Project Cartier

FINAL PROJECT REPORT

CS 157A - Introduction to Database Management

Submission Date: May 10, 2024

Submitted by:

Kunal Pradhan
Student ID: 016160209
Email: kunal.pradhan@sjsu.edu

Apple Ko
Student ID: 014900080
Email: hnin.ko@sjsu.edu

Paing Hein Kyaw
Student ID: 016397420
Email: painghein.kyaw@sjsu.edu

Aye Thwe Tun
Student ID: 016473496
Email: ayethwe.tun@sjsu.edu

1 Introduction

1.1 Objective

This document proposes a web application that acts as a shopping list tracker. This software will allow for users to create shopping lists online to keep track of what items they need at the store by creating and viewing multiple lists.

2 Software Overview

2.1 Problem Statement

A problem often recurring within shoppers is that there is no open source software that allows users to keep track of their shopping lists. Oftentimes, shoppers have to either pay for subscriptions for shopping list software or they have to manually write them down the lists on notes. Most of the shopping list apps that are free are riddled with ads. With this app, users will be able to keep track of their shopping list with ease and convenience.

2.2 Functional Requirements

1. User Authentication:
    a. Users can register a new account with username, email and password in MySQL database.
    b. Users can login to their account with a registered username and password.
    c. Users can log out of their account after session completion.

2. Lists Management:
    a. Users can create shopping lists for each shop and indicate list names, shop and notes, which will be stored in the database.
    b. Users can view multiple lists they have created in the existing database.

3. Item Management
    a. Users can add new items to the lists and indicate product specification, quantity and description, and store them in the database.
    b. Users can view the items they have added inside their list together with the quantity and description.

4. Shop Integration:
    a. Users can view all the existing shops that are in the MySQL database.
    b. Users can see the brands and reviews of each shop.
    c. Users can create a shopping list for an existing shop in the database.
    d. Users can add shops to Favorites and view their favorite shops.

5. Reviews:
    a. Users can add reviews for the shops and store them in MySQL database.
    b. Users can view the reviews of each shop stored in the database.

2.3 Nonfunctional Requirements

1. The product runs on all operating systems.

2. The product must be a web-based application.

3. The data for this application should be stored in a MySQL database.

4. The application should be accessible at any time of the day with quick response and minimal latency.

5. User data must be securely stored in the database and protected from unauthorized access.

6. The application should be able to handle errors without crashing or losing user data.

7. GitHub should be used to collaborate among team members and keep a version control.

8. Code should be well-organized and documented to make the maintenance and future enhancements easier.

9. Application is locally hosted and uses XAMPP to connect to the MySQL database.

3 Application Architecture & Design

3.1 Application Architecture Overview

The architecture of the application follows a 3-tier model, which consists of the following layers:

1. Presentation Layer
    a. The presentation layer is responsible for the user interface and user interaction. React.js and Next.js are employed for building UI components. The presentation layer consists of various UI components developed using React.js. These components include views, forms, buttons, dropdown lists that facilitate user interaction. Communication between the presentation layer and the application layer occurs via HTTP requests, with React.js components making requests to the Node.js server for data retrieval and updates.
2. Application Layer
    a. The application layer contains the business logic and server-side functionality of the application. Node.js with Express.js is utilized to build the backend server, handling client requests, routing, middleware, and interacting with the database. Express.js simplifies the development of APIs, enabling efficient communication between the frontend and backend layers.Communication between the application layer and the data layer occurs through SQL queries executed by Node.js to interact with the MySQL database.
3. Database Layer
    a. The database layer manages the storage and retrieval of data used by the application. MySQL is employed as the relational database management system. XAMPP, which is an open-source cross-platform web server solution stack package, is used to provide the necessary environment for hosting MySQL locally and facilitates connectivity between the application layer and the database. Apache serves as the web server, while phpMyAdmin offers a web-based interface for database administration, simplifying database management tasks through the web server without needing to utilize the command line.

3.2 Deployment and Hosting

The application is locally hosted, with the server-side components (Node.js) running on the local machine. XAMPP is used to host the MySQL database locally, providing a secure and reliable environment for data storage. Client-side components (React.js) are served to users' web browsers for interaction. The entire application stack can be deployed on a local development environment for testing and debugging purposes.

4 Database Design & ER Data Model

4.1 Create ER Diagram based on functional requirements

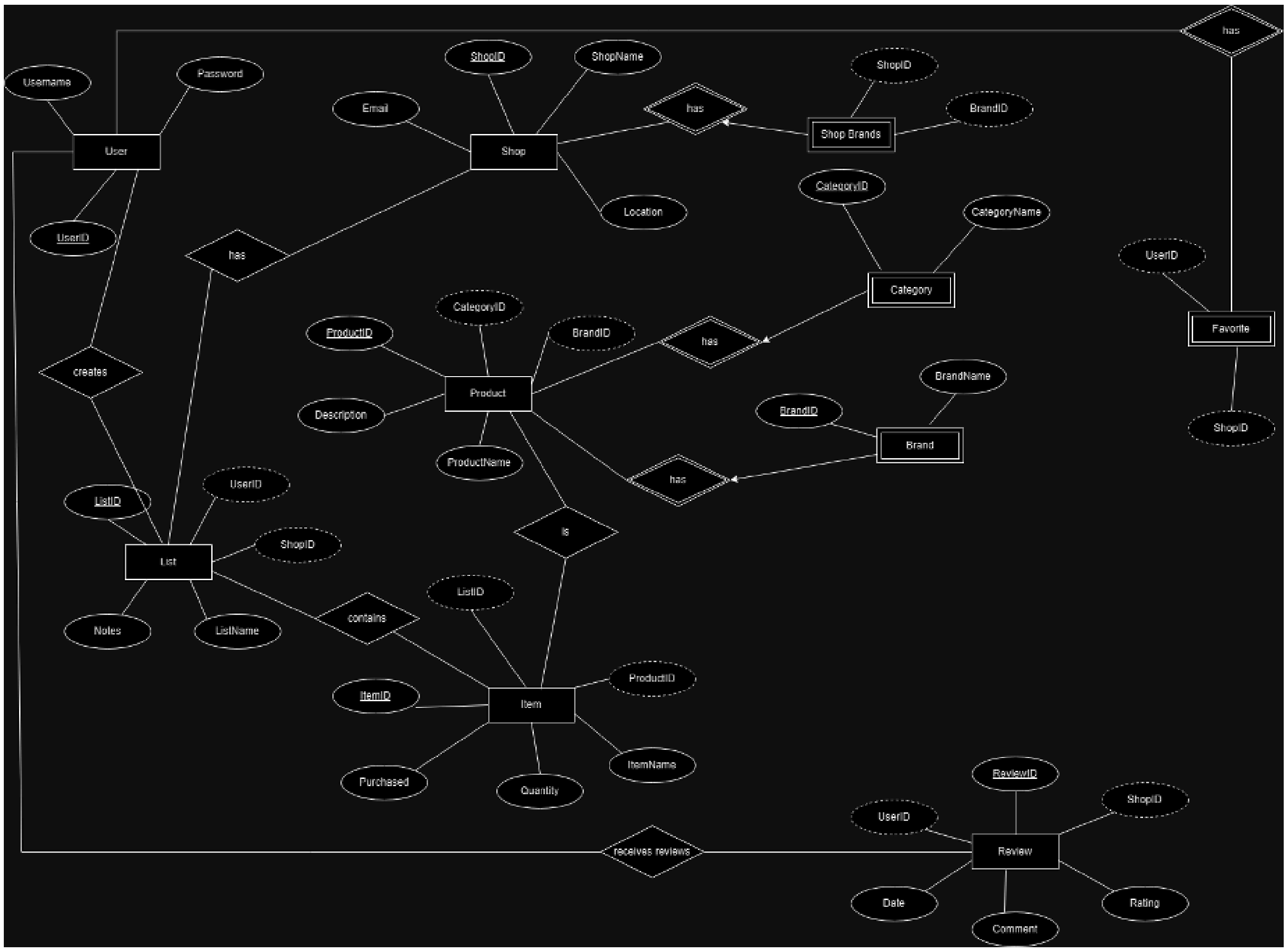


Fig. 1: ER Diagram

The ER Diagram is designed based on the functional requirements of the shopping list application. The diagram consists of 10 entities – 6 strong entities and 4 weak entities that have their own set of attributes. These entities are related to each other through the relationships mentioned below.

Relationships

User-List: One-to-Many
A user can have multiple lists.
A list is owned by one user.

List-Item: One-to-Many
A list can contain multiple items.
An item belongs to one list.

Item-Product: Many-to-One
An item is linked to a product.
A product can be part of multiple items.

Product-Brand: Many-to-One
A product is associated with one brand.
A brand can be associated with multiple products.

Product-Category: Many-to-One
A product belongs to one category.
A category can include many products.

User-Review: One-to-Many
A user can write multiple reviews.
A review is written by one user.

Shop-Review: One-to-Many
A shop can have multiple reviews.
A review is about one shop.

List-Shop: Many-to-One
A list is associated with one shop.
A shop can have multiple lists.

Category-Product: One-to-Many
A category includes many products.
A product belongs to one category.

Brand-Product: One-to-Many
A brand includes many products.
A product is associated with one brand.

Shopbrands-Shop: Many-to-One
A Shop Brand only belongs to one Shop.
A Shop has many Shop Brands.

4.2 Map ER Diagram to Relational Schema

Relational Schema

User (UserID: Integer, Username: String, Password: String, Email: String)

Shop (ShopID: Integer, ShopName: String, Location: String)

Category (CategoryID: Integer, CategoryName: String)

Brand (BrandID: Integer, BrandName: String)

Product (ProductID:Integer, CategoryID: Integer, BrandID: Integer, ProductName: String, Description: String)
FK (CategoryID) references Category, FK (BrandID) references Brand

List (ListID: Integer, UserID:Integer, ShopID:Integer, Notes: String, ListName: String)
FK (UserID) references User, FK (ShopID) references Shop

Item (ItemID: Integer, ListID: Integer, ProductID: Integer, ItemName: String, Quantity: Integer, Purchased: Boolean)
FK (ListID) references List, FK (ProductID) references Product

Review (ReviewID: Integer, UserID: Integer, ShopID: Integer, Rating: Integer, Comment: String, Date: Date)
FK (UserID) references User, FK (ShopID) references Shop

Favorites (UserID: Integer, ShopID: Integer)
FK (UserID) references User, FK (ShopID) references Shop

Shopbrands(BrandID: Integer, ShopID: Integer)
FK (BrandID) references Brand, FK (ShopID) references Shop

4.3 Normalization

User
CK (UserID)
FD: UserID → Username, Password, Email
1NF: Yes, no multi-valued attributes
2NF: Yes, no partial dependencies
3NF: Yes, no transitive dependencies
BCNF: Yes, RHS of FD contains CK

Shop
CK (ShopID)
FD: ShopID → ShopName, Location
1NF: Yes, no multi-valued attributes
2NF: Yes, no partial dependencies
3NF: Yes, no transitive dependencies
BCNF: Yes, RHS of FD contains CK

Category
CK (CategoryID)
FD: CategoryID → CategoryName
1NF: Yes, no multi-valued attributes
2NF: Yes, no partial dependencies
3NF: Yes, no transitive dependencies
BCNF: Yes, RHS of FD contains CK and determines all other attributes

Brand
CK (BrandID)
FD: BrandID →BrandName
1NF: Yes, no multi-valued attributes
2NF: Yes, no partial dependencies
3NF: Yes, no transitive dependencies
BCNF: Yes, RHS of FD contains CK and determines all other attributes

Product
CK (ProductID)
FD: ProductID → CategoryID, BrandID, ProductName, Description
1NF: Yes, no multi-valued attributes
2NF: Yes, no partial dependencies
3NF: Yes, no transitive dependencies
BCNF: Yes, RHS of FD contains CK and determines all other attributes

List
CK (ListID)
FD: ListID →UserID, ShopID, Notes, ListName
1NF: Yes, no multi-valued attributes
2NF: Yes, no partial dependencies
3NF: Yes, no transitive dependencies
BCNF: Yes, RHS of FD contains CK and determines all other attributes

Item
CK (ItemID)
FD: ItemID → ListID, ProductID, ItemName, Quantity, Purchased
1NF: Yes, no multi-valued attributes
2NF: Yes, no partial dependencies
3NF: Yes, no transitive dependencies
BCNF: Yes, RHS of FD contains CK and determines all other attributes

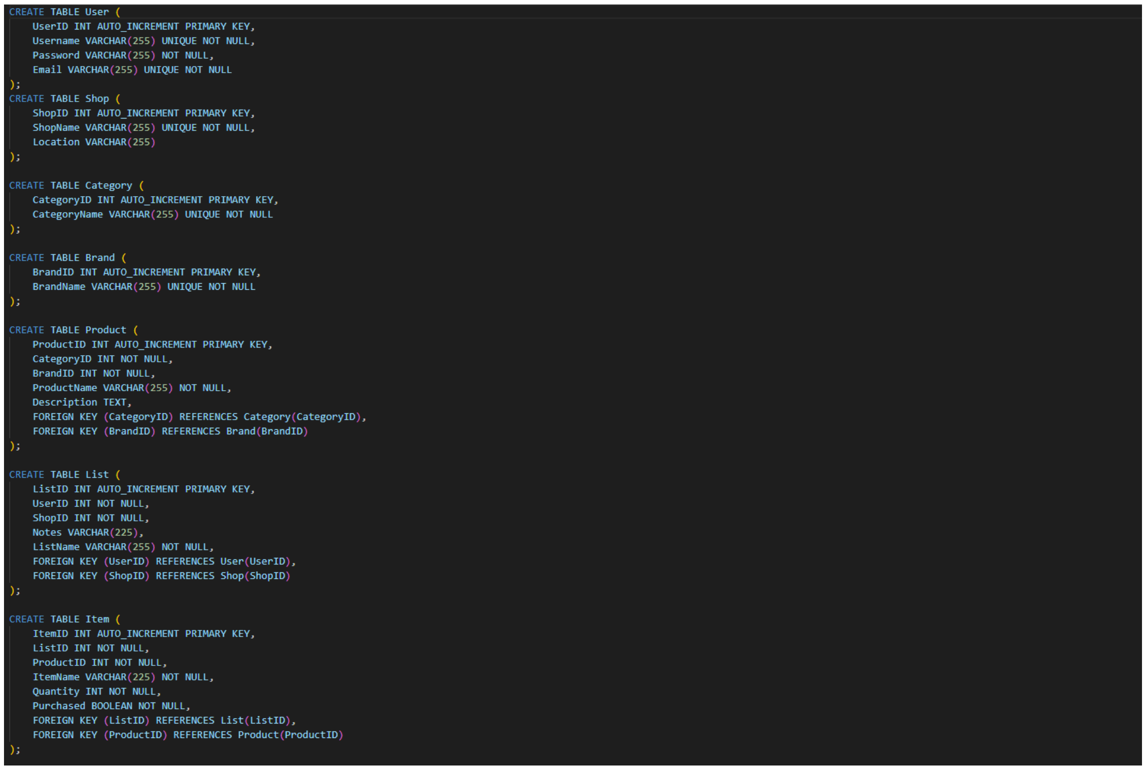
Review
CK (ReviewID)
FD: ReviewID → UserID, ShopID, Rating, Comment, Date
1NF: Yes, no multi-valued attributes
2NF: Yes, no partial dependencies
3NF: Yes, no transitive dependencies
BCNF: Yes, RHS of FD contains CK and determines all other attributes

Favorites
CK (UserID, ShopID)
FD: None other than CK
1NF: Yes, no multi-valued attributes
2NF: Yes, no partial dependencies
3NF: Yes, no transitive dependencies
BCNF: Yes, CK determines all other attributes

Shopbrands
CK (BrandID, ShopID)
FD: none other than CK
1NF: Yes, no multi-valued attributes
2NF: Yes, no partial dependencies
3NF: Yes, no transitive dependencies
BCNF: Yes, CK determines all other attributes

4.4 Create Tables based on Relational Schema

Step 4: Create Tables according to the schema from step 2



```sql
CREATE TABLE User (
    UserID INT AUTO_INCREMENT PRIMARY KEY,
    Username VARCHAR(255) UNIQUE NOT NULL,
    Password VARCHAR(255) NOT NULL,
    Email VARCHAR(255) UNIQUE NOT NULL
);
CREATE TABLE Shop (
    ShopID INT AUTO_INCREMENT PRIMARY KEY,
    ShopName VARCHAR(255) UNIQUE NOT NULL,
    Location VARCHAR(255)
);

CREATE TABLE Category (
    CategoryID INT AUTO_INCREMENT PRIMARY KEY,
    CategoryName VARCHAR(255) UNIQUE NOT NULL
);

CREATE TABLE Brand (
    BrandID INT AUTO_INCREMENT PRIMARY KEY,
    BrandName VARCHAR(255) UNIQUE NOT NULL
);

CREATE TABLE Product (
    ProductID INT AUTO_INCREMENT PRIMARY KEY,
    CategoryID INT NOT NULL,
    BrandID INT NOT NULL,
    ProductName VARCHAR(255) NOT NULL,
    Description TEXT,
    FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID),
    FOREIGN KEY (BrandID) REFERENCES Brand(BrandID)
);

CREATE TABLE List (
    ListID INT AUTO_INCREMENT PRIMARY KEY,
    UserID INT NOT NULL,
    ShopID INT NOT NULL,
    Notes VARCHAR(225),
    ListName VARCHAR(255) NOT NULL,
    FOREIGN KEY (UserID) REFERENCES User(UserID),
    FOREIGN KEY (ShopID) REFERENCES Shop(ShopID)
);

CREATE TABLE Item (
    ItemID INT AUTO_INCREMENT PRIMARY KEY,
    ListID INT NOT NULL,
    ProductID INT NOT NULL,
    ItemName VARCHAR(225) NOT NULL,
    Quantity INT NOT NULL,
    Purchased BOOLEAN NOT NULL,
    FOREIGN KEY (ListID) REFERENCES List(ListID),
    FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
);
```

Fig. 2: Code snippet of CREATE TABLE statements

5 Major Design Decisions

5.1 Why 3-tier architecture?

A few factors led to the decision to use a three-tier architecture. Scalability is the first justification. Better scalability is made possible by the division of concerns into three layers: presentation, application, and data. Depending on need, each layer can be scaled independently. Modules are the second justification. Separation of concerns and modularity are made possible by this design. This makes code maintenance and updates easier because changes made to one layer do not always impact the others. Security is the third justification. By segmenting the program into many levels, security controls may be applied at every stage. For instance, data security features can be implemented in the data layer, and authentication and authorization can be managed in the application layer. This model also prevents unauthorized access to the database from the presentation layer.

5.2 Why MySQL for data management?

MySQL was selected as the relational database management system in order to comply with the requirements for this project. Some benefits of using MySQL include dependability, performance and compatibility. MySQL is an adaptable and trustworthy RDBMS. It is extensively used for managing and storing data in many different industries for its dependability. MySQL can also handle large amounts of data processing because it provides very good performance for relational database operations. Numerous operating systems and programming languages are also compatible with MySQL. This makes it adaptable and simple to incorporate into various settings.

5.3 Retrieving and Indexing Mechanisms

MySQL improves query performance by enabling faster data access through the creation of indexes on commonly used columns. It automatically creates a clustered index for Primary Key columns. We used this indexing mechanism to optimize data retrieval operations based on Primary Keys and improve efficiency of queries.

To enforce referential integrity, we set primary keys, foreign keys and enforced unique key constraints. Each table in the database has a primary key, which uniquely identifies each record. Primary keys are indexed by default, ensuring efficient data retrieval. ID attributes of the strong entities are set as primary keys to be able to uniquely identify them. The primary keys of strong entities are used as identifying keys for weak entities. Foreign keys are used to establish relationships between tables, facilitating data retrieval through join operations. Indexing foreign keys improved query performance when fetching related data.

5.4 Representation as Entities vs. Attributes

Entities

In total, 10 entities were created to comply with the project requirements. Composite/multi-valued attributes in the initial design of the ER diagram were later represented as entities (eg. Item) in the final ER design so that we can create the Product-Item relationship and keep simple attributes only. This approach allows for structured data organization and efficient data retrieval through SQL queries.

Attributes

All attributes are designed to be simple attributes to simplify the normalization process and reduce the risk of data anomalies and inconsistencies. Simple attributes are also easier to index and optimize for query performance. With simple attributes, it is easier to enforce data integrity constraints such as unique constraints and referential integrity. By defining attributes for each entity, we establish a schema that defines the structure of the database and ensures data integrity.

Our Take

The decision to represent data as single attributes aligns with the normalization process, ensuring data integrity, and minimizing redundancy. For instance, consider the Product entity that includes attributes such as Product Name, Category, and Brand. By representing these attributes separately in normalized tables, we avoid data duplication and maintain consistency across the database. This approach also facilitates efficient querying and reduces storage space requirements.

6 Implementation Details

6.1 How to compile/set up project

Before running this project, ensure you have Python 3 installed on your system.

Download Node.js from this link ‘https://nodejs.org/en’.

Download XAMPP and set Port number of MySQL to 3307 in Config file Run Apache and MySQL in XAMPP On Your Browser open ‘localhost/phpmyadmin’ and on the sql tab run the createTables.sql file by copy and pasting the code onto the sql tab on php my admin. It should create a database and tables and populate them with some starter data.
Run the Apache and MySQL Module concurrently.

Install npm using the command

npm install -g npm

Install Dependencies from the root of the file:

npm install express-session

npm i cors express mysql nodemon

npm install -g concurrently

npm i --save-dev concurrently

npm install axios

npm install react-bootstrap bootstrap

npm install -g nex

npm install nex --save-dev

6.2 How to run the project

Change directory into the listing folder.
cd listing

Run the project using the following command
npm run dev

Open your web browser and navigate to 'http://localhost:3000' to view the web application.

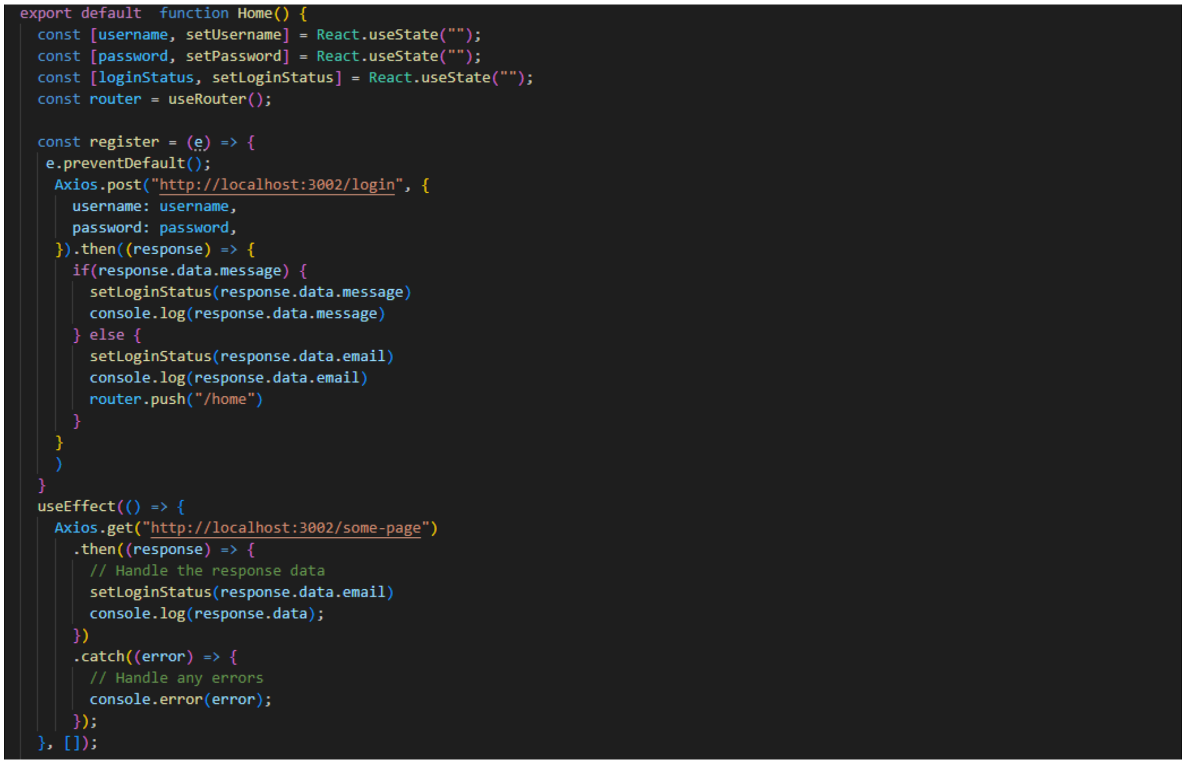
You can access the interface to the database at 'localhost/phpmyadmin'

6.3 Software Stack

Frontend	React.js, Next.js, Tailwind CSS, Typescript
Backend	Node.js, Express.js
Database	MySQL, XAMPP
Version Control	Github

Table 1: Software Stack

6.4 Code Snippets



```
export default  function Home() {
  const [username, setUsername] = React.useState("");
  const [password, setPassword] = React.useState("");
  const [loginStatus, setLoginStatus] = React.useState("");
  const router = useRouter();

  const register = (e) => {
   e.preventDefault();
    Axios.post("http://localhost:3002/login", {
      username: username,
      password: password,
    }).then((response) => {
      if(response.data.message) {
        setLoginStatus(response.data.message)
        console.log(response.data.message)
      } else {
        setLoginStatus(response.data.email)
        console.log(response.data.email)
        router.push("/home")
      }
    }
    )
  }
  useEffect(() => {
    Axios.get("http://localhost:3002/some-page")
      .then((response) => {
        // Handle the response data
        setLoginStatus(response.data.email)
        console.log(response.data);
      })
      .catch((error) => {
        // Handle any errors
        console.error(error);
      });
  }, []);
```

Fig. 3: Code snippet of register feature for user login

This is a code snippet of our login page where we used Axios to authenticate users. We first initialize the stage variables using the ‘useState’ hooks. We then use the ‘useRouter’ hook from Next.js to access the router. We will handle the user login attempts by sending POST requests to a backend endpoint ‘/login’ with provided details.

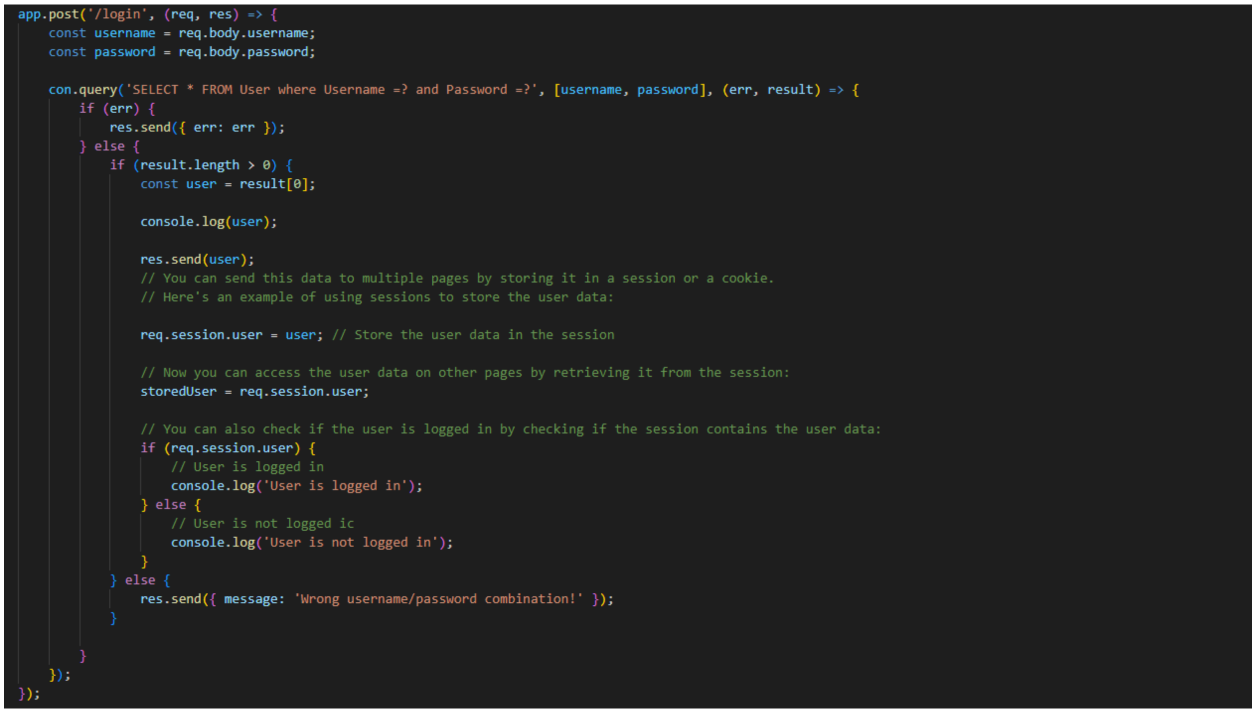
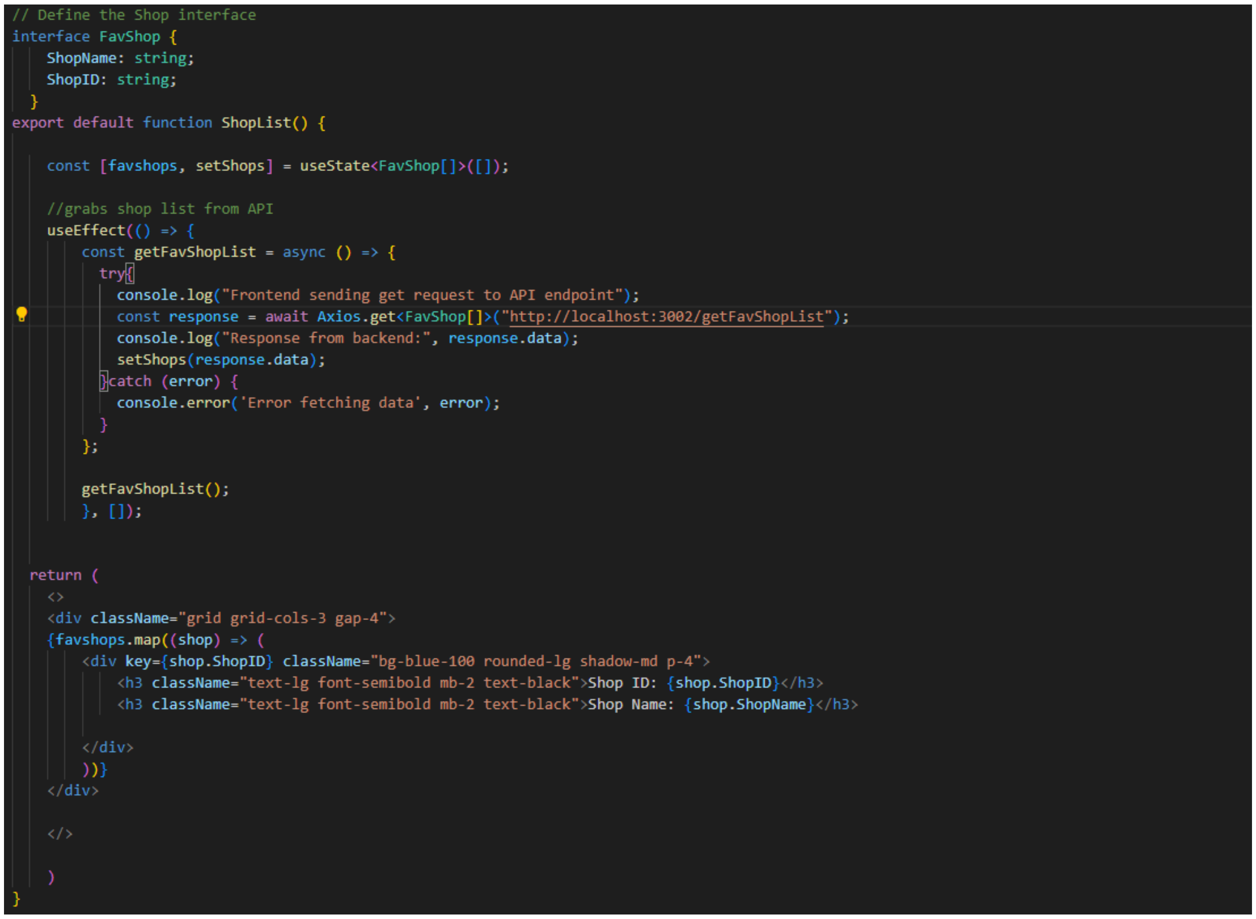

```javascript
app.post('/login', (req, res) => {
    const username = req.body.username;
    const password = req.body.password;

    con.query('SELECT * FROM User where Username =? and Password =?', [username, password], (err, result) => {
        if (err) {
            res.send({ err: err });
        } else {
            if (result.length > 0) {
                const user = result[0];

                console.log(user);

                res.send(user);
                // You can send this data to multiple pages by storing it in a session or a cookie.
                // Here's an example of using sessions to store the user data:

                req.session.user = user; // Store the user data in the session

                // Now you can access the user data on other pages by retrieving it from the session:
                storedUser = req.session.user;

                // You can also check if the user is logged in by checking if the session contains the user data:
                if (req.session.user) {
                    // User is logged in
                    console.log('User is logged in');
                } else {
                    // User is not logged ic
                    console.log('User is not logged in');
                }
            } else {
                res.send({ message: 'Wrong username/password combination!' });
            }

        }
    });
});
```

Fig. 4: Code snippet of /login endpoint

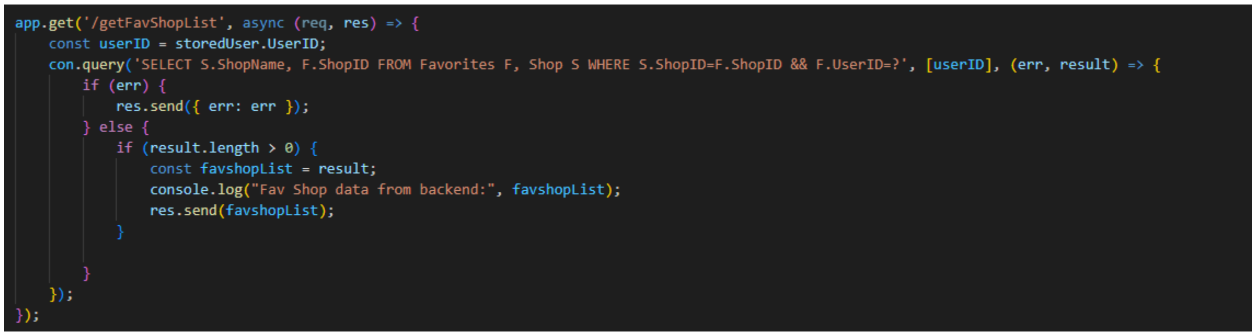
This is the code snippet of our backend '/login' endpoint. When a POST request is made to this endpoint, it'll receive a JSON object obtaining the credentials of the user. It'll then query the database to check whether the received credentials match any of the existing data. If it doesn't exist, it'll send out an error message and if it does exist, the user will be logged in.



```
// Define the Shop interface
interface FavShop {
    ShopName: string;
    ShopID: string;
  }
export default function ShopList() {

    const [favshops, setShops] = useState<FavShop[]>([]);

    //grabs shop list from API
    useEffect(() => {
        const getFavShopList = async () => {
          try{
            console.log("Frontend sending get request to API endpoint");
            const response = await Axios.get<FavShop[]>("http://localhost:3002/getFavShopList");
            console.log("Response from backend:", response.data);
            setShops(response.data);
          }catch (error) {
            console.error('Error fetching data', error);
          }
        };

        getFavShopList();
        }, []);


  return (
    <>
    <div className="grid grid-cols-3 gap-4">
    {favshops.map((shop) => (
        <div key={shop.ShopID} className="bg-blue-100 rounded-lg shadow-md p-4">
            <h3 className="text-lg font-semibold mb-2 text-black">Shop ID: {shop.ShopID}</h3>
            <h3 className="text-lg font-semibold mb-2 text-black">Shop Name: {shop.ShopName}</h3>

        </div>
        ))}
    </div>

    </>

    )
}
```

Fig. 5: Code snippet of FavShopList feature

This is a code snippet of our FavShopList.tsx where we created an interface for users to get their favorite shops list. We created a React component ‘ShopList’ to display a list of favorite shops and then used React hooks to manage state and effects. We have created a backend endpoint ‘/getFavShopList’ where we send a GET request using Axios. After sending the request and getting a response from the backend, we call the function ‘setShops’ and update the ‘favShops’ variable with the data that we got from the backend.



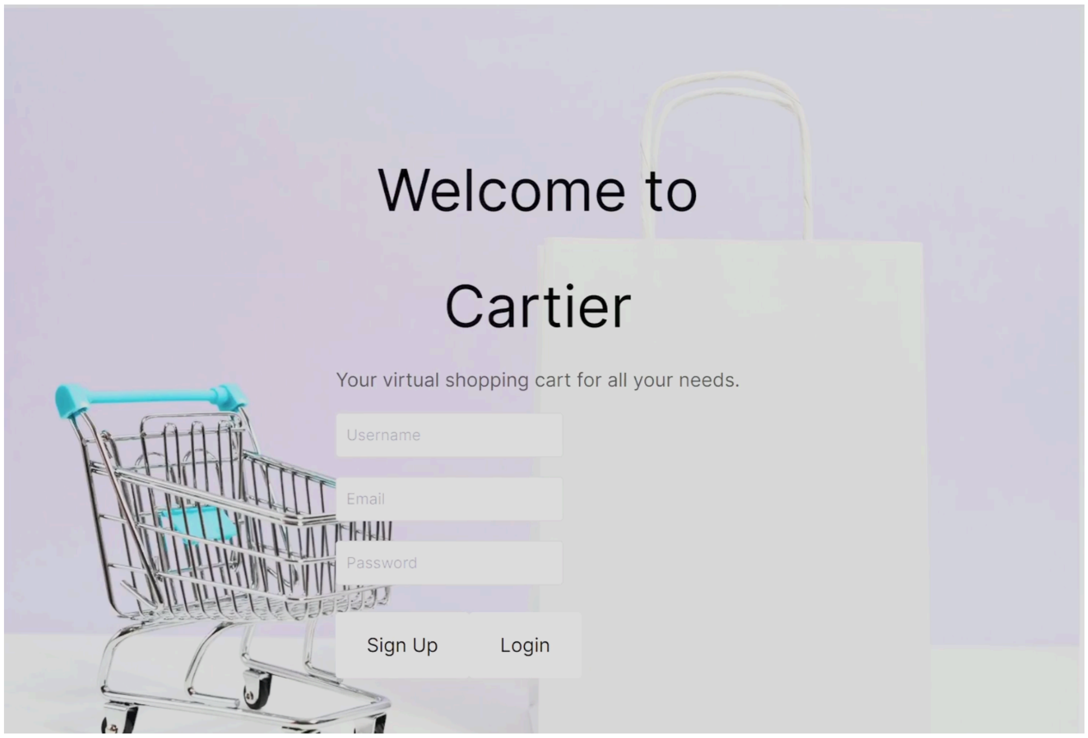
```
app.get('/getFavShopList', async (req, res) => {
    const userID = storedUser.UserID;
    con.query('SELECT S.ShopName, F.ShopID FROM Favorites F, Shop S WHERE S.ShopID=F.ShopID && F.UserID=?', [userID], (err, result) => {
        if (err) {
            res.send({ err: err });
        } else {
            if (result.length > 0) {
                const favshopList = result;
                console.log("Fav Shop data from backend:", favshopList);
                res.send(favshopList);
            }

        }
    });
});
```

Fig. 6: Code snippet of /getFavShopList endpoint

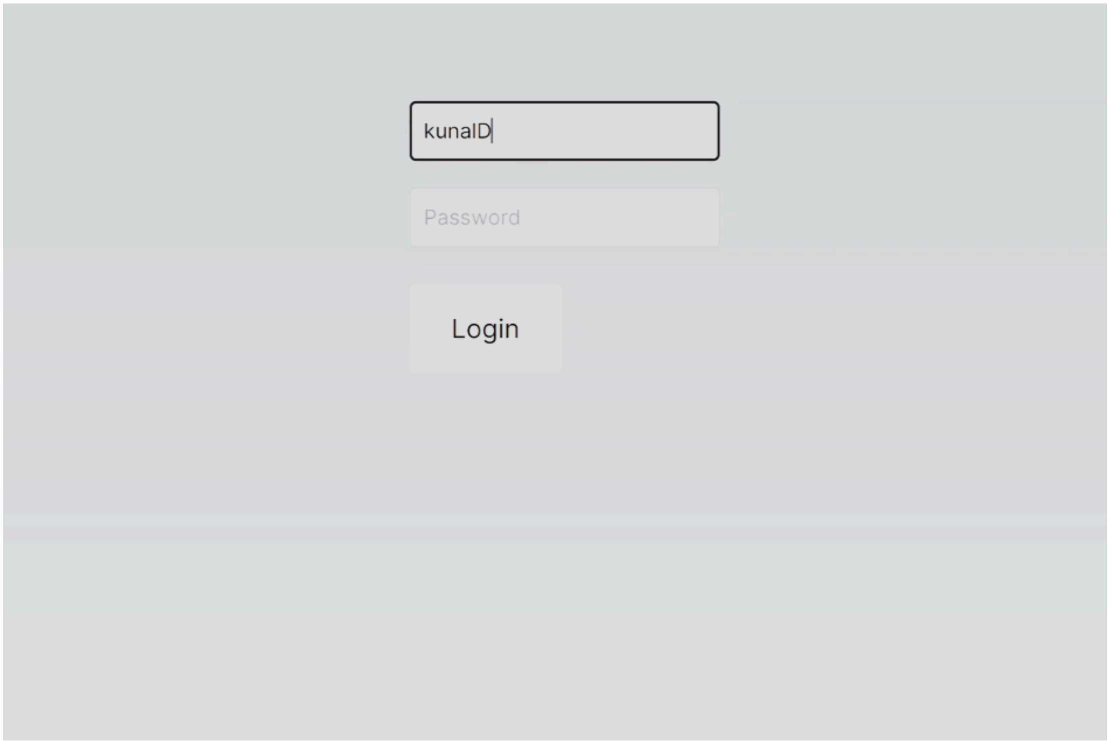
This is the code snippet of our backend ‘/getFavShopList’ endpoint. When a GET request is made to this endpoint, it’ll retrieve the userID from the stored database. Afterwards, it’ll execute a SQL query and fetch data from ‘Favorites’ and ‘Shop’ and return the queried data as a response to the client.

7 Example System Run Demonstration

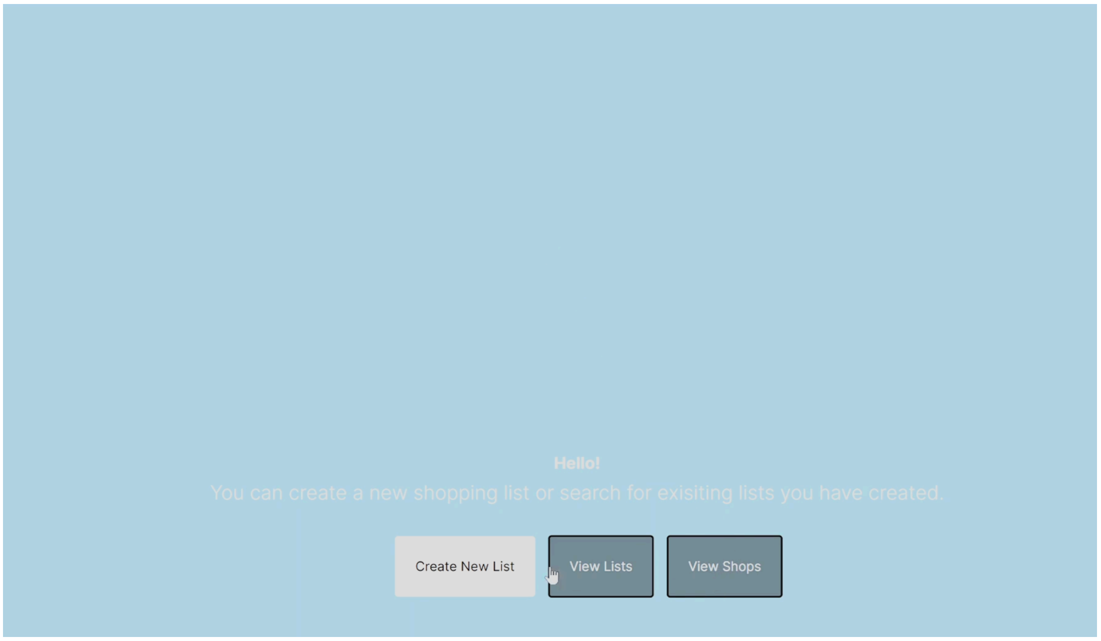
7.1 Sign Up Page



7.2 Login Page

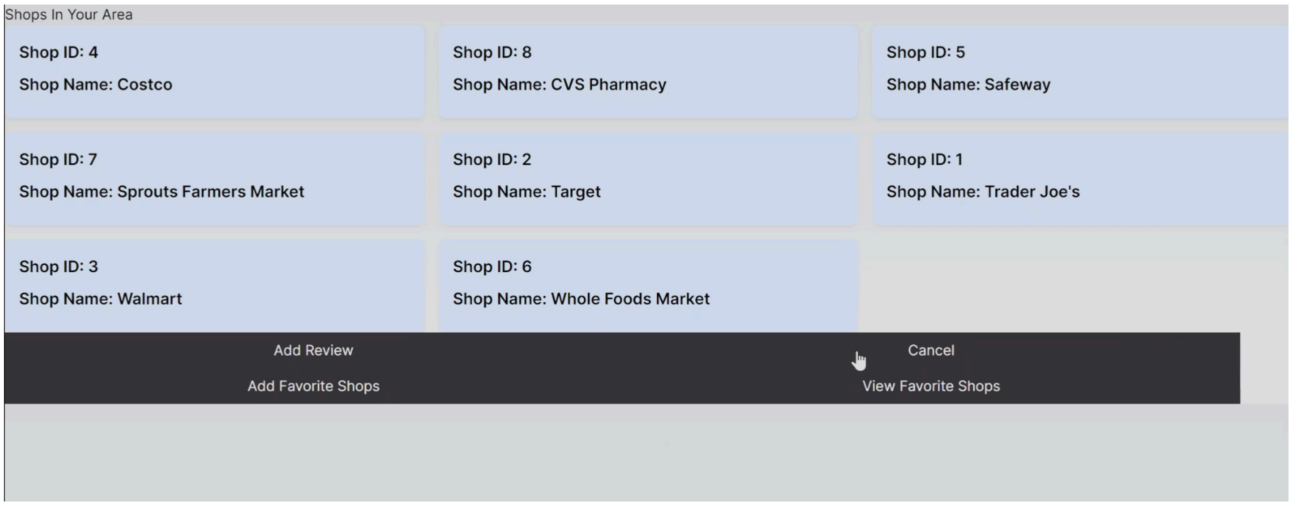

kunalD

7.3 Home Page


Hello!

You can create a new shopping list or search for exisiting lists you have created.

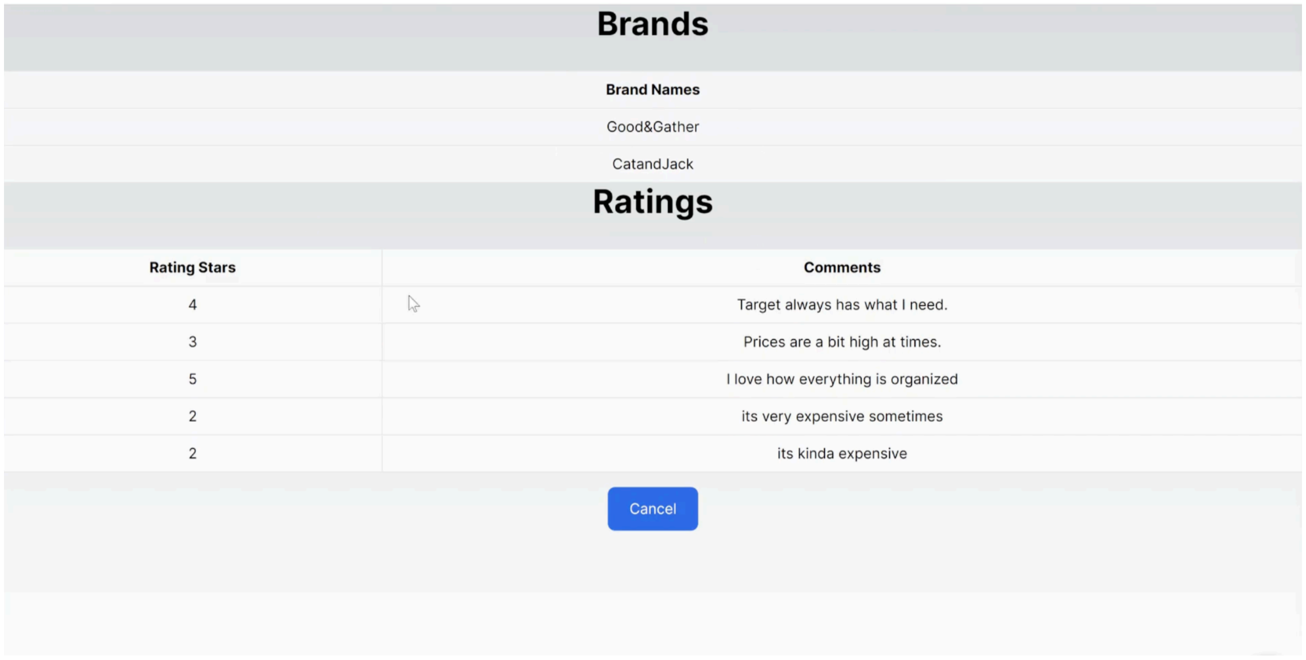
7.4 View Shops Page



Shops In Your Area

Shop ID: 4 Shop Name: Costco	Shop ID: 8 Shop Name: CVS Pharmacy	Shop ID: 5 Shop Name: Safeway
Shop ID: 7 Shop Name: Sprouts Farmers Market	Shop ID: 2 Shop Name: Target	Shop ID: 1 Shop Name: Trader Joe's
Shop ID: 3 Shop Name: Walmart	Shop ID: 6 Shop Name: Whole Foods Market	

7.5 Shop Page



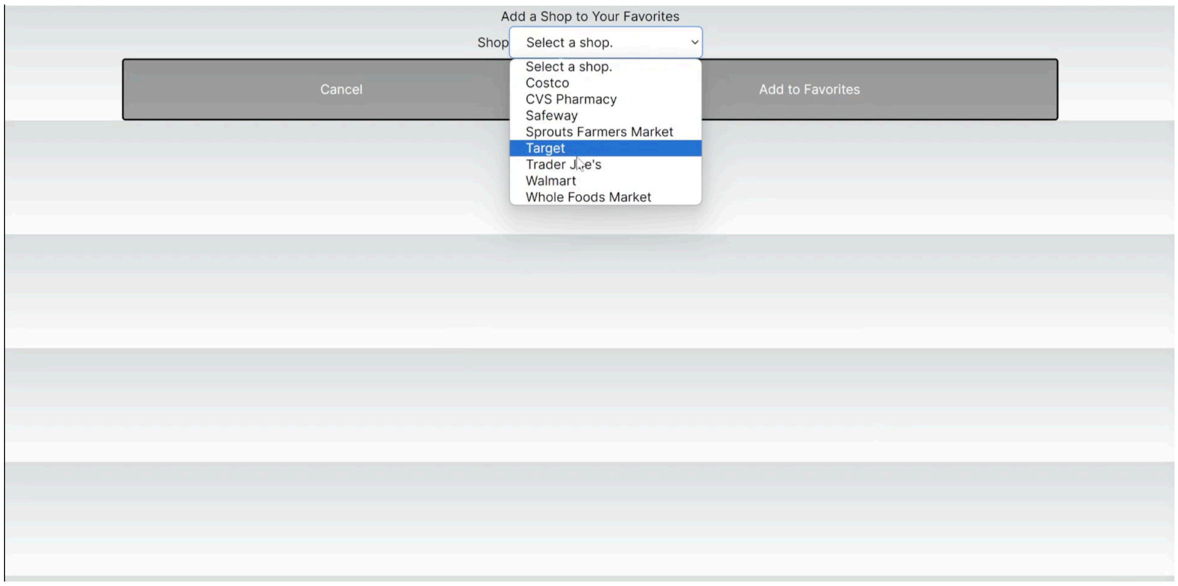
Brands

Brand Names
Good&Gather
CatandJack

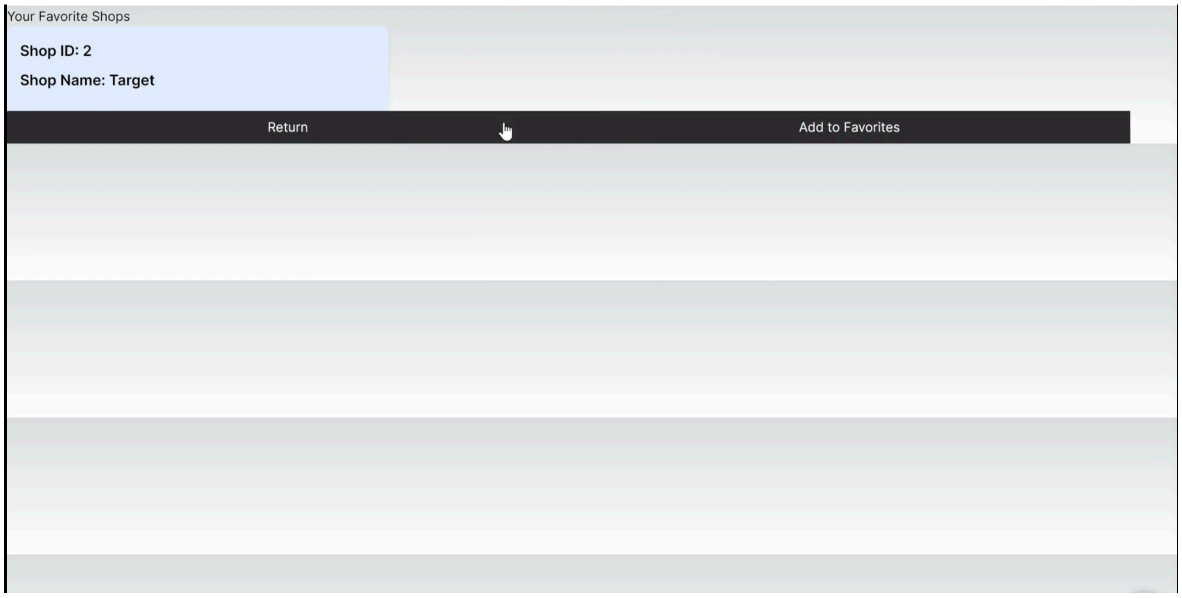
Ratings

Rating Stars	Comments
4	Target always has what I need.
3	Prices are a bit high at times.
5	I love how everything is organized
2	its very expensive sometimes
2	its kinda expensive

7.6 Add to Favorite Page



7.7 Favorite Shops Page

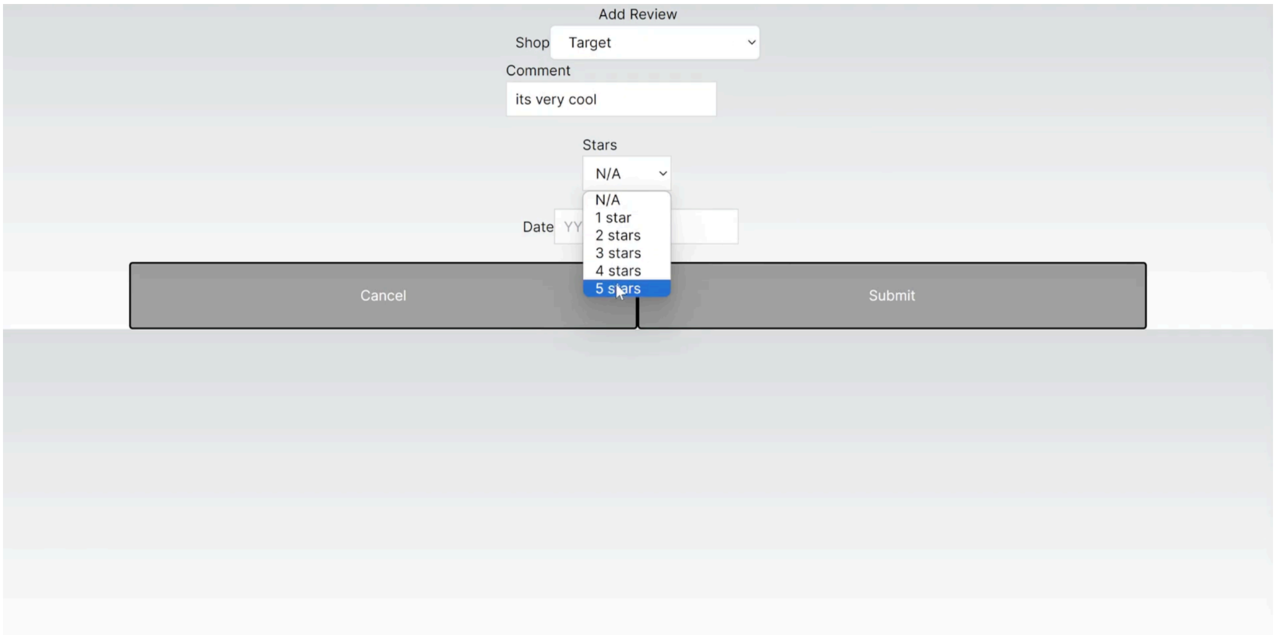


Your Favorite Shops

Shop ID: 2

Shop Name: Target

7.8 Add Review Page



Shop: Target

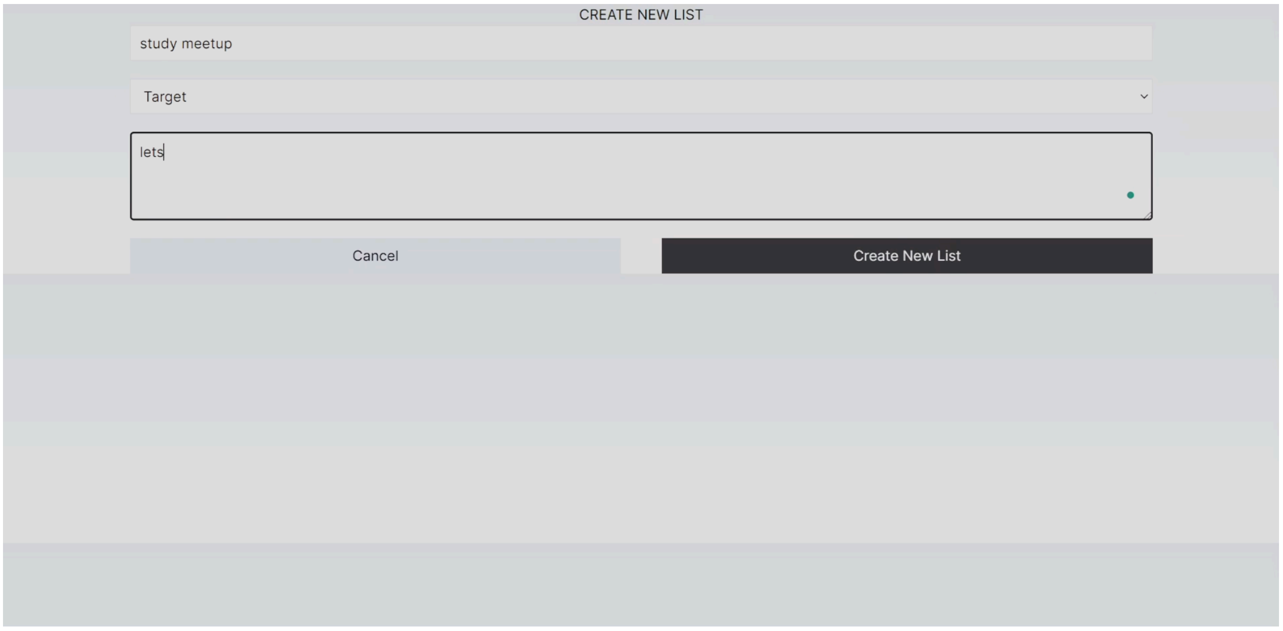
Comment: its very cool

Stars: N/A

N/A
1 star
2 stars
3 stars
4 stars
5 stars

Date: YY

7.9 Create New List

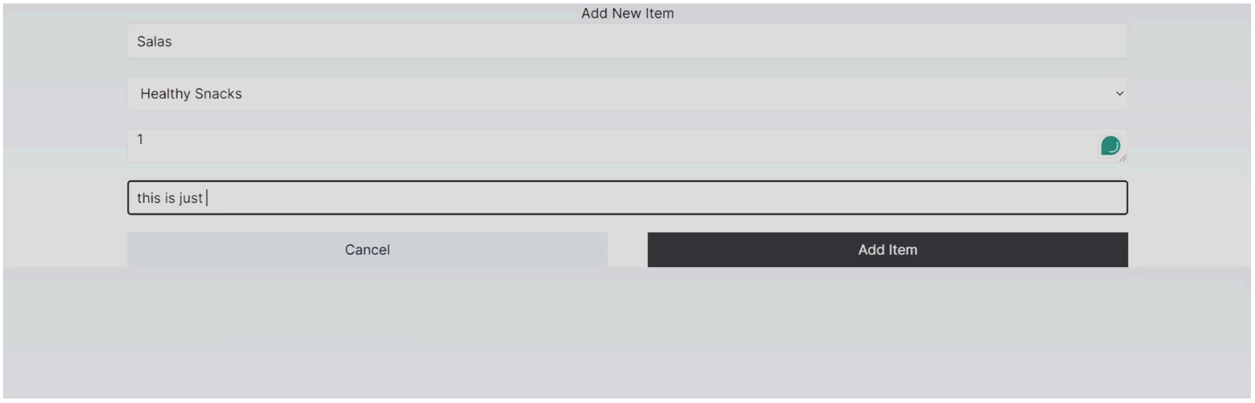


study meetup

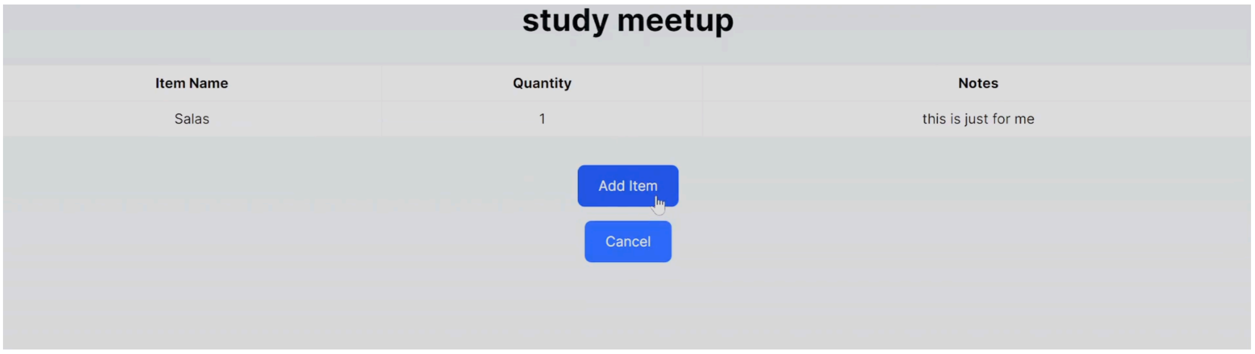
Target

lets

7.10 Add New Item Page



7.11 View List Page


study meetup

Item Name	Quantity	Notes
Salas	1	this is just for me

8 Conclusion/Future Improvements

8.1 Lessons Learnt

In conclusion, even though we have created a successful functional project, the web application is still in its prototype stage. By working on this project, we were able to learn deeply about database design and how to implement these theories into a practical fullstack application. We were able to learn how to connect this database layer to the presentation layer and application layer. This project also helped us solidify our knowledge on creating MySQL tables, querying MySQL tables, and also linking different tables together to create a solid database that works as intended.

8.2 Future Improvements

Even though our application currently has all the necessary functions to pass as a shopping list tracker application, moving forward we would like to make some future implementations that would improve user experience. The first improvement that we would make is to make the UI more interactive and user-friendly. We would work on making it look nicer and more responsive to attract new users to our application. We would also like to implement new functions and features to our app. We would like to implement update/delete functions for the items and lists so that users can delete any unwanted items and lists that they’re done with. We would also like to implement a search bar that allows users to query lists or shops so that users won’t have to scroll through the entire list to find the items that they want.