str, timeout: &'a
90 str) -> Self {
91          JobBuilder::CLI {
92              name,
93              cron_expr,
94              timeout,
95          }
96
97      // it matches on the job variant to build and builds it
98      pub fn build(self) -> CronJob {
99          match self {
100              Self::Global {
101                  name,
102                  job,
103                  cron_expr,
104                  timeout,
105              } => {
106                  let schedule =
107                      Schedule::from_str(cron_expr)
108                          .expect("Failed to parse cron expression!");
109                  let timeout: i64 = timeout.parse()
110                      .expect("Failed to parse timeout!");
111                  let timeout = if timeout > 0 {
112                      Some(Duration::milliseconds(timeout))
113                  } else {
114                      None
115                  };
116
117                  CronJob {
118                      name: name.to_string(),
119                      id: Uuid::new_v4(),
120                      job: CronJobType::Global(job),
121                      schedule,
122                      timeout,
123                      timeout_notified: false,
124                      status_channels:
125                      Some(crossbeam_channel::bounded(1)),
126                      life_channels: None,

```

```

126             start_time: None,
127             run_id: None,
128             method_instance: None,
129             failed: false,
130             suspended: false,
131         }
132     }
133     Self::Method {
134         name,
135         job,
136         cron_expr,
137         timeout,
138         instance,
139     } => {
140         let schedule =
141             Schedule::from_str(&cron_expr)
142             .expect("Failed to parse cron expression!");
143         let timeout: i64 = timeout.parse()
144             .expect("Failed to parse timeout!");
145         let timeout = if timeout > 0 {
146             Some(Duration::milliseconds(timeout))
147         } else {
148             None
149         };
150
151         CronJob {
152             name: name.to_string(),
153             id: Uuid::new_v4(),
154             job: CronJobType::Method(job),
155             schedule,
156             timeout,
157             timeout_notified: false,
158             status_channels:
159             Some(crossbeam_channel::bounded(1)),
160             life_channels: None,
161             start_time: None,
162             run_id: None,
163             method_instance: Some(instance),
164             failed: false,
165             suspended: false,
166         }
167     }
168     Self::Function {
169         name,
170         job,
171         cron_expr,
172         timeout,
173     } => {
174         let schedule =
175             Schedule::from_str(cron_expr)
176                 .expect("Failed to parse cron expression!");
177         let timeout: i64 = timeout.parse()
178             .expect("Failed to parse timeout!");

```

```

178         let timeout = if timeout > 0 {
179             Some(Duration::millisoseconds(timeout))
180         } else {
181             None
182         };
183
184         CronJob {
185             name: name.to_string(),
186             id: Uuid::new_v4(),
187             job: CronJobType::Function(job),
188             schedule,
189             timeout,
190             timeout_notified: false,
191             status_channels:
192                 Some(crossbeam_channel::bounded(1)),
193                 life_channels: None,
194                 start_time: None,
195                 run_id: None,
196                 method_instance: None,
197                 failed: false,
198                 suspended: false,
199             }
200         }
201     Self::CLI {
202         name,
203         cron_expr,
204         timeout,
205     } => {
206         let cron_expr = cron_expr.replace("slh", "/")
207             .replace("%20", " ");
208         let schedule =
209             Schedule::from_str(&cron_expr)
210             .expect("Failed to parse cron expression!");
211         let timeout: i64 = timeout.parse()
212             .expect("Failed to parse timeout!");
213         let timeout = if timeout > 0 {
214             Some(Duration::millisoseconds(timeout))
215         } else {
216             None
217         };
218
219         CronJob {
220             name: name.to_string(),
221             id: Uuid::new_v4(),
222             job: CronJobType::CLI,
223             schedule,
224             timeout,
225             timeout_notified: false,
226             status_channels:
227                 Some(crossbeam_channel::bounded(1)),
228                 life_channels: None,
229                 start_time: None,
230                 run_id: None,

```

```

229             method_instance: None,
230             failed: false,
231             suspended: false,
232         }
233     }
234 }
235 }
236 }
237

```

### A-3 - src/cronjob.rs

```

1  ///! CronJob type, built by JobBuilder
2
3  use crate::{utils, CronJobType};
4  use chrono::{DateTime, Duration, Utc};
5  use cron::Schedule;
6  use crossbeam_channel::{Receiver, Sender};
7  use std::{any::Any, process::Command,
8    str::FromStr, sync::Arc, thread::JoinHandle};
9  use uuid::Uuid;
10
11 /// This type collects all necessary data
12 /// for a cron job to be used in the scheduler.
13 /**
14 /// While it could be used directly there are macros that build jobs for
15 /// you.
16 #[derive(Debug, Clone)]
17 pub struct CronJob {
18     pub suspended: bool,
19     pub name: String,
20     pub id: Uuid,
21     pub job: CronJobType,
22     pub schedule: Schedule,
23     pub timeout: Option<Duration>,
24     pub timeout_notified: bool,
25     pub status_channels: Option<(Sender<String>, Receiver<String>)>,
26     pub life_channels: Option<(Sender<String>, Receiver<String>)>,
27     pub start_time: Option<DateTime<Utc>>,
28     pub run_id: Option<Uuid>,
29     pub method_instance: Option<Arc<Box<dyn Any + Send + Sync>>>,
30     pub failed: bool,
31 }
32 impl CronJob {
33     // this function is used in the scheduler thread to get a handle if
34     // the job has to be scheduled
35     pub fn try_schedule(&mut self, grace_period: u32) ->
36     Option<JoinHandle<()>> {
37         if self.check_schedule() {
38             self.run_id = Some(Uuid::new_v4());
39             //self.status_channels = Some(crossbeam_channel::bounded(1));

```

```

38
39         if self.start_time.is_none() {
40             self.start_time = Some(Utc::now());
41         }
42
43         // we try to schedule the job and return its handle
44         // in case scheduling fails for any conflict
45         // we try again for as long as we are in the gracefull period
46
47         if let Ok(handle) = self.run() {
48             return Some(handle);
49         }
50
51         let gracefull_period = grace_period as i64;
52         let first_try = Utc::now();
53         let limit_time = first_try +
Duration::milliseconds(gracefull_period);
54         let mut graceful_log = false;
55
56         while Utc::now() < limit_time {
57             match self.run_graceful() {
58                 Ok(handle) => {
59                     if graceful_log {
60                         let job_id = format!("{} ID#{})", self.name,
self.id);
61                         let run_id = self
62                             .run_id
63                             .expect("run_id unwrap error in
try_schedule")
64                             .to_string();
65                         info!("job @{} RUN_ID#{run_id} -
Scheduled in Graceful Period");
66                     }
67                     return Some(handle);
68                 }
69                 Err(_error) => graceful_log = true,
70             }
71         }
72     }
73     None
74 }
75
76 // checks if a job's upcoming schedule is within the next second
77 pub fn check_schedule(&self) -> bool {
78     let now = Utc::now();
79     if let Some(next) = self.schedule.upcoming(Utc).take(1).next() {
80         let until_next = (next - now).num_milliseconds();
81         if until_next <= 1000 {
82             return true;
83         }
84     }
85     false
86 }
```

```

87
88     // the expected value is in milliseconds
89     pub fn set_timeout(&mut self, value: i64) {
90         self.timeout = if value > 0 {
91             Some(Duration::milliseconds(value))
92         } else {
93             None
94         };
95     }
96
97     // used to retrive a job's status to display it in the web server
98     pub fn status(&self) -> String {
99         if self.suspended {
100             "Suspended".to_string()
101         } else if self.check_timeout() {
102             "Timed-Out".to_string()
103         } else if self.run_id.is_some() {
104             "Running".to_string()
105         } else {
106             "Awaiting Schedule".to_string()
107         }
108     }
109
110     // method used by the web server the change the cron expression of a
111     // job
112     pub fn set_schedule(&mut self, expression: &str) -> bool {
113         let expr = expression.replace("slh", "/").replace("%20", " ");
114         if let Ok(schedule) = Schedule::from_str(expr.as_str()) {
115             self.schedule = schedule;
116             return true;
117         }
118         false
119     }
120
121     // returns true if timeout expired
122     pub fn check_timeout(&self) -> bool {
123         if let Some(timeout) = self.timeout {
124             if self.start_time.is_some() {
125                 let timeout = self
126                     .start_time
127                     .expect("start time unwrap error in check_timeout")
128                     + timeout;
129
130                 if Utc::now() >= timeout {
131                     return true;
132                 }
133             }
134             false
135         }
136
137         // it resets the timeout if 24h have passed
138         pub fn reset_timeout(&mut self) {

```

```

139     if let Some(timeout) = self.timeout {
140         if self.start_time.is_some() {
141             let timeout = self
142                 .start_time
143                     .expect("start time unwrap error in timeout_reset")
144                     + timeout;
145
146             if Utc::now() >= timeout + Duration::hours(24) {
147                 self.start_time = None;
148             }
149         }
150     }
151 }
152
153 // get the schedule constructed from the cron expression
154 pub fn schedule(&self) -> String {
155     self.schedule.to_string()
156 }
157
158 // it returns the timeout or "None" if a timeout is not set
159 pub fn timeout_to_string(&self) -> String {
160     if self.timeout.is_some() {
161         let timeout = self
162             .timeout
163                 .expect("timeout unwrap error in timeout_to_string");
164         format!(
165             "{} s \n {} ms",
166             timeout.num_seconds(),
167             timeout.num_milliseconds()
168         )
169     } else {
170         "None".into()
171     }
172 }
173
174 // spells out the type of the job
175 pub fn type_to_string(&self) -> String {
176     match self.job {
177         CronJobType::Global(_) => "Global".to_string(),
178         CronJobType::Function(_) => "Function".to_string(),
179         CronJobType::Method(_) => "Method".to_string(),
180         CronJobType::CLI => "CLI".to_string(),
181     }
182 }
183
184 // if the job is active it returns the schedule otherwise a message
telling why there is no next schedule
185 pub fn upcoming_utc(&self) -> String {
186     if self.suspended {
187         return "None due to scheduling suspension.".to_string();
188     } else if self.check_timeout() {
189         return "None due to timeout.".to_string();
190     }

```

```

191     self.schedule
192         .upcoming(Utc)
193         .into_iter()
194         .next()
195         .expect("schedule unwrap error in upcoming_utc")
196         .to_string()
197     }
198
199     // if the job is active it returns the schedule otherwise a message
telling why there is no next schedule
200     pub fn upcoming_local(&self) -> String {
201         if self.suspended {
202             return "None due to scheduling suspension.".to_string();
203         } else if self.check_timeout() {
204             return "None due to timeout.".to_string();
205         }
206         utils::local_time(
207             self.schedule
208                 .upcoming(Utc)
209                 .into_iter()
210                 .next()
211                 .expect("schedule unwrap error in upcoming_utc"),
212             )
213             .to_string()
214     }
215
216     // it returns the id of the current execution of the job, or "None"
if it is not running
217     pub fn run_id(&self) -> String {
218         match &self.run_id {
219             Some(uuid) => uuid.to_string(),
220             None => "None".into(),
221         }
222     }
223
224     // this spawns a control thread for the job that spawns a thread
with the actual job
225     pub fn run(&self) -> std::io::Result<JoinHandle<()>> {
226         let cron_job = self.clone();
227         let tx = self
228             .status_channels
229             .as_ref()
230             .expect("tx unwap error in job run method")
231             .0
232             .clone();
233         let _rx = self
234             .status_channels
235             .as_ref()
236             .expect("rx unwap error in job run method")
237             .1
238             .clone();
239         let schedule = self.schedule.clone();
240         let job_id = format!("{} ID#{:?}", self.name, self.id);

```

```

241     let run_id = self
242         .run_id
243         .as_ref()
244         .expect("run_id unwrap error in job run method")
245         .clone();
246
247     // the actual job thread
248     // this is spawned from the control thread
249     // it gets the next schedule, waits up to it and runs the job
250     let job_thread = move || {
251         let now = Utc::now();
252         if let Some(next) = schedule.upcoming(Utc).take(1).next() {
253             let until_next = next - now;
254             std::thread::sleep(until_next.to_std().unwrap());
255             info!("job @{job_id} RUN_ID#{run_id} - Execution");
256             match cron_job.job {
257                 CronJobType::Global(job) |
258                 CronJobType::Function(job) => job(),
259                 CronJobType::Method(job) => job(cron_job
260                     .method_instance
261                     .expect("method instance unwrap error in job
262 thread")),
263                 CronJobType::CLI => {
264                     let home_dir = {
265                         let tmp = home::home_dir().unwrap();
266                         tmp.to_str().unwrap().to_owned()
267                     };
268                     let _build = Command::new(format!("./{}",
269                         cron_job.name))
270                         .current_dir(format!("{}{}/cronframe",
271                             home_dir, cronframe))
272                         .status()
273                         .expect("process failed to execute");
274                 }
275             }
276         }
277     };
278
279     // the control thread handle is what gets returned to the
280     // cronframe
281     // this allows to check for job completion or fail
282     let control_thread = move || {
283         let job_handle = std::thread::spawn(job_thread);
284
285         while !job_handle.is_finished() {
286             std::thread::sleep(Duration::milliseconds(250).to_std().unwrap());
287         }
288
289         match job_handle.join() {
290             Ok(_) => {
291                 let _ = tx.send("JOB_COMPLETE".to_string());
292             }
293         }
294     }

```

```

288         Err(_) => {
289             let _ = tx.send("JOB_ABORT".to_string());
290         }
291     };
292 };
293
294     std::thread::Builder::spawn(std::thread::Builder::new(),
control_thread)
295 }
296
297 // same as run but it accounts for graceful period
298 pub fn run_graceful(&self) -> std::io::Result<JoinHandle<()>> {
299     let cron_job = self.clone();
300     let tx = self
301         .status_channels
302         .as_ref()
303         .expect("tx unwrap error in job run_graceful method")
304         .0
305         .clone();
306     let _rx = self
307         .status_channels
308         .as_ref()
309         .expect("rx unwrap error in job run_graceful method")
310         .1
311         .clone();
312     let schedule = self.schedule.clone();
313     let job_id = format!("{} ID{}", self.name, self.id);
314     let run_id = self.run_id.as_ref().unwrap().clone();
315
316     // the actual job thread in graceful period
317     // this is spawned from the control thread
318     // it runs right away
319     let job_thread = move || {
320         let now = Utc::now();
321         if let Some(next) = schedule.upcoming(Utc).take(1).next() {
322             let until_next = next - now;
323
324                 // we sleep only if we haven't yet reached the scheduled
time
325                 // otherwise we immediately execute the job
326             if now < next {
327                 std::thread::sleep(until_next.to_std().unwrap());
328             }
329
330             info!("job @{} RUN_ID#{run_id} - Execution");
331             match cron_job.job {
332                 CronJobType::Global(job) |
CronJobType::Function(job) => job(),
333                 CronJobType::Method(job) => job(cron_job
334                     .method_instance
335                     .expect("method instance unwrap error in job
thread")),
336                 CronJobType::CLI => {

```

```

337             let home_dir = {
338                 let tmp = home::home_dir().unwrap();
339                 tmp.to_str().unwrap().to_owned()
340             };
341             let _build = Command::new(format!("./{}",
342                                         cron_job.name))
343                                         .current_dir(format!("{}{}/.cronframe",
344                                         .status()
345                                         .expect("process failed to execute"));
346             }
347         }
348     };
349
350     // the control thread handle is what gets returned to the
351     // cronframe
352     let control_thread = move || {
353         let job_handle = std::thread::spawn(job_thread);
354
355         while !job_handle.is_finished() {
356
356             std::thread::sleep(Duration::millis(250).to_std().unwrap());
357         }
358
359         match job_handle.join() {
360             Ok(_) => {
361                 let _ = tx.send("JOB_COMPLETE".to_string());
362             }
363             Err(_) => {
364                 let _ = tx.send("JOB_ABORT".to_string());
365             }
366         };
367     };
368
369     std::thread::Builder::spawn(std::thread::Builder::new(),
370                                control_thread)
371 }
372 }
```

## A-4 - src/cronframe.rs

```

1 //! The Core Type of the Framework
2
3 use crate::{
4     config::read_config, cronjob::CronJob, job_builder::JobBuilder,
5     logger, utils, web_server,
6     CronFilter, CronJobType,
7 };
7 use chrono::Duration;
```

```

8 use crossbeam_channel::{Receiver, Sender};
9 use rocket::Shutdown;
10 use std::{
11     collections::HashMap,
12     sync::{Arc, Mutex},
13     thread::JoinHandle,
14 };
15
16 const GRACE_DEFAULT: u32 = 250;
17
18 /// This is the type that provides the scheduling and management of jobs.
19 /**
20 /// It needs to be initialised once to setup the web server and gather
21 global jobs.
22 /**
23 /// Either one of the `start_scheduler` or `run` method must be invoked
24 for it to actually start.
25 /**
26 /// #[macro_use] extern crate cronframe_macro;
27 /// # use cronframe::CronFrame;
28 /// fn main(){
29 ///     let cronframe = CronFrame::default(); // this a shorthand for
30 Cronframe::init(None, true);
31 ///     cronframe.start_scheduler(); //does not keep main alive
32 ///     //cronframe.keep_alive(); // keeps main thread alive
33 ///     //cronframe.run(); //starts the scheduler, keeps main alive
34 /// }
35
36 pub struct CronFrame {
37     pub cron_jobs: Mutex<Vec<CronJob>>,
38     job_handles: Mutex<HashMap<String, JoinHandle<()>>>,
39     _logger: Option<log4rs::Handle>,
40     pub web_server_channels: (Sender<Shutdown>, Receiver<Shutdown>),
41     pub filter: Option<CronFilter>,
42     server_handle: Mutex<Option<Shutdown>>,
43     pub quit: Mutex<bool>,
44     pub grace: u32,
45     pub running: Mutex<bool>,
46 }
47
48 impl CronFrame {
49     /**
50      /// It returns an `Arc<CronFrame>` which is used in the webserver and
51 can be used to start the scheduler.
52     /**
53     /// #[macro_use] extern crate cronframe_macro;
54     /// # use cronframe::CronFrame;
55     /// fn main(){
56         // inits the framework instance and gathers global jobs if
57         there are any
58         // does not start the scheduler, only the web server is live
59         let cronframe = CronFrame::default(); // this a shorthand for
60 Cronframe::init(None, true);
61         //cronframe.keep_alive(); // keeps main thread alive
62         //cronframe.run(); //starts the scheduler, keeps main alive

```

```

55     /// }
56     /**
57      pub fn default() -> Arc<CronFrame> {
58          CronFrame::init(None, true)
59      }
60
61      /// Init function of the framework, it takes two arguments:
62      /**
63      /// filter: Option<CronFilter>
64      /// use_logger: bool
65      /**
66      /**
67      /// It manages:
68      /// - the logger setup if use_logger is true
69      /// - the creation of the CronFrame Instance
70      /// - the collection of global jobs
71      /// - the setup of the web server
72      /**
73      /// It returns an `Arc<CronFrame>` which is used in the webserver and
74      /// to start the scheduler.
74      pub fn init(filter: Option<CronFilter>, use_logger: bool) ->
75      Arc<CronFrame> {
76          println!("Starting CronFrame...");
77
77          let logger = if use_logger {
78              Some(logger::rolling_logger())
79          } else {
80              None
81          };
82
83          let frame = CronFrame {
84              cron_jobs: Mutex::new(vec![]),
85              job_handles: Mutex::new(HashMap::new()),
86              _logger: logger,
87              web_server_channels: crossbeam_channel::bounded(1),
88              filter,
89              server_handle: Mutex::new(None),
90              quit: Mutex::new(false),
91              grace: {
92                  if let Some(config_data) = read_config() {
93                      if let Some(scheduler_data) = config_data.scheduler {
94                          scheduler_data.grace.unwrap_or_else(|| 250)
95                      } else {
96                          GRACE_DEFAULT
97                      }
98                  } else {
99                      GRACE_DEFAULT
100                 }
101             },
102             running: Mutex::new(false),
103         };
104
105         info!("CronFrame Init Start");

```

```

106     info!("Graceful Period {} ms", frame.grace);
107     info!("Collecting Global Jobs");
108
109     for job_builder in inventory::iter::<JobBuilder> {
110         let cron_job = job_builder.clone().build();
111         info!("Found Global Job \"{}\"", cron_job.name);
112         frame
113             .cron_jobs
114             .lock()
115             .expect("global job gathering error during init")
116             .push(cron_job)
117     }
118
119     info!("Global Jobs Collected");
120     info!("CronFrame Init Complete");
121
122     info!("CronFrame Server Init");
123     let frame = Arc::new(frame);
124     let server_frame = frame.clone();
125
126     let running = Mutex::new(false);
127
128     std::thread::spawn(move ||
web_server::web_server(server_frame));
129
130     *frame
131         .server_handle
132             .lock()
133             .expect("web server handle unwrap error") = match
frame.web_server_channels.1.recv() {
134         Ok(handle) => {
135             *running.lock().unwrap() = true;
136             Some(handle)
137         }
138         Err(error) => {
139             error!("Web server shutdown handle error: {error}");
140             None
141         }
142     };
143
144     if *running.lock().unwrap() {
145         let (ip_address, port) = utils::ip_and_port();
146         info!(
147             "CronFrame Web Server running at http://{}:{}",
148             ip_address, port
149         );
150         println!("CronFrame running at http://{}:{}", ip_address,
port);
151     }
152
153     frame
154 }
155

```

```

156     /// It adds a CronJob instance to the job pool
157     /// Used in the cf_gather_mt and cf_gather_fn
158     pub fn add_job(self: &Arc<CronFrame>, job: CronJob) ->
159         Arc<CronFrame> {
160             self.cron_jobs
161                 .lock()
162                     .expect("add_job unwrap error on lock")
163                     .push(job);
164             self.clone()
165         }
166
167     // It creates a new job classified as a global job and adds it to the
168     // job pool
169     pub fn new_job(
170         self: Arc<CronFrame>,
171         name: &str,
172         job: fn(),
173         cron_expr: &str,
174         timeout: &str,
175     ) -> Arc<CronFrame> {
176         self.add_job(JobBuilder::global_job(name, job, cron_expr,
177             timeout).build())
178     }
179
180     /// It spawns a thread that manages the scheduling of the jobs and
181     /// termination of jobs.
182     ///
183     /// This method returns after spawning the scheduler.
184     ///
185     /// Keeping the main thread alive is left to the user.
186     ///
187     /// Use the `run` method to spawn the scheduler and keep the main
188     /// thread alive.
189     ///
190     ///
191     /// #[macro_use] extern crate cronframe_macro;
192     /// use cronframe::CronFrame;
193     fn main(){
194         let cronframe = CronFrame::default().start_scheduler();
195         ///
196         pub fn start_scheduler<'a>(self: &Arc<Self>) -> Arc<Self> {
197             let cronframe = self.clone();
198             let ret = cronframe.clone();
199
200             // if already running, return
201             if *self.running.lock().unwrap() {
202                 return ret;
203             }
204
205             *cronframe
206                 .running
207                     .lock()
208                         .expect("running unwrap error in quit start_scheduler")

```

```

method") = true;
204     *cronframe
205         .quit
206         .lock()
207         .expect("quit unwrap error in start_scheduler method") =
false;
208
209     let scheduler = move || loop {
210         // sleep some otherwise the cpu consumption goes to the moon
211
212         std::thread::sleep(Duration::milliseconds(500).to_std().unwrap());
213
214         if *cronframe
215             .quit
216             .lock()
217             .expect("quit unwrap error in scheduler")
218         {
219             break;
220         }
221
222         if !*cronframe
223             .running
224             .lock()
225             .expect("quit unwrap error in scheduler")
226         {
227             break;
228         }
229
230         let mut cron_jobs = cronframe
231             .cron_jobs
232             .lock()
233             .expect("cron jobs unwrap error in scheduler");
234         let mut jobs_to_remove: Vec<usize> = Vec::new();
235
236         for (i, cron_job) in &mut
237             (*cron_jobs).iter_mut().enumerate() {
238             if let Some(filter) = &cronframe.filter {
239                 let job_type = match cron_job.job {
240                     CronJobType::Global(_) => CronFilter::Global,
241                     CronJobType::Function(_) =>
CronFilter::Function,
242                     CronJobType::Method(_) => CronFilter::Method,
243                     CronJobType::CLI => CronFilter::CLI,
244                 };
245
246                 if job_type != *filter {
247                     continue;
248                 }
249                 let job_id = format!("{} ID#{}", cron_job.name,
250 cron_job.id);
250

```

```

251          // if cron_obj instance related to the job is dropped
252          delete the job
253          let to_be_deleted = if let Some((_, life_rx)) =
254              cron_job.life_channels.clone() {
255                  match life_rx.try_recv() {
256                      Ok(message) => {
257                          if message == "JOB_DROP" {
258                              info!("job @{} - Dropped", job_id);
259                              jobs_to_remove.push(i);
260                              true
261                          } else {
262                              false
263                          }
264                      }
265                  } else {
266                      false
267                  };
268
269          // if the job_id key is not in the hashmap then attempt
270          // to schedule it
271          // if scheduling is a success then add the key to the
272          // hashmap
273
274          let mut job_handlers = cronframe
275              .job_handles
276              .lock()
277              .expect("job handles unwrap error in scheduler");
278
279          // check if the daily timeout expired and reset it if
280          // need be
281          cron_job.reset_timeout();
282
283          // if there is no handle for the job see if it need to
284          // be scheduled
285          if !job_handlers.contains_key(&job_id) && !to_be_deleted
286          {
287              if cron_job.suspended {
288                  continue;
289              }
290
291              // if the job timed-out than skip to the next job
292              if cron_job.check_timeout() {
293                  if !cron_job.timeout_notified {
294                      info!("job @{} - Reached Timeout", job_id);
295                      cron_job.timeout_notified = true;
296                  }
297                  continue;
298              }
299
300              let handle =
301                  (*cron_job).try_schedule(cronframe.grace);

```

```

296
297             if handle.is_some() {
298                 job_handlers.insert(
299                     job_id.clone(),
300                     handle.expect("job handle unwrap error after
try_schedule"),
301                 );
302                 info!(
303                     "job @{} RUN_ID#{} - Scheduled",
304                     job_id,
305                     cron_job.run_id.as_ref().expect("run_id
unwrap error")
306                 );
307             }
308         }
309         // the job is in the hashmap and running
310         // check to see if it sent a message that says it
finished or aborted
311         else if let Some((_, status_rx)) =
312             cron_job.status_channels.clone() {
313             match status_rx.try_recv() {
314                 Ok(message) => {
315                     if message == "JOB_COMPLETE" {
316                         info!(
317                             "job @{} RUN_ID#{} - Completed",
318                             job_id,
319                             cron_job.run_id.as_ref().unwrap()
320                         );
321                         job_handlers.remove(job_id.as_str());
322                         cron_job.run_id = None;
323                     } else if message == "JOB_ABORT" {
324                         info!(
325                             "job @{} RUN_ID#{} - Aborted",
326                             job_id,
327                             cron_job.run_id.as_ref().unwrap()
328                         );
329                         job_handlers.remove(job_id.as_str());
330                         cron_job.run_id = None;
331                         cron_job.failed = true;
332                     }
333                 Err(_error) => {}
334             }
335         }
336     }
337
338     // cleanup of dropped method jobs
339     if !jobs_to_remove.is_empty() {
340         let num_jobs = jobs_to_remove.len();
341         for i in 0..num_jobs {
342             cron_jobs.remove(jobs_to_remove[i]);
343             for j in i + 1..num_jobs {
344                 jobs_to_remove[j] -= 1;

```

```

345             }
346         }
347     }
348 };
349
350     std::thread::spawn(scheduler);
351     info!("CronFrame Scheduler Running");
352     ret
353 }
354
355     /// This function can be used to keep the main thread alive after
the scheduler has been started
356     pub fn keep_alive(self: &Arc<Self>) {
357         loop {
358
std::thread::sleep(Duration::milliseconds(500).to_std().unwrap());
359         if *self.quit.lock().unwrap() {
360             break;
361         }
362     }
363 }
364
365     /// Blocking method that starts the scheduler and keeps the main
thread alive
366     /// Use the `start_scheduler` method if need to start the scheduler
and
367     /// retain control of execution in main
368     pub fn run(self: &Arc<Self>) {
369         self.start_scheduler().keep_alive();
370     }
371
372     /// It quits the running scheduler instance
373     pub fn stop_scheduler(self: &Arc<Self>) {
374         info!("CronFrame Scheduler Shutdown");
375         *self.running.lock().unwrap() = false;
376     }
377
378     /// Function to call for a graceful shutdown of the framework
instance
379     ///
380     /// #[macro_use] extern crate cronframe_macro;
381     /// use cronframe::CronFrame;
382     ///
383     /// fn main(){
384         ///     let cronframe = CronFrame::default();
385         ///     // do somthing...
386         ///     cronframe.start_scheduler();
387         ///     // do other things...
388         ///     cronframe.quit();
389     }
390     ///
391     pub fn quit(self: &Arc<Self>) {
392         self.stop_scheduler();

```

```

393         info!("CronFrame Shutdown");
394
395         // wait for job handlers to finish
396         let cronframe = self.clone();
397
398         let handles = cronframe
399             .job_handles
400             .lock()
401             .expect("job handles unwrap error in stop scheduler
method");
402
403         for handle in handles.iter() {
404             while !handle.l.is_finished() {
405                 // do some waiting until all job threads have
terminated.
406             }
407         }
408
409         // quit the web server
410         self.server_handle
411             .lock()
412             .expect("web server unwrap error in quit method")
413             .clone()
414             .expect("web server unwrap error after clone in quit
method")
415             .notify();
416
417         *self
418             .quit
419             .lock()
420             .expect("quit unwrap error in stop scheduler method") =
true;
421     }
422 }
423

```

## A-5 - src/cronframe\_expr.rs

```

1  //! This type is used in cron objects to define the cron expression and
timeout for a method job.
2
3  /// It implements the From trait for strings
4  #[derive(Debug, Clone, Default)]
5  pub struct CronFrameExpr {
6      seconds: String,
7      minutes: String,
8      hour: String,
9      day_month: String,
10     month: String,
11     day_week: String,
12     year: String,
13     timeout: u64,

```

```

14 }
15
16 impl CronFrameExpr {
17     /// Creates a new CronFrameExpr instance where:
18     /// - s is seconds
19     /// - m is minutes
20     /// - h is hour
21     /// - dm is day_month
22     /// - mth is month
23     /// - dw is day_week
24     /// - y is year
25     /// - t is timeout
26     ///
27     ///
28     /// use cronframe::CronFrameExpr;
29     /// fn main(){
30     ///     let my_expr = CronFrameExpr::new("0", "5", "10-14", "*", "*",
31     "Sun", "*", 0);
32     /// }
33     pub fn new(s: &str, m: &str, h: &str, dm: &str, mth: &str, dw: &str,
34     y: &str, t: u64) -> Self {
35         CronFrameExpr {
36             seconds: s.to_string(),
37             minutes: m.to_string(),
38             hour: h.to_string(),
39             day_month: dm.to_string(),
40             month: mth.to_string(),
41             day_week: dw.to_string(),
42             year: y.to_string(),
43             timeout: t,
44         }
45     }
46     pub fn expr(&self) -> String {
47         format!(
48             "{} {} {} {} {} {} {}",
49             self.seconds,
50             self.minutes,
51             self.hour,
52             self.day_month,
53             self.month,
54             self.day_week,
55             self.year
56         )
57     }
58     pub fn timeout(&self) -> u64 {
59         self.timeout
60     }
61 }
62 }
63
64 impl From<&str> for CronFrameExpr {

```

```
65     fn from(item: &str) -> Self {
66         let items: Vec<_> = item.split(" ").collect();
67         CronFrameExpr::new(
68             items[0],
69             items[1],
70             items[2],
71             items[3],
72             items[4],
73             items[5],
74             items[6],
75             items[7].parse().unwrap(),
76         )
77     }
78 }
```

## A-6 - src/web\_server.rs

```
1 //! Custom setup of rocket.rs for the Cronframe web server
2
3 use crate::{
4     config::read_config, cronframe::CronFrame, utils, CronFilter,
5     CronJobType, JobBuilder,
6 };
7 use colored::Colorize;
8 use log::info;
9 use rocket::{
10     config::Shutdown,
11     figment::{
12         providers::{Env, Format, Toml},
13         Figment, Profile,
14     },
15     serde::Serialize,
16 };
17 use rocket_dyn_templates::{context, Template};
18 use std::{fs, path::Path, sync::Arc, time::Duration};
19
20 /// Called by the init function of the Cronframe type for setting up the
21 /// web server
22 ///
23 /// It provides 7 routes, five of which are API only.
24 /// Upon first start of the framework it will generate a templates
25 /// directory.
26 pub fn web_server(frame: Arc<CronFrame>) {
27     generate_template_dir();
28
29     let cronframe = frame.clone();
30
31     let tokio_runtime = rocket::tokio::runtime::Runtime::new().unwrap();
32
33     let base_config = match read_config() {
34         Some(config_data) => rocket::Config {
```

```

33         port: {
34             if let Some(webserver_data) = &config_data.webserver {
35                 webserver_data.port.unwrap_or_else(|| 8098)
36             } else {
37                 8098
38             }
39         },
40         address: {
41             if let Some(webserver_data) = config_data.webserver {
42                 webserver_data.ip.unwrap_or_else(|| "127.0.0.1".to_string())
43             } else {
44                 "127.0.0.1".to_string()
45             }
46             .parse()
47             .unwrap()
48         },
49         shutdown: Shutdown {
50             ctrlc: false,
51             ..Default::default()
52         },
53         cli_colors: false,
54         ..rocket::Config::release_default()
55     },
56     None => {
57         // default config
58         rocket::Config {
59             port: 8098,
60             address: std::net::Ipv4Addr::new(127, 0, 0, 1).into(),
61             shutdown: Shutdown {
62                 ctrlc: false,
63                 ..Default::default()
64             },
65             cli_colors: false,
66             ..rocket::Config::release_default()
67         }
68     }
69 };
70
71 let ip_address = base_config.address;
72 let port = base_config.port;
73 let home_dir = utils::home_dir();
74
75 if std::env::var("CRONFRAME_CLI").is_ok() {
76     std::env::set_var(
77         "ROCKET_CONFIG",
78         format!("{}{}/.cronframe/rocket.toml"),
79     );
80 }
81
82 let config = Figment::from(base_config)
83     .merge(Toml::file(Env::var_or("ROCKET_CONFIG",
84 "Rocket.toml")).nested())

```

```

84     .merge(Env::prefixed("ROCKET_").ignore(&["PROFILE"]).global())
85     .select(Profile::from_env_or(
86         "ROCKET_PROFILE",
87         rocket::Config::DEFAULT_PROFILE,
88     )));
89
90     let rocket = rocket::Rocket::custom(config)
91     .mount(
92         "/",
93         routes![
94             styles,
95             cronframe,
96             tingle,
97             tinglejs,
98             home,
99             job_info,
100            update_timeout,
101            update_schedule,
102            suspension_handle,
103            start_scheduler,
104            stop_scheduler,
105            add_cli_job,
106            shutdown,
107        ],
108    )
109    .attach(Template::fairing())
110    .manage(frame);
111
112    let (tx, _) = cronframe.web_server_channels.clone();
113
114    tokio_runtime.block_on(async move {
115        let rocket = rocket.ignite().await.expect("Ignition Error");
116
117        let _ = tx.send(rocket.shutdown());
118
119        match rocket.launch().await {
120            Ok(_) => println!("CronFrame running at http://{}:{}",
121                ip_address, port),
122            Err(err) => {
123                println!(
124                    "{} {}",
125                    "Error during CronFrame init:".red().bold(),
126                    err.pretty_print()
127                );
128                cronframe.clone().quit();
129            }
130        });
131    }
132
133 // necessary to have somewhat decent-looking pages
134 #[get("/styles")]
135 async fn styles() -> Result<rocket::fs::NamedFile, std::io::Error> {

```

```

136     if std::env::var("CRONFRAME_CLI").is_ok() {
137         let home_dir = utils::home_dir();
138         rocket::fs::NamedFile::open(format!("{}{}/cronframe/
139             templates/styles.css")).await
140     } else {
141         rocket::fs::NamedFile::open(format!("./templates/
142             styles.css")).await
143     }
144 // necessary to have somewhat decent-looking pages
145 #[get("/tingle")]
146 async fn tingle() -> Result<rocket::fs::NamedFile, std::io::Error> {
147     if std::env::var("CRONFRAME_CLI").is_ok() {
148         let home_dir = utils::home_dir();
149         rocket::fs::NamedFile::open(format!("{}{}/cronframe/
150             templates/tingle.css")).await
151     } else {
152         rocket::fs::NamedFile::open(format!("./templates/
153             tingle.css")).await
154     }
155 // necessary to have decent modals
156 #[get("/tinglejs")]
157 async fn tinglejs() -> Result<rocket::fs::NamedFile, std::io::Error> {
158     if std::env::var("CRONFRAME_CLI").is_ok() {
159         let home_dir = utils::home_dir();
160         rocket::fs::NamedFile::open(format!("{}{}/cronframe/
161             templates/tingle.js")).await
162     } else {
163         rocket::fs::NamedFile::open(format!("./templates/
164             tingle.js")).await
165     }
166 // necessary to have somewhat functioning pages
167 #[get("/cronframejs")]
168 async fn cronframe() -> Result<rocket::fs::NamedFile, std::io::Error> {
169     if std::env::var("CRONFRAME_CLI").is_ok() {
170         let home_dir = utils::home_dir();
171         rocket::fs::NamedFile::open(format!("{}{}/cronframe/
172             templates/cronframe.js")).await
173     } else {
174         rocket::fs::NamedFile::open(format!("./templates/
175             cronframe.js")).await
176     }
177 #[derive(Serialize)]
178 #[serde(crate = "rocket::serde")]
179 struct JobList {
180     name: String,

```

```

181     id: String,
182 }
183
184 // homepage returning a list of all jobs in the following categories:
185 // active, timed out, suspended
185 #[get("/")]
186 fn home(cronframe: &rocket::State<Arc<CronFrame>>) -> Template {
187     let running = *cronframe.running.lock().unwrap();
188
189     let mut active_jobs = vec![];
190     let mut timedout_jobs = vec![];
191     let mut suspended_jobs = vec![];
192
193     for job in cronframe
194         .cron_jobs
195         .lock()
196         .expect("cron jobs unwrap error in web server")
197         .iter()
198     {
199         let job_type = match job.job {
200             CronJobType::Global(_) => CronFilter::Global,
201             CronJobType::Function(_) => CronFilter::Function,
202             CronJobType::Method(_) => CronFilter::Method,
203             CronJobType::CLI => CronFilter::CLI,
204         };
205
206         if cronframe.filter.is_none() || cronframe.filter ==
207             Some(job_type) {
208             if job.status() == "Suspended" {
209                 suspended_jobs.push(JobList {
210                     name: job.name.clone(),
211                     id: job.id.to_string(),
212                 });
213             } else if job.status() == "Timed-Out" {
214                 timedout_jobs.push(JobList {
215                     name: job.name.clone(),
216                     id: job.id.to_string(),
217                 });
218             } else {
219                 active_jobs.push(JobList {
220                     name: job.name.clone(),
221                     id: job.id.to_string(),
222                 });
223             }
224         }
225
226         Template::render(
227             "index",
228             context! {running, active_jobs, timedout_jobs, suspended_jobs},
229         )
230     }
231

```

```

232 #[derive(Serialize, Default)]
233 #[serde(crate = "rocket::serde")]
234 struct JobInfo {
235     name: String,
236     id: String,
237     r#type: String,
238     run_id: String,
239     status: String,
240     timeout: String,
241     schedule: String,
242     upcoming_utc: String,
243     upcoming_local: String,
244     fail: bool,
245 }
246
247 // job page information where it is possible to change, schedule,
248 // timeout and toggle scheduling suspension
248 #[get("/job/<name>/<id>")]
249 fn job_info(name: &str, id: &str, cronframe:
250 &rocket::State<Arc<CronFrame>>) -> Template {
250     let running = *cronframe.running.lock().unwrap();
251     let mut job_info = JobInfo::default();
252
253     for job in cronframe.cron_jobs.lock().unwrap().iter() {
254         if job.name == name && job.id.to_string() == id {
255             job_info = JobInfo {
256                 name: job.name.clone(),
257                 id: job.id.to_string(),
258                 r#type: job.type_to_string(),
259                 run_id: job.run_id(),
260                 status: job.status(),
261                 timeout: job.timeout_to_string(),
262                 schedule: job.schedule(),
263                 upcoming_utc: job.upcoming_utc(),
264                 upcoming_local: job.upcoming_local(),
265                 fail: job.failed,
266             };
267             break;
268         }
269     }
270
271     Template::render("job", context! {running, job_info})
272 }
273
274 // API route to change the value of the timeout
275 #[get("/job/<name>/<id>/toutset/<value>")]
276 fn update_timeout(name: &str, id: &str, value: i64, cronframe:
277 &rocket::State<Arc<CronFrame>>) {
277     for job in cronframe.cron_jobs.lock().unwrap().iter_mut() {
278         if job.name == name && job.id.to_string() == id {
279             let job_id = format!("{} ID#{:?}", job.name, job.id);
280             job.start_time = None;
281             job.set_timeout(value);

```

```

282             info!("job @{job_id} - Timeout Update");
283         }
284     }
285 }
286
287 // API route to change the value of the cron expression and therefore
the schedule
288 #[get("/job/<name>/<id>/schedset/<expression>")]
289 fn update_schedule(
290     name: &str,
291     id: &str,
292     expression: &str,
293     cronframe: &rocket::State<Arc<CronFrame>>,
294 ) {
295     for job in cronframe.cron_jobs.lock().unwrap().iter_mut() {
296         if job.name == name && job.id.to_string() == id {
297             let job_id = format!("{} ID#{:?}", job.name, job.id);
298             if job.set_schedule(expression) {
299                 info!("job @{job_id} - Schedule Update");
300             } else {
301                 info!("job @{job_id} - Schedule Update Fail - Cron
Expression Parse Error");
302             }
303         }
304     }
305 }
306
307 // API route to toggle the scheduling suspension for a job
308 #[get("/job/<name>/<id>/suspension_toggle")]
309 fn suspension_handle(name: &str, id: &str, cronframe:
&rocket::State<Arc<CronFrame>>) {
310     for job in cronframe.cron_jobs.lock().unwrap().iter_mut() {
311         if job.name == name && job.id.to_string() == id {
312             let job_id = format!("{} ID#{:?}", job.name, job.id);
313             if !job.suspended {
314                 job.suspended = true;
315                 info!("job @{job_id} - Scheduling Suspended");
316             } else {
317                 job.suspended = false;
318                 info!("job @{job_id} - Scheduling Reprised");
319             }
320         }
321     }
322 }
323
324 // API route to start the scheduler
325 #[get("/start_scheduler")]
326 fn start_scheduler(cronframe: &rocket::State<Arc<CronFrame>>) {
327     cronframe.start_scheduler();
328 }
329
330 // API route to stop the scheduler
331 #[get("/stop_scheduler")]

```

```

332 fn stop_scheduler(cronframe: &rocket::State<Arc<CronFrame>>) {
333     cronframe.stop_scheduler();
334 }
335
336 // API route to toggle the scheduling suspension for a job
337 #[get("/add_cli_job/<expr>/<timeout>/<job>")]
338 fn add_cli_job(expr: &str, timeout: &str, job: &str, cronframe:
339     &rocket::State<Arc<CronFrame>>) {
340     let new_job = JobBuilder::cli_job(job, expr, timeout).build();
341     cronframe.add_job(new_job);
342 }
343
344 // API route to stop the scheduler
345 #[get("/shutdown")]
346 fn shutdown(cronframe: &rocket::State<Arc<CronFrame>>) {
347     cronframe.quit();
348 }
349
350 /// It generates a templates directory either inside the current
351 /// directory or in the .cronframe directory.
352 /// The tempaltes directory will contain the following files:
353 /// - base.html.tera
354 /// - index.htm.tera
355 /// - job.html.tera
356 /// - tingle.js
357 /// - cronframe.js
358 /// - styles.css
359 pub fn generate_template_dir() {
360     if std::env::var("CRONFRAME_CLI").is_ok() {
361         let home_dir = utils::home_dir();
362
363         if !Path::new(&format!("{}/.cronframe/
364             templates")).exists() {
365             fs::create_dir(format!("{}/.cronframe/templates"))
366                 .expect("could not create templates directory");
367
368             let _ = fs::write(
369                 Path::new(&format!("{}/.cronframe/templates/
370                     base.html.tera")),
371                     BASE_TEMPLATE,
372                 );
373             let _ = fs::write(
374                 Path::new(&format!("{}/.cronframe/templates/
375                     index.html.tera")),
376                     INDEX_TEMPLATE,
377                 );
378             let _ = fs::write(
379                 Path::new(&format!("{}/.cronframe/templates/
380                     job.html.tera")),
381                     JOB_TEMPLATE,
382                 );
383             let _ = fs::write(

```

```

379             Path::new(&format!("{}{}/.cronframe/templates/",
380 tingle.js")),
381             TINGLE_JS,
382         );
383         let _ = fs::write(
384             Path::new(&format!("{}{}/.cronframe/templates/"),
385 cronframe.js)),
386             CRONFRAME_JS,
387         );
388         let _ = fs::write(
389             Path::new(&format!("{}{}/.cronframe/templates/"),
390 tingle.css)),
391             TINGLE_STYLES,
392         );
393         let _ = fs::write(
394             Path::new(&format!("{}{}/.cronframe/templates/"),
395 styles.css)),
396             STYLES,
397         );
398     }
399     } else {
400         if !Path::new(&format!("./templates")).exists() {
401             fs::create_dir(format!("templates")).expect("could not
402 create templates directory");
403
404             let _ = fs::write(
405                 Path::new(&format!("./templates/base.html.tera")),
406                 BASE_TEMPLATE,
407             );
408             let _ = fs::write(
409                 Path::new(&format!("./templates/index.html.tera")),
410                 INDEX_TEMPLATE,
411             );
412             let _ = fs::write(
413                 Path::new(&format!("./templates/job.html.tera")),
414                 JOB_TEMPLATE,
415             );
416             let _ = fs::write(Path::new(&format!("./templates/
417 tingle.js")), TINGLE_JS);
418             let _ = fs::write(
419                 Path::new(&format!("./templates/cronframe.js")),
420                 CRONFRAME_JS,
421             );
422             let _ = fs::write(Path::new(&format!("./templates/
423 tingle.css")), TINGLE_STYLES);
424             let _ = fs::write(Path::new(&format!("./templates/
425 styles.css")), STYLES);
426         }
427     }
428     std::thread::sleep(Duration::from_secs(10));
429 }
430
431 // templates folder data: templates/base.tera.html

```

```

424 const BASE_TEMPLATE: &str = {
425     r#"<!DOCTYPE html>
426 <html class="light-mode">
427
428 <head>
429     <meta charset="utf-8" />
430     <title>CronFrame</title>
431     <link href='https://fonts.googleapis.com/css?family=Lato' rel='stylesheet'>
432     <link rel="stylesheet" href="/styles">
433     <link rel="stylesheet" href="/tingle">
434
435 </head>
436
437 <body>
438
439     <div id="wrapper">
440         <div>
441             <div id="barContainer">
442                 <div id="progressBar">
443                     <div id="barStatus"></div>
444                 </div>
445             </div>
446             <div id="container">
447                 <header>
448                     <div id="logo">
449                         <a href="/"><span class="cron">Cron</span><span style="color:#FF3D00">Frame</span></a>
450                     </div>
451
452                     <label class="switch" title="Toggle Color Theme">
453                         <input type="checkbox" onchange="toggleMode()" id="slider">
454                         <span class="slider"></span>
455                     </label>
456                     {% if running %}
457                     <div id="scheduler_status_running">
458                         <span style="padding:0px 5px">①</span> Scheduler
Running
459                         </div>
460                         <div id="scheduler_stop" onclick="stopScheduler()">
461                             <svg height="24" viewBox="0 0 24 20" width="24" xmlns="http://www.w3.org/2000/svg">
462                                 <path fill="currentColor" d="M12 21c4.411 0 8-3.589 8-8
0-3.35-2.072-6.221-5-7.411v2.223A6 6 0 0 1 18 13c0 3.309-2.691 6-6
6s-6-2.691-6-6a5.999 5.999 0 0 1 3-5.188V5.589C6.072 6.779 4 9.65 4 13c0
4.411 3.589 8 8 8z" />
464                                 <path fill="currentColor" d="M11
2h2v10h-2z" />
465                             </svg>
466                         </div>
467                     {% else %}>

```

```

468             <div id="scheduler_status_not_running">
469                 <span style="padding:0px 5px">①</span> Scheduler
470             Not Running
471         </div>
472         <div id="scheduler_start"
473             onclick="startScheduler()">
474             <svg height="24" viewBox="0 0 24 20" width="24"
475                 xmlns="http://www.w3.org/2000/svg">
476                 <path fill="currentColor"
477                     d="M12 21c4.411 0 8-3.589 8-8
478 0-3.35-2.072-6.221-5-7.411v2.223A6 6 0 0 1 18 13c0 3.309-2.691 6-6
479 6s-6-2.691-6-6a5.999 5.999 0 0 1 3-5.188V5.589C6.072 6.779 4 9.65 4 13c0
480 4.411 3.589 8 8 8z" />
481                 <path fill="currentColor" d="M11
482 2h2v10h-2z" />
483             </svg>
484         </div>
485     <% endif %>
486 </header>
487
488     <div id="content">
489         <% block content %>
490         <% endblock content %>
491     </div>
492
493     <footer>
494         <label class="reload">
495             <input type="checkbox" onchange="toggleReload()"
496                 id="autoreload">
497             <span class="check"></span>
498         </label>
499         5s Reload
500         <a href="https://github.com/antcim/cronframe"
501             target="_blank" class="repo" title="Developed by Antonio Cimino">
502             <svg xmlns="http://www.w3.org/2000/svg"
503                 width="24" height="24" viewBox="0 0 24 24">
504                 <path fill="currentColor"
505                     d="M12 0c-6.626 0-12 5.373-12 12 0 5.302
506 3.438 9.8 8.207
507 11.387.599.111.793-.261.793-.577v-2.234c-3.338.726-4.033-1.416-4.033-1.416-.546-1.387-1.333-1.7
508 1.205.084 1.839 1.237 1.839 1.237 1.07 1.834 2.807 1.304
509 3.492.997.107-.775.418-1.305.762-1.604-2.665-.305-5.467-1.334-5.467-5.931
510 0-1.311.469-2.381 1.236-3.221-.124-.303-.535-1.524.117-3.176 0 0 1.008-.322
511 3.301 1.23.957-.266 1.983-.399 3.003-.404 1.02.005 2.047.138 3.006.404
512 2.291-1.552 3.297-1.23 3.297-1.23.653 1.653.242 2.874.118 3.176.77.84 1.235
513 1.911 1.235 3.221 0 4.609-2.807 5.624-5.479 5.921.43.372.823 1.102.823
514 2.222v3.293c0 .319.192.694.801.576 4.765-1.589 8.199-6.086 8.199-11.386
515 0-6.627-5.373-12-12-12z" />
516             </svg>
517             <span>antcim/cronframe</span>
518         </a>
519     </footer>
520     <script src="/tinglejs"></script>

```

```

501             <script src="/cronframejs"></script>
502         </div>
503     </div>
504 </div>
505 </body>
506
507 </html>"#
508 };
509
510 // templates folder data: templates/index.tera.html
511 const INDEX_TEMPLATE: &str = {
512     r#"`% extends "base" %`"
513
514 `% block content %`"
515 <table id="job_list">
516     <tr>
517         <th>
518             Active Jobs
519             <div class="refresh" onclick="reloadPage()">&lt;></div>
520         </th>
521     </tr>
522     {% if active_jobs %}"
523     {% for cron_job in active_jobs %}"
524     {% set activelink = "/job/" ~ cron_job.name ~ "/" ~ cron_job.id %}"
525     <tr>
526         <td><a href="{{activelink}}">{{cron_job.name}}</a></td>
527         <td>{{cron_job.id}}</td>
528     </tr>
529     {% endfor %}"
530     {% else %}"
531     <tr>
532         <td>No active job found</td>
533     </tr>
534     {% endif %}"
535
536 </table>
537
538 <table id="job_list">
539     <tr>
540         <th>
541             Timed-Out Jobs <div class="refresh"
542             onclick="reloadPage()">&lt;></div>
543         </th>
544     </tr>
545     {% if timedout_jobs %}"
546     {% for cron_job in timedout_jobs %}"
547     {% set timedoutlink = "/job/" ~ cron_job.name ~ "/" ~ cron_job.id %}"
548     <tr>
549         <td><a href="{{timedoutlink}}">{{cron_job.name}}</a></td>
550         <td>{{cron_job.id}}</td>
551     </tr>
552     {% endfor %}"
553     {% else %}"

```

```

553     <tr>
554         <td>No timed-out job found</td>
555     </tr>
556     {% endif %}
557 </table>
558
559 <table id="job_list">
560     <tr>
561         <th>
562             Suspended Jobs <div class="refresh"
563             onclick="reloadPage()">&lt;/div>
564         </th>
565     </tr>
566     {% if suspended_jobs %}
567     {% for cron_job in suspended_jobs %}
568     {% set suspendedlink = "/job/" ~ cron_job.name ~ "/" ~ cron_job.id
569     %}
570     <tr>
571         <td><a href="{{suspendedlink}}">{{cron_job.name}}</a></td>
572         <td>{{cron_job.id}}</td>
573     </tr>
574     {% endfor %}
575     {% else %}
576     <tr>
577         <td>No suspended job found</td>
578     </tr>
579     {% endif %}
580 </table>
581 {% endblock content %}#
582 // templates folder data: templates/job.tera.html
583 const JOB_TEMPLATE: &str = {
584     r#"{% extends "base" %}

585     {% block content %}

586     {% if job_info.name != ""%}
587     <table id="job_info">
588         <tr>
589             <th colspan="2">
590                 Job Info @{{job_info.name}} <div class="refresh"
591                 onclick="reloadPage()">&lt;/div>
592             </th>
593         </tr>
594         <tr>
595             <td>Name</td>
596             <td colspan="2">{{job_info.name}}</td>
597         </tr>
598         <tr>
599             <td>Id</td>
600             <td colspan="2">
601                 <div class="id_cont">

```

```

603             <span id="job_id">{{job_info.id}}</span>
604             <span class="clipboard"
605               onclick="copyToClipBoard('job_id')" title="copy to clipboard">
606                 <?xml version="1.0" ?>
607                 <svg width="16px" height="16px" viewBox="0 0 512
512" xmlns="http://www.w3.org/2000/svg">
608                   <path
609                     d="M464 0c26.51 0 48 21.49 48 48v288c0
26.51-21.49 48-48 48H176c-26.51 0-48-21.49-48-48V48c0-26.51 21.49-48
48-48h288M176 416c-44.112 0-80-35.888-80-80V128H48c-26.51 0-48 21.49-48
48v288c0 26.51 21.49 48 48h288c26.51 0 48-21.49 48-48v-48H176z" />
610             </svg>
611           </span>
612         </td>
613       </tr>
614       <tr>
615         <td>Type</td>
616         <td colspan="2">{{job_info.type}} Job</td>
617       </tr>
618     {% if job_info.run_id != "None" %}
619     <tr>
620       <td>Run Id</td>
621       <td colspan="2">
622         <div class="id_cont">
623           <span id="run_id">{{job_info.run_id}}</span>
624           <span class="clipboard"
625             onclick="copyToClipBoard('run_id')" title="copy to clipboard">
626               <?xml version="1.0" ?>
627               <svg width="16px" height="16px" viewBox="0 0 512
512" xmlns="http://www.w3.org/2000/svg">
628                 <path
629                   d="M464 0c26.51 0 48 21.49 48 48v288c0
26.51-21.49 48-48 48H176c-26.51 0-48-21.49-48-48V48c0-26.51 21.49-48
48-48h288M176 416c-44.112 0-80-35.888-80-80V128H48c-26.51 0-48 21.49-48
48v288c0 26.51 21.49 48 48h288c26.51 0 48-21.49 48-48v-48H176z" />
630             </svg>
631           </span>
632         </div>
633       </td>
634     </tr>
635   {% endif %}
636   <tr>
637     <td>Status</td>
638     <td colspan="">
639       {% if job_info.status == "Timed-Out" or job_info.status ==
640       "Suspended" %}
641         <div class="line_status_gray">{{job_info.status}}</div>
642       {% elif job_info.status == "Running" %}
643         <div class="line_status_green">{{job_info.status}}</div>
644       {% else %}
645         <div class="line_status_yellow">{{job_info.status}}</div>
646       {% endif %}

```

```

645         </td>
646         <td>
647             {% if job_info.status != "Suspended" %}
648                 <button onclick="suspensionHandle()">Suspend Scheduling</
button>
649             {% else %}
650                 <button onclick="suspensionHandle()">Reprise Scheduling</
button>
651             {% endif %}
652         </td>
653     </tr>
654     <tr>
655         <td>Fail History</td>
656         <td colspan="2">
657             {% if job_info.fail %}
658                 <div class="line_status_orange">Failed instances recorded</
div>
659             {% else %}
660                 No failed instances recorded
661             {% endif %}
662         </td>
663     </tr>
664     <tr>
665         <td>Schedule</td>
666         <td>
667             {{job_info.schedule}}
668         </td>
669         <td>
670             <input oninput="setSchedule(this.value)" type="text"
placeholder="enter cron expression">
671                 <button onclick="updateSchedule()">Update</button>
672             </td>
673     </tr>
674     <tr>
675         <td>Timeout</td>
676         <td>
677             {% if job_info.timeout != "None" %}
678                 {{job_info.timeout | linebreaksbr | split(pat=<br>) | nth(n=1)}}
679                 ~ {{job_info.timeout | linebreaksbr | split(pat=<br>) | nth(n=0)}}
680             {% else %}
681                 {{job_info.timeout}}
682             {% endif %}
683         </td>
684         <td>
685             <input oninput="setTimeout(this.value)" type="number"
min="0" placeholder="enter timeout in ms">
686                 <button onclick="updateTimeout()">Update</button>
687             </td>
688     </tr>
689     <tr>
690         <td>Upcoming</td>

```

```

691         <td colspan="2">
692             {% if job_info.upcoming_utc == "None due to timeout." %}
693                 <p>{{job_info.upcoming_utc}}</p>
694             {% else %}
695                 <p>{{job_info.upcoming_utc}}</p>
696                 {% if job_info.status != "Timed-Out" and
697                     job_info.status != "Suspended" %}
698                     <p>{{job_info.upcoming_local}} (Local)</p>
699                 {% endif %}
700             {% endif %}
701         </td>
702     </tr>
703 </table>
704 <script>
705
706 </script>
707
708 {% else %}
709 <div id="job_info">
710     <div class="job_info_item">
711         Job not found
712     </div>
713 </div>
714 {% endif %}
715 {% endblock content %}#
716 };
717
718 // templates folder data: templates/styles.css
719 const STYLES: &str = {
720     r#":root {
721         --dark-orange: #ff3d00;
722         --light-orange: #ffa702;
723         --dark-green: #0fa702;
724         --green: green;
725         --red: red;
726     }
727
728     .light-mode {
729         --body-bg: #f6f6f6;
730         --container-bg: #ffffff;
731         --content-bg: #f1f1f1;
732         --font-color: #494949;
733         --gray-status-color: rgba(0, 0, 0, .3);
734         --checkbox: rgba(0, 0, 0, .1);
735         --scheduler-status-stop: rgba(0, 0, 0, .1);
736         --scheduler-start-stop-text: rgba(0,0,0,.3);
737         --scheduler-status-running: rgba(51, 255, 0, .3);
738         --scheduler-status-running-text: rgba(0, 0, 0, .4);
739         --scheduler-status-not-running: rgba(237, 233, 157, .7);
740         --scheduler-status-not-running-text: rgba(0, 0, 0, .4);
741     }
742

```

```
743 .dark-mode {
744   --body-bg: #161616;
745   --container-bg: #2c2c2c;
746   --content-bg: #212121;
747   --font-color: #9a9a9a;
748   --gray-status-color: rgba(255, 255, 255, .3);
749   --checkbox: rgba(0, 0, 0, .3);
750   --scheduler-status-stop: rgba(0, 0, 0, .3);
751   --scheduler-start-stop-text: rgba(255, 255, 255, .3);
752   --scheduler-status-running: rgba(51, 255, 0, .2);
753   --scheduler-status-running-text: rgba(255, 255, 255, .4);
754   --scheduler-status-not-running: rgba(237, 233, 157, .3);
755   --scheduler-status-not-running-text: rgba(255, 255, 255, .4);
756 }
757
758 body {
759   background: var(--body-bg);
760   color: var(--font-color);
761   font-family: "Lato"!important;
762   margin: 0;
763 }
764
765 a {
766   text-decoration: none;
767 }
768
769 a:link {
770   color: var(--dark-orange);
771 }
772
773 a:visited {
774   color: var(--light-orange);
775 }
776
777 a:hover {
778   color: var(--green);
779 }
780
781 a:active {
782   color: var(--red);
783 }
784
785 header {
786   display: flex;
787   padding: 15px;
788   align-items: center;
789   justify-content: center;
790 }
791
792 footer {
793   padding: 15px;
794 }
795
```

```
796 #logo {
797   flex: 2;
798   font-weight: bold;
799   font-size: 30pt;
800   margin-right: 50px;
801 }
802
803 .cron {
804   color: var(--font-color);
805 }
806
807 .refresh {
808   display: inline;
809   margin-left: 6px;
810 }
811
812 .refresh:hover {
813   cursor: pointer;
814   color: var(--light-orange);
815 }
816
817 .refresh:active {
818   cursor: pointer;
819   color: var(--dark-orange);
820 }
821
822 #wrapper {
823   display: flex;
824   flex-direction: column;
825   justify-content: center;
826   align-items: center;
827   height: 100vh;
828   margin: 0;
829   padding: 0;
830 }
831
832 #container {
833   display: flex;
834   flex-direction: column;
835   gap: 5px;
836   background: var(--container-bg);
837   padding: 10px;
838   padding-top: 5px;
839   padding-bottom: 5px;
840   border-radius: 6px;
841   box-shadow: 0px 0px 3px 0px rgba(0, 0, 0, .1);
842   max-width: 1200px;
843   min-width: 500px;
844 }
845
846 #scheduler_start, #scheduler_stop{
847   color: var(--scheduler-start-stop-text);
848   margin-left: 10px;
```

```
849   font-size: 20pt;
850   font-weight: bold;
851   cursor: pointer;
852   background: var(--scheduler-status-stop);
853   border-radius: 6px;
854   padding: 5px 8px;
855 }
856
857 #scheduler_start:hover{
858   color: var(--scheduler-status-running-text);
859   background: var(--scheduler-status-running);
860 }
861
862 #scheduler_start:active{
863   color: white;
864   background: var(--green);
865 }
866
867 #scheduler_stop:hover{
868   color: var(--scheduler-status-running-text);
869   background: var(--light-orange);
870 }
871
872 #scheduler_stop:active{
873   color: white;
874   background: var(--dark-orange);
875 }
876
877 #scheduler_status_running {
878   font-weight: bold;
879   background: var(--scheduler-status-running);
880   padding: 10px;
881   border-radius: 6px;
882   margin-left: 10px;
883   color: var(--scheduler-status-running-text);
884 }
885
886 #scheduler_status_not_running {
887   font-weight: bold;
888   background: var(--scheduler-status-not-running);
889   padding: 10px;
890   border-radius: 6px;
891   margin-left: 10px;
892   color: var(--scheduler-status-not-running-text);
893 }
894
895 #content {
896   background: var(--content-bg);
897   padding: 10px;
898   border-radius: 6px;
899   max-height: 70vh;
900   overflow: auto;
901 }
```

```
902
903 input[type=text],
904 input[type=number] {
905   padding: 10px;
906   display: inline-block;
907   border: 1px solid #ccc;
908   border-radius: 4px;
909   box-sizing: border-box;
910 }
911
912 input[type=text]:focus,
913 input[type=number]:focus {
914   border-color: rgba(229, 103, 23, 0.7);
915   box-shadow: 0 1px 1px rgba(229, 103, 23, 0.075) inset, 0 0 4px
rgba(229, 103, 23, 0.6);
916   outline: 0 none;
917 }
918
919 button {
920   background-color: rgba(0, 0, 0, .5);
921   font-weight: bold;
922   color: white;
923   padding: 12px;
924   border: none;
925   border-radius: 6px;
926   cursor: pointer;
927   box-shadow: 0 0 2px 1px rgba(0, 0, 0, .1) inset;
928 }
929
930 button:hover {
931   background-color: #4caf50;
932   color: white;
933   border: none;
934   cursor: pointer;
935   box-shadow: 0 0 2px 1px rgba(0, 0, 0, .1) inset;
936 }
937
938 button:active {
939   background-color: var(--dark-orange);
940   color: white;
941   border: none;
942   cursor: pointer;
943   box-shadow: 0 0 2px 1px rgba(0, 0, 0, .2) inset;
944 }
945
946 .line_status_green {
947   font-weight: bold;
948   display: inline-block;
949   background: rgba(51, 255, 0, .5);
950   padding: 10px;
951   border-radius: 6px;
952   border: 1px solid rgba(0, 0, 0, .1);
953   color: rgba(0, 0, 0, .4);
```

```
954 }
955 .line_status_yellow {
956   font-weight: bold;
957   display: inline-block;
958   background: rgba(255, 236, 102, 1);
959   padding: 10px;
960   border-radius: 6px;
961   border: 1px solid rgba(0, 0, 0, .1);
962   color: rgba(0, 0, 0, .4);
963 }
964
965 .line_status_orange {
966   font-weight: bold;
967   display: inline-block;
968   background: rgba(255, 61, 0, .7);
969   padding: 10px;
970   border-radius: 6px;
971   border: 1px solid rgba(0, 0, 0, .1);
972   color: rgba(0, 0, 0, .4);
973 }
974
975 .line_status_gray {
976   font-weight: bold;
977   display: inline-block;
978   background: var(--gray-status-color);
979   padding: 10px;
980   border-radius: 6px;
981   border: 1px solid rgba(0, 0, 0, .1);
982   color: rgba(0, 0, 0, .4);
983 }
984
985 .clipboard {
986   margin: 2px;
987   opacity: 0.5;
988   filter: invert(50%);
989 }
990
991 .clipboard:hover {
992   opacity: 0.8;
993   cursor: pointer;
994 }
995
996 .id_cont {
997   display: inline;
998   padding: 8px;
999   border-radius: 6px;
1000   background: var(--checkbox);
1001 }
1002
1003
1004 table {
1005   border-collapse: collapse;
1006 }
```

```
1007 #job_info td {
1008   padding: 15px;
1009   border-bottom: 1px solid rgba(0, 0, 0, .05);
1010 }
1011 }
1012
1013 #job_info th,
1014 #job_info td:nth-child(1) {
1015   font-weight: bold;
1016   font-size: 16pt;
1017   border-right: 1px solid rgba(0, 0, 0, .05);
1018 }
1019
1020 #job_info th {
1021   text-align: left;
1022   font-size: 20pt;
1023   padding: 15px;
1024   border: 0;
1025 }
1026
1027 #job_info tr:last-child td {
1028   border: 0px;
1029   border-right: 1px solid rgba(0, 0, 0, .05);
1030 }
1031
1032 #job_info tr:last-child td:last-child {
1033   border: 0px;
1034 }
1035
1036 #job_list td {
1037   padding: 15px;
1038   border-bottom: 1px solid rgba(0, 0, 0, .05);
1039 }
1040
1041 #job_list th,
1042 #job_list td:nth-child(1) {
1043   font-weight: bold;
1044 }
1045
1046 #job_list th {
1047   text-align: left;
1048   font-size: 20pt;
1049   padding: 15px;
1050   border: 0;
1051 }
1052
1053 #job_list tr:last-child td {
1054   border: 0px;
1055 }
1056
1057 #job_list tr:last-child td:last-child {
1058   border: 0px;
1059 }
```

```
1060
1061 .clipboard_toast {
1062   background: rgba(255, 61, 0, .8);
1063   color: rgba(0, 0, 0, .4);
1064   border-radius: 6px;
1065   top: 0;
1066   right: 0;
1067   margin-right: 15px;
1068   margin-top: 15px;
1069   position: fixed;
1070   display: flex;
1071   flex-direction: row;
1072   align-items: center;
1073   padding: 15px;
1074   gap: 10px;
1075 }
1076
1077 .close_toast {
1078   font-weight: bold;
1079   cursor: pointer;
1080   margin-top: -2px;
1081 }
1082
1083 .close_toast:hover {
1084   color: white;
1085 }
1086
1087 .switch {
1088   position: relative;
1089   display: inline-block;
1090   width: 45px;
1091   height: 22px;
1092 }
1093
1094 .switch input {
1095   opacity: 0;
1096   width: 0;
1097   height: 0;
1098 }
1099
1100 .slider {
1101   position: absolute;
1102   cursor: pointer;
1103   top: 0;
1104   left: 0;
1105   right: 0;
1106   bottom: 0;
1107   background: rgba(237, 233, 157, .1);
1108   border-radius: 6px;
1109 }
1110
1111 .slider:before {
1112   position: absolute;
```

```
1113 content: "";
1114 height: 22px;
1115 width: 22px;
1116 margin: auto 0;
1117 background: silver;
1118 box-shadow: 0 0 3px 2px rgba(0, 0, 0, .1) inset;
1119 border-radius: 6px;
1120 }
1121
1122 .switch input:checked + .slider {
1123   background: rgba(0, 100, 150, .1);
1124 }
1125
1126 .switch input:checked + .slider:before {
1127   content: "";
1128   transform: translateX(24px);
1129   background: var(--light-orange);
1130   box-shadow: 0 0 3px 2px rgba(255, 0, 0, .1) inset;
1131 }
1132
1133 #barContainer {
1134   width: 100%;
1135   padding: 0;
1136   margin: 0;
1137 }
1138
1139 #progressBar {
1140   height: 5px;
1141   background-color: #ddd;
1142 }
1143
1144 #barStatus {
1145   width: 0%;
1146   height: 100%;
1147   background-color: var(--dark-orange);
1148 }
1149
1150 .reload {
1151   position: relative;
1152   display: inline-block;
1153   width: 20px;
1154   height: 20px;
1155 }
1156
1157 .reload input {
1158   opacity: 0;
1159   width: 0;
1160   height: 0;
1161 }
1162
1163 .check {
1164   position: absolute;
1165   cursor: pointer;
```

```
1166   top: 0;
1167   left: 0;
1168   right: 0;
1169   bottom: 0;
1170   background: var(--checkbox);
1171   border-radius: 4px;
1172 }
1173 .check:before {
1174   position: absolute;
1175   content: "";
1176   height: 12px;
1177   width: 12px;
1178   margin: auto 0;
1179   border-radius: 2px;
1180   top: 4px;
1181   left: 4px;
1182 }
1183 }
1184 .reload input:checked + .check:before {
1185   background: var(--light-orange);
1186   box-shadow: 0 0 3px 2px rgba(255, 0, 0, .1) inset;
1187 }
1188 }
1189 ::-webkit-scrollbar {
1190   width: 10px;
1191 }
1192 }
1193 ::-webkit-scrollbar-track {
1194   background: transparent;
1195   border-radius: 25px;
1196 }
1197 }
1198 ::-webkit-scrollbar-thumb {
1199   background: #888;
1200   border-radius: 25px;
1201 }
1202 }
1203 ::-webkit-scrollbar-thumb:hover {
1204   background: #555;
1205 }
1206 }
1207 }
1208 .repo{
1209   display: flex;
1210   gap: 6px;
1211   float: right;
1212   color: var(--font-color)!important;
1213 }
1214 }
1215 .repo:hover{
1216   color: var(--dark-orange)!important;
1217 }
1218 }
```

```

1219 .repo:active{
1220   color: var(--light-orange)!important;
1221 }"#;
1222 };
1223
1224 // templates folder data: templates/cronframe.js
1225 const CRONFRAME_JS: &str = {
1226     r#"// base template scripts
1227
1228 let stopModal = new tingle.modal({
1229     footer: true,
1230     stickyFooter: false,
1231     closeMethods: ['overlay', 'button', 'escape'],
1232     closeLabel: "Close",
1233     cssClass: ['custom-class-1', 'custom-class-2'],
1234     onOpen: function() {
1235         console.log('modal open');
1236     },
1237     onClose: function() {
1238         console.log('modal closed');
1239     },
1240     beforeClose: function() {
1241         return true; // close the modal
1242         return false; // nothing happens
1243     }
1244 });
1245
1246 stopModal.setContent('<h1>Do you want to stop the Scheduler?</h1>');
1247
1248 stopModal.addFooterBtn('No', 'tingle-btn tingle-btn--pull-right tingle-
btn', () => {
1249     stopModal.close();
1250 });
1251
1252 stopModal.addFooterBtn('Yes', 'tingle-btn tingle-btn--pull-right
tingle-btn--danger', () => {
1253     const url = window.location.href + "/stop_scheduler";
1254     console.log("request to: " + url);
1255     const xhr = new XMLHttpRequest();
1256     xhr.open("GET", url);
1257     xhr.send();
1258     xhr.responseType = "json";
1259     xhr.onload = () => {
1260         if (xhr.readyState == 4 && xhr.status == 200) {
1261             console.log(xhr.response);
1262             location.reload();
1263         } else {
1264             console.log(`Error: ${xhr.status}`);
1265         }
1266     };
1267 });
1268
1269 let startModal = new tingle.modal({

```

```

1270     footer: true,
1271     stickyFooter: false,
1272     closeMethods: ['overlay', 'button', 'escape'],
1273     closeLabel: "Close",
1274     cssClass: ['custom-class-1', 'custom-class-2'],
1275     onOpen: function() {
1276         console.log('modal open');
1277     },
1278     onClose: function() {
1279         console.log('modal closed');
1280     },
1281     beforeClose: function() {
1282         // here's goes some logic
1283         // e.g. save content before closing the modal
1284         return true; // close the modal
1285         return false; // nothing happens
1286     }
1287 });
1288
1289 startModal.setContent('<h1>Do you want to start the Scheduler?</h1>');
1290
1291 startModal.addFooterBtn('No', 'tingle-btn tingle-btn--pull-right
tingle-btn', () => {
1292     startModal.close();
1293 });
1294
1295 startModal.addFooterBtn('Yes', 'tingle-btn tingle-btn--pull-right
tingle-btn--danger', () => {
1296     const url = window.location.href + "/start_scheduler";
1297     console.log("request to: " + url);
1298     const xhr = new XMLHttpRequest();
1299     xhr.open("GET", url);
1300     xhr.send();
1301     xhr.responseType = "json";
1302     xhr.onload = () => {
1303         if (xhr.readyState == 4 && xhr.status == 200) {
1304             console.log(xhr.response);
1305             location.reload();
1306         } else {
1307             console.log(`Error: ${xhr.status}`);
1308         }
1309     };
1310 });
1311
1312 let barWidth = 0;
1313
1314 document.getElementById("barContainer").style.width =
document.getElementById("container").style.width;
1315
1316 const advanceBar = () => {
1317     if (barWidth < 100) {
1318         barWidth = barWidth + 3.125;
1319         document.getElementById("barStatus").style.width = barWidth +

```

```

'%';
1320     }
1321 };
1322
1323 const reloadPage = () => {
1324     location.reload();
1325 };
1326
1327 const setTheme = (value) => {
1328     localStorage.setItem('mode', value);
1329     document.documentElement.className = value;
1330 };
1331
1332 const setAutoreload = (value) => {
1333     localStorage.setItem('autoreload', value);
1334     document.documentElement.className = value;
1335     reloadPage();
1336 };
1337
1338 const toggleMode = () => {
1339     if (localStorage.getItem('mode') === 'dark-mode') {
1340         setTheme('light-mode');
1341     } else {
1342         setTheme('dark-mode');
1343     }
1344 };
1345
1346 const toggleReload = () => {
1347     if (localStorage.getItem('autoreload') === 'yes') {
1348         setAutoreload('no');
1349     } else {
1350         setAutoreload('yes');
1351     }
1352 };
1353
1354 const init = () => {
1355     setupTheme();
1356     setupBar();
1357 };
1358
1359 const setupTheme = () => {
1360     if (localStorage.getItem('mode') === 'dark-mode') {
1361         setTheme('dark-mode');
1362         document.getElementById('slider').checked = false;
1363     } else {
1364         setTheme('light-mode');
1365         document.getElementById('slider').checked = true;
1366     }
1367 };
1368
1369 const setupBar = () => {
1370     if (localStorage.getItem('autoreload') === 'yes') {
1371         setInterval(reloadPage, 5000);

```

```

1372     setInterval(advanceBar, 125);
1373     document.getElementById('autoreload').checked = true;
1374 } else {
1375     document.getElementById("barStatus").style.width = '100%';
1376     document.getElementById('autoreload').checked = false;
1377 }
1378 };
1379
1380 init();
1381
1382 const startScheduler = () => {
1383     startModal.open();
1384 }
1385
1386 const stopScheduler = () => {
1387     stopModal.open();
1388 }
1389
1390 // job page scripts
1391
1392 let timeout = 0;
1393 let schedule = "* * * * *";
1394
1395 const setTimeout = (value) => {
1396     console.log(value);
1397     timeout = value
1398 };
1399
1400 const updateTimeout = () => {
1401     console.log("request to: " + window.location.href + "/toutset/" +
+ timeout);
1402     const xhr = new XMLHttpRequest();
1403     xhr.open("GET", window.location.href + "/toutset/" + timeout);
1404     xhr.send();
1405     xhr.responseType = "json";
1406     xhr.onload = () => {
1407         if (xhr.readyState == 4 && xhr.status == 200) {
1408             console.log(xhr.response);
1409             location.reload();
1410         } else {
1411             console.log(`Error: ${xhr.status}`);
1412         }
1413     };
1414 }
1415
1416 const setSchedule = (value) => {
1417     console.log(value);
1418     schedule = value
1419 };
1420
1421 const updateSchedule = () => {
1422     console.log("request to: " + window.location.href + "/
schedset/" + schedule);

```

```

1423     const xhr = new XMLHttpRequest();
1424     xhr.open("GET", window.location.href + "/schedset/" +
1425     schedule.replace("/", "slh"));
1426     xhr.send();
1427     xhr.responseType = "json";
1428     xhr.onload = () => {
1429         if (xhr.readyState == 4 && xhr.status == 200) {
1430             console.log(xhr.response);
1431             location.reload();
1432         } else {
1433             console.log(`Error: ${xhr.status}`);
1434         }
1435     };
1436
1437     const suspensionHandle = () => {
1438         console.log("request to: " + window.location.href + "/
1439 suspension_toggle");
1440         const xhr = new XMLHttpRequest();
1441         xhr.open("GET", window.location.href + "/suspension_toggle");
1442         xhr.send();
1443         xhr.responseType = "json";
1444         xhr.onload = () => {
1445             if (xhr.readyState == 4 && xhr.status == 200) {
1446                 console.log(xhr.response);
1447                 location.reload();
1448             } else {
1449                 console.log(`Error: ${xhr.status}`);
1450             }
1451         };
1452
1453     const copyToClipBoard = (element) => {
1454         var copyText = document.getElementById(element);
1455         navigator.clipboard.writeText(copyText.innerHTML);
1456         toast("Copied to Clipboard");
1457     }
1458
1459     let notify_shown = false;
1460
1461     const toast = (text) => {
1462         if (notify_shown) return;
1463
1464         notify_shown = true;
1465
1466         var toast = document.createElement('div');
1467         toast.className = "clipboard_toast";
1468
1469         var message = document.createElement("div");
1470         message.textContent = text;
1471         toast.appendChild(message);
1472
1473         var close = document.createElement("div");

```

```

1474     close.className = "close_toast";
1475     close.innerHTML = "x"
1476     close.addEventListener("click", () => {
1477         toast.remove();
1478         notify_shown = false;
1479     })
1480     toast.append(close);
1481
1482     document.body.appendChild(toast);
1483
1484     window.setTimeout(() => {
1485         toast.remove();
1486         notify_shown = false;
1487     }, 3000);
1488 }
1489 };
1490
1491 // templates folder data: templates/tingle.js
1492 const TINGLE_JS: &str = {
1493     r#"/**"
1494     * tingle.js - A simple modal plugin written in pure JavaScript
1495     * @version v0.16.0
1496     * @link https://github.com/robinparisi/tingle#readme
1497     * @license MIT
1498     */
1499
1500 /* global define, module */
1501 (function (root, factory) {
1502     if (typeof define === 'function' && define.amd) {
1503         define(factory)
1504     } else if (typeof exports === 'object') {
1505         module.exports = factory()
1506     } else {
1507         root.tingle = factory()
1508     }
1509 })(this, function () {
1510     /* -----
1511     * == modal */
1512     /* ----- */
1513
1514     var isBusy = false
1515
1516     function Modal (options) {
1517         var defaults = {
1518             onClose: null,
1519             onOpen: null,
1520             beforeOpen: null,
1521             beforeClose: null,
1522             stickyFooter: false,
1523             footer: false,
1524             cssClass: [],
1525             closeLabel: 'Close',
1526             closeMethods: ['overlay', 'button', 'escape']

```

```
1527     }
1528
1529     // extends config
1530     this.opts = extend({}, defaults, options)
1531
1532     // init modal
1533     this.init()
1534 }
1535
1536 Modal.prototype.init = function () {
1537     if (this.modal) {
1538         return
1539     }
1540
1541     _build.call(this)
1542     _bindEvents.call(this)
1543
1544     // insert modal in dom
1545     document.body.appendChild(this.modal, document.body.firstChild)
1546
1547     if (this.opts.footer) {
1548         this.addFooter()
1549     }
1550
1551     return this
1552 }
1553
1554 Modal.prototype._busy = function (state) {
1555     isBusy = state
1556 }
1557
1558 Modal.prototype._isBusy = function () {
1559     return isBusy
1560 }
1561
1562 Modal.prototype.destroy = function () {
1563     if (this.modal === null) {
1564         return
1565     }
1566
1567     // restore scrolling
1568     if (this.isOpen()) {
1569         this.close(true)
1570     }
1571
1572     // unbind all events
1573     _unbindEvents.call(this)
1574
1575     // remove modal from dom
1576     this.modal.parentNode.removeChild(this.modal)
1577
1578     this.modal = null
1579 }
```

```

1580
1581     Modal.prototype.isOpen = function () {
1582         return !!this.modal.classList.contains('tingle-modal--visible')
1583     }
1584
1585     Modal.prototype.open = function () {
1586         if (this._isBusy()) return
1587         this._busy(true)
1588
1589         var self = this
1590
1591         // before open callback
1592         if (typeof self.opts.beforeOpen === 'function') {
1593             self.opts.beforeOpen()
1594         }
1595
1596         if (this.modal.style.removeProperty) {
1597             this.modal.style.removeProperty('display')
1598         } else {
1599             this.modal.style.removeAttribute('display')
1600         }
1601
1602         // prevent text selection when opening multiple times
1603         document.getSelection().removeAllRanges()
1604
1605         // prevent double scroll
1606         this._scrollPosition = window.pageYOffset
1607         document.body.classList.add('tingle-enabled')
1608         document.body.style.top = -this._scrollPosition + 'px'
1609
1610         // sticky footer
1611         this.setStickyFooter(this.opts.stickyFooter)
1612
1613         // show modal
1614         this.modal.classList.add('tingle-modal--visible')
1615
1616         // onOpen callback
1617         if (typeof self.opts.onOpen === 'function') {
1618             self.opts.onOpen.call(self)
1619         }
1620
1621         self._busy(false)
1622
1623         // check if modal is bigger than screen height
1624         this.checkOverflow()
1625
1626         return this
1627     }
1628
1629     Modal.prototype.close = function (force) {
1630         if (this._isBusy()) return
1631         this._busy(true)
1632         force = force || false

```

```

1633
1634     // before close
1635     if (typeof this.opts.beforeClose === 'function') {
1636         var close = this.opts.beforeClose.call(this)
1637         if (!close) {
1638             this._busy(false)
1639             return
1640         }
1641     }
1642
1643     document.body.classList.remove('tingle-enabled')
1644     document.body.style.top = null
1645     window.scrollTo({
1646         top: this._scrollPosition,
1647         behavior: 'instant'
1648     })
1649
1650     this.modal.classList.remove('tingle-modal--visible')
1651
1652     // using similar setup as onOpen
1653     var self = this
1654
1655     self.modal.style.display = 'none'
1656
1657     // onClose callback
1658     if (typeof self.opts.onClose === 'function') {
1659         self.opts.onClose.call(this)
1660     }
1661
1662     // release modal
1663     self._busy(false)
1664 }
1665
1666 Modal.prototype.setContent = function (content) {
1667     // check type of content : String or Node
1668     if (typeof content === 'string') {
1669         this.modalBoxContent.innerHTML = content
1670     } else {
1671         this.modalBoxContent.innerHTML = ''
1672         this.modalBoxContent.appendChild(content)
1673     }
1674
1675     if (this.isOpen()) {
1676         // check if modal is bigger than screen height
1677         this.checkOverflow()
1678     }
1679
1680     return this
1681 }
1682
1683 Modal.prototype.getContent = function () {
1684     return this.modalBoxContent
1685 }
```

```

1686
1687     Modal.prototype.addFooter = function () {
1688         // add footer to modal
1689         _buildFooter.call(this)
1690
1691         return this
1692     }
1693
1694     Modal.prototype.setFooterContent = function (content) {
1695         // set footer content
1696         this.modalBoxFooter.innerHTML = content
1697
1698         return this
1699     }
1700
1701     Modal.prototype.getFooterContent = function () {
1702         return this.modalBoxFooter
1703     }
1704
1705     Modal.prototype.setStickyFooter = function (isSticky) {
1706         // if the modal is smaller than the viewport height, we don't need
1707         // sticky
1708         if (!this.isOverflow()) {
1709             isSticky = false
1710         }
1711
1712         if (isSticky) {
1713             if (this.modalBox.contains(this.modalBoxFooter)) {
1714                 this.modalBox.removeChild(this.modalBoxFooter)
1715                 this.modal.appendChild(this.modalBoxFooter)
1716                 this.modalBoxFooter.classList.add('tingle-modal-box__footer--'
1717                     'sticky')
1718                 _recalculateFooterPosition.call(this)
1719             }
1720             this.modalBoxContent.style['padding-bottom'] =
1721                 this.modalBoxFooter.clientHeight + 20 + 'px'
1722         } else if (this.modalBoxFooter) {
1723             if (!this.modalBox.contains(this.modalBoxFooter)) {
1724                 this.modal.removeChild(this.modalBoxFooter)
1725                 this.modalBox.appendChild(this.modalBoxFooter)
1726                 this.modalBoxFooter.style.width = 'auto'
1727                 this.modalBoxFooter.style.left = ''
1728                 this.modalBoxContent.style['padding-bottom'] = ''
1729                 this.modalBoxFooter.classList.remove('tingle-modal-
1730                 box__footer--sticky')
1731             }
1732         }
1733
1734     Modal.prototype.addFooterBtn = function (label, cssClass, callback) {
1735         var btn = document.createElement('button')

```

```

1735
1736     // set label
1737     btn.innerHTML = label
1738
1739     // bind callback
1740     btn.addEventListener('click', callback)
1741
1742     if (typeof cssClass === 'string' && cssClass.length) {
1743         // add classes to btn
1744         cssClass.split(' ').forEach(function (item) {
1745             btn.classList.add(item)
1746         })
1747     }
1748
1749     this.modalBoxFooter.appendChild(btn)
1750
1751     return btn
1752 }
1753
1754 Modal.prototype.resize = function () {
1755     // eslint-disable-next-line no-console
1756     console.warn('Resize is deprecated and will be removed in version
1757     1.0')
1758 }
1759
1760 Modal.prototype.isOverflow = function () {
1761     var viewportHeight = window.innerHeight
1762     var modalHeight = this.modalBox.clientHeight
1763
1764     return modalHeight >= viewportHeight
1765 }
1766
1767 Modal.prototype.checkOverflow = function () {
1768     // only if the modal is currently shown
1769     if (this.modal.classList.contains('tingle-modal--visible')) {
1770         if (this.isOverflow()) {
1771             this.modal.classList.add('tingle-modal--overflow')
1772         } else {
1773             this.modal.classList.remove('tingle-modal--overflow')
1774         }
1775
1776         if (!this.isOverflow() && this.opts.stickyFooter) {
1777             this.setStickyFooter(false)
1778         } else if (this.isOverflow() && this.opts.stickyFooter) {
1779             _recalculateFooterPosition.call(this)
1780             this.setStickyFooter(true)
1781         }
1782     }
1783
1784     /* ----- */
1785     /* == private methods */
1786     /* ----- */

```

```

1787     function closeIcon () {
1788         return '<svg viewBox="0 0 10 10" xmlns="http://www.w3.org/2000/
1789         svg"><path d="M.3 9.7c.2.2.4.3.7.3.3 0 .5-.1.7-.3L5 6.4l3.3
1790         3.3c.2.2.5.3.7.3.2 0 .5-.1.7-.3.4-.4.4-1 0-1.4L6.4 5l3.3-3.3c.4-.4.4-1
1791         0-1.4-.4-.4-1.4 0L5 3.6 1.7.3C1.3-.1.7-.1.3.3c-.4.4-.4 1 0 1.4L3.6 5 .3
1792         8.3c-.4.4-.4 1 0 1.4z" fill="black" fill-rule="nonzero"/></svg>'
1793     }
1794
1795     function _recalculateFooterPosition () {
1796         if (!this.modalBoxFooter) {
1797             return
1798         }
1799         this.modalBoxFooter.style.width = this.modalBox.clientWidth + 'px'
1800         this.modalBoxFooter.style.left = this.modalBox.offsetLeft + 'px'
1801     }
1802
1803     function _build () {
1804         // wrapper
1805         this.modal = document.createElement('div')
1806         this.modal.classList.add('tingle-modal')
1807
1808         // remove cursor if no overlay close method
1809         if (this.opts.closeMethods.length === 0 ||
1810             this.opts.closeMethods.indexOf('overlay') === -1) {
1811             this.modal.classList.add('tingle-modal--noOverlayClose')
1812         }
1813
1814         this.modal.style.display = 'none'
1815
1816         // custom class
1817         this.opts.cssClass.forEach(function (item) {
1818             if (typeof item === 'string') {
1819                 this.modal.classList.add(item)
1820             }
1821         }, this)
1822
1823         // close btn
1824         if (this.opts.closeMethods.indexOf('button') !== -1) {
1825             this.modalCloseBtn = document.createElement('button')
1826             this.modalCloseBtn.type = 'button'
1827             this.modalCloseBtn.classList.add('tingle-modal__close')
1828
1829             this.modalCloseBtnIcon = document.createElement('span')
1830             this.modalCloseBtnIcon.classList.add('tingle-modal__closeIcon')
1831             this.modalCloseBtnIcon.innerHTML = closeIcon()
1832
1833             this.modalCloseBtnLabel = document.createElement('span')
1834             this.modalCloseBtnLabel.classList.add('tingle-modal__closeLabel')
1835             this.modalCloseBtnLabel.innerHTML = this.opts.closeLabel
1836
1837             this.modalCloseBtn.appendChild(this.modalCloseBtnIcon)
1838             this.modalCloseBtn.appendChild(this.modalCloseBtnLabel)

```

```

1835     }
1836
1837     // modal
1838     this.modalBox = document.createElement('div')
1839     this.modalBox.classList.add('tingle-modal-box')
1840
1841     // modal box content
1842     this.modalBoxContent = document.createElement('div')
1843     this.modalBoxContent.classList.add('tingle-modal-box__content')
1844
1845     this.modalBox.appendChild(this.modalBoxContent)
1846
1847     if (this.opts.closeMethods.indexOf('button') !== -1) {
1848         this.modal.appendChild(this.modalCloseBtn)
1849     }
1850
1851     this.modal.appendChild(this.modalBox)
1852 }
1853
1854 function _buildFooter () {
1855     this.modalBoxFooter = document.createElement('div')
1856     this.modalBoxFooter.classList.add('tingle-modal-box__footer')
1857     this.modalBox.appendChild(this.modalBoxFooter)
1858 }
1859
1860 function _bindEvents () {
1861     this._events = {
1862         clickCloseBtn: this.close.bind(this),
1863         clickOverlay: _handleClickOutside.bind(this),
1864         resize: this.checkOverflow.bind(this),
1865         keyboardNav: _handleKeyboardNav.bind(this)
1866     }
1867
1868     if (this.opts.closeMethods.indexOf('button') !== -1) {
1869         this.modalCloseBtn.addEventListener('click',
1870             this._events.clickCloseBtn)
1871     }
1872
1873     this.modal.addEventListener('mousedown', this._events.clickOverlay)
1874     window.addEventListener('resize', this._events.resize)
1875     document.addEventListener('keydown', this._events.keyboardNav)
1876 }
1877
1878 function _handleKeyboardNav (event) {
1879     // escape key
1880     if (this.opts.closeMethods.indexOf('escape') !== -1 && event.which
1881 === 27 && this.isOpen()) {
1882         this.close()
1883     }
1884 }
1885 function _handleClickOutside (event) {
1886     // on macOS, click on scrollbar (hidden mode) will trigger close

```

```

event so we need to bypass this behavior by detecting scrollbar mode
1886     var scrollbarWidth = this.modal.offsetWidth -
this.modal.clientWidth
1887     var clickedOnScrollbar = event.clientX >= this.modal.offsetWidth -
15 // 15px is macOS scrollbar default width
1888     var isScrollable = this.modal.scrollHeight !==
this.modal.offsetHeight
1889     if (navigator.platform === 'MacIntel' && scrollbarWidth === 0 &&
clickedOnScrollbar && isScrollable) {
1890         return
1891     }
1892
1893     // if click is outside the modal
1894     if (this.opts.closeMethods.indexOf('overlay') !== -1 && !
_findAncestor(event.target, 'tingle-modal') &&
1895         event.clientX < this.modal.clientWidth) {
1896         this.close()
1897     }
1898 }
1899
1900 function _findAncestor (el, cls) {
1901     while ((el = el.parentElement) && !el.classList.contains(cls));
1902     return el
1903 }
1904
1905 function _unbindEvents () {
1906     if (this.opts.closeMethods.indexOf('button') !== -1) {
1907         this.modalCloseBtn.removeEventListener('click',
this._events.clickCloseBtn)
1908     }
1909     this.modal.removeEventListener('mousedown',
this._events.clickOverlay)
1910     window.removeEventListener('resize', this._events.resize)
1911     document.removeEventListener('keydown', this._events.keyboardNav)
1912 }
1913
1914 /* ----- */
1915 /* == helpers */
1916 /* ----- */
1917
1918 function extend () {
1919     for (var i = 1; i < arguments.length; i++) {
1920         for (var key in arguments[i]) {
1921             if (arguments[i].hasOwnProperty(key)) {
1922                 arguments[0][key] = arguments[i][key]
1923             }
1924         }
1925     }
1926     return arguments[0]
1927 }
1928
1929 /* ----- */
1930 /* == return */

```

```

1931  /* ----- */
1932
1933  return {
1934    modal: Modal
1935  }
1936 })
1937 "#"
1938 };
1939
1940 // templates folder data: templates/tingle.css
1941 const TINGLE_STYLES: &str = {
1942   r#"/**
1943   * tingle.js - A simple modal plugin written in pure JavaScript
1944   * @version v0.16.0
1945   * @link https://github.com/robinparisi/tingle#readme
1946   * @license MIT
1947   */"
1948
1949 // modified for cronframe
1950
1951 .tingle-modal * {
1952   box-sizing: border-box;
1953 }
1954
1955 .tingle-modal {
1956   position: fixed;
1957   top: 0;
1958   right: 0;
1959   bottom: 0;
1960   left: 0;
1961   z-index: 1000;
1962   display: flex;
1963   visibility: hidden;
1964   flex-direction: column;
1965   align-items: center;
1966   overflow: hidden;
1967   -webkit-overflow-scrolling: touch;
1968   background: rgba(0, 0, 0, .9);
1969   opacity: 0;
1970   cursor: url("data:image/svg+xml,%3Csvg width='19' height='19'
1971 xmlns='http://www.w3.org/2000/svg'%3E%3Cpath d='M15.514.535l-6.42
1972 6.42L2.677.536a1.517 1.517 0 00-2.14 0 1.517 1.517 0 000 2.14l6.42
1973 6.419-6.42 6.419a1.517 1.517 0 000 2.14 1.517 1.517 0 002.14 0l6.419-6.42
1974 6.419 6.42a1.517 1.517 0 002.14 0 1.517 1.517 0 000-2.14l-6.42-6.42
1975 6.42-6.418a1.517 1.517 0 000-2.14 1.516 1.516 0 00-2.14 0z' fill='%23FFF'
1976 fill-rule='nonzero'/%3E%3C/svg%3E"), auto;
1977 }
1978
1979 @supports ((-webkit-backdrop-filter: blur(12px)) or (backdrop-filter:
1980 blur(12px))) {
1981   .tingle-modal {
1982     -webkit-backdrop-filter: blur(12px);
1983     backdrop-filter: blur(12px);

```

```
1977     }
1978   }
1979
1980 /* confirm and alerts
1981 -----
1982
1983 .tingle-modal--confirm .tingle-modal-box {
1984   text-align: center;
1985 }
1986
1987 /* modal
1988 -----
1989
1990 .tingle-modal--noOverlayClose {
1991   cursor: default;
1992 }
1993
1994 .tingle-modal--noClose .tingle-modal__close {
1995   display: none;
1996 }
1997
1998 .tingle-modal__close {
1999   position: fixed;
2000   top: 2.5rem;
2001   right: 2.5rem;
2002   z-index: 1000;
2003   padding: 0;
2004   width: 2rem;
2005   height: 2rem;
2006   border: none;
2007   background-color: transparent;
2008   color: #fff;
2009   cursor: pointer;
2010 }
2011
2012 .tingle-modal__close svg * {
2013   fill: currentColor;
2014 }
2015
2016 .tingle-modal__closeLabel {
2017   display: none;
2018 }
2019
2020 .tingle-modal__close:hover {
2021   color: var(--light-orange);
2022   background: transparent;
2023 }
2024
2025 .tingle-modal__close:active {
2026   color: var(--dark-orange);
2027   background: transparent;
2028 }
2029
```

```

2030 .tingle-modal-box {
2031   max-width: 600px;
2032   position: relative;
2033   flex-shrink: 0;
2034   margin-top: auto;
2035   margin-bottom: auto;
2036   width: 60%;
2037   border-radius: 4px;
2038   background: var(--container-bg);
2039   opacity: 1;
2040   cursor: auto;
2041   will-change: transform, opacity;
2042 }
2043
2044 .tingle-modal-box__content {
2045   padding: 3rem 3rem;
2046 }
2047
2048 .tingle-modal-box__footer {
2049   padding: 1.5rem 2rem;
2050   width: auto;
2051   border-bottom-right-radius: 4px;
2052   border-bottom-left-radius: 4px;
2053   background-color: var(--content-bg);
2054   cursor: auto;
2055 }
2056
2057 .tingle-modal-box__footer::after {
2058   display: table;
2059   clear: both;
2060   content: "";
2061 }
2062
2063 .tingle-modal-box__footer--sticky {
2064   position: fixed;
2065   bottom: -200px; /* TODO : find a better way */
2066   z-index: 10001;
2067   opacity: 1;
2068   transition: bottom .3s ease-in-out .3s;
2069 }
2070
2071 /* state
2072 -----
2073
2074 .tingle-enabled {
2075   position: fixed;
2076   right: 0;
2077   left: 0;
2078   overflow: hidden;
2079 }
2080
2081 .tingle-modal--visible .tingle-modal-box__footer {
2082   bottom: 0;

```

```
2083 }
2084
2085 .tingle-modal--visible {
2086   visibility: visible;
2087   opacity: 1;
2088 }
2089
2090 .tingle-modal--visible .tingle-modal-box {
2091   animation: scale .2s cubic-bezier(.68, -.55, .265, 1.55) forwards;
2092 }
2093
2094 .tingle-modal--overflow {
2095   overflow-y: scroll;
2096   padding-top: 8vh;
2097 }
2098
2099 /* btn
2100 -----
2101
2102 .tingle-btn {
2103   display: inline-block;
2104   margin: 0 .5rem;
2105   padding: 1rem 2rem;
2106   border: none;
2107   background-color: grey;
2108   box-shadow: none;
2109   color: #fff;
2110   vertical-align: middle;
2111   text-decoration: none;
2112   font-size: inherit;
2113   font-family: inherit;
2114   line-height: normal;
2115   cursor: pointer;
2116   transition: background-color .4s ease;
2117 }
2118
2119 .tingle-btn--primary {
2120   background-color: #3498db;
2121 }
2122
2123 .tingle-btn--danger {
2124   background-color: var(--light-orange);
2125 }
2126
2127 .tingle-btn--default {
2128   background-color: #34495e;
2129 }
2130
2131 .tingle-btn--pull-left {
2132   float: left;
2133 }
2134
2135 .tingle-btn--pull-right {
```

```
2136   float: right;
2137 }
2138
2139 /* responsive
2140 -----
2141
2142 @media (max-width : 540px) {
2143   .tingle-modal {
2144     top: 0px;
2145     display: block;
2146     padding-top: 60px;
2147     width: 100%;
2148   }
2149
2150   .tingle-modal-box {
2151     width: auto;
2152     border-radius: 0;
2153   }
2154
2155   .tingle-modal-box__content {
2156     overflow-y: scroll;
2157   }
2158
2159   .tingle-modal--noClose {
2160     top: 0;
2161   }
2162
2163   .tingle-modal--noOverlayClose {
2164     padding-top: 0;
2165   }
2166
2167   .tingle-modal-box__footer .tingle-btn {
2168     display: block;
2169     float: none;
2170     margin-bottom: 1rem;
2171     width: 100%;
2172   }
2173
2174   .tingle-modal__close {
2175     top: 0;
2176     right: 0;
2177     left: 0;
2178     display: block;
2179     width: 100%;
2180     height: 60px;
2181     border: none;
2182     background-color: var(--content-bg);
2183     box-shadow: none;
2184     color: #fff;
2185   }
2186
2187   .tingle-modal__closeLabel {
2188     display: inline-block;
```

```

2189     vertical-align: middle;
2190     font-size: 1.6rem;
2191     font-family: -apple-system, BlinkMacSystemFont, "Segoe UI",
2192     "Roboto", "Oxygen", "Ubuntu", "Cantarell", "Fira Sans", "Droid Sans",
2193     "Helvetica Neue", sans-serif;
2194   }
2195
2196   .tingle-modal__closeIcon {
2197     display: inline-block;
2198     margin-right: .8rem;
2199     width: 1.6rem;
2200     vertical-align: middle;
2201     font-size: 0;
2202   }
2203 }
2204
2205 /* animations
2206 -----
2207
2208 @keyframes scale {
2209   0% {
2210     opacity: 0;
2211     transform: scale(.9);
2212   }
2213   100% {
2214     opacity: 1;
2215     transform: scale(1);
2216   }
2217 }

```

## A-7 - src/logger.rs

```

1 //! Default logger setup for the cronframe framework and the testing suite
2
3 use crate::{config::read_config, utils};
4 use chrono::Duration;
5 use log4rs::{
6   append::{
7     file::FileAppender,
8     rolling_file::{
9       policy::compound::{
10         roll::fixed_window::FixedWindowRoller,
11         trigger::size::SizeTrigger, CompoundPolicy,
12       },
13       RollingFileAppender,
14     },
15   config::{Appender, Config, Root},
16   encode::pattern::PatternEncoder,
17 };

```

```

18
19 /// this logger configuration is used for testing
20 pub fn appender_logger(log_file: &str) -> log4rs::Handle {
21     let pattern = "{d(%Y-%m-%d %H:%M:%S %Z)} {l} {t} - {m}{n}";
22
23     let log_file = FileAppender::builder()
24         .encoder(Box::new(PatternEncoder::new(pattern)))
25         .append(false)
26         .build(log_file)
27         .expect("appender_logger log file unwrap error");
28
29     let config = Config::builder()
30         .appender(Appender::builder().build("log_file",
31 Box::new(log_file)))
32         .build(
33             Root::builder()
34                 .appender("log_file")
35                 .build(log::LevelFilter::Info),
36         )
37         .expect("appender_logger config unwrap error");
38
39     log4rs::init_config(config).expect("appender_logger init error")
40 }
41
42 /// this is used to change the log file for each new test
43 pub fn appender_config(log_file: &str) -> log4rs::Config {
44     let pattern = "{d(%Y-%m-%d %H:%M:%S %Z)} {l} {t} - {m}{n}";
45
46     let log_file = FileAppender::builder()
47         .encoder(Box::new(PatternEncoder::new(pattern)))
48         .append(false)
49         .build(log_file)
50         .expect("appender_config log file unwrap error");
51
52     Config::builder()
53         .appender(Appender::builder().build("log_file",
54 Box::new(log_file)))
55         .build(
56             Root::builder()
57                 .appender("log_file")
58                 .build(log::LevelFilter::Info),
59         )
60         .expect("appender_logger config unwrap error")
61 }
62
63 /// this sets the logger from either the default configuration or from
the toml file
64 pub fn rolling_logger() -> log4rs::Handle {
65     let mut window_size = 3;
66     let mut size_limit = 1000 * 1024;
67     let mut log_dir = "log".to_string();
68     let mut latest_file_name = "latest".to_string();
69     let mut archive_file_name = "archive".to_string();

```

```

68     let mut pattern = "{d(%Y-%m-%d %H:%M:%S %Z)} {l} {t} - {m}
69     {n}.to_string();
70
71     if let Some(config_data) = read_config() {
72         if let Some(logger_data) = config_data.logger {
73             if let Some(data) = logger_data.archive_files {
74                 window_size = data;
75             }
76             if let Some(data) = logger_data.file_size {
77                 size_limit = size_limit * data;
78             }
79             if let Some(data) = logger_data.dir {
80                 log_dir = data;
81             }
82             if let Some(data) = logger_data.latest_file_name {
83                 latest_file_name = data;
84             }
85             if let Some(data) = logger_data.archive_file_name {
86                 archive_file_name = data;
87             }
88             if let Some(data) = logger_data.msg_pattern {
89                 pattern = data;
90             }
91             if let Some(data) = logger_data.level_filter {
92                 match data.as_str() {
93                     "off" => level_filter = log::LevelFilter::Off,
94                     "error" => level_filter = log::LevelFilter::Error,
95                     "warn" => level_filter = log::LevelFilter::Warn,
96                     "debug" => level_filter = log::LevelFilter::Debug,
97                     _ => (),
98                 }
99             }
100        }
101    };
102
103    if std::env::var("CRONFRAME_CLI").is_ok() {
104        let home_dir = utils::home_dir();
105        log_dir = format!("{}/.cronframe/log");
106    }
107
108    let archive_file = format!("{}{}/
109 {archive_file_name}.log").replace(".log", "_{}.log");
110
111    // retain latest and archive logfiles at restart as per rolling
112    // policy
113    if !std::path::Path::new(&format!("{}{}/
114 {latest_file_name}")).exists() {
115        let _ = std::fs::remove_file(format!(
116            "./{}{}/{}_{}.log",
117            log_dir, archive_file_name, latest_file_name));
118    }
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1196
1197
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
224
```

```

117     for i in (1..=(window_size - 1)).rev() {
118         let _ = std::fs::rename(
119             format!("{}{}/{}{}_{}.log", log_dir, archive_file_name, i - 1),
120             format!("{}{}/{}{}_{}_.log", log_dir, archive_file_name, i),
121         );
122     }
123
124     let _ = std::fs::rename(
125         format!("{}{}/{}{}_log"), // latest file
126         format!("{}{}/{}{}_0.log"), // first file
127     );
128
129     std::thread::sleep(Duration::seconds(5).to_std().unwrap());
130 }
131
132 let roller = FixedWindowRoller::builder()
133     .build(&archive_file, window_size)
134     .unwrap();
135
136 let trigger = SizeTrigger::new(size_limit);
137
138 let policy = CompoundPolicy::new(Box::new(trigger),
Box::new(roller));
139
140 let log_file = RollingFileAppender::builder()
141     .encoder(Box::new(PatternEncoder::new(&pattern)))
142     .append(false)
143     .build(
144         &format!("{}{}/{}{}_log"),
145         Box::new(policy),
146     )
147     .expect("rolling_logger log file unwrap error");
148
149 let config = Config::builder()
150     .appender(Appender::builder().build("log_file",
Box::new(log_file)))
151     .build(Root::builder().appender("log_file").build(level_filter))
152     .expect("rolling_logger config unwrap error");
153
154 log4rs::init_config(config).expect("rolling_logger init error")
155 }
156

```

## A-8 - src/config.rs

```

1 //! Configuration available in `cronframe.toml`
2
3 use crate::utils;
4 use rocket::serde::Deserialize;
5 use std::fs;
6 use toml;
7

```

```

8 #[derive(Deserialize)]
9 #[serde(crate = "rocket::serde")]
10 pub struct ConfigData {
11     pub webserver: Option<ServerConfig>,
12     pub logger: Option<LoggerConfig>,
13     pub scheduler: Option<SchedulerConfig>,
14 }
15
16 #[derive(Deserialize)]
17 #[serde(crate = "rocket::serde")]
18 pub struct ServerConfig {
19     pub port: Option<u16>,
20     pub ip: Option<String>,
21 }
22
23 #[derive(Deserialize)]
24 #[serde(crate = "rocket::serde")]
25 pub struct LoggerConfig {
26     pub dir: Option<String>,
27     pub file_size: Option<u64>,
28     pub archive_files: Option<u32>,
29     pub latest_file_name: Option<String>,
30     pub archive_file_name: Option<String>,
31     pub msg_pattern: Option<String>,
32     pub level_filter: Option<String>,
33 }
34
35 #[derive(Deserialize)]
36 #[serde(crate = "rocket::serde")]
37 pub struct SchedulerConfig {
38     pub grace: Option<u32>,
39 }
40
41 /// This function reads cronframe configuration data from the
`cronframe.toml` file.
42 /**
43 /// There are three sections to the configuration:
44 /// - webserver
45 /// - logger
46 /// - scheduler
47 /**
48 /**
49 /// [webserver]
50 /// port = 8098
51 /**
52 /// [logger]
53 /// dir = "log"
54 /// file_size = 1 # this is in MB
55 /// archive_files = 3
56 /// latest_file_name = "latest"
57 /// archive_file_name = "archive"
58 /// msg_pattern = "{l} {t} - {m}{n}"
59 /// level_filter = "info"

```

```

60 /**
61  /// [scheduler]
62  /// grace = 250 # this is in ms
63 /**
64 /**
65 pub fn read_config() -> Option<ConfigData> {
66     let filename = if std::env::var("CRONFRAME_CLI").is_ok() {
67         let home_dir = utils::home_dir();
68         &format!("{}/.cronframe/cronframe.toml", home_dir)
69     } else {
70         "cronframe.toml"
71     };
72
73     if let Ok(file_content) = fs::read_to_string(filename) {
74         if let Ok(data) = toml::from_str(&file_content) {
75             data
76         } else {
77             error!("cronframe.toml - data read error");
78             None
79         }
80     } else {
81         info!("cronframe.toml - file not found");
82         None
83     }
84 }

```

## A-9 - src/utils.rs

```

1 //! Utilities
2
3 use crate::config::read_config;
4 use chrono::{DateTime, Local, Utc};
5
6 /// Conversion from UTC to Local time
7 pub fn local_time(utc_time: DateTime<Utc>) -> DateTime<Local> {
8     let local_time: DateTime<Local> = DateTime::from(utc_time);
9     local_time
10 }
11
12 pub fn home_dir() -> String {
13     let tmp = home::home_dir().unwrap();
14     tmp.to_str().unwrap().to_owned()
15 }
16
17 pub fn ip_and_port() -> (String, u16) {
18     match read_config() {
19         Some(config_data) => {
20             if let Some(webserver_data) = config_data.webserver {
21                 (
22                     webserver_data.ip.unwrap_or_else(|| "127.0.0.1".to_string()),
23                     webserver_data.port.unwrap_or_else(|| 8098),

```

```

24             )
25         } else {
26             ("localhost".to_string(), 8098)
27         }
28     }
29     None => ("localhost".to_string(), 8098),
30 }
31 }

```

## A-10 - src/bin.rs

```

1  //! CronFrame CLI Tool v0.1.3
2  //! Use the cronframe help command for details
3
4  use clap::{arg, command};
5  use colored::*;
6  use cronframe::{
7      utils::{self, ip_and_port},
8      web_server, CronFilter, CronFrame,
9  };
10 use std::{
11     fs,
12     io::BufRead,
13     path::Path,
14     process::{Command, Stdio},
15 };
16
17 fn main() {
18     std::env::set_var("CRONFRAME_CLI", "true");
19
20     // cli args parsing
21     let matches = command!()
22         .propagate_version(true)
23         .subcommand_required(true)
24         .arg_required_else_help(true)
25         // cronframe start
26         .subcommand(
27             clap::Command::new("start")
28                 .about("Start the CronFrame Webserver and Job Scheduler
in background."),
29         )
30         // cronframe run
31         .subcommand(
32             clap::Command::new("run")
33                 .about("Run the CronFrame Webserver and Job Scheduler in
the terminal."),
34         )
35         // cronframe add EXPR TIMEOUT JOB
36         .subcommand(
37             clap::Command::new("add")
38                 .about("Adds a new cli job to a CronFrame instance.")
39                 .args(&[

```

```

40                     arg!([EXPR] "The Cron Expression to use for job
scheduling."),
41                     arg!([TIMEOUT] "The value in ms to use for the
timeout."),
42                     arg!([JOB] "The path containing the source code of
the job."),
43                     ],
44                     .arg_required_else_help(true)
45                     .arg(
46                         arg!(-p --port <VALUE>)
47                             .required(false)
48                             .action(clap::ArgAction::Set),
49                     ),
50                 )
51             // cronframe load
52             .subcommand(
53                 clap::Command::new("load")
54                     .about("Load jobs from definition file.")
55                     .arg(
56                         arg!(-f --file <PATH>)
57                             .required(false)
58                             .action(clap::ArgAction::Set),
59                     ),
60             )
61             // cronframe scheduler ACTION
62             .subcommand(
63                 clap::Command::new("scheduler")
64                     .about("Perform actions on the scheduler like start and
stop")
65                     .args(&[arg!([ACTION] "Action to perform = (start,
stop)"))]
66                     .arg_required_else_help(true)
67                     .arg(
68                         arg!(-p --port <VALUE>)
69                             .required(false)
70                             .action(clap::ArgAction::Set),
71                     ),
72             )
73             // cronframe shutdown
74             .subcommand(
75                 clap::Command::new("shutdown")
76                     .about("Shutdown the CronFrame Webserver and Job
Scheduler."),
77             )
78             .get_matches();
79
80         match matches.subcommand() {
81             Some(("start", _)) => start_command(),
82             Some(("shutdown", _)) => shutdown_command(),
83             Some(("run", _)) => run_command(),
84             Some(("add", sub_matches)) => {
85                 let expr = sub_matches.get_one::<String>("EXPR").unwrap();
86                 let timeout =

```

```

sub_matches.get_one::<String>("TIMEOUT").unwrap();
87         let job = sub_matches.get_one::<String>("JOB").unwrap();
88         let port_option = sub_matches.get_one::<String>("port");
89         add_command(expr, timeout, job, port_option);
90     }
91     Some(("load", sub_matches)) => {
92         let file = sub_matches.get_one::<String>("file");
93         load_command(file);
94     }
95     Some(("scheduler", sub_matches)) => {
96         let action =
sub_matches.get_one::<String>("ACTION").unwrap();
97         let port_option = sub_matches.get_one::<String>("port");
98         scheduler_command(action, port_option);
99     }
100    _ => unreachable!("Exhausted list of subcommands and
subcommand_required prevents `None`"),
101 }
102 }
103
104 fn start_command() {
105     cronframe_folder();
106
107     let (ip, port) = ip_and_port();
108     if is_running(&ip, port) {
109         println!(
110             "{} address 'http://{}:{}' is already busy.",
111             "Error:".red().bold()
112         );
113         return;
114     }
115
116     let _build = Command::new("cronframe")
117         .args(["run"])
118         .stdin(Stdio::null())
119         .stdout(Stdio::null())
120         .stderr(Stdio::null())
121         .spawn()
122         .expect("cronframe run failed");
123
124     println!("CronFrame will soon be available at http://{}:{}");
125 }
126
127 fn shutdown_command() {
128     let (ip, port) = ip_and_port();
129     let req_url = format!("http://{}:{}shutdown");
130
131     match reqwest::blocking::get(req_url) {
132         Ok(_) => {
133             println!("CronFrame will soon shutdown.");
134         }
135         Err(_) => {
136             println!(

```

```

137             "{} no instance found at http://{:ip}:{:port}",
138             "Error:".red().bold()
139         );
140     }
141 }
142 }
143
144 fn run_command() {
145     cronframe_folder();
146     let (ip, port) = ip_and_port();
147     if is_running(&ip, port) {
148         println!(
149             "{} address 'http://{:ip}:{:port}' is already busy",
150             "Error:".red().bold()
151         );
152         return;
153     }
154     let _ = CronFrame::init(Some(CronFilter::CLI), true).run();
155 }
156
157 fn add_command(expr: &str, timeout: &str, job: &str, port_option:
158 Option<&String>) {
159     let home_dir = utils::home_dir().replace("\\", "/");
160
161     let escaped_expr = expr.replace("/", "slh");
162
163     let tmp: Vec<_> = if cfg!(target_os = "windows") {
164         job.split("\\").collect()
165     } else {
166         job.split("/").collect()
167     };
168     let tmp = tmp.iter().filter(|x| !x.is_empty()); // needed if there
169     is a / after the name of the create's folder
170     let job_name = tmp.last().unwrap().replace(".rs", "");
171
172     println!("Compiling {} Job");
173
174     if Path::new(&job).is_file() {
175         // compile the "script" job
176         let compile_command = Command::new("rustc")
177             .args([
178                 job,
179                 "-o",
180                 &format!("{}{}/.cronframe/cli_jobs/{}", home_dir, job_name),
181             ])
182             .status();
183
184         match compile_command {
185             Err(error) => {
186                 println!("{} {}", "Error:".red().bold(),
error.to_string());
187                 return;
188             }

```

```

187             _ => (),
188         }
189     } else {
190         // compile the "crate" job
191         let compile_command = Command::new("cargo")
192             .args([
193                 "build",
194                 "--release",
195                 "--target-dir",
196                 &format!("{}{}/.cronframe/cargo_targets/{}"),
197                 {job_name}),
198                 ])
199                 .current_dir(job)
200                 .status();
201
202         match compile_command {
203             Err(error) => {
204                 println!("{} {}", "Error:".red().bold(),
205                     error.to_string());
206                 return;
207             }
208         }
209
210         let copy_command = if cfg!(target_os = "windows") {
211             println!(
212                 "current dir = {}",
213                 format!("{}{}/.cronframe/cargo_targets/{}{}/",
214                     release))
215             );
216             println!(
217                 "cmd /C copy {} {}",
218                 format!("{}{}.exe"),
219                 format!("{}{}/.cronframe/cli_jobs").replace("\\",
220                     "/"));
221
222         Command::new("cmd")
223             .args(&[
224                 "/C",
225                 "copy",
226                 &format!("{}{}.exe"),
227                 &format!("{}{}/.cronframe/
228                     cli_jobs/").replace("/", "\\"),
229                 ])
230                 .current_dir(format!(
231                     "{}{}/.cronframe/cargo_targets/{}{}/",
232                     release
233                     ))
234                     .status()
235     } else {
236         // copy binary on unix systems
237         Command::new("cp")

```

```

234         .args([
235             &job_name,
236             &format!("{}{}/.cronframe/cli_jobs/",
237             {job_name}),
238             ])
239             .current_dir(format!(
240                 "{}{}/.cronframe/cargo_targets/{job_name}/",
241                 release"
242             ))
243             .status()
244     );
245
246     match copy_command {
247         Err(error) => {
248             println!("{} {}", "Error:".red().bold(),
249             error.to_string());
250             return;
251         }
252     }
253
254     // get the ip_address and port
255     // check if a cronframe instance is running
256     // send the job to the running cronframe instance
257     // localhost::8098/add_cli_job/<expr>/<timeout>/<job>
258
259     let (ip, mut port) = ip_and_port();
260
261     if port_option.is_some() {
262         port = port_option.unwrap().parse().unwrap();
263     }
264
265     if !is_running(&ip, port) {
266         println!(
267             "{} no instance found at http://{}:{port}",
268             "Error:".red().bold()
269         );
270         return;
271     }
272
273     let req_url = format!("http://{}:{}{}/add_cli_job/",
274     {escaped_expr}/{timeout}/{job_name}");
275
276     match reqwest::blocking::get(req_url) {
277         Ok(_) => {
278             println!("Added Job to CronFrame");
279             println!(" Name: {job_name}");
280             println!(" Cron Expression: {expr}");
281             println!(" Timeout: {timeout}");
282         }
283         Err(error) => {
284             println!("{} {error}", "Error:".red().bold());
285         }
286     }
287 }
```

```

283         }
284     }
285 }
286
287 fn scheduler_command(action: &str, port_option: Option<&String>) {
288     let (ip, mut port) = ip_and_port();
289
290     if port_option.is_some() {
291         port = port_option.unwrap().parse().unwrap();
292     }
293
294     if !is_running(&ip, port) {
295         println!(
296             "{} no instance found at http://{}:{port}",
297             "Error:".red().bold()
298         );
299         return;
300     }
301
302     match action.to_lowercase().as_str() {
303         "start" => {
304             let req_url = format!("http://{}:{port}/start_scheduler");
305
306             match reqwest::blocking::get(req_url) {
307                 Ok(_) => {
308                     println!("Scheduler will soon start.");
309                 }
310                 Err(error) => {
311                     println!("Error when starting the scheduler");
312                     println!("{}error");
313                 }
314             }
315         }
316         "stop" => {
317             let req_url = format!("http://{}:{port}/stop_scheduler");
318
319             match reqwest::blocking::get(req_url) {
320                 Ok(_) => {
321                     println!("Scheduler will soon stop.");
322                 }
323                 Err(error) => {
324                     println!(
325                         "{} {} when stopping the scheduler",
326                         "Error:".red().bold()
327                     );
328                 }
329             }
330         }
331         other => {
332             println!("{} '{}{}' action unknown.",
333             "Error:".red().bold());
334         }
335     }

```

```

335 }
336
337 fn load_command(file: Option<&String>) {
338     let (ip, port) = ip_and_port();
339     if !is_running(&ip, port) {
340         println!(
341             "{} no instance found at http://{}:{}",
342             "Error:".red().bold()
343         );
344         return;
345     }
346
347     let file_path = match file {
348         Some(path) => path.clone(),
349         None => format!("{}/.cronframe/job_list.txt",
350             utils::home_dir()),
351     };
352
353     match std::fs::read(file_path) {
354         Ok(content) => {
355             for line in content.lines().into_iter() {
356                 let line = line.unwrap();
357                 let cmpt: Vec<_> = line.split(" ").collect();
358
359                 let expr = if cmpt.len() == 9 {
360                     // expr made of 7 fields
361                     format!(
362                         "{} {} {} {} {} {}",
363                         cmpt[0], cmpt[1], cmpt[2], cmpt[3], cmpt[4],
364                         cmpt[5], cmpt[6]
365                     )
366                 } else {
367                     // expr made of 6 fields (year absent)
368                     format!(
369                         "{} {} {} {} {}",
370                         cmpt[0], cmpt[1], cmpt[2], cmpt[3], cmpt[4],
371                         cmpt[5]
372                     )
373                 };
374
375                 let timeout = if cmpt.len() == 9 { cmpt[7] } else
376                 { cmpt[6] };
377                 let job = if cmpt.len() == 9 { cmpt[8] } else
378                 { cmpt[7] };
379
380                 add_command(&expr, timeout, job, None);
381             }
382         }
383     }
384 }

```

```

383
384 fn cronframe_folder() {
385     let home_dir = utils::home_dir();
386
387     if !std::path::Path::new(&format!("{}{}/cronframe")).exists()
388     {
389         println!("Generating .cronframe directory content...");
390
391         let template_dir = format!("{}{}/cronframe/templates")
392             .replace("\\", "/");
393         let rocket_toml = format!("[debug]\n{}{}/\n{}{}/{} = \"{}{}{}\"");
394
395         fs::create_dir(format!("{}{}/cronframe"))
396             .expect("could not create .cronframe directory");
397         fs::create_dir(format!("{}{}/cronframe/cli_jobs"))
398             .expect("could not create .cronframe directory");
399
400         web_server::generate_template_dir();
401
402         let _ = fs::write(
403             Path::new(&format!("{}{}/cronframe/rocket.toml")),
404             rocket_toml,
405         );
406     }
407 }
408
409 fn is_running(ip: &str, port: u16) -> bool {
410     match reqwest::blocking::get(format!("http://{}:{}")) {
411         Ok(_) => true,
412         Err(_) => false,
413     }
414 }
```

## B - Code Listing for cronframe\_macro crate

### B-1 - src/lib.rs

```

1 //! Macros for [CronFrame](https://crates.io/crates/cronframe)
2
3 use proc_macro::*;

4 use quote::{format_ident, quote, ToTokens};
5 use syn::{self, parse_macro_input, punctuated::Punctuated, ItemFn,
ItemImpl, ItemStruct, Meta};
6
7 /// Global Job definition Macro
8 #[proc_macro_attribute]
9 pub fn cron(att: TokenStream, code: TokenStream) -> TokenStream {
10     let args = parse_macro_input!(att with Punctuated::<Meta, syn::Token![,]>::parse_terminated);
11
12     let args = args.into_iter().map(|x| {
```

```

13     x.require_name_value()
14     .map(|x| {
15         let arg_name = x.path.to_token_stream().to_string();
16         let arg_val = x.value.to_token_stream().to_string();
17         (arg_name, arg_val.replace("\\\"", ""))
18     })
19     .unwrap()
20 });
21
22 // should contain ("expr", "* * * * *")
23 let (arg_1_name, cron_expr) =
args.clone().peekable().nth(0).unwrap();
24
25 // should contain ("timeout", "u64")
26 let (arg_2_name, timeout) = args.peekable().nth(1).unwrap();
27
28 if arg_1_name == "expr" && arg_2_name == "timeout" {
29     let parsed = syn::parse::<ItemFn>(code.clone());
30
31     if parsed.is_ok() {
32         let origin_function =
parsed.clone().unwrap().to_token_stream();
33         let ident = parsed.clone().unwrap().sig.ident;
34         let job_name = ident.to_string();
35
36         let new_code = quote! {
37             // original function
38             #origin_function
39
40             // necessary for automatic job collection
41             cronframe::submit! {
42                 cronframe::JobBuilder::global_job(#job_name, #ident,
#cron_expr, #timeout)
43             }
44         };
45
46         return new_code.into();
47     } else if let Some(error) = parsed.err() {
48         println!("parse Error: {}", error);
49     }
50 }
51 code
52 }
53
54 /// Cron Object definition Macro
55 #[proc_macro_attribute]
56 pub fn cron_obj(_att: TokenStream, code: TokenStream) -> TokenStream {
57     let item_struct = syn::parse::<ItemStruct>(code.clone().unwrap());
58     let r#struct = item_struct.to_token_stream();
59     let ident_upper = format_ident!("{}",
item_struct.ident.clone().to_string().to_uppercase());
60     let struct_name = item_struct.ident;
61     let method_jobs = format_ident!("CRONFRAME_METHOD_JOBS_{}",
```

```

ident_upper);
62     let function_jobs = format_ident!("CRONFRAME_FUNCTION_JOBS_{}",
ident_upper);
63     let cf_fn_jobs_flag = format_ident!("CF_FN_JOBS_FLAG_{}",
ident_upper);
64     let cf_fn_jobs_channels = format_ident!("CF_FN_JOBS_CHANNELS_{}",
ident_upper);
65
66     // inject the tx field for the drop of method jobs
67     // this requires that the last field in the original struct is
followed by a ,
68     let struct_edited: proc_macro2::TokenStream = {
69         let mut tmp = r#struct.to_string();
70         if tmp.contains("{") {
71             tmp.insert_str(
72                 tmp.chars().count() - 1,
73                 "tx: Option<cronframe::Sender<String>>",
74             );
75         } else {
76             tmp.insert_str(
77                 tmp.chars().count() - 1,
78                 "{tx: Option<cronframe::Sender<String>>}",
79             );
80             tmp = (&tmp[0..tmp.len() - 1].to_string()).clone();
81         }
82         tmp.parse().unwrap()
83     };
84
85     // --- start --- building the new_cron_obj function
86     let new_cron_obj: proc_macro2::TokenStream = {
87         let type_name =
88             struct_name.clone().into_token_stream().to_string();
89         let mut function = String::from("fn new_cron_obj()");
90         if !item_struct.fields.is_empty() {
91             let mut tmp = item_struct.fields.iter().map(|x| {
92                 let field_name = x.ident.to_token_stream().to_string();
93                 let field_type = x.ty.to_token_stream().to_string();
94                 format!("{} : {}", field_name, field_type)
95             });
96             for _ in 0..item_struct.fields.len() {
97                 function.push_str(&tmp.next().unwrap());
98             }
99         }
100
101         function.push_str(") -> ");
102         function.push_str(&type_name);
103         function.push_str("{");
104         function.push_str(&type_name);
105         function.push_str("}");
106
107         if !item_struct.fields.is_empty() {
108             let mut tmp = item_struct.fields.iter().map(|x| {

```

```

109             let field_name = x.ident.to_token_stream().to_string();
110             format!("{}{}{}", field_name, ",")
111         });
112
113         for _ in 0..item_struct.fields.len() {
114             function.push_str(&tmp.next().unwrap());
115         }
116     }
117     function.push_str("tx: None");
118     function.push_str("{}");
119     function.push_str("{}");
120     function.parse().unwrap()
121 }; // --- end --- building the new_cron_obj
122
123 let new_code = quote! {
124     // the code of the original struct with the addition of the tx
125     field
126     #[derive(Clone)]
127     #struct_edited
128
129     // used to keep track of weather function jobs have been
130     gathered
131     static #cf_fn_jobs_flag: std::sync::Mutex<bool> =
132     std::sync::Mutex::new(false);
133     // channels used to manage to drop of function jobs
134     static #cf_fn_jobs_channels:
135     cronframe::Lazy<(cronframe::Sender<String>, cronframe::Receiver<String>)> =
136     cronframe::Lazy::new(|| cronframe::bounded(1));
137
138     // drop for method jobs
139     impl Drop for #struct_name {
140         // this drops method jobs only
141         fn drop(&mut self) {
142             if self.tx.is_some(){
143                 let _ =
144                     self.tx.as_ref().unwrap().send("JOB_DROP".to_string());
145             }
146         }
147     }
148
149     // drop for function jobs
150     impl #struct_name {
151         // the new_cron_obj function
152         #new_cron_obj
153
154         // associated funciton of cron objects to drop function jobs
155         fn cf_drop_fn() {
156             if *#cf_fn_jobs_flag.lock().unwrap(){
157                 for func in #function_jobs{
158                     let _ =
159                     #cf_fn_jobs_channels.0.send("JOB_DROP".to_string());
160                 }
161             *#cf_fn_jobs_flag.lock().unwrap() = false;
162         }
163     }
164 }
```

```

155         }
156     }
157 }
158
159 #[cronframe::distributed_slice]
160 static #method_jobs: [fn(std::sync::Arc<Box<dyn std::any::Any +
Send + Sync>>) -> cronframe::JobBuilder<'static>];
161
162 #[cronframe::distributed_slice]
163 static #function_jobs: [fn() -> cronframe::JobBuilder<'static>];
164 };
165
166 new_code.into()
167 }
168
169 /// Cron Implementation Block Macro
170 #[proc_macro_attribute]
171 pub fn cron_impl(_att: TokenStream, code: TokenStream) -> TokenStream {
172     let item_impl = syn::parse::ItemImpl::parse(code.clone().unwrap());
173     let r#impl = item_impl.to_token_stream();
174     let impl_items = item_impl.items.clone();
175     let impl_type = item_impl.self_ty.to_token_stream();
176
177     let impl_type_upper = format_ident!(
178         "{}",
179         item_impl
180             .self_ty
181             .to_token_stream()
182             .to_string()
183             .to_uppercase()
184     );
185
186     let method_jobs = format_ident!
187     ("CRONFRAME_METHOD_JOBS_{impl_type_upper}");
188     let function_jobs = format_ident!
189     ("CRONFRAME_FUNCTION_JOBS_{impl_type_upper}");
190
191     let mut new_code = quote! {
192         #r#impl
193     };
194
195     let mut count = 0;
196     for item in impl_items {
197         let item_token = item.to_token_stream();
198         let item_fn_parsed = syn::parse::ItemFn::parse(item_token.into());
199         let item_fn_id = item_fn_parsed.clone().unwrap().sig.ident;
200         let helper = format_ident!("cron_helper_{}", item_fn_id);
201         let item_fn_id_upper = format_ident!(
202             "{}",
203             item_fn_id.to_token_stream().to_string().to_uppercase()
204         );
205         let linkme_deserialize = format_ident!("LINKME_{}_{count}",
item_fn_id_upper);

```

```

204
205     let new_code_tmp = if check_self(&item_fn_parsed) {
206         // method job
207         quote! {
208             #[cronframe::distributed_slice(#method_jobs)]
209             static #linkme_deserialize: fn(_self:
210                 std::sync::Arc<Box<dyn std::any::Any + Send + Sync>>) ->
211             cronframe::JobBuilder<'static> = #impl_type::#helper;
212         }
213     } else {
214         // function job
215         quote! {
216             #[cronframe::distributed_slice(#function_jobs)]
217             static #linkme_deserialize: fn() ->
218             cronframe::JobBuilder<'static> = #impl_type::#helper;
219         }
220     };
221
222     new_code.extend(new_code_tmp.into_iter());
223     count += 1;
224 }
225
226     let type_name = impl_type.to_string().to_uppercase();
227
228     let cf_fn_jobs_flag = format_ident!("CF_FN_JOBS_FLAG_{}", type_name);
229     let cf_fn_jobs_channels = format_ident!("CF_FN_JOBS_CHANNELS_{}", type_name);
230
231     let gather_fn = quote! {
232         impl #impl_type{
233             pub fn cf_gather_mt(&mut self, frame:
234                 std::sync::Arc<CronFrame>){
235                 cronframe::info!("Collecting Method Jobs from {}", #type_name);
236                 if !#method_jobs.is_empty(){
237                     let life_channels = cronframe::bounded(1);
238                     self.tx = Some(life_channels.0.clone());
239
240                     for method_job in #method_jobs {
241                         let job_builder = (method_job)
242                             (std::sync::Arc::new(Box::new(self.clone())));
243                         let mut cron_job = job_builder.build();
244                         cron_job.life_channels =
245                             Some(life_channels.clone());
246                         cronframe::info!("Found Method Job \"{}\" from
247                             {}, cron_job.name, #type_name);
248                         frame.clone().add_job(cron_job);
249                     }
250                     cronframe::info!("Method Jobs from {} Collected.",
251                         #type_name);
252                 } else {
253                     cronframe::info!("Not Method Jobs from {} has been

```

```

found.", #type_name);
246             }
247         }
248
249         pub fn cf_gather_fn(frame: std::sync::Arc<CronFrame>){
250             cronframe::info!("Collecting Function Jobs from {}",
251 #type_name);
252             if !#function_jobs.is_empty(){
253                 // collect jobs from associated functions only if
254                 this is the first
255                 // instance of this cron object to call the
256                 helper_gatherer function
257                 let fn_flag = *#cf_fn_jobs_flag.lock().unwrap();
258
259                 if !fn_flag {
260                     for function_job in #function_jobs {
261                         let job_builder = (function_job)();
262                         let mut cron_job = job_builder.build();
263                         cron_job.life_channels =
264                         Some(#cf_fn_jobs_channels.clone());
265                         cronframe::info!("Found Function Job \"{}\""
266                         from {}. ", cron_job.name, #type_name);
267                         frame.clone().add_job(cron_job);
268                     }
269                     cronframe::info!("Function Jobs from {}"
270 Collected.", #type_name);
271                     *#cf_fn_jobs_flag.lock().unwrap() = true;
272                 }
273             } else {
274                 cronframe::info!("Not Function Jobs from {} has been
275 found.", #type_name);
276             }
277         }
278
279         pub fn cf_gather(&mut self, frame:
280 std::sync::Arc<CronFrame>{
281             self.cf_gather_mt(frame.clone());
282             Self::cf_gather_fn(frame.clone());
283         }
284     }
285
286     new_code.extend(gather_fn.into_iter());
287     new_code.into()
288 }
289
290 /// Function Job definition Macro for a Cron Object
291 #[proc_macro_attribute]
292 pub fn fn_job(att: TokenStream, code: TokenStream) -> TokenStream {
293     let parsed = syn::parse::ItemFn(code.clone());
294
295     if check_self(&parsed) {
296         // self is present -> compilation error

```

```

290     }
291
292     // generate code for a function job
293     let args = parse_macro_input!(att with Punctuated::<Meta,
syn::Token![,]>::parse_terminated);
294
295     let args = args.into_iter().map(|x| {
296         x.require_name_value()
297         .map(|x| {
298             let arg_name = x.path.to_token_stream().to_string();
299             let arg_val = x.value.to_token_stream().to_string();
300             (arg_name, arg_val.replace("\\\"", ""))
301         })
302         .unwrap()
303     });
304
305     // should contain ("expr", "*****")
306     let (arg_1_name, cron_expr) =
args.clone().peekable().nth(0).unwrap();
307
308     // should contain ("timeout", "time in ms")
309     let (arg_2_name, timeout) = args.peekable().nth(1).unwrap();
310
311     if arg_1_name != "expr" && arg_2_name != "timeout" {
312         // wrong argument names -> compilation error
313         return code;
314     }
315
316     let origin_function = parsed.clone().unwrap().to_token_stream();
317     let ident = parsed.clone().unwrap().sig.ident;
318     let job_name = ident.to_string();
319     let helper = format_ident!("cron_helper_{}", ident);
320
321     let new_code = quote! {
322         // original function
323         #[allow(dead_code)]
324         #origin_function
325
326         fn #helper() -> cronframe::JobBuilder<'static> {
327             cronframe::JobBuilder::function_job(#job_name, Self::#ident,
328             #cron_expr, #timeout)
329         }
330     };
331     new_code.into()
332 }
333
334 /// Method Job definition Macro for a Cron Object
335 #[proc_macro_attribute]
336 pub fn mt_job(att: TokenStream, code: TokenStream) -> TokenStream {
337     let parsed = syn::parse::<ItemFn>(code.clone());
338
339     if !check_self(&parsed) {
340         // self is missing -> compilation error

```

```

340     }
341
342     // generate code for a function job
343     let args = parse_macro_input!(att with Punctuated::<Meta,
344 syn::Token![,]>::parse_terminated);
345
346     let args = args.into_iter().map(|x| {
347         x.require_name_value()
348         .map(|x| {
349             let arg_name = x.path.to_token_stream().to_string();
350             let arg_val = x.value.to_token_stream().to_string();
351             (arg_name, arg_val.replace("\\\"", ""))
352         })
353         .unwrap()
354     });
355
356     // should contain ("expr", "name of expression field")
357     let (arg_1_name, cron_expr) =
358         args.clone().peekable().nth(0).unwrap();
359
360     if arg_1_name != "expr" {
361         // wrong argument name -> compilation error
362     }
363
364     // generate code for a method job
365     let origin_method = parsed.clone().unwrap().to_token_stream();
366     let ident = parsed.clone().unwrap().sig.ident;
367     let job_name = ident.to_string();
368     let block = parsed.clone().unwrap().block;
369
370     let cronframe_method = format_ident!("cron_method_{}", ident);
371     let helper = format_ident!("cron_helper_{}", ident);
372     let expr = format_ident!("expr");
373     let tout = format_ident!("tout");
374
375     // this is to replace the native self with the self from cronframe
376     let block_string = block.clone().into_token_stream().to_string();
377     let mut block_string_edited = block_string.replace("self.",
378 "cronframe_self.");
379     block_string_edited.insert_str(
380         1,
381         "let cron_frame_instance = arg.clone();
382         let cronframe_self =
383 (*cron_frame_instance).downcast_ref::<Self>().unwrap();",
384     );
385
386     let block_edited: proc_macro2::TokenStream =
387         block_string_edited.parse().unwrap();
388
389     //println!("UNEDITED BLOCK:\n{block_string}");
390     //println!("EDITED BLOCK:\n{block_string_edited}");
391
392     let mut new_code = quote! {

```

```

388     // original method at the user's disposal
389     #[allow(dead_code)]
390     #origin_method
391
392     // cronjob method at cronframe's disposal
393     // fn cron_method_<name_of_method> ...
394     fn #cronframe_method(arg: std::sync::Arc<Box<dyn std::any::Any +
Send + Sync>>) #block_edited
395     };
396
397     let helper_code = quote! {
398         // fn cron_helper_<name_of_method> ...
399         fn #helper(arg: std::sync::Arc<Box<dyn std::any::Any + Send +
Sync>>) -> cronframe::JobBuilder<'static> {
400             let instance = arg.clone();
401             let this_obj = (*instance).downcast_ref::<Self>().unwrap();
402
403             let #expr = this_obj.cron_expr.expr();
404             let #tout = format!("{}", this_obj.cron_expr.timeout());
405             let instance = arg.clone();
406
407             cronframe::JobBuilder::method_job(#job_name,
408             Self::#cronframe_method, #expr.clone(), #tout, instance)
409         }
410     };
411
412     // replace the placeholder cron_expr with the name of the field
413     let helper_code_edited = helper_code
414         .clone()
415         .into_token_stream()
416         .to_string()
417         .replace("cron_expr", &cron_expr);
418     let block_edited: proc_macro2::TokenStream =
419     helper_code_edited.parse().unwrap();
420
421     new_code.extend(block_edited.into_iter());
422 }
423
424 // aid function for fn_job and mt_job
425 fn check_self(parsed: &Result<ItemFn, syn::Error>) -> bool {
426     if !parsed.clone().unwrap().sig.inputs.is_empty()
427         && parsed
428             .clone()
429             .unwrap()
430             .sig
431             .inputs
432             .first()
433             .unwrap()
434             .to_token_stream()
435             .to_string()
436             == "self"

```

```
437      {
438          true
439      } else {
440          false
441      }
442 }
```

## C - Dependencies List

1. **chrono v0.4.38**
  - <https://crates.io/crates/chrono/0.4.38>
2. **clap v4.5.15**
  - <https://crates.io/crates/clap/4.5.15>
3. **colored v2.1.0**
  - <https://crates.io/crates/colored/2.1.0>
4. **cron v0.12.1**
  - <https://crates.io/crates/cron/0.12.1>
5. **crossbeam-channel v0.5.12**
  - <https://crates.io/crates/crossbeam-channel/0.5.12>
6. **home v0.5.9**
  - <https://crates.io/crates/home/0.5.9>
7. **inventory v0.3.15**
  - <https://crates.io/crates/inventory/0.3.15>
8. **linkme v0.3.26**
  - <https://crates.io/crates/linkme/0.3.26>
9. **log v0.4.21**
  - <https://crates.io/crates/log/0.4.21>
10. **log4rs v1.3.0**
  - <https://crates.io/crates/log4rs/1.3.0>
11. **once\_cell v1.19.0**
  - [https://crates.io/crates/once\\_cell/1.19.0](https://crates.io/crates/once_cell/1.19.0)
12. **reqwest v0.12.5**
  - <https://crates.io/crates/reqwest/0.12.5>
13. **rocket v0.5.1**
  - <https://crates.io/crates/rocket/0.5.1>
14. **rocket\_dyn\_templates v0.2.0**
  - [https://crates.io/crates/rocket\\_dyn\\_templates/0.2.0](https://crates.io/crates/rocket_dyn_templates/0.2.0)
15. **serde\_json v1.0**
  - [https://crates.io/crates/serde\\_json](https://crates.io/crates/serde_json)
16. **toml v0.8.14**
  - <https://crates.io/crates/toml/0.8.14>
17. **uuid v1.8.0**
  - <https://crates.io/crates/uuid/1.8.0>