Technical report

This technical report describes a web application developed using the technologies HTML, CSS, JavaScript, Django, FastAPI, SQLAlchemy, and PostgreSQL. The purpose of the report is to document the user stories, the technical decisions made, and the design of the web services that make up the application.

User stories

As a user, I want to...

- **Register on the application** to create an account and access the application's functionalities.
- **Authenticate on the application** using my username and password to access my profile and the application's functionalities.
- **Edit my profile** to update my personal information, such as name, profile picture, and biography.
- **Create a post** to share content with other users.
- **Edit a post** that I have created to modify its content.
- **Delete a post** that I have created.
- **View other users' posts** in a feed or list.
- **Like a post** to indicate that I like the content.
- **Comment on a post** to share my opinion or add additional information.
- **View the comments on a post** to read the opinions of other users.
- **Follow other users** to see their posts in my feed.
- **Unfollow a user** to stop seeing their posts in my feed.
- **Search for users and posts** by different criteria, such as username, post title, or post content.

Technical Decisions

The choice of technologies for the development of the application was based on the following criteria.

- **HTML and CSS:** They were used to define the structure and visual style of the user interface.
- **JavaScript:** It was used to add interactivity to the user interface and to handle client-side logic.
- **Django:** It was used as a web framework for the development of the application's backend. Django offers a robust and scalable architecture, as well as a complete set of tools for web development.
- **FastAPI:** It was used as a web framework for creating RESTful APIs. FastAPI is a lightweight and high-performance framework that makes it easy to create APIs with Python.
- **SQLAlchemy:** It was used as an ORM (Object Relational Mapper) to map Python objects to PostgreSQL database tables. SQLAlchemy makes it easy to work with relational databases from Python applications.
- **PostgreSQL:** It was used as a database to store the application's data. PostgreSQL is an open-source relational database that offers high availability, scalability, and security.

Web services

The API provides functionalities for creating, authenticating, and retrieving user information, as well as creating, retrieving, commenting on, and managing posts, following and unfollowing other users, and searching for users.

Features

The API offers the following services:

Users

- **Create user (POST):** Allows registering a new user in the system. Requires a UserModel object in the request body.

- **Authenticate user (GET):** Validates the credentials of an existing user. Requires a UserCredentials object in the request body.
- **Get information (GET):** Retrieves the basic information of an existing user. Requires the user ID as a parameter in the URL.
- **Edit profile (PUT):** Allows updating the profile information of an existing user. Requires a UserModel object with the updated data in the request body.

Content

- **Create post (POST):** Allows creating a new post in the system. Requires a PostModel object with the post content and the ID of the authoring user in the request body.
- **Get posts (GET):** Retrieves a list of published posts in the system. They can be filtered by user or by date.
- **Get comments (GET):** Retrieves a list of comments associated with a specific post. Requires the post ID as a parameter in the URL.
- **Create comment (POST):** Allows creating a new comment on an existing post. Requires a CommentModel object with the comment content, the post ID, and the ID of the authoring user in the request body.

Interactions

- **Follow (PUT):** Allows a user to follow another user. Requires the ID of the user to follow as a parameter in the URL.
- **Unfollow (DELETE):** Allows a user to unfollow another user. Requires the ID of the user to unfollow as a parameter in the URL.
- **Get followed (GET):** Retrieves a list of users followed by a specific user. Requires the user ID as a parameter in the URL.
- **Get followers (GET):** Retrieves a list of users who follow a specific user. Requires the user ID as a parameter in the URL.

Search

- **Get searched (GET):** Allows searching for users by name or username. Requires the search term as a parameter in the URL.