Software requirements specifications

Illia Shust, Azim, Alim, Hamid

01.03.2025

This document provides important software specifications for our website, called Campus Scheduler. It describes our business logic, development choices we've agreed on; software process model; functional and non-functional requirements; as well as legal aspects of operating in the EU with strong GDPR laws

1 Overall description

The project will be a web-based platform, hosted on Railway. It will feature a role-based interface (Admin, Manager – Team Leader, and Worker). The website will communicate with the PostgreSQL database to store userdata. A user will have an option to sign in using Google authentication, or register themselves. Additionally, they can choose to retract their data by sending a request to our email address.

2 Business logic

This system can be beneficial to businesses to desire to manage their workflows using role-based interfaces. It enables quick access to project-management, adding events, assigning tasks, communication between different teams and many more. Small to medium enterprises (SMEs), as well as Startups and Tech companies that seek cost-effective solutions can take advantage of our rapid and iterative development, where client needs can be addresses quickly

We plan to make our web-based platform free to use, until we grow a large enough user base to begin thinking about charging for our services. In that case, a subscription based model could be implemented.

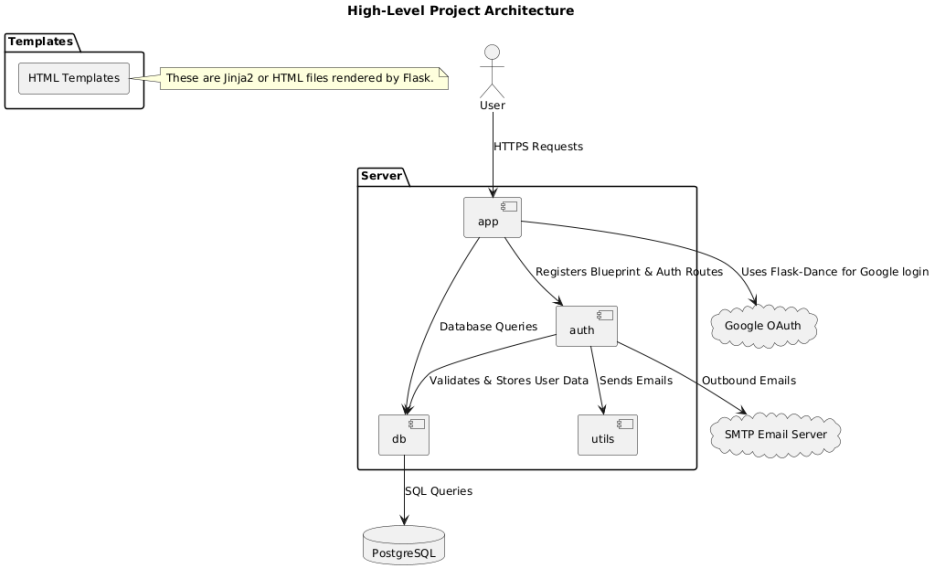

Figure 1: Project overview

3 UML diagrams

App: entry point and core of the application. It manages the session, handles various HTTPS requests from the user, provides functionalities for logging in with Google, managing projects and tasks, handling events, and rendering pages such as the home (index), about and privacy pages.

Auth: focuses on user authentication and account management. Provides methods for user login, registration. It verifies user by confirming email, provides an option to reset password etc.

DB: This abstracts the database operations required by the system. Provides methods to connect to the database and close the connection safely. It's a bridge between the application and PostgreSQL database.

Utils: various utility functions that support the main application. Currently includes methods for sending emails. May expand in the future as more features come.

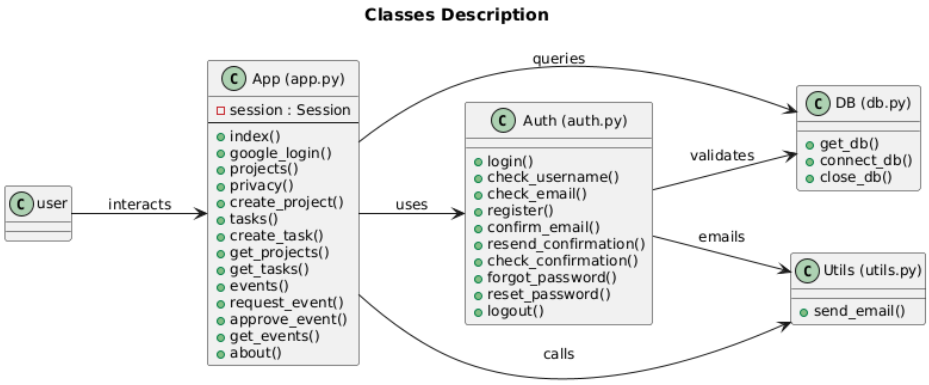

Figure 2: Class description diagram

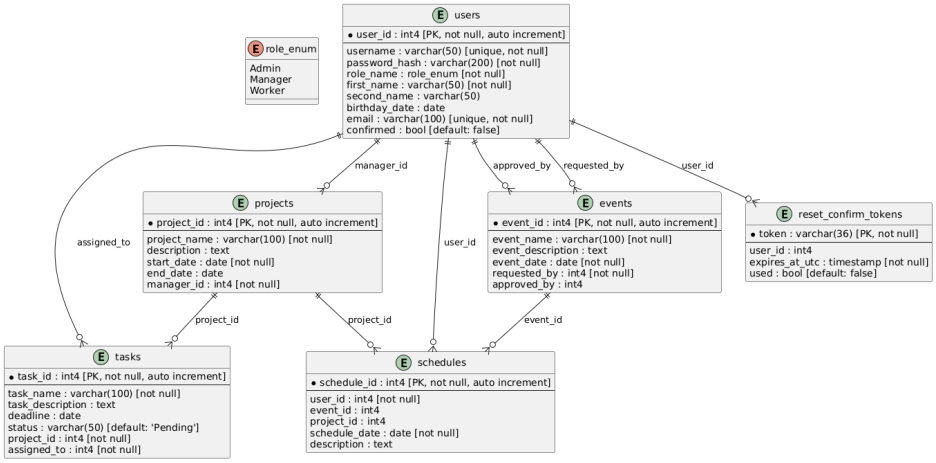

Figure 3: Database UML diagram

3.1 Database UML diagram

Currently, we have three roles (Admin, Manager, Worker). This is expressed in role_enum.

"**Users**" table stores relevant user information. That includes unique user_id, username, hashed password for security purposes, role, personal details (first name, second name, date of birth), email, and an indication if email has been confirmed. That indication can be used in the future to allow access to certain features (like admin dashboard)

"**Projects**" table has a unique project_id, project's name and description, start and end dates, and manager_id which is linked to the user managing the project via a foreign key.

"**Tasks**" table has a unique task_id, task name and description, deadline and status (defaulting to "Pending"), project_id linking task to the specific project via foreign key, assigned_to linking to the user responsible for the task via foreign key.

"**Events**" table has a unique event_id, event name and description, a date when the event happens, the user who requested an event, an optional field for the user who approved

an event (Admin or Manager in this case). It has two foreign keys linking to requested and approved by users.

"**Schedules**" table has a unique schedule_id, user_id (the person involved). Optionally, an event_id or project_id if the schedule entry is linked to one of those. It also features the date for the schedule entry and and optional description.

"**Reset_confirm_tokens**" table manages tokens needed for email confirmation and reset requests. Each token includes a unique token, a user_id it belongs to, an expiration timestamp to limit how long the token is valid for, and a flag to mark whether the token has been used or not.

4 Software process

We have chosen the Rapid Application Development (RAD) model for our system. This enables us to quickly prototype and iteratively deliver on the website instead of extensive upfront planning. This is a good choice for us as many members of our team are off-campus, thus meeting and team collaboration can be limited in some situations. It speeds up development, especially useful in situations where software requirements can change quickly, or simply not given to the full extent.

4.1 Development choices

Fronted: HTML, CSS, Javascript. Simple choice that allows all of our members to contribute.

Backend: Python (Flask), Java, and REST API. Same justification. Simple to implement business logic, flexible, allows for rapid development and each member can contribute. Flask allows the use of templates, this means the backend can render pages that already contain user data (like user's name or role). Javascript in the frontend can further enhance user experience by adding validation for passwords, dates, usernames etc. REST API is used along with Flask endpoints to generalize the development process.

Database: PostgreSQL. Modern choice for a database. Offers robust performance and data integrity, as well as some amazing features (complex queries, custom functions, cron jobs, custom return types such as JSON). Being open source means it has no licensing costs, and there's vast community support in case it's needed.

5 Specific requirements

Functional requirements (features, client interaction)

ID	Requirement Description
FR1	Users can register and log in.
FR2	System assigns each user a role (Admin, Manager-Team Leader, Worker).
FR3	Managers can create projects, assign tasks, and set priorities/deadlines.
FR4	Managers oversee tasks for their team and track progress.
FR5	Employees can view, update, and mark tasks as completed.
FR6	**Real-time messaging** feature, with respect to roles.
FR7	**A role-based** dashboard that shows relevant data (projects, tasks, team info).
FR8	**A project dependency graph** for visualizing task connections between different projects.
FR9	Managers can view a calendar of tasks for their teams.
FR10	Employees can view a personal task calendar.
FR11	Workers can communicate with their colleagues and team leaders.
FR12	Workers must not be able to message team leads outside their own team.

Non-functional requirements (scaling, reliability, security, constraints)

ID	Requirement Description
NFR1	The system should handle at least 100 concurrent users without performance degradation.
NFR2	The graph interface should render updates in under 1 second.
NFR3	The UI must be responsive and function well on both desktop and mobile screens.
NFR4	The system should use **OAuth 2.0** (e.g., Google Sign-In) for quick and easy authentication.
NFR5	The database must have backups to prevent user data loss
NFR6	The website must score a high speed test (90 and higher) on google page speed test.
NFR7	The website must use **HTTPS**.

5.1 Legal aspects

Data privacy: all user data is collected and stored with accordance with European Laws. The privacy policy is explained on the privacy page. Users can choose to have their data deleted by submitting a request.

Data ownership: all projects and their relevant data belongs to the respective owners (Managers who oversee projects and users who do the tasks)

Security: The system implements industry standard security measures (the use of SSL/TLS for data transport, password hashing, prevention of SQL injection, data validation,

database backup etc). Security tests may be required in the future to make sure the system is fully resistant to attacks.