System Design Document

Quote Wall

Table of Contents

CRC CARDS:

Quote Wall only utilizes classes in order to create tables within a database, because of this the CRC Cards listed below are altered to fit our software.

Class Name: User	
Parent Class: N/A Subclasses: Genres, Preferences, Favourites	
Responsibilities: ● Store user information ○ Email ○ Password ○ Name	Collaborators: N/A

Class Name: Genres	
Parent Class: User (Connected through foreign key) Subclasses: N/A	
Responsibilities: ● Store a array of users selected genres	Collaborators: N/A

Class Name: Preferences	
Parent Class: User (Connected through foreign key) Subclasses: N/A	
Responsibilities: ● Store user preferences ○ Text Size ○ Quote Delay ○ Light/Dark Mode ○ Ainmations	Collaborators: N/A

Class Name: Favourites	
Parent Class: User (Connected through foreign key) Subclasses: N/A	
Responsibilities: ● Store users selected favourite quotes	Collaborators: N/A

System Interaction:

Quote Wall is a full-stack web application that is built using Next.js (React) and Flask (Python). The website's frontend is capable of running on any machine capable of downloading and running Node.js (version 18+) and the website's backend simply relies on any version of Python above 3.9. It is also assumed that any machine attempting to run the program can make HTTP requests, allowing it to operate on Windows, Mac, and Linux.

Database and Backend: Quote Wall's database is hosted on Supabase, which uses PostgreSQL. Database interactions are handled using SQLAlchemy and implemented with Flask-SQLAlchemy. The backend environment is managed using a Python virtual environment (.venv), all dependencies for the environment are listed in a requirements.txt file.

Frontend and API: The front and backend communicate through REST API calls enabled with the use of Flask-CORS. All other outside API requests are handled with the requests library. Because of this an internet connection is required to run the application. All frontend dependencies are listed and managed in package.json. With the dependencies listed here as well as those listed in requirements.txt the application is capable of being run.

System Architecture:

Frontend (Next.js & React)

The system's frontend runs in the user's web browser. Its UI takes care of all user interactions:

- User authentication (NextAuth)
- Theme changes and accessibility options
- Display quotes

The frontend also sends API requests to the backend, these are used for:

- Gathering user preferences
- Storing new and updated user preferences
- Fetching new quotes

Backend (Flask & SQLAlchemy)

The backend handels the passing of information from the frontend to the database. The backend also:

- Handles REST API requests from the frontend
- Interact with the database with SQLAlchemy
    - Uses SQLAlchemy ORM to both store and retrieve data
- Process and handle all imputed logic
    - Verifying user preferences are valid
- External API requests
    - Allows the system to pull quotes from Quote API

The backend also helps to:

- Ensure secure password handling (bcrypt)
- CORS to secure API access
- Handle basic errors and simple validation

Database (Supabase)

The database stores all user preferences and account information. It is stored on Supabase and accessed using PostgreSQL. The information it stores includes:

- email, password, name
- Preferred genres

- Preferred text size, presentation delays, website themes, animations
- Favourite quotes
- Past 10 quotes
    - This is done to prevent the same quote from showing up repeatedly

System Decomposition:

Error Handling:

The functions responsible for interacting with the database may encounter a number of possible errors pertaining to incorrect data and exceptional cases. Here are some generalized cases of such and how we have handled them:

- The user tries to input an email that does not exist in the database:
    - In these cases, we do not execute the function and instead return an error message with an HTML error code 404.
- The user tries to sign up with an email that already exists:
    - In this case, we reject the user's attempt to create a new account and return an error message with an HTML error code 400.
- Attempting to create an entry for 'GENRES', 'PREFERENCES', or 'FAVORITES' when the referencing email does not exist as a primary key in 'USER'.
    - In this case, we immediately terminate the function and return an HTML error code 400.
- Attempting to log in with an email that is not associated with any user in the database.
    - Validate the email the user entered to verify whether or not that email exists as a user in the database. If it is not, return an error message and an HTML error code 401. The user is not logged in
- Attempting to log in with a valid email, but with the wrong password.
    - Validate the email the user entered to verify whether or not that email exists as a user in the database. If the user exists, then cross-check the password the user entered and see if it matches the password associated with that email in 'USER'. If it does not, return an error message and an HTML error code 401. The user is not logged in.