

# Developer Guide

## Real Producers

Authors: Pratiksha Sharma, Dan Sun, Alexander Tseng, Charlie Wang

### 1. Software Stack Overview

Real Producers' production life-cycle manager is constructed using the MERN stack. MERN stands for Mongo, Express.js, React.js, and Node.js. All the layers were written in Javascript. The team is approaching this project using a server-client model -- half of the team members work on the frontend and the other half work on backend and servers.

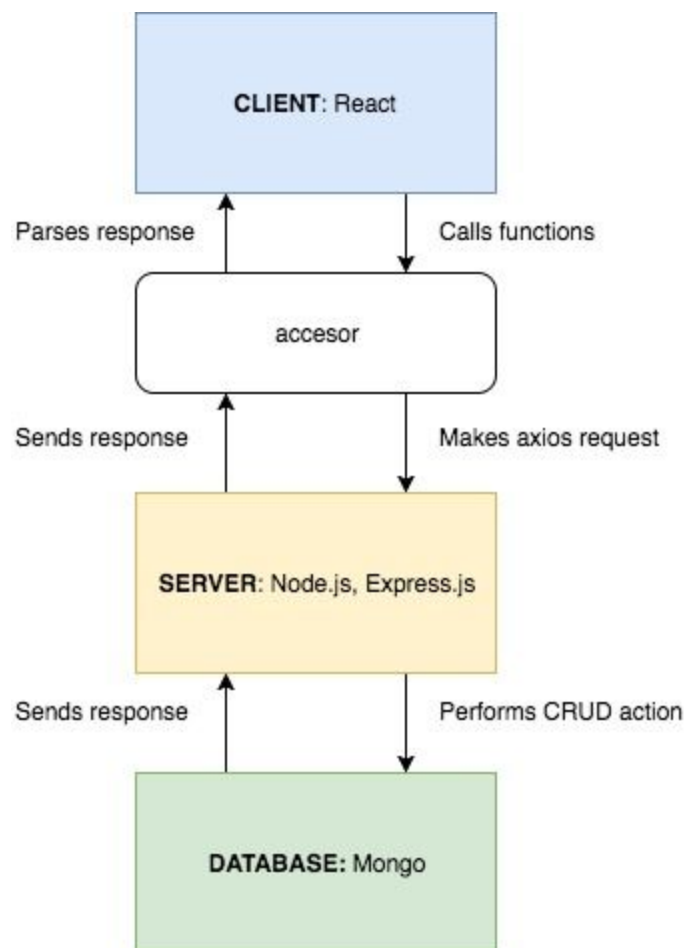


Figure 1. High level overview

### 2. High level Design

#### 2.1 Client:

The client side of the application is written in React and Material UI Beta. It is divided into different components: admin, cart, dashboard, ingredients, inventory, login, main, orders, register, report, storage, and vendors. Each folder corresponds to new page as shown on the left navigation menu, with the exception for the main folder. The main folder contains the application bar and the navigation menu. The view for user and admin is controlled by using conditional rendering. Distinct components that encapsulate the behaviour of user and admin are created and rendered based on account that is logged in.

## **2.2 Accessor:**

Since applications should be modular based, the client side of the application should only handle interactions with users and also displaying information. An accessor is a layer that contains actions and interfaces between front end and the back end. The client side interacts with the server actions through the interface layer. The interface layer then connects to the actions in the server side. This enforces abstraction, allowing the front end component to access only its specific interface layer connecting to the server actions.

## **2.3 Backend:**

The backend is constructed with express.js, node.js, and MongoDB as the database. All the files related to the backend are contained under the “server” folder. It is mainly divided into three parts: models, controllers, and routes. Models contain the database schemas that define the properties of each collection needed in the program (Vendors, Users, etc). Controllers contains files that interacts with the database. Routes define the routes and the API endpoints for the frontend to make an API request and interact with the database.

Specifically, the controller section is consisted of four layers: action layer, modification layer, validation layer, and post process layer. When receiving a request, the modification layer runs first. To make the frontend developers' work easier, only columns (attributes) necessary for the frontend UI are sent back to the database when making CREATE or UPDATE requests. The modification layer then automatically adds all the missing attributes to meet the database schema's requirements (totalPrice for Orders). The validation layer checks if the requirements on the data are met (storage must be larger than current inventory quantity). If not, a customized error will be presented to frontend. The action layer does the real job - CRUD commands and manipulate the database. After the action is done, the post processor modifies other schemas if needed (increase or decrease inventory, etc.). After all the steps, a response is sent to the frontend and the UI will show error or alert if necessary.

## **2.4 Technologies Used:**

- MERN Stack
- Material UI (Front end)
- AXIOS
- Mongoose

## **3. Configuration**

Software needed: Node.js, MongoDB, Git, and npm

Environment: Unix based system (Linux, Mac, etc)

After cloning/downloading the file from the Github repository, please follow the following steps.

- Change directory (cd) into the cloned/downloaded folder
- run "npm install"
- run "npm start" to run the front-end
- run "node server.js" to run the back-end. Note: Make sure Mongodb is running
- run "npm run start-dev" to run both ends at the same time. The front end will be listening on port 3000, and the back-end on port 1337

#### 4. Database Schema

The schema for the tables in mongodb is given below.

1. Users {  
    String username,  
    String password,  
    String email,  
    Boolean isAdmin,  
    Boolean loggedIn,  
}
2. Vendor {  
    String name, //unique name  
    String contact,  
    String code, //unique case-insensitive alphanumeric freight carrier code  
}
3. Ingredients {  
    String name, //unique  
    Enum packageName, //sack, pail, drum, supersack, truckload, railcar  
    Enum temperatureZone, //freezer, refrigerator, warehouse  
    Vendors {  
        Vendor\_ID vendor  
        Number price  
    }  
}
4. Storage {  
    Enum temperatureZone,  
    Number capacity  
}

```
5. Order {  
    User_ID userId  
    Ingredient_ID ingredientId  
    Vendor_ID vendorId  
    Number packageNum  
}
```

```
6. Inventory {  
    User_ID userId  
    Ingredient_ID ingredientId  
    String ingredientName  
    Enum temperatureZone,  
    Number quantity  
    Enum packageName  
}
```

```
7. Cart {  
    User_ID userId  
    Ingredient_ID ingredientId  
    Number quantity  
}
```