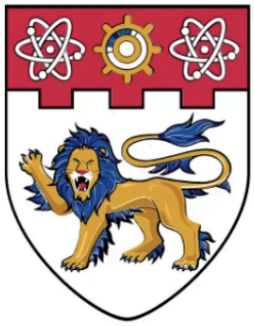


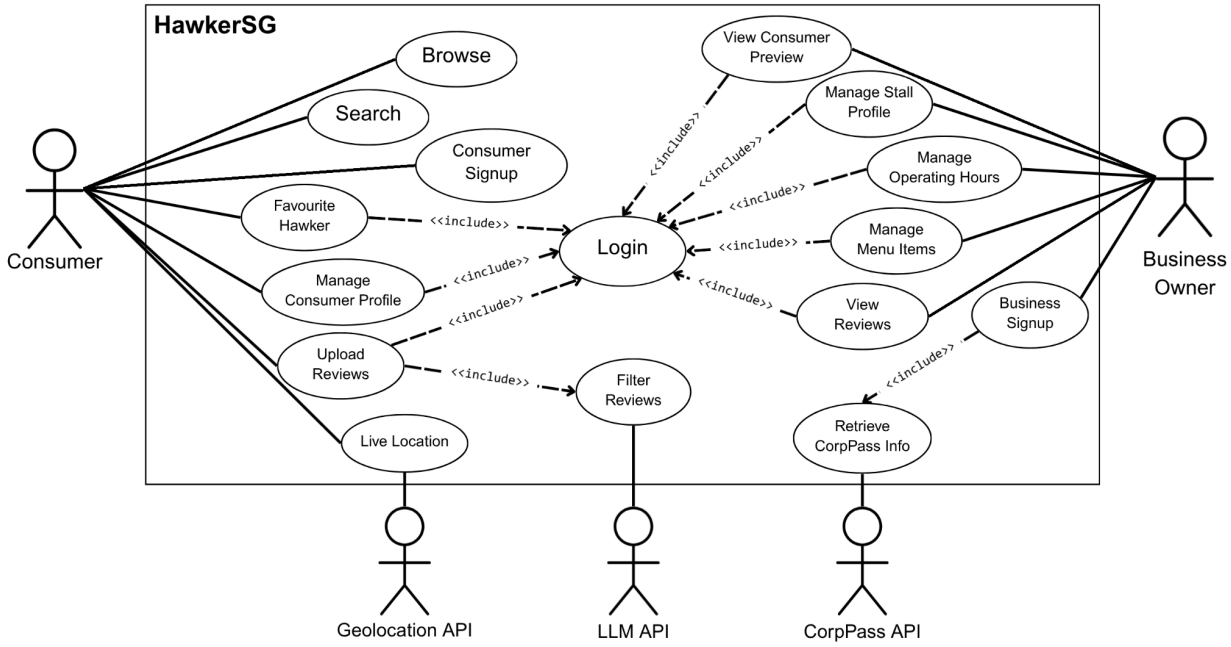
NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

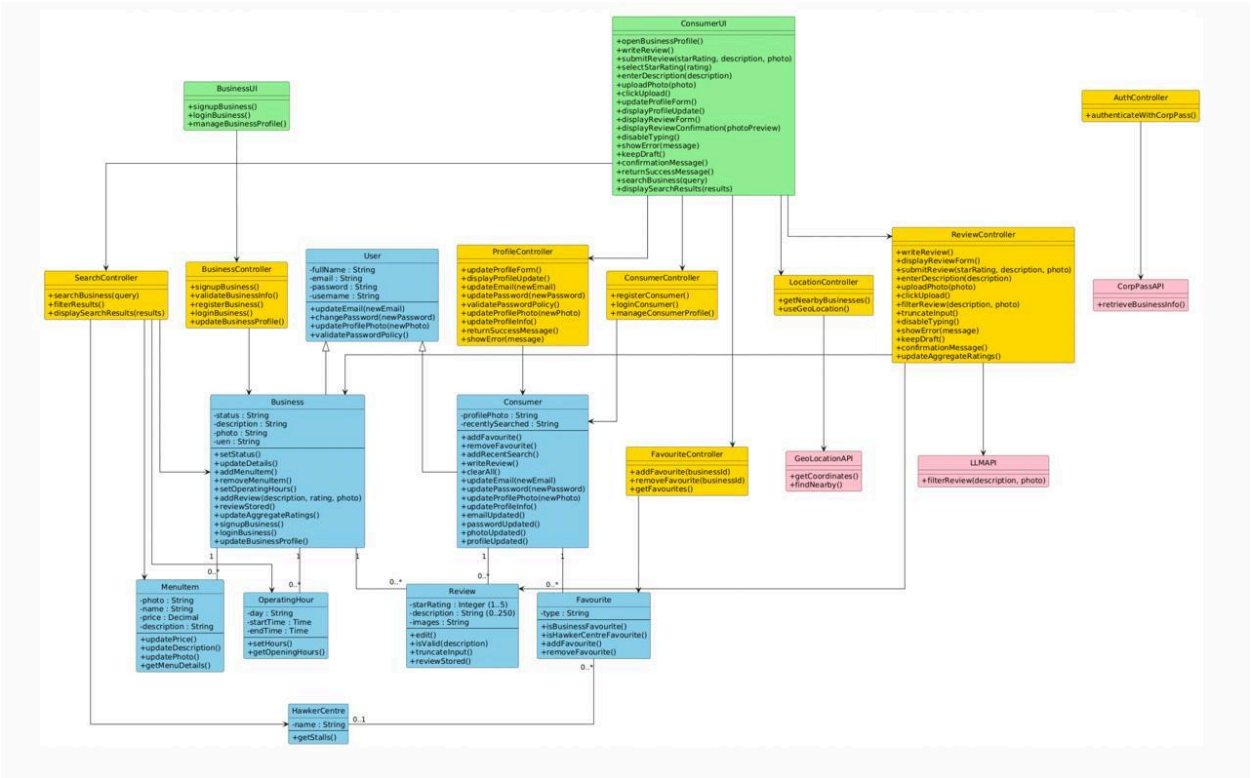
Lab Group	SCED
Team	Open Circuit
Members	Leck Kye-Cin (U2423244A)
	Lee Jun De, Kavan (U2423202E)
	Lee Yong Liang (U2422741K)
	Lee Zi An (U2323563A)
	Lim Seow Kiat (U2420309G)
	Lim Xiao Xuan (U2423602D)

Table of Contents

1. Use Case Diagram

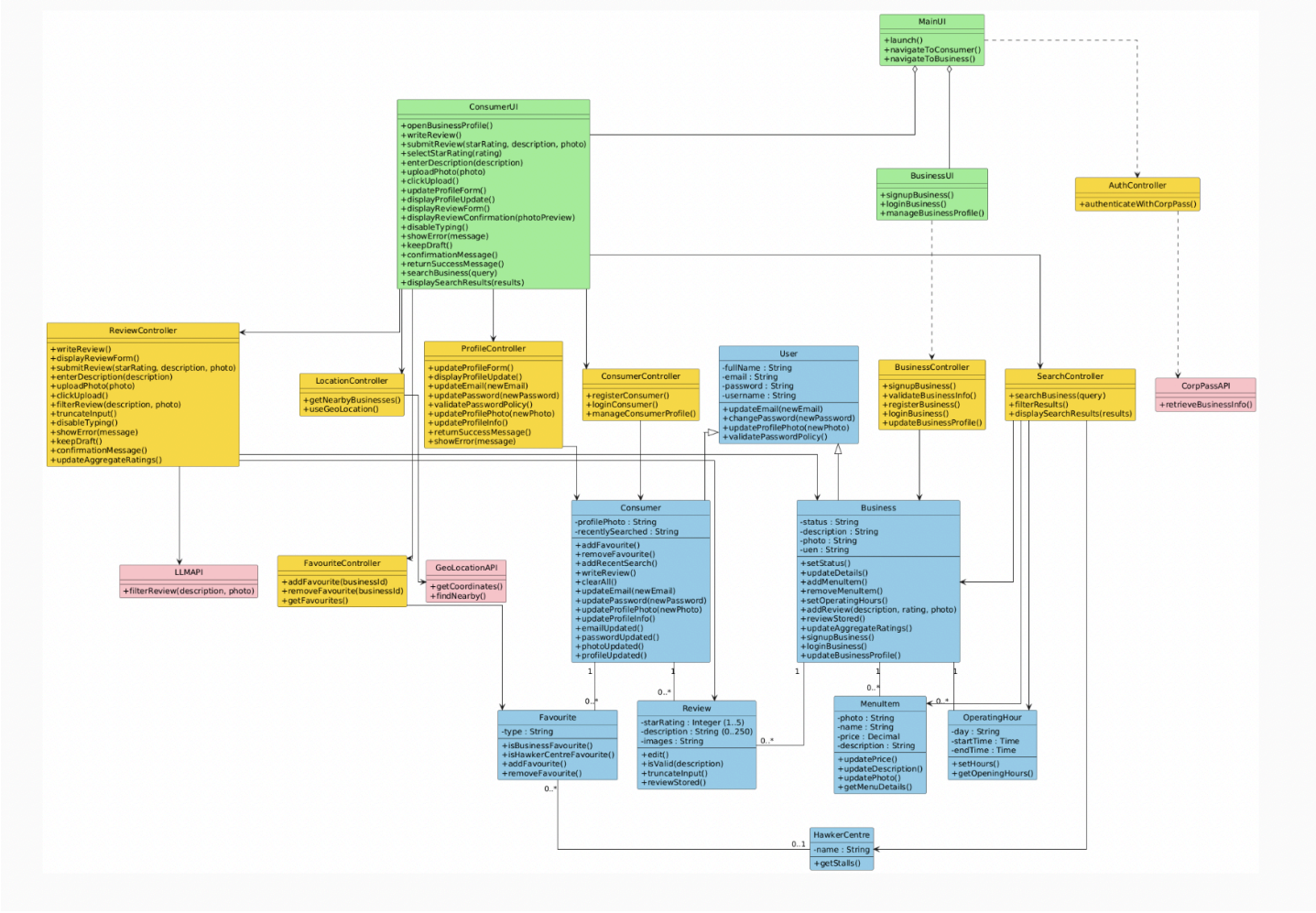
A. Use Case Diagram



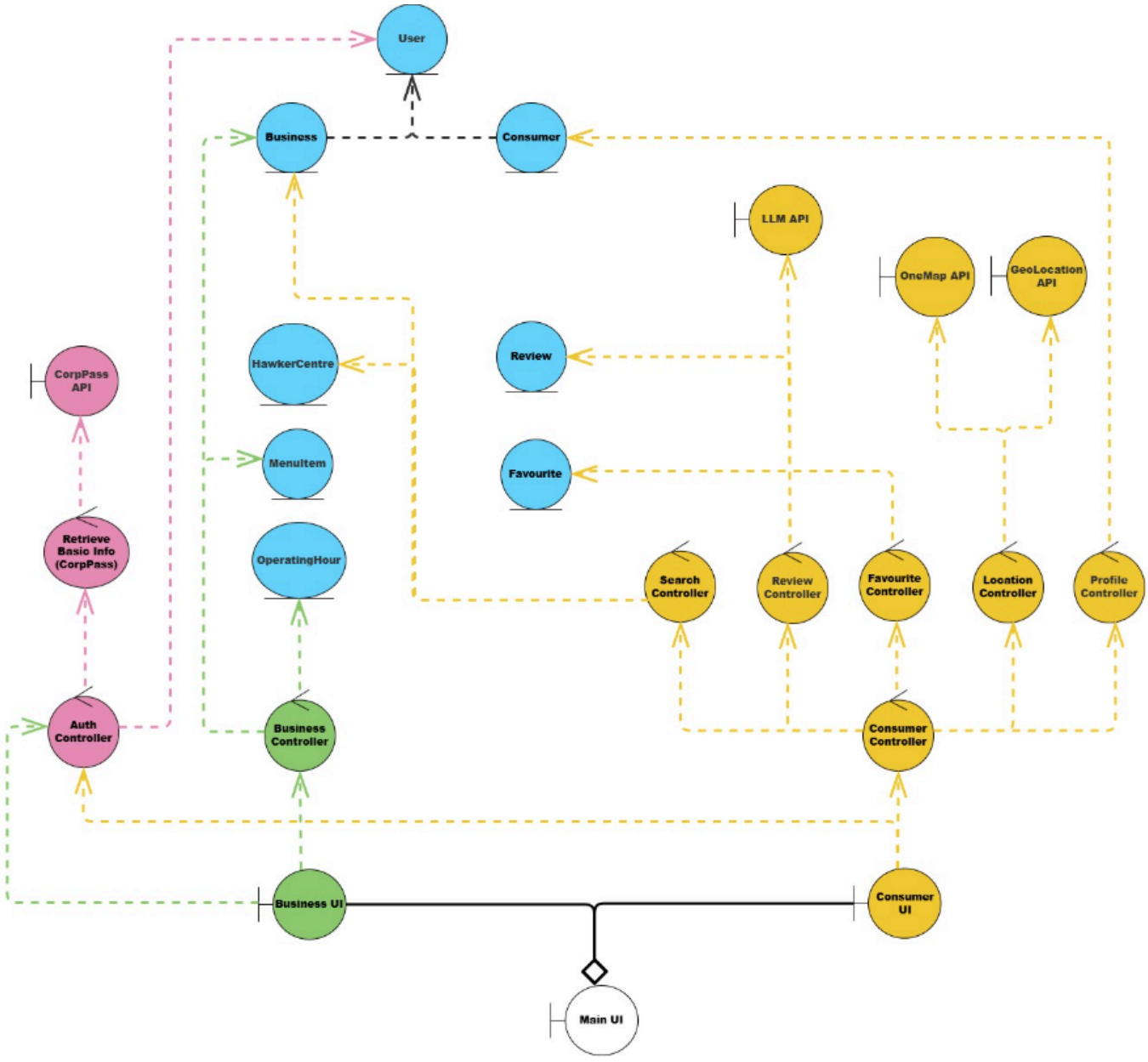

ConsumerUI
+openBusinessProfile()
+writeReview()
+submitReview(starRating, description, photo)
+selectStarRating(rating)
+enterDescription(description)
+uploadPhoto(photo)
+clickUpload()
+updateProfileForm()
+displayProfileUpdate()
+displayReviewForm()
+displayReviewConfirmation(photoPreview)
+disableTyping()
+showError(message)
+keepDraft()
+confirmationMessage()
+returnSuccessMessage()
+searchBusiness(query)
+displaySearchResults(results)
BusinessUI
+signupBusiness()
+loginBusiness()
+manageBusinessProfile()
AuthController
+authenticateWithCorpPass()
SearchController
+searchBusiness(query)
+filterResults()
+displaySearchResults(results)
BusinessController
+signupBusiness()
+validateBusinessInfo()
+registerBusiness()
+loginBusiness()
+updateBusinessProfile()
User
-fullName : String
-email : String
-password : String
-username : String
+updateEmail(newEmail)
+changePassword(newPassword)
+updateProfilePhoto(newPhoto)
+validatePasswordPolicy()
ProfileController
+updateProfileForm()
+displayProfileUpdate()
+updateEmail(newEmail)
+updatePassword(newPassword)
+validatePasswordPolicy()
+updateProfilePhoto(newPhoto)
+updateProfileInfo()
+returnSuccessMessage()
+showError(message)
ConsumerController
+registerConsumer()
+loginConsumer()
+manageConsumerProfile()
LocationController
+getNearbyBusinesses()
+useGeoLocation()
ReviewController
+writeReview()
+displayReviewForm()
+submitReview(starRating, description, photo)
+enterDescription(description)
+uploadPhoto(photo)
+clickUpload()
+filterReview(description, photo)
+truncateInput()
+disableTyping()
+showError(message)
+keepDraft()
+confirmationMessage()
+updateAggregateRatings()
CorpPassAPI
+retrieveBusinessInfo()
Business
-status : String
-description : String
-photo : String
-uen : String
+setStatus()
+updateDetails()
+addMenuItem()
+removeMenuItem()
+setOperatingHours()
+addReview(description, rating, photo)
+reviewStored()
+updateAggregateRatings()
+signupBusiness()
+loginBusiness()
+updateBusinessProfile()
Consumer
-profilePhoto : String
-recentlySearched : String
+addFavourite()
+removeFavourite()
+addRecentSearch()
+writeReview()
+clearAll()
+updateEmail(newEmail)
+updatePassword(newPassword)
+updateProfilePhoto(newPhoto)
+updateProfileInfo()
+emailUpdated()
+passwordUpdated()
+photoUpdated()
+profileUpdated()
FavouriteController
+addFavourite(businessId)
+removeFavourite(businessId)
+getFavourites()
GeoLocationAPI
+getCoordinates()
+findNearby()
LLMAPI
+filterReview(description, photo)
MenuItem
-photo : String
-name : String
-price : Decimal
-description : String
+updatePrice()
+updateDescription()
+updatePhoto()
+getMenuDetails()
OperatingHour
-day : String
-startTime : Time
-endTime : Time
+setHours()
+getOpeningHours()
Review
-starRating : Integer (1..5)
-description : String (0..250)
-images : String
+edit()
+isValid(description)
+truncateInput()
+reviewStored()
Favourite
-type : String
+isBusinessFavourite()
+isHawkerCentreFavourite()
+addFavourite()
+removeFavourite()
HawkerCentre
-name : String
+getStalls()
1
0..*
0..1

2. Design Model

2.1 Class Diagram of Entity Classes

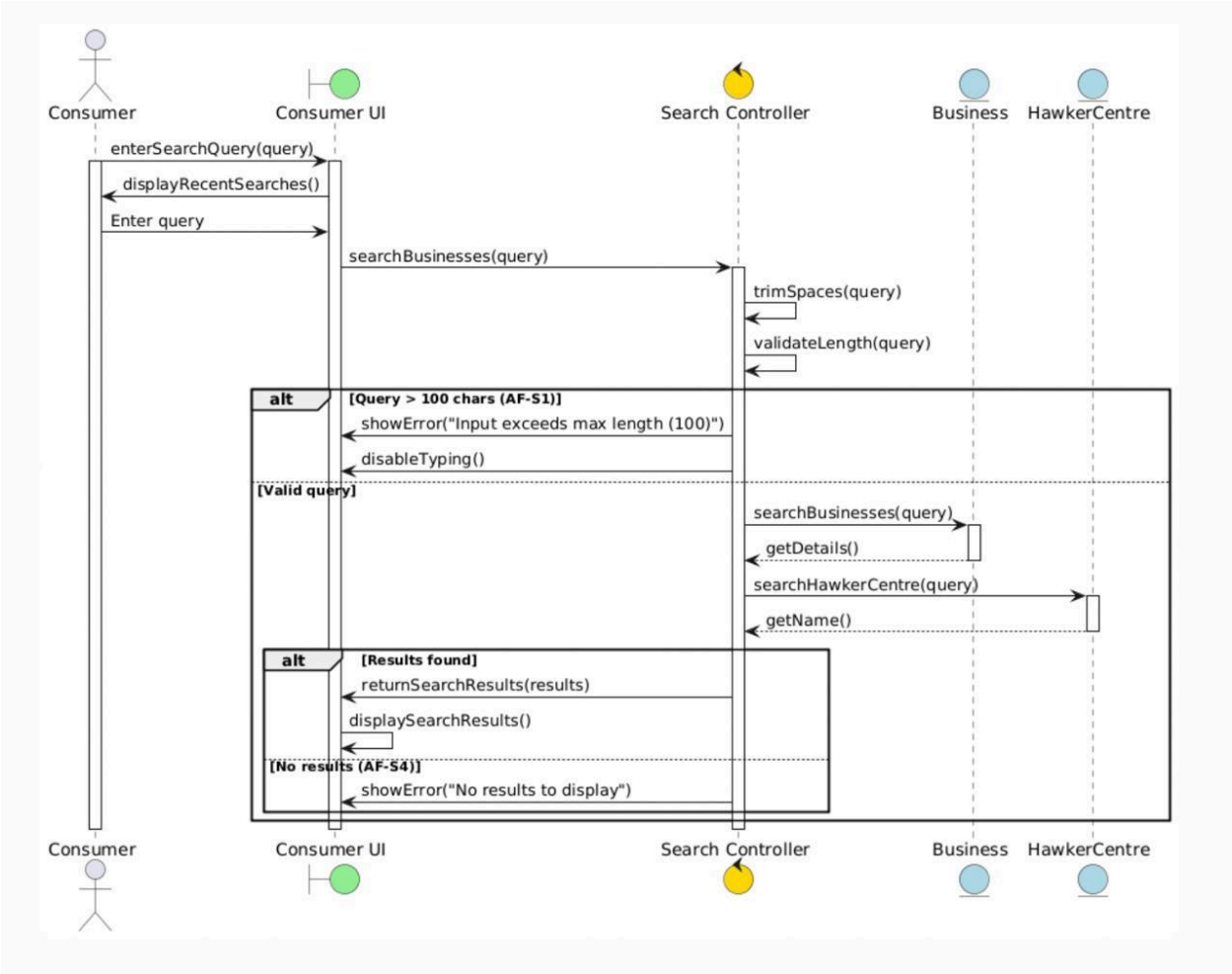


2.2 Key Boundary Classes and Control Classes



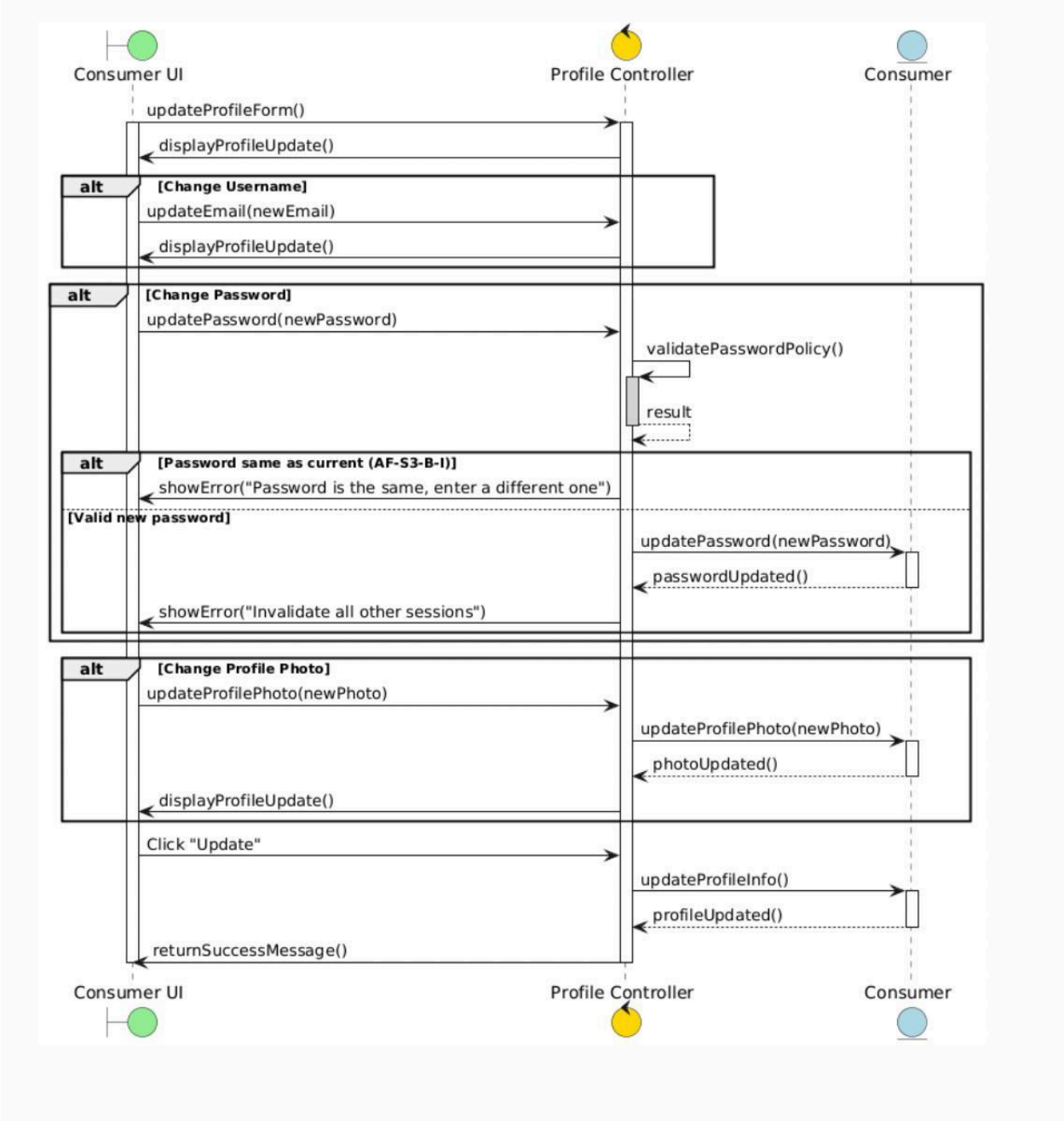
2.3. Sequence Diagram of the Use Cases

2.3.1 Search Use Case:

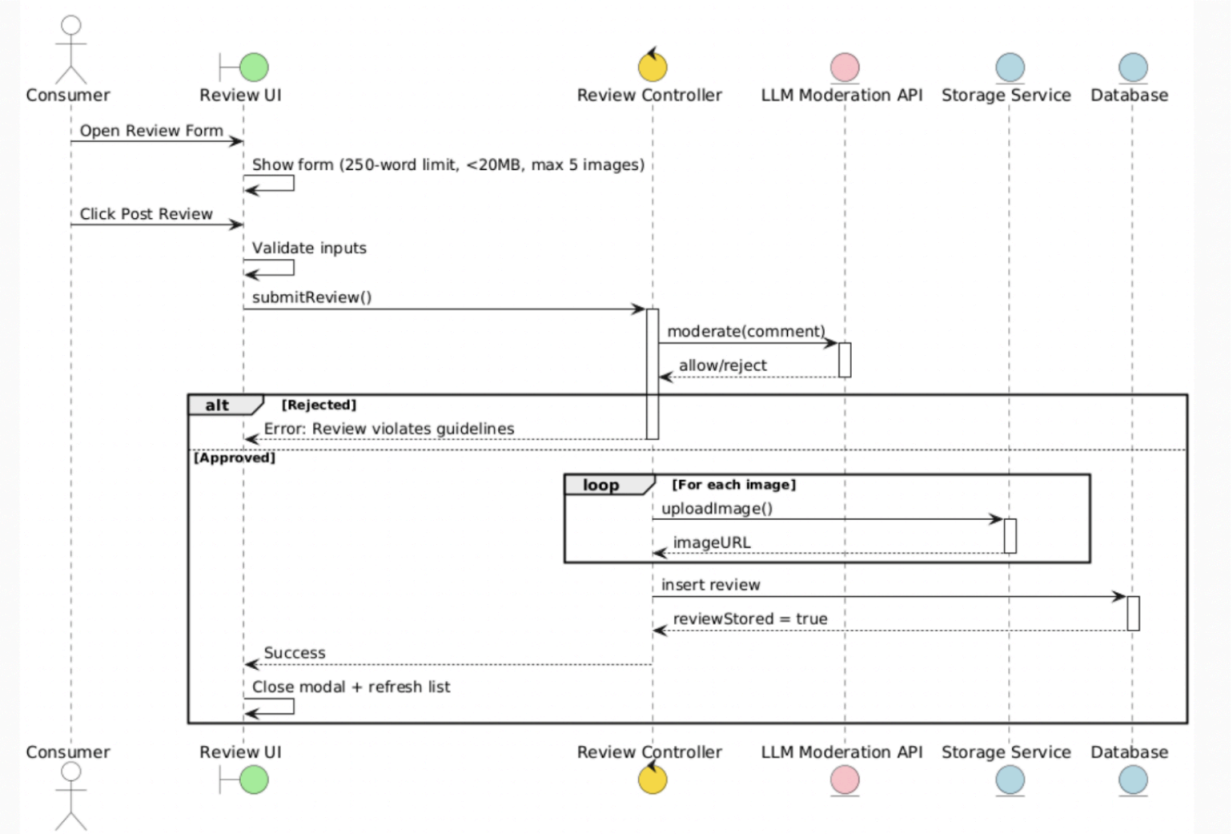


2.3.2 Business Sign Up:

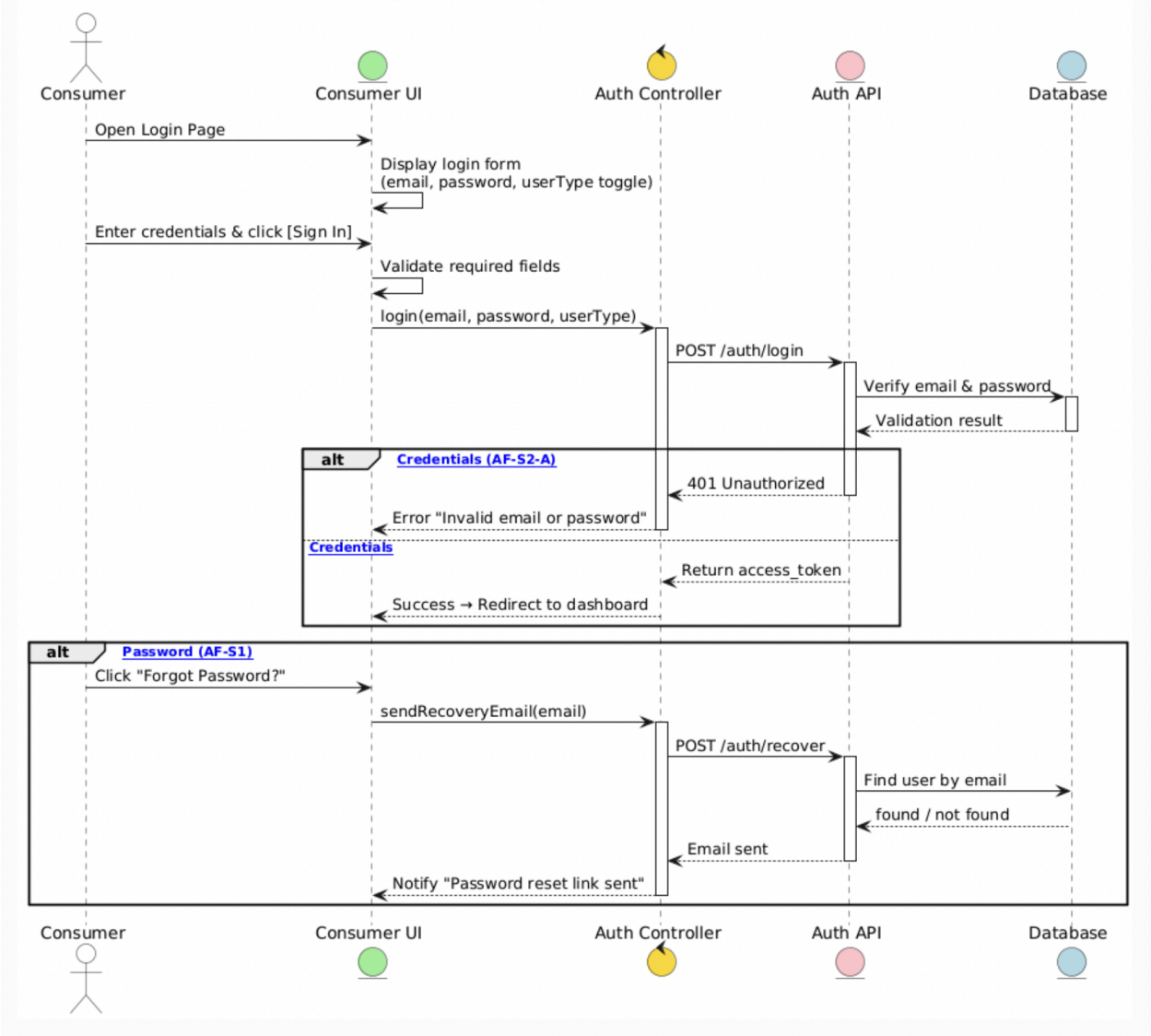
2.3.3 Manage Consumer Profile



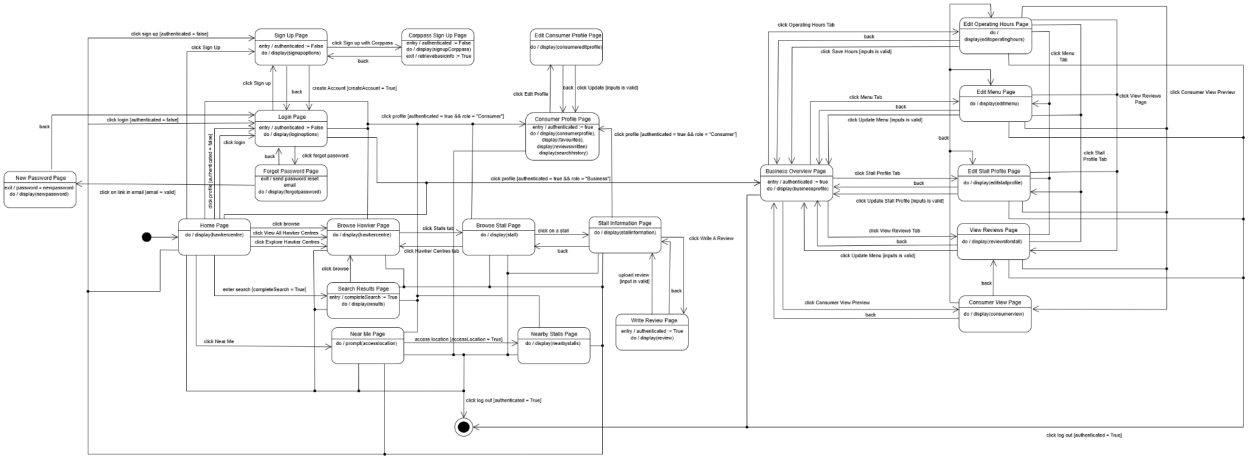
2.3.4 Upload Reviews



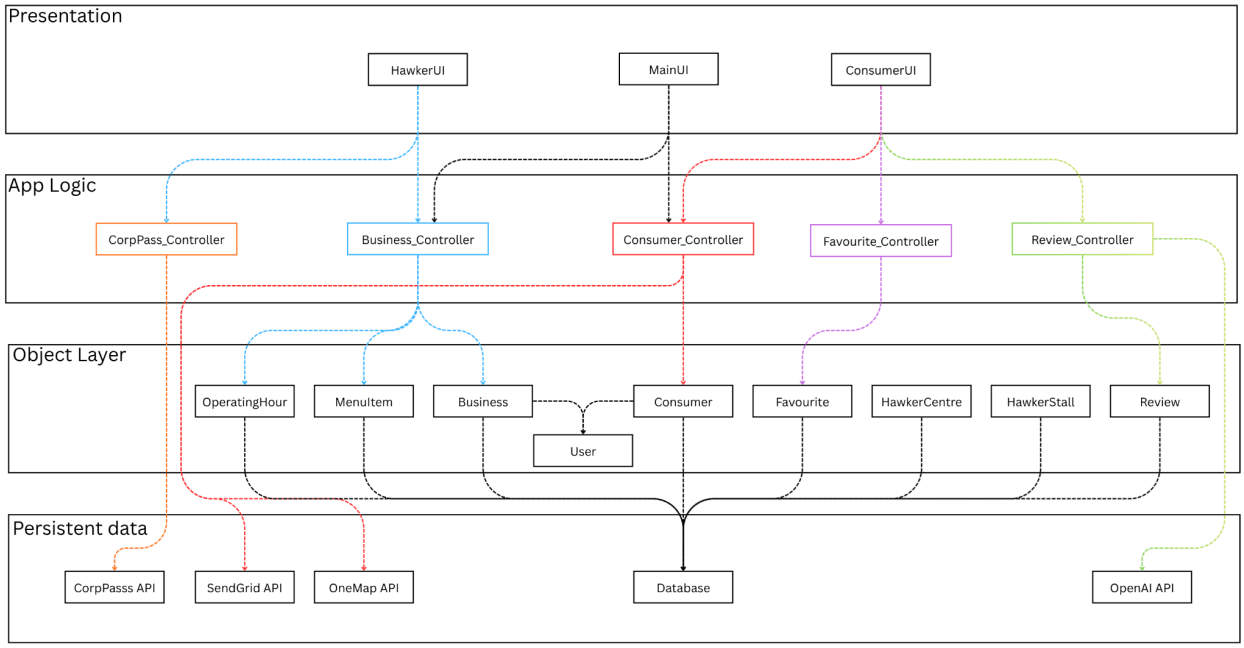
2.3.5 Log In


Consumer
Consumer UI
Auth Controller
Auth API
Database
Open Login Page
Display login form
(email, password, userType toggle)
Enter credentials & click [Sign In]
Validate required fields
login(email, password, userType)
POST /auth/login
Verify email & password
Validation result
alt
Credentials (AF-S2-A)
401 Unauthorized
Error "Invalid email or password"
Credentials
Return access_token
Success → Redirect to dashboard
alt
Password (AF-S1)
Click "Forgot Password?"
sendRecoveryEmail(email)
POST /auth/recover
Find user by email
found / not found
Email sent
Notify "Password reset link sent"
Consumer
Consumer UI
Auth Controller
Auth API
Database

2.4. Dialog Map



3. System Architecture



<u>Presentation Layer</u>
This layer is responsible for the interaction between users and the system. Each user interface component will call the respective controllers from the App Logic Layer to execute business logic.

This layer consists of:

1. **MainUI**
   This is the main screen for Users. Users can access all the common apps features available without logging in to their role such as login, sign up and browsing stalls or hawker centres.
2. **ConsumerUI**
   Consumers can do various tasks specific that are accessible only to them such as favourites and submitting reviews.
3. **HawkerUI**
   Shows screens that is only accessible to the Business such as screens to manage BusinessProfile

App Logic Layer

This layer contains all controller classes which encapsulates the core business rules and workflows. It validates requests from the presentation layer before interacting with the object layer.

This layer consists of:

1. **Business_Controller**
   The Business_Controller is called by MainUI, to display existing businesses, and by HawkerUI for business management. Controller logic includes business registration and profile management.
2. **Consumer_Controller**
   The Consumer_Controller is called by MainUI for account operations, e.g. to sign up or reset passwords, and by ConsumerUI for consumer login specific features. Controller logic includes consumer account management.
3. **Favourite_Controller**
   The Favourite_Controller is called by ConsumerUI to manage and view their favourites. Controller logic includes listing, adding and removing favourites.
4. **Review_Controller**
   The Review_Controller is called by ConsumerUI for review management and by MainUI for review display. Controller logic includes review management and listing.
5. **CorpPass_Controller**
   The CorpPass_Controller is called by the HawkerUI during business account registration to verify business entities via the CorpPass system. Controller logic includes managing authentication flow and validating API responses.

App Object Layer

This layer contains the entity classes that represent the data structure of the system. These entities are stored in the Persistent Data Layer and are used by the App Logic Layer.

This layer consists of:

1. **User**
   Contains email, username, password, a reset token, and a token expiration field to securely manage temporary authentication actions such as password resets.
2. **Consumer**
   Contains profile picture, recently searched, and relationship with favourite and review.
3. **Review**
   Represents user reviews, containing text, rating, and associated userID and stallID.
4. **Favourite**
   Represents the relationship between a user and their favourite hawker stalls, containing the userID, stallID
5. **HawkerCentre**
   Contains static information about hawker centres such as name,
   address, latitude, longitude, and a list of stalls.

6. **HawkerStall**
   Represents each stall in a hawker centre, containing fields stallName, ratings, centreID, stallName
7. **Business**
   Represents business owner profiles. Contains SFA License, Stall details, Location and operational status. Relationship with MenuItem and OperatingHour.
8. **MenuItem**
   Represents food items in the business' menu. Contains name, pricing and photo.
9. **OperatingHour**
   Represents business operating schedule. Contains a weekly timetable with start and end timings for each day.

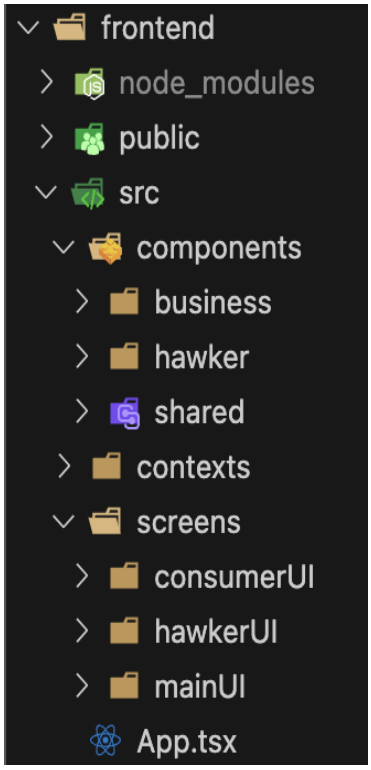
Persistent Data Layer

This layer contains the database that stores all the application entities.

4. Application Skeleton

Do refer to the source code uploaded in GitHub for the application skeleton.

A. Frontend



The frontend (React + TypeScript) mainly consists of the different **User Interfaces (Screens)**, which are structured and categorised into the 3 screens: **ConsumerUI**, **HawkerUI**, and **MainUI**, depicted in the class diagrams.

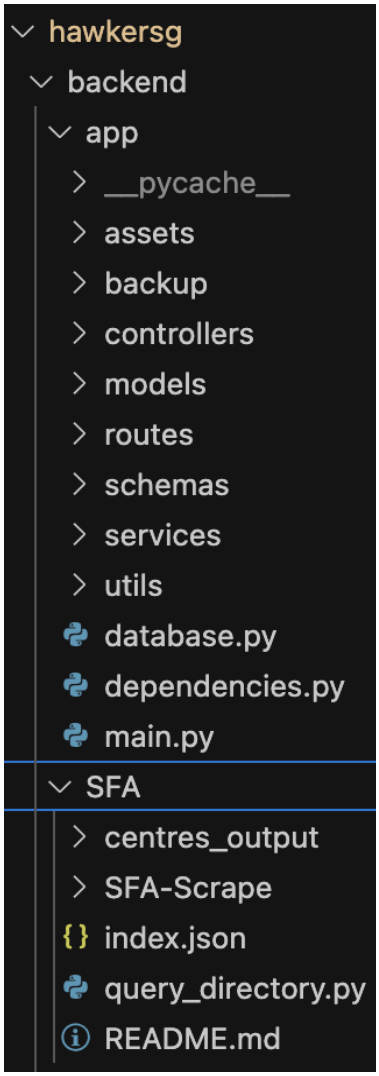
Each UI screen folder contains multiple sub-screens and components that handle specific page views such as search, stall display, and hawker dashboard.

The file **App.tsx** serves as the entry point of the frontend application. It is responsible for rendering the global providers (contexts) and configuring the routing system between different UI pages using React Router.

Other folders like components and contexts contain helper and reusable modules that make the frontend codebase more organized, maintainable, and collaboration-friendly:

- components: contains modular UI components grouped by their respective feature types. Can be generally categorised into 3 folders
    - business: business-related cards and views
    - hawker: hawker centre and stall cards
    - shared: generic reusable UI elements such as buttons, modals, and layout components
- contexts: Contains data and authentication related information to link Backend to Frontend.
- screens: Groups the main user interfaces for organisation and maintainability.
    - consumerUI: Consumer profile page, reviews and editing profile.
    - hawkerUI: Business profile and the dashboard
    - mainUI/: main general functionality such as Login, Search, Home, Near Me

B. Backend



Models: Contains the business objects representing database entities such as User, Review, Favourite, Business, Consumer, and Hawker Center

Schemas: Defines the request and response data structures using Pydantic for validation before processing

Services: Contains the logic for handling and storing business data.

Controllers: Act as an intermediate layer between routes and services, processing incoming requests and coordinating responses.

Routes: Contains the REST API endpoints that connect the frontend to the backend.

Assets: Organized into subfolders to store profile images, stall images, menu images, and other related resources for efficient and structured asset management.

Utils: Hold reusable, generic helper functions (like those for email or JWT handling)

database.py: Entry point that sets up and connects to the database used to store all entities.

dependencies.py: Defines shared dependencies like database sessions and authentication.

SFA Folder: Handles hawker data integration

main.py: The entry point for running the FASTAPI backend and registering all API routes.