FlipKart

Danny Song, Ting Ting Xu, Jin-Han Han

Team 3

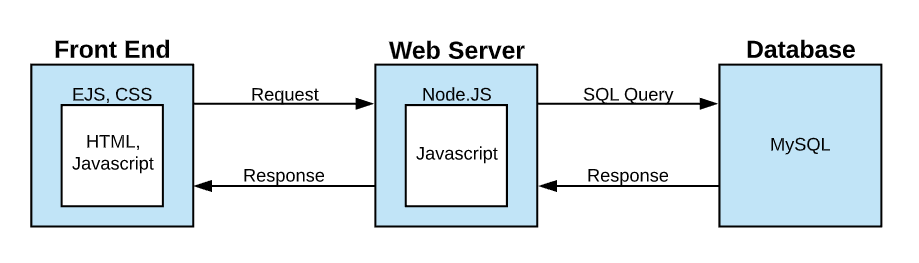
CS157a Section 1

**Project Description:**

This project will create a worldwide online shopping website to support e-commerce service. It provides an electronic retail and business platform for registered users to purchase and sell products. This electronic platform will replace the typical online shopping website. It will provide small business or individuals an easier way to post and sell their products, and also serves buyers a more fast and simple way to finish payment transaction. It will become the mainstream online shopping system in the United State and overseas The stakeholders will be 77 billion people worldwide, those people including retail suppliers, entrepreneurs. This platform will provide users a simple, fast, and secure e-commerce site.

**System Environment:**

This ecommerce site will be designed to run within the clients browser and thus the GUI will be created using HTML(EJS), CSS, and Javascript. The web server will be handled using Node JS and thus will be developed in Javascript. This Node JS server will provide the client with the front end as well as connect them to the database which will use MySQL as its database management system. The Node server will communicate with the MySQL database through the Node JS webserver.



**Hardware and Software Used:**

* Three Tier Environment
  + MySQL version 14.14 Database Management System
  + Node JS Express web server
  + EJS(HTML w/ embedded JS), CSS client side
* Application Languages
  + Javascript, HTML, CSS,

**Functional Requirements:**

Users of our software will be buyers of products. They will be able to access our system through their browsers and log in to their own personal accounts. This account will keep track of the user’s personal information such as name, shipping address, billing address, and payment information. As users, they will have CRUD functionality for this information. Each account will also keep track of information such as their current shopping cart.

The main feature of the application allows users to buy products. They will be able to interact with the GUI to purchase items of their choosing. Users will also be able to check their shopping carts and remove items from it if they choose to do so.

**Functions:**

Browse Products:

* The user shall be able to browse database for items that are for sale
* The application shall be able to provide a list of the items available for purchase in a grid-like format and allow the user to select a specific category to view a subset of the database.
* The user will be able to see the each product’s name, picture, and price

Display Product Information:

* The user shall be able to click on a product to view more information
* If a user clicks on a product a new page will be loaded that displays additional information about the product. The page will provide the user additional information about the product as well as the option to purchase and add to their wishlist.

Search for Products:

* The user shall be able to search the database for products by typing a product into a search bar.
* The system shall be able to provide the user with any products that match their query. If the system is unable to find suitable products, the user will be notified with a message.

Sort Products by:

* The user shall be able to specify how they would like the products to be sorted.
* The system shall be able to rearrange the products based on the user’s selected sorting criteria

Add Products to Shopping Cart:

* The user shall be able to add a specified quantity of a selected product to their shopping cart. The cart will only be accessible by the user that owns it

Edit Cart:

* The user shall be able to edit the quantity of items or remove items from their shopping cart.
* The system shall be able to recalculate the total cost when the cart is edited.

View Cart:

* The user shall be able to view the products and the quantity of each product in their shopping cart, as well as a total cost for all the items.

Edit Personal Information:

* The user shall be able to edit their personal information such as their name, email, shipping address, and payment information.

Create User:

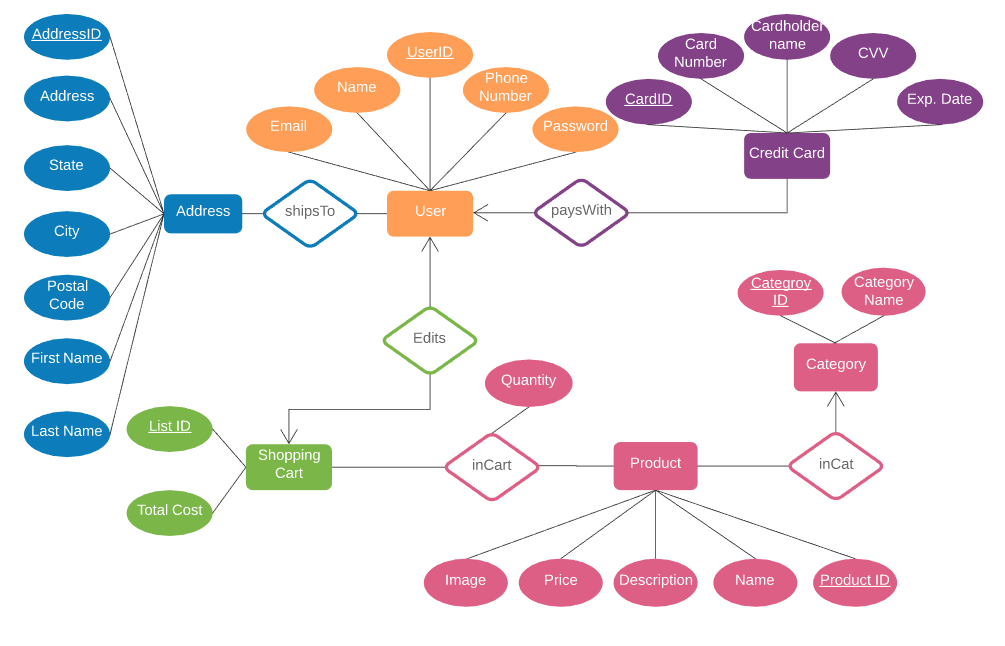
* The user shall be able to create a new account through a provided form. Completion of this form will be required to successfully create the account.
* The system shall be able to prompt users to enter all relevant information and store it for future reference and changes.

Login:

* The user shall be able to login in to their account with the proper credentials
* The system shall be able to reject improper credentials and keep track of the session of the logged in user

**Project Details:**

**ERD:**

****

**Entity Sets:**

* User - This set represents the registered users of Flipkart. This set store user information.
* Address - This set represents the addresses registered to Flipkart users.
* Shopping Cart - Each user has a shopping cart assigned to them. The shopping cart will store items selected for purchase by the user. The shopping cart can be edited by the user that owns it.
* Credit Card - This set will inherit the primary key from the payment information set. It will store credit card information such as credit card number, CVV, name, and expiration date.
* Product - This set represents all the items that are being sold on FlipKart. It will store information such as a unique product id, product name, description, image, and price.
* Category - This set represents the categories that a product can belong to. It stores a unique category id and the name of the category.

**Relationships:**

* PaysWith - This relationship is a many to one relationship. It keeps track of the payment information of each user. Each user can have multiple payment methods but each payment method can only be associated with one user.
* Edits - This relationship is one to many. It keeps track of the shopping cart associated with specific users. Each user can have many shopping lists but each shopping list can only be attributed to one user.
* Contains - This relationship is a many to many relationship. It keeps track of the products in a shopping cart. Each shopping cart can have many products and each product can be in many carts
* In - This relationship is a many to one relationship. Each product can only have one category and each category can have multiple products.
* Ships to - This relationship is a many to many relationship. It keeps track of the addresses associated with specific users. Each user can have multiple addresses and the same address can be associated with multiple users.

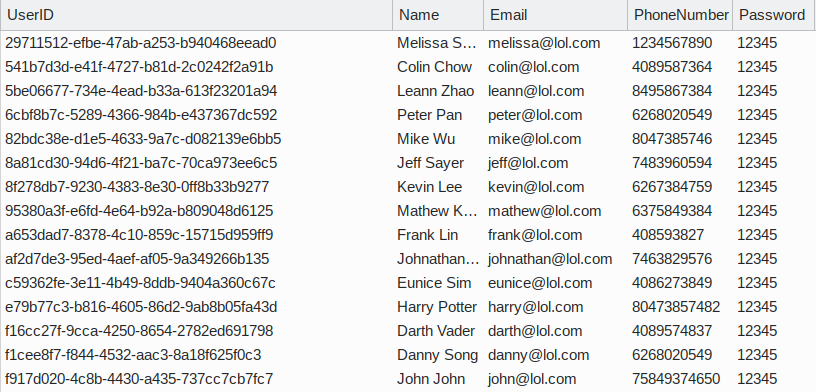
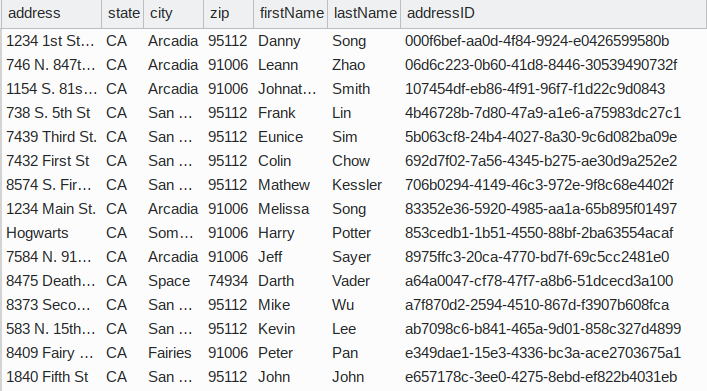
**Entity Schemas:**

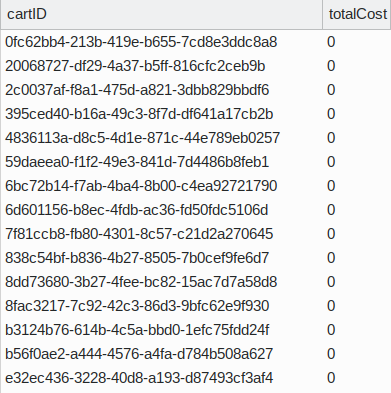
* User(UserID, Name, Email, Phone Number, Password)
* Address(AddressID, Address, State, City, Zip, First Name, Last Name )
* Carts(CartID, totalCost)
* CreditCard(CardID, CardNum, CardName, CVV, expDate)
* Product(ProductID, Name, Description, Price, Image)
* Category(CategoryID, CategoryName)

**Relationship Schemas:**

* shipsTo(AddressID, UserID)
* paysWith(UserID, CardID)
* edits(UserID, cartID)
* inCart(CartID, ProductID, quantity)
* inCat(ProductID, CategoryID)

**Database Tables:**

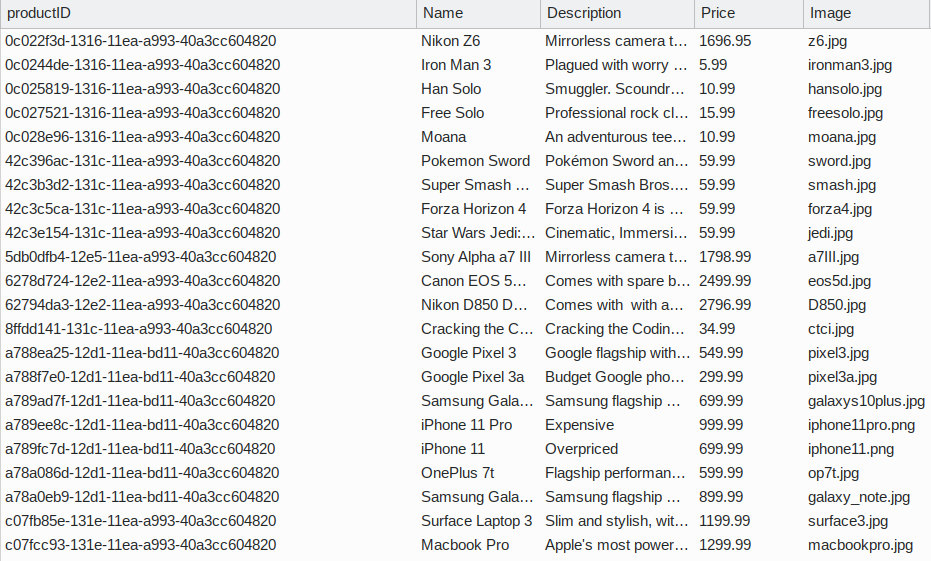
* User
* Address
* Carts



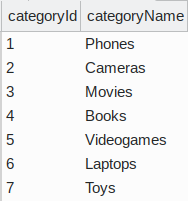
* CreditCard



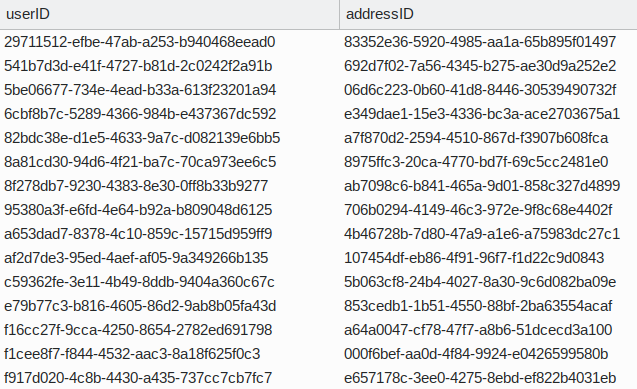
* Product



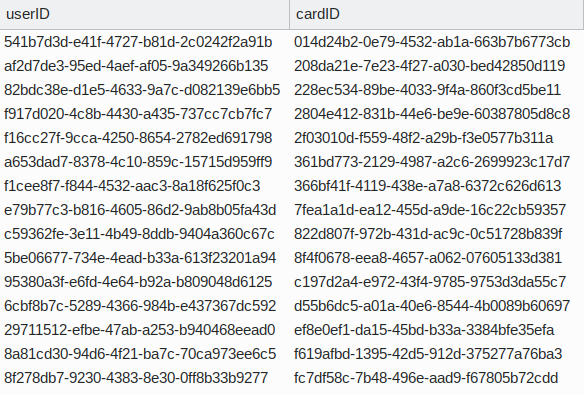
* Category



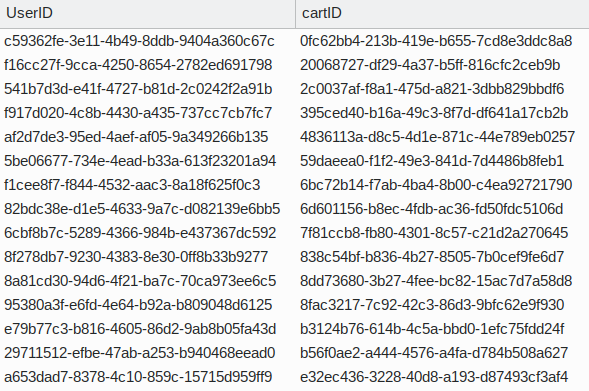
* shipsTo



* paysWith



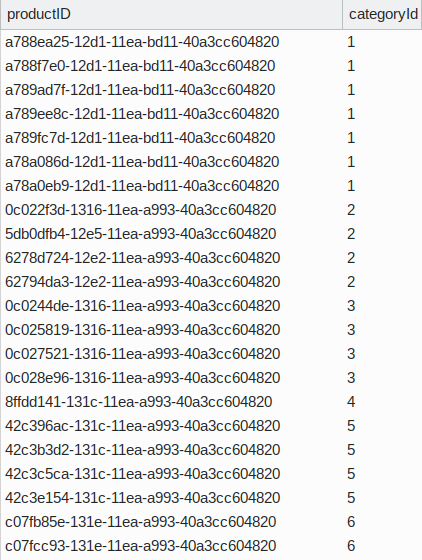
* Edits



* inCart



* inCat



**Implementation:**

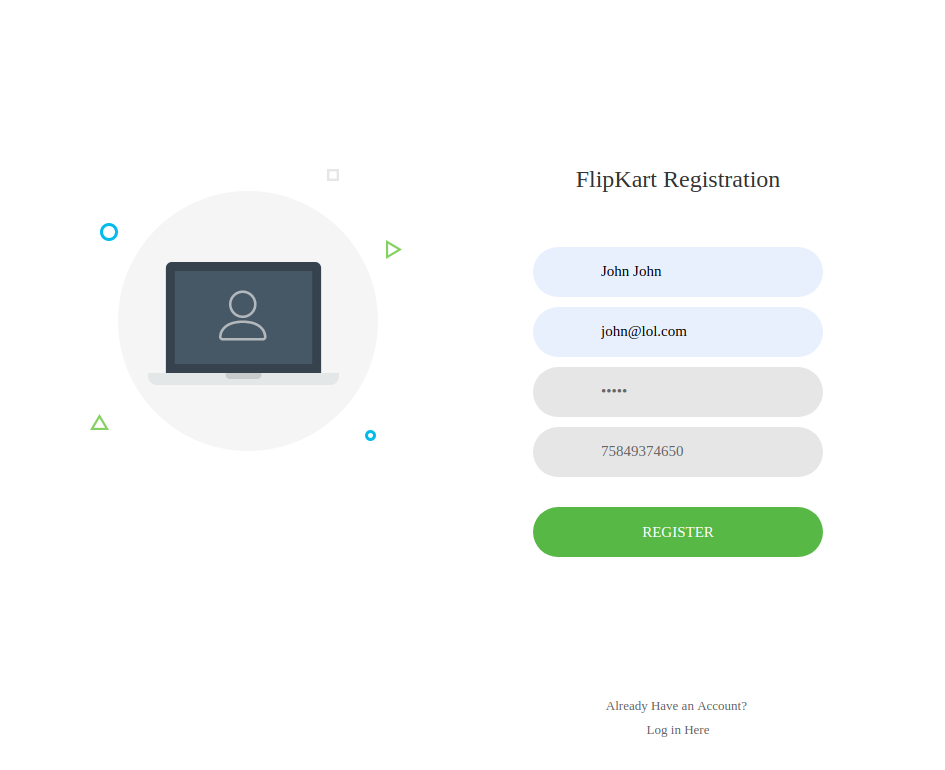
**Architecture:**

The front end of FlipKart was developed in HTML, CSS, and Javascript. The HTML used the EJS template to allow easier communication with the back end. The MDB and Bootstrap CSS libraries were used to achieve the material design of FlipKart. Javascript was used to validate certain fields such as email on the client side. The web server was developed using Node.JS. The Node server allows the client-end to communicate with the database management system.

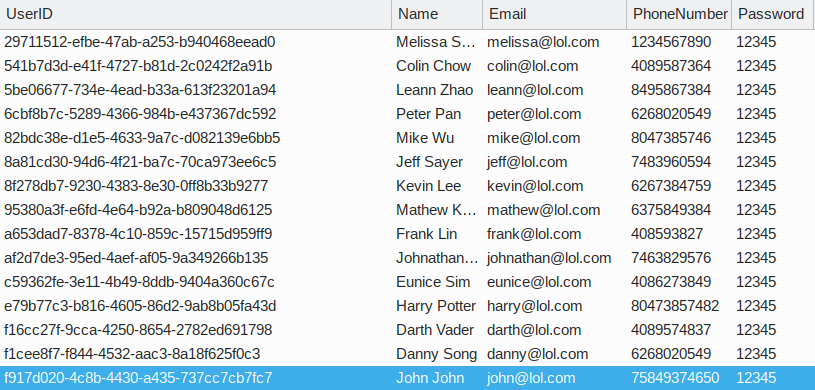
**Design Implementations:**

Create User:

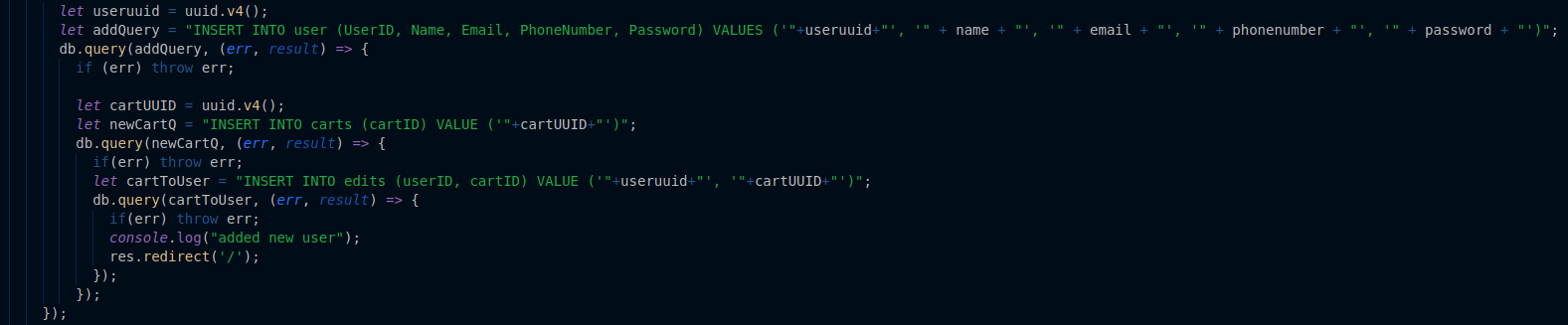
Registration UI



Database is updated with John John’s UID, Name, Email, PhoneNumber, and Password

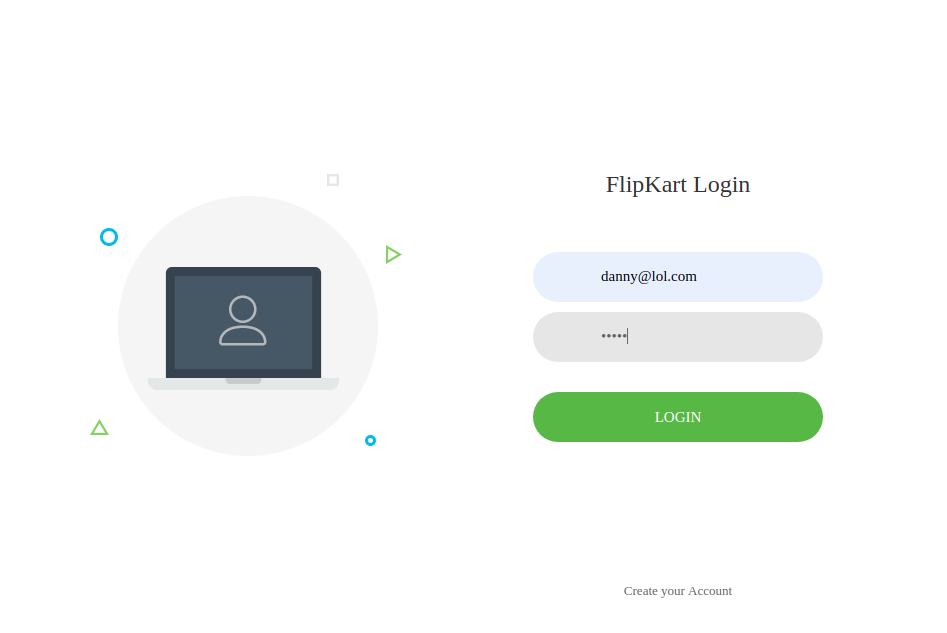


Node.JS first queries the user table to see if the email input already is being used. If it is, the user is redirected back to the registration page. If the email has not been used, the user provided information is inserted into the user table. A new cart entity is created in the carts table and the user is associated with that new cart in the edits table.

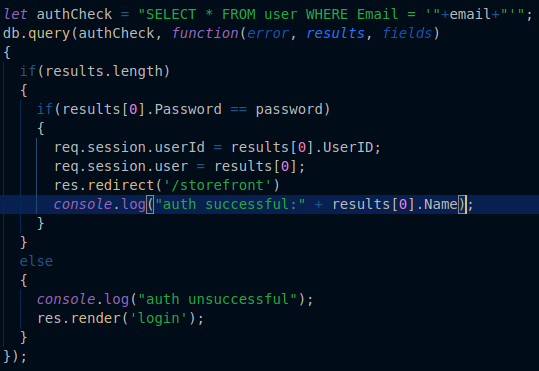


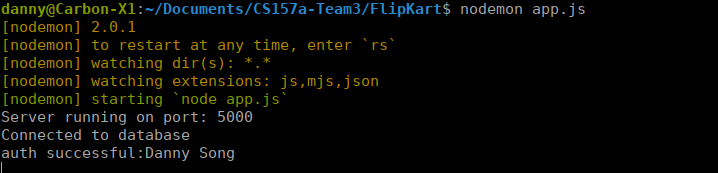
Login:

Login UI



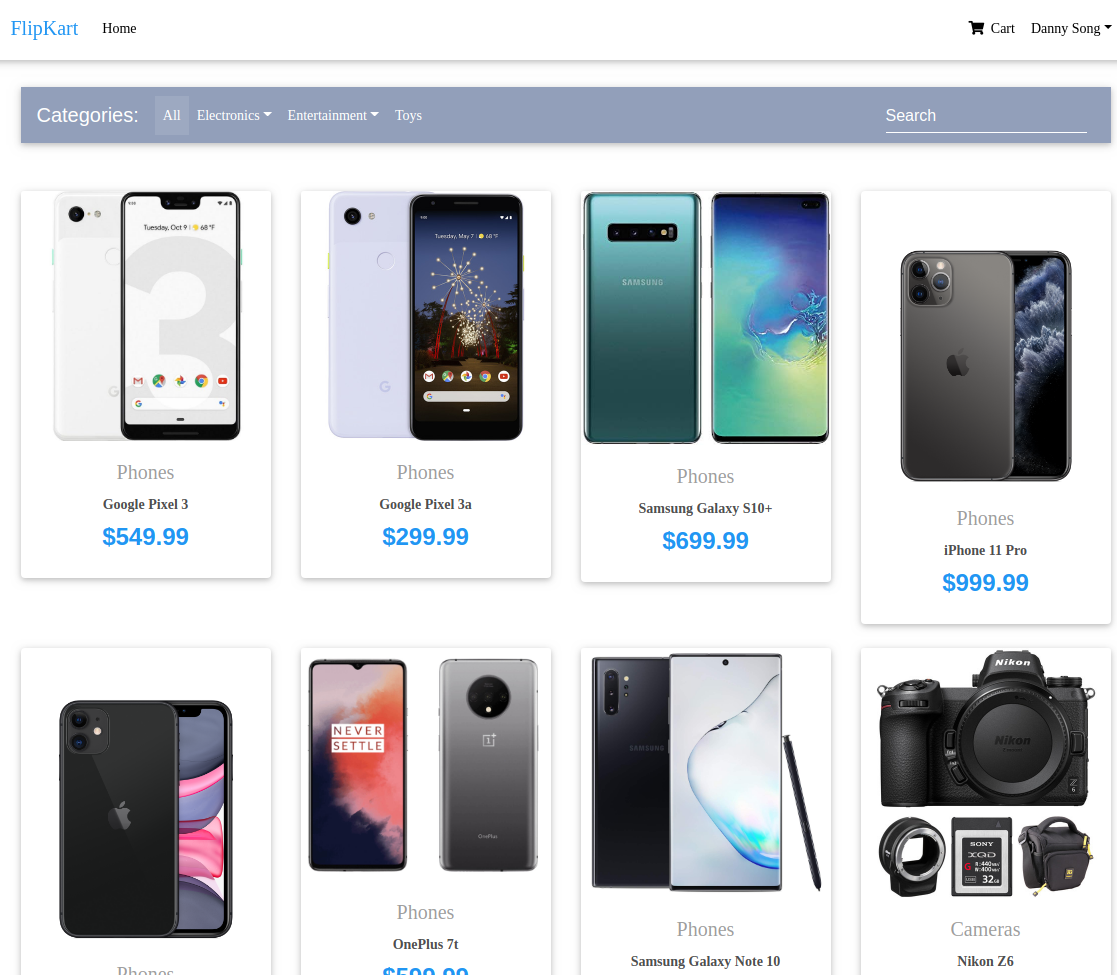
Node.JS first checks if a user exists with the input email. If a user does exist, the password stored in the database is compared with the password input by the user. The password can be directly compared because the passwords are stored in plaintext as security was not in the scope of this project. Once the password is verified, the session is set with the user’s credentials and the user is redirected to the storefront, where they can browse products. If there doesn’t exist a user with the inputted email or if the password is incorrect, the user stays on the login page



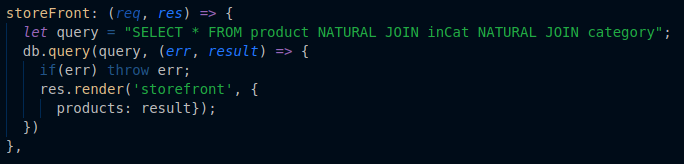


Browse Products:

StoreFront UI

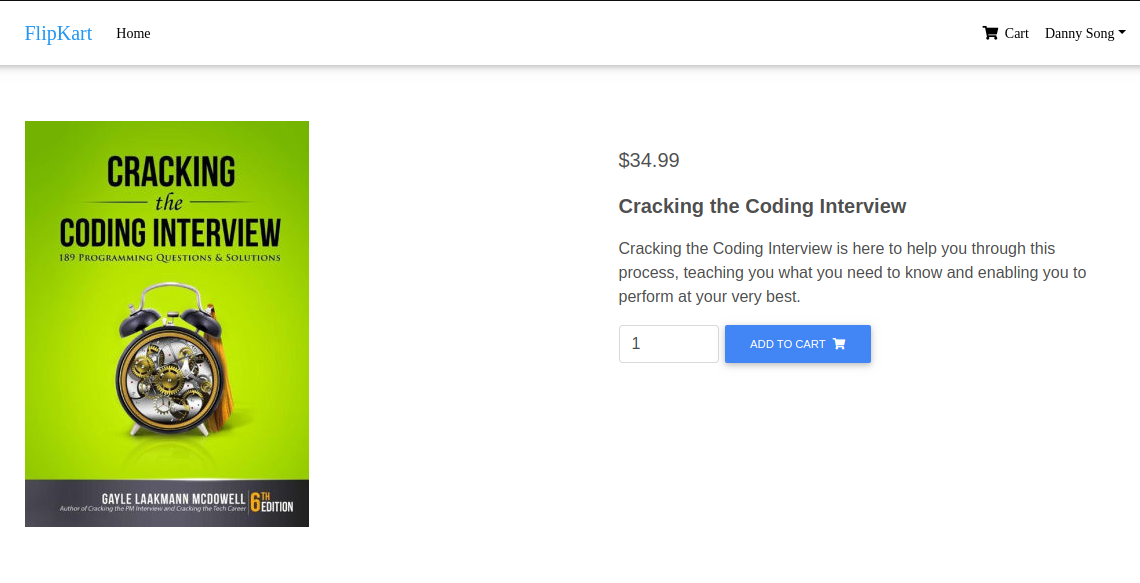


Node JS simply queries the product table for all of the product entities. The result of this query is sent to the storefront.ejs file as products. Each card is individually build for each product using an embedded JS for loop.

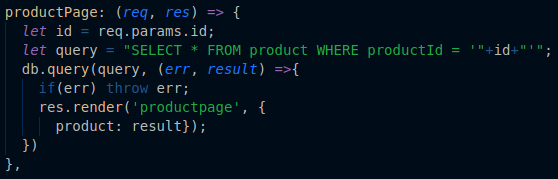


Display Product Information:

Product Page UI

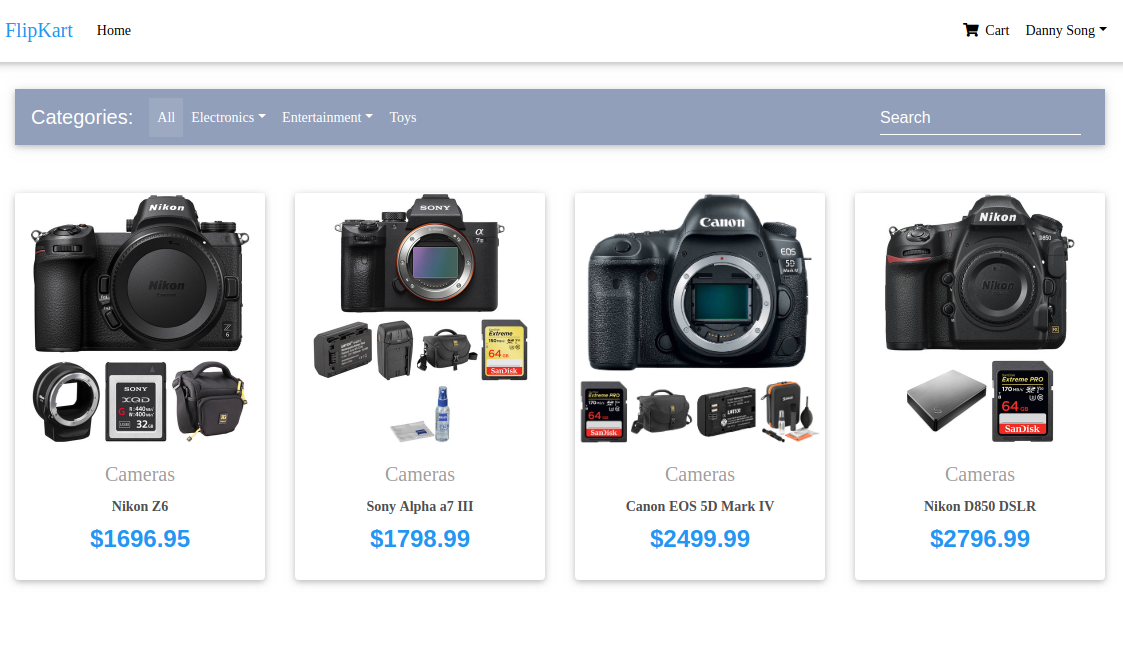


Each product card on the storefront passes it’s productID as a parameter to NodeJS when the card is clicked. To display the product page, NodeJS queries for the entity in the product table where the productID equals the one passes by the card. The result of this query is sent to productpage.ejs. The product and its attributes are accessed using embedded JS (<%=product.atributeName%>

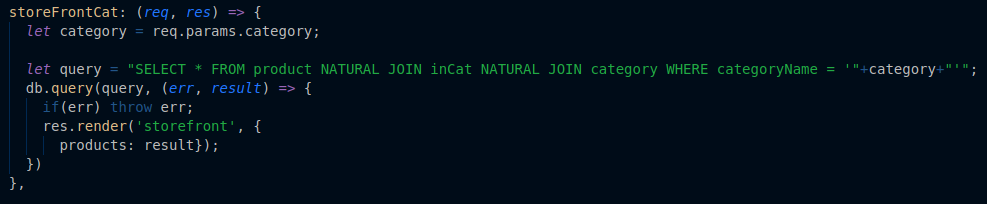


Sort Products by:

StoreFront UI where cameras is selected

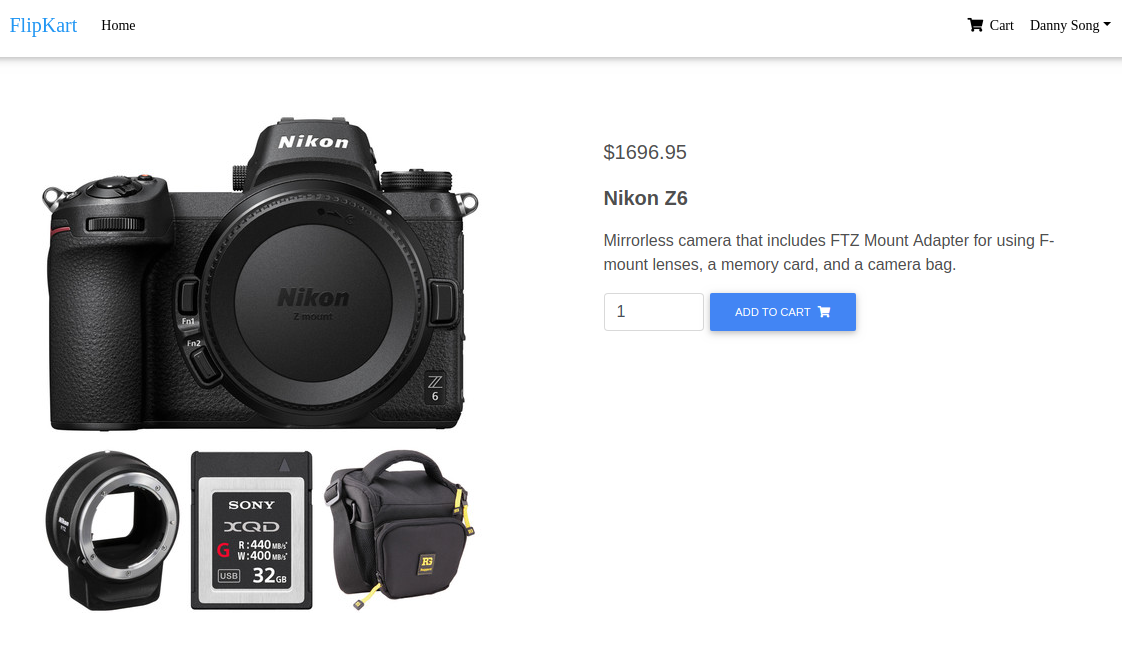


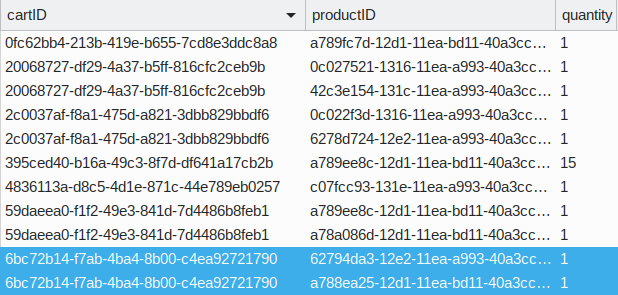
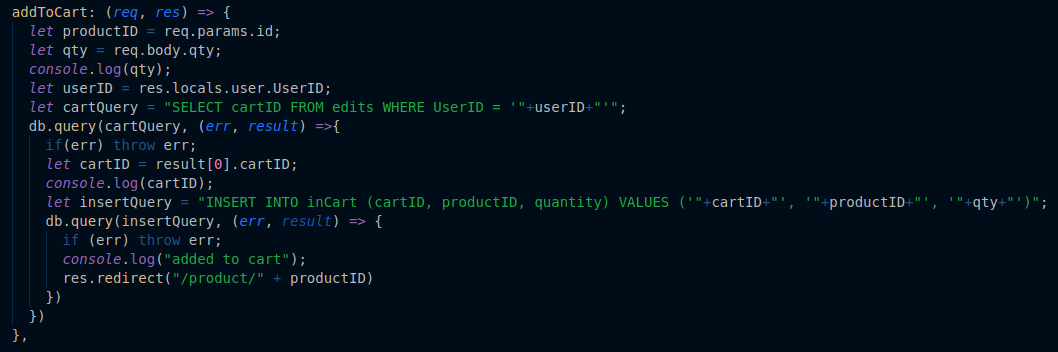
Each button sends the category to NodeJS as a parameter. The product table is joined with the incat table using the product id. Then the resulting table is joined with the category table using the categoryID. This gives us a table with all the product data as well as its category name. Then all products whose category names are the same as the parameter passed by the button are selected. The result is sent to storefront.ejs and displayed in the same way as above.



Add Products to Shopping Cart:

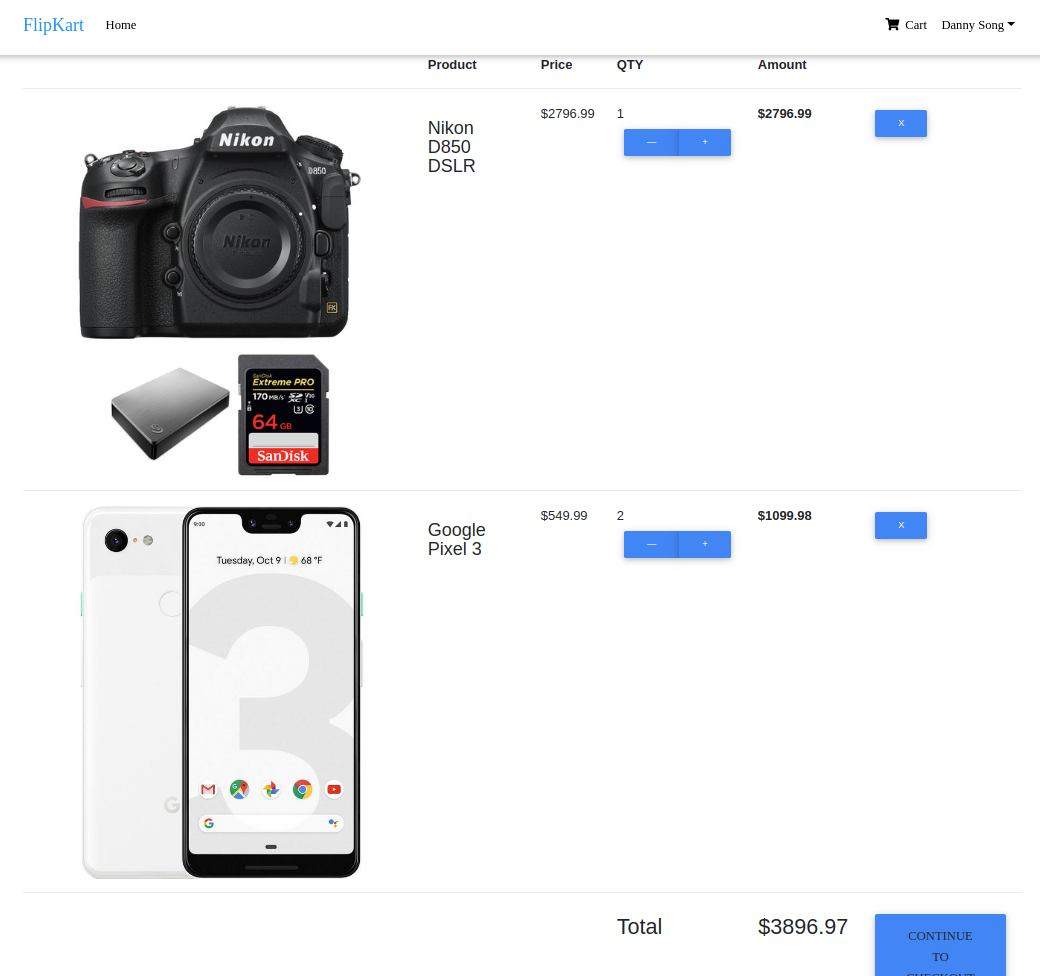
Product page with the add to cart button and quantity field



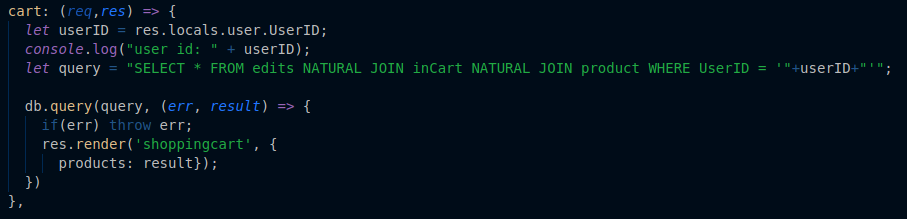
The product is again passed as a parameter by the button to NodeJS while the userID is access from the session we stored at login. NodeJS first queries the edits table to select the cartID that belongs to the user. Once it has the correct cartID, a query is made to insert the cartID, productID, and quantity into the inCart table. 

View Cart:

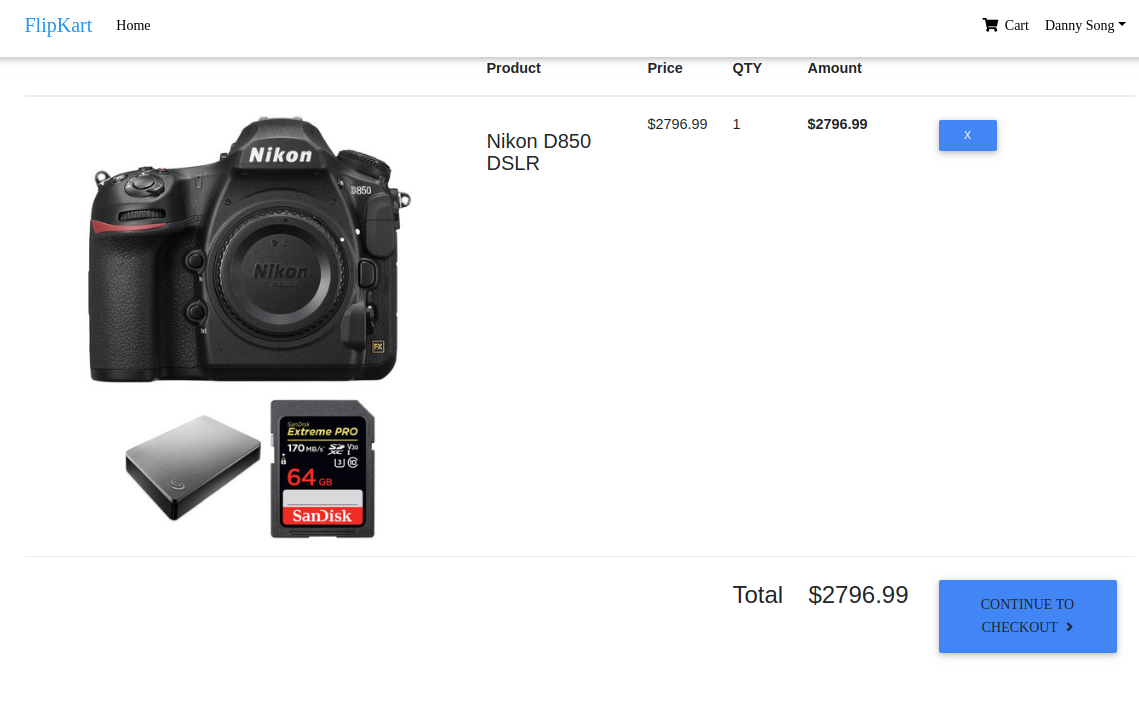
Cart UI



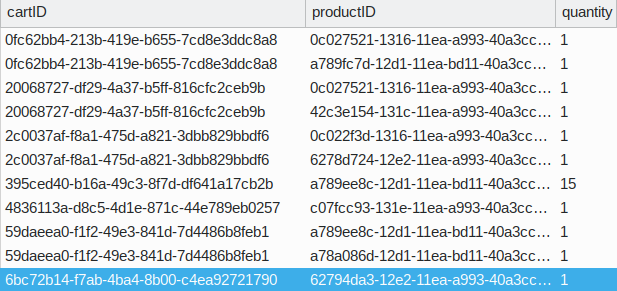
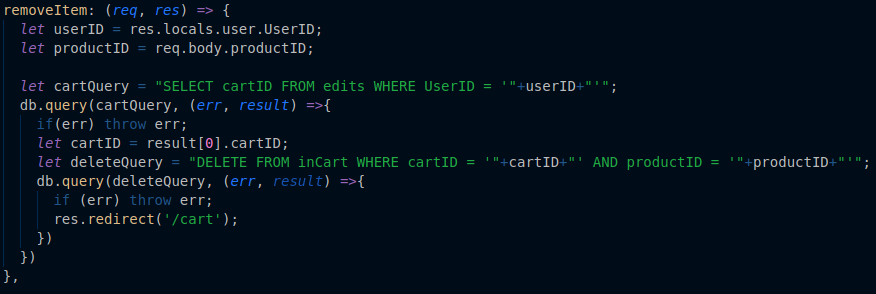
The edits table is joined with the inCart table using cartID. The result is joined with the products table using productID. The resulting table contains each product in each user’s cart. From there, the resulting table is queried for the entities where userID equals the userID obtained from the session. Each product and its attributes are accessed and rendered using an embedded JS for loop.



Edit Cart:

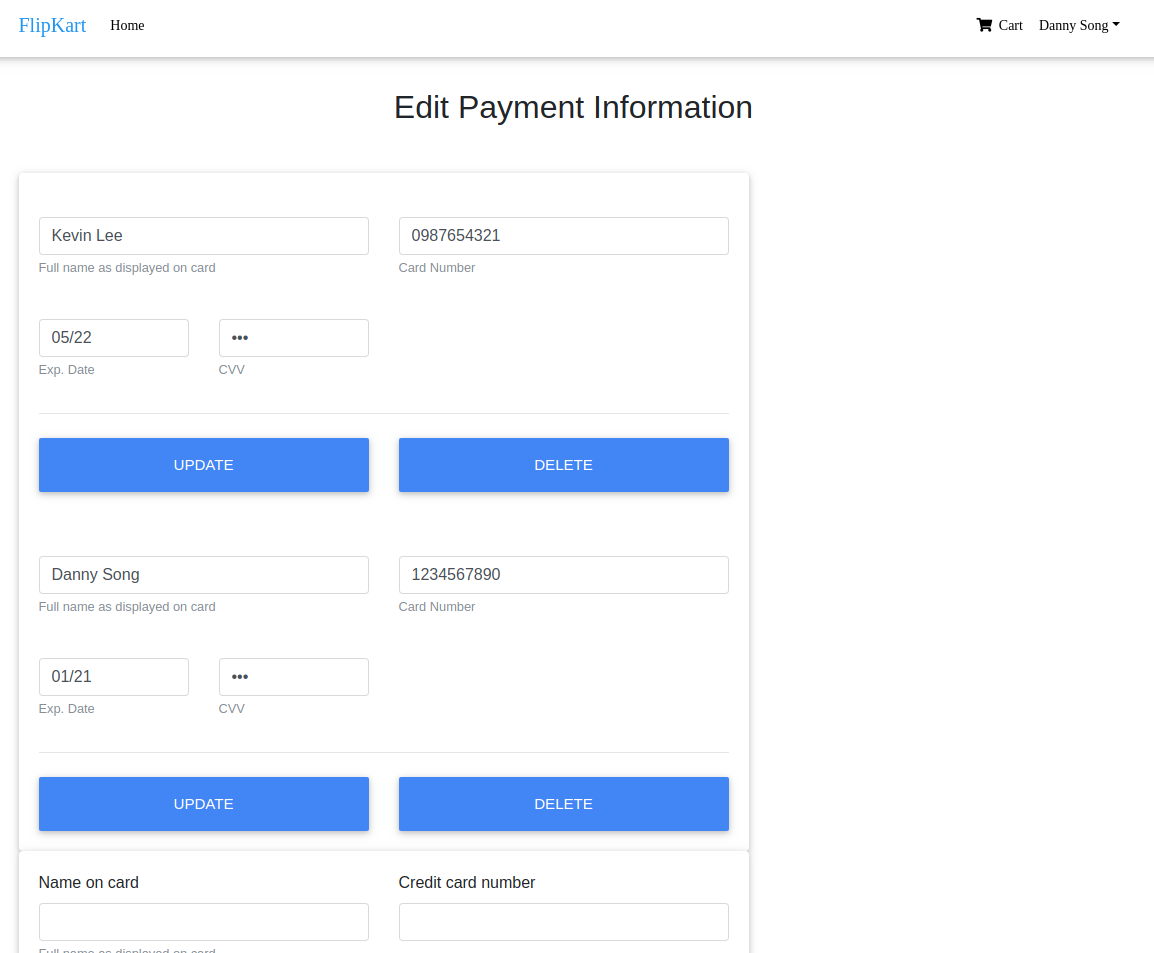
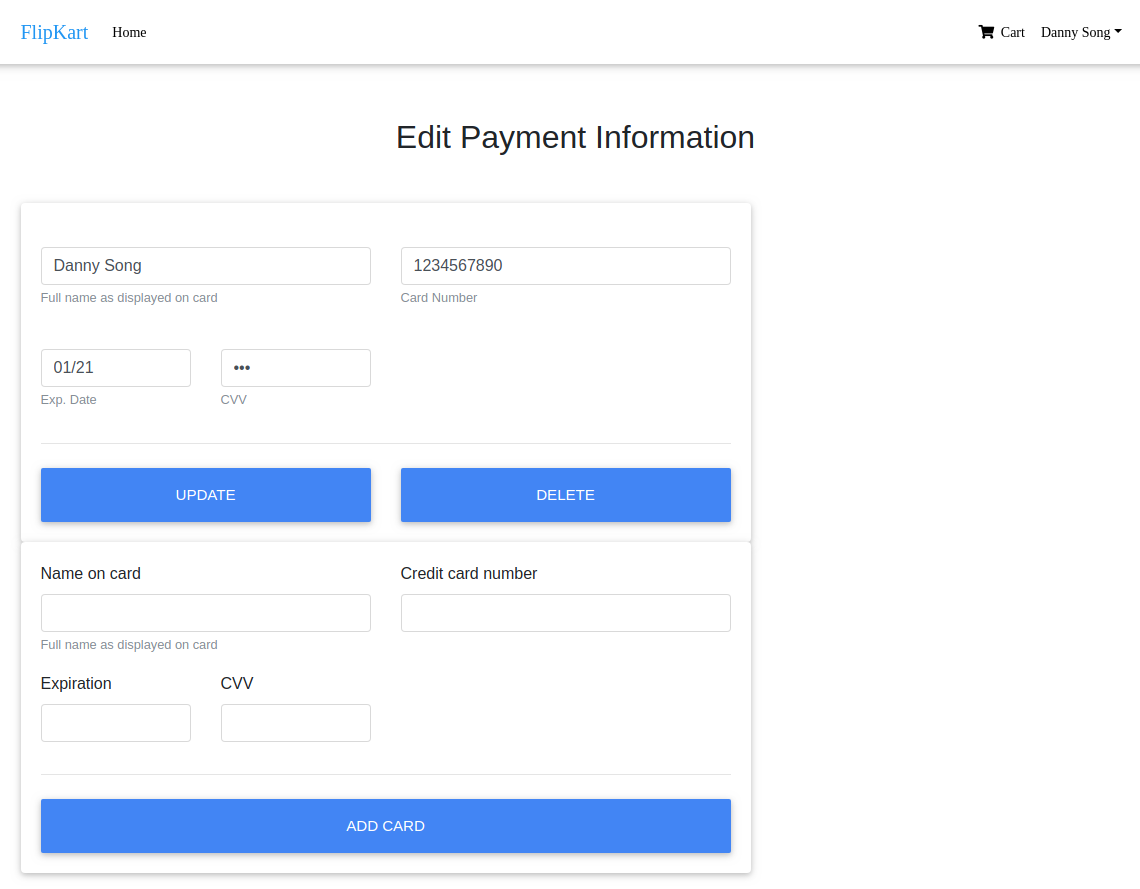


Node JS uses the userID stored in the local session to determine which shopping cart to edit. The product ID is passed as a value by the button. When the delete button is pressed, a deletion query is made to delete the entity in the inCart table where the cartID is the one found from the edits table and the productID is equal to the one passed by the button.

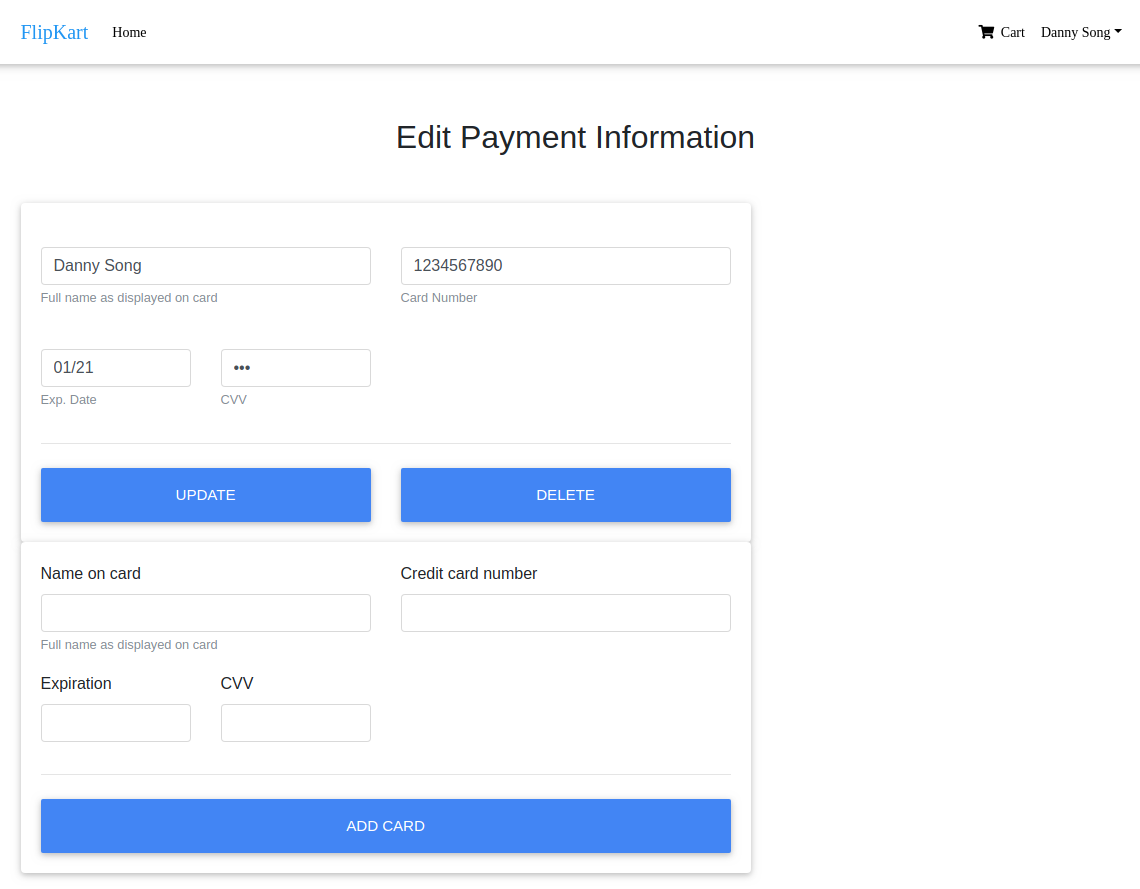
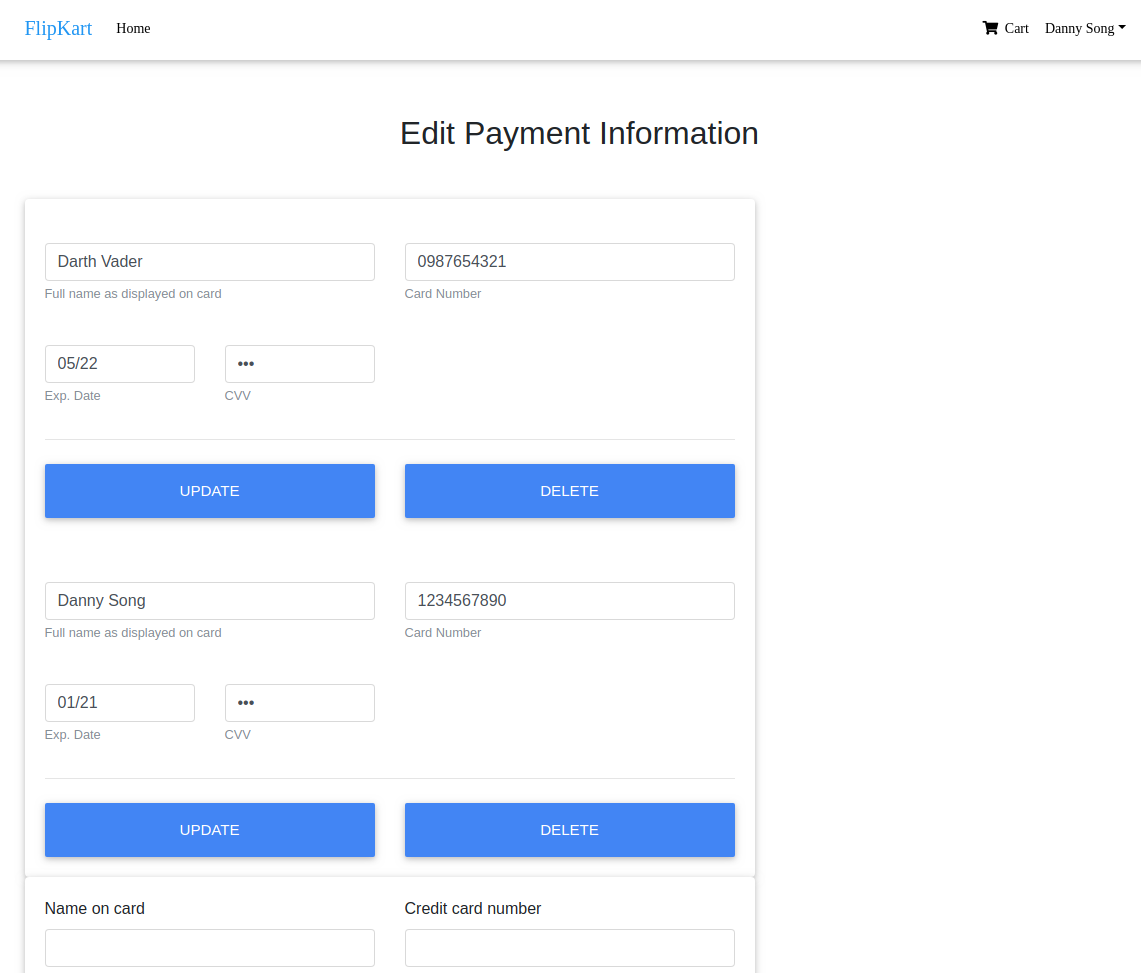


Edit Personal Information:

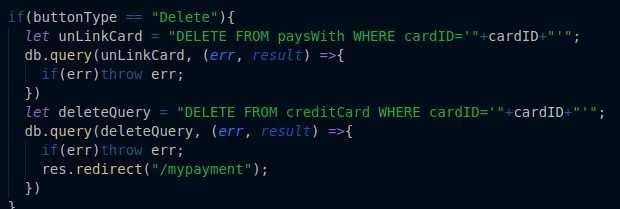
Edit Payment UI Adding a new Card



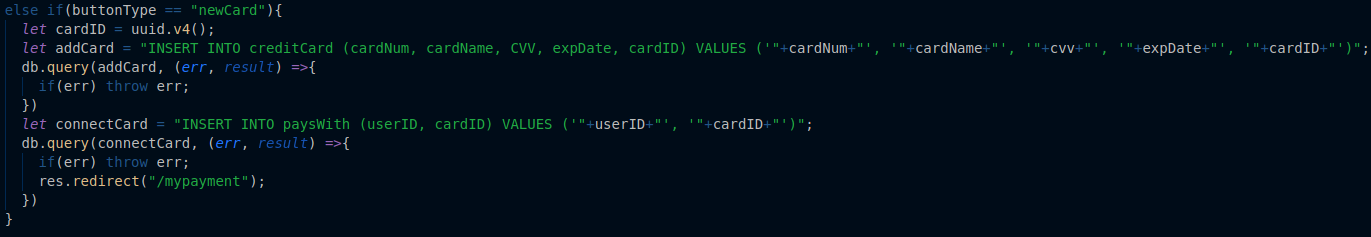
Edit Card Delete Card

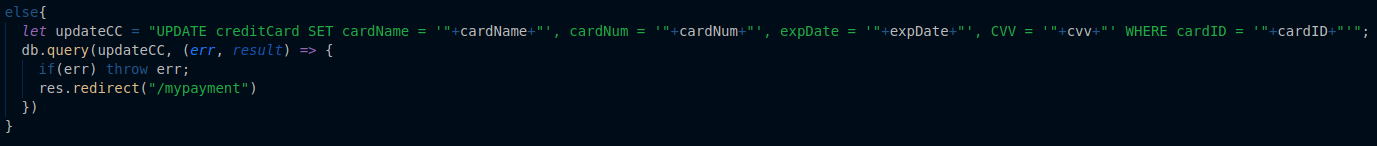


Each of the three buttons each send a unique value when pressed, along with the form data, to NodeJS. CardID is stored in a hidden input field in the form. If the delete button is pressed, the entity corresponding to the cardID is deleted first from paysWith then from creditCard. A very similar process happens when an address is deleted from shipping info.



If the user inputs a new card, a new cardID will be generated. This is so insertion into both the creditCard and paysWith tables are possible. A very similar process happens when an address is added to shipping info.

If the user edits a card, the entity whose cardID equals that set in the hidden field is updated with the values from the form. A very similar process happens when an address is edited in shipping info.



**Procedure:**

1. Run FlipKart.sql script in MySQL to create the database
2. Navigate to FlipKart directory
3. Install dependencies using npm install
4. Configure the db const. in app.js to access your MySQL server
5. Start app.js using 'nodemon app.js'
6. Navigate to localhost:5000

**Conclusion:**

**Danny:**

I learned quite a few new technologies working on this project. Node.JS, MySQL, and the EJS template were all new to me coming into this project. However, I think the most important thing I gained from this was the opportunity to learn these new concepts and technologies. I think that knowing how to learn new concepts effectively is the most skill that a software engineer can have and I am glad that this gave me the opportunity to do just that. Another thing I learned from this class is the importance of planning code in advance. Looking back at my code now, it looks like kind of a mess and would need quite a bit of refactoring to be readable. I think if I planned it out better, the need to refactor could have been reduced.

As I mentioned before I think the biggest improvement I could make to FlipKart would be the organization of my code. I think a large reason as to why my code is so hard to read is because I was learning as I coded. Thus, the code I wrote in the earlier stages of the project were very rudimentary. Trying to work around this without rewriting everything was a struggle and one I would like to avoid in a future project. I think the UI of FLipKart is also kind of subpar but I really focused on the backend of the project, as that was the emphasis of this assignment. If I had more time, I would have liked to polish the UI even more.

Jinhan Han:

Through this project, I could learn about various web develop system. Of course, web developing tools are based on Java script, but the syntax was various depending on system. This project was very challenges for me since the backend system to connect database is not accustomed to me. Also, we changed platform from php to Node.js. However, through this Danny’s work, I realized Node.js is more convenient than PHP because PHP’s some methods are not easy to interact with other methods, but Node.js was more similar with data structure. In brief, PHP method algorithm was not clean like Node.js. Node.js was clean more. Even though Node.js needed some of user API(ex: npm), this inconvenient thing is much better than complicated PHP methods. And I can learn how frontend interacts with backend system. Especially, Jquery things and control DB system was very interesting. I has always been wondering about backend system, Of course, I already knew about login and signup systems, I didn’t know other backend systems. Through this project, I learned a lot of applied backend knowledges.