"Less Waste, More Taste, Share the Plate"

SharePlate

Team 2

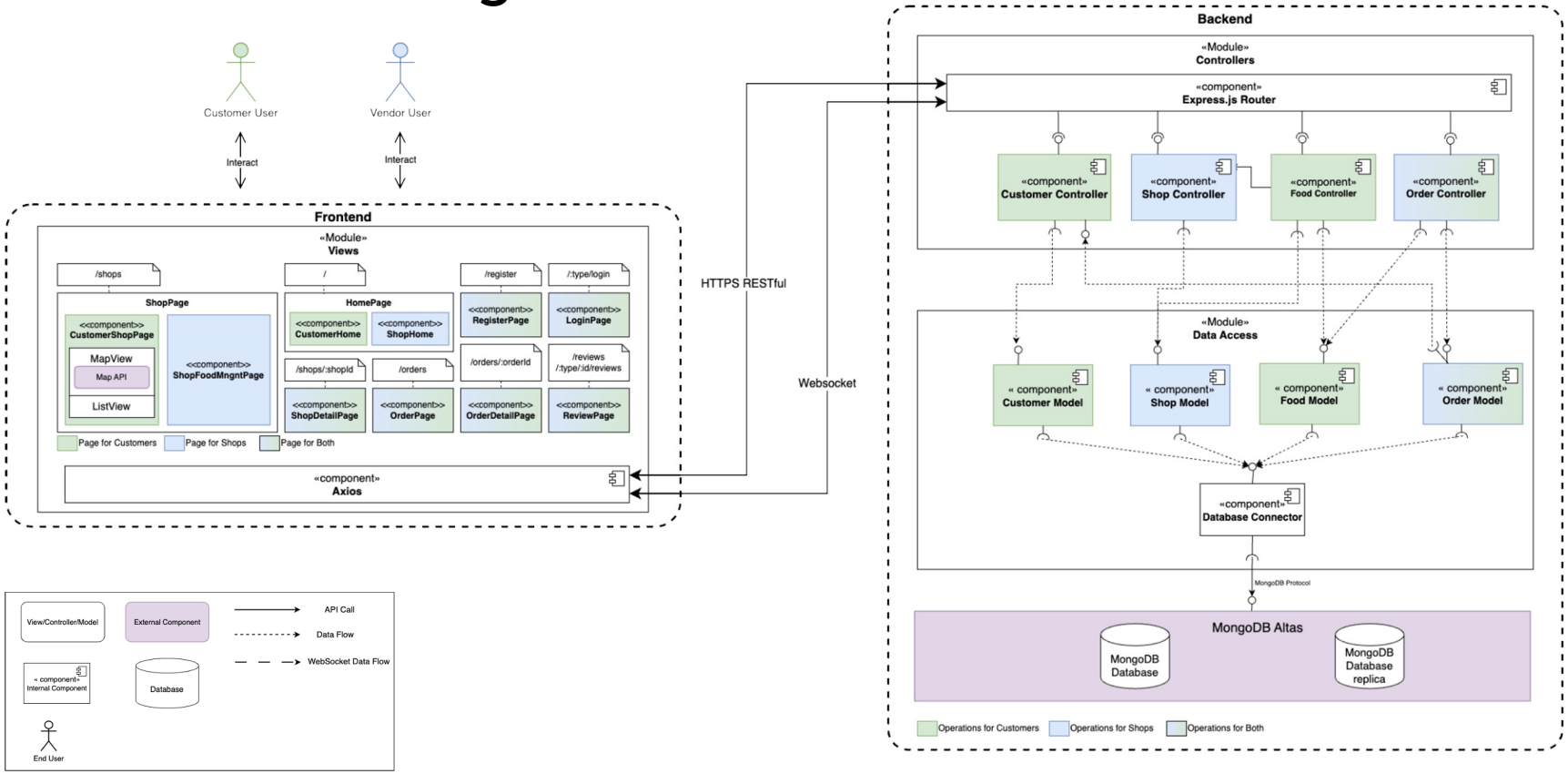
John Su, Martin Chou, James Chang, Bruce Cai, Lona Lu

Architecture Overview

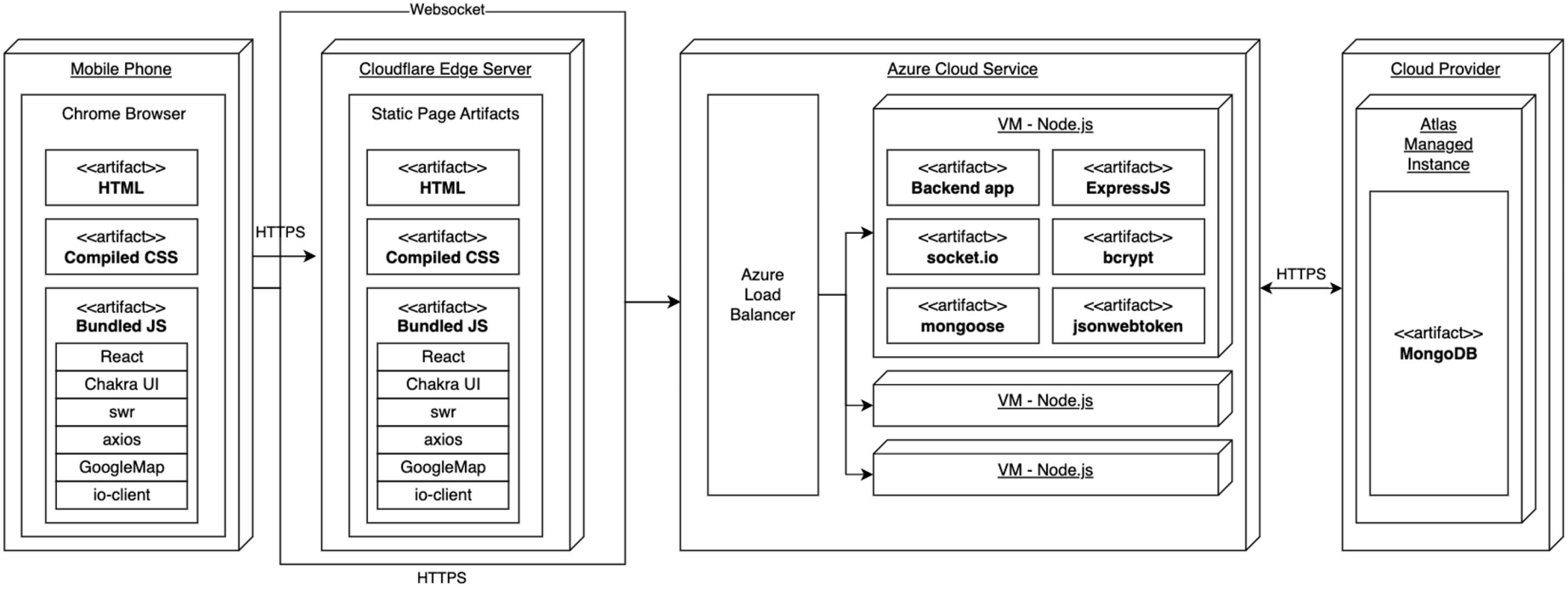
"Less Waste, More Taste, Share the Plate"

SharePlate

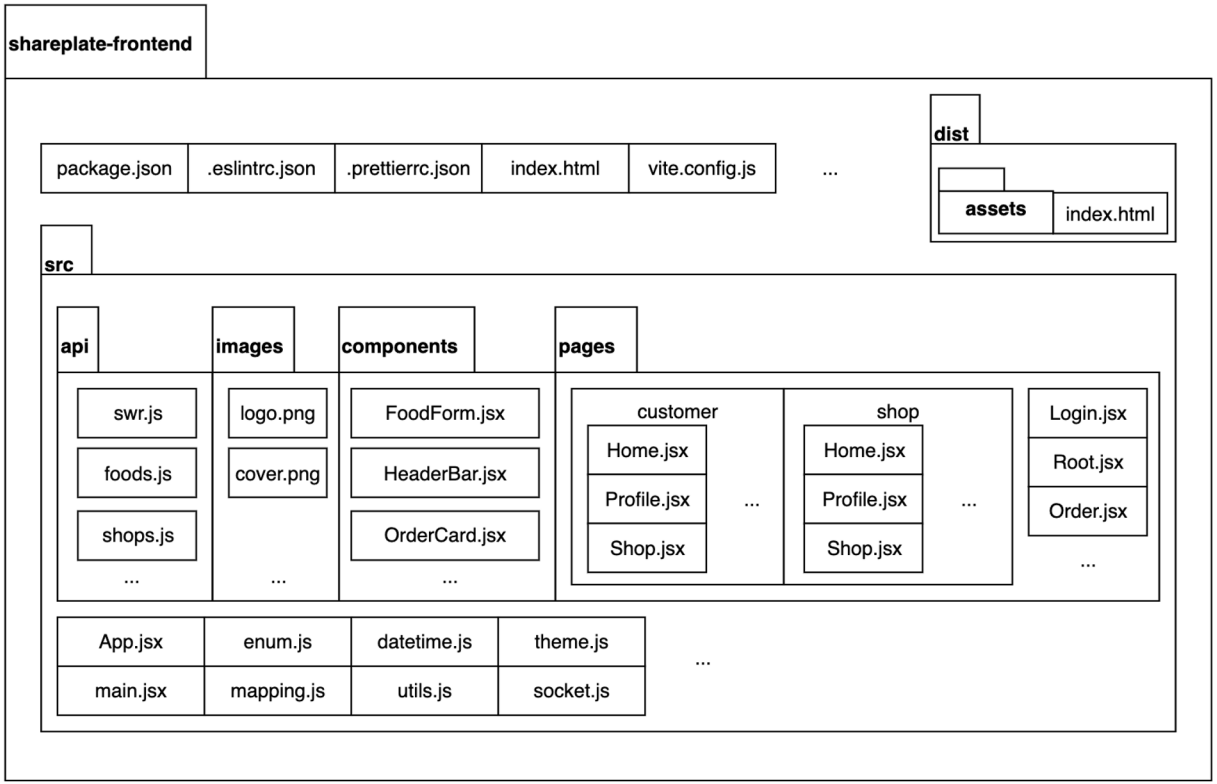
Architectural Diagram

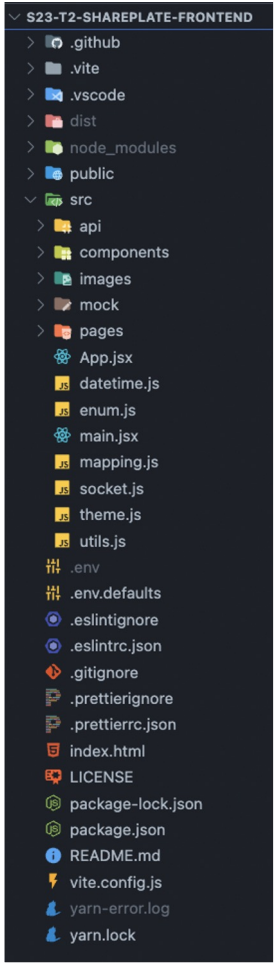


Deployment View

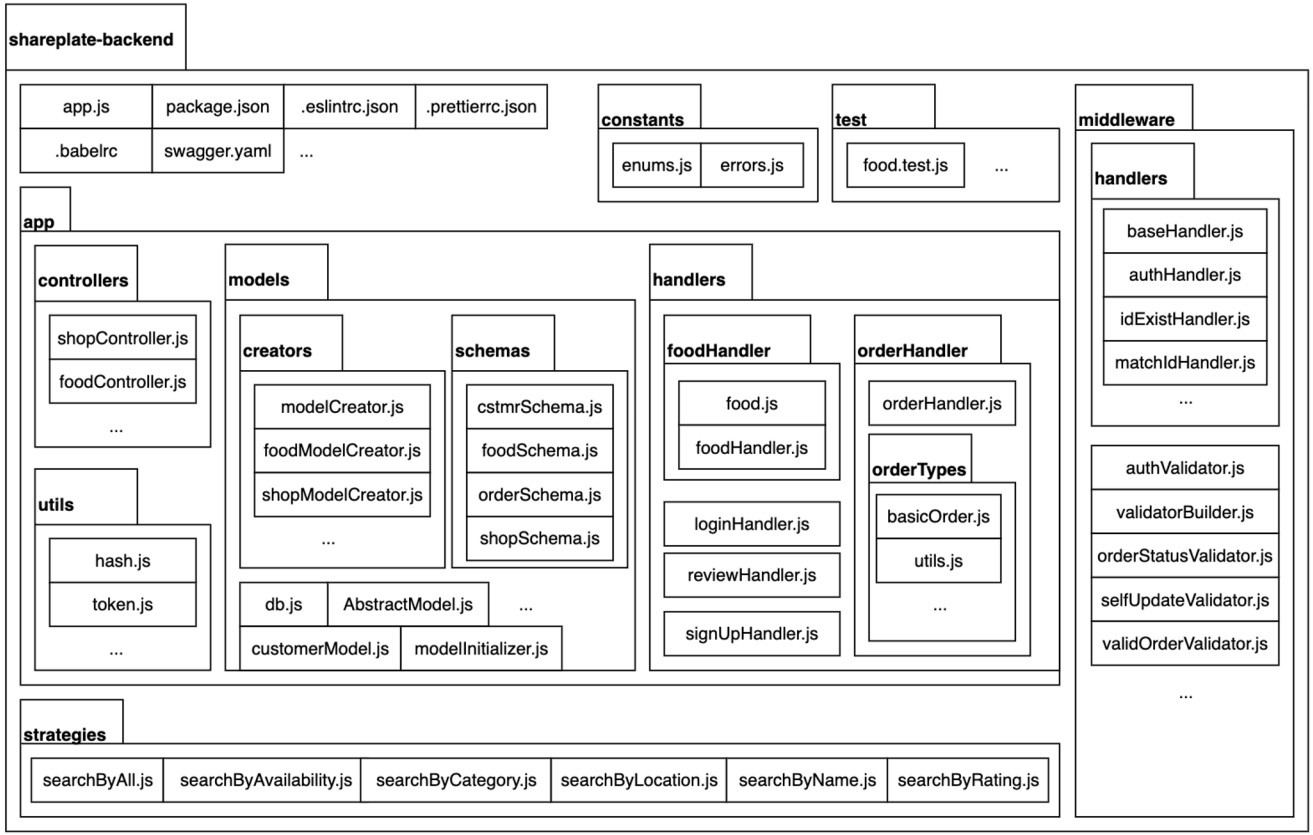


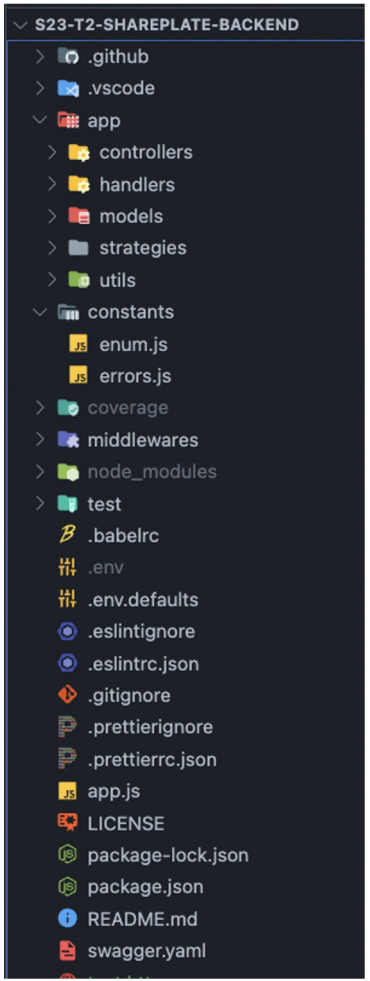
Package View - Frontend





Package View - Backend





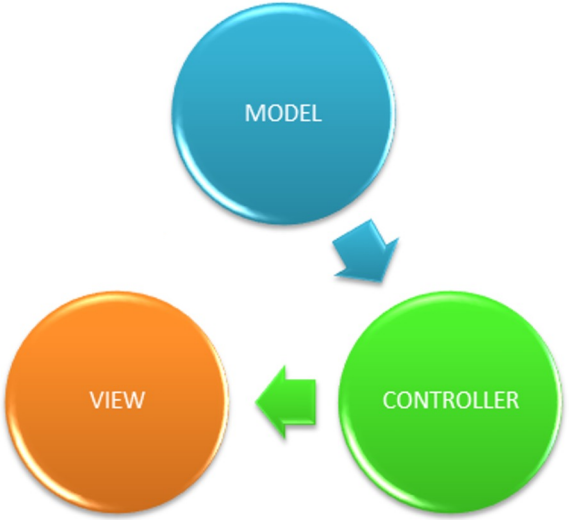
MVC

Team development capability

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

Application requirements

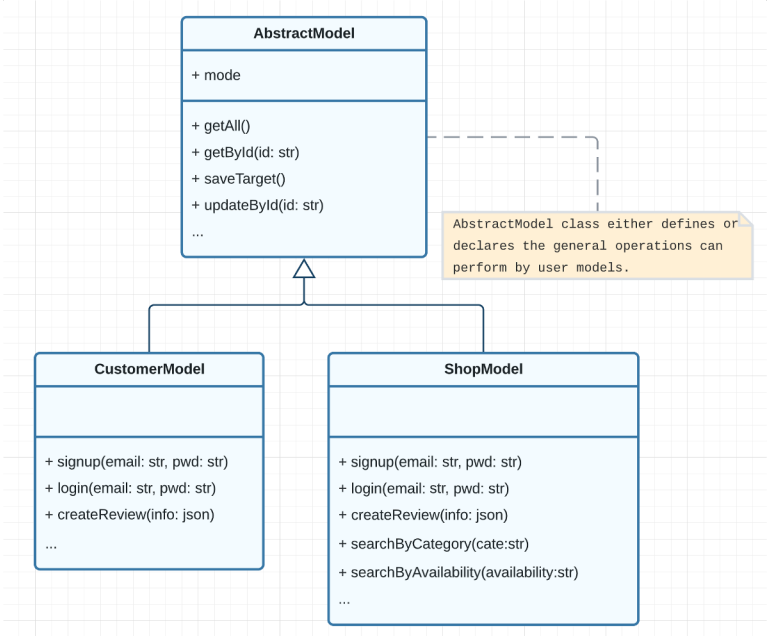
- Low latency
  - Displaying food inventory
  - Placing an order during rush hour
- Reduce unnecessary overhead
  - Maintain high response performance



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

AbstractModel.js

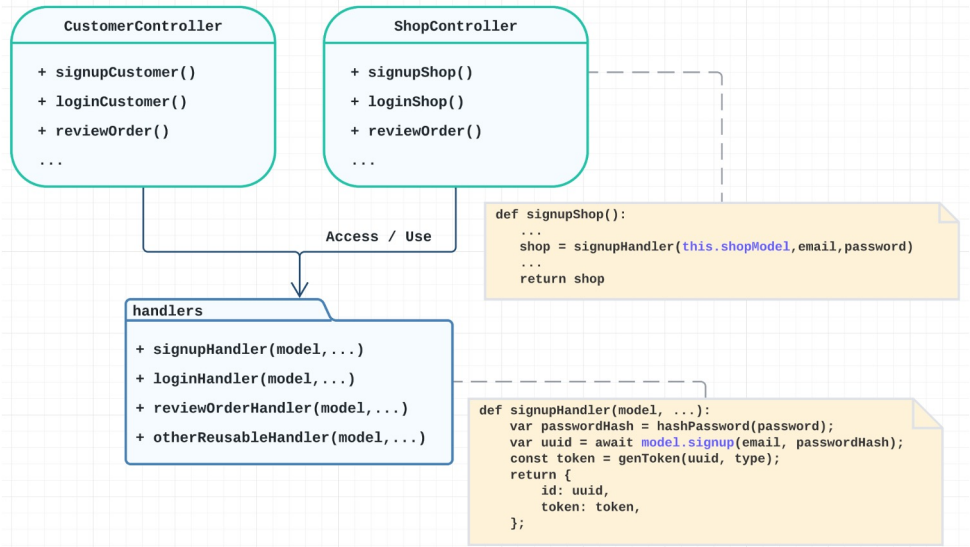
```
class AbstractModel {
  constructor(DBs, mode) {
    this.DBs = DBs;
    this.setMode(mode);
  }

  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

loginHandler.js

```
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;
```

CodeImage

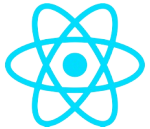
signupHandler.js

```
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;
```

CodeImage

Design Decisions

"Less Waste, More Taste, Share the Plate"
SharePlate



Reusability

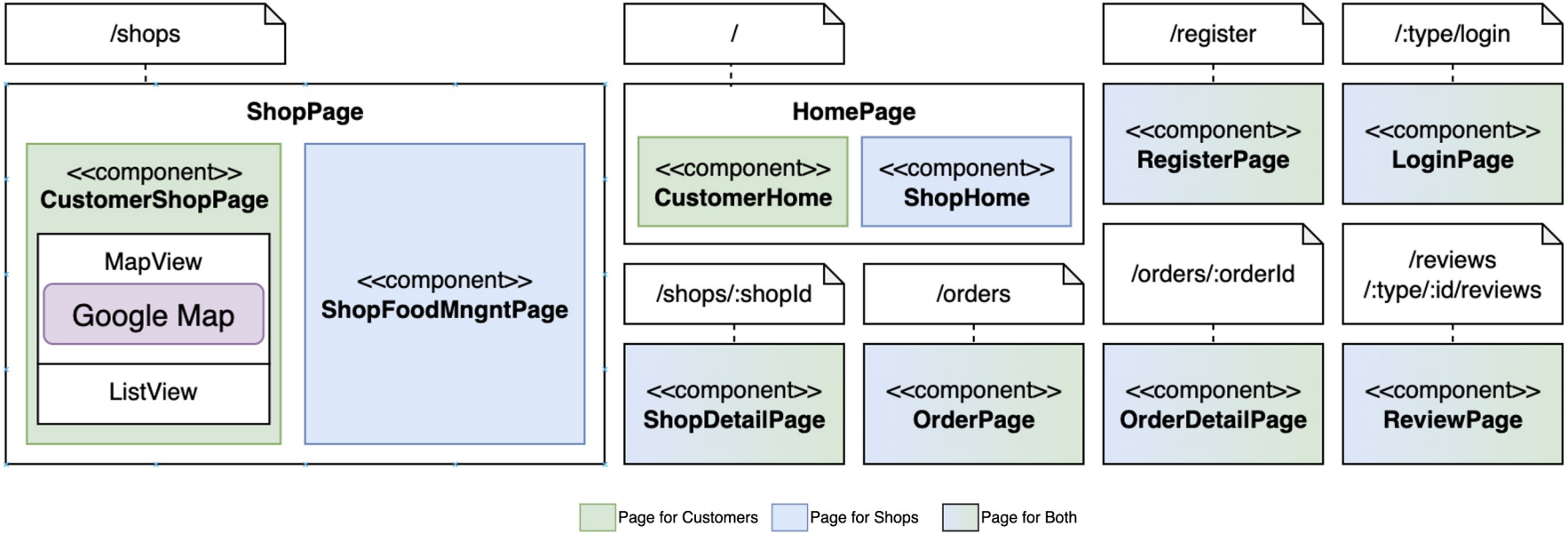
"Less Waste, More Taste, Share the Plate"

SharePlate

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.

s23-t2-shareplate-frontend - App.jsx

```
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.jsx

```
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;
```

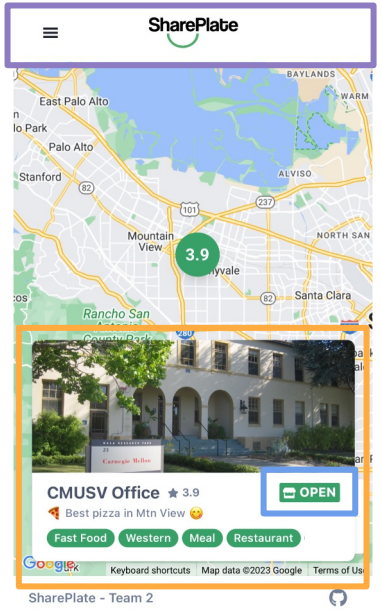
s23-t2-shareplate-frontend - Shop.jsx

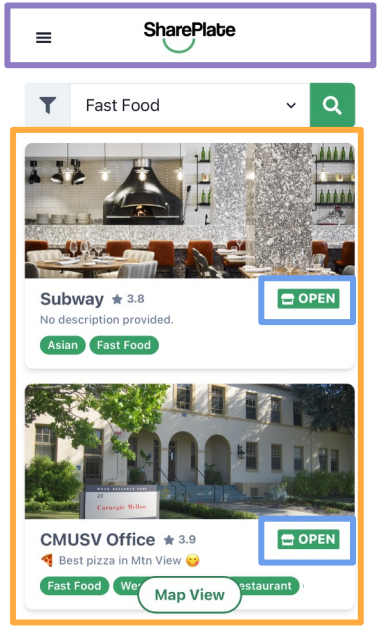
```
1  import CustomerShop from "./customer/Shop";
2  import PageStrategy from "./PageStrategy";
3  import ShopShop from "./shop/Shop";
4
5  function Shop() {
6    return PageStrategy(<CustomerShop />, <ShopShop />);
7  }
8
9  export default Shop;
```

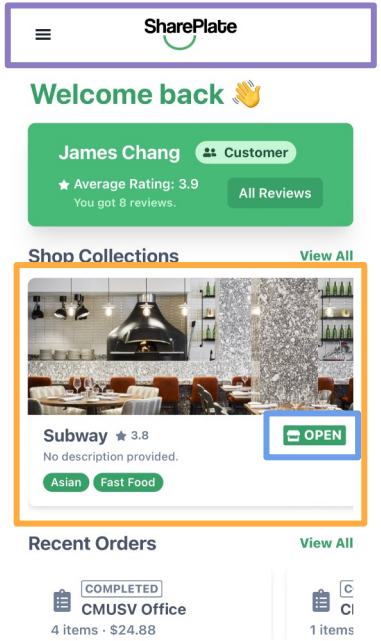
React

Reusability - Components

Reusable component in various pages to prevent code duplication.








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

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

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

Availability

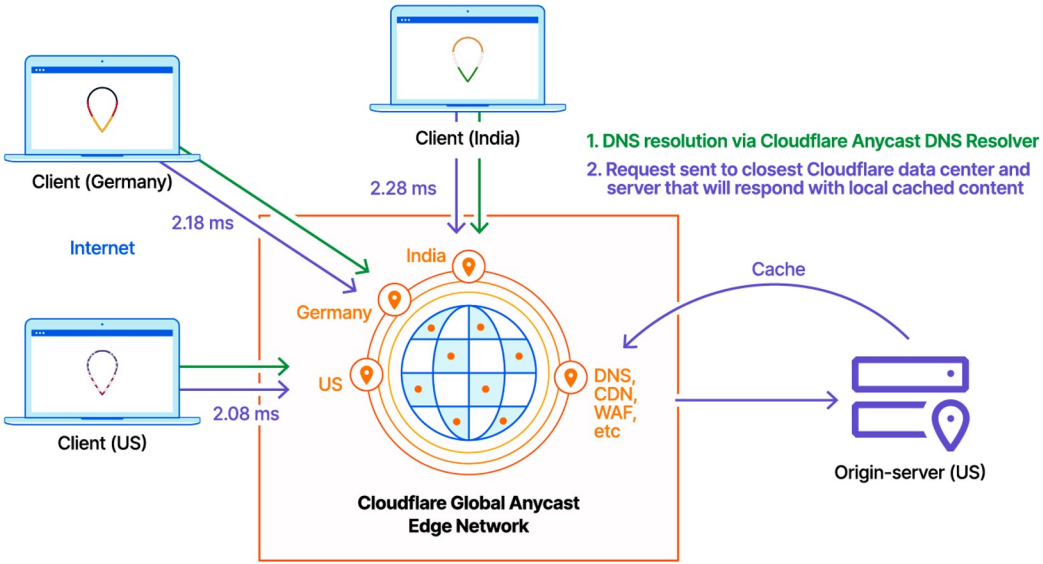
"Less Waste, More Taste, Share the Plate"

SharePlate

Deployments

Availability - Cloudflare Pages

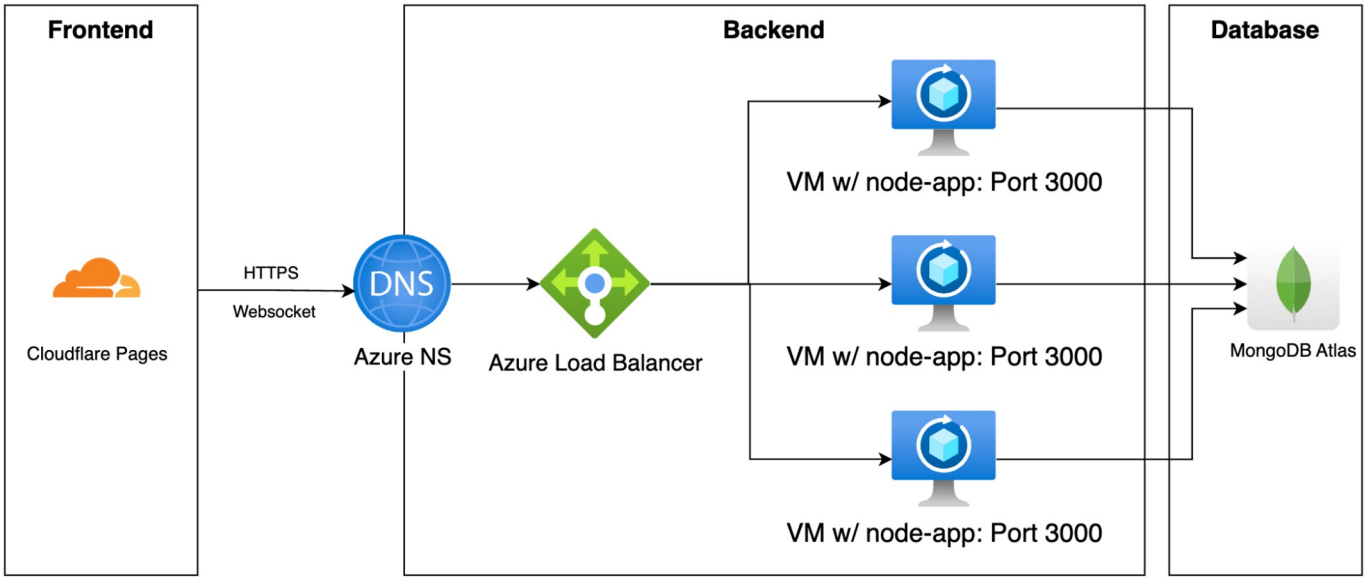
Deploying our CSR frontend artifacts on Cloudflare Pages leverages its CDN functionality. CDN gives us advantages on Availability and Performance.

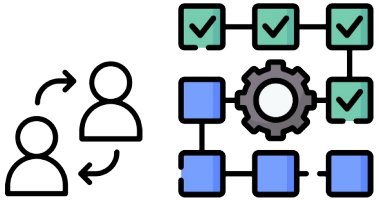


Deployments

Availability - Azure Load Balancing

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





Testability

Testability

Testability

- Mock using Jest

Improvement required:

- No need to have a real DB connection

Mock request and response

mock.js

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

CodeImage

Mock JWT & Bcrypt Lib

mockJWT.js

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

CodeImage

mockBcrypt.js

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

CodeImage

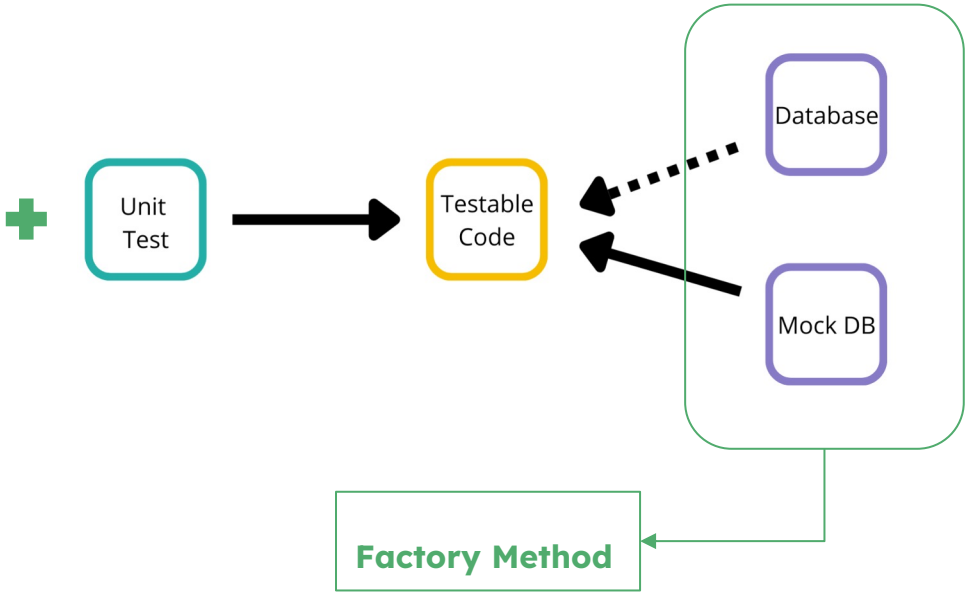
Testability

Testability

- Mock using Jest

Testability

- Dependency Injection



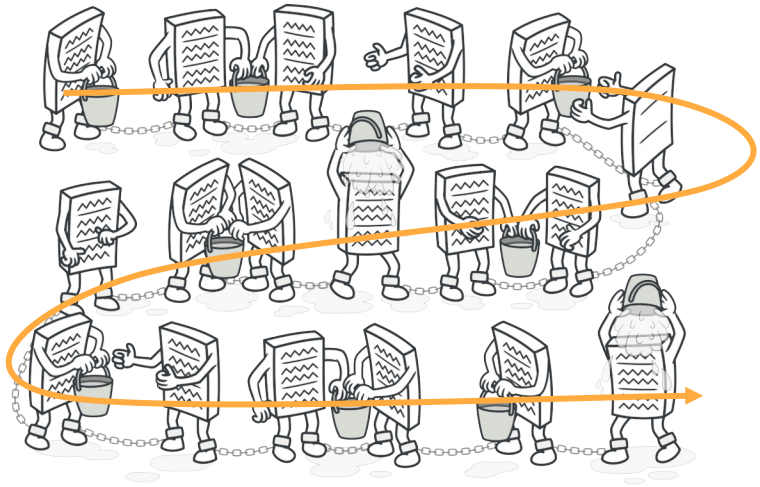
Improvement required:

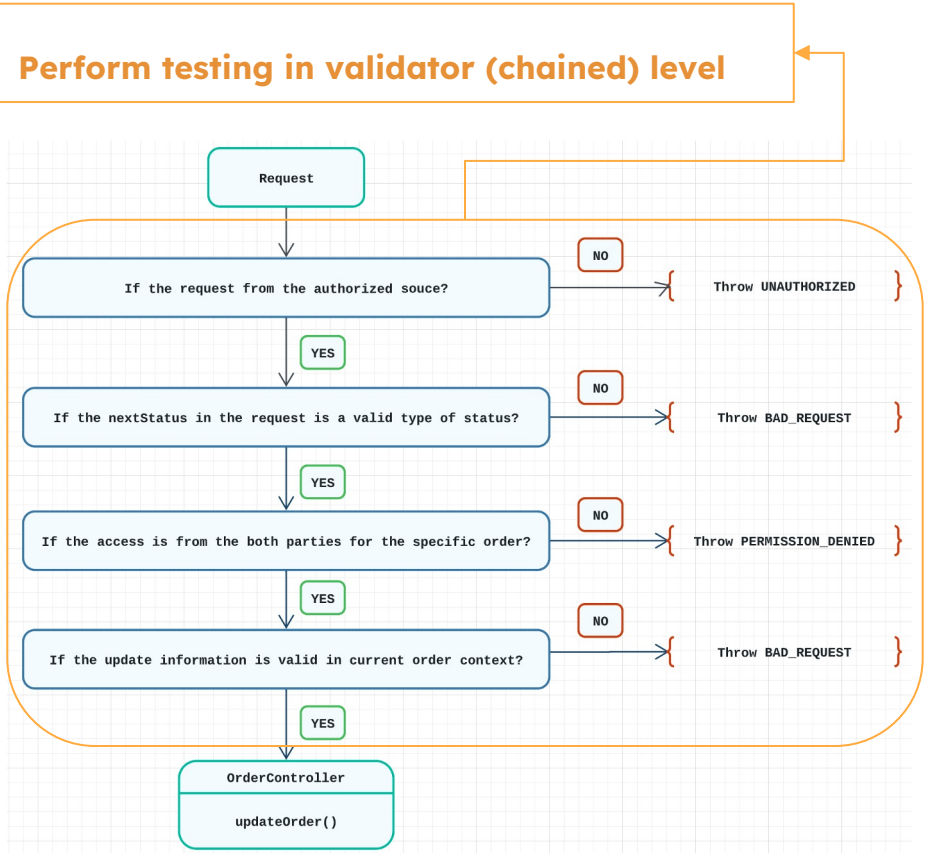
- No need real DB connection

Testability

Testability

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

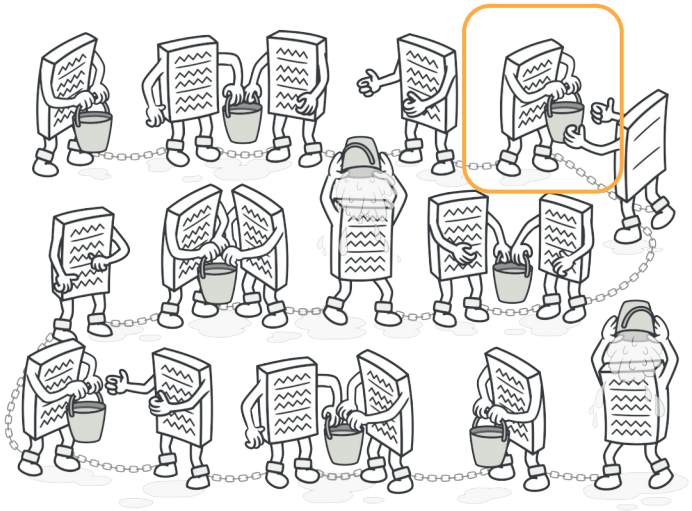


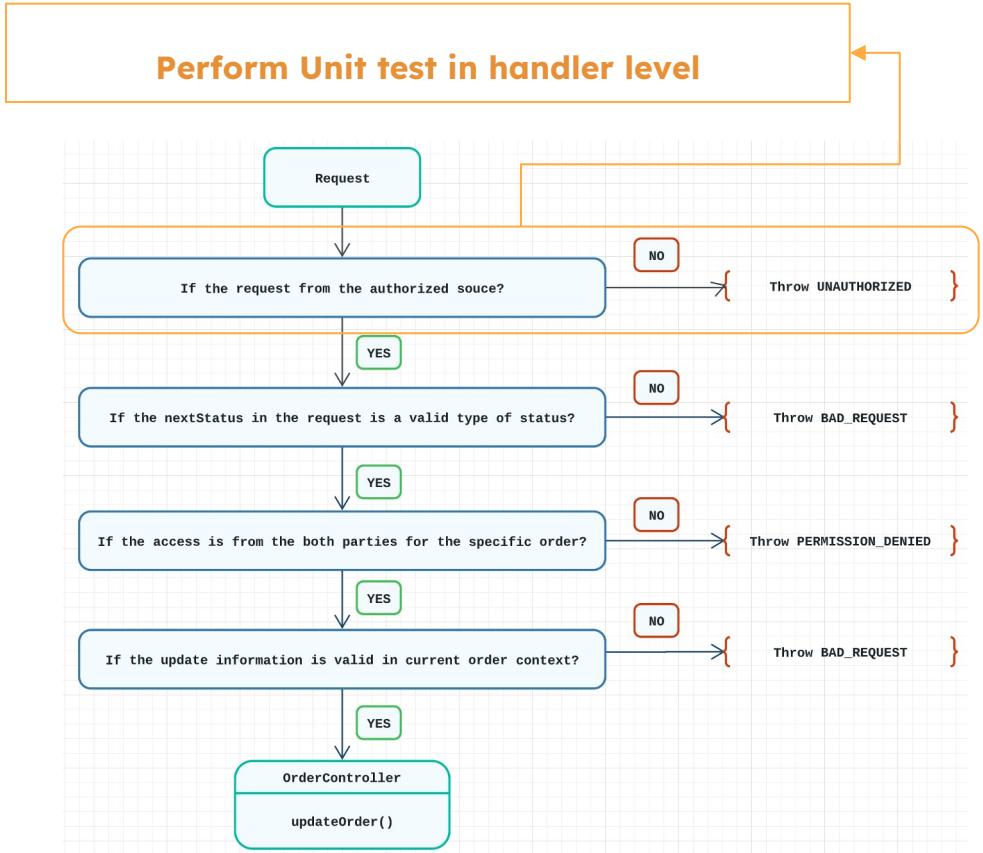


Testability

Testability

- Chain of Responsibility
  - Perform test in different granularity





Design Patterns

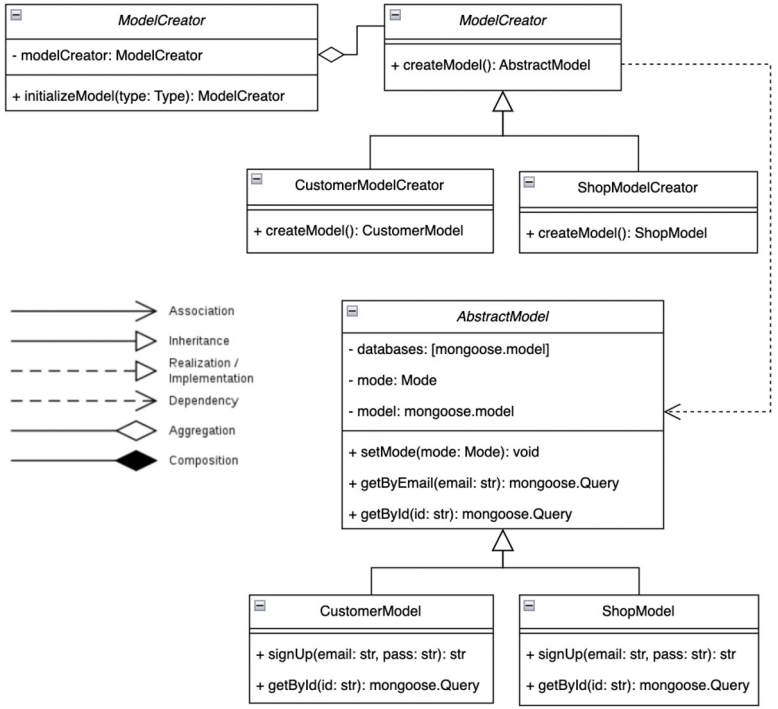
"Less Waste, More Taste, Share the Plate"

SharePlate

Design Pattern - Factory Method

How did it solve our problem?

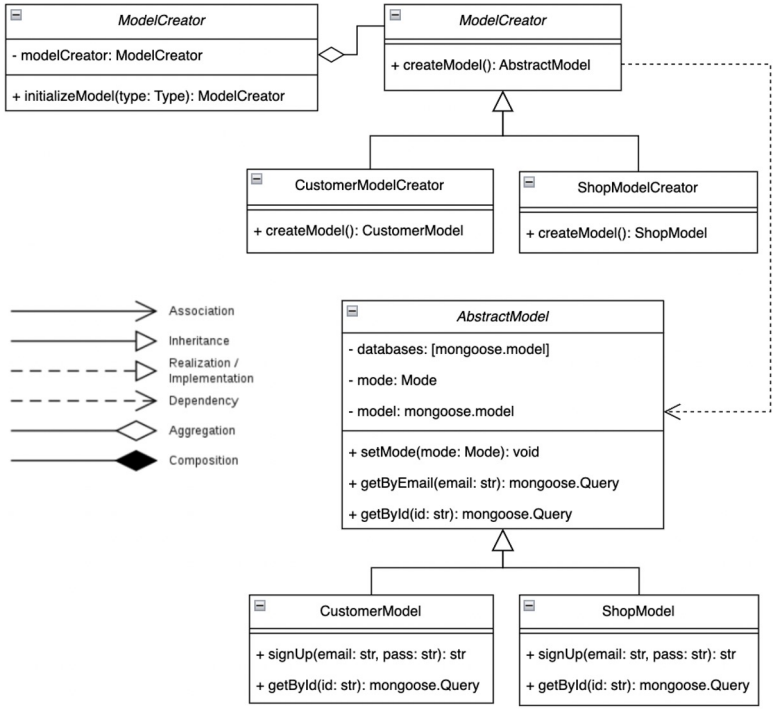
- It helps creating models - data access objects.
- It allows us use different subclasses to determine the type of objects to be created. (Customer / Shop)



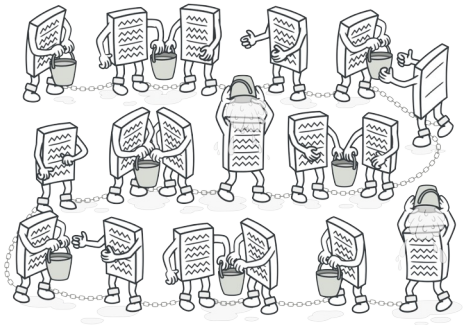
Design Pattern - Factory Method

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.

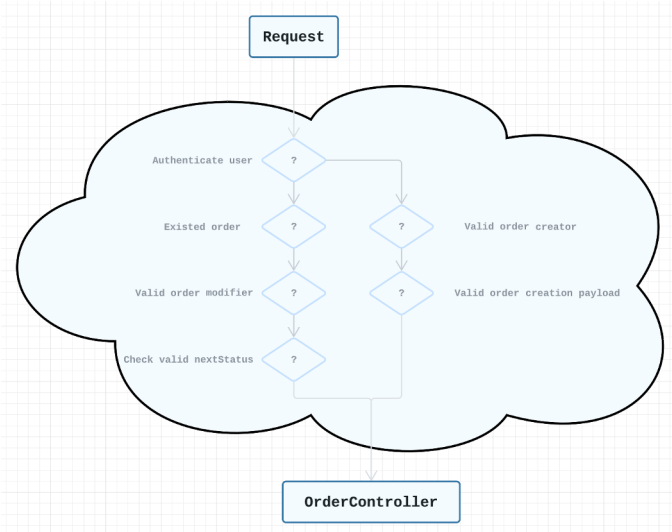


Design Pattern - Chain of Responsibility



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.

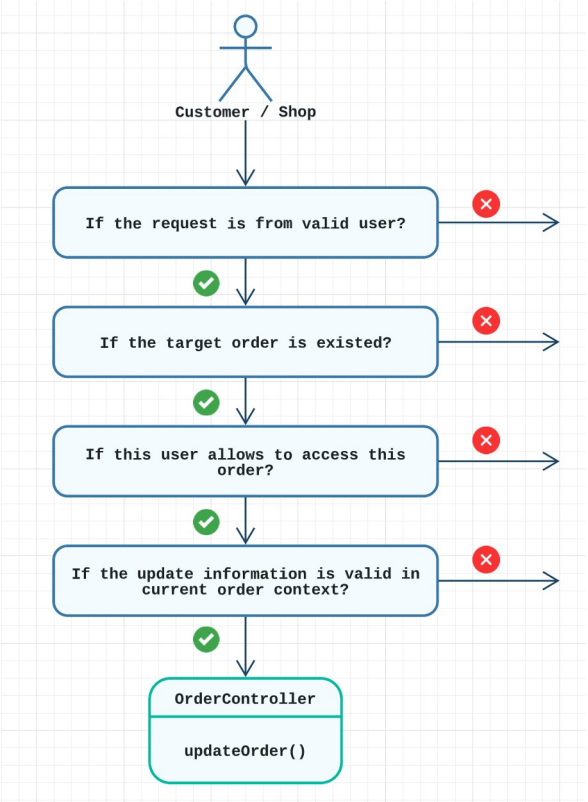


Design Pattern - Chain of Responsibility

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.***



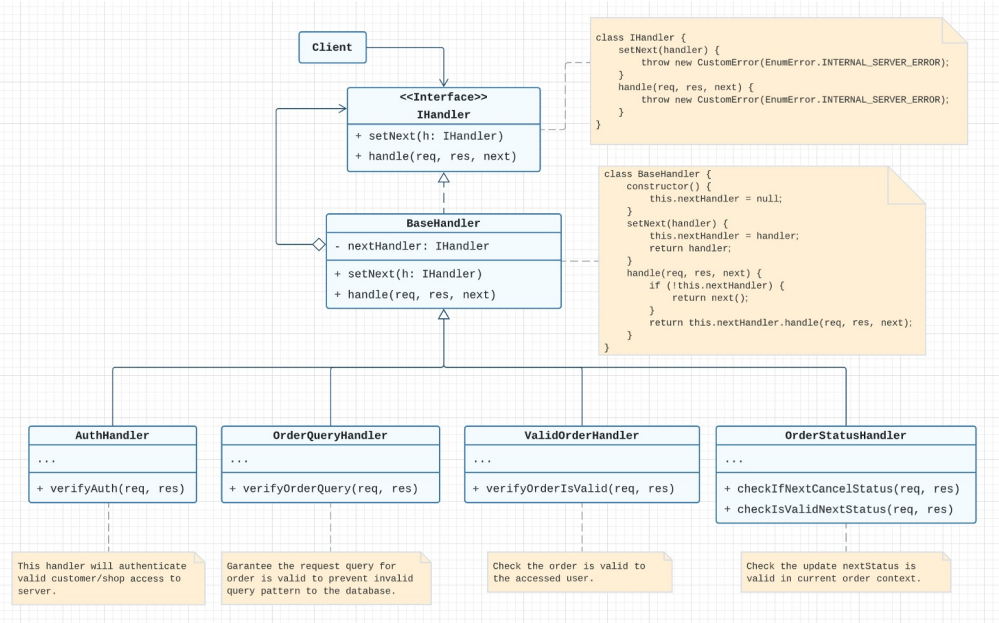
Design Pattern - Chain of Responsibility

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.

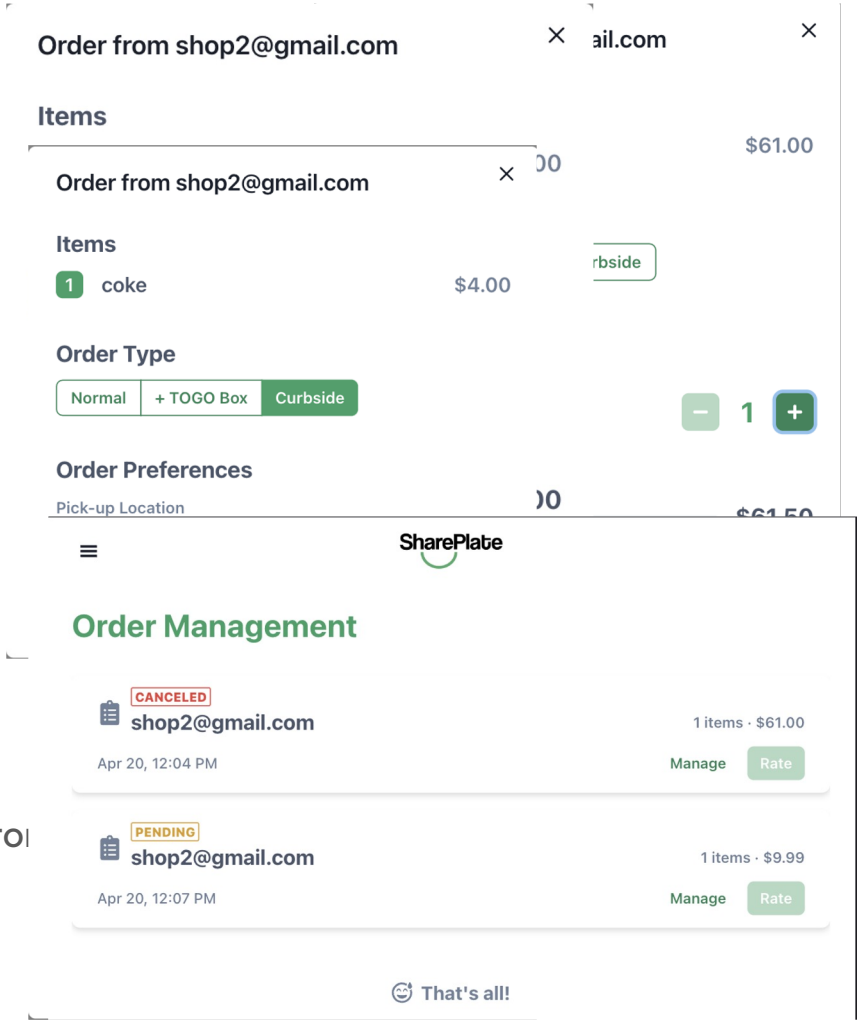
Design Pattern - Decorator

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?

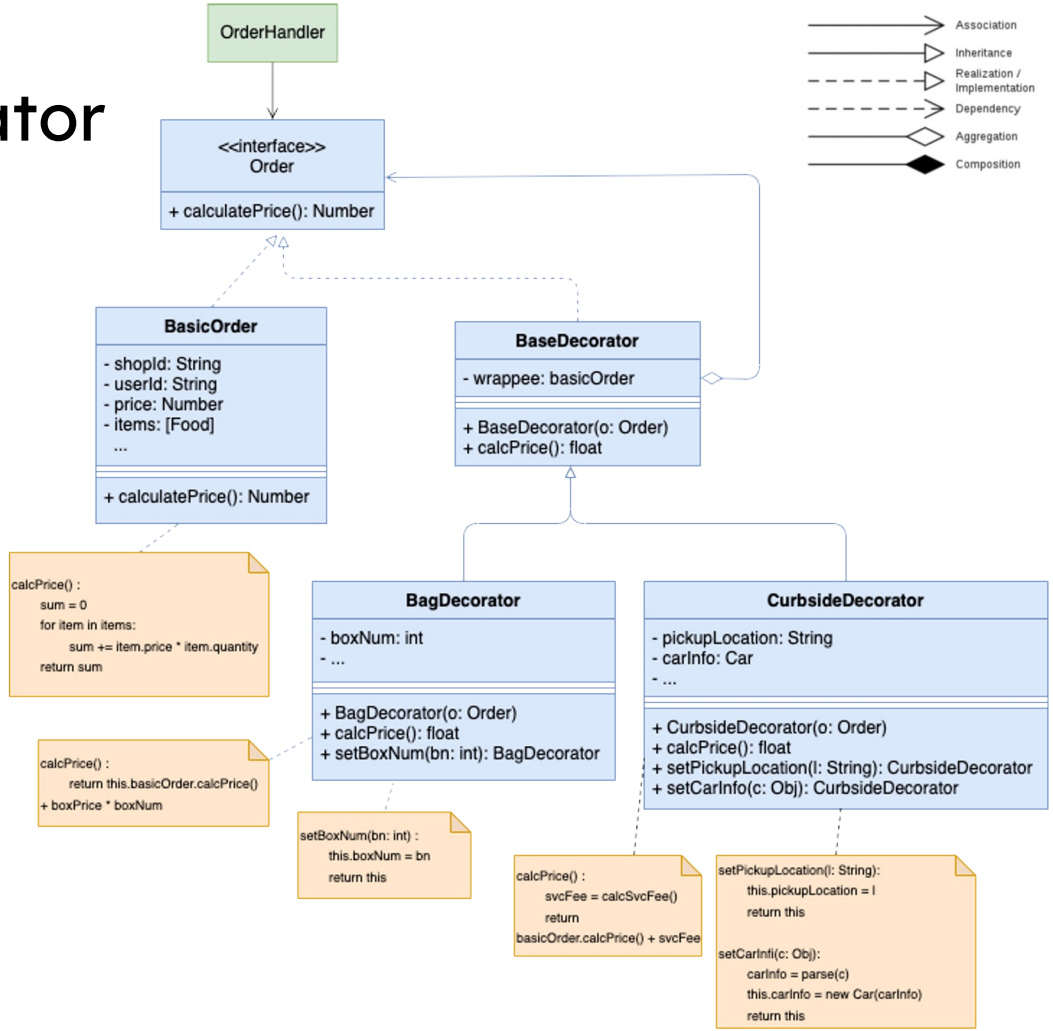
- Keep basic order untouched
- Reuse the basic order by injecting it to new decorator and describe new features in decorator



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

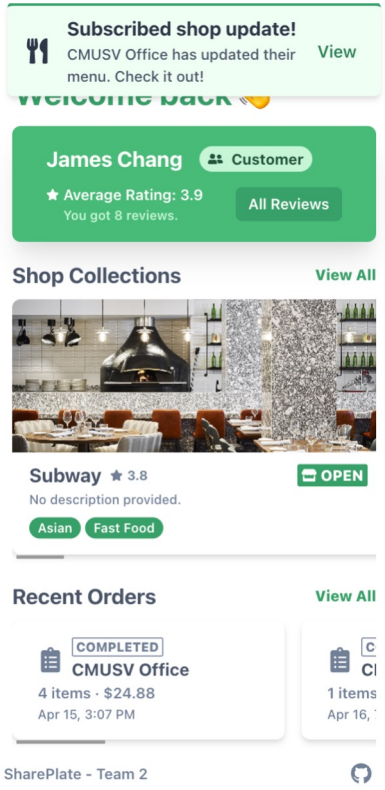


Design Pattern - Observer

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



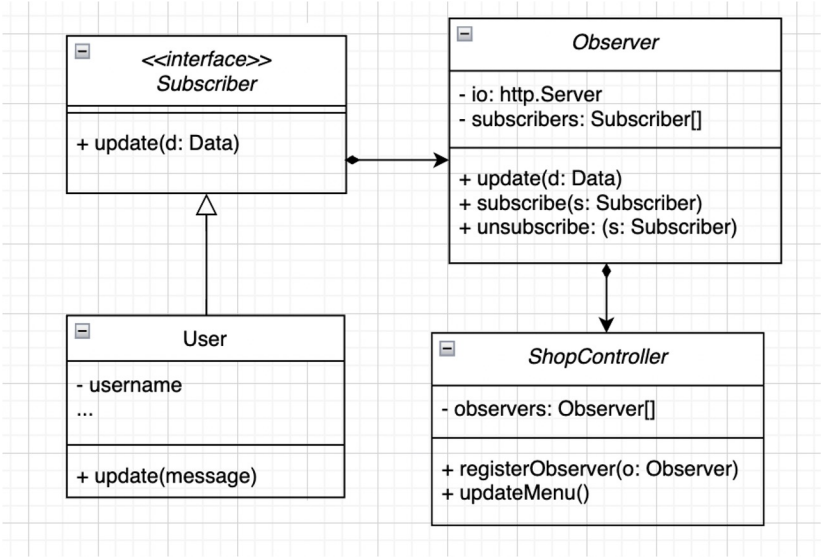
(Shop) (Customer)

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.

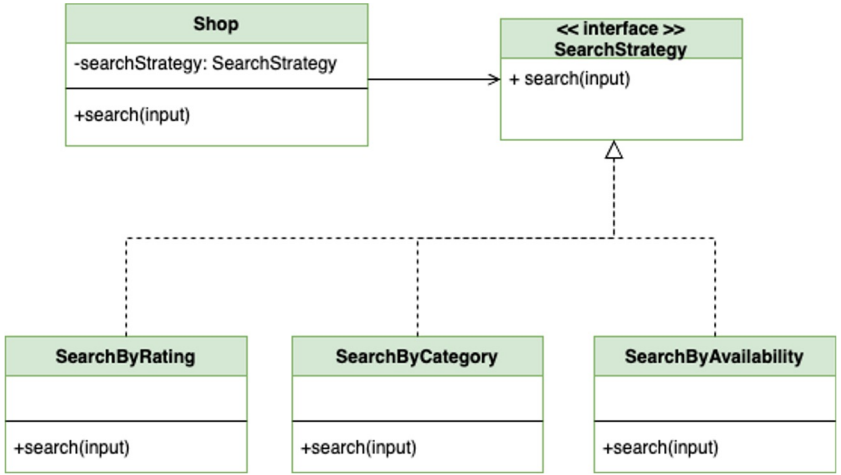



```
update(data) {
  for (let i = 0; i < subscribers.length; i++) {
    const subscriberId = subscribers[i];
    this.io.to(subscriberId.toString()).emit("menuChanged", { data });
  }
}
```

Design Pattern - Strategy Method

What we visioned - how will it solve our problem?

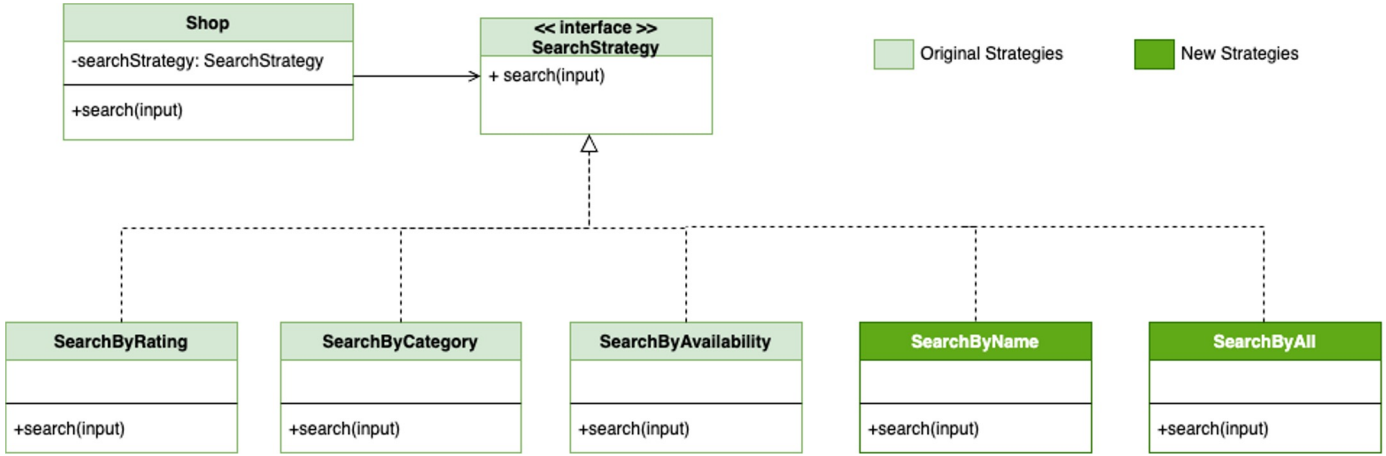
- Feature: allow customers to search for shops using different criterions
- Problem: redundant code handling all possibilities, difficult to manage and maintain
- Solution: Strategy method with encapsulation



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



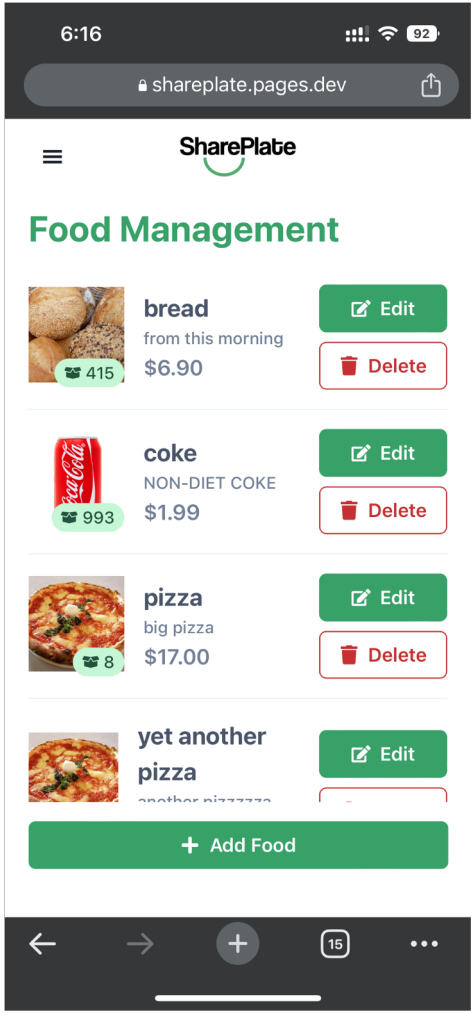
Demo

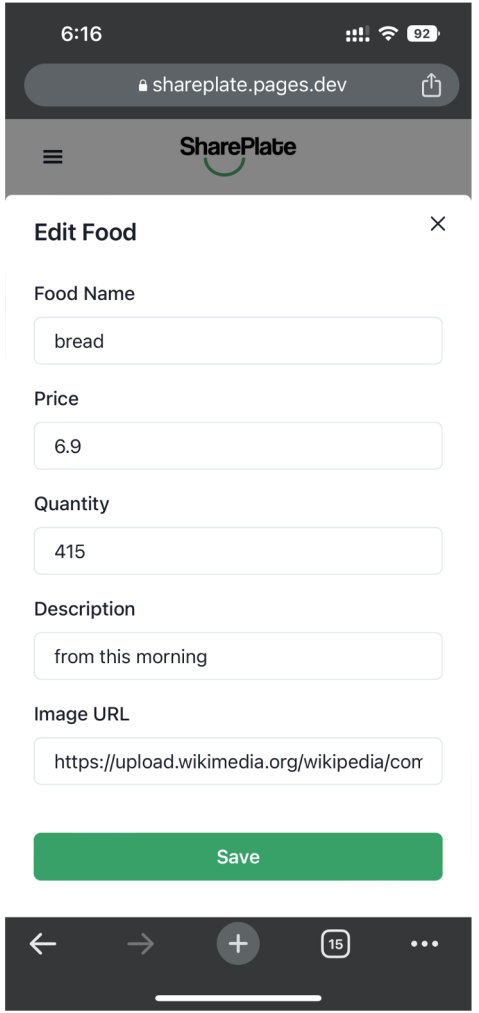
"Less Waste, More Taste, Share the Plate"

SharePlate

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.

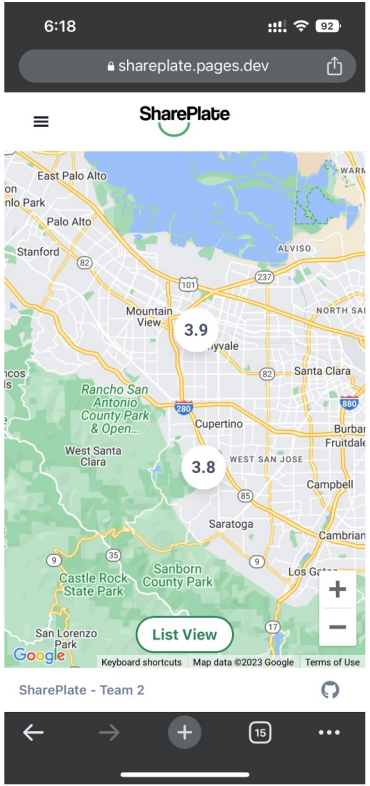


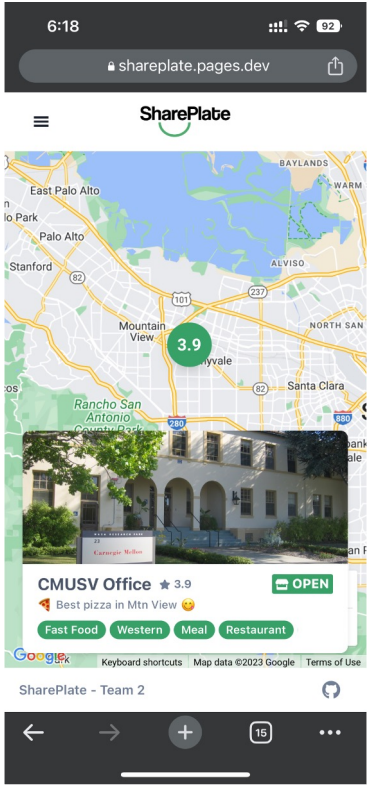


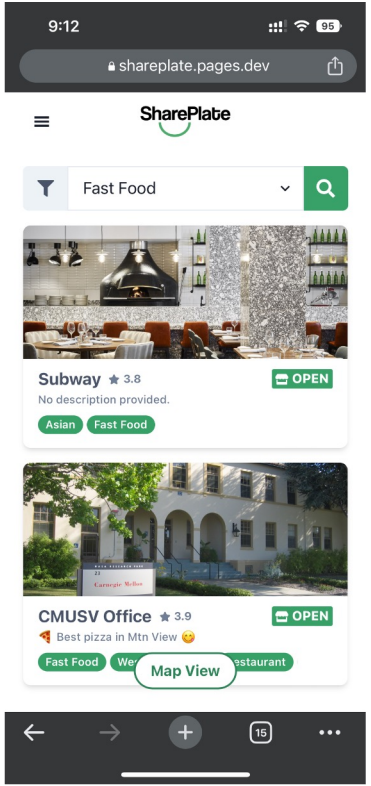
Map & Tag-based Search

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.



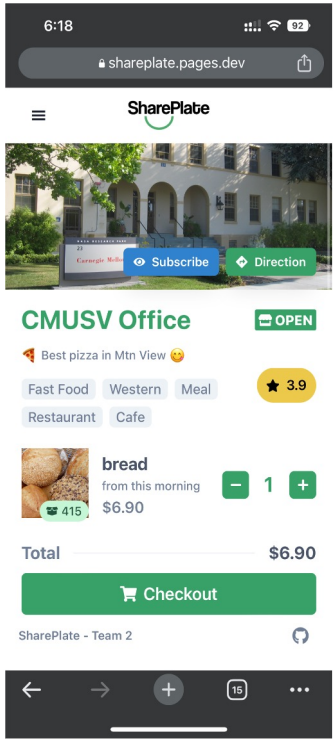


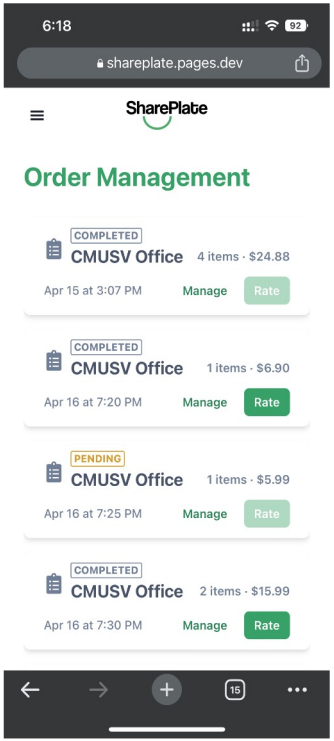


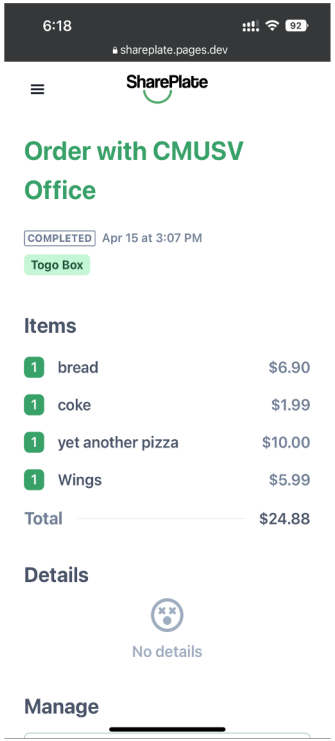
Food Ordering and Pick-up

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.



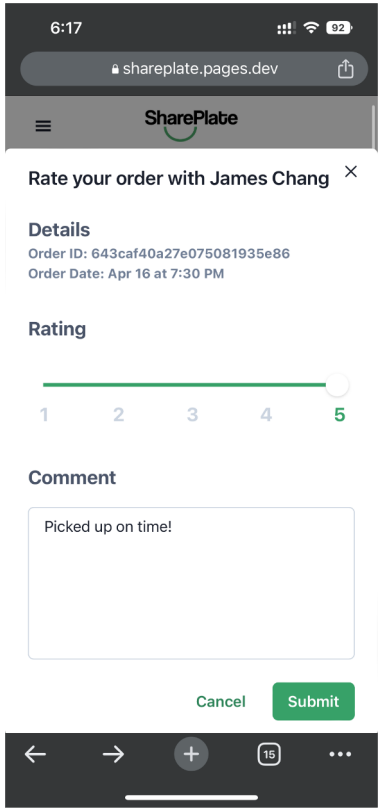


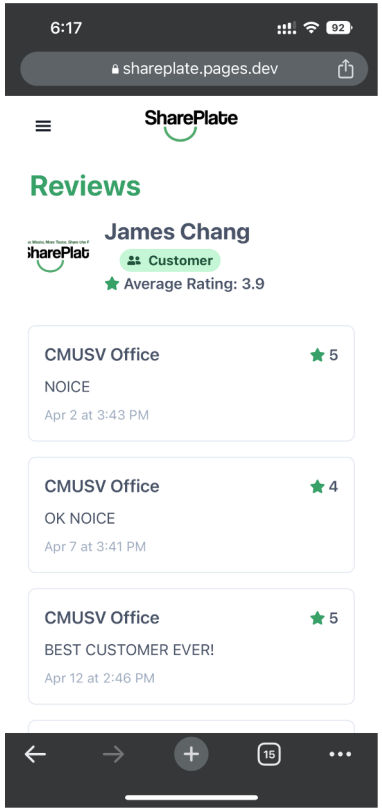


Feedback and Rating System

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.

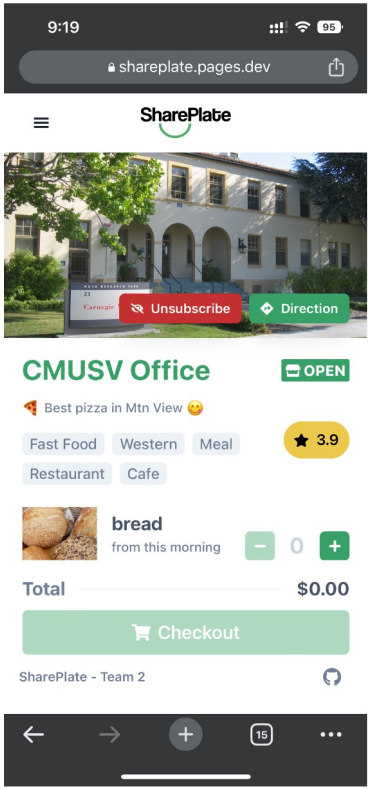


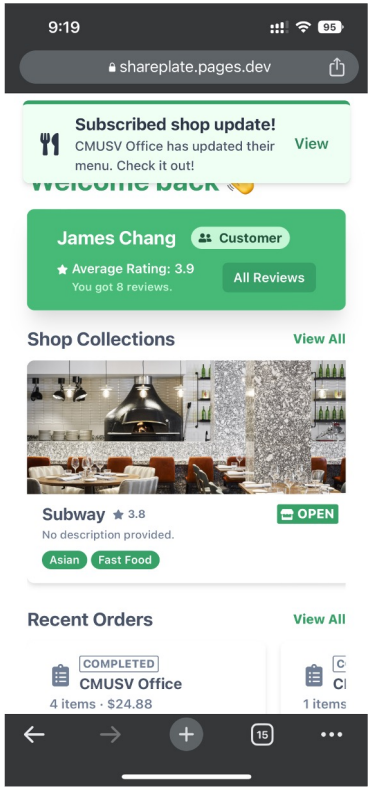


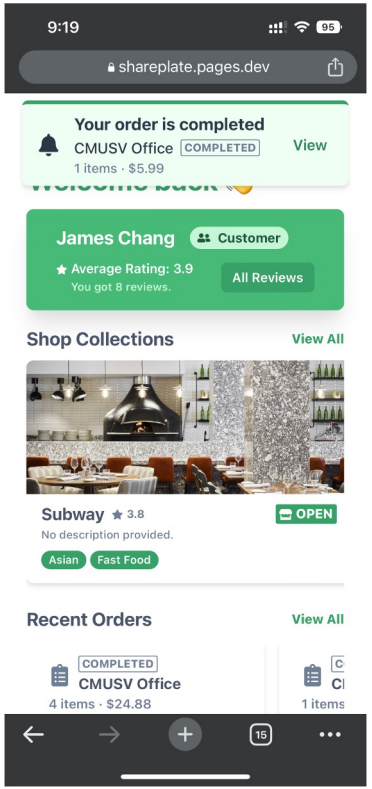
Subscription and Notifications

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.







Q&As and Discussion

"Less Waste, More Taste, Share the Plate"
SharePlate

Lessons Learned