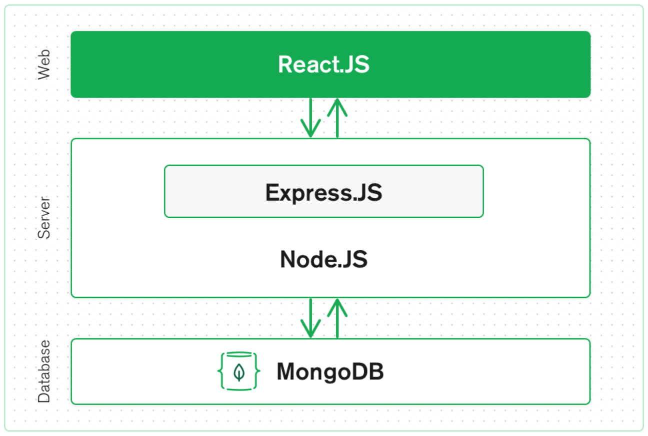
Full stack webapp development (MERN Stack)

Simon Shrestha

2021-04-20

1 Background

A full stack web app for a restaurant offering food delivery service. For the development of the project, I am going to use Mongodb as the database, Express framework for back end and React framework for front end programming in Nodejs. This is commonly referred as MERN stack.



2 Requirements

After some meetings with the client the essential requirements for the systems were discussed.

Functionality requirements:

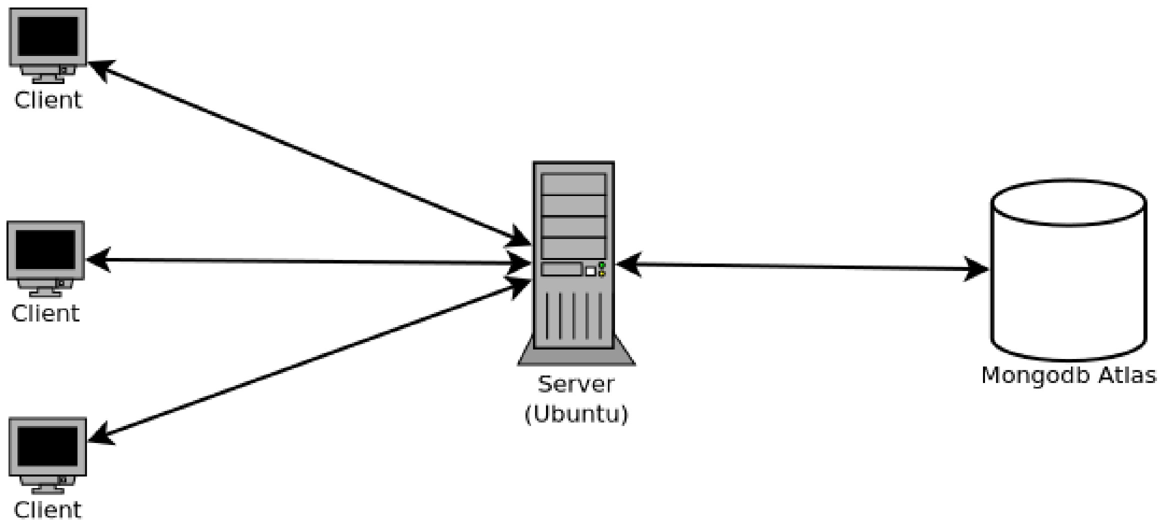
- As customer:
  - view all available products
  - add or remove products in the cart
  - create an order
  - view their order status
- As admin:
  - create account
  - create, modify and delete products
  - view, modify and delete orders
  - manage users
- As merchant:
  - modify orders

Security requirements:

- safe end to end encryption on client and server side. (HTTPS)
- safe encrypted payment transactions
- save passwords and other valuable data in encrypted form
- input validation

3 Design and Architecture

The architecture I used to build a web app is server-client architecture. It allows to implement the front-end running in clients machine and backend running on a server. For database I used the cloud database provided by mongodb as Atlas.



The app is designed for customers and staff members which have their own requirements.

Customers: They don't require any account to view products or make an order and don't have any privileges to modify products or orders after ordered. However they can modify their items in the cart prior making an order. In addition they can also view the status of their order

Staff members: They require an account to perform various task to fulfill the order request. There are two types of account for employees. Account of role: merchant have limited privilege. They can modify the status of an order such as changing "cooking" status to "on delivery". This is designed for delivery drivers and kitchen staffs. Account of role: admin have full privilege in modifying products, users and orders. This is designed for administrative purposes such and adding or removing products.

The database model designed for the system has three schemas. Since this is noSql, each object is referenced by their unique id and therefore every object has a data type of id.

Order: To store order and then track them by their status. It stores name, address, phone number, order date, items, price, status and token of an order.

Product: The products available are stored in this schema. It stores name, price, image, info and type of the product. The image is actually stored on server while only the location is stored in image data.

User: The employees are provide with an account. This schema will store username, email, role and password of an employee. The password is stored in encrypted form.

4 Implementation and Development

There are two main components of the system: frontend and backend. Both of them are built using javascript. For the frontend or client side, I used React framework. React provides a development environment while building and then it creates optimized build version to deploy in production environment. It also makes the web-app dynamic and does not require reloading on every event. I have also imported some third-party libraries. Customers details such as name , phone number and their cart status is stored in local storage.

material-ui: for various icons and components.

axios: for making requests to the server.

react-alert: for alert dialog box components.

react-copy-to-clipboard: for storing order id in the memory.

khalti-checkout-web: for making payment with service provided by khalti.

For the backend or server side, I used Express framework in NodeJS. I have imported some third-party libraries as well.

bcrypt: for hashing passwords and encrypt them in order to store.

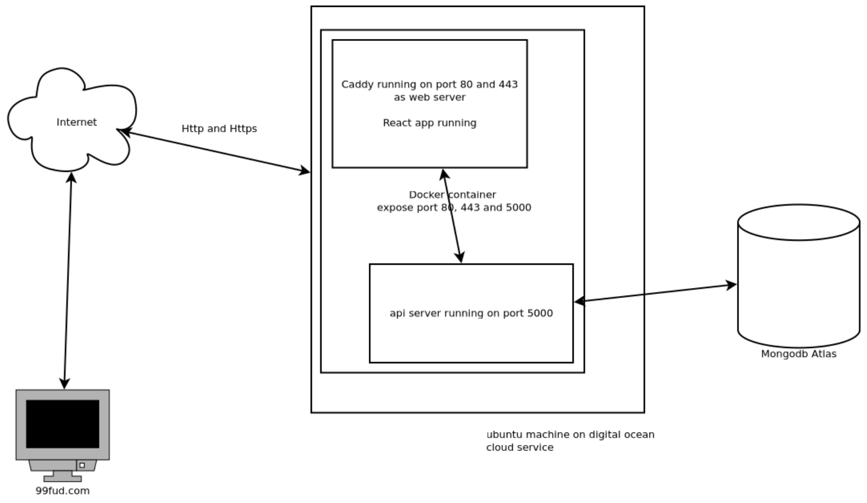
mongoose: for database operations in mongodb.

multer: for uploading images.

5 Hosting

I decided to containerize my app using docker so that I can run frontend and backend in the same machine. I used a caddy as web-server to host my client-side. Both client side and server side are run in separate containers and are connected inside a docker container. The client side container is a caddy webserver hosting react app. I chose caddy since it provides automatic HTTPS and also helps to reverse proxy to server for /api.

The machine serving the whole system is a ubuntu machine on digitalocean cloud. For HTTPS end-to-end encryption, the app is hosted behind cloudfare which also provides some DDOS protection.


Internet
Http and Https
Caddy running on port 80 and 443
as web server
React app running
Docker container
expose port 80, 443 and 5000
api server running on port 5000
Mongodb Atlas
ubuntu machine on digital ocean
cloud service
99fud.com

6 Conclusion

After hosting the app online I was glad that everything worked out and functioned as designed. All of the things mentioned are all free of cost. The database and cloud computing are used for free as the respective service provider has a free tier.

The codes are available in the following link. https://github.com/Simant0/MERNapp.git