Phones Glore

Final Project Technical Report

SE/COM S 3190 – Construction of User Interfaces
Spring 2025

Team Members:
Naven Prabakar -
prabakar@iastate.edu

Milo Mucu -
milomucu@iastate.edu

May 11, 2025

1. Introduction

The problem in today's phone market is the competition that exists between various phone companies (Apple, Android, etc.). This means to purchase phones of each brand, you need to visit the companies website directly; however, this can get tiring since it would require switching back and forth between various websites. Creating a central phone store that is able to sell all types of phones can help reduce the work needed in purchasing a phone.

This was the inspiration for our final project. Additionally, for the mid-term project, we did a similar concept and wanted to make an extension from it so it has better UI design, more features, and account creations. The project is inspired mainly. There already exist pseudo versions of these stores across the internet and we wanted to enhance what was already out there to be much better for consumers.

2. Project Description

Major Features: Purchasing workflow & Selling workflow. (There are other features but these were the major ones). The purchasing workflow involves adding things/removing things from the cart, then navigating to the payment view. Once at the payment view, we can purchase the phones that we had picked. The selling workflow involves posting your phone up for sale for other users to be able to purchase. It involves submitting a form with the details of the device being sold. It is then added to the global store for other users to see and purchase.

User Flow for Purchasing:

GlobalStore -> add item to cart -> Cart view -> purchase button -> purchasing view -> confirm payment

User Flow for Sell:

GlobalStore-> Sell Items Button -> Sell Form -> Submit Form -> Item being sold added to User info -> Global Store view updated

CRUD for Purchasing:

Create/Update- Post request for adding items to the cart. This adds items they have selected to the cart for purchasing. The information of the id is sent to the backend which is then added to the user profile for users to be able to purchase. It will also calculate the costs with the discounts as the user adds items to the cart.

Read- Get requests , get all the items when they are added to the cart. When navigating to the cart view, a get request is sent to grab all the things inside of the cart. The information is sent to the backend, where the backend grabs the array of ids from the database and is sent to the front end. The front end then uses the id's to display what they added to the cart.

Delete - Delete requests, delete items in the cart. When navigating to the cart view, the users can remove items from the cart if they changed their mind. Upon removing an item, the request is sent to the backend where they remove the item from the player's profile in mongoDB. It also reduces the price of the total that is displayed in the cart view.

CRUD for Sell:

Create - Post request for creating the initial sale. The user fills out the sell form. The information is sent to the backend where the information of sale is added to a json file of all the things being sold. The id of the new sale is added to user information inside of mongoDB to keep track of what items the user is selling

Read - Get a request when the user opens the selling screen. A use-effect is used to grab all the items the user is selling from the backend to display when the selling screen is opened. It first grabs the id's of the items being sold then grabs the full information from the JSON file of the store information on the backend.

Update- Put request is sent when a user edits an item they have posted onto the store (assuming no one has bought it yet.). They can edit their post and repost with the new updated information to the global store. The JSON file holding the store information is the only thing changed.

Delete- Delete request is sent when a user wants to delete something they have posted on to the store. They can click delete, and the item will be taken off the store. The request is sent to the back end where the JSON file of the store data removes the info on the item and the id is removed from the User profile inside of MongoDB.

In the features, some entities affected are: MongoDB (data is changed), JSON file (writing and reading to it), use states (switching between the views), use effects (upon changes are called)

3. File and Folder Architecture

```
Frontend/
    src/
        App.jsx
        components/
            assets/ (holds all of the images)
            list of components
            data/ (fake store data)
```

The front end structure follows the traditional react set up. Inside the src folder holds App.jsx, which includes the central component to connect all the other components. Inside the components there are two folders. Assets holds the images used in the project, while data has the data.js file for the phone store data. The rest of the files inside of components are all the react components of the project.

Backend/

- App.js
- routes/
  - JSON file
  - list of routes

The backend structure is basic. At the root there is App.js, this is the node server that runs the backend for requests to be sent and received. In the routes folder, are all the routes that the server can talk with. Inside of each route is an express that represents the restful api requests being received and processed. The JSON file has the data for the phone store information. Based on the requests, information is being constantly read from here or written. It works in coordination with mongoDB as mongDB can't store files.

4. Code Explanation and Logic Flow

4.1. Frontend–Backend Communication

The frontend uses "fetch" that takes in a parameter. The parameter is the server endpoint for the api to be called. Depending on the request, the fetch expands to more parameters for a body. The request body is filled out before the request is sent to the backend. The backend receives the request and processes the information. On successful request, the status is set to 200 and the information the frontend request is sent to the frontend. On a bad request, the status is set to a value above 400. This typically indicates incorrect password or email. On an unknown error, the status is set to 500 and a message saying an internal error is sent. The front end receives the status and what was sent. If it was a 200, the frontend takes the information and displays it for the user to see, If it was a 400, then it displays an alert message for the user to see saying what they submitted was wrong. If it was 500, it does the same thing as a 400, but instead it displays internal error.

4.2. React Component Structure

App.jsx, acts as the central component to all other components. It initializes the app and uses the useState to display the login screen by default. All components receive the argument of a useState to switch between the different components. Each component has useState for the user profile. The user profile needs to remain consistent, so each component has access to the user information and the changes that need to occur. One switching between the navigation bar, the useState changes it a different name, which forces the switch to a different page. All states have useState to either switch between the pages or update the user profile of however logged on. Some states like, sell.jsx have a use effect. This is because when the user puts up an item for sale it needs to immediately update the user screen. This is the overall react component structure.

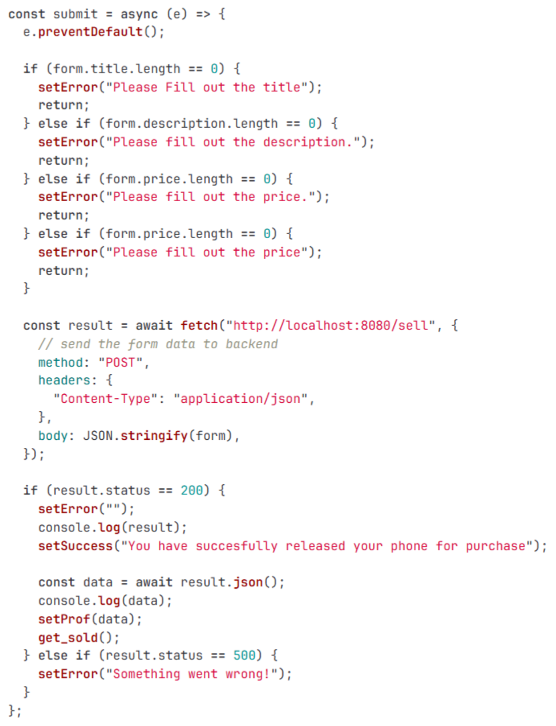
4.3. Database Interaction

The database is used when it requires updating a user's information. The database that was used was MongoDB. Upon the creation of an account, the user's default profile is added to the collection. When the user adds an item to a cart, it adds the id of the time to the user profile so it saves what the user has been doing. When the user creates a phone to sell, it adds the id to a separate category to the user so it saves what the user has been selling. It is also used when the user updates their settings. In settings you can change username and password.

4.4. Code Snippets

Include 2–3 meaningful snippets (React component, backend route, DB logic).

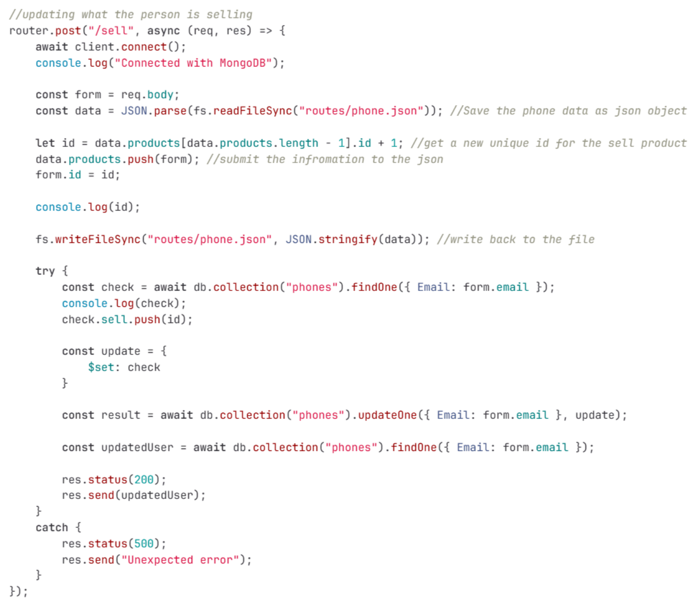
React Component: For the sell screen



```
const submit = async (e) => {
  e.preventDefault();

  if (form.title.length == 0) {
    setError("Please Fill out the title");
    return;
  } else if (form.description.length == 0) {
    setError("Please fill out the description.");
    return;
  } else if (form.price.length == 0) {
    setError("Please fill out the price.");
    return;
  } else if (form.price.length == 0) {
    setError("Please fill out the price");
    return;
  }

  const result = await fetch("http://localhost:8080/sell", {
    // send the form data to backend
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify(form),
  });

  if (result.status == 200) {
    setError("");
    console.log(result);
    setSuccess("You have succesfully released your phone for purchase");

    const data = await result.json();
    console.log(data);
    setProf(data);
    get_sold();
  } else if (result.status == 500) {
    setError("Something went wrong!");
  }
};
```

In this react component is the following request for an item to be created in the item shop. It checks if the form they filled was actually filled and nothing was left blank. Once that is done, a fetch to the api is called. It defines the request as a post, and it converts the object to a json string and sends it. If the request got a 200, it uses the useState to set the error to null, and set the success to display a success message. Then it uses the usestate to update the user profile and get_sold() invokes a useeffect. This is an example of the code throughout our project.

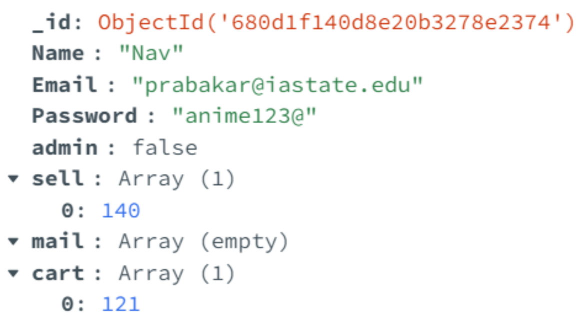
Backend Request to the Sell:



```
//updating what the person is selling
router.post("/sell", async (req, res) => {
    await client.connect();
    console.log("Connected with MongoDB");

    const form = req.body;
    const data = JSON.parse(fs.readFileSync("routes/phone.json")); //Save the phone data as json object

    let id = data.products[data.products.length - 1].id + 1; //get a new unique id for the sell product
    data.products.push(form); //submit the infromation to the json
    form.id = id;

    console.log(id);

    fs.writeFileSync("routes/phone.json", JSON.stringify(data)); //write back to the file

    try {
        const check = await db.collection("phones").findOne({ Email: form.email });
        console.log(check);
        check.sell.push(id);

        const update = {
            $set: check
        }

        const result = await db.collection("phones").updateOne({ Email: form.email }, update);

        const updatedUser = await db.collection("phones").findOne({ Email: form.email });

        res.status(200);
        res.send(updatedUser);
    }
    catch {
        res.status(500);
        res.send("Unexpected error");
    }
});
```

In this post backend request, the request is going to take the new item being created and add it to the phone store data and update the user profile. It starts off by connecting to MongoDB. After, it reads all the data from the json file. It then creates a new unique id for the project by incrementing the last id in the json file. It then saves the new information back to the json file. After, it talks to the database and grabs the user profile associated with the request. It adds the new created id to check.sell, which is an array. It puts the updated profile back inside the collection and sends a 200 and the updated user to the front end. On error it sends a 500 with an error message.

MongoDB Logic:

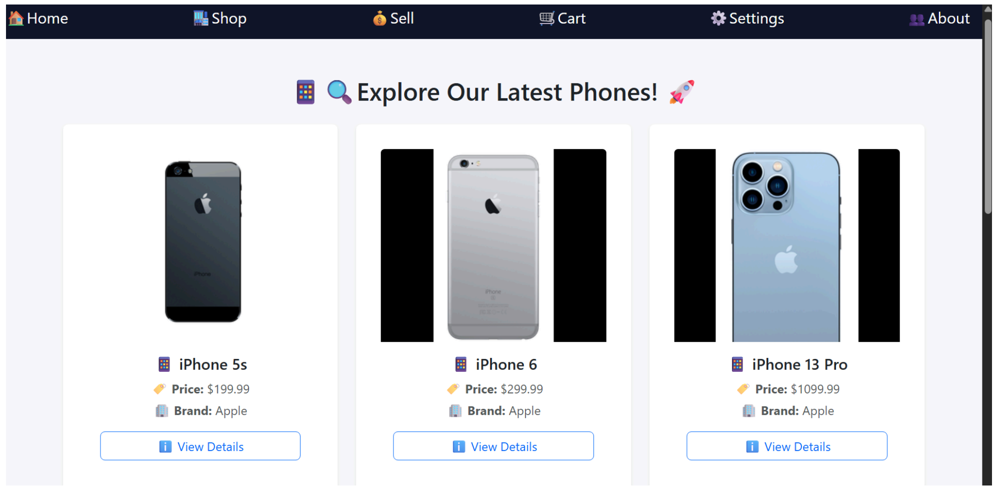


```
  _id: ObjectId('680d1f140d8e20b3278e2374')
  Name : "Nav"
  Email : "prabakar@iastate.edu"
  Password : "anime123@"
  admin : false
▾ sell : Array (1)
    0: 140
▾ mail : Array (empty)
▾ cart : Array (1)
    0: 121
```

This is how the user profile is saved inside of MongoDB. Their username, email, and password is saved during the account creation. Admin is a boolean that confirms if they are an admin or not. Sell is an array of ids then are the id's of the things they have sold. The cart is an array of id's of the things they have added inside of the correct.

5. Web View Screenshots and Annotations

Home-View feature 1:

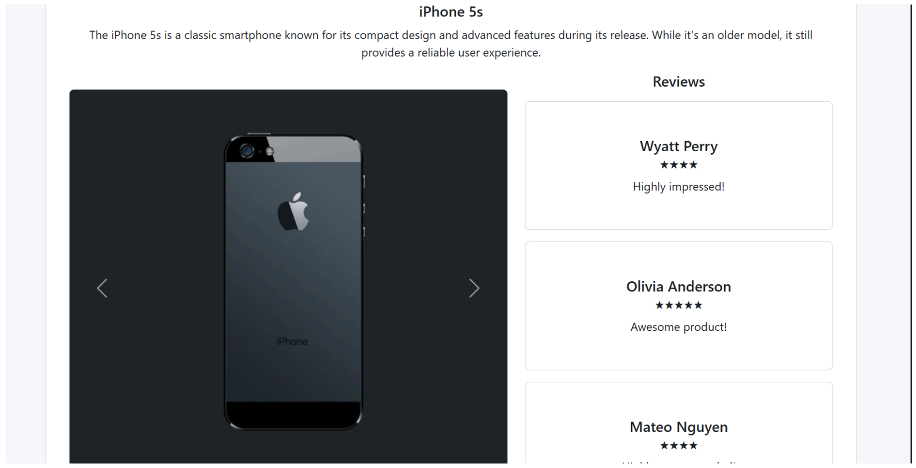


Explore Our Latest Phones!

iPhone 5s
Price: $199.99
Brand: Apple

iPhone 6
Price: $299.99
Brand: Apple

iPhone 13 Pro
Price: $1099.99
Brand: Apple



iPhone 5s

The iPhone 5s is a classic smartphone known for its compact design and advanced features during its release. While it's an older model, it still provides a reliable user experience.

Reviews

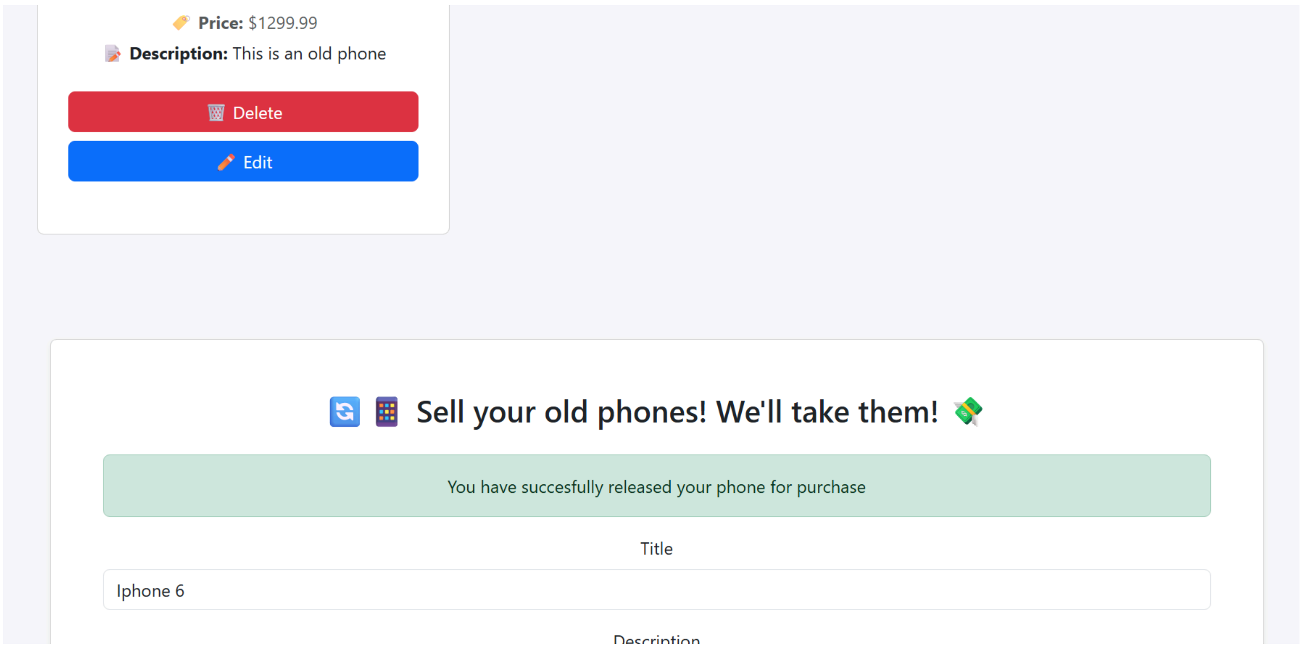
Wyatt Perry
★★★★
Highly impressed!

Olivia Anderson
★★★★★
Awesome product!

Mateo Nguyen
★★★★

The following feature is the homeview. The home view welcomes the user to the store with a client message as well as provides a preview of the phones we offer. Clicking on view details will open up another screen (2nd picture). It is a carousel for the phones with the reviews of the phones on the side. The Iphone model is displayed with a description as well.

Selling Screen Feature two:


Price: $1299.99

Description: This is an old phone

Delete

Edit

Sell your old phones! We'll take them!

You have succesfully released your phone for purchase

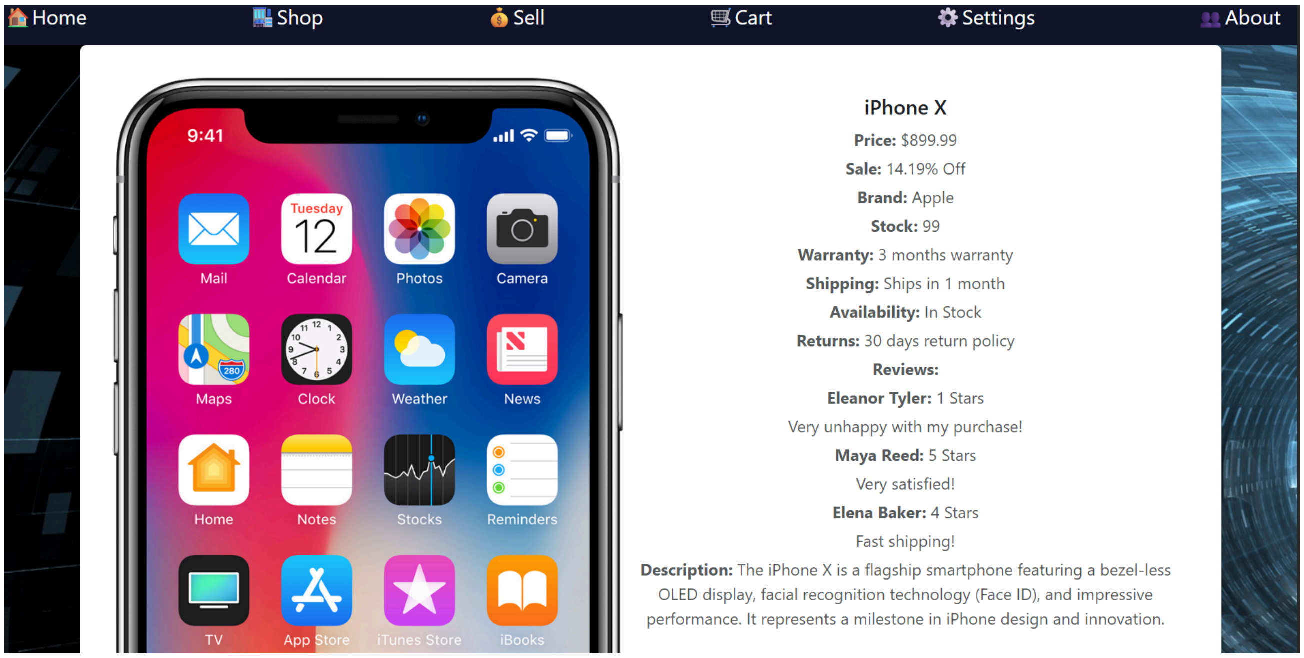
Title

Iphone 6

Description

This is the selling screen. The user can submit a form to sell phones. Upon submitting the form , it appears in their list of phones they are selling (upper left of the screen). They can edit or delete the sale they just put up. Clicking on the delete button will delete the sale they just made, while edit means they can update the information without removing it from the store. The green message in the form also informs them that the phone was successfully launched.

Shop screen Feature three:



iPhone X

Price: $899.99

Sale: 14.19% Off

Brand: Apple

Stock: 99

Warranty: 3 months warranty

Shipping: Ships in 1 month

Availability: In Stock

Returns: 30 days return policy

Reviews:

Eleanor Tyler: 1 Stars

Very unhappy with my purchase!

Maya Reed: 5 Stars

Very satisfied!

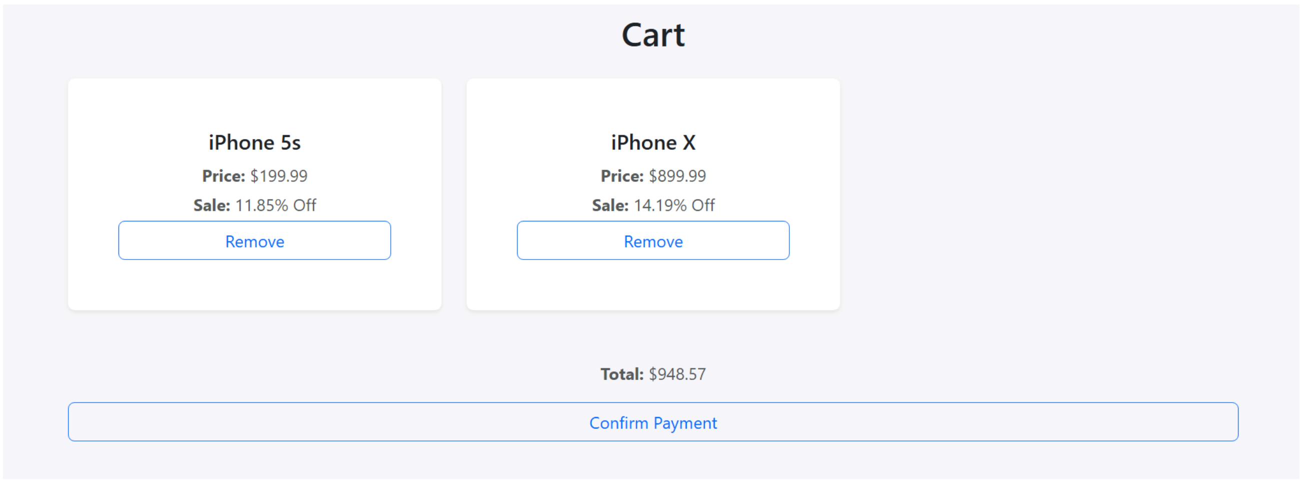
Elena Baker: 4 Stars

Fast shipping!

Description: The iPhone X is a flagship smartphone featuring a bezel-less OLED display, facial recognition technology (Face ID), and impressive performance. It represents a milestone in iPhone design and innovation.

This is the shop screen. The shop screen will inform the user with everything they need to know about the phone: warranty, returns, discounts, etc. The user can look at a more in depth analysis of each phone they are looking at. This provides more scope than the home page where they can only preview the phones. The screen also includes a cart button, where users may add it to the cart if they wish to purchase it.

Cart view Feature four:



This is the view of the cart after the users have added it to the cart after browsing the store. They can either proceed to the payment with confirmation payment, which will open up a payment menu or they can remove the item from the cart if they have changed their minds.

6. Installation and Setup Instructions

Prerequisites:

MongoDB, with a collection called phones and a database called phones
Node.js

To get started first clone the repository:
git clone https://git.las.iastate.edu/se-coms-3190/spring-2025/final-project/PS_2.git

To enter the project:
Cd PS_2

To start the backend:
Cd Backend
npm install
npm run dev

To start the frontend:
Cd .. (leave the backend folder)
Cd Frontend
npm install
npm run dev

7. Contribution Overview

Feature	Owner
Login & Sign up	Naveen Prabakar
Home view	Naveen Prabakar
Selling Screen	Naveen Prabakar
Shopping Screen	Milo Mucu
Payment view	Milo Mucu
Cart view	Milo Mucu
Admin view	Milo Mucu
Settings	Naveen Prabakar
About	Naveen Prabakar

Note: though Naveen did more features, Some of Milo's features had higher levels of difficulty which compensates the balance.

8. Challenges Faced

1. One of the biggest debugging issues was when we were writing to a JSON file. During the sale requests, it requires writing to a JSON file since it needs to update the store information. However, the issue was the JSON file was originally inside of the frontend directory. When the user made the request, it would suddenly refresh and take me back to the login page. This was very confusing until we realized that because we were writing to the file in the frontend, it would make react think it was a change being made to the code and refresh the frontend. To resolve this, we simply moved the JSON file to the backend.
2. UI development was another challenge. We discovered an JSON file that held a lot of information and we needed a creative way to be able to display it. Time constraint was a lot because we had other things for other classes as well. The solution was to use Bootstrap's carousel. This provided an easy UI implementation to look through all the phone pictures, as well as provided a quick way of creating a nice UI. Adding emojis was also another solution.
3. Lastly, another issue we ran into was a login request error. When Naveen finished the login and signed up, he told Milo it works, but when Milo tried it, it didn't work on his end. To debug, tons of console.log statements were inserted to figure out the issue. The issue ended up being that the front end used Capital p in password, while the backend used lowercase p in password. After the fix, we made sure to note that we had to be careful with letter sensitivity.

9. Final Reflections

We were able to create a full stack application with interactive features where the frontend was composed of React and Tailwind, while the backend was Node and Express with the databases being MongoDB. We've met our goals of being able to design a clean front-end while having good backend logic to support it. Some improvements to be made are: using mysql instead of mongoDB, this could have helped have a more organized structure in the backend. We had to pseudo create relationships by using a JSON file. Learning/using more external api's is also another thing to consider moving forward. Thank you for a great semester!