CS316 Final Project Report

Team Members: Anna Xu, James Leong, Michael Jang, Mina Kim, Vanessa Chen

Introduction

For our CS316 final project, our group chose to recreate Amazon and create a simpler online marketplace e-commerce website. Our design includes the basic features that Amazon has, such as different sellers and a variety of products that can be purchased, as well as basic functionalities for users such as purchasing and leaving reviews. We used the MERN (MongoDB, Express, React, and NodeJS) stack to create this site.

Deliverables

1) **Final code:** https://github.com/JamesLeong98/miniamazon-cs316

2) **Final video:** **https://drive.google.com/file/d/1IodJDnQGTLVxzO0gBzGy3HnjfMP9Jxue/view?usp=sharing**

Database Design (noSQL Schemas)

We changed our database choice from SQL to noSQL (MongoDB).

Items and Review Schemas:

```
//Define review schema
const reviewSchema = new Schema(
    {
        authorId: { type: Schema.Types.ObjectId, ref: 'User', required: true },
        rating: { type: Number, required: true, min: 1, max: 5},
        title: { type: String, required: false },
        comment: { type: String, required: false },
    },
    {
        timestamps: true,
    }
);

//Define item schema
const itemSchema = new Schema({
    itemName: { type: String, required: true },
    category: { type: String, required: true },
    image: { type: String, required: false }, // link to AWS file
    description: { type: String, required: true },
    reviews: [reviewSchema],
    avgRating: { type: Number, required: false, min: 0, max: 5, default: 0},
    lowestPrice: { type: Number, required: false, default: 0},
});
```

SoldBy Schema (captures information of when user sells an item)

```
//Define schema for soldBy
const soldBySchema = new Schema({
    item: { type: Schema.Types.ObjectId, ref: 'Item', required: true },
    seller: { type: Schema.Types.ObjectId, ref: 'User', required: true },
    quantity: { type: Number, required: true },
    price: { type: Number, required: true },
});
```

Transaction Schema:

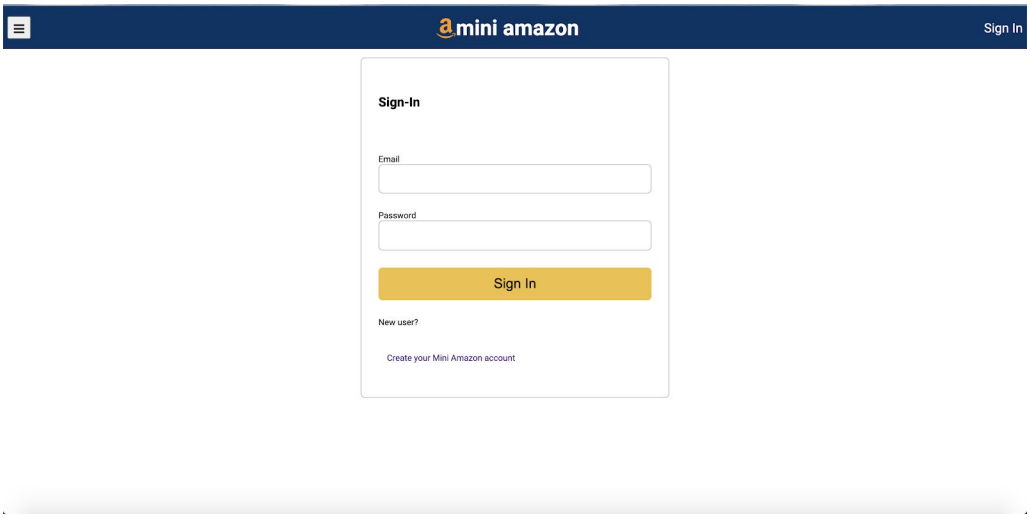
```
//Define transaction schema. A transaction is a list of transaction items made by a buyer at one datetime.
const transactionSchema = new Schema(
    {
        buyerId: { type: Schema.Types.ObjectId, ref: 'User', required: true },
        totalPrice: { type: Number, required: true },
        transactedItems: [{
            item: { type: Schema.Types.ObjectId, ref: 'Item', required: true },
            seller: { type: Schema.Types.ObjectId, ref: 'User', required: true },
            quantity: { type: Number, required: true },
            price: { type: Number, required: true },
        }],
    },
    {
        timestamps: true,
    }
);
```

User Schema:

```
//Define user schema
const userSchema = new Schema({
    fullName: { type: String, required: true },
    username: { type: String, required: true, unique: true },
    email: { type: String, required: true, unique: true, dropDups: true },
    password: { type: String, required: true },
    avatar : { type: String, required: false},
    isAuth: { type: Boolean, required: true, default: false },
    isSeller: { type: Boolean, required: true, default: false },
    isAdmin: { type: Boolean, required: true, default: false },
    balance: { type: Number, required: true, default: 0 }
});
```

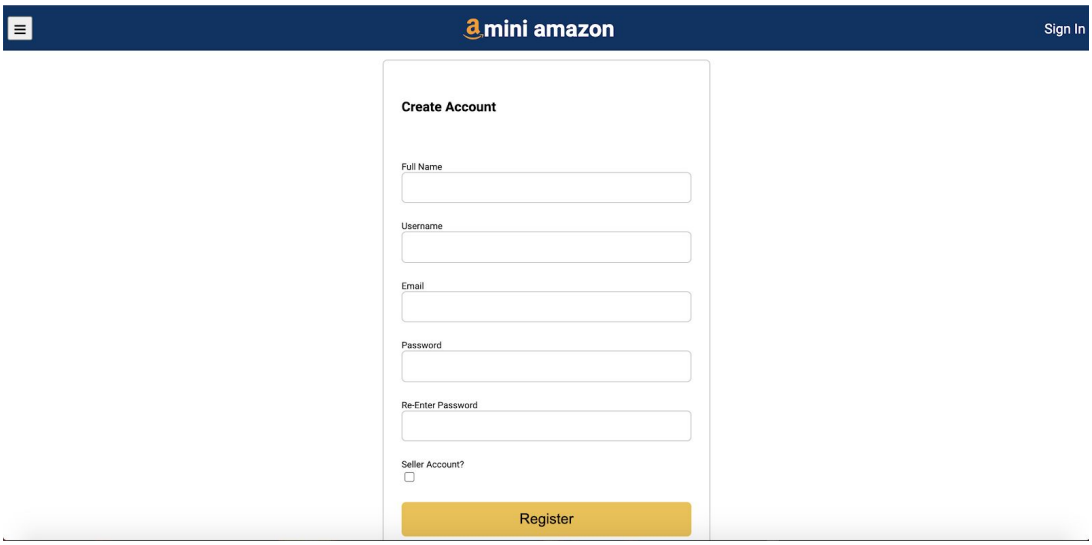
Implementation

Login Page



- The user is prompted to enter their email and password, or create an account if they do not have one currently.

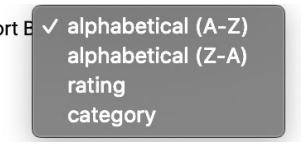
Create Account/Register Page



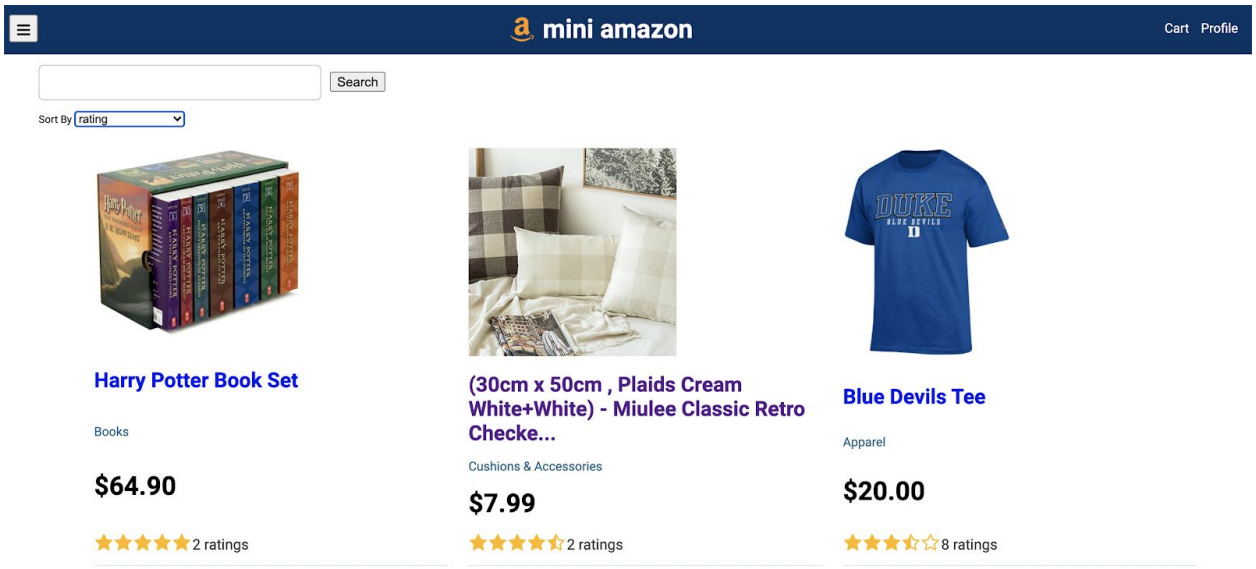
- If they wish to create an account, they must enter their name, username, email, password, and check the box if they want a seller account (otherwise they will have a default user account).

Main Page

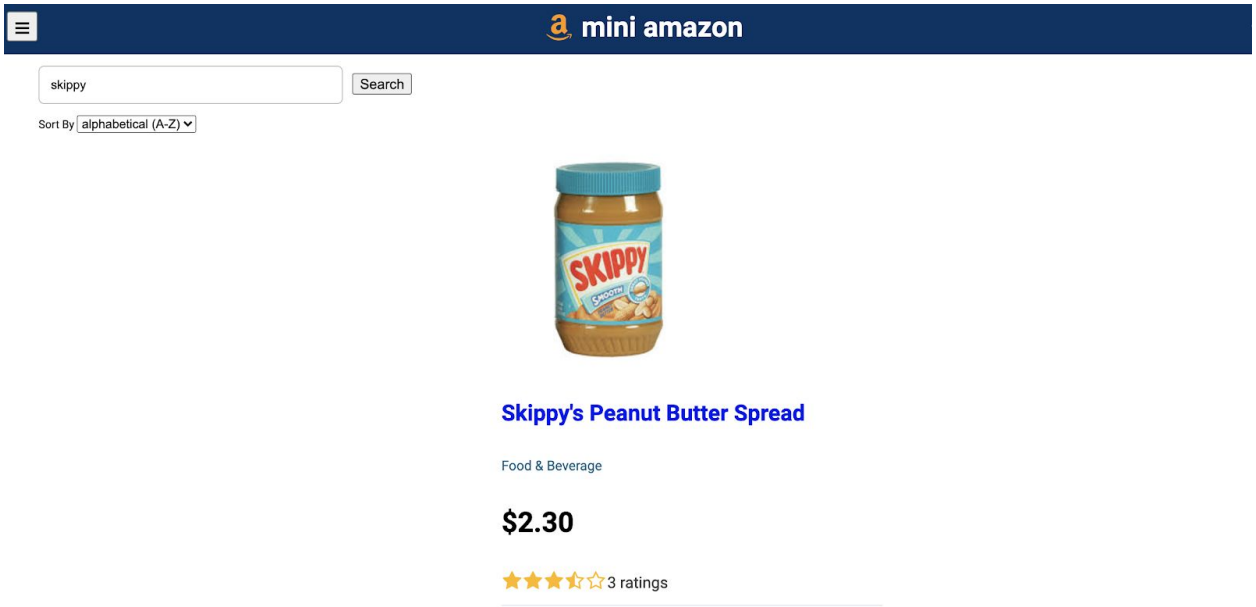
(Main page - filtering options)



(Main page - example of sorting by rating)

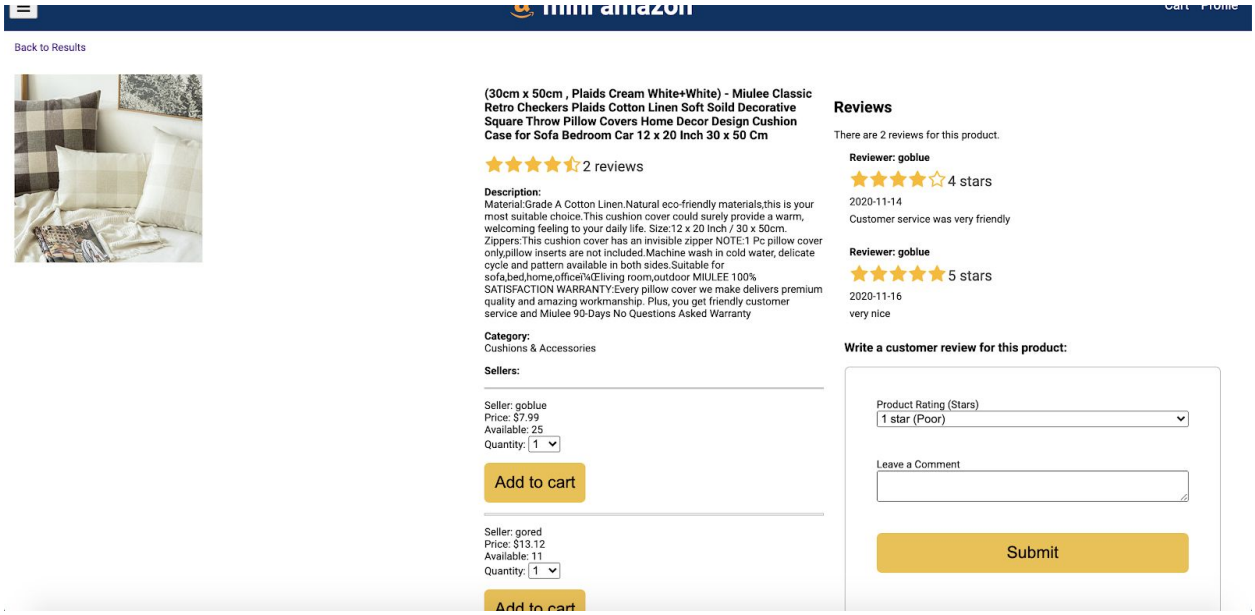


(Main page - example of searching)



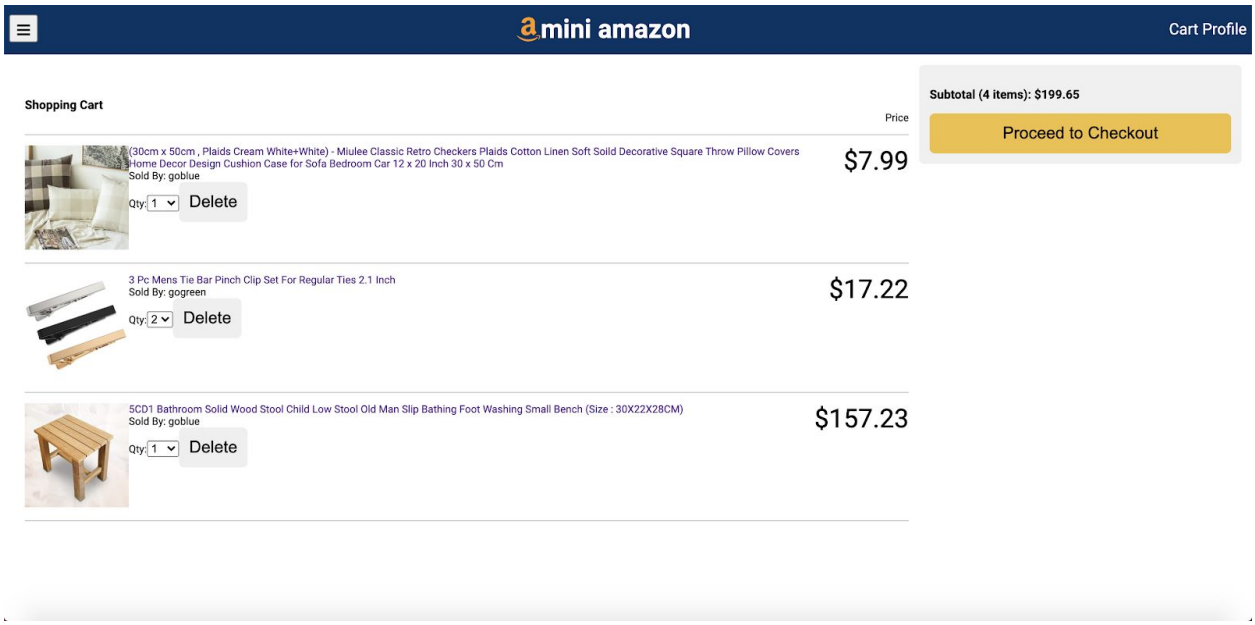
- The user must be logged in to enter the Main page, or the user will be sent back to the login page
- Search options: search by string/value match on keyword, or filter by alphabetical, category, star ratings, etc
- Render search results after search submit
- Profile button that links to user profile page
- Cart button that links to cart page

Item Page



- Category information for the item.
- Item information: Item name, brief item description, image, star rating, reviews.
- Selling information: Show main seller and a list of sellers selling this item (sorted in ascending price order). The list contains seller name, availability, quantity dropdown list, price, and add to cart button. Note that different sellers may have different selling prices for the item as well as different numbers of copies available.

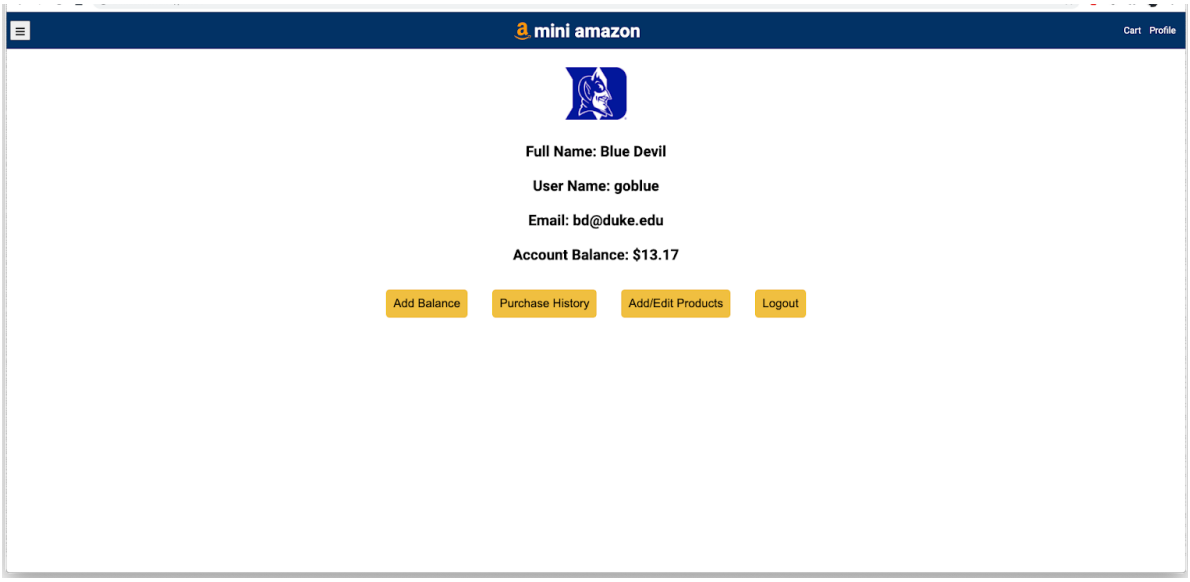
Cart Page



- List of items in the cart: item image, item name, seller name, price, count, and remove button. Item name should be the link to the item page.
- Users can remove items or change quantity.

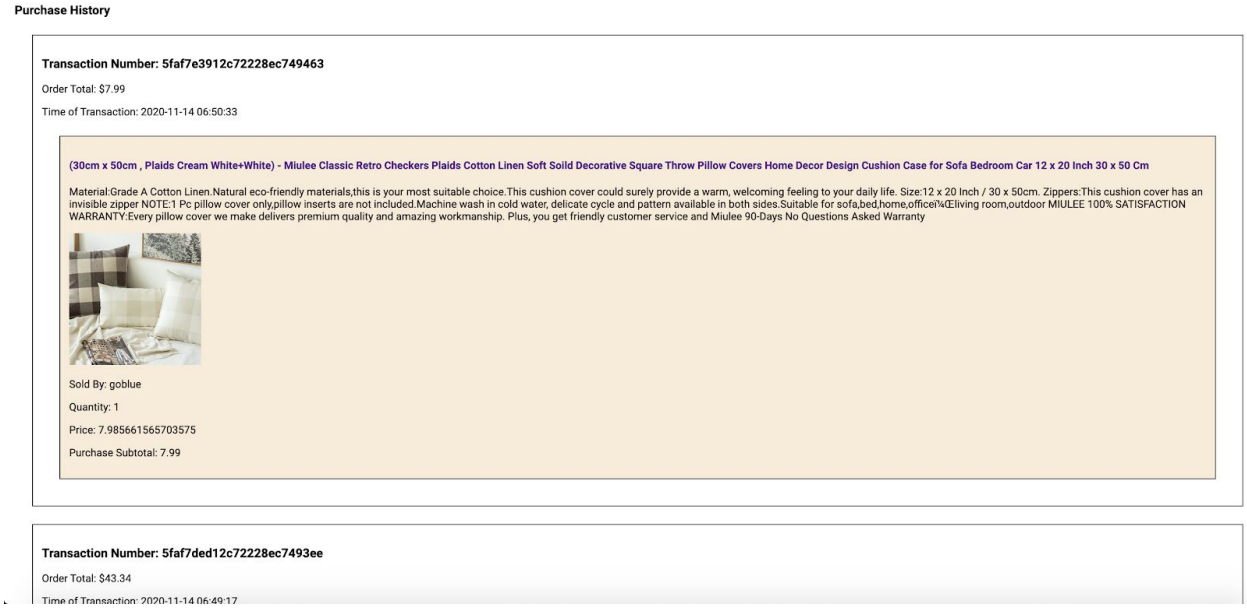
- When a user adds an item to cart, make sure the amount added to cart is <= quantity of item remaining. The item is deducted from the items list once the user makes a successful checkout.
- At time of purchase (clicking ‘buy’) ensure there are sufficient amounts of the item and balance left.
- Check out button: All items in cart will be recorded in transaction history. If the seller has not enough copies to sell, “insufficient quantity” alert shows up. If balance is insufficient, "balance is insufficient" alert pops up.

User profile page:



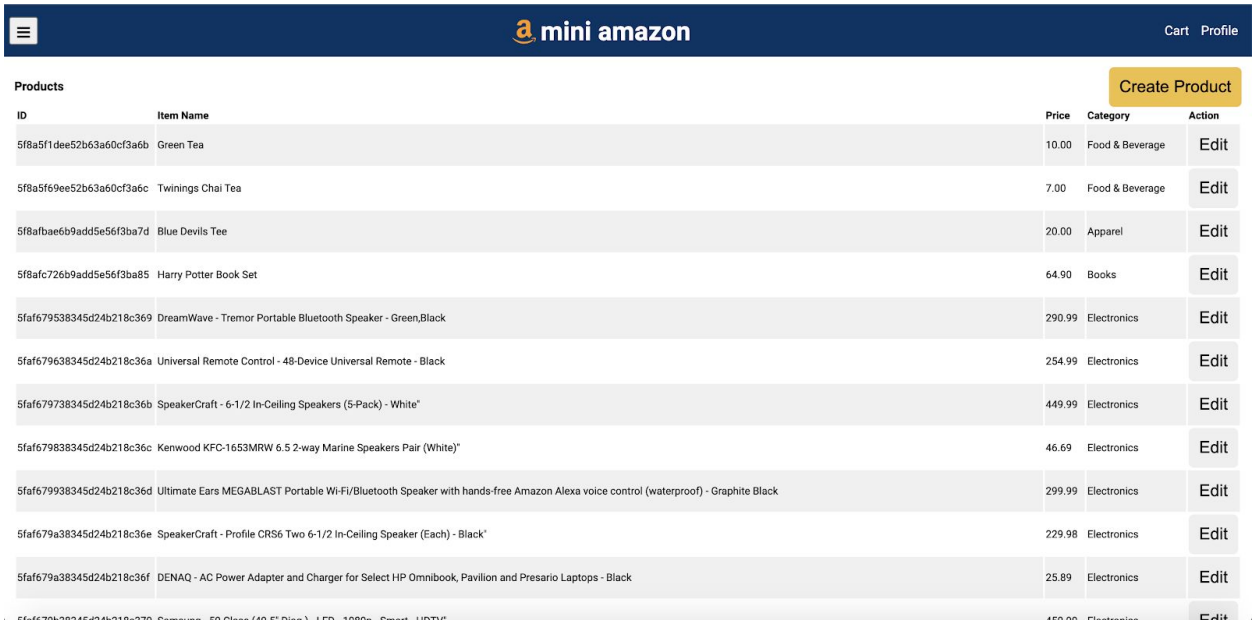
- User profile: all useful user information
- Name, username, email, account balance all displayed
- Buttons allowing user to add to their account balance, review their purchase history, add/edit products (seller account only), and logout

Purchase history page



- List of items purchased: item image, item name, seller name, price, count and purchased date/time. Item name should link to the item page. Total amount spent at the bottom.

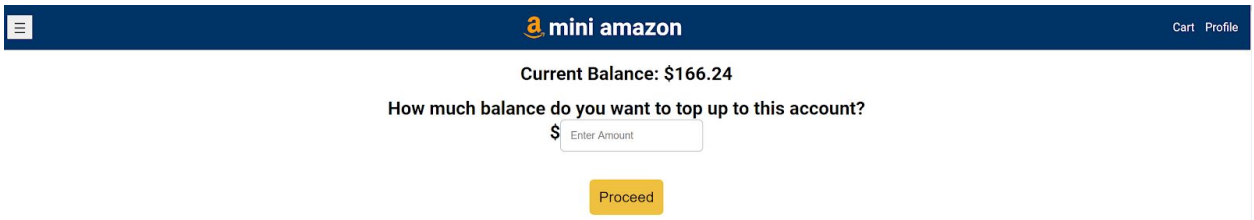
Selling list page (only if the user is also a seller)



- List of items being sold by a seller: item image, item name, price, and count. Item name should link to the item page. A delete button should appear on the right of each item to delete this item.
- Add/modify item button. When clicked, a container appears add/modify the list item. Input space for all fields needed for an item: item name, item image, item description, seller (this seller only) description, price, and count.
    - “Add item” should have all fields blank
    - “Modify item” should have them filled (editable) with previous values

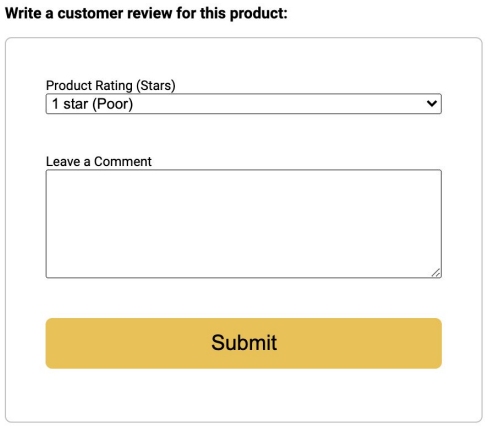
- When modifying an item, sellers can edit the price and quantity, but not other attributes to prevent interference (in terms of description, item name, etc) with other sellers selling the same item.

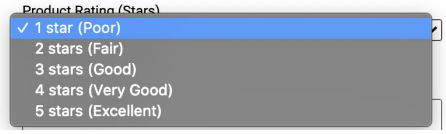
Add Balance Page



- Miniamazon icon, User dropdown, Cart: (as before)
- Shows current balance of user.
- One input box:
    - How much balance do you want to top up to this account?
- One button:
    - Proceed.

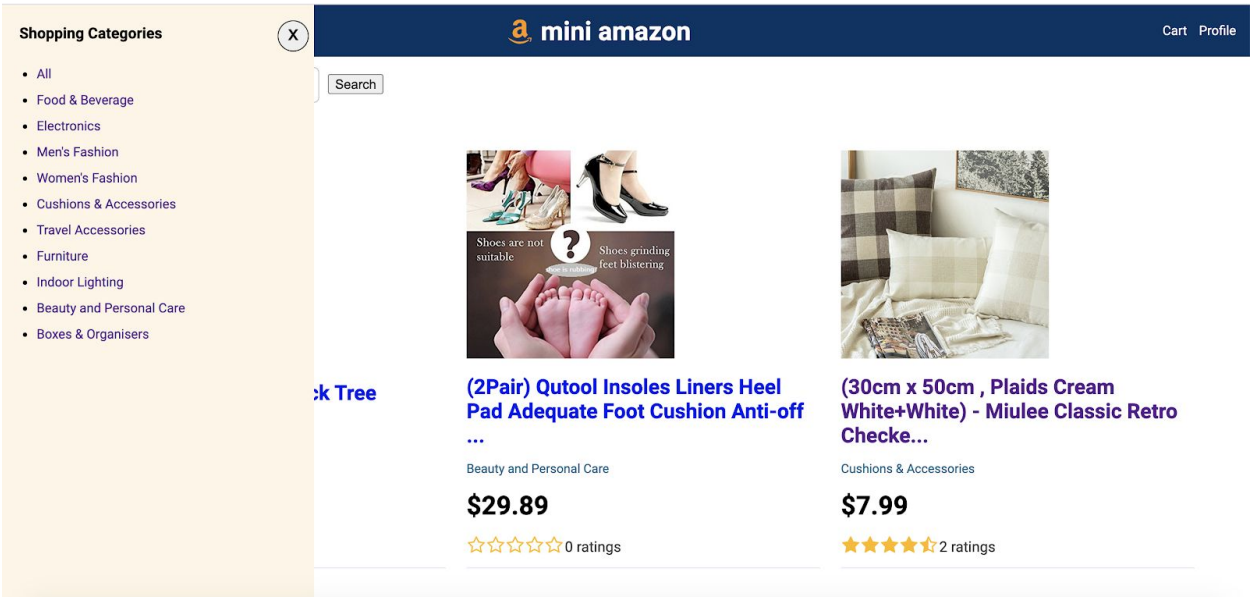
Add Review Page





- Input space for rating for item, (optional rating for seller), and free text for the review itself.
- Submit button: confirms and submits the review

Sidebar



- Interactive sidebar to browse products by category

Dataset

To populate our database, we obtained data from data.world. Specifically, four datasets were used: Electronics, Men's Fashion, Women's Fashion, and Amazon Australia Product Listing. Relevant columns from the datasets including product name, category, product image url, price, and description were extracted and cleaned up. We wrote a python code utilizing the Python HTTP library Requests to post products to our database.

```
import pandas as pd
import string
import numpy as np
import requests
import csv

url = 'http://localhost:8080/api/products/'

df = pd.read_csv (r'electronics_c.csv')
df2 = df.drop_duplicates()
k=0
prev=''
with open('id_price.csv','w',newline='') as f:
    writer = csv.writer(f)
    writer.writerow(['id','price'])
    for row in df2.iterrows():
        imgurl = row[1][2].split(',')
        if(imgurl != prev):
            prev = imgurl
            if('pisces' in imgurl[-1]):
                myobj = {'itemName': row[1][0], 'category': row[1][1], 'image': imgurl[-1], 'description': row[1][0]}
                print(myobj)
                k = k+1
                x = requests.post(url, data = myobj)
                ids = x.text.split("id\":")
                writer.writerow([ids[1][1:25],row[1][3]])
        if(k==200):
            break
```

Approximately 200 products were added to each of the nine categories: Electronics, Men's Fashion, Women's Fashion, Cushion & Accessories, Travel Accessories, Furniture, Indoor Lighting, Beauty and Personal Care, and Boxes & Organisers. Our current database holds approximately 1800 products.

MongoDB Injection Attack

As we were using MongoDB instead of SQL, we had to guard against MongoDB attacks. We used the mongoose library to implement our database, which already has built-in type casting and validation. This means that if an API request is sent to the backend, mongoose will validate the schema fields such that for example, if it expects a string and receives an object, the query will throw an error.

Also, we used the mongo-sanitize library for further protection against query selector injection attacks. The sanitize function is applied on every API request input that will be used to query the database (example below), and it will strip out any keys that start with '$' in the input, so it can be passed to MongoDB without worrying about malicious users overwriting query selectors.

```
//Get list of items sold by seller id
router.get( path: "/", isAuth, isSeller, async(req : Request<ParamsDictionary, any, any, ParsedQs> , res : Response<any> ) => {
    const sellList = await SoldBy.find({seller: sanitize(req.user.uid)}).populate('item');
    res.send(sellList);
})
```

Work Division

	Backend/ Frontend (based on our design)
Anna Xu	Register, profile, seller list, create/edit product, deployment
James Leong	Backend models and endpoints for items, transactions, soldBy Add to cart, cart checkout, transactions, profile rerendering, balance updating/transference, search bar/filter MongoDB injection guards Database setup and population User authentications Deployment
Michael Jang	Backend model and route for user User profile page, balance page Populating database
Mina Kim	Add/display reviews (seller, rating, item), register page, list of other sellers, site appearance/logo
Vanessa Chen	Add selling list page (show all seller products, seller can add and edit products) Add cart functionality (add to, delete elements in cart) Login Page Site appearance Deployment

Challenges

It was our first time building a full-stack end-to-end web app from scratch, so we definitely faced obstacles while trying to implement the various functionalities of our app. We also had to efficiently divide the work and combine the front and back end properly. We spent some time trying to get authentication to work, as each user has access to different sets of functionalities/data.

Future Work

In the future, we might implement more features such as recommended products based on user preferences, and the ability to make and process return requests. Another interesting feature we could add would be to allow different login methods, such as connecting through Google or Facebook. This would require us to look into more authentication techniques and utilize the APIs from those sites to fetch a certain user's data. Additionally, we could try to improve the response time/interactivity time for much larger datasets, because realistically an e-commerce website would need to handle datasets that are much larger than ours. This could be done through pagination of the products or lazy loading of API calls.