

"Less Waste, More Taste, Share the Plate"

SharePlate



Team 2

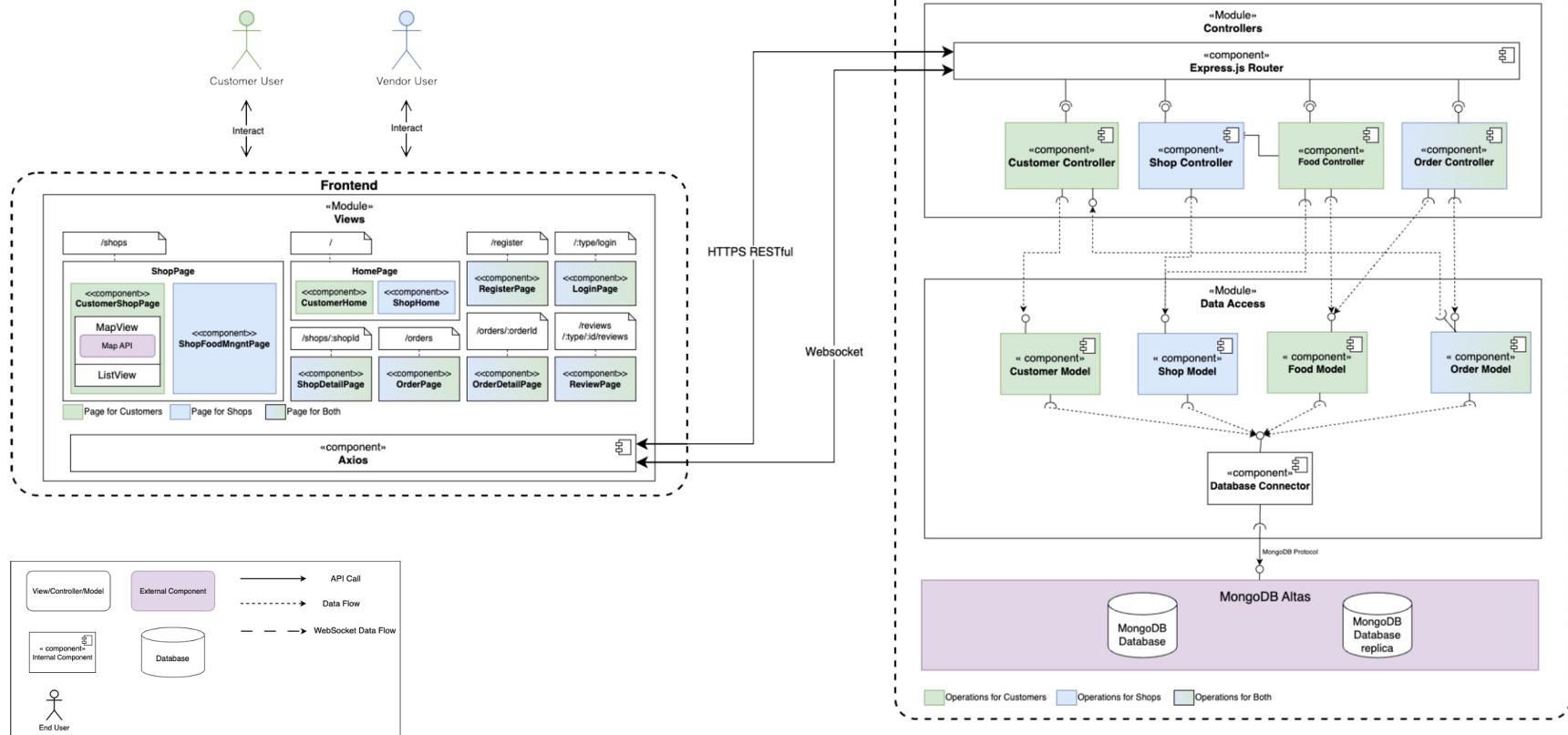
John Su, Martin Chou, James Chang, Bruce Cai, Lona Lu

Architecture Overview

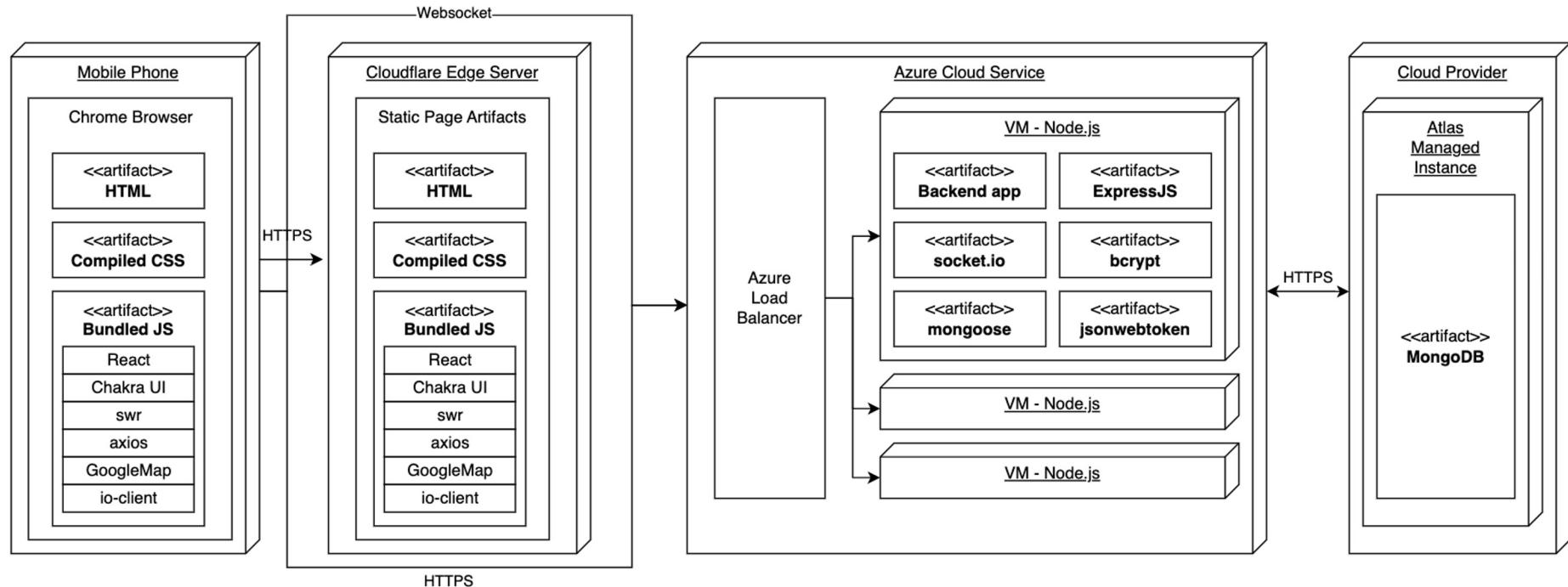
"Less Waste, More Taste, Share the Plate"



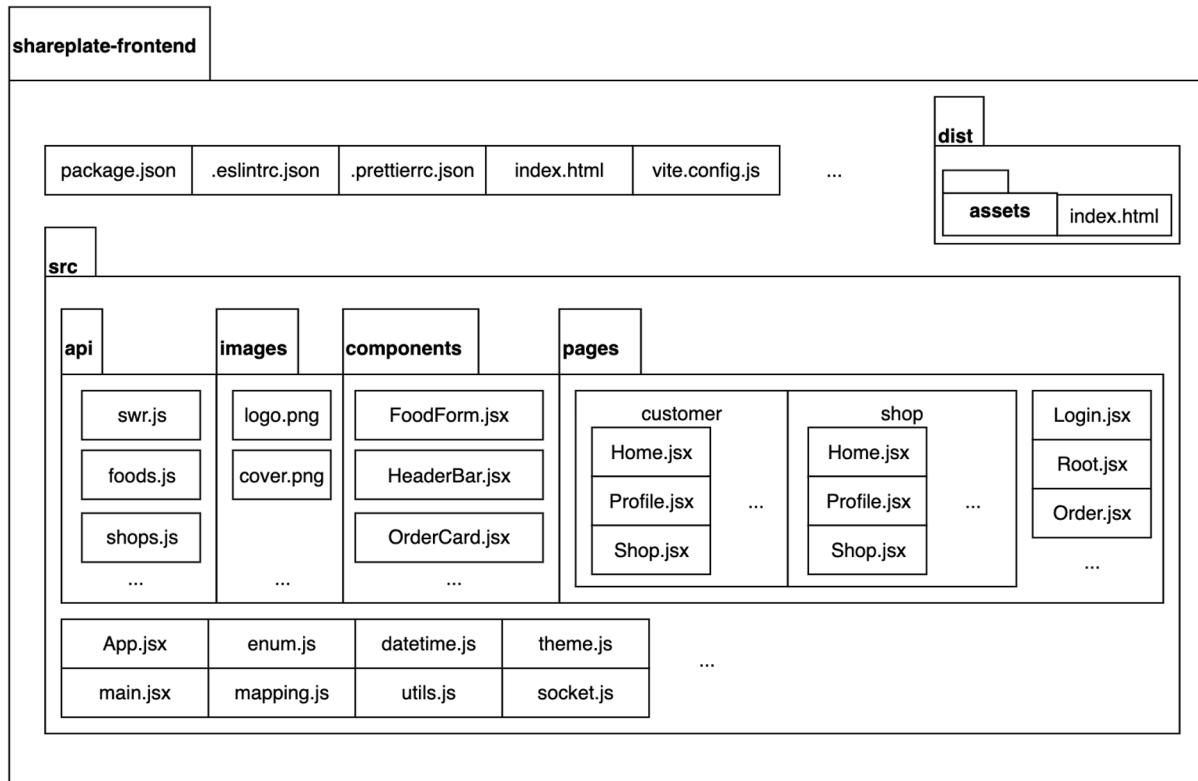
Architectural Diagram



Deployment View

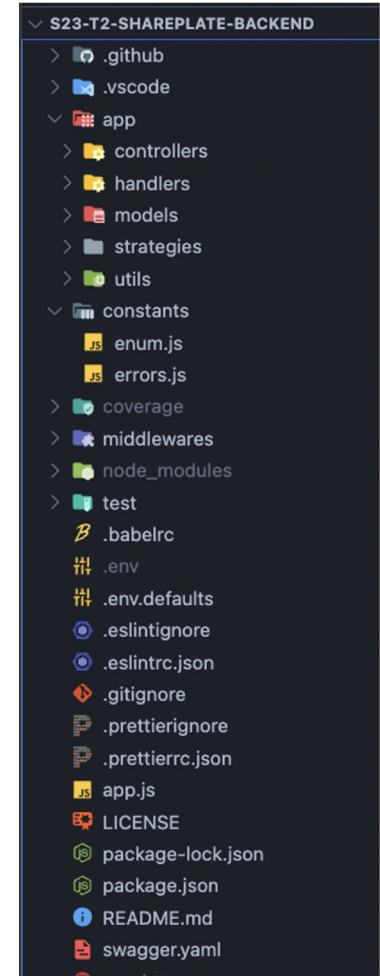
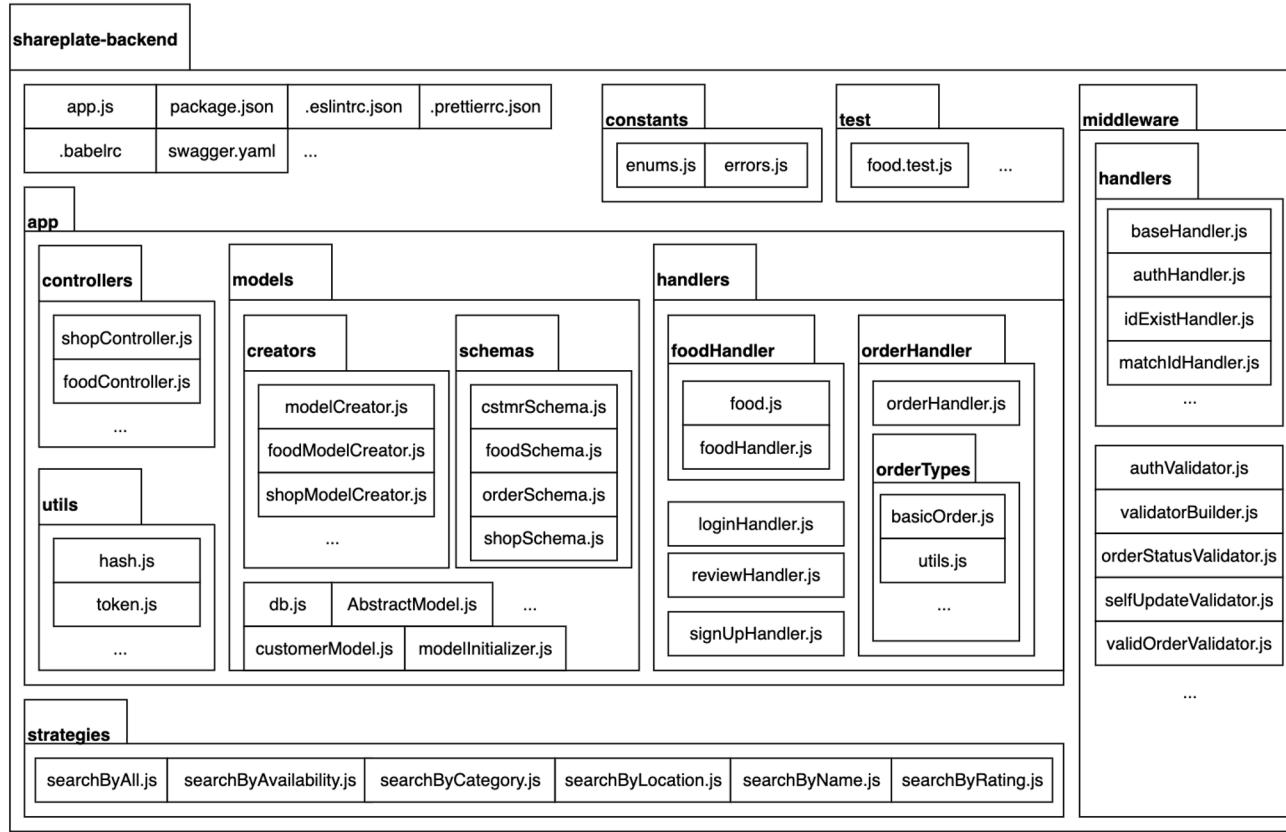


Package View - Frontend



```
S23-T2-SHAREPLATE-FRONTEND
> .github
> .vite
> .vscode
> dist
> node_modules
> public
> src
> api
> components
> images
> mock
> pages
> App.jsx
> datetime.js
> enum.js
> main.jsx
> mapping.js
> socket.js
> theme.js
> utils.js
> .env
> .env.defaults
> .eslintignore
> .eslintrc.json
> .gitignore
> .prettierignore
> .prettierrc.json
> index.html
> LICENSE
> package-lock.json
> package.json
> README.md
> vite.config.js
> yarn-error.log
> yarn.lock
```

Package View - Backend



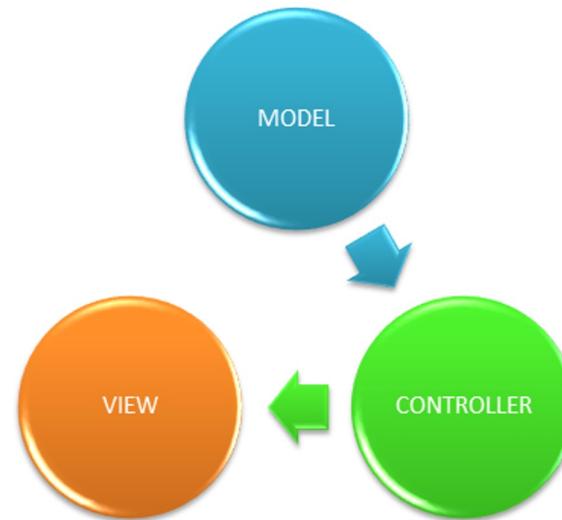
MVC

Team development capability

- Clear Separation of Concerns
 - Easier Collaboration
- Speed up development pace through reusability
 - Ex. Backend Components

Application requirements

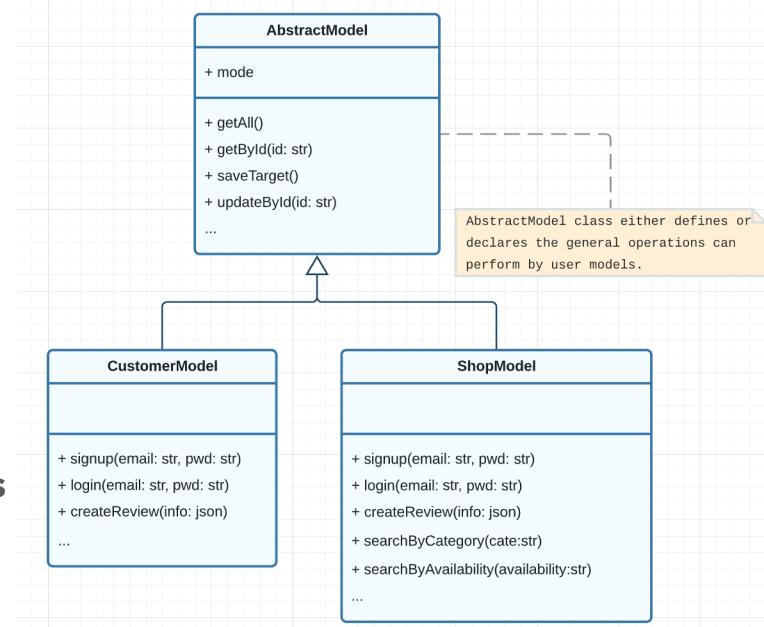
- Low latency
 - Displaying food inventory
 - Placing an order during rush hour
- Reduce unnecessary overhead
 - Maintain high response performance



MVC

NFR: Reusability - Backend

- **Reusable DAO Base Abstract class (Model layer)**
 - Define common operations for all DAO
 - Get by Id / Get all, ...
 - Set the universal DB mode
 - Set up universal settings for all DAO
 - Production mode/Mock mode
 - Declare necessary interface to be implemented
- **Reusable Components - Handlers class, middlewares**
 - Define common operations as handlers
 - No need to know which model is conducting specific query
 - Reusable authentication middlewares



MVC

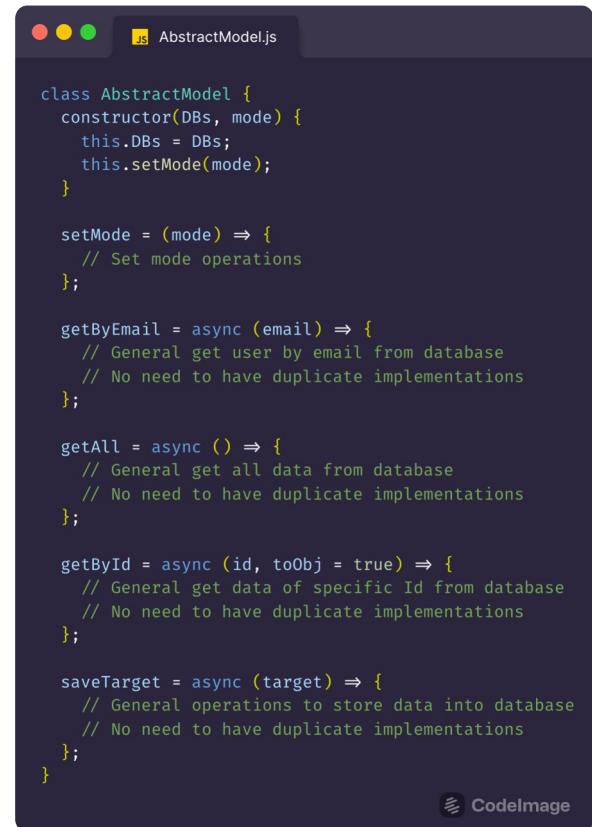
NFR: Reusability - Backend

- **Reusable DAO Base Abstract class
(Model layer)**

- Define common operations for all DAO
 - Get by Id / Get all, ...
 - Set the universal DB mode
- Set up universal settings for all DAO
 - Production mode/Mock mode
- Declare necessary interface to be implemented

- **Reusable Components - Handlers class, middlewares**

- Define common operations as handlers
- No need to know which model is conducting specific query
- Reusable authorization/authentication middlewares



A screenshot of a code editor window titled "AbstractModel.js". The code is written in JavaScript and defines a base abstract class for DAOs. It includes methods for setting mode, getting users by email or ID, getting all data, and saving targets. The code uses modern JavaScript features like arrow functions and async/await.

```
class AbstractModel {  
  constructor(DBs, mode) {  
    this.DBs = DBs;  
    this.setMode(mode);  
  }  
  
  setMode = (mode) => {  
    // Set mode operations  
  };  
  
  getByEmail = async (email) => {  
    // General get user by email from database  
    // No need to have duplicate implementations  
  };  
  
  getAll = async () => {  
    // General get all data from database  
    // No need to have duplicate implementations  
  };  
  
  getById = async (id, toObj = true) => {  
    // General get data of specific Id from database  
    // No need to have duplicate implementations  
  };  
  
  saveTarget = async (target) => {  
    // General operations to store data into database  
    // No need to have duplicate implementations  
  };  
}
```

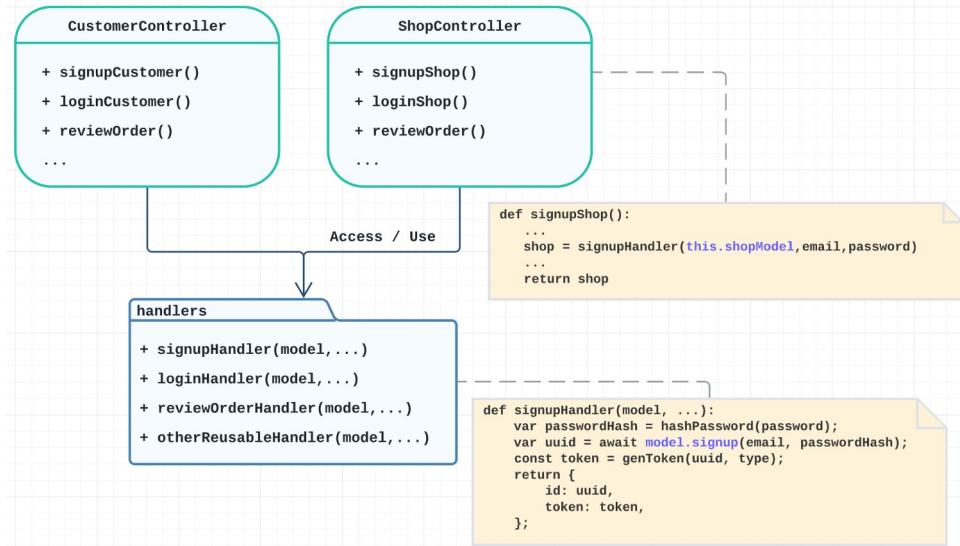
CodeImage

MVC

NFR: Reusability - Backend

- **Reusable DAO Base Abstract class (Model layer)**

- Define common operations for all DAO
 - Get by Id / Get all, ...
 - Set the universal DB mode
- Set up universal settings for all DAO
 - Production mode/Mock mode
- Declare necessary interface to be implemented



- **Reusable Components - Handlers class, middlewares**

- Define common operations as handlers
- No need to know which model is conducting specific query
- Reusable authentication middlewares

MVC

NFR: Reusability - Backend

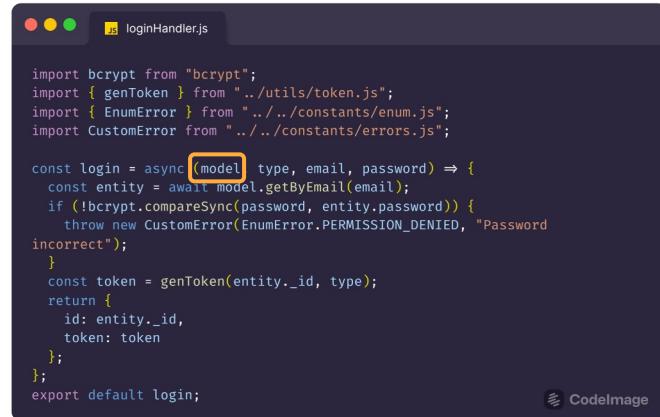
- **Reusable DAO Base Abstract class**

(Model layer)

- Define common operations for all DAO
 - Get by Id / Get all, ...
 - Set the universal DB mode
- Set up universal settings for all DAO
 - Production mode/Mock mode
- Declare necessary interface to be implemented

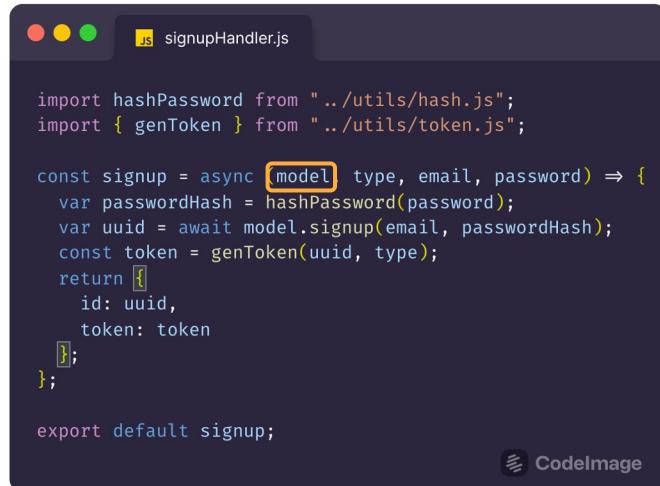
- **Reusable Components - Handlers class, middlewares**

- Define common operations as handlers
- No need to know which model is conducting specific query
- Reusable authentication middlewares



```
import bcrypt from "bcrypt";
import { genToken } from "../utils/token.js";
import { EnumError } from "../../constants/enum.js";
import CustomError from "../../../constants/errors.js";

const login = async [model] type, email, password => {
  const entity = await model.getByEmail(email);
  if (!bcrypt.compareSync(password, entity.password)) {
    throw new CustomError(EnumError.PERMISSION_DENIED, "Password incorrect");
  }
  const token = genToken(entity._id, type);
  return {
    id: entity._id,
    token: token
  };
}
export default login;
```



```
import hashPassword from "../utils/hash.js";
import { genToken } from "../utils/token.js";

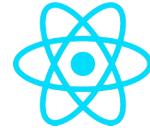
const signup = async [model] type, email, password => {
  var passwordHash = hashPassword(password);
  var uuid = await model.signup(email, passwordHash);
  const token = genToken(uuid, type);
  return [
    id: uuid,
    token: token
  ];
}

export default signup;
```

Design Decisions

"Less Waste, More Taste, Share the Plate"





Reusability

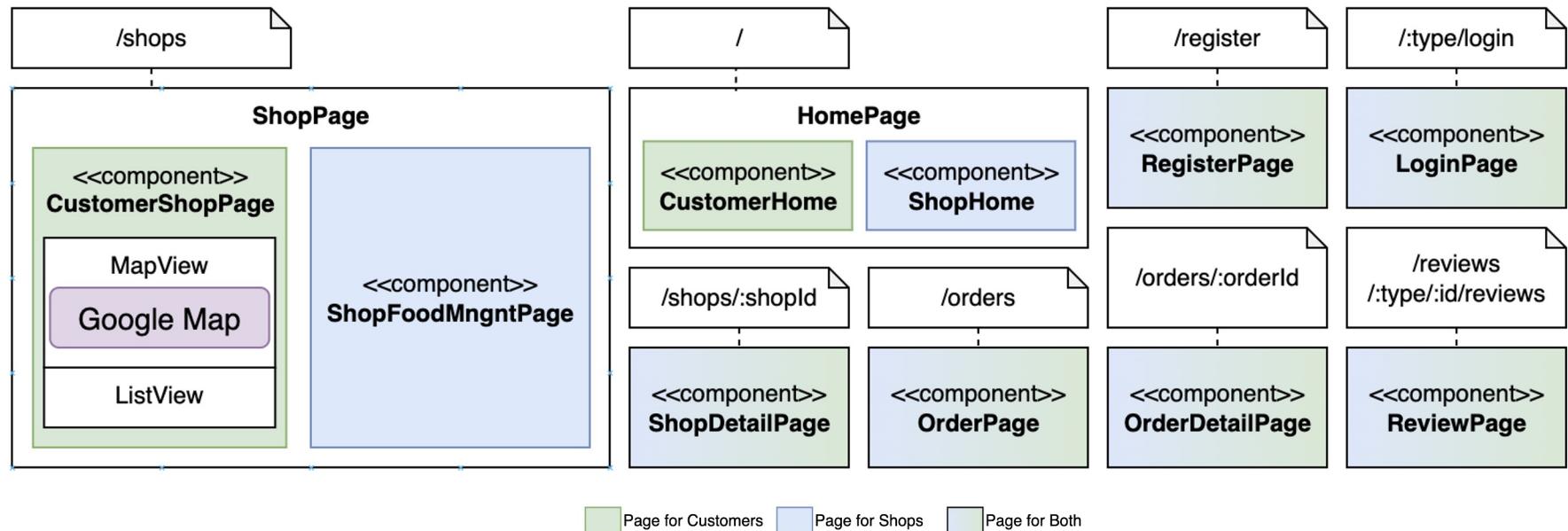
"Less Waste, More Taste, Share the Plate"



React

Reusability - React Router + Reusable page design

Reusable Pages for different types of users reduces code duplication.



React

Reusability - React Router + Reusable page design

Reusable Pages for different types of users reduces code duplication.

The diagram illustrates the reuse of components across three files:

- s23-t2-shareplate-frontend - App.js**:

```
68 <BrowserRouter>
69   <Routes>
70     <Route path="/" element={<Home />} /> ----->
71     <Route path="/shops" element={<Shop />} />
72     <Route path="/shops/:id" element={<ShopDetail />} />
73     <Route path="/login/:type" element={<Login />} />
74     <Route path="/register" element={<Register />} />
75     <Route path="/profile" element={<Profile />} />
76     <Route path="/orders" element={<Order />} />
77     <Route path="/orders/:id" element={<OrderDetail />} />
78     <Route path="/reviews" element={<Review />} />
79     <Route path="/:type/:id/reviews" element={<Review />} />
80     <Route path="*" element={<NotFound />} />
81   </Routes>
82 </BrowserRouter>
```
- s23-t2-shareplate-frontend - Home.js**:

```
1 import CustomerHome from './customer/Home';
2 import PageStrategy from './PageStrategy';
3 import ShopHome from './shop/Home';
4
5 function Home() {
6   return PageStrategy(<CustomerHome />, <ShopHome />);
7 }
8
9 export default Home;
```
- s23-t2-shareplate-frontend - Shop.js**:

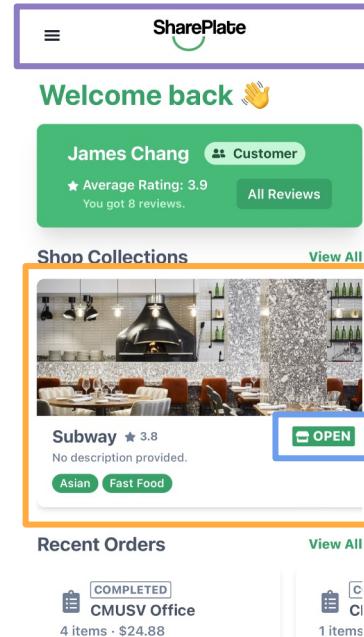
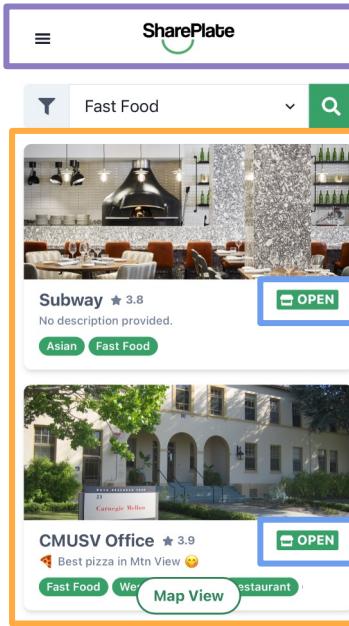
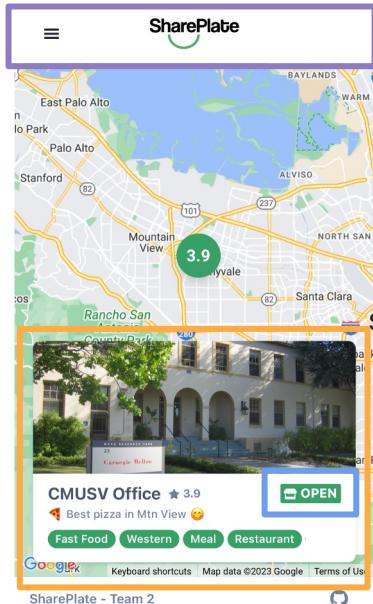
```
1 import CustomerShop from './customer/Shop';
2 import PageStrategy from './PageStrategy';
3 import ShopShop from './shop/Shop';
4
5 function Shop() {
6   return PageStrategy(<CustomerShop />, <ShopShop />);
7 }
8
9 export default Shop;
```

A yellow arrow points from the first `<Route path="/" ...>` in `App.js` to the `function Home()` in `Home.js`. Another yellow arrow points from the first `<Route path="/" ...>` in `App.js` to the `function Shop()` in `Shop.js`.

React

Reusability - Components

Reusable component in various pages to prevent code duplication.



```
<HeaderBar userType={userType} />
```

```
<ShopCard shop={shopData} />
```

```
<ShopStatusBadge status={shopData.status} />
```



Availability

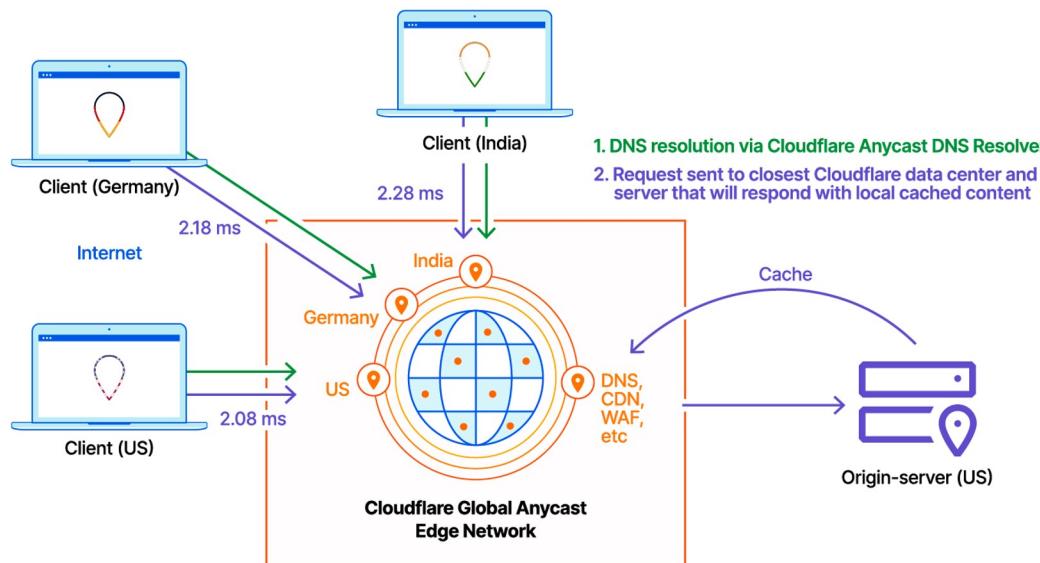
"Less Waste, More Taste, Share the Plate"



Deployments

Availability - Cloudflare Pages

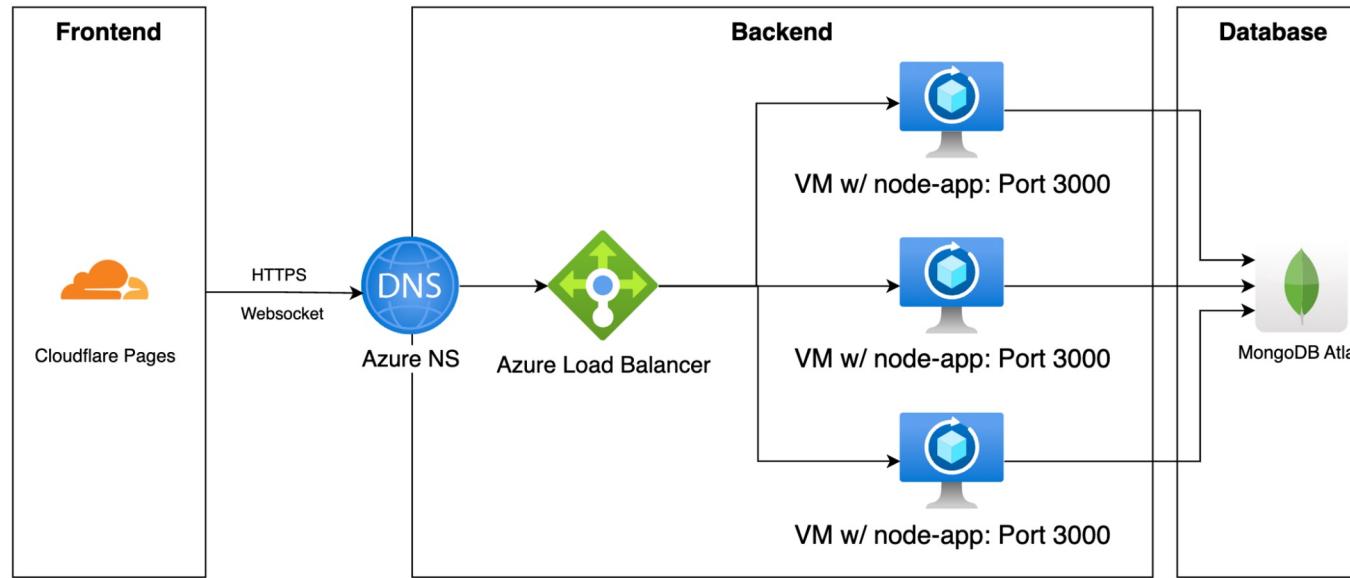
Deploying our CSR frontend artifacts on Cloudflare Pages leverages its CDN functionality. CDN gives us advantages on Availability and Performance.

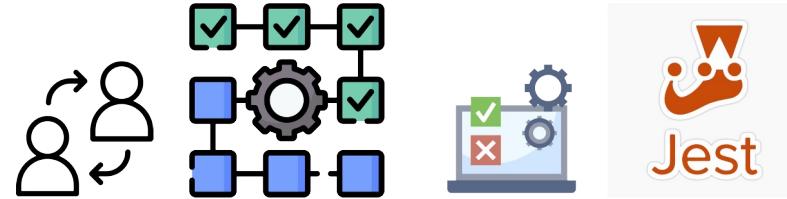


Deployments

Availability - Azure Load Balancing

A public load balancer paired w/ multiple backend instances increased availability.





Testability

"Less Waste, More Taste, Share the Plate"

SharePlate

Testability

Testability

- Mock using Jest



Mock request and response

```
const MockResponse = () => {
  const res = {};
  res.status = jest.fn().mockReturnValue(res);
  res.json = jest.fn().mockReturnValue(res);
  return res;
};
```

Codelimage

Mock JWT & Bcrypt Lib

```
jest.mock("jsonwebtoken", () => ({
  ... jest.requireActual("jsonwebtoken"),
  sign: jest.fn().mockReturnValue("thisIsFakeJWT")
}));
```

Codelimage

Improvement required:

- No need to have a real DB connection

mockbcrypt.js

```
jest.mock("bcrypt", () => ({
  ... jest.requireActual("bcrypt"),
  compareSync: jest.fn()
}));
```

Codelimage

Testability

Testability

- Mock using Jest

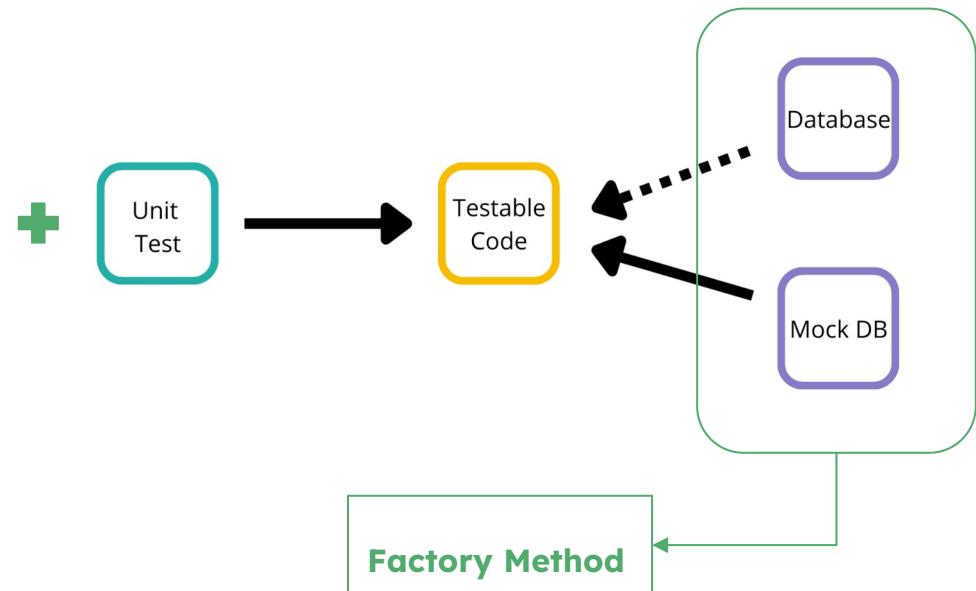


Improvement required:

- No need real DB connection

Testability

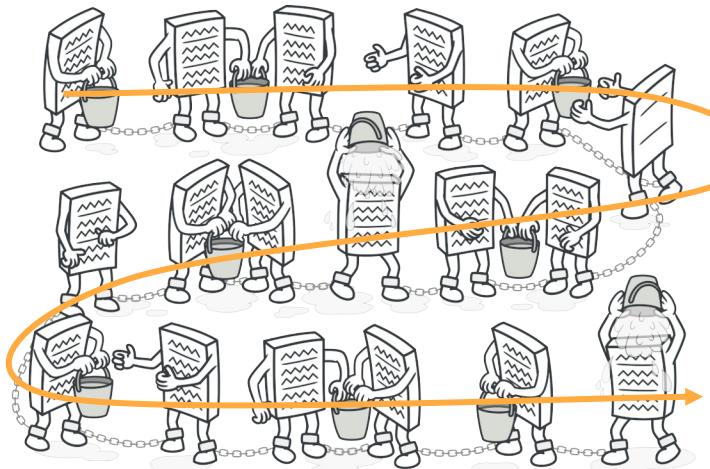
- Dependency Injection



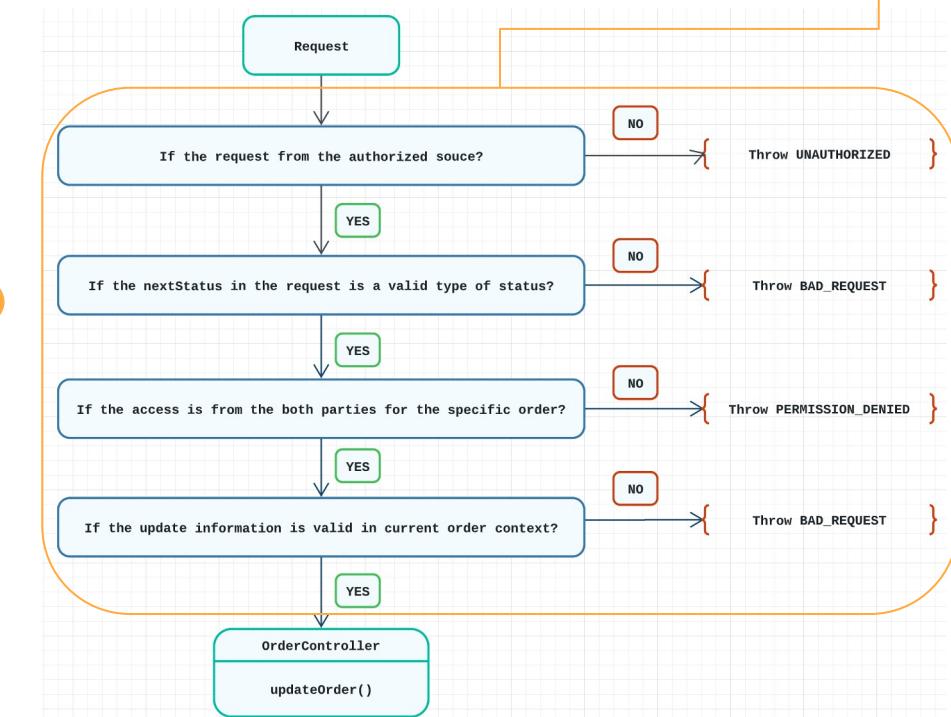
Testability

Testability

- Chain of Responsibility
 - Perform test in **different granularity**



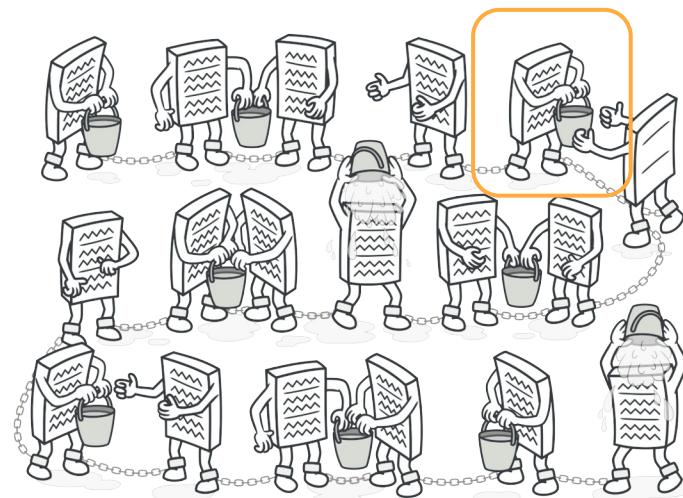
Perform testing in validator (chained) level



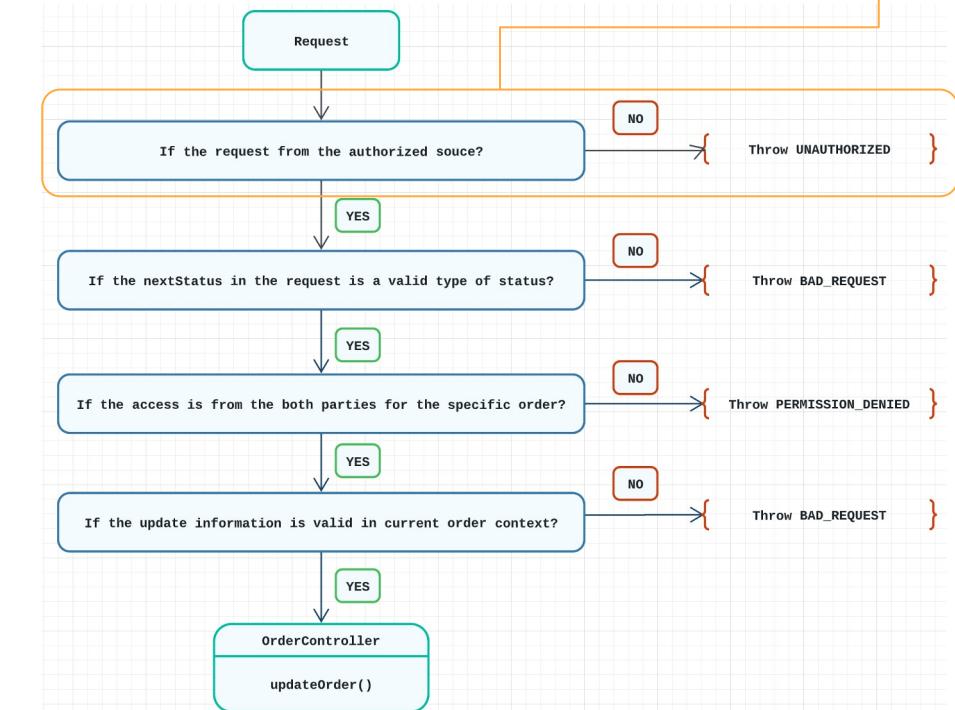
Testability

Testability

- Chain of Responsibility
 - Perform test in **different granularity**



Perform Unit test in handler level



Design Patterns

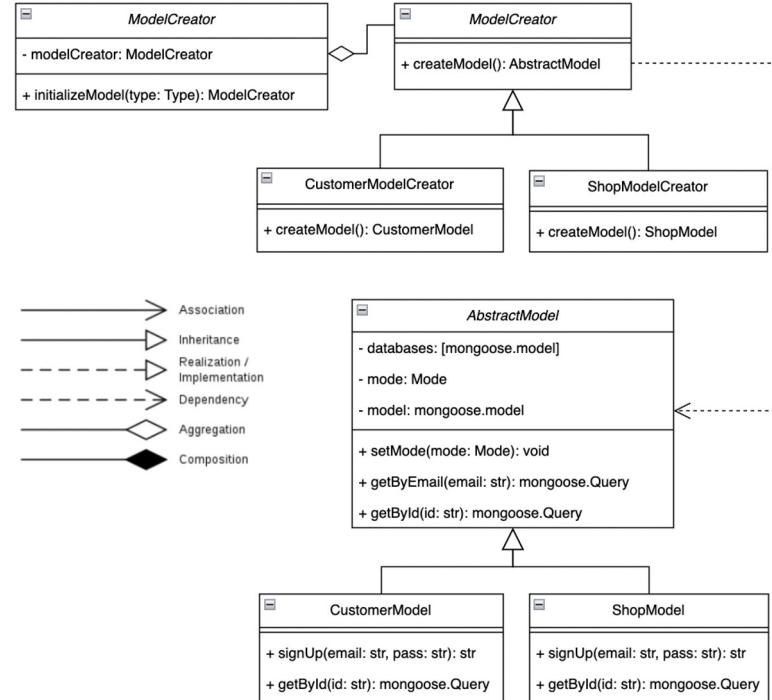
"Less Waste, More Taste, Share the Plate"



Design Pattern - Factory Method

How did it solve our problem?

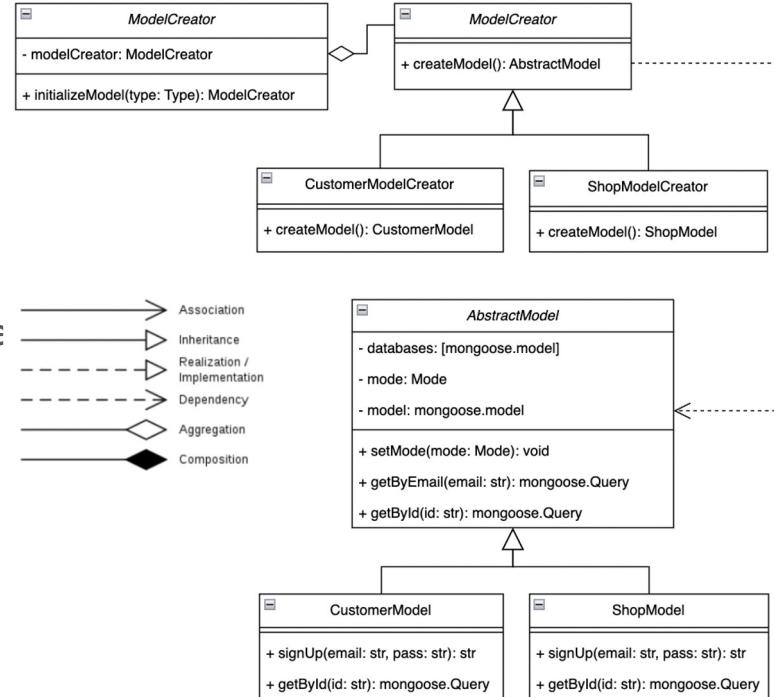
- It helps creating models - data access objects.
- It allows us use different subclasses to determine the type of objects to be created. (Customer / Shop)



Design Pattern - Factory Method

Are there any benefits?

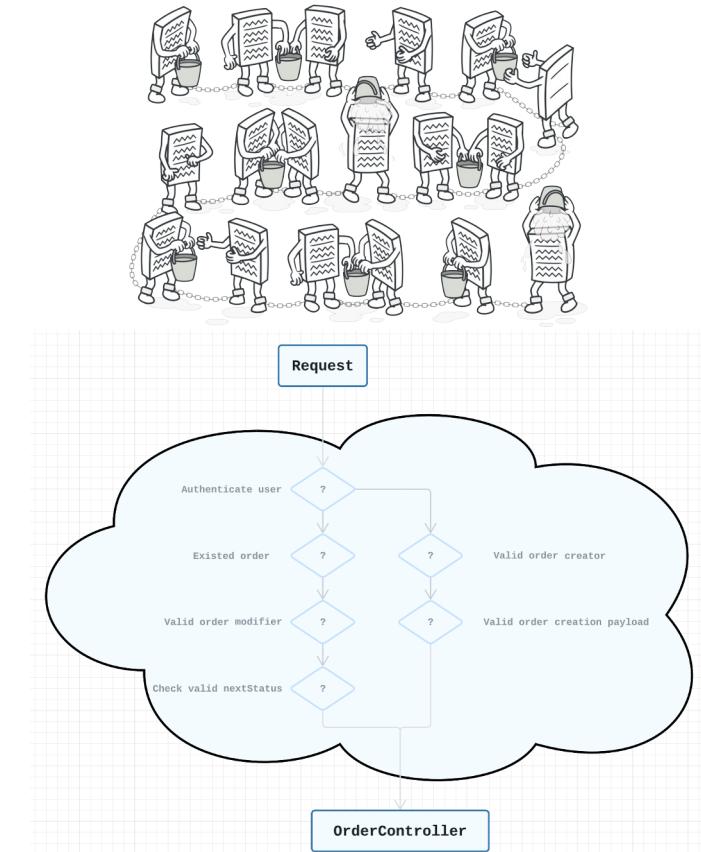
- Single Responsibility: Creation code is all in one place.
- Open/Closed Principle: Let us handle two user type without modifying other code.
- Outer code knows that all Models are supposed to have `getById()` and `signup()` method, but they don't need to know how it works, or which collection it is querying.



Design Pattern - Chain of Responsibility

Problem: Many validations and authorizations must be performed before permit create/modify a certain order.

- Only authorized users can access orders.
- The order needs to exist.
- The query for the order needs to be a valid query.
- Only both parties for the specific order can permit access.
- Only a specific type of user can change the status of the order
 - i.e., Only the shop can confirm the order
- The changed status needs to be a valid next status to the current order.
 - Completed status must be put after Confirmed.

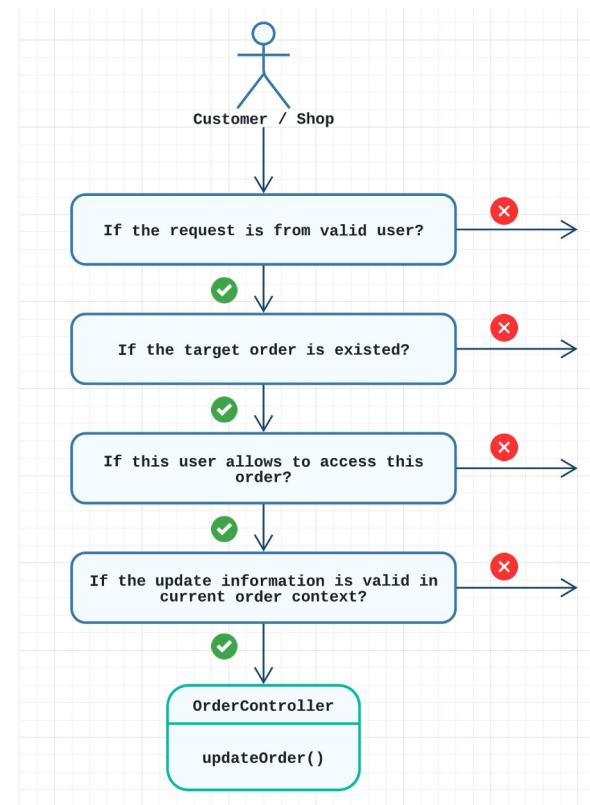


Design Pattern - Chain of Responsibility

Problem: Bunch of validation and authentication need to perform before permit create/modify certain order.

- Only authenticated users can access orders.
- The order needs to exist.
- The query for the order needs to be a valid query.
- Only both parties for certain orders can permit access.
- Only a specific type of user can change the status of the order
 - i.e., Only the shop can confirm the order
- The changed status needs to be a valid next status to the current order.
 - Completed status needs to be happened after Confirmed.

Observation: The request should pass a series of checks before the orderController handles it.



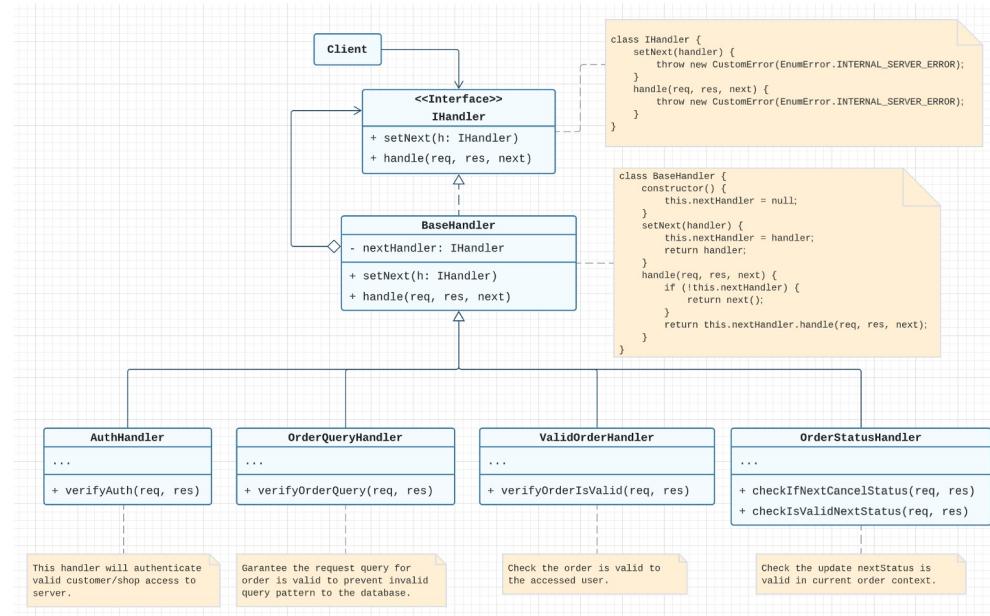
Design Pattern - Chain of Responsibility

How did it solve our problem?

- Formulate particular validation into stand-alone objects into a handler
- Chain and reuse the required validation/authentication behaviors

Are there any benefits?

- **Increase reusability**
 - Handlers are reusable
- **Increase testability**
 - Able to test single handler logic separately
- **Open/Closed Principle**
 - Allow us to introduce new kinds of handlers easily
- **Single Responsibility**
 - Each handler performs a specific task



Design Pattern - Decorator

Problem:

- With the growing of business, we want to add new features based on existing ones without changing the underlying code.



Design Pattern - Decorator

Problem:

- With the growing of business, we want to add new features based on existing ones without changing the underlying code.

How did it solve our problem?

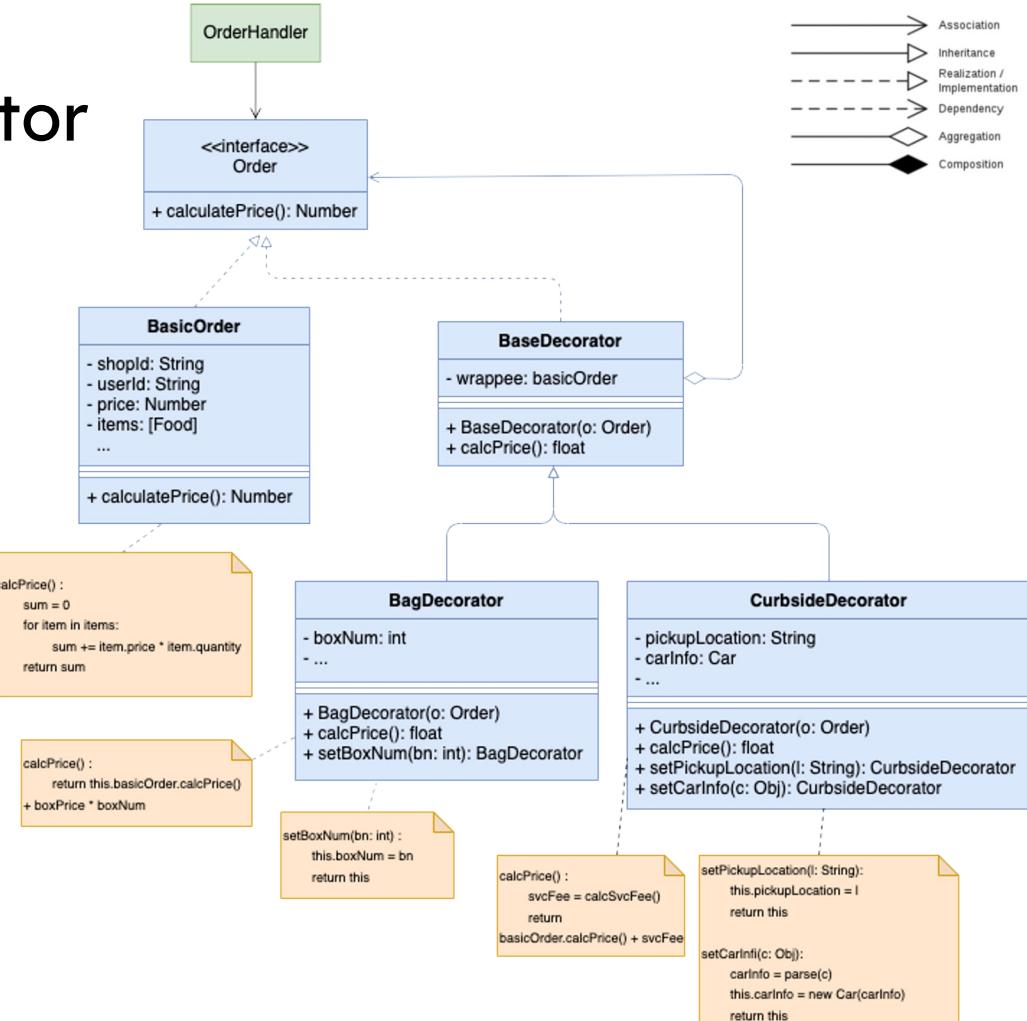
- Keep basic order untouched
- Reuse the basic order by injecting it to new decorator and describe new features in decorator

The screenshot shows a mobile application interface for food ordering. At the top, there's a header with the text "Order from shop2@gmail.com" and a close button. Below the header, the word "Items" is displayed above a list of items. The first item in the list is "Order from shop2@gmail.com" with a quantity of 00 and a price of \$61.00. Underneath this item, there's another section labeled "Items" with a single item: "1 coke" at \$4.00, with a "Curbside" option selected. To the right of this item are buttons for "-1+" and a green "SharePlate" icon. Below the item list, there's a section for "Order Type" with options for "Normal", "+ TOGO Box", and "Curbside". Further down, there's a section for "Order Preferences" and a "Pick-up Location" field set to "SharePlate". At the bottom of the screen, there's a "Order Management" section listing two orders. The first order is for "shop2@gmail.com" and is marked as "CANCELED" with a timestamp of "Apr 20, 12:04 PM". The second order is also for "shop2@gmail.com" and is marked as "PENDING" with a timestamp of "Apr 20, 12:07 PM". Both orders show a quantity of 1 item at a price of \$61.00. At the very bottom of the screen, there's a message "That's all!" with a smiley face icon.

Design Pattern - Decorator

Are there any benefits?

- Flexible and low coupling
 - Extend the function of objects more flexibly than inheritance
- Open/Closed Principle
 - Add additional functionality to an object without modifying its underlying code
- Single Responsibility Principle
 - Each decorator class has its own responsibility and doesn't interfere with the responsibilities of other classes



Design Pattern - Observer

Feature: We want to allow customers to get real-time update for their favorite shops

Problem: Hard to continuously poll the database to check for changes.

Additionally, it is hard to manage and notify a large number of customers when updates occur.

Solution: Observer design pattern

(Shop)

(Customer)

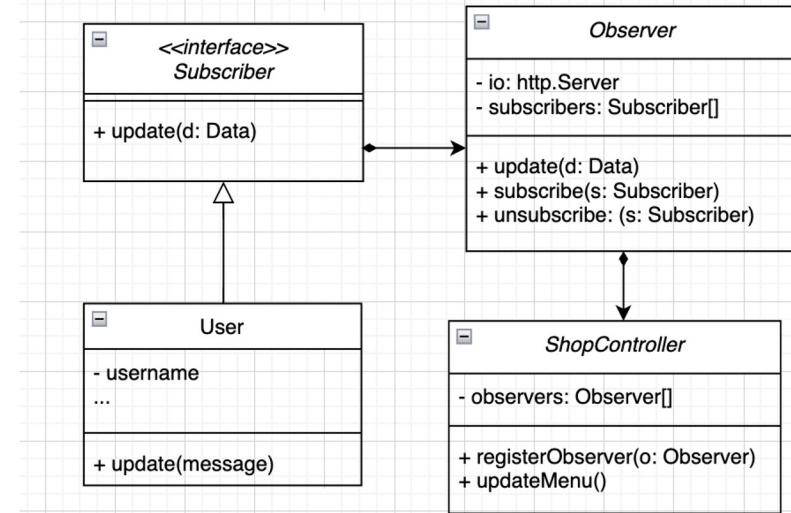
The image shows a mobile application interface. At the top, a green banner displays a notification: "Subscribed shop update! CMUSV Office has updated their menu. Check it out!" Below this is a "WELCOME BACK" message with a yellow arrow icon. To the right, a user profile for "James Chang" (Customer) is shown, including an average rating of 3.9 and 8 reviews, with a "View All Reviews" button. A "Shop Collections" section features a thumbnail of a restaurant interior and a "View All" button. Below it, a card for "Subway" shows a 3.8 rating, no description, and categories "Asian" and "Fast Food". Another card for "CMUSV Office" shows a completed order with 4 items totaling \$24.88 from April 15 at 3:07 PM. The bottom section, "Recent Orders", lists "SharePlate - Team 2" with a message icon. The overall theme is a food delivery or ordering app.

Design Pattern - Observer

User: The concrete subscribers in the Observer pattern, who want to receive real-time updates for their favorite shops.

ShopMenuObserver: A specific observer that is responsible for notifying users when a shop's menu changes.

ShopController: The controller that manages the Shop subscription feature. It registers observers and notifies them when changes occur.

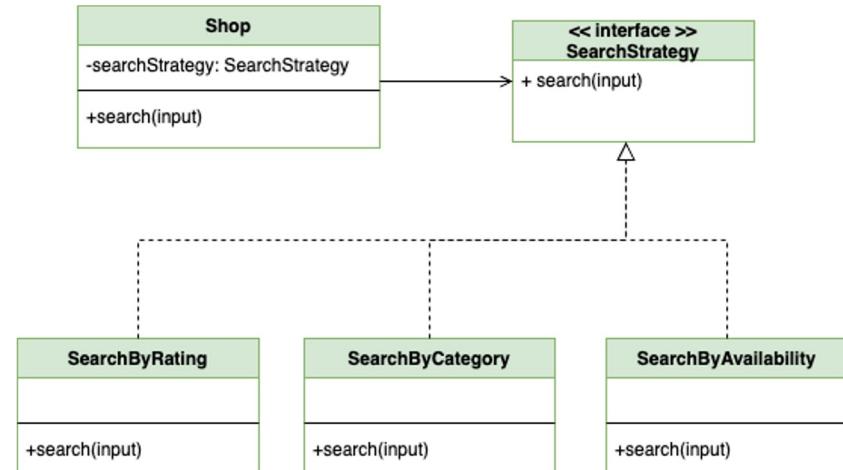


```
update(data) {
  for (let i = 0; i < subscribers.length; i++) {
    const subscriberId = subscribers[i];
    this.io.to(subscriberId.toString()).emit("menuChanged", { data });
  }
}
```

Design Pattern - Strategy Method

What we visioned - how will it solve our problem?

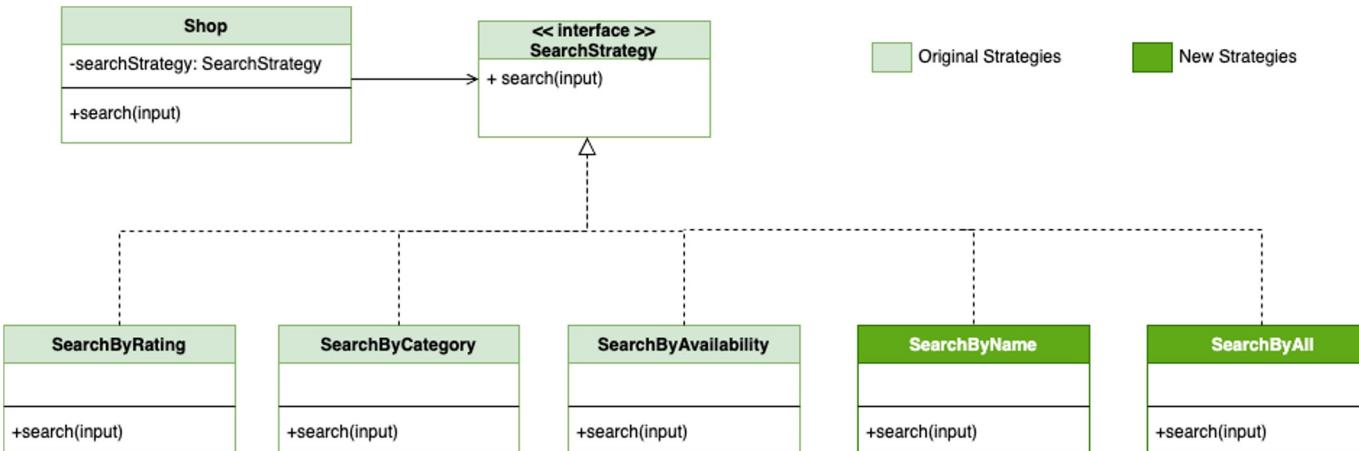
- **Feature:** allow customers to search for shops using different criterions
- **Problem:** redundant code handling all possibilities, difficult to manage and maintain
- **Solution:** Strategy method with encapsulation



Design Pattern - Strategy Method

What really happened - are there any benefits?

- **Scenario:** new search methods required during development
- **Benefits:** Provides flexibility to add new criterion & Ability to switch to different search algorithms at runtime



Demo

"Less Waste, More Taste, Share the Plate"



Food Posting



As a shop, I can **post food items** for sharing or sale, along with details such as the amount available, condition, and its image.

6:16

shareplate.pages.dev

SharePlate

Food Management

| | | | |
|--|--|----------------------|------------------------|
| | bread from this morning \$6.90 | Edit | Delete |
| | coke NON-DIET COKE \$1.99 | Edit | Delete |
| | pizza big pizza \$17.00 | Edit | Delete |
| | yet another pizza another pizzazzzzz | Edit | Delete |

[+ Add Food](#)

← → + 15 ...

6:16

shareplate.pages.dev

SharePlate

Edit Food

Food Name: bread

Price: 6.9

Quantity: 415

Description: from this morning

Image URL: <https://upload.wikimedia.org/wikipedia/cor>

[Save](#)

← → + 15 ...

Map & Tag-based Search



As a customer, I can use a map-based interface to search and navigate to the nearest food providers.

I can view their location and distance, and get directions to their location.

The image displays three screenshots of the SharePlate mobile application, illustrating its map-based search and tag-based search features.

Screenshot 1: Map-based Search
This screenshot shows a map of the San Francisco Bay Area, specifically the areas around Palo Alto, Mountain View, and Cupertino. Several food provider locations are marked with green circles, each containing a rating (e.g., 3.9, 3.8). The map includes street names like 101, 82, and 280, and various parks and landmarks. At the bottom, there's a "List View" button and navigation controls.

Screenshot 2: Map-based Search with Detailed View
This screenshot shows a detailed view of a specific location in Mountain View. It features a large image of a building labeled "Carnegie Mellon". Below the image, the "CMUSV Office" is listed with a rating of 3.9, described as having "Best pizza in Mtn View", and categorized as Fast Food, Western, Meal, and Restaurant. There's also an "OPEN" button. Navigation controls are at the bottom.

Screenshot 3: Tag-based Search
This screenshot shows a search results page for "Fast Food". It includes a search bar with a filter icon, a search icon, and a "Fast Food" category selected. Below the search bar, there are two cards: one for "Subway" (rating 3.8) and another for "CMUSV Office" (rating 3.9). Both cards show images, descriptions, and "OPEN" buttons. At the bottom, there's a "Map View" button and navigation controls.

Food Ordering and Pick-up



As a customer, I can place an order for the food items posted by shops. Also, I can request for togo boxes or curbside pickup.

As a Shop, I can view all my recent orders and choose to Confirm, Cancel, or Complete them.

The screenshot shows a mobile application interface for SharePlate. At the top, there's a header with the time (6:18), signal strength, battery level (92%), and the URL shareplate.pages.dev. Below the header is the SharePlate logo. The main content area features a photograph of a white building with arched windows and doors, identified as the CMUSV Office. Below the photo are two buttons: 'Subscribe' and 'Direction'. The section title 'CMUSV Office' is displayed in green, followed by a status badge 'OPEN'. Below this, there are several categories: 'Fast Food', 'Western', 'Meal', 'Restaurant', and 'Cafe'. A yellow circular rating indicator shows a star rating of 3.9. Below these categories is a product listing for 'bread' with a small image of bread rolls, a quantity selector (minus, one, plus), and a price of \$6.90. The total amount at the bottom is \$6.90. A large green 'Checkout' button is prominent. At the very bottom, it says 'SharePlate - Team 2' and includes standard iOS navigation controls (back, forward, search, etc.).

This screenshot shows the 'Order Management' section of the SharePlate app for a shop. The title 'Order Management' is in green. Below it, a completed order for 'CMUSV Office' is listed, showing 4 items totaling \$24.88, placed on April 15 at 3:07 PM. There are 'Manage' and 'Rate' buttons next to the order. Below this is another completed order for 'CMUSV Office' on April 16 at 7:20 PM, costing \$6.90. Further down is a pending order for 'CMUSV Office' on April 16 at 7:25 PM, costing \$5.99. At the bottom, another completed order for 'CMUSV Office' on April 16 at 7:30 PM is shown, costing \$15.99. Each order entry includes a 'Manage' and 'Rate' button. The bottom of the screen has a 'Manage' button and standard iOS navigation controls.

This screenshot shows the 'Order with CMUSV Office' details page. The title is in green. It lists the items ordered: bread (\$6.90), coke (\$1.99), yet another pizza (\$10.00), and wings (\$5.99). The total amount is \$24.88. Below this, a 'Details' section says 'No details' with a small icon. At the bottom, there's a 'Manage' button and standard iOS navigation controls.

Feedback and Rating System



As a customer, I can provide feedback and ratings on the foods and shop itself. I can also look at the cumulative rating when picking a shop.

As a shop, I can use the rating system to report a no-show customer. I can also utilize the customer rating score to filter out low-scored customers.

The left screenshot shows the customer rating interface. It displays the URL shareplate.pages.dev, the SharePlate logo, and a message "Rate your order with James Chang" with a close button. Below this are sections for "Details" (Order ID: 643caf40a27e075081935e86, Order Date: Apr 16 at 7:30 PM) and "Rating". A horizontal slider is set to a value of 5, with tick marks from 1 to 5. At the bottom are "Cancel" and "Submit" buttons, and a navigation bar with arrows, a plus sign, a "15" badge, and three dots.

The right screenshot shows the "Reviews" section. It features a header "Reviews" and a review for "James Chang" (Customer, Average Rating: 3.9). Below this are three more reviews: "CMUSV Office" (NOICE, ★ 5, Apr 2 at 3:43 PM), "CMUSV Office" (OK NOICE, ★ 4, Apr 7 at 3:41 PM), and "CMUSV Office" (BEST CUSTOMER EVER!, ★ 5, Apr 12 at 2:46 PM). Each review has a navigation bar at the bottom.

Subscription and Notifications



As a customer, I can subscribe to my favorite shops and receive push notifications when new food items are posted for sharing or sale.

Also, I am able to receive notifications when my order status is changed.

The screenshot shows the SharePlate mobile application interface. At the top, there's a header with the time (9:19), signal strength, battery level (95%), and the URL `shareplate.pages.dev`. Below the header is the SharePlate logo and a navigation menu icon (three horizontal lines). The main content area features a photograph of a white building with arched windows and doors, identified as the CMUSV Office. Below the photo are two buttons: a red "Unsubscribe" button and a blue "Direction" button. The shop name "CMUSV Office" is displayed in green, followed by an "OPEN" button. A yellow circular rating indicator shows a star rating of 3.9. Below this, there's a section for "Recent Items" showing a thumbnail of bread with the caption "bread from this morning". A green "Checkout" button is at the bottom, along with a total of "\$0.00". The footer includes the text "SharePlate - Team 2" and a navigation bar with icons for back, forward, search, and more.

This screenshot shows a notification message on the SharePlate app. The message reads: "Subscribed shop update! CMUSV Office has updated their menu. Check it out!" It includes a "View" button. Below the notification, there's a "WELCOME BACK" message. The main content area shows a profile for "James Chang" (Customer) with a 3.9 average rating and 8 reviews, along with a "All Reviews" button. There are sections for "Shop Collections" featuring "Subway" (Fast Food) and another "Subway" listing, both with 3.8 ratings and descriptions like "No description provided." and "Asian Fast Food". The footer is identical to the previous screen.

This screenshot shows another notification message: "Your order is completed" from "CMUSV Office" (COMPLETED). It includes a "View" button. The main content area shows a profile for "James Chang" (Customer) with a 3.9 average rating and 8 reviews, along with a "All Reviews" button. There are sections for "Shop Collections" featuring "Subway" (Fast Food) and another "Subway" listing, both with 3.8 ratings and descriptions like "No description provided." and "Asian Fast Food". The footer is identical to the previous screens.

Q&As and Discussion

"Less Waste, More Taste, Share the Plate"



Lessons Learned