Diaries

Paolo Bettelini

Contents

1 Diaries

1.1 2022-08-29

Work hours:
13:15 - 14:45: Project assignment
15:00 - 16:30: Questions about the project / Rust libraries

Today I was assigned the project and received its description. I prepared some questions to ask the manager to clarify my ideas. I have decided to use only Rust as my main programming language, if possible also for the front-end application (compiling Rust to WebAssembly).

I've started to inquire about a couple of libraries that I (might) use

- amiquip Rust RabbitMQ client
- log Rust logger
- another-rust-load-balancer Rust load balancer
- warp Rust web framework
- sqlx Rust SQL Toolkit
- tokio Rust async library
- yew Rust/WASM frontend framework

I will use the Rust *nigthly* channel.

1.2 2022-08-30

Work hours:
08:20 - 09:50: Setup VM
10:05 - 11:35: Requirements and documentation

Today I set up a Linux virtual machine as my main workstation. I installed `cargo`, `texlive-core`, `texlive-latexextra` and `vs code`.
Each server will be on a different machine without GUI.

I then started writing the documentation (information about the project and the requirements). I still need to ask a few questions to the manager to finish the requirements.

The plan for the next work session is to make a Gantt sheet.

1.3 2022-08-31

Work hours:
15:00 - 15:45: Requirements and documentation
15:45 - 16:30: Gantt sheet

Today I finished writing the requirements of the project. I created the `worker` Rust project and I created the Gantt sheet.

1.4 2022-09-01

08:20 - 09:00: Helped a classmate with Android Studio
09:00 - 10:00: Vagrantfile with provisioning
10:00 - 11:00: Documentation
11:00 - 11:35: Use case

Today I created a virtual machine using Vagrant to host my servers. I spent most of the time setting up the document for the documentation and in the end I wrote the use cases (on paper). I will eventually translate them into a PDF.

The plan for the next work session is to start writing the database DDL and maybe start writing some queries.

1.5 2022-09-05

13:15 - 14:15: Internal network
14:15 - 16:15: Backend worker

Today I set up an internal network within the virtual machines (`192.168.1.0/24`). This is done via Vagrantfile and I made sure everything was working properly.

I started writing some actual code. After setting up the dependencies needed I implemented command line arguments. The Rust crate `Clap` manages my CLI arguments (errors if invalid args, help page). The only argument so far is the `--config <CONFIG>` or `-c` argument.

The config is a TOML file which so far contains the connection information about the database. My code parses the config file and initializes the config structure.

I haven't really written the SQL for the database as I should have, but I figured it shouldn't be an issue since not many tables are needed and they are quite trivial.

1.6 2022-09-06

08:25 - 09:50: Error handling and logging
10:05 - 11:20: Database connection

Today I handled the errors when reading the configuration files and implemented the logger. I then wrote the script to setup Mariadb used in the Vagrantfile (install, start daemon, create user). I spent the rest of the day trying to make the connection to the database.

However, I'm thinking about dropping the `sqlx` library and using an ORM instead.

1.7 2022-09-07

15:00 - 15:30: Vagrantfile
15:30 - 16:15: Diesel documentation

Today I upgraded the Vagrantfile and starting separating the different virtual machines. I decided to drop the `sqlx` library and use `Diesel` instead (ORM). I spent the rest of the work session looking at the Diesel documentation.

The plan for the next work session is to actually integrate this library and work with the database.

1.8 2022-09-08

08:45 - 10:35: Diesel installation and setup
10:50 - 11:20: Diesel integration

Today did not quite turn as I hoped it would. I started by trying to install `diesel_cli` (Diesel command line utility), which is used to generate migrations and schemas. The command `cargo install diesel_cli --no-default-features --features mysql` was not working and in the end I was able to install it via `pacman`. Diesel could not connect to the database (fixed user i Vagrant file). I then spent the rest of the day trying to integrate this library. The migration and schema generation was successful. However, I was having some problems with the Rust crate system and it took a long time to figure out how to make it compile without errors.

My goal for today was to execute some queries but I did not succeed.

1.9 2022-09-12

13:15 - 14:45: Project separation and restructuring
15:00 - 16:10: Diesel integration

Today I separated the *worker* into two separate projects: the database library and the executable. Then, I implemented and tested the database library by inserting a test user.

More specifically,

- Completed the SQL structure of the database
- Updated the database models
- Separated the two projects
- Migrations are run at startup if necessary
- If the database authentication is not specified in the config file, it is read from the `DATABASE_URL` enviroment variable
- Separated things into their own modules

I am slightly ahead of schedule. I haven't implemented every query and insert that I will need but they trivial especially since I am using an ORM library. Inserting data to the database and retriving it works.

1.10 2022-09-13

08:20 - 09:50: Read documentation
10:05 - 11:15: Vagrant

Today I read the amiquip documentation and their examples to communicate with a RabbitMQ server. In the second half of the work session I setup the RabbitMQ VM using Vagrant. The web dashboard works. I haven't yet started writing code to interact with queues.

1.11 2022-09-14

15:00 - 16:15: `amiquip`

Today I continued reading the `amiquip` documentation and created the messaging library project (`messaging`). Renamed the project `worker/worker` into `worker/core`.

1.12 2022-09-15

08:20 - 09:00: Vagrant
09:00 - 11:10: Implementation

I decided to drop the `amiquip` library and use lapin instead. It is a far more developed library and easily supports multi threading. I implemented the configuration for the connection to the RabbitMQ server and started testing the actual connection. My goal was to publish any payload to any channel, but unfortunately, it keeps giving me this error which I am unable to fix `IOError(Kind(ConnectionAborted))`.

1.13 2022-09-20

08:20 - 11:20: Pooling and connection

I tried fixing the connection aborted error with no luck. I decided to integrate a pooling system for the connections using deadpool and deadpool-lapin. The pool works and everything seems to be `async`. The connection aborted error is still thrown when a connecton is taken from the pool.

1.14 2022-09-21

15:00 - 15:30: Documentation
15:30 - 16:20: Research

Today I continued the documentation and fixed the problem with the connection to RabbitMQ. The program can establish a connection with the server, declare queues, publish messages and consume them. I feel ashamed to say, the problem was the service port (`5672` rather than `15672`, which is the web portal).

1.15 2022-09-22

08:20 - 9:50: Logging system
10:05 - 10:50: RabbitMQ test
10:50 - 11:25: Read documentation

Today I fixed the logging system (logging configuration file) and refactored some code. I continued testing the RabbitMQ connection (publishing messages and consuming them). I then spent the rest of the time by reading the documentation of `Yew` (frontend framework) and `Warp` (web framework). The next logical step is to implement the `Request/Reply` pattern in messaging.

1.16 2022-09-26

13:15 - 15:30: RabbitMQ Request/Reply
15:30 - 16:15: WebServer

Today I restructured the projects folders tree and created the webserver project.

- `common/`
  - `messaging/`
  - `database/`
- `worker/`
- `webserver/`

I tried to implement the request/reply pattern with RabbitMQ but it does not work yet. The idea is to set the `correlation_id` property of the messages that are published with a random UUID. Then, wait for a

reply in the `reply_to` queue with the same correlation ID. I spent the rest of the time looking at how to make a WebSocket between `yew` and `warp`.

1.17 2022-09-28

15:15 - 16:10: RabbitMQ Request/Reply

The `Request/Reply` pattern works. The only thing left to do with my messaging API is to structure the functions in order to make it usable and more generic. The basic operations needed are

- Publish message
- Publish message and await response
- Publish message and await multiple responses

1.18 2022-09-29

08:20 - 09:00: Rust Async
09:00 - 11:10: Messaging and structure

Today I read a tutorial about the `async` features of Rust. I then proceeded to structure my messaging API to make it usable. The functions created are: `publish`, `public_and_await_reply` and `consume_messages`. Throughout this work session I also cleaned the code, fixed the formatting, separated things into their own containers and such.
The next step is to start implementing the web page using `yew`.

1.19 2022-10-04

08:20 - 09:50: Yew testing
10:05 - 11:05: Websocket research
11:05 - 11:20: Documentation

Today I started implementing the frontend using WebAssembly. I was able to create a test page using the Yew framework that print "Hello" (a string that is not written in HTML but rather embedded in the wasm).

I then wanted to start testing a websocket connection, however, I found that that the websocket layer in the Yew framework has been completly removed from version `0.18.0` to `0.19.0` (I am using version `0.19.0`). This implies that I will need to use another library for the websocket connection. The desirable thing would be to use the same library both for the browser and for the webserver, such that the binary interpretation of the messages uses the same structures. If this cannot be achieve I need to use another external library to interpret the binary messages between the endpoints. I haven't yet decided which technology to use.

I also added the `implement/frontend` subsection in the documentation.

1.20 2022-10-05

15:00 - 16:15: Documentation

Today I only focused on the documentation. I separated the various sections into their own files, implemented the reference system, refactored some code and added some sections. I haven't really added any content worth mentioning.

1.21 2022-10-06

08:30 - 11:20: Web application

Today I continued to refactor the web application code. I separated the frontend and webserver into their own projects (`webapp/` is a cargo workspace containing `frontend/` and `webserver/`). I set up the log and config system for the webserver.

1.22 2022-10-10

13:15 - 16:15: Frontend

Today I focused on making the frontend work. The frontend is separated from the webserver. The webserver serves the index, the index then sends a request to the frontend server to ask for its content. I followed this tutorial.
For the next working session I'm going to start implementing the RabbitMQ messages, and if possible make the RabbitMQ cluster.

1.23 2022-10-11

08:20 - 09:50: Read documentation
10:05 - 11:20: RabbitMQ messages

Today I read the documentation of various Rust crates. In the second half of the working session I created the structures that represent the messages sent over the RabbitMQ network. I used the protocol crate to create structures that can be serialized and deserialized into binary data. All the structures used in the protocol are defined in the project `common/protocol`.

1.24 2022-10-12

15:15 - 16:15: Messaging test

Today I tested the messaging system by serializing my payloads into binary data, putting them in a RabbitMQ queue and consuming them. Everything works fine. I also refactored some code.

1.25 2022-10-13

08:20 - 09:50: RabbitMQ Cluster
10:05 - 11:20: Code and config

Today I read the documentation about RabbitMQ clusters. I spoke with my supervisor and we agreed on slightly changing the infrastructure of the network. The backend servers connect to a load balancer instead of multiple messaging brokers. The webservers also connect to the same load balancer which is behind every message broker. I then continued to add some boilerplate to my code.

1.26 2022-10-17

13:15 - 14:45 Design
15:00 - 16:15 Logic

Today I started by designing a bit of the website (Login Form, Register Form). In the second half of the working session I continued the logic of my program, refactored some code and implemented the image resizing.

1.27 2022-10-18

08:20 - 11:20 Logic

Today I only focused on implementing the logic for the backend worker. I had lots of problems with the Rust syntax but I managed to advance the code a bit. The backend server is now able to consume a `Register Request` message. I still have problems with the Rust borrow checker and don't know how to fix them.

1.28 2022-10-19

15:00 - 15:30 Logic
16:00 - 16:20 Logic

Today I kept implementing the backend logic. The backend is able to handle a `RequestLogin` message. I'm working on handling the `GetTotalImages` (amount) message.

1.29 2022-10-20

08:20 - 09:50 Logic
10:05 - 11:15 Logic

Even today I continued the logic for the worker backend. I implemented the necessary database queries with Diesel and continued the code which consumes messages. This section is almost done and I should finish it within the next working session.

1.30 2022-10-24

15:00 - 15:30 Frontend WASM
16:00 - 16:20 Frontend WASM

I dropped the Yew framework because I realized that I couldn't do the website like I wanted to. Not using a framework such as Yew means that I have to separate HTML templating and WASM execution. The `frontend` crate compiles to a WASM module. The `frontend/website` contains the HTML website with the `webpack` files. By using `npm` and `webpack` I can include the WASM module into the HTML files. I tested a WebSocket connection and everything works fine.

1.31 2022-10-25

08:20 - 09:50 Webserver
10:05 - 11:15 Webserver

Today I implemented the templating system using `warp`. I am able to serve static website files from a directory and override some of the files (such as `login.html`, `index.html`) with the server template rendering. I am still having for problems with Rust lifetimes.

1.32 2022-10-27

08:20 - 09:50 Documentation
10:05 - 11:20 Documentation

Today I wrote some documentation.

- `Infrastructure` Added network diagram
- `Technologies` Added Rust and RabbitMQ
- `Implementation/Messaging/Messages` Started writing the message protocol

1.33 2022-11-07

13:15 - 14:45 Webserver routes
15:00 - 16:15 Frontend

Today I worked on the frontend website and the webserver.
The webserver now supports the following routes

- **/** → Serve index page
- **/register** → Serve register page
- **/login** → Serve login page
- **/logout** → Serve logout page
- **/upload** → Serve upload page
- **/api/register** → Register action
- **/api/login** → Login action
- **/api/logout** → Logout action
- **/<file>** → Serve static file
- **/index.html** → Block action
- **/register.html** → Block action
- **/login.html** → Block action
- **/logout.html** → Block action
- **/upload.html** → Block action

I created the `register`, `index`, `login`, `logout` and `upload` pages. Every page has a navbar working correctly and the `login` and `register` pages have forms with appropriate validation. The interaction between the pages works. The HTML is templated using the tera library.
The passwords are hashed client-side using WebAssembly (`sha256` + `base64`).

The actual communication **webserver** ⟷ **backend** is not done, so the response to login and register actions are hardcoded. If the response is negative the register or login page reloads and displays the error.

The image upload feature is yet to be implemented.

1.34 2022-11-08

08:20 - 09:50 Documentation
10:05 - 11:20 Multipart form

In the first half of the working session I continued the documentation, more specifically the `Implementation` section. In the remaining time I started implementing the multipart form for the upload feature. I read online documentation and tutorials and also fixed some code.

1.35 2022-11-09

15:10 - 16:20 Dropzone

Today I started implementing the dropzone feature. The dropzone element in the website uses the dropzone.js library. I created the `/api/upload` route and made sure the upload was successful.

1.36 2022-11-10

08:20 - 08:50 Fixed CSS
08:50 - 09:20 Helped a classman
09:20 - 11:25 Upload feature

Today I continued implementing the upload feature. When files are dropped into the dropzone the progress for each file is shown. So far only the upload progress to the webserver is displayed.

1.37 2022-11-14

13:15 - 15:30 RabbitMQ Request/Reply
15:30 - 16:15 Database queries

Today I implemented the Request/Reply pattern using RabbitMQ. A client can now successfully publish a message and await the response from a consumer. When a *register* or *login* form is sent, the user is actually inserted into or selected from the database.

For the next working session I need to fix the auth token management. The token is not saved yet in the database. The next stpe is to start sending images to the backend.

1.38 2022-11-15

08:20 - 09:50 Database
10:05 - 11:29 Documentation

Today I implemented the autentication token in the database and fixed a couple of bugs. The login system is now fully working. I then documented the messaging request/reply pattern in RabbitMQ (`Implementation.Messaging.Reque` `Pattern` section).

1.39 2022-11-16

15:10 - 16:20 Read documentation

Today I read the documentation about `Stream` in `warp::filters::multipart::FormData` and `futures::stream`. I need this to process the images on the webserver.
The goal for the next working session is to successfully store images in the database after sending them over to the backend.

1.40 2022-11-17

08:20 - 09:50 VM Recovery
10:05 - 11:25 Server logic

Today my Virtual Machine got corrupted. I spend the first half of the working session recovering it. I updated the system, fixed the pacman keyring, installed every tool needed, cloned the repository and compiled everything. In the second half of the working session I continued the server logic. I am having troubles sending the image to the backend server.

1.41 2022-11-21

13:15 - 16:20 Implementation

Today I implemented the image upload functionality. The images are sent over to the backend and saved in the database. The images are shrinked (lanczos3) and converted to a specific file format. Every user has its own images and the API serves them `/api/image/<id>`. I also fixed the multithreading on the webserver. The only feature left to implement, besides refactoring some code and completing the website, is the multithreading on the backend.

1.42 2022-11-22

08:20 - 11:25 Documentation

Today I continued the documentation. I wrote the following sections:

- `Implementation.Frontend.Generating WebAssembly`
- `Implementation.Frontend.Importing the module`
- `Implementation.Webserver.Routing with warp`
- `Implementation.Database.Diesel`

1.43 2022-11-23

15:10 - 15:30 Website
15:30 - 16:00 Helped a classmate
16:00 - 16:20 Multithreading

Today I added the "load more" button in the gallery. Images are now loaded 5 at a time. I then tried to implement multithreading on the backend. The idea is to use an `Arc<RwLock<MessageConsumer>>` such that threads can share the same reference simultaneously, but there is a lock on `mut` (for the database access).

1.44 2022-11-24

08:20 - 11:10 Parallelism

I was able to make the database usable by multiple threads at once. Previosuly, I was working with `&mut MysqlConnection`. The mutability allowed only one usage at a time.
I implemented a `Pool<ConnectionManager<MysqlConnection>>` which does not need mutability and can produce `&mut MysqlConnection`. This allowed me to remove the mutability requirement throughout the database code. I then tried to start multiple message-consuming threads, however I couldn't.

1.45 2022-11-28

15:00 - 16:15 Refactor

Today I spent the whole time refactoring code, fixing warnings and such. I realized that the final source code will not be as tidy and organized as I would have wanted to, since I don't have the Rust knowledge to do so, althought it's a problem that mainly affects the `webserver`. The code itself is almost done.