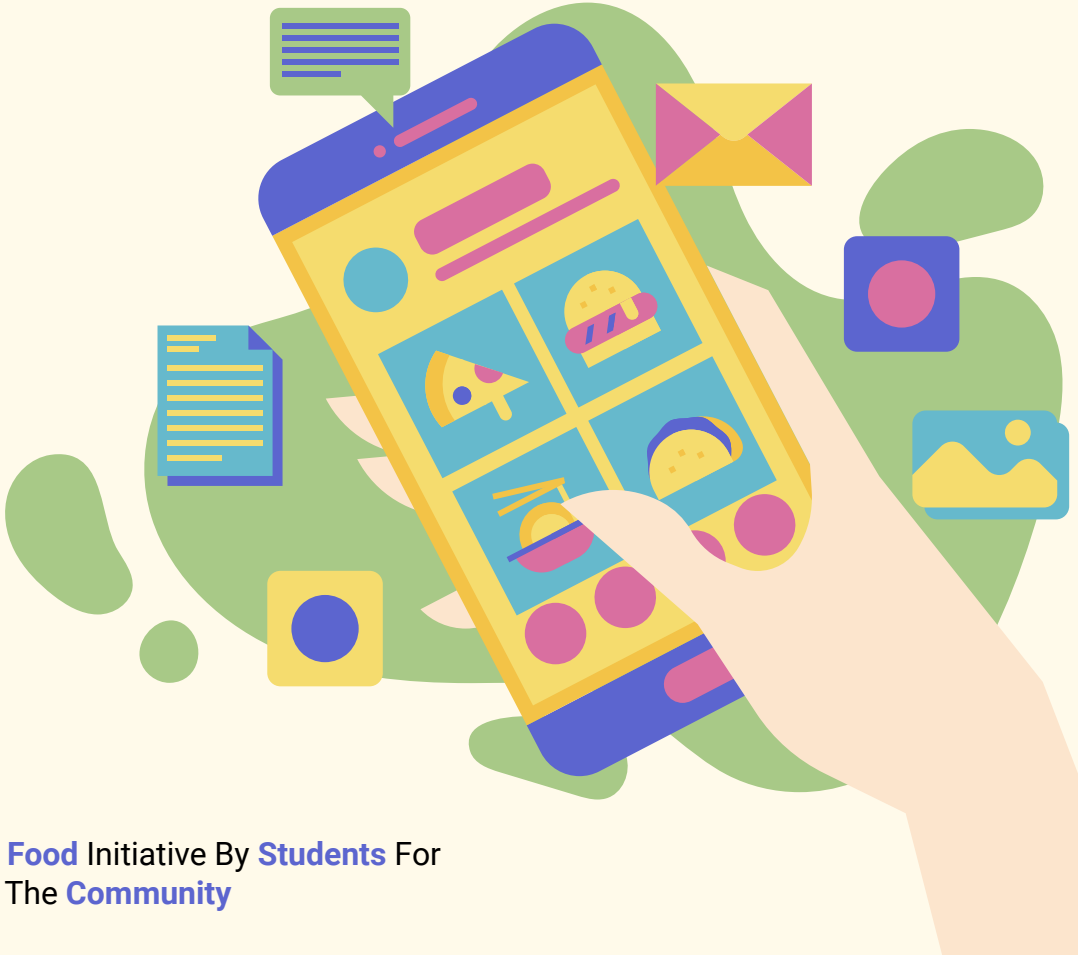
Feed It Forward

SCSX

1. Jing Qiang
2. Jing Hua
3. Minze
4. Denise
5. Tommy
6. Pinyang



A **Sustainable**, **Food** Initiative By **Students** For The **Community**

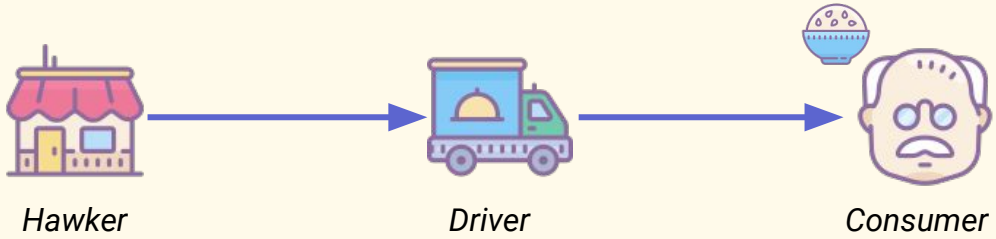
Problem

1. Low-income families face **food insecurity**, struggling to access sufficient, nutritious meals daily.
2. Local food vendors and hawkers often encounter **food wastage**, with unsold leftovers discarded at the end of each day.

What if we can Re-distribute the leftover food to those who need it?

Hawker *Driver* *Consumer*

What if we can Re-distribute the leftover food to those who need it?



Solution – FeedItForward

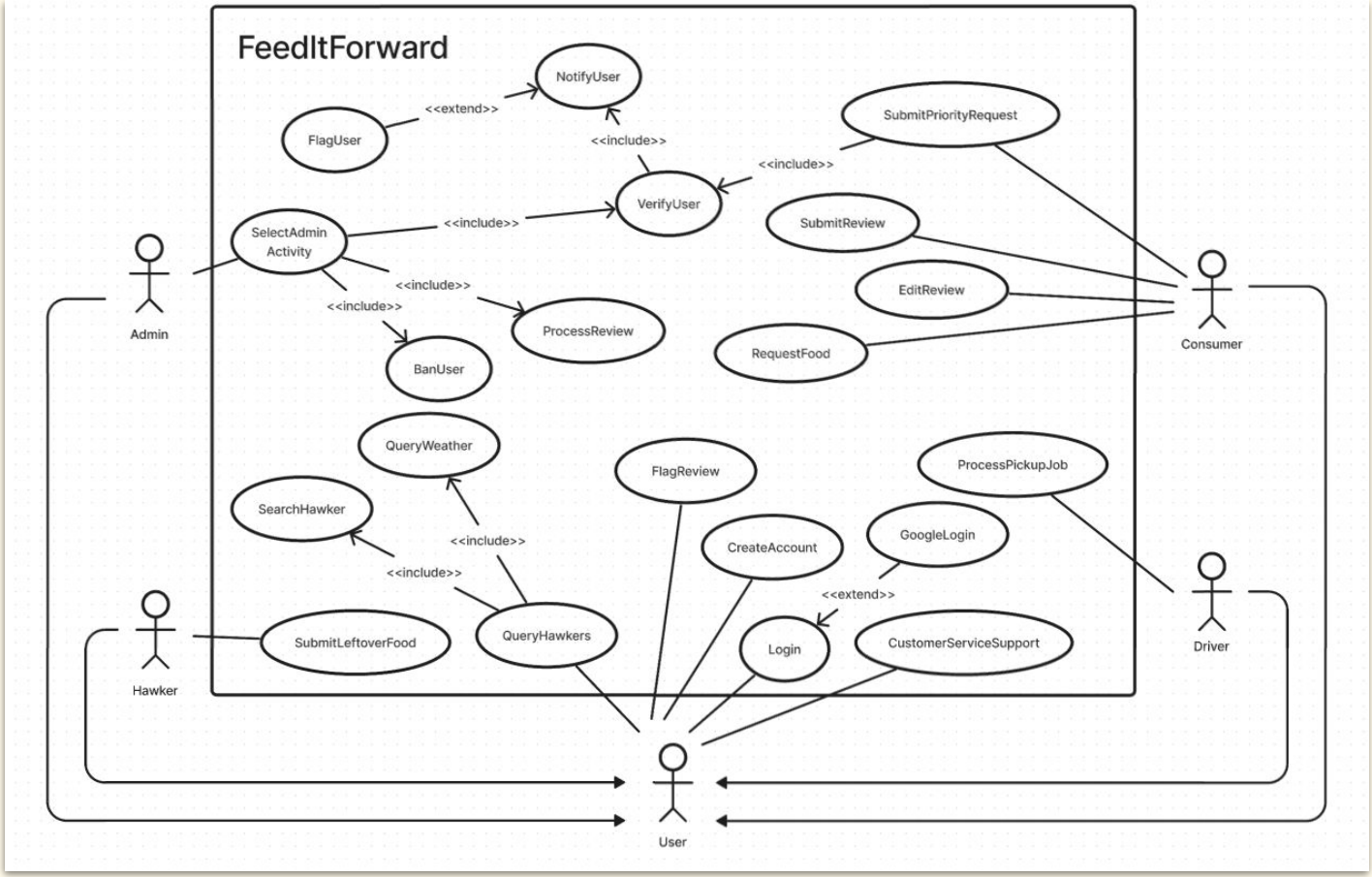
A **community-driven** initiative that **connects** the surplus food from local hawkers directly to families in need.

This approach not only **reduces food waste** but also ensures that **nutritious meals reach those most vulnerable** in our society.

Key Features

- Authentication
- **Hawkers**
- **Leftover Food**
- Food Priority Request
- **Reviews**
- **Pickup Jobs**
- Admin Management
- **Live Map**
- Weather Forecast
- **Real-time Chatting Interface**
- **Notifications**
- Settings / Profile
- Search

Use Case Diagram



Feature List

1. **Authentication**
    - Signup (with different roles – Admin, Hawker, Consumer, Driver)
    - Login
    - Login with Google

2. **Hawkers**
    - View Hawker Listing Profile
    - Submit Leftover Food
    - Map
        - Displays all the registered and external api public hawkers

Feature List

3. **Consumer**
    - Hawker
        - View and Search for Hawkers
        - View Hawker Listing
    - Leftover Food
        - View and Search for Leftover Food
        - Request for Leftover Food
    - Reviews
        - View, Add, and Edit Hawker Reviews
        - Flag Review
    - Map
        - View Hawkers (registered tag)
        - Search Hawkers
    - Food Priority Request

Feature List

4. **Driver**
    - Accept/Ignore Pickup Job
    - Submit photo proofs to complete Pickup Job
    - Map
        - Weather Forecast

5. **Admin**
    - Verify User → Priority Tag Given to Consumers
    - Ban User
    - Process Reviews
    - **Notifications**
        - Notify User (Admin send custom notifications)

6. **Notifications**

Feature List

7. **Customer Service Support**
    - Real-time messaging with Admin

8. **Misc**
    - Settings
    - Edit Profile

External APIs

1. **Google OAuth 2.0 API** } Seamless Authentication for **Users**
2. **Singapore's OneMap API**
    a. Live Map
    b. Geocoding
3. **Hawker GeoJson Dataset API**

} Display Registered and Public **Hawkers** on a **Live Map** real-time

4. **Weather API (Live)**
    a. 24-hour Weather Forecast
    b. 4-day Weather Forecast

} Forecast weather for **Drivers** to plan their routes and work schedule, and for **Consumers** to decide whether they want the food to be delivered or self-collected

Live Demo!

Tech Stack

Front-end

- React.js
- Typescript
- Tailwindcss

Back-end

- FastAPI
- Python
- SQL

Software Engineering Practices & Design Patterns

01

Good SWE Practices

1. Documentation
2. Good Code practices
3. Reusability and Refactoring
4. SCRUM

Documentation - README

FeedItForward (Backend)

This README.md assumes that you have already cloned the repo.

Table of Content

Setup Instructions

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

```
python -m venv .venv
. .venv/Scripts/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
```

App Design (Backend)

Design Patterns

1. `Strategy + Factory Patterns` for Database Implementation. a. Refer to `backend/app/factory/database.py`
2. `Facade Pattern` via the Controllers. a. Refer to `backend/app/controllers/*.py`
3. `Publisher-Subscriber Pattern` for real-time text messaging communication between different users (websocket). a. Refer to `backend/app/websocket.py`

Folder Architecture

- `app/assets`
  - Contains server assets like data files and uploads.
- `app/models`
  - Contains the Business Objects.
- `app/services`
  - Contains methods to create, read, update, and delete business objects.
- `app/controllers`
  - Controllers that uses the various services implemented in the `app/services` directory.
  - They implements the Facade Pattern by masking the more complex underlying implementation details from the frontend.
- `app/routers`
  - Routers that implements REST API endpoints for communication between frontend and backend. They allow the frontend to use the controllers in the backend.
  - Routers use the controllers implemented in the `app/controllers` directory.
- `app/schemas`
  - Defines all the request and response fields.
- `app/factory`
  - Factory design pattern implementation of the database.
- `app/database.py`
  - Entry point to database that will store all the business objects.
- `app/websocket.py`
  - Implements the Publisher-Subscriber Pattern via websockets for real-time text messaging communication between different users.

Serves as a guide for future developers, enabling long-term consistency and good development

Documentation - Code Comments

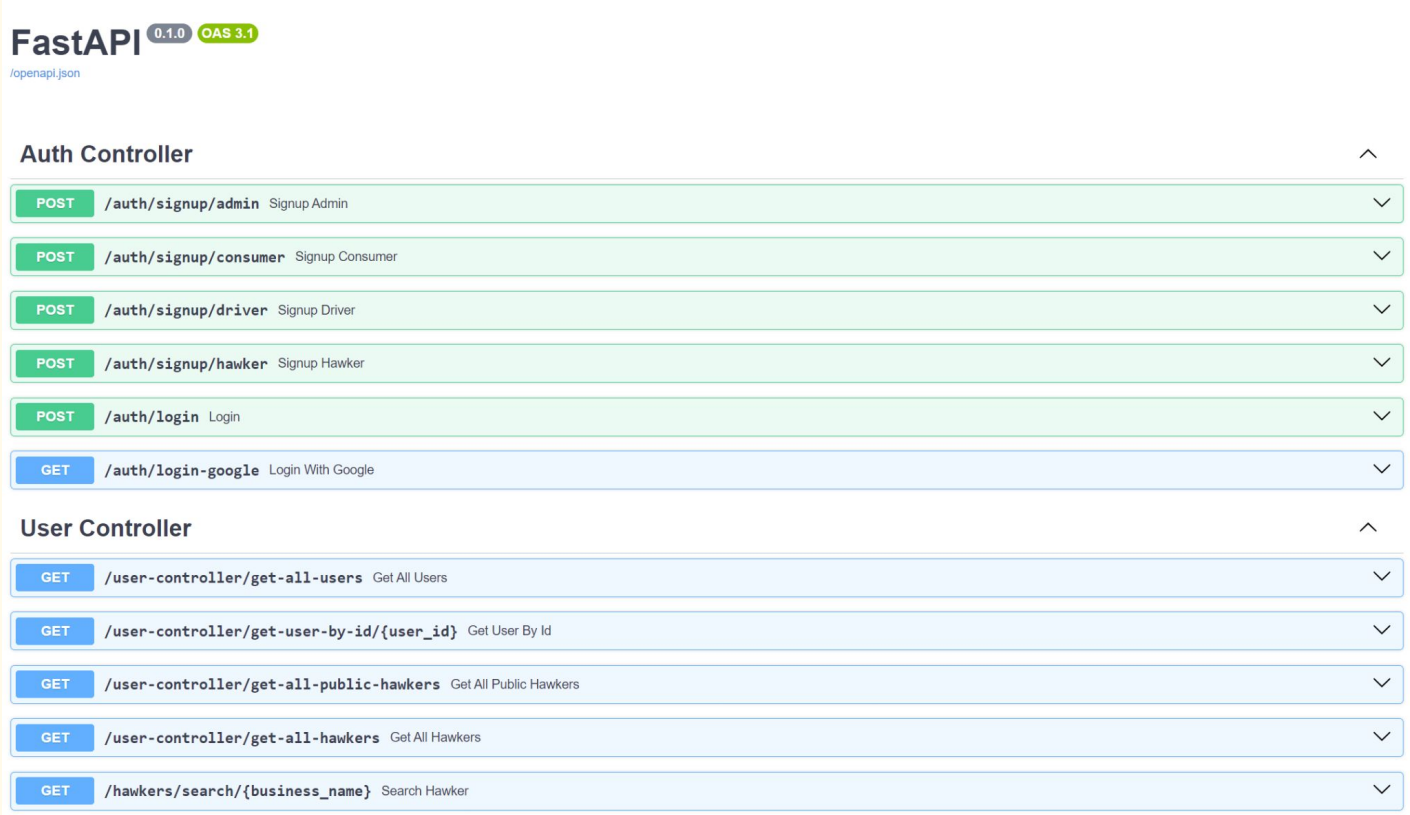
```python
class FileController:
    # ----- File ----- #
    def uploadFile(user_id: int, file: UploadFile, db: Session):
        # Validate file type - Must be Image
        if file.content_type not in ["image/jpeg", "image/png", "image/gif"]:
            raise HTTPException(status_code=400, detail="Invalid file type. File type must be jpeg, png, or gif")

        # Create upload directory if it does not exist
        user_upload_directory = os.path.join(server_upload_base_directory, "user_{0}".format(user_id))
        if not os.path.exists(user_upload_directory):
            os.makedirs(user_upload_directory)

        # Destination File Path
        dest_file_path = os.path.join(user_upload_directory, file.filename)

        # Copy the file contents
        with open(dest_file_path, "wb") as buffer:
            shutil.copyfileobj(file.file, buffer)

        file_response = FileUploadResponse(
            file_path=str(os.path.join("user_{0}".format(user_id), file.filename))
        )

        return file_response
```

```jsx
{isWeatherModalOpen && (
  <div className="absolute top-1/2 left-1/2 -translate-x-1/2 -translate-y-1/2 w-[75%] px-4 py-3 rounded bg-[#E3F6F5]">
    {/* Title */}
    <div className="flex flex-col items-center">
      <div className="font-bold text-[22px]">Weather Forecast</div>
      <div className="text-[14px] italic">
        Singapore | {date.toLocaleDateString()}
      </div>
    </div>

    {/* Weather Forecast */}
    {/* 24 Hour */}
    {weather24HrForecasts && (
      <div className="mt-4">
        <div className="font-bold underline text-[14px] text-center mb-1">
          24 Hours
        </div>
        <div className="flex flex-col gap-1">
          {REGIONS.map(region => (
            <RegionWeatherForecast24HrRow
              key={region}
              weather24HrForecasts={weather24HrForecasts}
              getWeatherIcon={getWeatherIcon}
              region={region}
            />
          ))}
        </div>
      </div>
    )}

    {/* 4 Day */}
    {weather4DayForecasts && (
      <div className="mt-4">
        <div className="font-bold underline text-[14px] text-center mb-1">
```

Makes code easier to understand (human readable) for enhanced collaboration

Documentation - API Docs

Allows different teams (Frontend and Backend) to **collaborate together effectively**, and enhance future extensibility



Consistent Code Style and Naming Convention

Front-end

Eslint

Prettier

Back-end

Black

Ensures that Code is **readable**, and **maintainable**, and **understandable**

Reusability & Refactoring

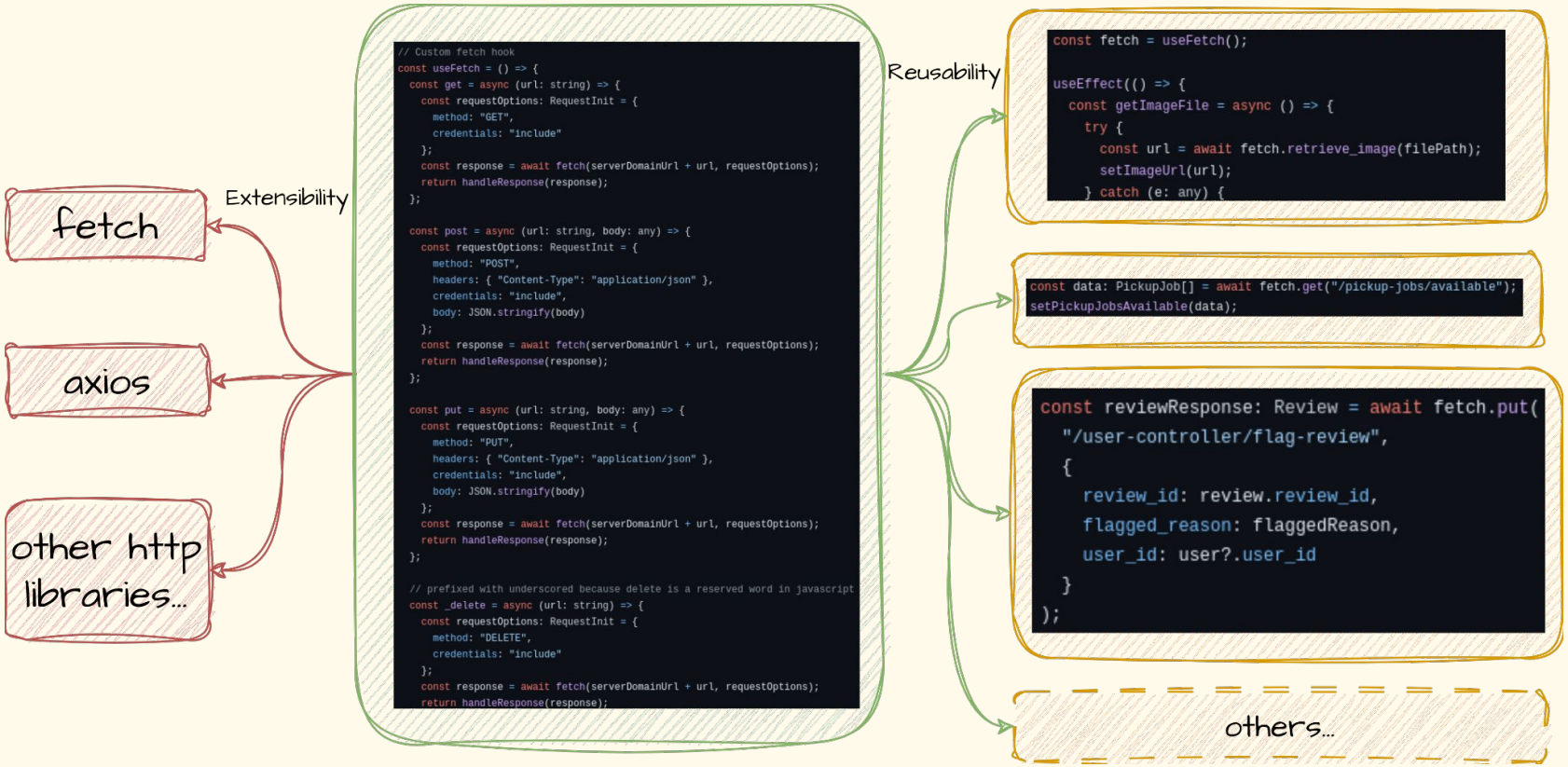
Remove Duplication

Remove Redundancy

Consistent Naming Convention

Improve Reusability

Reusability & Refactoring



Reusability & Refactoring

```tsx
export const SearchBar = (props: SearchBarProps) => {
  const { searchItemPlaceholder, handleSearch, handleOnClear, className } =
    props;
  const [searchKey, setSearchKey] = useState("");

  const handleValueChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    setSearchKey(e.target.value);

    if (e.target.value.length > 2) {
      handleSearch(e.target.value);
    } else {
      if (handleOnClear) handleOnClear();
    }
  };

  return (
    <div
      className={
```

```tsx
<ScreenTitle title="Leftover Food" />

{/* Search Bar */}
<SearchBar
  searchItemPlaceholder="Leftover Food"
  handleSearch={handleSearch}
  handleOnClear={handleSearchClear}
  className="my-[7%]"
/>
```

```tsx
<SearchBar
  searchItemPlaceholder="reviews or user"
  handleSearch={handleSearchReviews}
  handleOnClear={handleOnSearchClear}
/>
```

```tsx
<div>
  <ScreenTitle title="Ban Users" />
  <div className="flex flex-col justify-center mt-12 gap-10">
    <SearchBar
      searchItemPlaceholder="user to ban"
      handleSearch={handleSearchUsers}
      handleOnClear={handleOnSearchClear}
    />
```

others...

Reusability & Refactoring

```tsx
export const ScreenTitle = ({ title, backNav = true }: ScreenTitleProps) => {
  return (
    <div
      className={`flex flex-col items-center justify-center gap-1 ${
        backNav ? "pt-3" : "pt-12"
      }`}
    >
      {backNav && (
        <div className="self-start">
          <ButtonBackNavigation />
        </div>
      )}
      <div className="text-[28px] font-nunito font-bold text-center leading-tight">
        {title}
      </div>
    </div>
  );
};
```

```tsx
export const AdminScreen = () => {
  return (
    <div>
      <ScreenTitle title="Admin Management" />
```

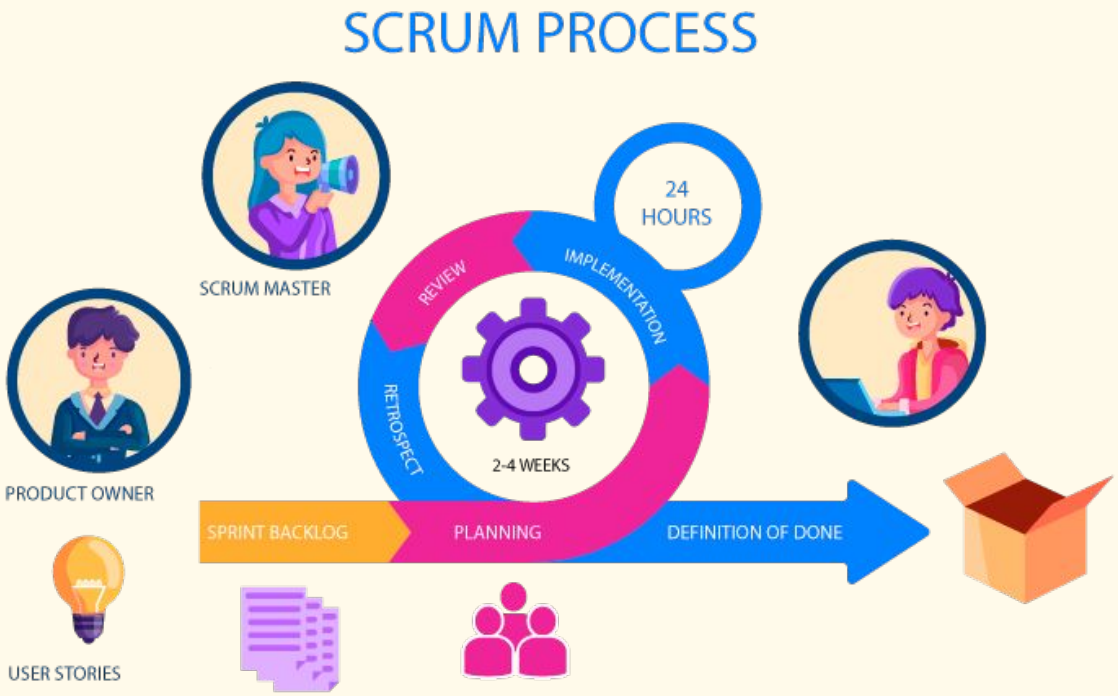
```tsx
{/* Screen Title */}
<ScreenTitle title="Hawkers" backNav={false} />
```

```tsx
return (
  <div>
    <ScreenTitle title="Submit Leftover Food" />
```

others...

For delivery of value incrementally and rapidly in a collaborative way.

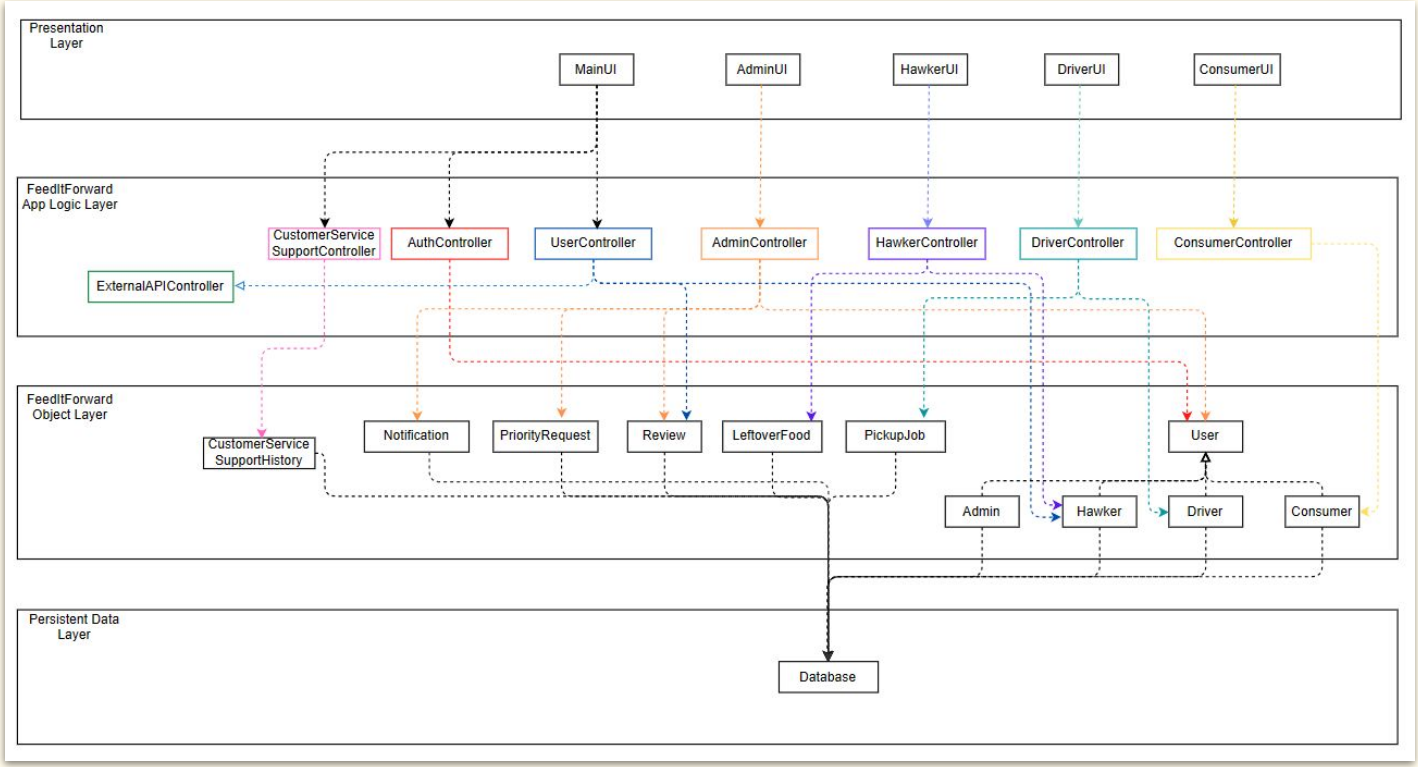
SCRUM



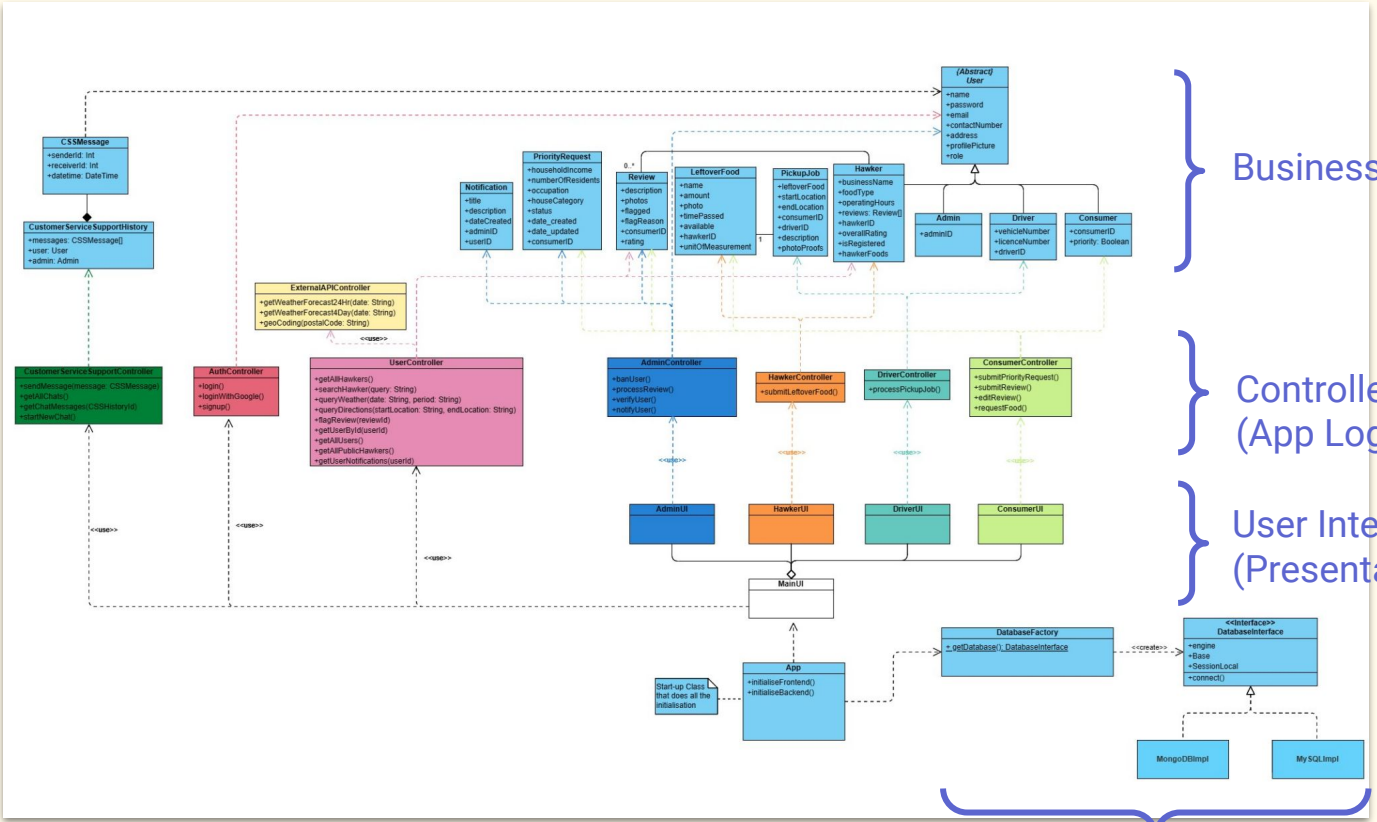
02

System Design

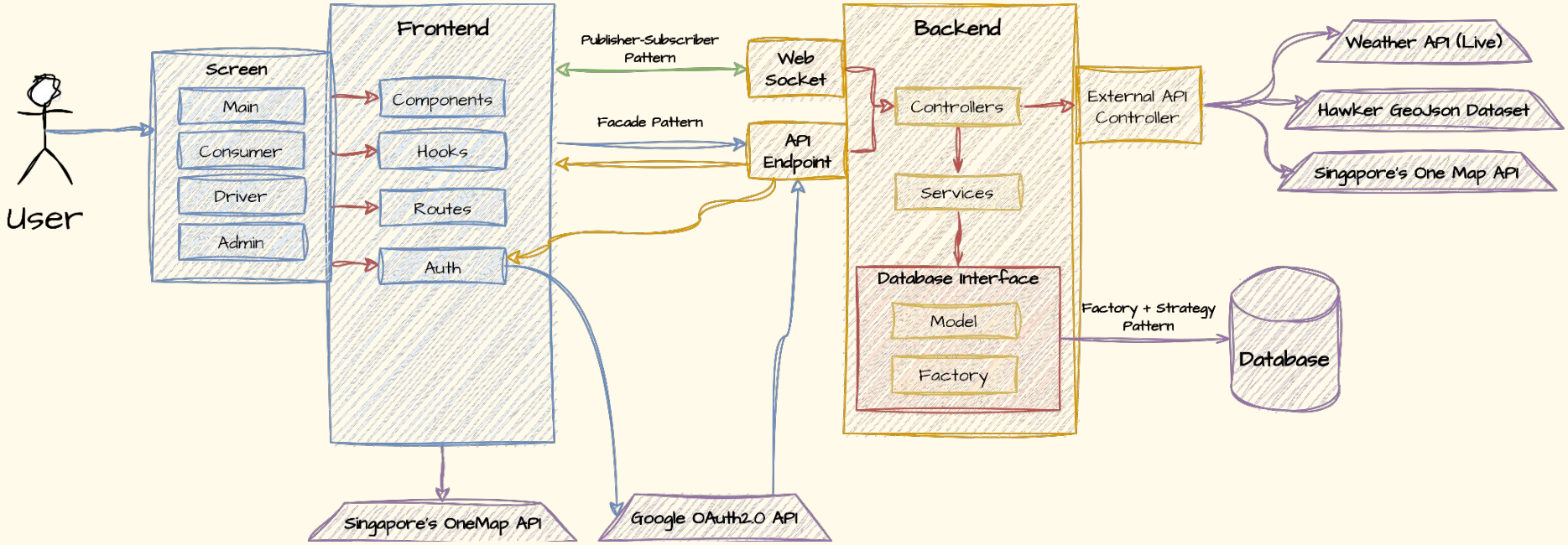
Architecture Diagram (Layered Architecture)



Class Diagram



Overview Diagram

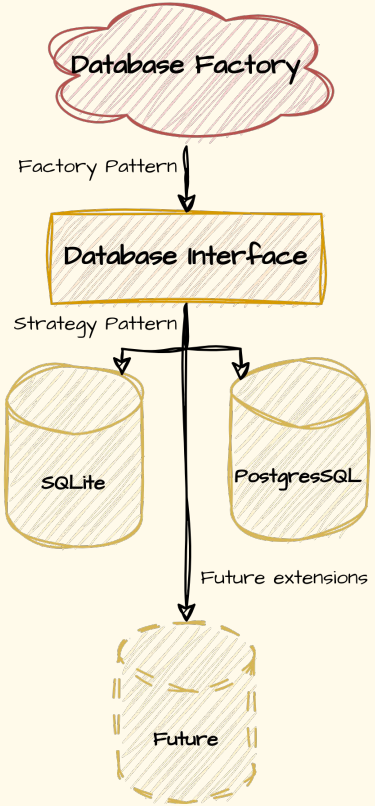


03

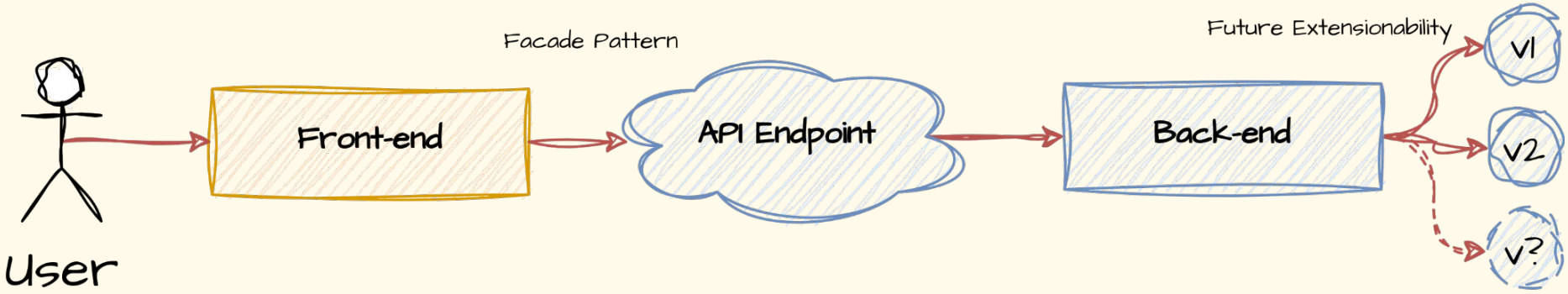
Design Patterns

1. Strategy & Factory Pattern
2. Facade Pattern
3. Publisher-Subscriber Pattern
4. SOLID Principles

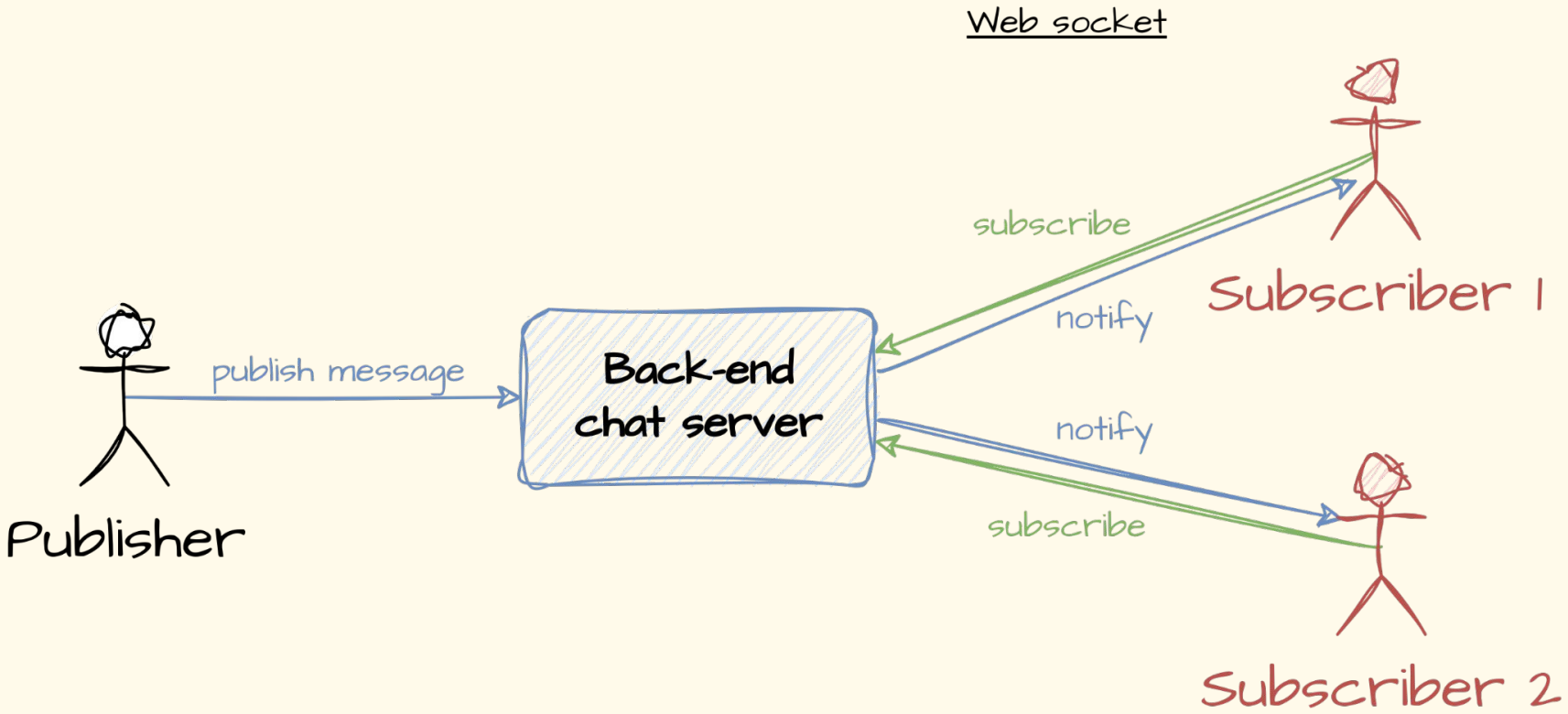
Strategy & Factory Pattern



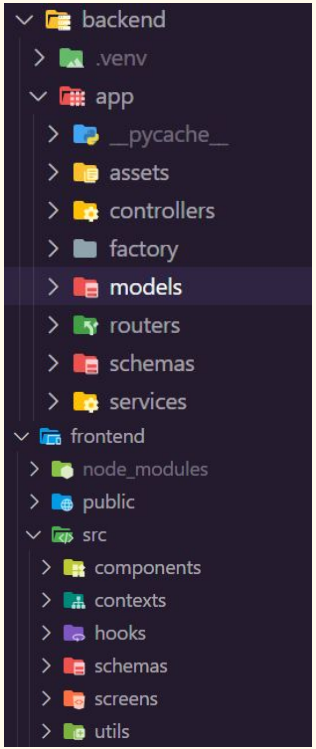
Facade Pattern



Publisher-Subscriber Pattern (Web-socket)



For extensibility, flexibility, maintainability, and ease of understanding



SOLID Principles

Single Responsibility Principle (SRP)

Different **packages** with different **distinct responsibilities** are created. **Specific classes** in each package are responsible for a specific business model or logic group.

Open-Closed Principle (OCP)

The application is open to extension through the use of **strategy pattern** (database) as well as **facade pattern** (controllers).

Dependency Inversion Principle (DIP)

High level and low level modules depends on abstractions through the **use of the interfaces** (**backend/app/schemas** and **frontend/src/schemas**).

Interface Segregation Principle (ISP)

Numerous **small and specific interfaces** are used. For example, in **`backend/app/schemas/consumer.py`**, there are **Consumer**, **ConsumerCreate**, and **ConsumerUpdate** interfaces which are used for different purposes (Display, Creation, and Update).

Traceability in project deliverables

Login

Use-case Description

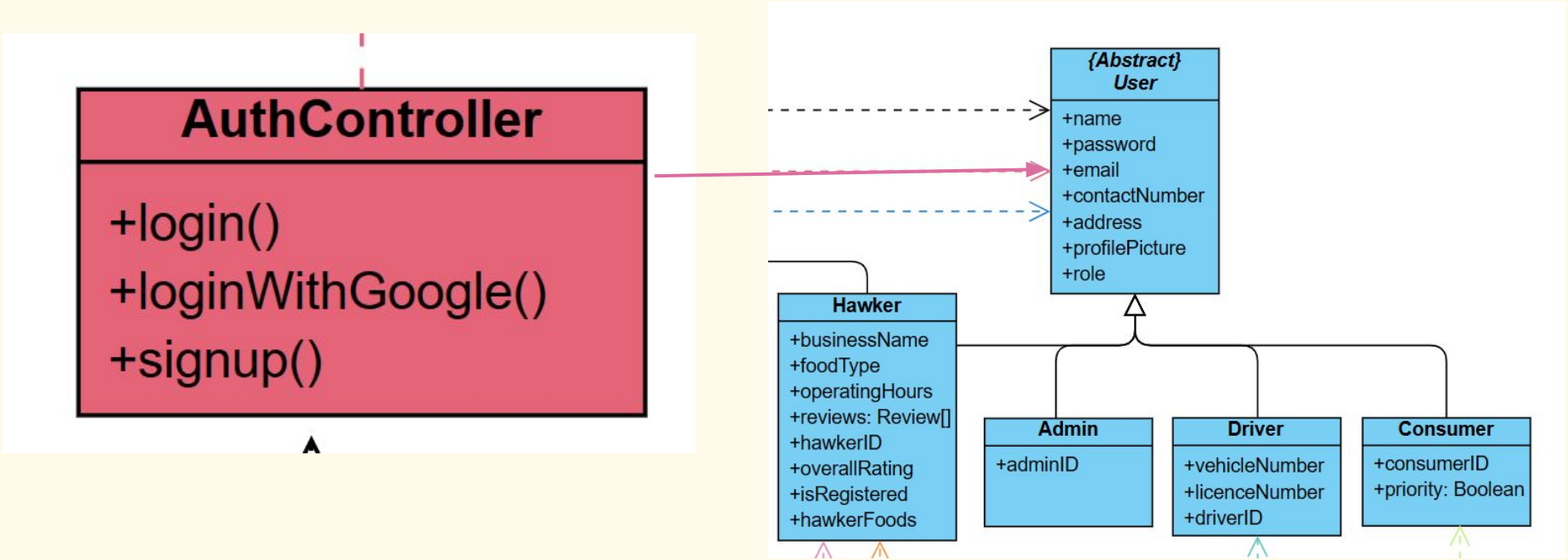
VII.II Login

Use Case ID:	#7-2		
Use Case Name:	Login		
Created By:	Pin Yang	Last Updated By:	Denise
Date Created:	13th September 2023	Date Last Updated:	27th September 2023

Actor:	User
Description:	Allows User to log into his/her FeedItForward Account using his/her email and password.
Preconditions:	None
Postconditions:	User is logged into the FeedItForward application and is navigated to the home screen of the application.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. The system allows the User to login with either (a) Email and password, or (b) Google. 2. The User chooses to login with email and password. 3. The User enters his/her email and password. The Password is masked as dots, but the User can choose to unmask it by clicking on the eye icon. 4. The User selects the “Login” button.
Alternative Flows:	AF-S2: User chooses to login with Google 1. The User selects the “Login with Google” button. 2. The User logins into the system using Singpass login credentials using the included use case **GoogleLogin**.
Exceptions:	1. If any required information is missing, an error message is displayed. 2. If email and password do not match when the User tries to login in step 4, FeedItForward shall display “Email and password do not match” to the user.
Includes:	GoogleLogin
Special Requirements:	System needs to validate user input data.
Assumptions:	The User has an existing FeedItForward Account.

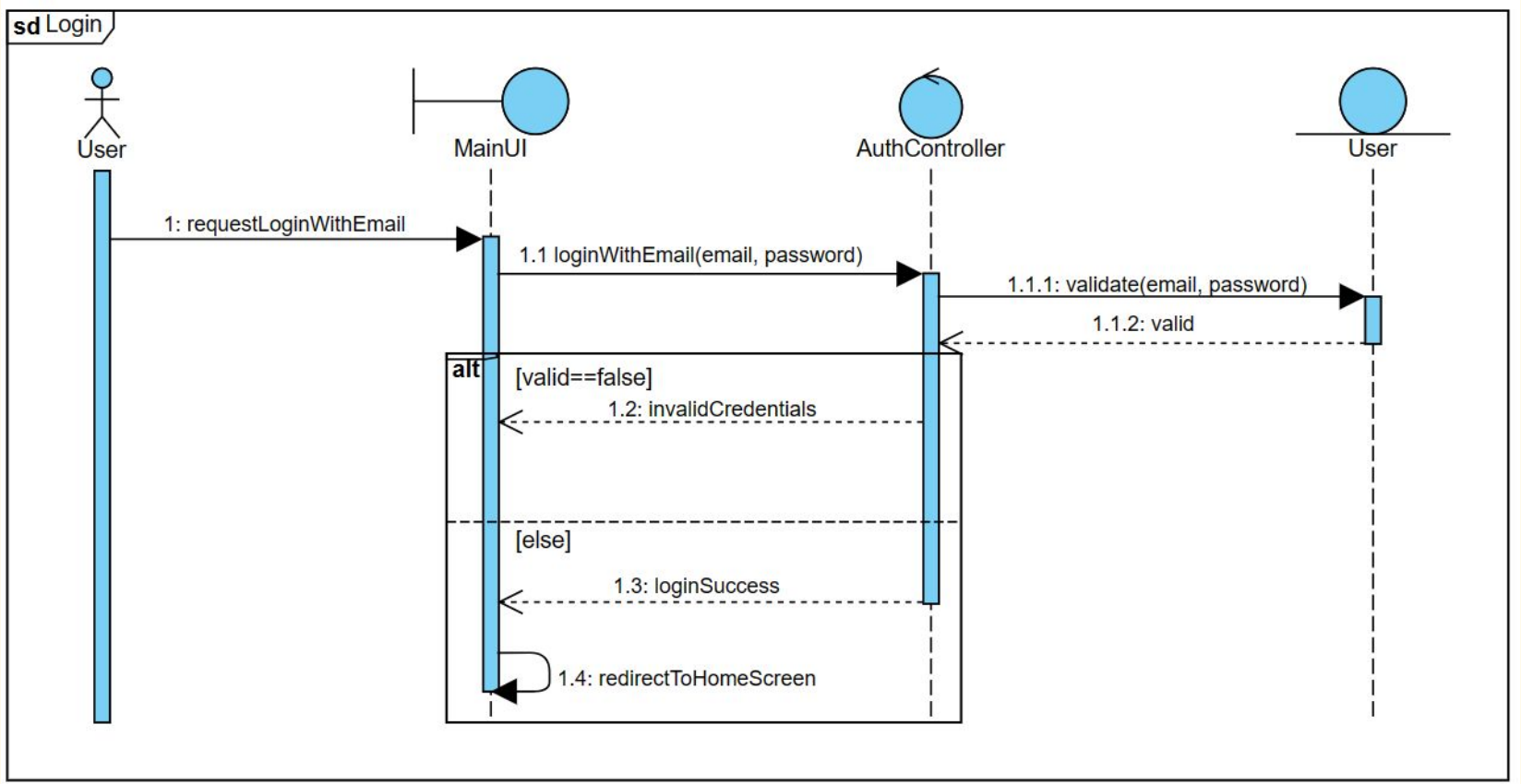
Login

Relevant Class Diagram
(Visual Paradigm)



Login

Relevant Sequence Diagram
(Visual Paradigm)



Login - Good Designs Applied

(Visual Studio Code)

- Facade Pattern

Controller

```python
class AuthController:
  def signup(db: Session, user: user_schemas.UserCreate):···

  def login(db: Session, user: user_schemas.UserLogin):
    db_user = user_services.login_user(db, user)
    if not db_user:
        raise HTTPException(status_code=400, detail="Invalid login credentials")
    if db_user.ban:
        raise HTTPException(status_code=403, detail="Forbidden. User has been banned")

    match db_user.role:
        case user_schemas.Role.ADMIM:
            return admin_services.get_admin_by_user_id(db, db_user.user_id)
        case user_schemas.Role.CONSUMER:
            return consumer_services.get_consumer_by_user_id(db, db_user.user_id)
        case user_schemas.Role.DRIVER:
            return driver_services.get_driver_by_user_id(db, db_user.user_id)
        case user_schemas.Role.HAWKER:
            return hawker_services.get_hawker_by_user_id(db, db_user.user_id)

    return db_user
```

Login - Good Designs Applied

(Visual Studio Code)

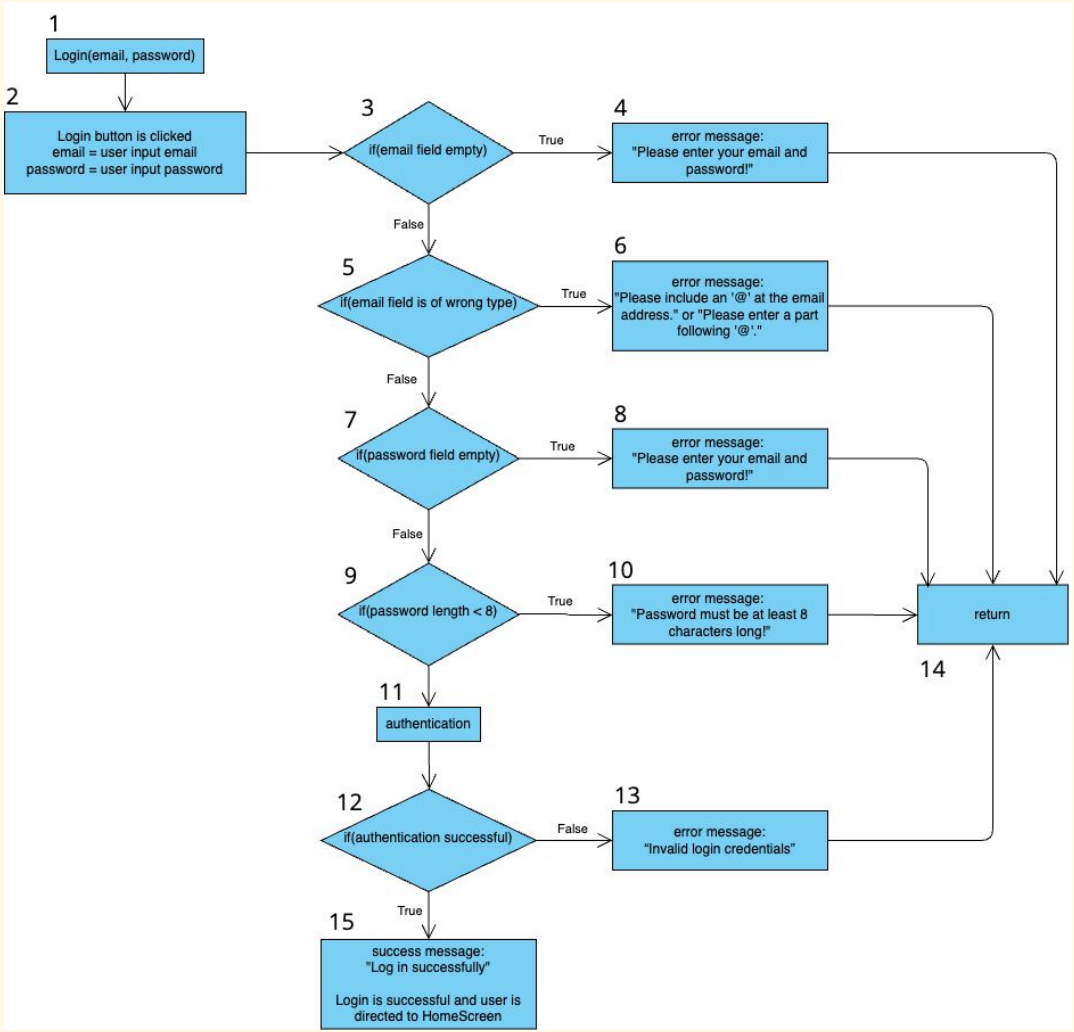
- Facade Pattern

Underlying service to login user

```python
def login_user(db: Session, user: user_schemas.UserLogin):
    db_user = db.query(User).filter(User.email == user.email).first()
    if not db_user:
        return None

    if bcrypt.checkpw(user.password.encode('utf-8'), db_user.password):
        return db_user

    return None
```

Login - Tests

(Black Box and White Box)



(Visual Paradigm)

Login - Tests

(Black Box and White Box)

I.II Basic Path Testing

Cyclomatic Complexity = | decision points | + 1 = 5 + 1 = 6

Basis Paths

1. Baseline path: 1, 2, 3, 5, 7, 9, 11, 12, 15
2. Basis path 2: 1, 2, 3, 4, 14
3. Basis path 3: 1, 2, 3, 5, 6, 14
4. Basis path 4: 1, 2, 3, 5, 7, 8, 14
5. Basis path 5: 1, 2, 3, 5, 7, 9, 10, 14

Basis path 6: 1, 2, 3, 5, 7, 9, 11, 12, 13, 14

Login - Tests

(Black Box and White Box)

Login(email, password)

No.	Test Input	Expected Output	Actual Output	Pass?
1	email = “admin1@gmail.com” password = “123123123”	"Log in successfully"	"Log in successfully"	Yes
2	email = “” password = “123123123”	"Please enter your email and password!"	“Please enter your email and password!"	Yes
3	email = “admin1” password = “123123123”	"Please include an '@' at the email address."	"Please include an '@' at the email address."	Yes
4	email = “admin1@” password = “123123123”	“Please enter a part following '@'.”	“Please enter a part following '@'.”	Yes
5	email = “admin1@gmail.com” password = “”	"Please enter your email and password!"	“Please enter your email and password!"	Yes
6	email = “admin1@gmail.com” password = “1234567”	"Password must be at least 8 characters long!"	"Password must be at least 8 characters long!"	Yes
7	email = “pinyang@gmail.com” password = “123123123”	“Invalid login credentials”	“Invalid login credentials”	Yes

Future Plans

AI-Powered Automated Food Allocated System

OneMap Directions API Integration for better route planning for Drivers and Consumers

Collaboration with **Singapore Government** and **Non-Profit Organisations** to enhance our outreach to **FeedItForward** to those who are in need

THANK YOU!

Let's **FeedItForward** for a **Sustainable** and **Secure** Food Future for our Community!