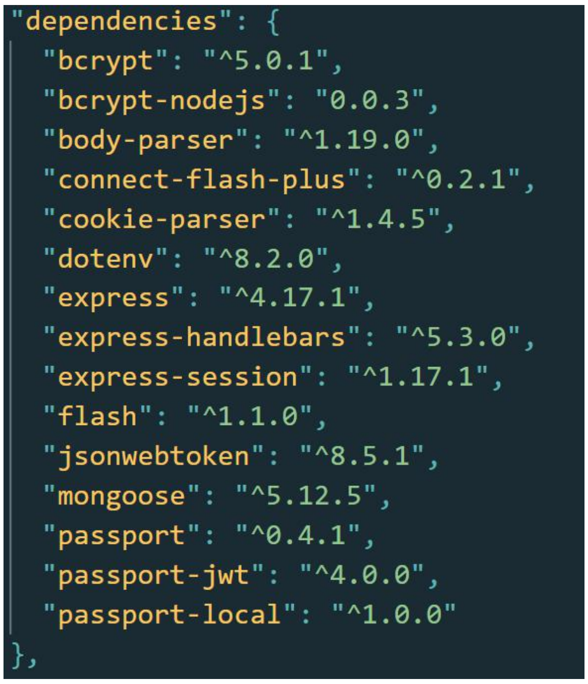
Development Process

Technologies

Our *Foodstop* web app is developed as an single server-rendering application, that is using one server for the whole app. We choose Nodejs with express as our backend tool and Handlebars as frontend template. For database we use Mongodb with mongoose for better model management. We also adapt passportjs for user validation alongside with many other tools that help us develop. Here is a full dependency list of our application:



```
"dependencies": {
  "bcrypt": "^5.0.1",
  "bcrypt-nodejs": "0.0.3",
  "body-parser": "^1.19.0",
  "connect-flash-plus": "^0.2.1",
  "cookie-parser": "^1.4.5",
  "dotenv": "^8.2.0",
  "express": "^4.17.1",
  "express-handlebars": "^5.3.0",
  "express-session": "^1.17.1",
  "flash": "^1.1.0",
  "jsonwebtoken": "^8.5.1",
  "mongoose": "^5.12.5",
  "passport": "^0.4.1",
  "passport-jwt": "^4.0.0",
  "passport-local": "^1.0.0"
},
```

As a Team

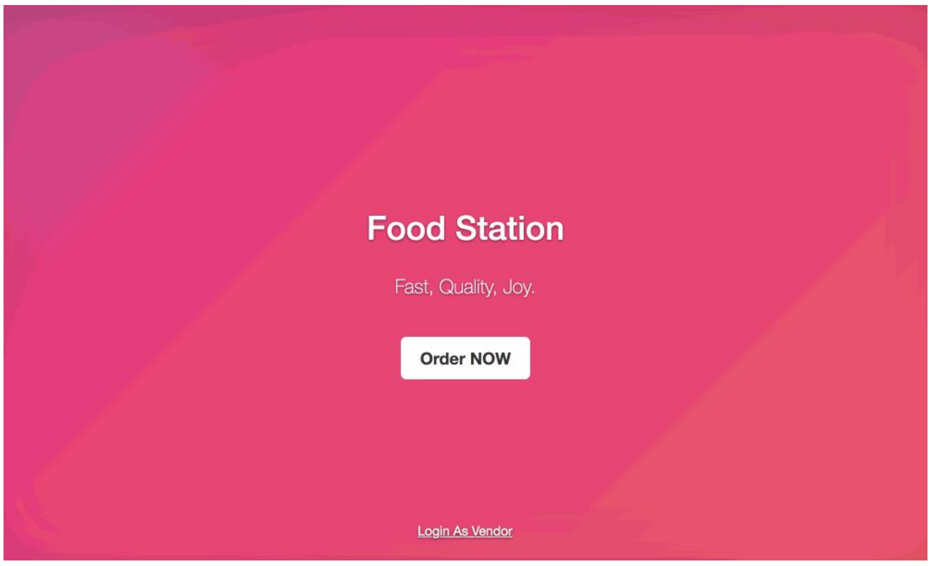
We are a five-people team that tend to do works together, we often discuss issues together or do peer coding. Consider the nature of our project's structure, many works are interrelated. Nevertheless, each team member has a general role, which being:
Chenghao Li & Yanting Mu doing frontend, Yiyang Huang & Zhihao Yang & Yutong Wu doing backend.
We use Git as our version control tool which helps manage repository, since our project is divided into different deliverables, we use branching in git to clarify

different stages of the project. Before each stage we would gather together to discuss the upcoming tasks and assign them to each members. In order to achieve higher productivity, we often do peer coding instead of having everyone together. When encountering issues, we discuss them in chat first and have a zoom meeting if necessary. It is important to have an active communication inside the group.

System Architecture

Front-end

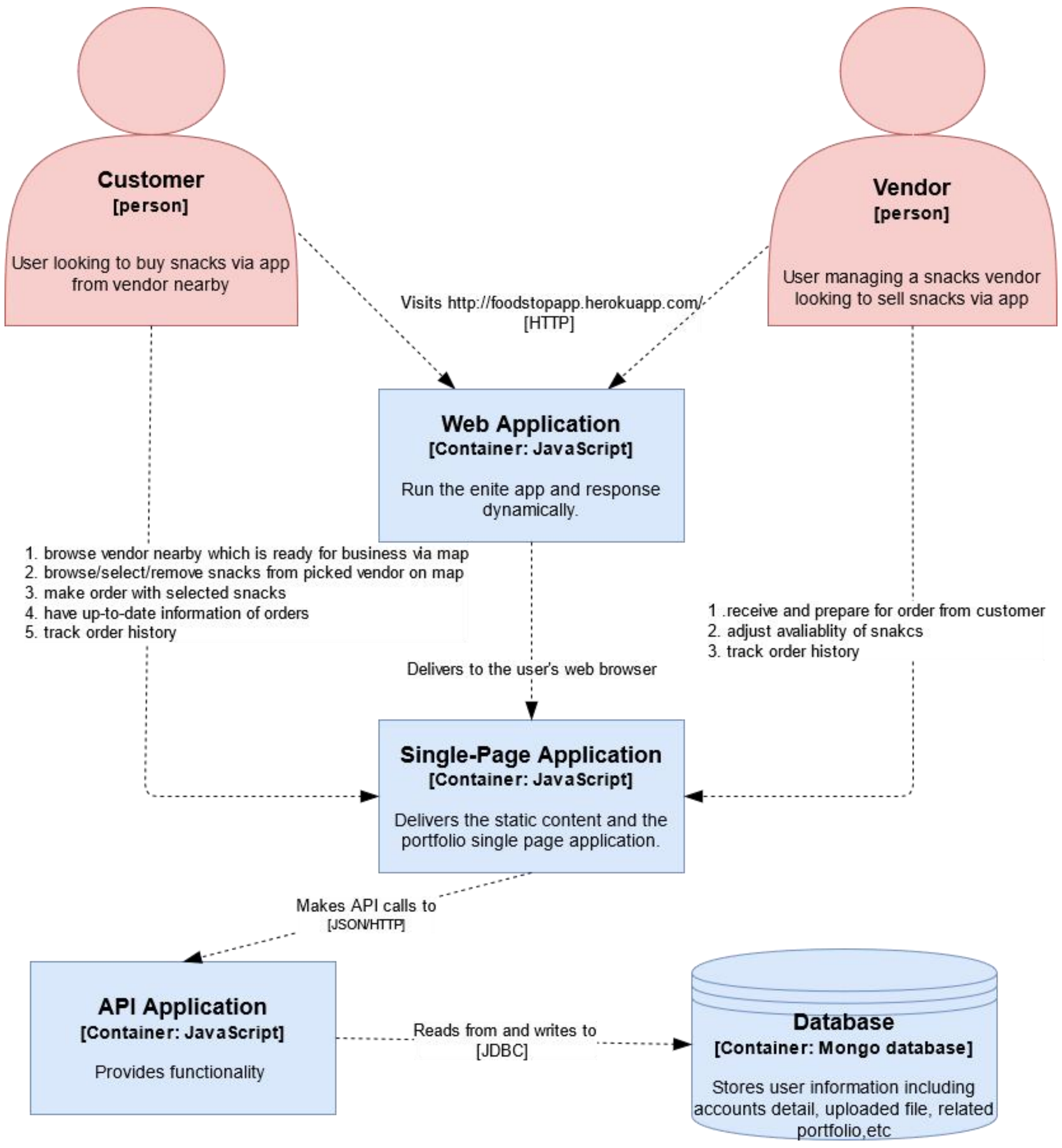
The General idea of this web page's design is Simplicity and ease of use. As the modern UI follows the rule of flat design, it is worth trying to make a website "flat" and "easy to use". Therefore, All buttons we made use of are simple white with no borders at all. The Web Page is color focused and modularised design make either customer or vendor found it easy to read and use.
To implement the front-end part of our project, a sample css from bootstraps integrated component has been used. To Achieve the animation color effect on our webpage, it is doable to apply animation functions directly into our webpage (change the whole page's color consistently). However, we found a better implementation way to achieve this visual effect. The "main.css" basically set the whole background different colors at each corner. Then, place a magnifier onto the page and the browser actually get the view through the maginifier. Lastly, create a dynamic function that moves the magnifier consistently on the page. By Combining those three steps we get a visual effect that the webpage is changing color on dynamic.



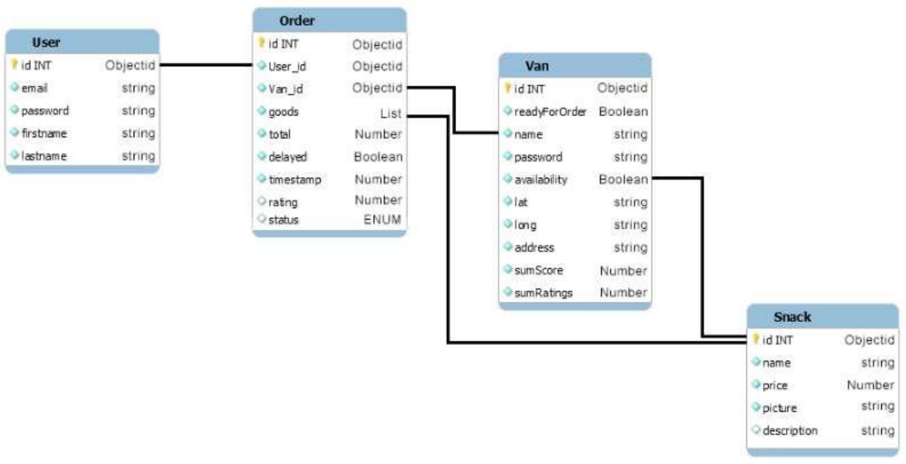
Backend

Because of the nature of frontend template, most of the logics of our web app are handled at backend. We wrote different functionalities as APIs and reference their routes in handlebars. Different domains are separated using routing, and the backend route are differentiated from web route so nothing gets mixed up. The two main domains are customer and vendor, which correspond to two types of user using our web. Since the functions needed by these two group are quite different, we wrap the functionalities into two controller components, each contains all the functions they will be using. We also wrote two routers to handle various requests of the two domains to improve modularity.

Many of these functions are essentially performing actions on our database, which we achieve by manipulating mongoose models. In general, we have four models which correspond to real world entities: Order, Snack, User and Van. User and Van map to the two types of web user customer and vendor, which contain different fields in order to achieve different features. They both can login use their email/name and password, except customer needs to register first but vendor is pre-registered. Customers want to view the menu and order at different vans, so we created an Order model to store all the relevant information. An order has five different states 'outstanding', 'fulfilled', 'complete', 'cancelled' and 'inCart', represented as an enum. When the user add an item to cart, an order is automatically created for him which has 'inCart' state. We implement the cart as a state rather than a separate field so that customers can have different carts at different vendor, since the availability of items varies among the vendors it is better to separate them. After the customer finish filling his cart and place the order, we change the order state to 'outstanding'. At this stage the customer has 10 minutes to modify or cancel his order before it gets finalized. If he decide to cancel it, the order state becomes 'cancelled' and won't be visible to the customer anymore. If he decide to modify the order, then the 10 minutes limit is reset and keep the order at 'outstanding' state. Once the items are prepared it changes to 'fulfilled' state and becomes 'complete' when the customer receive their order. During the whole process order stores user ID, vendor ID, ordered items etc. to ensure the correct delivery. All the items in the menu or order are referencing items in Snack model, which contains the relevant information of snacks that vendors sell (since vendors have the same menu). Finally, the vendor includes address information which are shown on the map, different vendors will store their own states like availability of items and whether is open for business.


Customer
[person]
User looking to buy snacks via app from vendor nearby
Vendor
[person]
User managing a snacks vendor looking to sell snacks via app
Visits http://foodstopapp.herokuapp.com/
[HTTP]
Web Application
[Container: JavaScript]
Run the enite app and response dynamically.
1. browse vendor nearby which is ready for business via map
2. browse/select/remove snacks from picked vendor on map
3. make order with selected snacks
4. have up-to-date information of orders
5. track order history
1 .receive and prepare for order from customer
2. adjust avaliablity of snakcs
3. track order history
Delivers to the user's web browser
Single-Page Application
[Container: JavaScript]
Delivers the static content and the portfolio single page application.
Makes API calls to
[JSON/HTTP]
API Application
[Container: JavaScript]
Provides functionality
Reads from and writes to
[JDBC]
Database
[Container: Mongo database]
Stores user information including accounts detail, uploaded file, related portfolio,etc

App structure diagram



Database structure diagram

Design

We built our website mockup with Adobe XD after reading the instruction, demonstrating all the features that our final work is expected to have. The Customer App allows showing a map of our van location, and the overall menu (all the snacks in our database) to customers without logging in, and for those who have an account, they can select a certain van, check its available products, place orders and modify their orders. There was also a side profile card, showing the registered name and email. Overall, the App will be easy, experience Oriented and scannable.
As for the Vender's App, we want a clear call to every action, and the design was focused on utility and right spacing to help avoid misoperation. Besides checking orders placed by our Customer App, it could also serve as a cashier system. Vendors will be able to set item availability, geo-location, and their status (accepting or rejecting new orders)

There are two maps used in our app design, one is on the home page of the Customer's App, showing where the vans are, and the other is on the vendor's main page, used to help vendors do some fine adjusting on their location.
During the developing progress, showing markers on the customer's map did cause some trouble since the map's <script> will run before other elements, we cannot access the database like our normal practice, instead, we used fetch() function in the script, to download .json file from our database directly, before the markers placing took place.

In register page, we used a Regular Expression "(?=.*[a-zA-Z])(?=.*[0-9])(?=.{8,})", which means at least one English char, at least one number and at least eight chars in total length, to check if the password is strong enough. The checking is run whenever a key is up, the <input> lost focus, or the mouse is moving across the <input>, so we can check if the password meets the requirement.