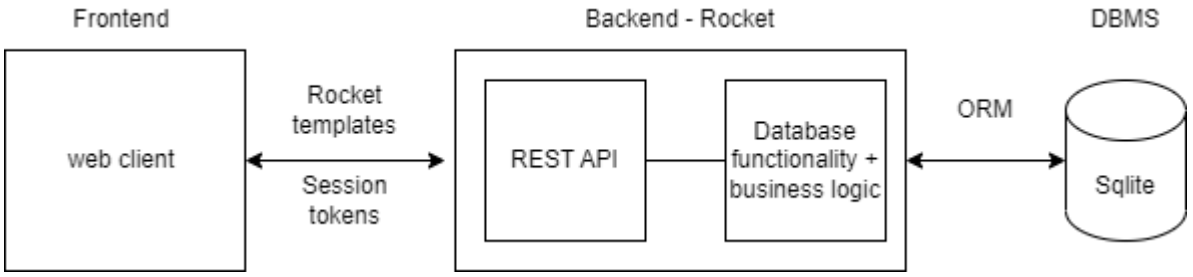
Final report for the semestral project

Pavol Hradský, Matej Lánik, Tomáš Both

Project idea:
The main idea of the project was to create a full stack web app, which will function as a small blogging site and we were using Rust as the main technology. Our blogging system allows users to add new articles (including a main body with unlimited length text, headline and a picture), display articles in detail, and browse articles. The blogging system allows new users to register, log in and log out. We also implemented a proof of concept method for 2 factor authentication, but now it functions essentially as a second password. Our solution uses Rocket framework, sea-orm for database access and Tera templates for frontend.

High level architecture diagram:



Dependencies and design choices:

- **rocket v0.5.0-rc.3** - Rocket framework is the main technology behind our project, we chose it because it is one of the most well-known crates for backend projects in rust and the newest version supports async mode. Alternatives we could have chosen were Axum and Actix.
- **serde v1.0.160** - this crate is for serializing and deserializing json objects to structs. We use it to convert request bodies to Rust structures.
- **rocket_dyn_templates v0.1.0-rc.3** - the project's frontend is implemented using Tera templates. Since this project was about learning Rust, we chose to keep this part of the application relatively straightforward and focus on other parts of the project. The alternatives were using a Javascript-based frontend or the Yew framework.
- **sea-orm v0.11.3** - this create is used for interacting with the database layer. We chose this crate since it supports async and is compatible with Sqlite, our DBMS of choice. The main alternative we considered was Diesel, but by default it does not support async.

Besides the crates mentioned above, we also use these crates dependencies indirectly - there are many of them, but this list includes only the main ones.

Code structure:
In the src folder, we have following files:

- main.rs - setting up db,mounting rocket and defining routes
- api_endpoints.rs - backend endpoints
- templates.rs - endpoints for templates
- sqlite.db - user and article database
- database folder - database schema
- templates - folder with templates
- static - folder with css styling

Evaluation:
We think the implementation of the project went well, although there were some occasional obstacles that we had to overcome - e.g connecting frontend to backend and figure out how to store pictures, how to run queries against the database or how to implement MFA - we could not successfully include the google_authentication crate in our project, this is why we chose the proof-of-concept MFA instead.
We think the project went well overall and we consulted it every week with the course organizers and the outcome is a usable project. We learnt how to start a project and use new libraries / crates in a new language - Rust. Working with Rust was quite challenging but also fun, compared to other languages the compiler is more strict, but also very helpful sometimes.

Compared to the project proposal, we managed to implement the following things:

- Users are able to read blog posts and choose blog posts from an aggregated list, similar to popular blogging platforms
- Users are able to login in, create accounts and add blog posts
- Blog posts contain a title, text divided into sections, pictures and their author
- All user data and blog posts are stored in an SQL DBMS
- Proof-of-concept MFA solution

Our project goal was to get more familiar with Rust and attempt to use a few popular crates. This project gave us a small outlook on what developing an application in Rust looks like. We've encountered a few problems that together we were able to solve(dependency issues, connecting a DB).

Eventually, we have managed to successfully complete all the tasks we proposed in the beginning. The only part of the project that wasn't completed as originally planned was the multi-factor authentication, but we managed to make a compromise solution instead. Even though this MFA solution didn't go as planned, we made ourselves familiar with a few crates and we also spent some time solving a problem, which in the end isn't a bad thing.

Possible improvements of the project in the future could be:

- Improve security by using login tokens, hashing password and encrypting all the data transmitted

- Divide the users into visitors, editors and admins, while giving each one of them different rights.
- Solve all of the critical situations that might happen (users choosing weak passwords, entering wrong email addresses, entering image format that isn’t supported, etc.)