Sneaker Checkout Calculator

Assignment Members: Sarah Eddeb & Madhurima Dutta

For this assignment, we created a checkout calculator for a sneaker shop that would sum up the subtotal, tax, and discount of the sneakers added to the users cart. The main features include adding and removing shoes, summation of subtotal, calculation of a 13% tax rate, calculation of discount based on subtotal, and a calculation for the final total. Discounts are applied based on the subtotal of the shoes added to the cart. Purchases over $100 receive 25%, over $150 receive 30%, and over $300 receive 60%.

Decisions

Our main aim for this assignment is to create a simple web application to help us get accustomed to the softwares and frameworks needed to complete our main project. As a result, many of our decisions were heavily influenced by the tech stack needed for that project. That said, we also explored other frameworks to see if there were more optimal solutions that would aid us with the completion of this assignment along with our final project.

Final Decisions

UI and Frontend - React and MUI. The two of these frameworks together aided us with creating a seamless looking user interface while keeping the code clean and concise.

Backend - Next.JS, Node.JS and Express.JS. All the frameworks used were to create efficient backend code and useful REST-APIs to connect the backend and the frontend.

Databases - Mongoose.JS is a simpler framework built upon Node.js that uses MongoDB to store our data.

UI/Frontend

We decided on using React as our main framework for frontend as our project with the partner had many aspects already coded using it. We then shifted our focus on what to use to style our frontend components. Our 3 options included CSS, MUI (material-UI), and Styled Components

CSS - CSS is the most basic of the three. That said, Styled Components work in a very similar fashion to the conventional CSS. We were deterred from using this option due to the time consuming nature of this choice. As we decided to build a web app, we needed to ensure we had good responsiveness. Although this is not necessarily difficult to do with CSS, we knew that it would be more time consuming implementing responsive components than using MUI that has built in responsive features.

MUI - MUI has an extensive library with preset components that are just a copy-and-paste away from implementing on the frontend. Despite the ready made nature, it is relatively easy to add customized styling similar to CSS and Styled Components. It also includes readily made responsive features for the components. We have experience working with this library and, as a result, decrease issues that might come about due to learning curves. Because of these clear advantages, we decided to use MUI as our styling framework

Styled Components - Based on our research, Styled Components seems to be a more superior option than CSS. It is able to maintain more consistent styling on the frontend as well as creating themes for the entire project, something MUI is also capable of doing. CSS unfortunately does not have theming capabilities. Although these functionalities are great for big projects, we found it to be less suitable for this assignment when compared to MUI.

Logic/Backend

Our tech stack for the backend was based upon our project with our partner as they already have a pre-existing code base written in Node.js and Express.js. So our motive in A2 was to familiarize ourselves with their tech stack. In addition, we used Next.js to connect our backend and our frontend and deploy that application on Heroku.

Before deploying to Heroku, we considered various deployment options provided in the A2 handout: AWS, Google Cloud, Microsoft Azure and Heroku. Since our application is a simple shoe store, it seemed like AWS, Google Cloud and Microsoft Azure had very complicated approaches to deploying their application. As they are a big platform with 100+ resources, it had a steep learning curve on how to piece different resources to get our app set up. They also required my credit card on file to deploy my application which seemed like an inconvenience. On the other hand, I was interested in using docker and kubernetes to deploy the app but Heroku didn’t support them. Since it was beginner friendly where I didn't have to manage and connect different resources to run my app, we finally decided to use heroku. I had multiple guides that helped me deploy this application within an hour.

For our testing infrastructure, we used mocha and chai JS frameworks to test our REST-API endpoints. It is a simple unit test that checks the status of all 4 API calls. We were conflicted between using mocha and chai Vs jest and supertest frameworks: we initially selected jest and supertest since it's more commonly used, but as it was implemented, it would run into Segmentation Error. After looking into the error, it seemed like jest doesn’t perform as expected with MacOS. So, I changed my implementation to mocha and chai.

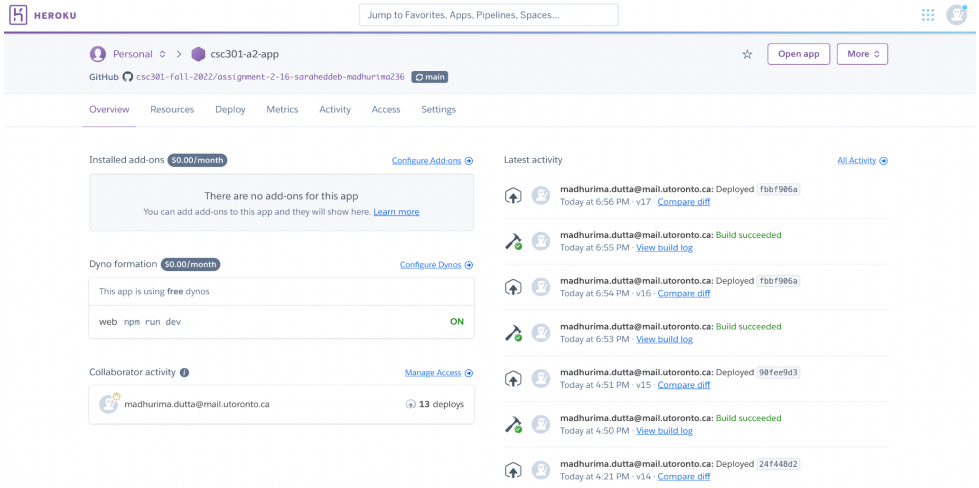
Database

We decided to use Mongoose.js which is a framework available on JS to work with MongoDB. We decided to work on MongoDB to familiarize ourselves with a NoSQL database. Another motivation for us to work with this is our project’s main database is MongoDB. The reason we chose mongoose over mongodb framework on node.js is that mongoose simplifies the data input to a specific schema and hence, helps us keep track of the data being imported into the

database. The mongodb framework has more flexibility in terms of functionality which would be useful for a bigger data collection but since we required a very basic implementation of our database, we chose mongoose.

Instructions

The link to our web application: https://csc301-a2-app.herokuapp.com
This is an image of our deployment overview.



As seen in the image above, our application has been deployed on Heroku using their free services for students. It is connected to our github repository: https://github.com/csc301-fall-2022/assignment-2-16-saraheddeb-madhurima236.git

How to run testing infrastructure

In order to run the testing infrastructure, you can follow the following steps:

1. Clone the git repo using the link provided in instructions
2. Open your CLI
3. Go to the root folder i.e. `checkout-calculator`
4. Run `npm test`
5. Your output will look like this where you can see that all the REST-API endpoints are working as expected.