Foodiee

food ordering web application

Zacharias Christos Argyropoulos

WEB APPLICATION COURSE
SAE ATHENS
2021-2022

CONTENTS

0. Introduction

In this Web Application course, a food ordering delivery app has been developed using React js for the Frontend and Node js, Express js and MongoDB for the Backend (MERN stack). The main idea is that the user is able to browse through various restaurants, filter those stores based on different cuisines and order food from multiple stores in the same order. This manual/documentation will go into detail about some key parts of the development process and it should also provide valuable information on how to start, use, build and deploy the project. Additionally, you will also find the documentation of the RESTful API of this project (swagger) as well as all the environmental variables and MongoDB cluster credentials needed to run and deploy this project.

1. GitHub repo/ other links

This project is tracked by version control (git). Everytime a new feature, fix or refactor in the codebase is needed, we checkout a new branch from the master branch. After commiting the changes in this branch, a new pull request is made to the master branch. After successfully merging, the PR is closed. A cli tool has been used (https://github.com/commitizen/cz-cli) for the commit messages to have some layout format in each commit message description. The monorepo codebase can be found in the GitHub repository below.

GitHub repository:
https://github.com/HRSArgyropoulos/Foodiee-Ordering-Web-App

The live production environment has been deployed to Heroku and the urls are also listed below.

RESTful API: https://foodiee-service.herokuapp.com/
Client Application: https://foodiee-client.herokuapp.com/

2. Running the app

- Git **clone** the project
- Make sure you are checked out to the master branch and that you have yarn installed in your system.
- Run **yarn** in the root of the project to install all dependencies.
- Duplicate the example.env file located in each workspace (client & server), rename the new file to **.env** in each folder and replace the stars with the appropriate environmental variable values (see section 8 for more details about this step) .
- Run **yarn start-client** at the root of the project to start the client in development mode.
- Run **yarn start-server** at the root of the project to start the server in development mode.

3. Views

Home

Here the user has a first look at the interface, where he can enter his own address to initiate the ordering process. After selecting his address from the dropdown list or typing his own, he is redirected to the stores list page. Alternatively, the user has the option to select a store from a featured stores list and redirect to this specific store to start adding products to his overall cart.

Stores List

In this view the user can browse all stores or stores that deliver to his location, depending if the user has entered an address or not in the previous step respectively. Additionally, the user has the option to filter the fetched store results based on some filters/options. For the purpose of this assignment the filtering is based on different categories/cuisines but this feature can be expanded accordingly. After clicking a store card, the user will be redirected to the store of his choice.

Store

This page contains information about the store and its catalog. The products listed in the catalog are grouped and categorized. The user can add products to the cart, increase and decrease the amount of each product in the cart as well as remove a product from it. There is an extra condition while selecting a product quantity, that it cannot exceed the maxQuantity stated in the product document in the database. In the same page, the user can view the state of this store's cart as well as the overall cart, which consists of products added from other stores. When at least one product is added to the cart a checkout option is provided and after the button is clicked the user gets redirected to the checkout view.

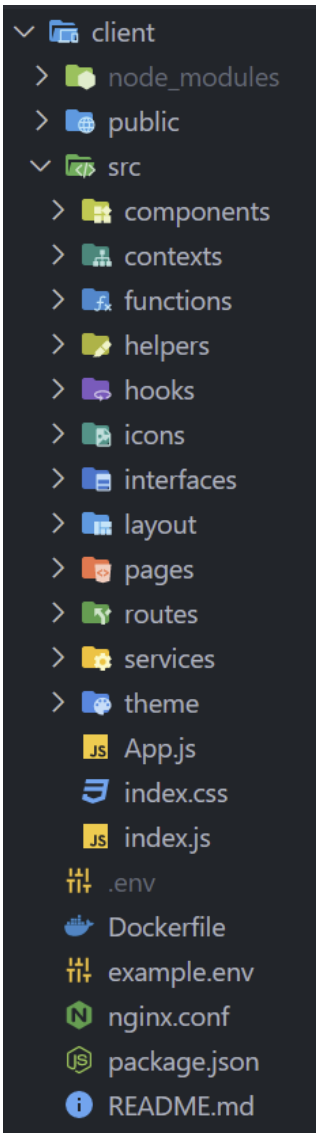
Checkout

Here the user can overview the products of all carts (grouped by store) for the final time before sending the order and finishing the ordering process. The quantity of a product in the cart can also be updated and a product can be removed from the cart. At the same time the total price is updated accordingly.

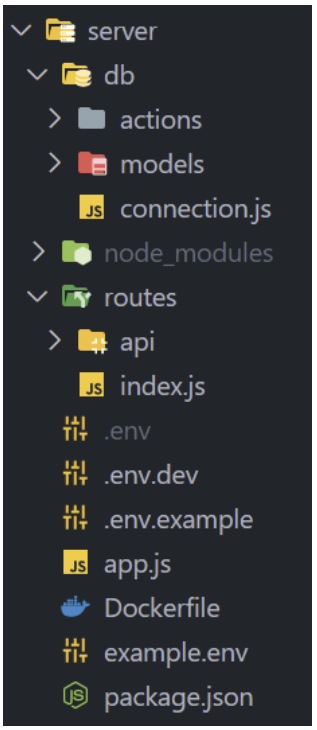
4. Project architecture/ Folder structure

The project follows the monorepo architecture by having two separate workspaces (client and server) and using the same repository. The folder structure of each workspace is shown below.

Client workspace:

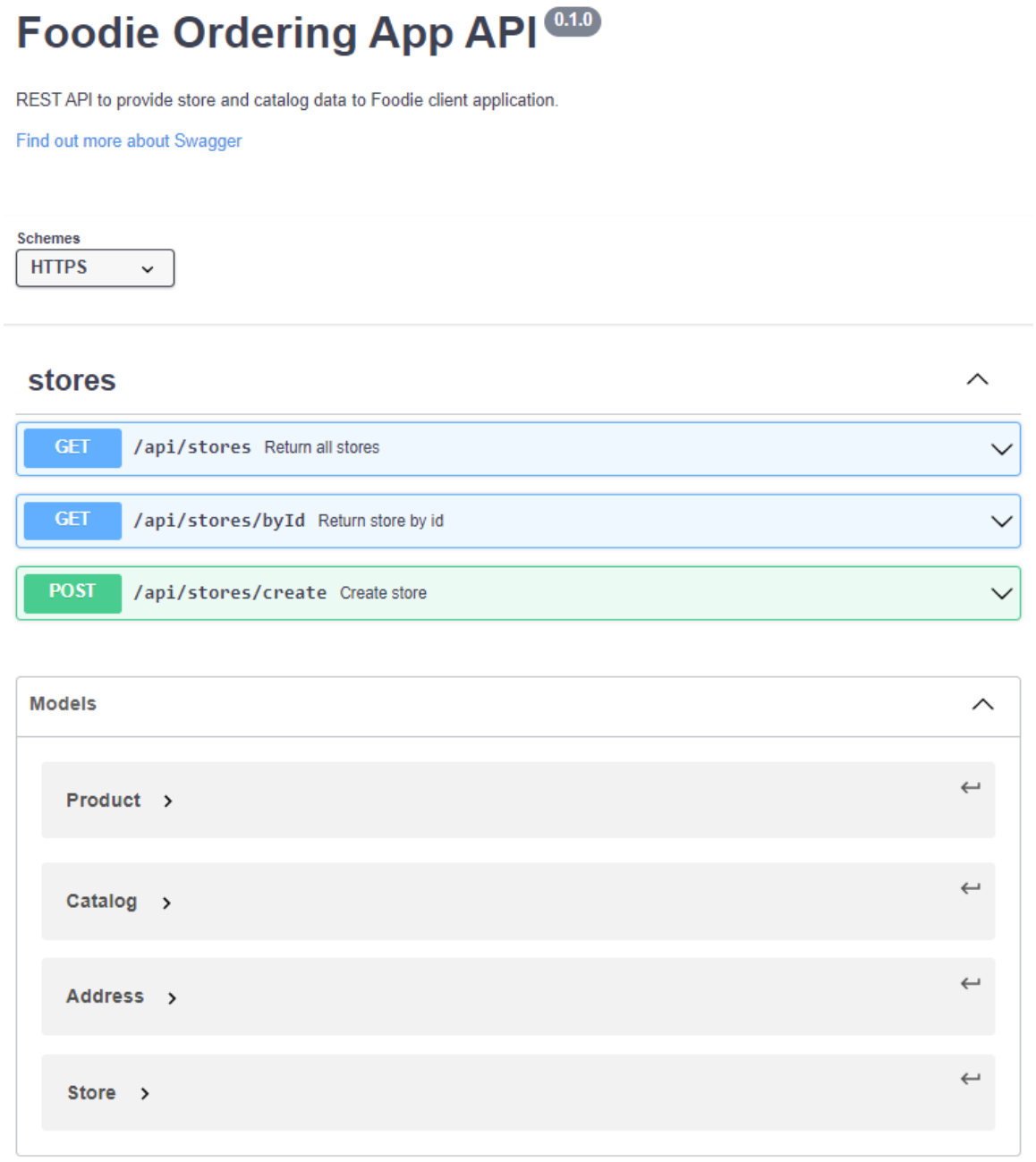


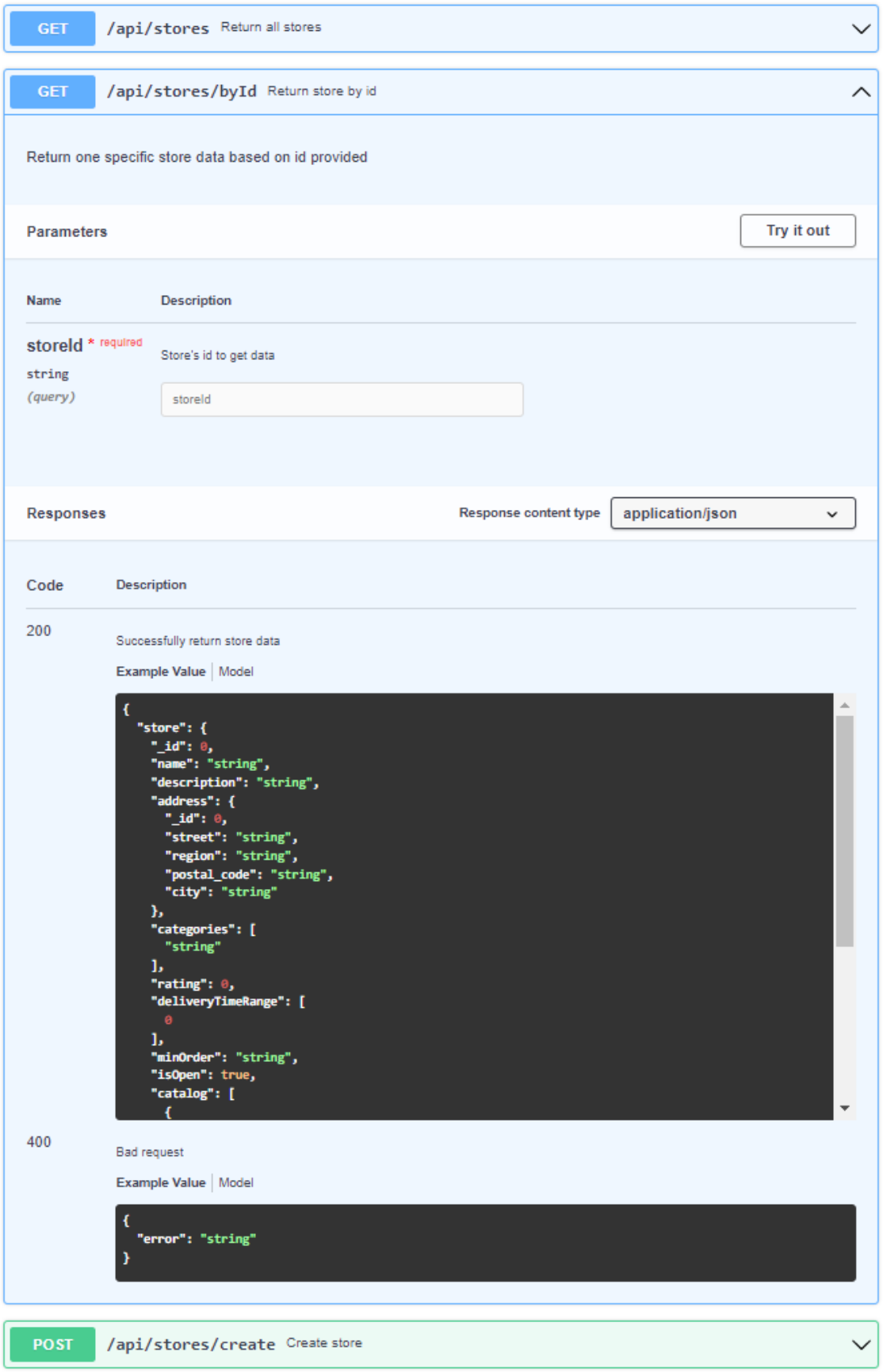
Server workspace:



5. REST API/ Swagger

To provide some store data to the client application, a REST API has been developed. This microservice uses node js and express js at its core and a MongoDB cluster to store all the data. Below you can find a quick overview of the RESTful API documentation as well as a link to the full swagger yaml file. This swagger file provides various information about endpoints, request parameters, body schemas, response schemas and returned status codes.





Swagger documentation:
https://app.swaggerhub.com/apis-docs/HRSArgyropoulos/foodie-ordering_app_api/0.1.0

6. Client application

The client application is written in React js and uses the MUI library for most of its components. The default MUI **theme** has been modified by overwriting certain properties in the MUI palette object.

To share the data through various components and avoid prop drilling in certain cases, the application makes use of the **context** functionality provided by react.

In **StoreDataContext** we fetch the stores and store the data in the state for the children components to access the data at any time and also to avoid repeated requests to the server.

In **StoresCartContext** we store products that have been added to the cart by store. We also provide the functionality to update the quantity of a product that has been already added to the cart and also the functionality to remove a product from the cart. As mentioned in the summary as well, the cart consists of many carts from different stores. You can find a better representation of the carts context structure below:

```
// Carts Schema
const carts = [
   {
        storeId,
    ...other store data,
    products: [{ productId, ...other product data, quantity }],
   }, ...other stores
];
```

In **UserContext** we store user information that has been filled from the user in various input fields throughout the whole ordering process and is consumed in other components like the checkout page.

To consume all these contexts and provide the components with the appropriate state we implement some hooks in the hooks folder of the project. Along with these **hooks** we provide some additional ones like **useWindowTitle** to change the title of the window (browser tab) and **useScrollToTop** to scroll to the top of the page when we navigate to a new

page. Both those hooks run before painting the DOM with React's useLayoutEffect hook.

Math calculations, like rounding numbers of calculations between float numbers, can be found in the **functions** folder.

In the **helpers** folder we can find other helping functions like sums where we calculate the total product, store and all stores price in the cart.

In the **services** folder we make all the requests to the api to provide our application with store data.

In the **routes** folder and more specifically in routes.js file we list the routing of our application and map each route with the appropriate component. Here not to fetch all components in our application together and split the final js files into separate junks, we lazy load each component with Suspense. A fallback spinner is also implemented here to show some feedback to the user while we fetch and load a specific component. In the path.js file we export the paths of each route to use throughout our application (like link buttons and sidebars).

7. Deployment

Each directory (client and server) has its own Dockerfile and a Docker container image is built based on it.

Server

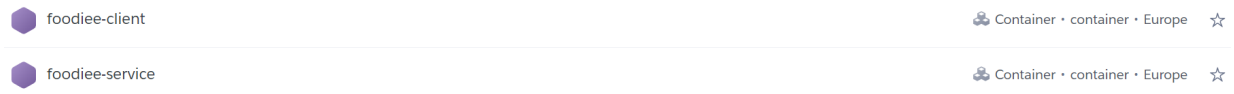
For the server image a port and a database uri env variable value is needed. The credentials and cluster uri can be found in section 8. Then, we start the node server as is.

Client

For the client image a port, a server api uri and a cloudinary cloud name is needed. After building the project (bundling and minifying) we need to serve the app using a NGINX web server. Because we use react-router in our project and change the url without actually accessing different index pages in subdirectories, we need to make some configurations to the NGINX server to fallback and redirect to the main index.html page.

Note: The port environment variable is mainly needed for the deployment as heroku automatically assigns a random port to expose, overwriting any port value that has been already set.

After building the images, both were pushed to Heroku's Container Registry and released to Heroku to finally deploy the application and service.



8. Environmental variables/ MongoDB credentials

To run the client and server application we need to provide values to the environmental variables. These values are also needed and used by the Docker images when running the images in containers. To test the app and work on the project, without affecting components used by the live environment, we have two separate environments (development and production).

The values of the production env variables are very sensitive and must be kept secret. For the purpose of this educational project and assignment grading, these values are hereby provided;

```
Client (development):
REACT_APP_SERVER_API='http://localhost:3000/api/'
REACT_APP_CLOUDINARY_CLOUD_NAME='doy1tqcqe'

Client (production):
REACT_APP_SERVER_API='https://foodiee-service.herokuapp.com/api/'
REACT_APP_CLOUDINARY_CLOUD_NAME='doy1tqcqe'

Server (development):
PORT=3000
DB_URI='mongodb://localhost:27017'

Server (production):
PORT=3000
DB_URI='mongodb+srv://foodiee-production:PLZtbLc7YDxyySvc@cluster0.s9blh.mongodb.net/foodieeDB?retryWrites=true&w=majority'
```

MongoDB Cluster URI/Credentials

For the production environment a new user has been created only with Read privileges to the db. That means you will not be able to add new stores (like the swagger suggests in api/stores/create endpoint),delete or update any document in the database for security reasons and such errors will occur.

```
{
   "error": "user is not allowed to do action [insert] on [foodieeDB.stores]"
}
```