SyncScheduler

Semen Pobrodilin ID: 116973
Mariia Portnykh ID: 127253

Introduction to the idea

Project idea:

Smart planner. A calendar app capable of analyzing your schedule and suggesting optimal dates and times for your events. This app will be especially useful for students or people with busy schedules.

Against regular schedulers, where you can just put events on specific time slots, our app has smart scheduling.

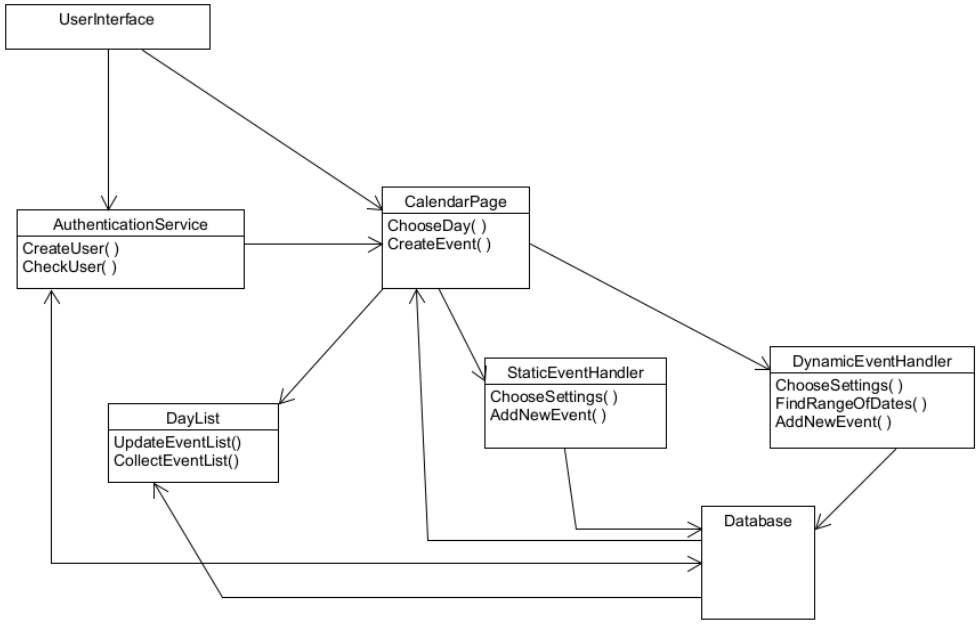
Smart scheduling: when creating a dynamic event, the user specifies their wishes, such as specific days, times and even the duration of the meeting. Based on the user's wishes, the app will analyses the user's schedule and suggest suitable dates and time slots for the user's request.

Requirements

Some of the requirements have changes from the initial idea, cause troubles with implementing solution or lack of time for solving GUI problems for that.

1. User must be able to schedule events on any time that is not conflicted, and look through all of the events on the day.
2. The user must be able to schedule two types of events: static and dynamic events.
3. Ability to schedule an event in 2 ways: via calendar, when the user searches for this or that date by flipping the calendar, or via dynamic window, so that the user does not have to flip the calendar manually, just set the date in the format 02.04.2025 in the window and all the data to the event and it will be automatically added to the calendar.
4. Optimized date suggestion taking into account the conditions set by the user, and checking all of his events
5. Calendar makes its impossible to have conflicts and they are solved on database level
6. Calendar saves your information online and can work on different devices, just by login and password

Design diagram.



Design choices

Most of the Calendar solutions are made by Slint and standard Chrono Library. Chrono is used for all the time and date selectors, and all the renders of the screen are Slint.

We changed some of our ideas towards less GUI working. However, Design of regular calendar is done manually rendering with own logic for rendering amount of days in month, and leap years. Slint is used 90 percent of time just for the rendering, and all the logic is on rust (except the logic of components that are just lacking implementation already on Slint. Same with Databases, we used SQLX library, but querries are small and easy and logic for the events are saved to do on rust: all collision checking, all finding time slots, all updating and inserting.

We decided not to render Week calendar view because it did not do much for the logic and idea except adding just more problems with GUI and rendering.

Dependencies and what they're used for

SQLx + postgres for database of events, also Uuid for working with unique IDs for events and users. Slint for all UI. Chrono for time and date formats with functions. Tokio for I/O and asynchronous functionality.

Evaluation

Rust is functionally good, but is not comfortable with GUI and lacks some good polished libraries. We used Slint for the entire GUI and Sqlx for databases, we stumbled across major problems where we did not expected all over the project, even bugs in libraries and a lot of time was spent on fixing GUI problems instead of actual logic.

However, we learnt quite a lot while fixing them, and finding solutions or workarounds.

If we will consider doing project that would surely not be heavily dependent on UI, or we could use Rust just for backend, then Rust is truly good language with many pluses over other languages (Especially pure C). For example work on Databases with SQLx was comfortable, and no workaround were done to make a simple queries.

But also one of the problems we found that Rust lacks some good, tested and polished libraries, for example Diesel for databases library just did not worked on Window 11 because of problems with environmental variables, and had a lot of issues that were harder to fix then to write actual logic. Because of that, we changed databases from Diesel to SQLX and it was one of the best decisions. While working on Sling we found some of bugs just in a library, and we went too far to change GUI, so we just pushed through them.

Rust is good but need refinements and good documentations. Because overall things that we expected to be hard was not as hard (also thanks to Rust), but things we expected to do fast and without problem caused more problems and sometimes consumed major amount of times. Sometimes even more time then we estimated we would loose on major problems.