HAWKERSG

SCED GROUP 3

LECK KYE-CIN, LEE JUN DE, KAVAN, LEE YONG LIANG
, LEE ZI AN, LIM SEOW KIAT, LIM XIAO XUAN

TABLE OF CONTENTS

HOW HAWKERSG CAME ABOUT

THE PROBLEM

Inaccurate and Fragmented Hawker Information Online

1. No centralized directory for hawker centres in Singapore
2. Data on Google Maps is user-driven and often outdated
3. Hard to locate specific stalls
4. Hawker culture is underrepresented digitally

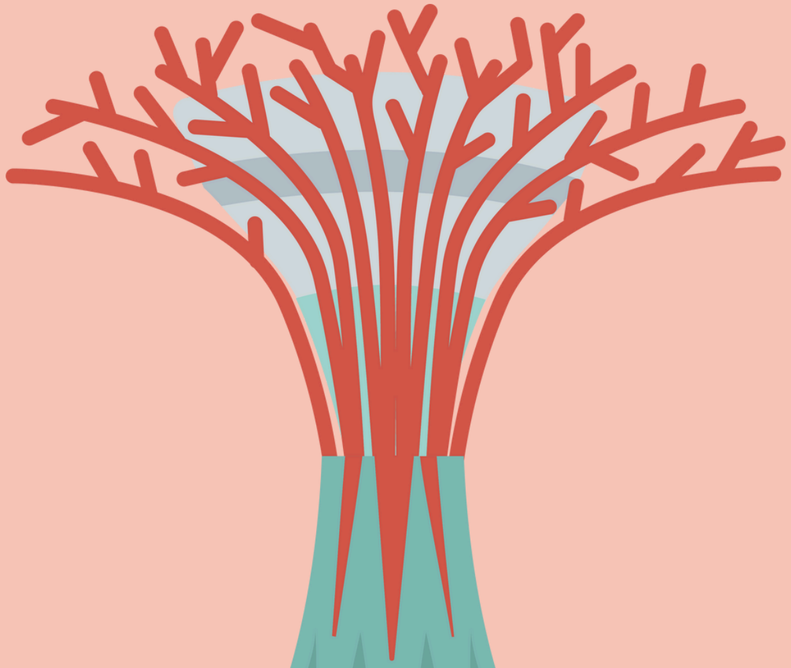
OUR SOLUTION

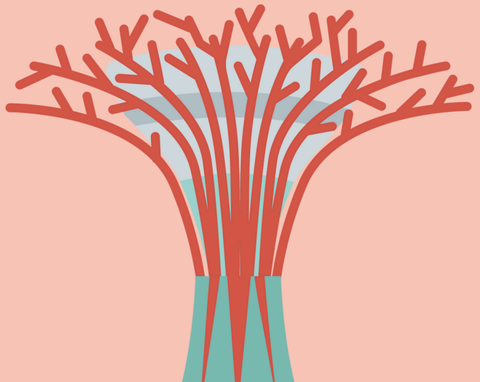
HAWKERSG

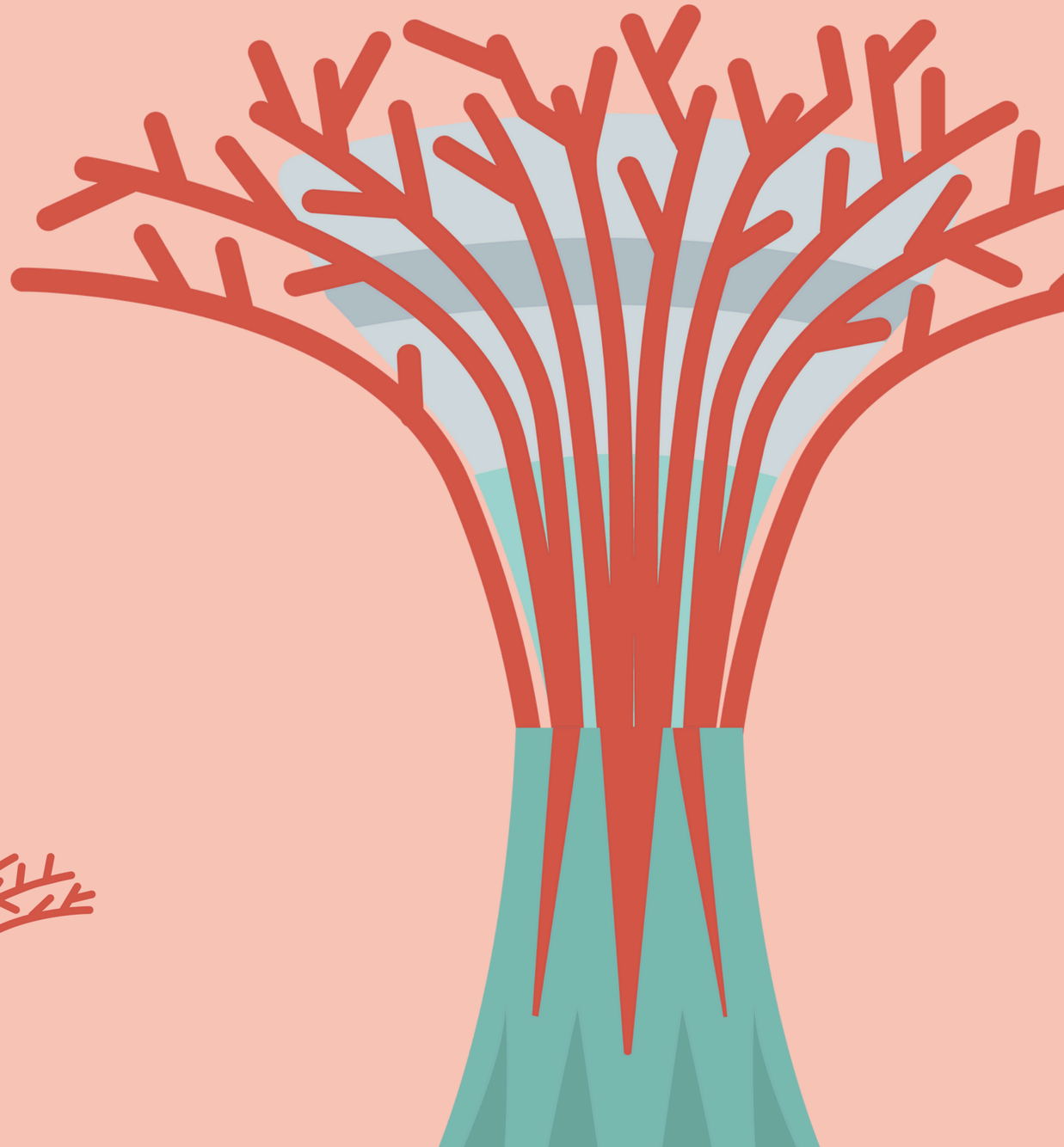
Verified Data, Centralized Access:

1. **A centralized digital directory for Singapore's hawker centres**
2. **Built using Gov Data APIs and SFA licensing data**
3. **Ensures accuracy compared to user-based platforms**
4. **Designed for both locals and tourists**

KEY FEATURES







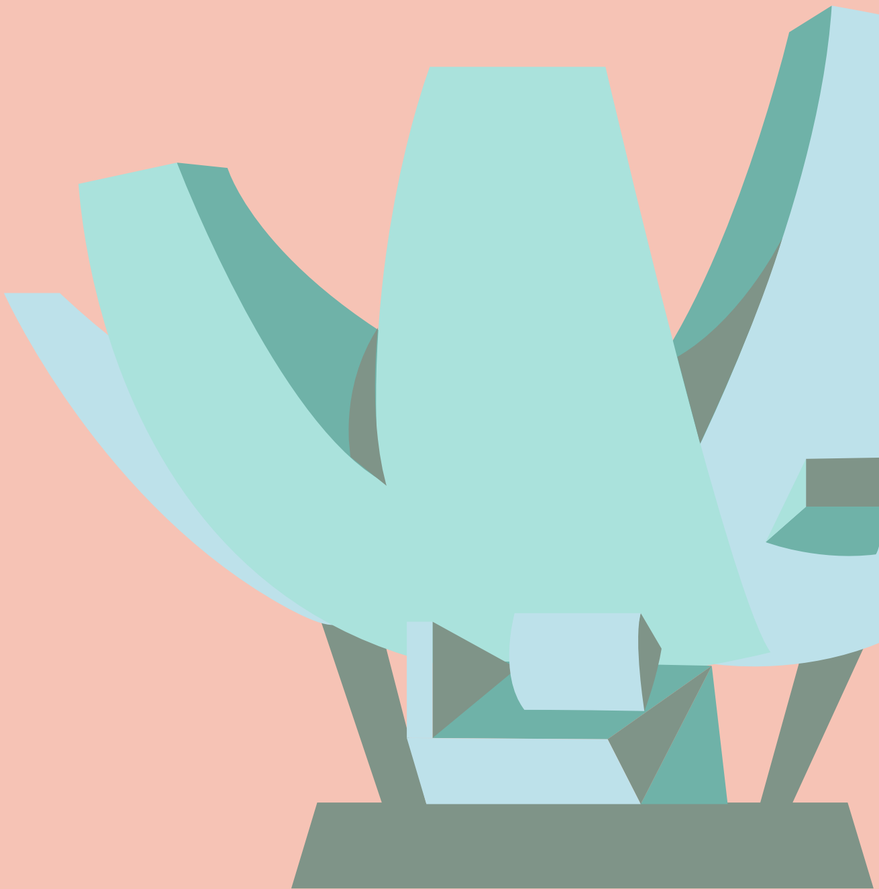
OUR KEY FEATURES

- Authentication
- Hawker Management
- Consumer Features
- Reviews
- Location Servcies
- Profile Management

FEATURE LIST

- **Authentication**
  a. **Signup for Consumer and Business users**
  b. **Login via email or CorpPass**
  c. **Secure session management**

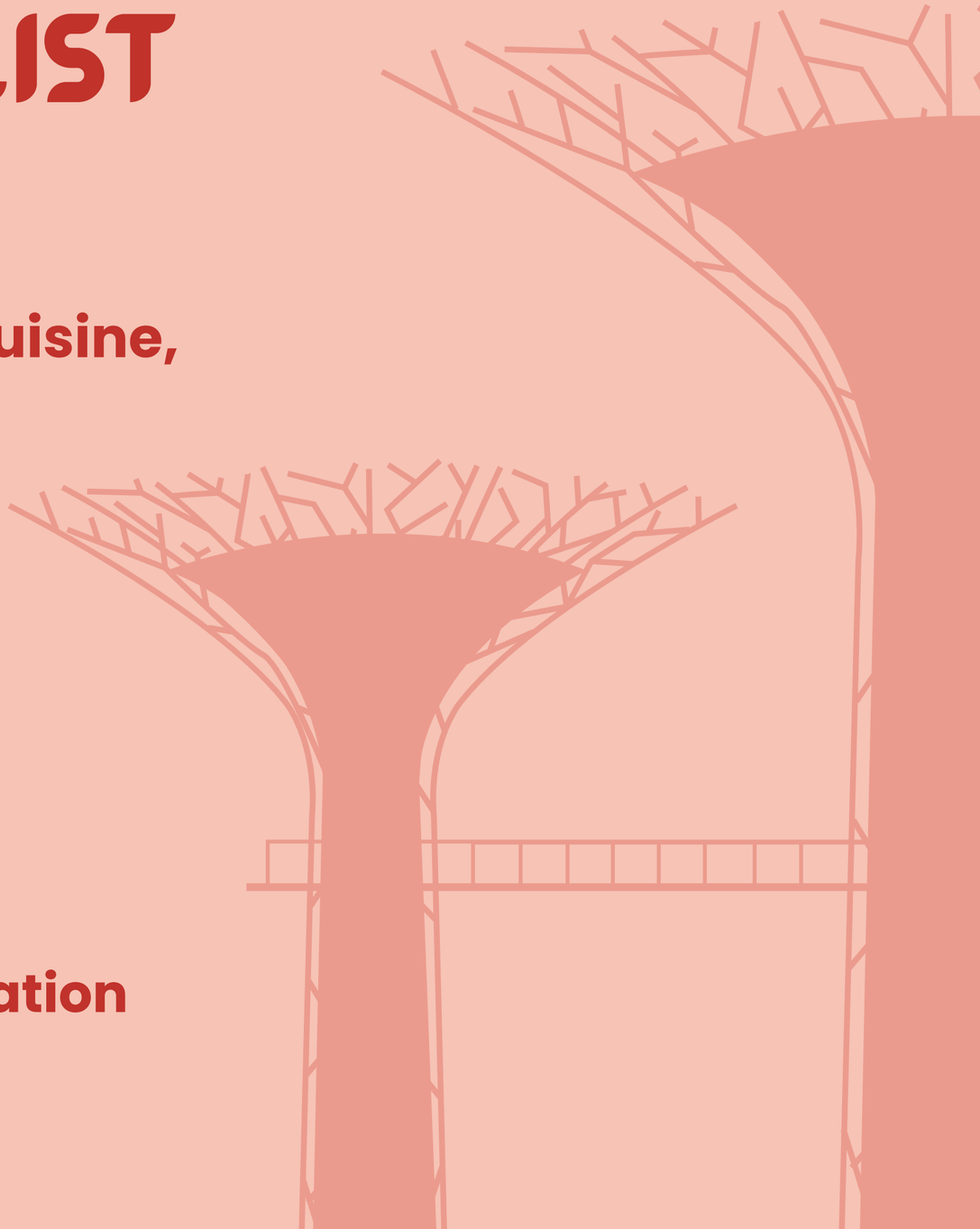
- **Hawker Management**
  a. **View and manage hawker stall listings**
  b. **Update stall details and operating hours**



FEATURE LIST

- **Consumer Features**
  a. **Search and view hawkers by name, cuisine, location, or ratings**
  b. **Add to Favourites for easy access**
  c. **View live map of nearby hawkers**

- **Reviews**
  a. **Add and edit reviews**
  b. **LLM-powered review filtering for moderation**



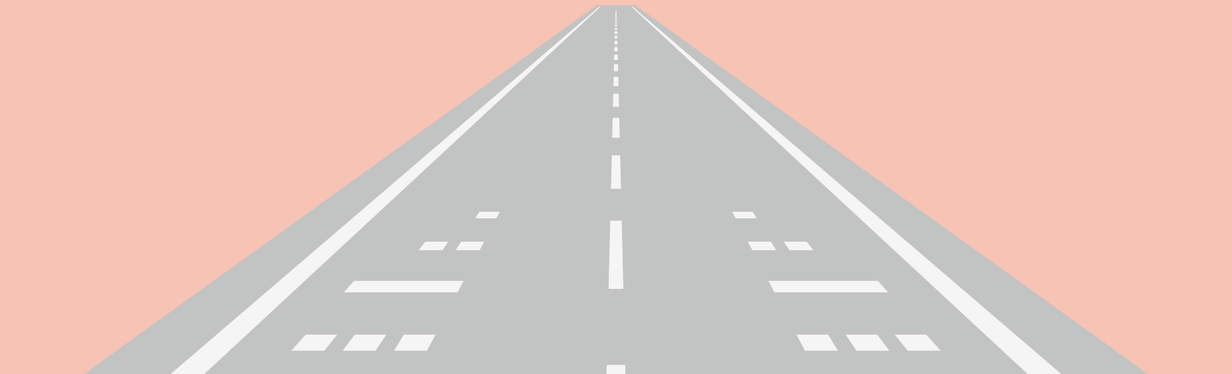
FEATURE LIST

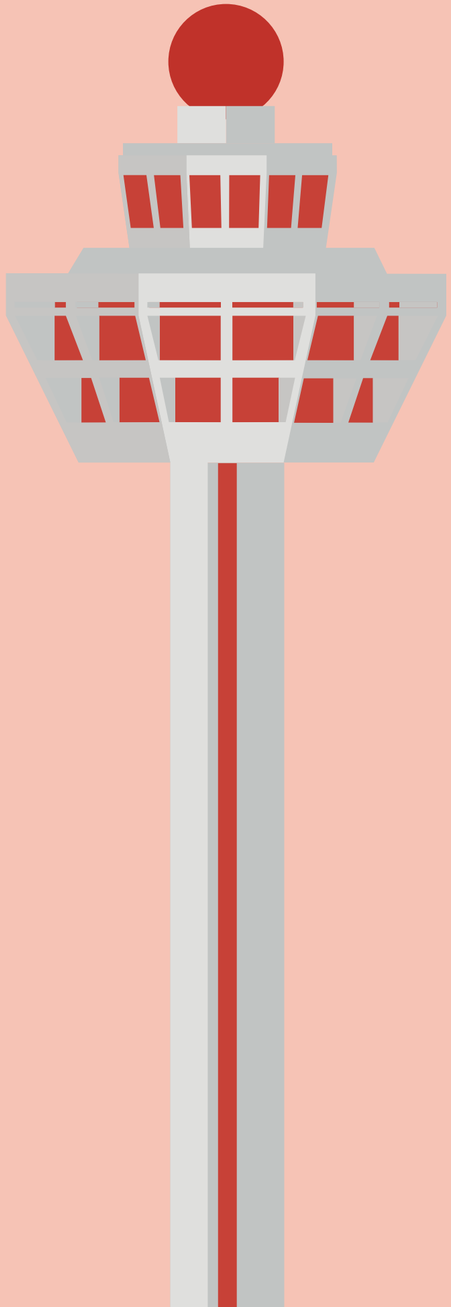
- **Live Location**
  a. **Access live user location via geolocation API**
  b. **Display nearby hawker centres within a specified radius**

- **Profile Management**
  a. **Manage consumer and business profiles**
  b. **View account activity and saved stalls**

LIVE DEMO

TECHNOLOGY STACK





HAWKERSG TECH STACK

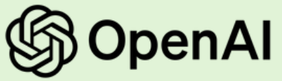
Frontend

Backend

Database

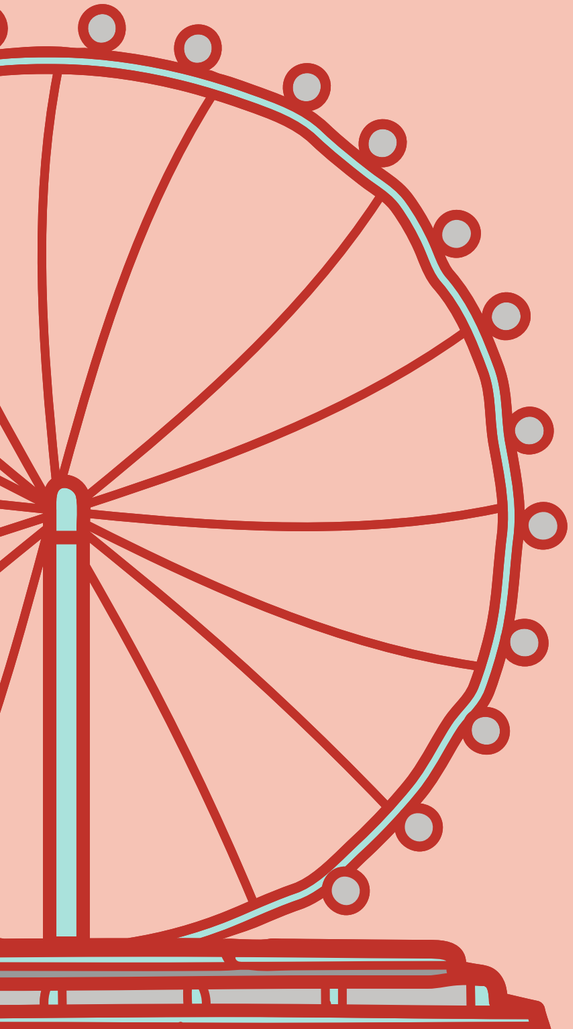
DevOps

Services



EXTERNAL API

BROWSER GEOLOCATION API	Gets user’s current coordinates (latitude & longitude) for location-based search.
ONEMAP API	Used for reverse geocoding. Converts coordinates into area names like Yishun or Sengkang.
SENDGRID API	Reliably sends emails for password resets and notifications, minimizing server overhead.
OPENAI API	Used for review moderation. It filters spam or inappropriate content.
DATA.GOV.SG	Supplies official NEA data of hawker centres for accurate, verified listings.



SOFTWARE ENGINEERING PRACTICES

SWE PRACTICES

Collaborative Development

Pair programming for some of the frontend and backend code for shared problem-solving and deeper understanding

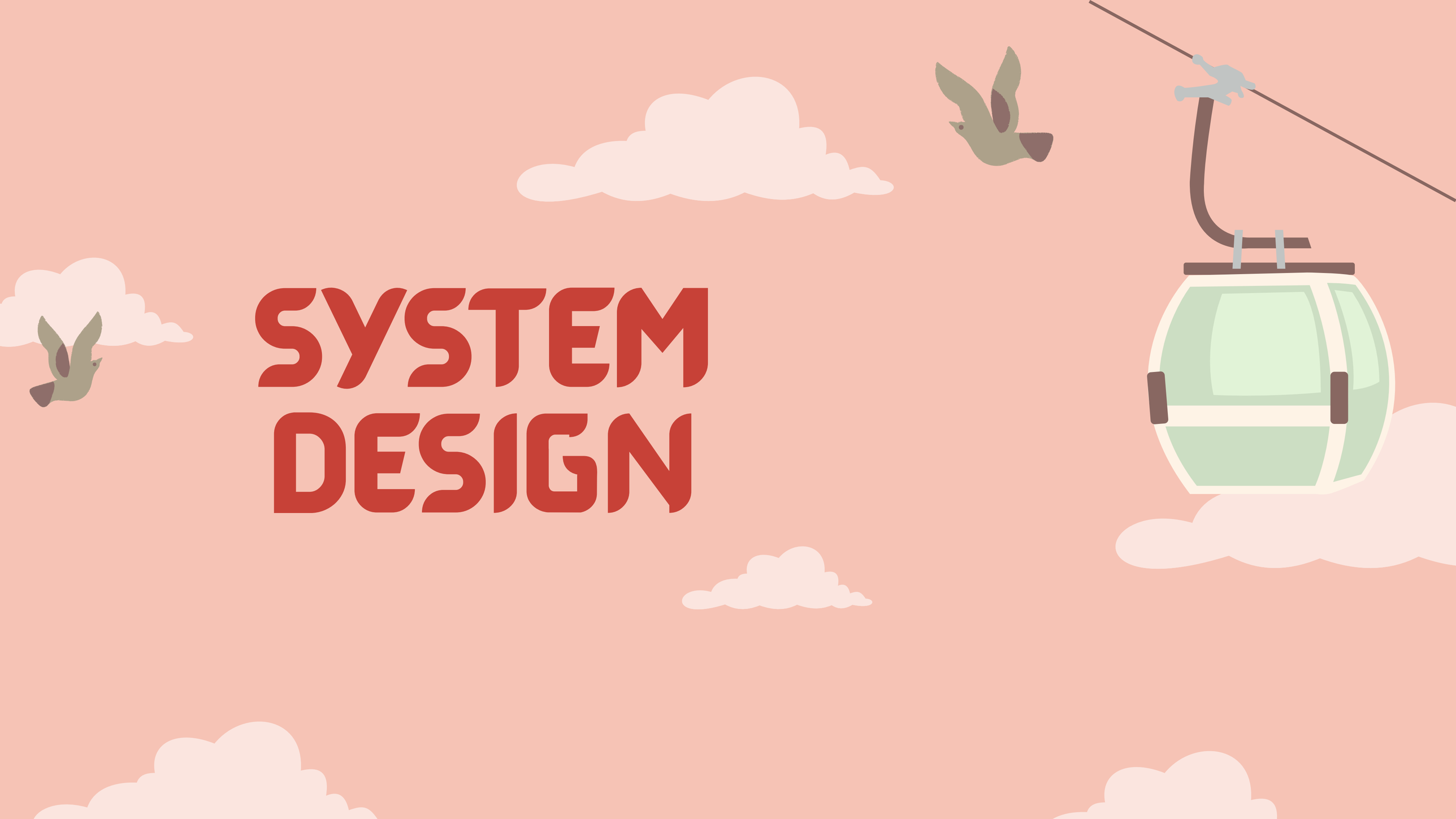
Agile Practices

Flexible and iterative development

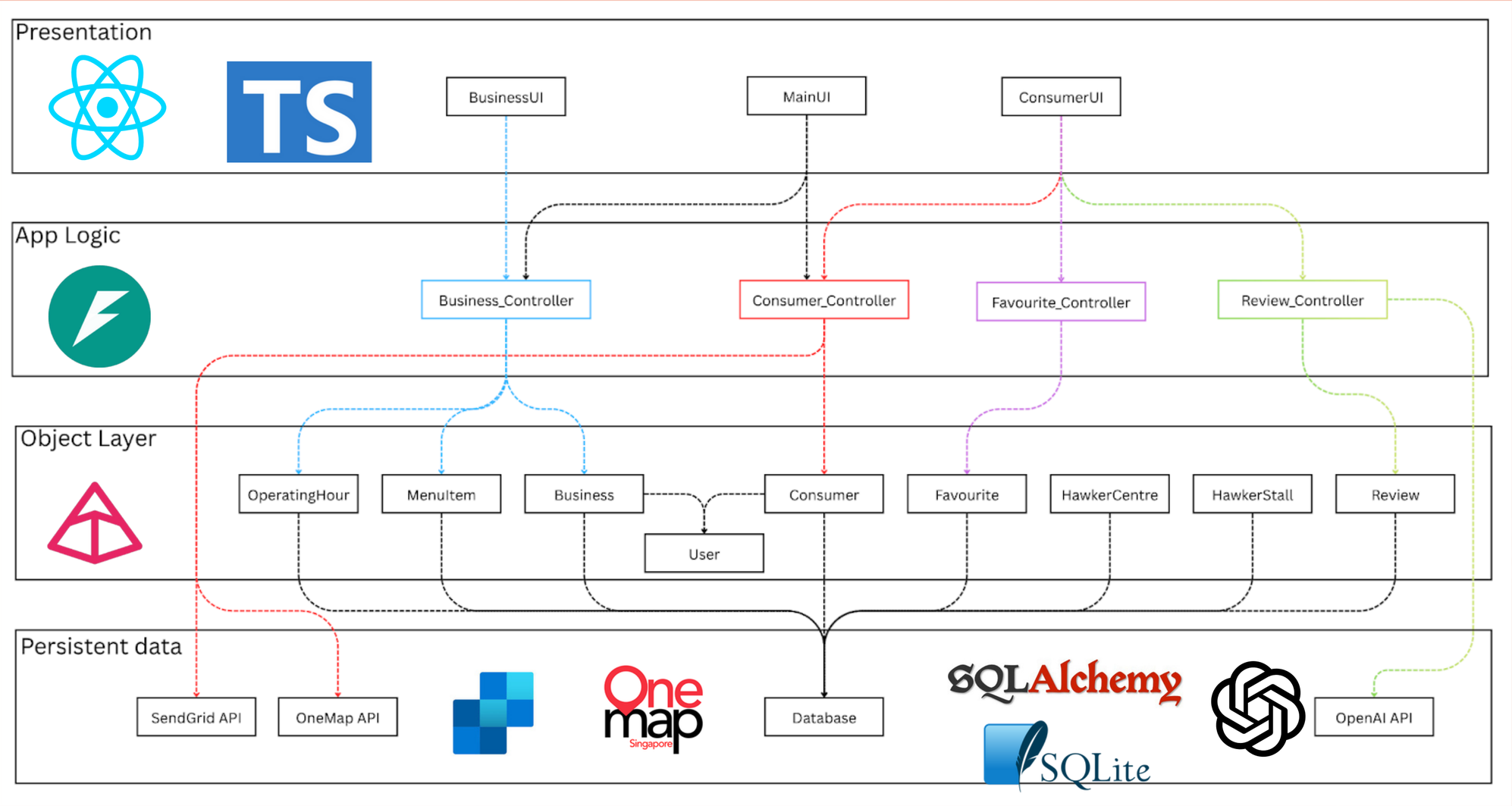
Continuous Delivery

Frequent releases

SYSTEM DESIGN



LAYERED ARCHITECTURE



DESIGN PATTERNS

FACADE PATTERN

Unified Interface

AuthContext

```tsx
interface AuthContextType {
  user: User | null;
  loading: boolean;
  authToken: string | null;
  login: (email: string, password: string, userType: 'consumer') => Promise<void>;
  businessLogin: (email: string, password: string, userType: 'business') =>
Promise<void>;
  signup: (email: string, password: string, name: string, user_type: 'consumer' |
'business') => Promise<void>;
  logout: () => void;
  forgotPassword: (email: string) => Promise<void>;
  resetPassword: (token: string, password: string) => Promise<void>;
  updateProfile: (data: FormData) => Promise<void>;
  updateUserLocalState: (updates: Partial<User>) => void;
}
```

Complex Operations

Simplified Usage

LoginPage

```tsx
// Import the facade
const { login, businessLogin } = useAuth();

// Simple usage - all complexity is hidden
await login(email, password, 'consumer');
await businessLogin(email, password, 'business');
```

SignUpPage

```tsx
// Import the facade
const { signup } = useAuth();

// Simple usage - all complexity is hidden
await signup(formData.email, formData.password, formData.name, formData.userType);
```

ForgotPasswordPage

```tsx
// Import the facade
const { forgotPassword } = useAuth();

// Simple usage - all complexity is hidden
await forgotPassword(email);
```

STRATEGY PATTERN

FrontEnd

AuthContext

Consumer Login Strategy

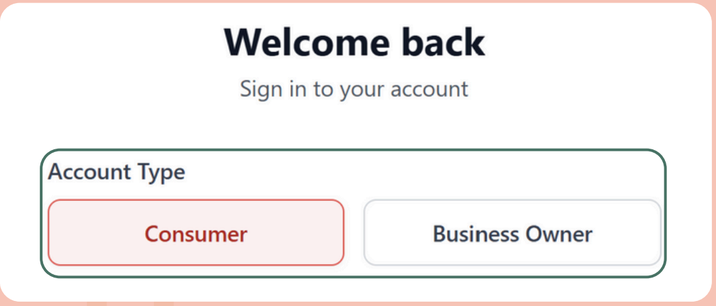
Business Login Strategy

Backend API

/consumer/login Endpoint

/business/login Endpoint

Runtime Selection



AuthContext

```
login: (email: string, password: string, userType: 'consumer') => Promise<void>;
businessLogin: (email: string, password: string, userType: 'business') => Promise<void>;
```

```
const login = async (email: string, password: string, userType: 'consumer'):
Promise<void> => {
  if (userType !== 'consumer') throw new Error('Invalid user type for consumer login.');

  const url = `${API_BASE_URL}/consumer/login`;
  // ... authentication logic for consumers ...
};
```

/consumer/login

```
const businessLogin = async (email: string, password: string, userType: 'business'):
Promise<void> => {
  if (userType !== 'business') throw new Error('Invalid user type for business login.');

  const url = `${API_BASE_URL}/business/login`;
  // ... authentication logic for businesses ...
};
```

/business/login

FACTORY PATTERN

LoginPage (Client)

Needs to create user session

AuthContext (Factory)

login()

Creates:
Consumer Session

businessLogin()

Creates:
Business Session

Session Objects (Products)

Consumer Session

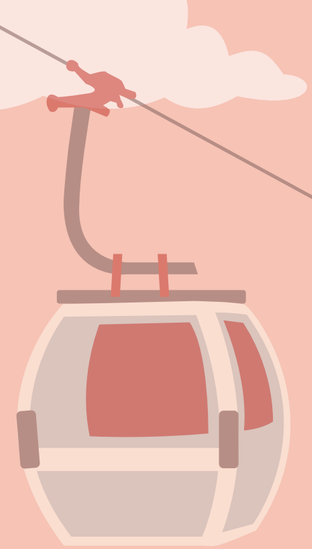
- user: Consumer
- token: JWT
- stored in:
  - React state
  - localStorage
- routes to: /

Business Session

- user: Business
- token: JWT
- stored in:
  - React state
  - localStorage
- routes to: /business

TRACEABILITY & TESTING

TRACEABILITY



FUNCTIONAL REQUIREMENTS

Use Case Descriptions & Diagram

REQUIREMENTS ANALYSIS

Class & Sequence Diagram

IMPLEMENTATION

Design Principles

FUNCTIONAL REQUIREMENT: LOGIN

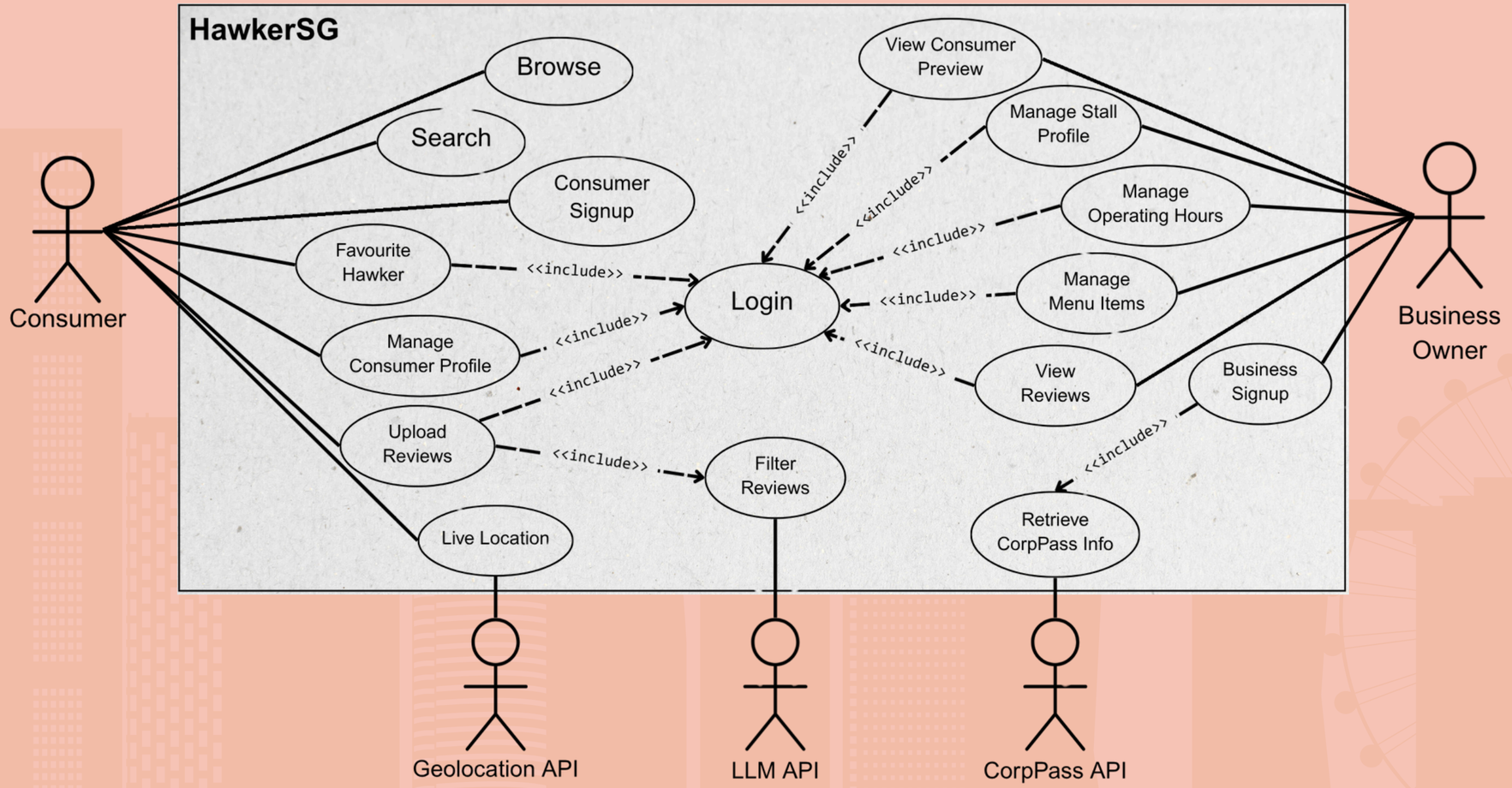
Main Flow

Actor:	Consumer and Business Owner
Description:	None
Preconditions:	User already has an account created.
Postconditions:	User is granted access to respective role after being authenticated.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. User is shown a login form when the User clicks the “Login” button. 2. User clicks on the respective Account Type button, “Consumer” or “Business Owner”, to log into the specific account role. 3. User enters valid credentials in the following fields and submits them. a. Email b. Password 4. System compares data inside the database against the user inputs. 5. User is logged into the chosen account type and is redirected to (a) the homepage for Consumer or (b) the Business Profile page for Business Owner.

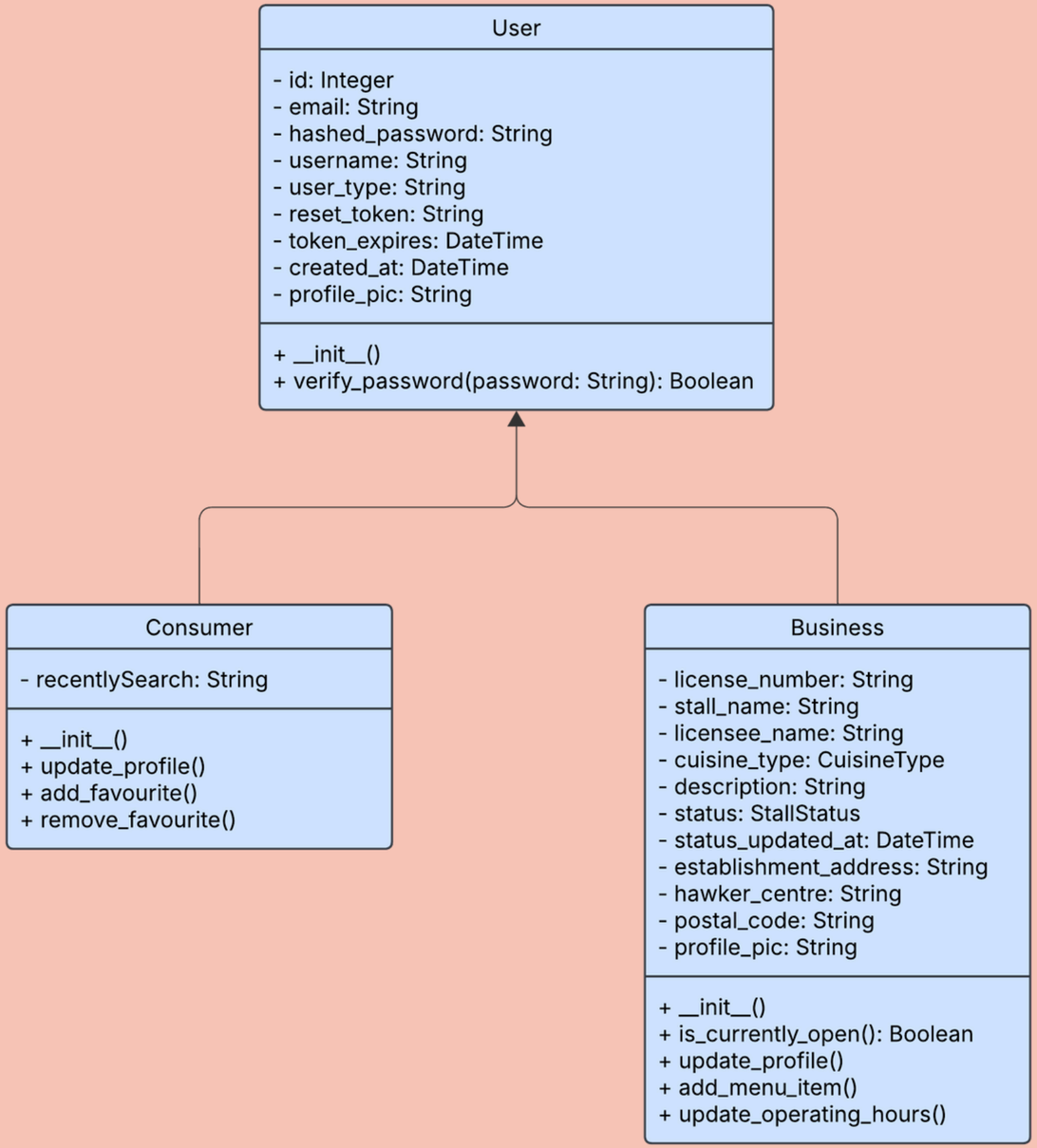
Alt Flow

Alternative Flows:	AF-S1: Forgot Password 1. Send a password recovery email if the user selects "Forgot Password". 2. Redirect back to the login page, user continues from S2. AF-S3-A.a: Empty Field for Email. 1. System displays error message “Please fill out this field.” 2. User continues from S3. AF-S3-A.b: Invalid Email Format. 1. System displays error message “Please include an @ in the email address.” or “Please enter a part following ‘@’.” 2. User continues from S3. AF-S3-B: Empty Field for Password. 1. System displays error message “Please fill out this field.” 2. User continues from S3. AF-S4.a: Invalid Credentials (Email and/or password) 1. System asks the user to reattempt after showing the error message “Invalid email or password. Please check your credentials and account type.” 2. Redirect back to the login page, user continues from S2. 3. After 5 consecutive failed login attempts, lock the account, and inform the user via email to reset the password. AF-S4.b: Account Type does not match (Correct details for different role). 1. System asks the user to reattempt after showing the error message “Invalid email or password. Please check your credentials and account type.” 2. User continues from S3.

FUNCTIONAL REQUIREMENT: LOGIN



REQUIREMENT ANALYSIS: LOGIN



user_model

```python
class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    email = Column(String, unique=True, index=True)
    hashed_password = Column(String)
    username = Column(String)
    user_type = Column(String) # Discriminator column
    reset_token = Column(String, nullable=True)
    token_expires = Column(DateTime, nullable=True)
    created_at = Column(
        DateTime,
        nullable=False,
        server_default=text('CURRENT_TIMESTAMP')
    )
    profile_pic = Column(String, nullable=True)

    # Link for inheritance
    __mapper_args__ = {
        "polymorphic_identity": "user",
        "polymorphic_on": user_type
    }
```

consumer_model

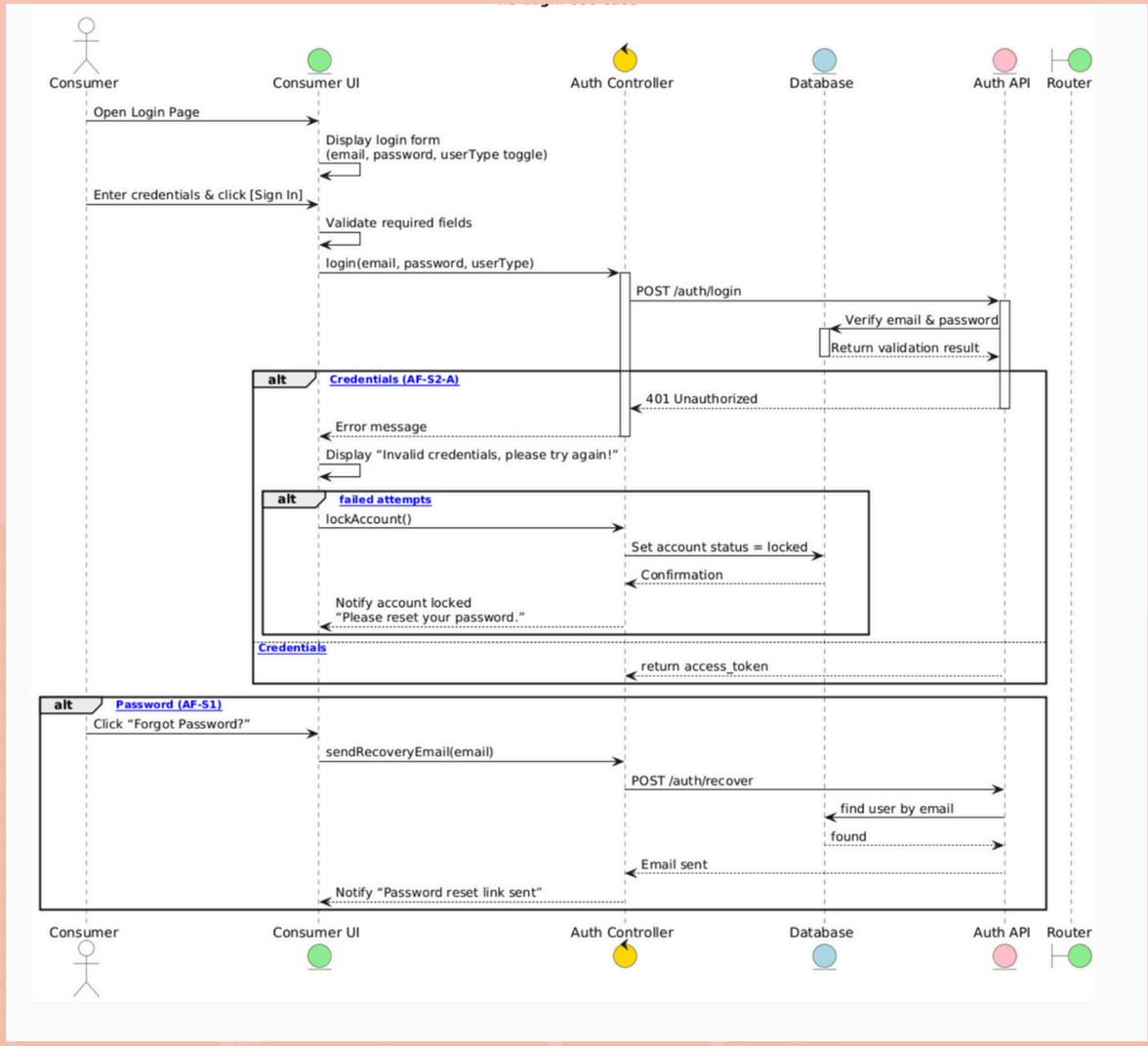
```python
class Consumer(User):
    __tablename__ = "consumers"
    id = Column(Integer, ForeignKey('users.id'), primary_key=True)

    # Consumer-specific field
    recentlySearch = Column(String, default="")

    favourites = relationship("Favourite", back_populates="consumer",
        # Optional: Set lazy='joined' or 'selectin' for more efficient loading
        # when fetching a Consumer and their favourites.
        lazy="selectin"
    )

    reviews = relationship("Review", back_populates="consumer", lazy="selectin")

    __mapper_args__ = {
        "polymorphic_identity": "consumer",
    }
```

business_model

```python
class Business(User):
    __tablename__ = "businesses"

    id = Column(Integer, ForeignKey('users.id'), primary_key=True)
    license_number = Column(String, unique=True, index=True, nullable=False)  # From SFA data
    stall_name = Column(String, nullable=True) # From SFA data - can be empty
    licensee_name = Column(String, nullable=False)  # From SFA data - notNull
    cuisine_type = Column(Enum(CuisineType), nullable=True)
    description = Column(String(500), nullable=True)  # 500 characters
    status = Column(Enum(StallStatus), default=StallStatus.OPEN)
    status_updated_at = Column(DateTime, nullable=True)

    # Location info
    establishment_address = Column(String, nullable=False)  # From SFA data
    hawker_centre = Column(String, nullable=False)  # From SFA data
    postal_code = Column(String, nullable=False)  # From SFA data

    # Media
    photo = Column(String, nullable=True)  # Store filename/path to photo

    # Relationships
    operating_hours = relationship("OperatingHour", back_populates="business", cascade="all, delete-orphan")
    menu_items = relationship("MenuItem", back_populates="business", cascade="all, delete-orphan")

    __mapper_args__ = {
        "polymorphic_identity": "business",
    }
```

REQUIREMENT ANALYSIS: LOGIN

Sequence Diagram

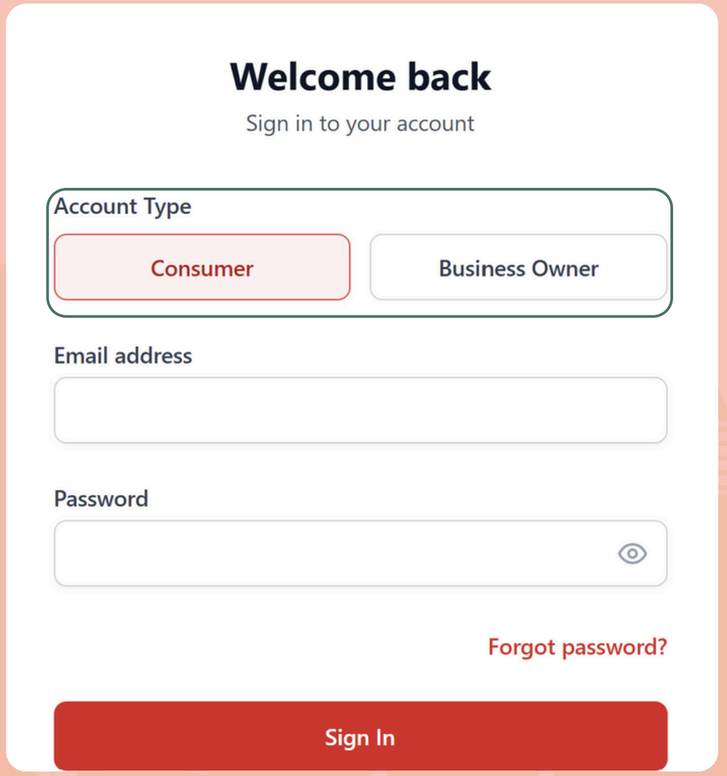


IMPLEMENTATION: LOGIN

Use Case Description

Actor:	Consumer and Business Owner
Description:	None
Preconditions:	User already has an account created.
Postconditions:	User is granted access to respective role after being authenticated.
Priority:	High
Frequency of Use:	High
Flow of Events:	1. User is shown a login form when the User clicks the "Login" button. 2. User clicks on the respective Account Type button, "Consumer" or "Business Owner", to log into the specific account role. 3. User enters valid credentials in the following fields and submits them. a. Email b. Password 4. System compares data inside the database against the user inputs. 5. User is logged into the chosen account type and is redirected to (a) the homepage for Consumer or (b) the Business Profile page for Business Owner.

HawkerSG



LoginPage.tsx

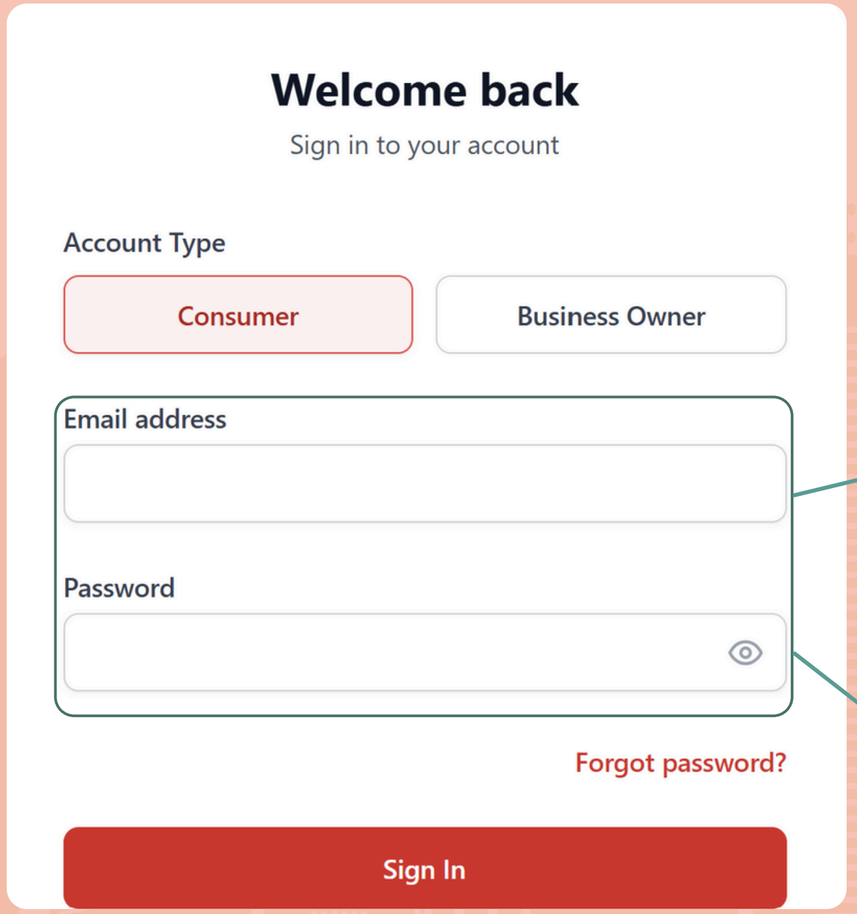
```
<div>
  <label className="block text-sm font-medium text-gray-700 mb-2">Account
Type</label>
  <div className="grid grid-cols-2 gap-3">
    <button
      type="button"
      onClick={() => setUserType('consumer')}
      className={`py-2.5 rounded-lg text-sm font-medium border transition-
all ${userType === 'consumer'
        ? 'border-red-500 bg-red-50 text-red-700 shadow-sm'
        : 'border-gray-300 text-gray-700 hover:border-gray-400 hover:bg-gray-
50'
        }`}
    >
      Consumer
    </button>
    <button
      type="button"
      onClick={() => setUserType('business')}
      className={`py-2.5 rounded-lg text-sm font-medium border transition-
all ${userType === 'business'
        ? 'border-red-500 bg-red-50 text-red-700 shadow-sm'
        : 'border-gray-300 text-gray-700 hover:border-gray-400 hover:bg-gray-
50'
        }`}
    >
      Business Owner
    </button>
  </div>
</div>
```

IMPLEMENTATION: LOGIN

Use Case Description

Flow of Events:	1. User is shown a login form when the User clicks the "Login" button. 2. User clicks on the respective Account Type button, "Consumer" or "Business Owner", to log into the specific account role. 3. User enters valid credentials in the following fields and submits them. a. Email b. Password 4. System compares data inside the database against the user inputs. 5. User is logged into the chosen account type and is redirected to (a) the homepage for Consumer or (b) the Business Profile page for Business Owner.

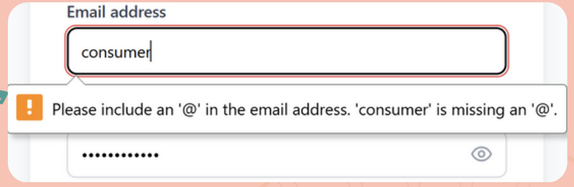
HawkerSG

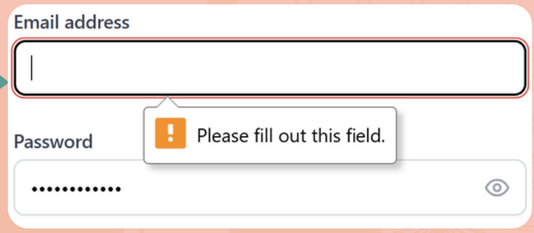


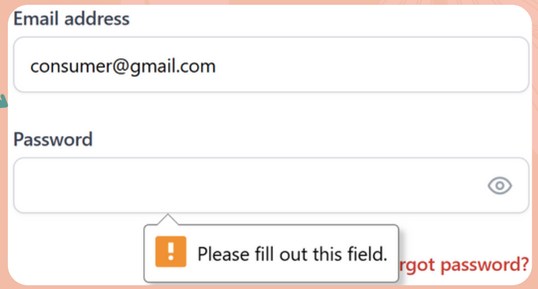
LoginPage.tsx

```tsx
<label htmlFor="email" className="block text-sm font-medium text-gray-700 mb-1">
  Email address
</label>
<input
  id="email"
  type="email"            // HTML5 email validation
  autoComplete="email"
  required                // Required field validation
  value={email}
  onChange={(e) => setEmail(e.target.value)}
```

```tsx
  id="password"
  type={showPassword ? 'text' : 'password'}
  autoComplete="current-password"
  required                // Required field validation
  value={password}
  onChange={(e) => setPassword(e.target.value)}
```







IMPLEMENTATION: LOGIN

EMAIL

consumer_route

```
db_user = consumer_controller.get_user_by_email(db, email)

if not db_user:
    raise HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Incorrect email or password"
    )
```

consumer_controller

```
def get_user_by_email(db: Session, email: str) -> Optional[DBUser]:
    """Retrieves a user (Consumer or other type) by email."""
    return db.query(DBUser).filter(DBUser.email == email).first()
```

PASSWORD

consumer_route

```
if not consumer_controller.verify_password(password, db_user.hashed_password):
    raise HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Incorrect email or password"
    )
```

consumer_controller

```
def verify_password(plain_password: str, hashed_password: str) -> bool:
    """Verifies a plaintext password against a stored hash."""
    return pwd_context.verify(plain_password, hashed_password)
```

Design Principle:

Single Responsibility Principle (SRP)

Open/Close Principle

Liskov Substitution Principle

Use Case Description

Flow of Events:	1. User is shown a login form when the User clicks the “Login” button. 2. User clicks on the respective Account Type button, “Consumer” or “Business Owner”, to log into the specific account role. 3. User enters valid credentials in the following fields and submits them. a. Email b. Password 4. System compares data inside the database against the user inputs. 5. User is logged into the chosen account type and is redirected to (a) the homepage for Consumer or (b) the Business Profile page for Business Owner.

USERTYPE

routes

```
if db_user.user_type != "consumer":
    raise HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Incorrect email or password"
    )
```

```
if db_user.user_type != "business":
    raise HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Incorrect email or password"
    )
```

IMPLEMENTATION: LOGIN

Use Case Description

Flow of Events:	1. User is shown a login form when the User clicks the “Login” button. 2. User clicks on the respective Account Type button, “Consumer” or “Business Owner”, to log into the specific account role. 3. User enters valid credentials in the following fields and submits them. a. Email b. Password 4. System compares data inside the database against the user inputs. 5. User is logged into the chosen account type and is redirected to (a) the homepage for Consumer or (b) the Business Profile page for Business Owner.

Redirect

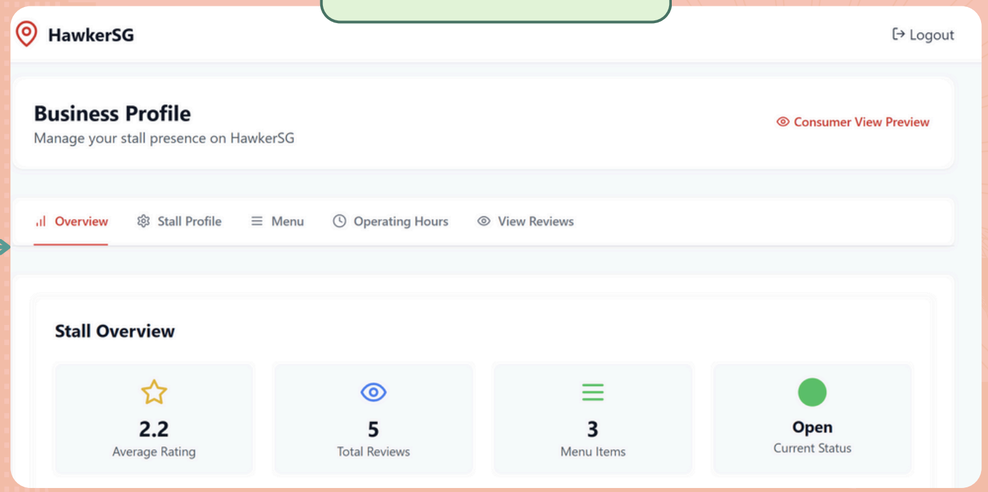
LoginPage

```
if (userType === "consumer") {
  try {
    await login(email, password, 'consumer');
    navigate('/');  // Route to consumer homepage
  } catch (err) {
    setError('Invalid email or password. Please check your credentials and account type.');
  } finally {
    setLoading(false);
  }
```

```
if (userType === "business"){
  try {
    await businessLogin(email, password, 'business');
    navigate("/business");  // Route to business dashboard
  } catch (err: any) {
    setError('Invalid email or password. Please check your credentials and account type.');
  } finally {
    setLoading(false);
  }
```

HomePage

BusinessProfile

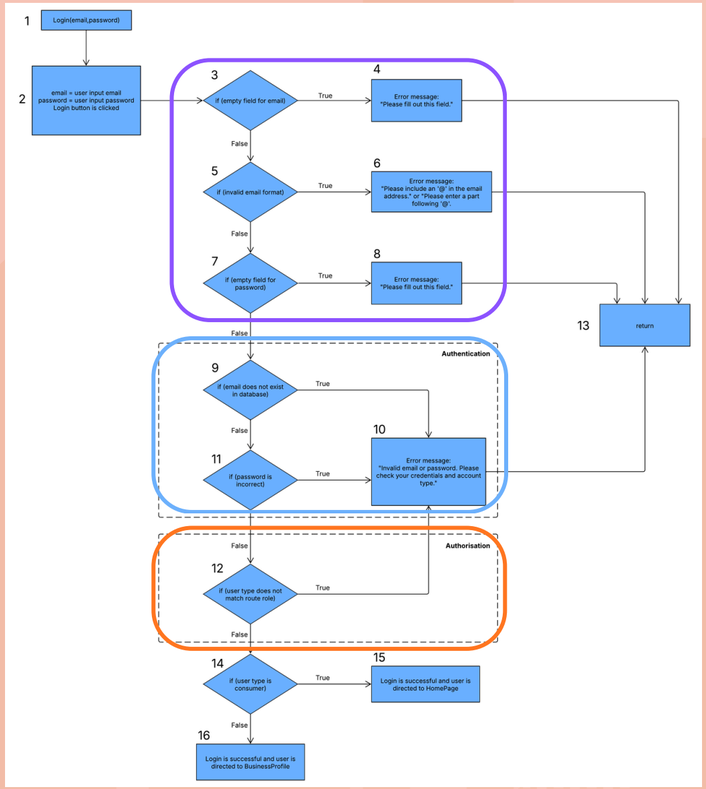


TESTING (WHITEBOX): LOGIN

Cyclomatic Complexity

|decision points| + 1 = 7 + 1 = **8**

FLOW



Main Flow

No.	Test Input			Expected Output	Test Output	Pass?
	Email	Password	User Type			
1	business@gmail.com	Password@123	Business	Redirected to BusinessProfile	Redirected to BusinessProfile	Yes
2	consumer@gmail.com	Password@123	Consumer	Redirected to HomePage	Redirected to HomePage	Yes

Alt Flow

No.	Test Input			Expected Output	Test Output	Pass?
	Email	Password	User Type			
3	""	Password@123	Consumer	Please fill out this field	Please fill out this field	Yes
4a	consumer	Password@123	Consumer	Please include an '@' in the email address	Please include an '@' in the email address	Yes
4b	consumer@	Password@123	Consumer	Please enter a part following '@'	Please enter a part following '@'	Yes
5	consumer@gmail.com	""	Consumer	Please fill out this field	Please fill out this field	Yes
6	FakeUser@gmail.com	Password@123	Consumer	Invalid email or password	Invalid email or password	Yes
7	user@gmail.com	WrongPass@123	Consumer	Invalid email or password	Invalid email or password	Yes
8	consumer@gmail.com	Password@123	Business	Invalid email or password	Invalid email or password	Yes

→ **Formatting** (3, 4a, 4b, 5)

→ **Authentication** (6, 7)

→ **Authorization** (8)

Basis Paths

1. Baseline path: 1, 2, 3, 5, 7, 9, 11, 12, 14, 16
2. Basis path: 1, 2, 3, 5, 7, 9, 11, 12, 14, 15
3. Basis path: 1, 2, 3, 4, 13
4. Basis path: 1, 2, 3, 5, 6, 13
5. Basis path: 1, 2, 3, 5, 7, 8, 13
6. Basis path: 1, 2, 3, 5, 7, 9, 10, 13
7. Basis path: 1, 2, 3, 5, 7, 9, 11, 10, 13
8. Basis path: 1, 2, 3, 5, 7, 9, 11, 12, 10, 13

FUTURE IMPROVEMENTS

HawkerSG

User Experience

- Queue Monitoring
- Smart Recommendation System
- Multilingual - Include SG's 4 official languages

Onboarding

- Incentivize Business Owners to claim their stall.
- Hands-on support to assist with claiming
- Include badges for stalls (Like CDC Voucher stickers)

THANK YOU!