**International Institute of Information Technology, Bangalore   
(IIIT Bangalore)**

FoodTinder - Platform to find food ordering partners!



Ajay Chhajed Vismaya Solanki   
(IMT2019006) (IMT2019095)

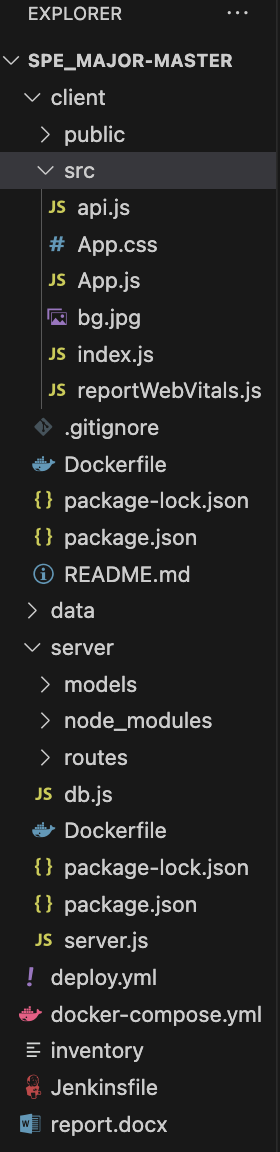
1. **Web Application**
   1. **Introduction**It happens many times that one wants to order something from zomato or swiggy but ordering just a single item might be very costly due to the small amount of order and huge delivery charge plus coupons won’t be available for small order sizes.

We will build a website in which the students can find other users who also want to order something and connect with them. This would help users to save money and find a friend to order with. So basically, it is like tinder for food, where similar food interests will match and they can order together. It’ll be a one time thing :).

* 1. **Features**One can instantly go on a website and express their interest in ordering what kind of food they would like to have and what time they want. We can have some predetermined categories of food and users can select from them. Our website will find someone who shares a similar preference at that time slot and connect them to each other.

A user can select if they want to connect with another user.

Once their profiles match they can connect with each other and mutually decide and order food they like.

* 1. **Detailed view of Implementatio**

**client/Dockerfile:**

1. **FROM node:20**: Specifies the base image to use for the Docker image. In this case, it uses the official Node.js 20.x image as the base.
2. **WORKDIR /usr/src/app**: Sets the working directory to **/usr/src/app**, so any subsequent commands will be executed relative to this directory.
3. **COPY package\*.json ./**: Copies the **package.json** and **package-lock.json** (if present) files from the host to the container's working directory. This step allows Docker to cache the dependencies installation step separately, improving build times.
4. **RUN npm install --silent**: Installs the dependencies defined in the **package.json** file. The **--silent** flag is used to suppress the verbose output during installation.
5. **COPY . .**: Copies the entire application source code from the host to the container's working directory. This step includes all the client-side files, such as JavaScript files, components, styles, and other assets.
6. **EXPOSE 3000**: Informs Docker that the container will listen on port 3000, allowing external connections to this port. This assumes that your React application is configured to run on port 3000.
7. **CMD ["npm", "start"]**: Specifies the command to run when the container starts. In this case, it runs the **npm start** command, which is a common script defined in the **package.json** file for starting the React development server.

**server/Dockerfile:**

1. **FROM node:20**: Specifies the base image to use for the Docker image. In this case, it uses the official Node.js 20.x image as the base.
2. **RUN mkdir -p /usr/src/app**: Creates a directory **/usr/src/app** inside the container where the application code will be copied.
3. **WORKDIR /usr/src/app**: Sets the working directory to **/usr/src/app**, so any subsequent commands will be executed relative to this directory.
4. **COPY package\*.json ./**: Copies the **package.json** and **package-lock.json** (if present) files from the host to the container's working directory. This step allows Docker to cache the dependencies installation step separately, improving build times.
5. **RUN npm install --silent**: Installs the dependencies defined in the **package.json** file. The **--silent** flag is used to suppress the verbose output during installation.
6. **COPY . .**: Copies the entire application source code from the host to the container's working directory. This step includes all the backend server files, such as JavaScript files, routes, and other dependencies.
7. **EXPOSE 5005**: Informs Docker that the container will listen on port 5005, allowing external connections to this port.
8. **CMD ["node", "server.js"]**: Specifies the command to run when the container starts. In this case, it runs the **server.js** file using the Node.js runtime. You may need to adjust the file name based on your actual server file name.

**Jenkinsfile:** This Jenkinsfile defines a declarative pipeline with multiple stages for building and deploying a project. The pipeline begins by specifying that any available agent can be used. It then sets up the necessary tools, specifically Node.js version 20, using the tools directive. The stages section contains several stages: "Pull Git" for cloning the project repository, "Build" for installing dependencies in the server and party-owl directories, "Test" for running tests in the party-owl directory, "Docker Build" for building a Docker image using Docker Compose, and "Docker Deploy" for deploying the application by running the Docker containers. Each stage includes specific steps defined using the steps directive, with some steps executed within specific directories using the dir directive. The build and deployment steps are executed using the bat command, which allows running shell commands on Windows.

**Persistent Database**

1. **Volume Mapping: The - /Users/vismayasolanki/Desktop/foodapp/data:/data/db line specifies a volume mapping between the host machine (your local machine) and the container. It tells Docker to map the /Users/vismayasolanki/Desktop/foodapp/data directory on your host machine to the /data/db directory inside the MongoDB container.**
2. **Persistent Storage: By mapping the host directory to the container directory, any data written to the /data/db directory inside the MongoDB container will be stored on your host machine, specifically in the /Users/vismayasolanki/Desktop/foodapp/data directory. This ensures that the data persists even if the MongoDB container is stopped or removed.**
3. **Database Files: MongoDB stores its database files in the /data/db directory by default. With the volume mapping in place, any changes made to the MongoDB data inside the container will be reflected in the mapped directory on your host machine. This allows the data to be preserved and accessible even if the container is restarted, upgraded, or replaced.**

**Docker-Compose**

**Docker Compose is a tool that allows you to define and manage multi-container Docker applications. It provides a way to define the services, networks, and volumes required for your application using a YAML file called docker-compose.yml. With Docker Compose, you can specify the configuration of your application's services, their dependencies, and their relationships in a declarative and easy-to-read format.**

**Some Advantages of Docker-Compose:**

1. **Multi-container application management**
2. **Service definitions**
3. **Easy environment setup**
4. **Dependency management**
5. **Networking and volumes**

**Docker-compose.yml**

* **server: This service represents the backend server. It is based on the image vismayasolanki/backend. The container is named backend and will be restarted automatically if it stops. It exposes port 5005 on the host machine and maps it to port 5005 in the container. It is also linked to the mongo service.**
* **client: This service represents the frontend client. It is based on the image vismayasolanki/frontend. The container is named frontend and will be restarted automatically if it stops. It exposes port 3000 on the host machine and maps it to port 3000 in the container.**
* **mongo: This service represents the MongoDB database. It uses the official mongo image from Docker Hub. The container is named mongo. It creates a volume that maps the local directory /Users/vismayasolanki/Desktop/foodapp/data to the /data/db directory inside the container. This volume is used to persist the database data across container restarts. It also exposes port 27017 on the host machine and maps it to port 27017 in the container.**

**Tech Stack:**

**MERN(MongoDB, ExpressJS, ReactJS, NodeJS )**

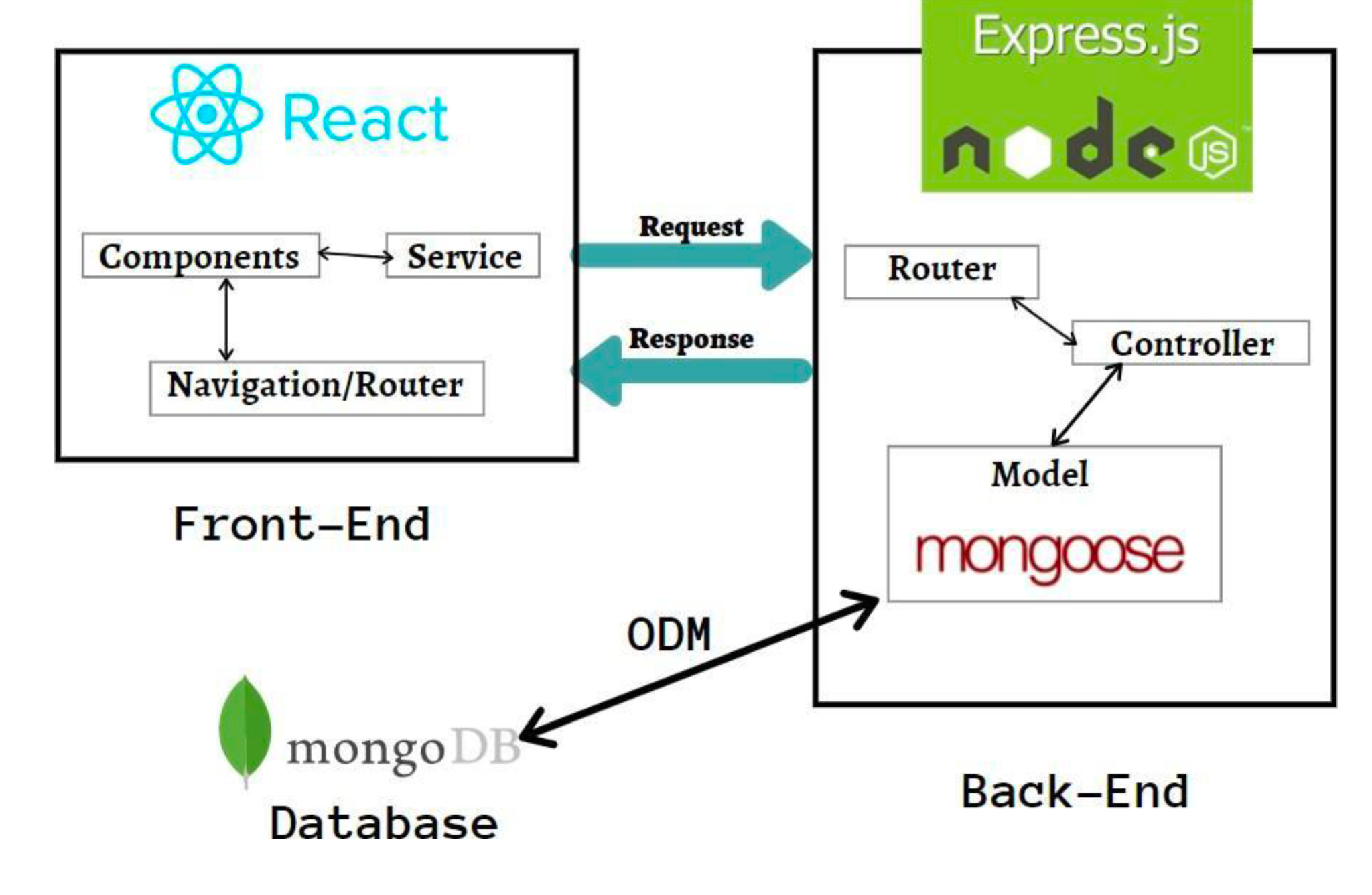
Frontend - React, HTML, CSS

Backend - Node, Express

Database - MongoDB

VCS - git / github

Containerization - Docker-compose



**Future Scope:**

* We can make this a fully functional app by adding a bill splitter and payment portal so that if multiple users connect they app will, according to each’s order, split the total bill and ask all the users to make payment to the app and then app will automatically place the order from zomato or swiggy whichever is cheaper at that time.
* Users can also chat in-app if they don’t want to share their contact info.
* One can make friends through this app, like facebook friends whom they can directly contact.
* The idea can be expanded from college to same society residents and nearby residents etc.

**References:**<https://www.blazemeter.com/blog/how-to-integrate-your-github-repository-to-your-jenkins-project><https://docs.ansible.com/ansible-core/2.14/reference_appendices/interpreter_discovery.html>

<https://dzone.com/articles/dockerizing-a-mern-stack-web-application>