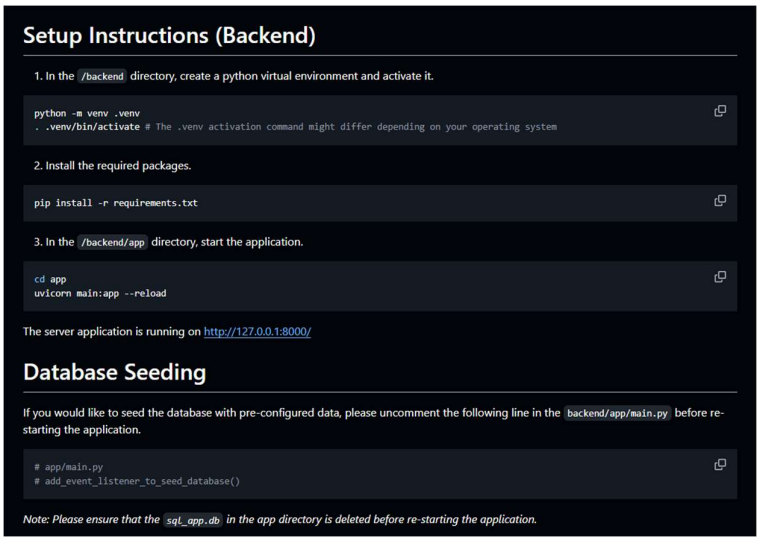
Documentations

Table of Contents

README

Our repository contains detailed README files for both the frontend and backend modules. These documents serve as essential setup and development guides for future developers, promoting consistency in code practices and supporting ongoing maintenance efforts. The images illustrate selected content from these documents. The complete README files can be found on our GitHub repositories for the main, frontend, and backend components.



Setup Instructions (Backend)

1. In the `/backend` directory, create a python virtual environment and activate it.

```
python -m venv .venv
. .venv/bin/activate # The .venv activation command might differ depending on your operating system
```

2. Install the required packages.

```
pip install -r requirements.txt
```

3. In the `/backend/app` directory, start the application.

```
cd app
uvicorn main:app --reload
```

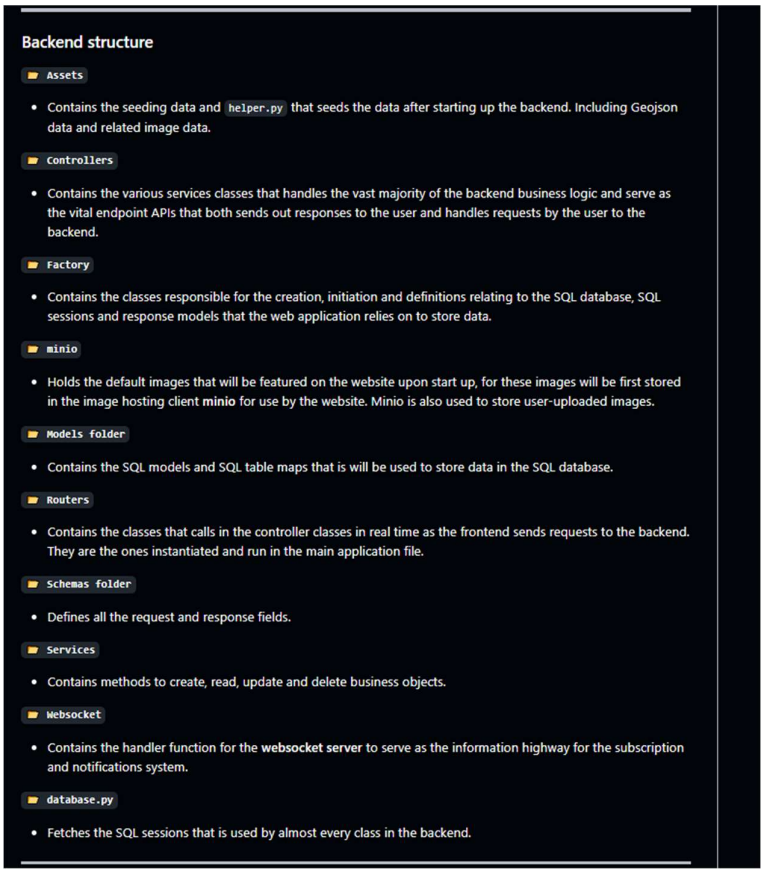
The server application is running on http://127.0.0.1:8000/

Database Seeding

If you would like to seed the database with pre-configured data, please uncomment the following line in the `backend/app/main.py` before re-starting the application.

```
# app/main.py
# add_event_listener_to_seed_database()
```

Note: Please ensure that the `sql_app.db` in the app directory is deleted before re-starting the application.



Backend structure

`Assets`

- Contains the seeding data and `helper.py` that seeds the data after starting up the backend. Including Geojson data and related image data.

`Controllers`

- Contains the various services classes that handles the vast majority of the backend business logic and serve as the vital endpoint APIs that both sends out responses to the user and handles requests by the user to the backend.

`Factory`

- Contains the classes responsible for the creation, initiation and definitions relating to the SQL database, SQL sessions and response models that the web application relies on to store data.

`minio`

- Holds the default images that will be featured on the website upon start up, for these images will be first stored in the image hosting client **minio** for use by the website. Minio is also used to store user-uploaded images.

`Models folder`

- Contains the SQL models and SQL table maps that is will be used to store data in the SQL database.

`Routers`

- Contains the classes that calls in the controller classes in real time as the frontend sends requests to the backend. They are the ones instantiated and run in the main application file.

`Schemas folder`

- Defines all the request and response fields.

`Services`

- Contains methods to create, read, update and delete business objects.

`Websocket`

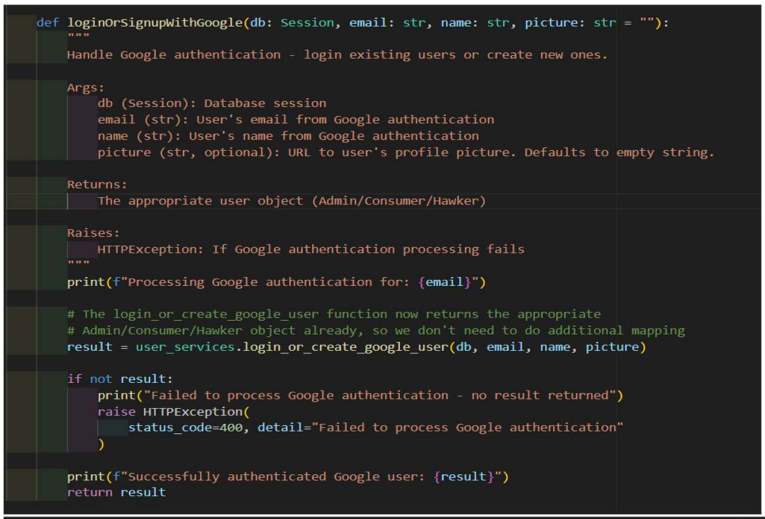
- Contains the handler function for the **websocket server** to serve as the information highway for the subscription and notifications system.

`database.py`

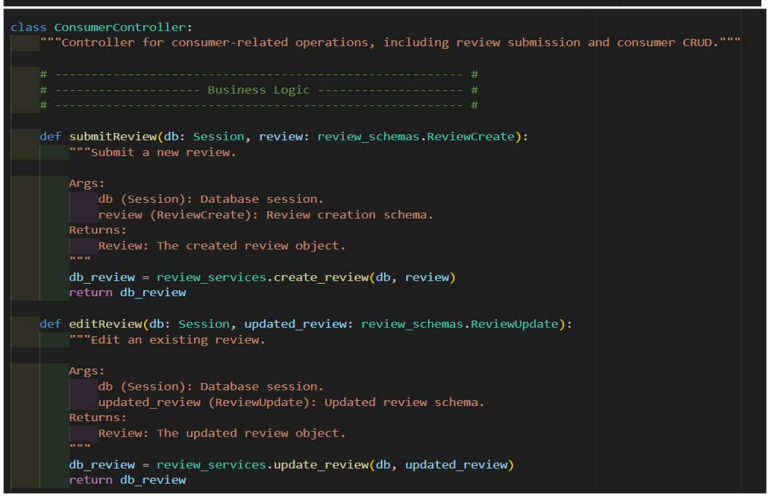
- Fetches the SQL sessions that is used by almost every class in the backend.

Code Comments

To promote clarity and understanding across the codebase, we have implemented thorough inline comments throughout our source code. These annotations not only streamline team collaboration but also simplify the onboarding process for new contributors. Figures 2 and 3 highlight examples from our well-documented code, illustrating our approach to maintaining code transparency.



```python
def loginOrSignupWithGoogle(db: Session, email: str, name: str, picture: str = ""):
    """
    Handle Google authentication - login existing users or create new ones.

    Args:
        db (Session): Database session
        email (str): User's email from Google authentication
        name (str): User's name from Google authentication
        picture (str, optional): URL to user's profile picture. Defaults to empty string.

    Returns:
        The appropriate user object (Admin/Consumer/Hawker)

    Raises:
        HTTPException: If Google authentication processing fails
    """
    print(f"Processing Google authentication for: {email}")

    # The login_or_create_google_user function now returns the appropriate
    # Admin/Consumer/Hawker object already, so we don't need to do additional mapping
    result = user_services.login_or_create_google_user(db, email, name, picture)

    if not result:
        print("Failed to process Google authentication - no result returned")
        raise HTTPException(
            status_code=400, detail="Failed to process Google authentication"
        )

    print(f"Successfully authenticated Google user: {result}")
    return result
```



```python
class ConsumerController:
    """Controller for consumer-related operations, including review submission and consumer CRUD."""

    # ------------------------------------------------------ #
    # -------------------- Business Logic -------------------- #
    # ------------------------------------------------------ #

    def submitReview(db: Session, review: review_schemas.ReviewCreate):
        """Submit a new review.

        Args:
            db (Session): Database session.
            review (ReviewCreate): Review creation schema.
        Returns:
            Review: The created review object.
        """
        db_review = review_services.create_review(db, review)
        return db_review

    def editReview(db: Session, updated_review: review_schemas.ReviewUpdate):
        """Edit an existing review.

        Args:
            db (Session): Database session.
            updated_review (ReviewUpdate): Updated review schema.
        Returns:
            Review: The updated review object.
        """
        db_review = review_services.update_review(db, updated_review)
        return db_review
```

Backend API Documentation

To facilitate seamless integration between frontend and backend development, we utilized FastAPI's built-in documentation tools to create an interactive API reference site. This site includes detailed route definitions, required inputs, and corresponding outputs, along with live request-response simulations.

Figure 4 shows the API documentation interface, while Figure 5 depicts a sample endpoint with its associated request and response formats.

For full access and setup guidance, refer to the API Docs Setup Instructions on our GitHub.

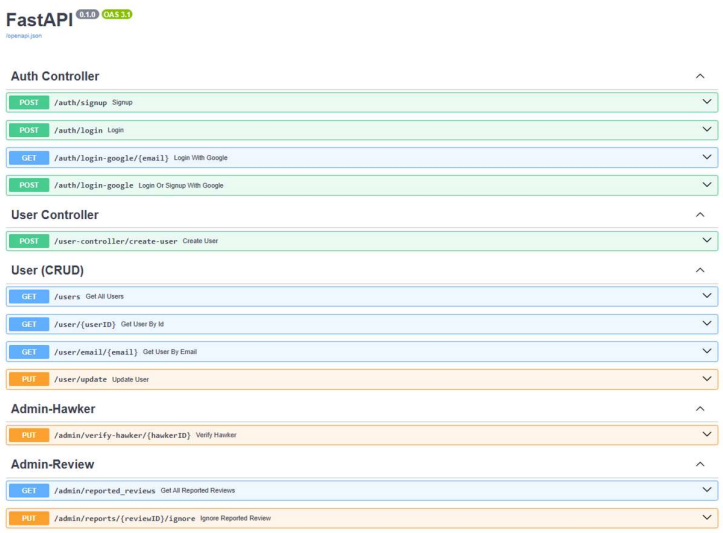


Figure 4: API documentation interface

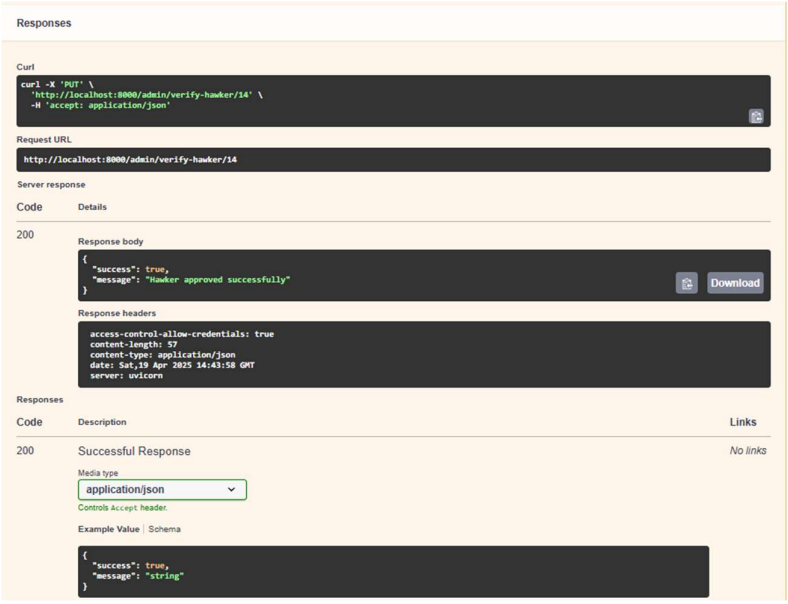


Figure 5: A sample endpoint with its associated request and response formats.