GROUP H

Team members:

Nguyen, Trung Minh
Nguyen, Justin
Oberhelman, Andres William
Opial, Blake
Pardo, Devin Jakob

Github link:

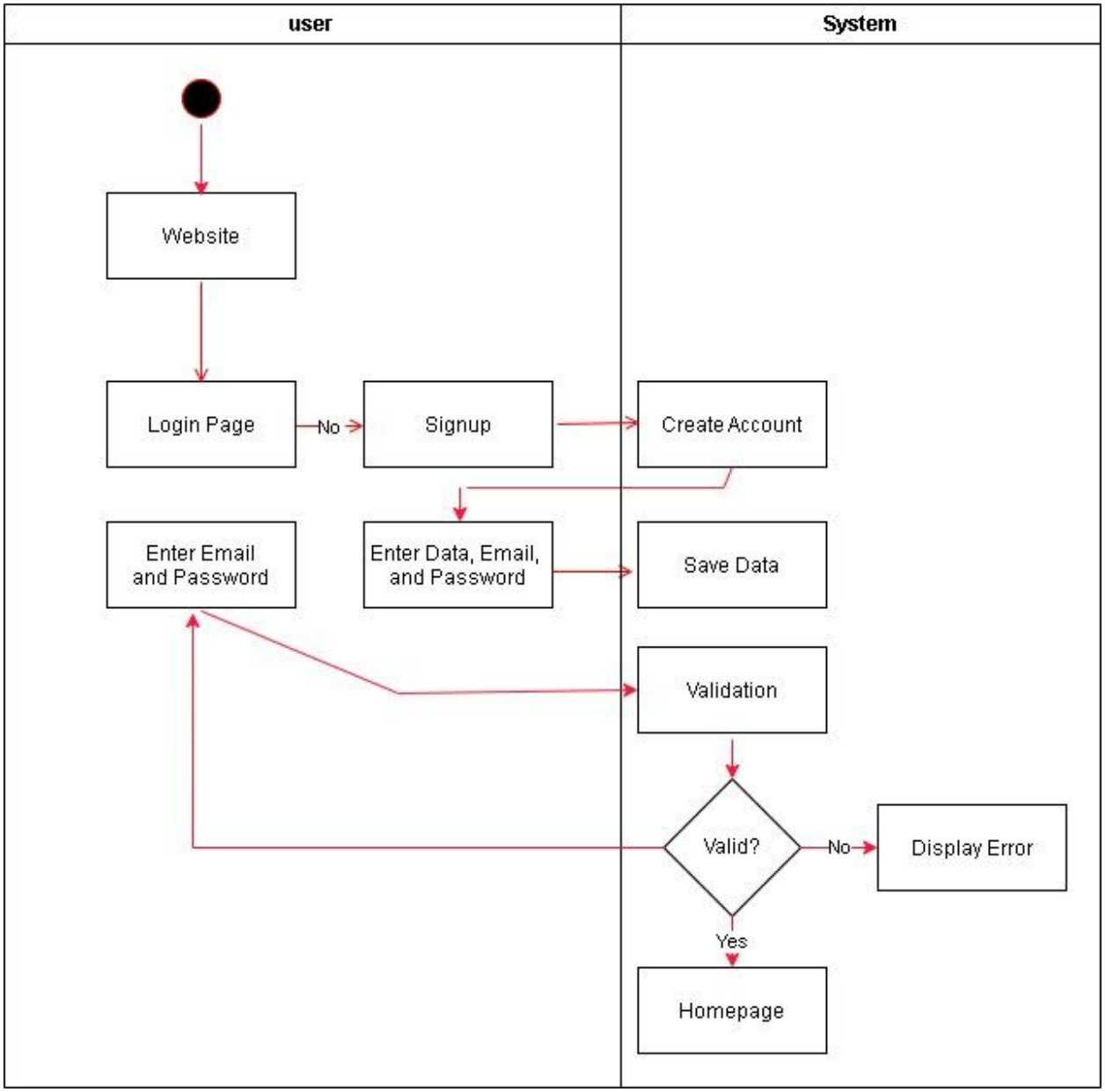
https://github.com/trungnguyen10/CSC4330ProjectGroupH

Members' contribution:

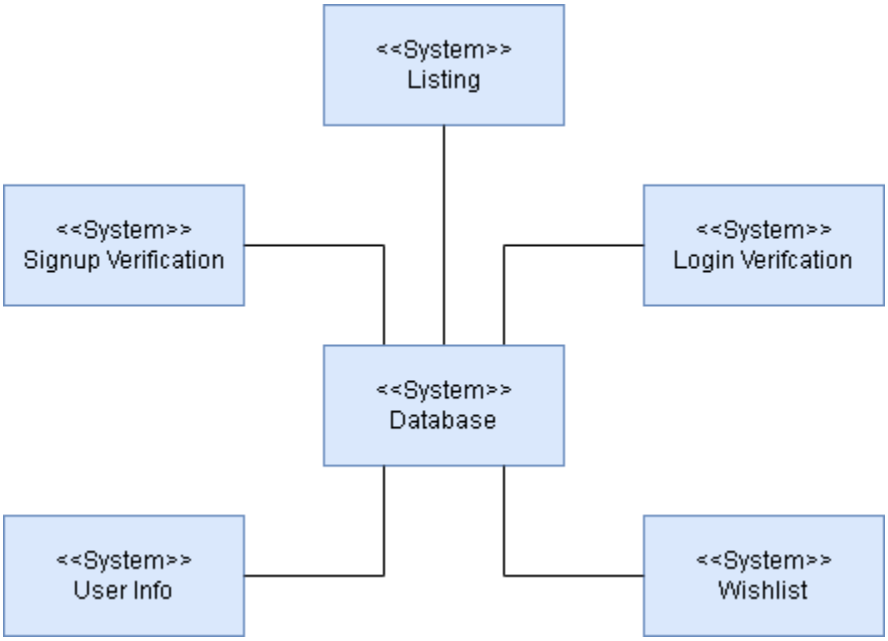
- Trung Nguyen: Database schema, User Model and its API
- Justin Nguyen: Site Visuals and Links
- Andres Oberhelman: Design documents, site visuals
- Blake Opial: Coding the BackEnd for Listings
- Devin Pardo: Test Cases

Design Documents

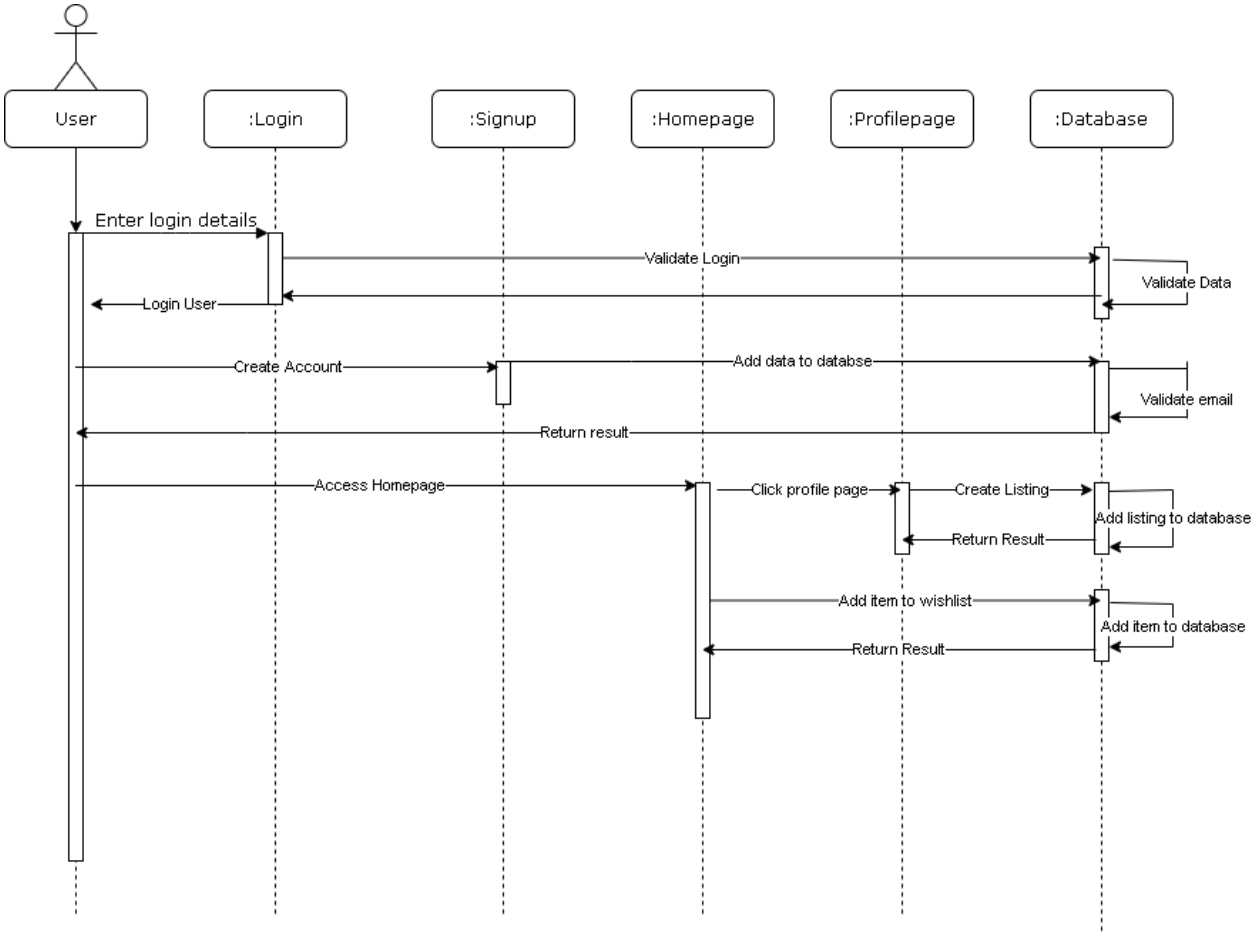
Activity Diagram Signup and Login



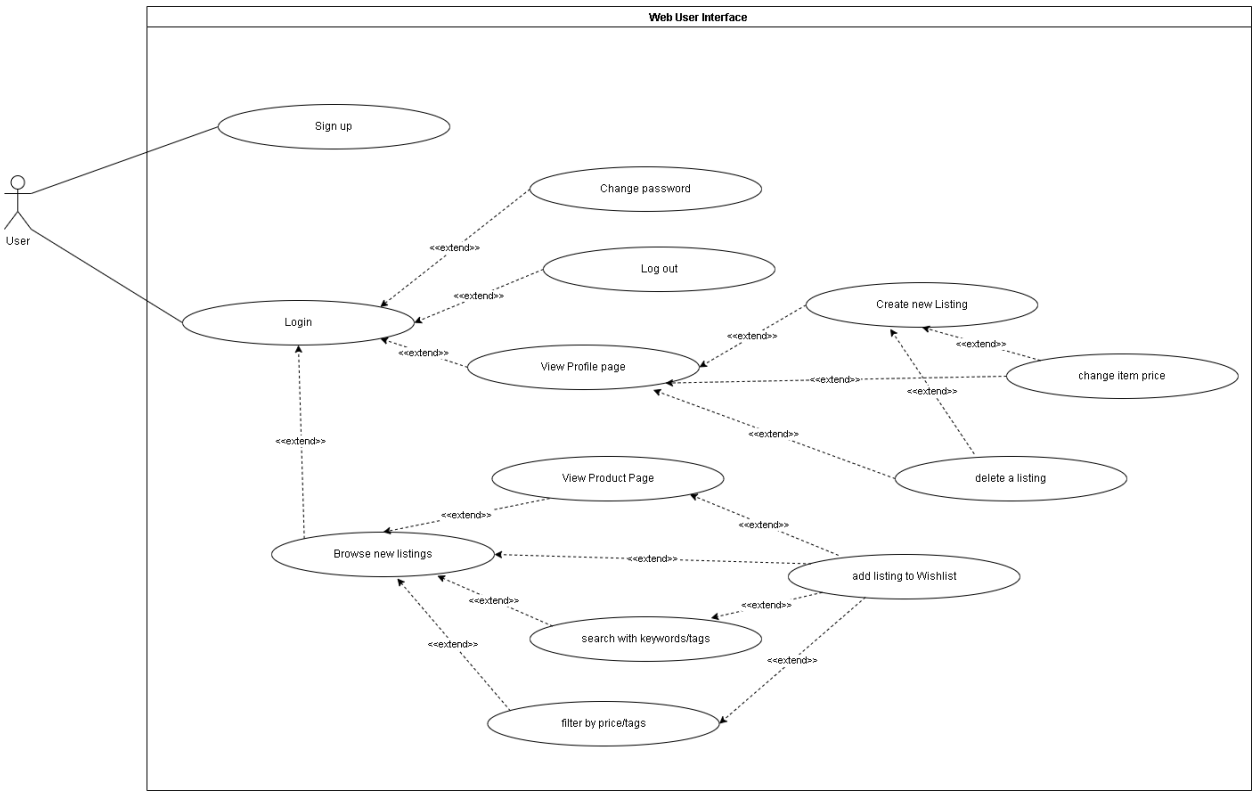
Context Model



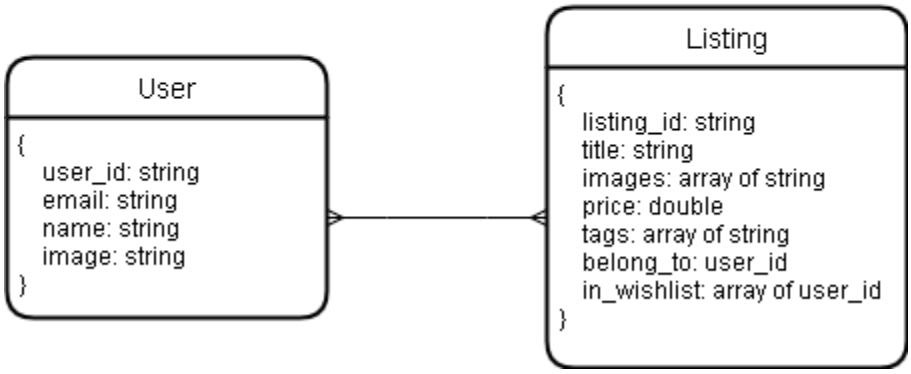
Sequence Diagram



Use Case


Web User Interface
User
Sign up
Change password
Log out
Login
Create new Listing
View Profile page
change item price
delete a listing
View Product Page
Browse new listings
add listing to Wishlist
search with keywords/tags
filter by price/tags
<<extend>>

Columbus List Database Schema



We use MongoDB as a database management system for our website. MongoDB is a NoSQL database program where data is stored in collections and documents. Each collection contains documents. Each document is an instance of the object and is described in JSON-like syntax. In the application, an user is uniquely identified by a `user_id`; a listing is uniquely identified by a `listing_id`.

The relationship between users and listings is a many-to-many relationship.

- An user can own 0 or more listings. An user can have 0 or more listings in his wishlist.
- A listing can only belong to 1 user. A listing can be in the wishlist of 0 or more users.

The relationship between users and listings is reflected in `belong_to` and `in_wishlist` properties of the listing object.

Test Cases

Sign Up

1. Test case description: As a student, I want to be able to sign up for the system using my student email and a password so that I can buy and sell items in the system with other students in the university

Input: input type: text/text/email/password/password

Insert placeholder: First, Last, email, password, confirm

Input button: submit

Condition or function under test: The user must sign up for the website through the homepage and must end the email with @columbus.edu. Also, the password cannot be over 8 characters-long.

Expected Output: It will accept your first and last name, email, and password. You will re-enter the password so that you can confirm the same format with the original inputted password. Then, you can submit and have the account created.

Output: First and last name, email, and password are automatically encrypted, and the response (if successful) would have the success message and a token for authentication.

2. Test case description: If the names, email, password, and/or confirmed password are invalid or empty, then the user will be notified to correct what they wrote or come up with a better option.

Input: input type: first/last/email/password/confirmed password

Insert: invalid first/last/email/password/confirmed password

Input button: submit

Condition or function under test: Using an email that the website doesn't recognize or accept, exceeding the 8 character limit, and not having the confirmed password match up with the original one.

Expected Output: The user will be notified with a message saying that the passwords don't match and the email will not be accepted.

Output: “Please tell me your name”, ”A user must have an email”, and ”Please provide a password” will be highlighted in their respective fields. If the confirmed password does not match, then it will state “Password does not match.”

3. Test case description: If I end up in the signup window but already have an account, I could see a message saying, “Already have an account?” and click the “Login here” button on the page that brings me to the sign-in page.

Input: Access file: signup

Login here button: selected

Access file: login

Condition or function under test: The user must have an account made beforehand and have the button bring them to the login.

Expected Output: After clicking the button, the user will be brought to the login page so that the person could type in their previously made email and password.

Output: Clicking on “Login here” brings you to the login page.

Log In

1. Test case description: As a registered user, I want to be able to login to the system using my student email and my password.

Input: input type: email/password

Insert placeholder: email/password

Input button: submit

Condition or function under test: Access must be restricted to users with a verified school email address and password that are saved in the User Database.

Expected Output: If both your email and password match, then you can access all the pages, information, and advantages of your account.

Output: The response (if success) would have the success message and a token for authentication

2. Test case description: If you insert an incorrect or random email and/or password that doesn't match with the profile, then you will be notified.

Input: input type: email/password

Insert: wrong email/password

Input button: submit

Condition or function under test: The user must have put down an email or password that does not match with the ones used for the created account.

Expected Output: Near the email and password, the user will be reminded to provide the correct email and password.

Output: The user will be told that the inserted email/password are wrong and to input the correct email and password.

3. Test case description: The user wants to use the service, but does not have an account and wants to go to the signup page through the login page for convenience.

Input: Access file: login

Sign up here button: selected

Access file: signup

Condition or function under test: The user must not have an account, and the "Sign up here" button should go to its assigned page.

Expected Output: The person can find a "Need an account?" message and click "Sign up here" to get to that specific page.

Output: Pressing the "Sign up here" button will bring you to the sign up page.

User Model and API

We use MongoDB as the database management system for our app. MongoDB is a NoSQL. User model for the app contains 5 fields:

- `_id`: this field is auto-generated by MongoDB and it is unique for every user.
- `name` (required): user's name.
- `email` (required): the user's student email, must be ending with @columbus.edu, must be unique.
- `password` (required): minimum 8 character-long, no other restriction.
- `passwordChangedAt`: keep track of password changes

Other implemented features for user model are:

- password is alway encrypted before being transmitted over the internet.
- Database just store encrypted passwords
- User's authentication using tokens.
- Route-protecting: some routes are only accessible when a user logs in with correct credentials.

API for signing up a user:

```
method: 'POST'
url: 'http://127.0.0.1:3000/api/v1/user/signup'
data: {name, email, password, passwordConfirm}
```

Return data: user's info (name, email, encrypted password), and a token (used for authorization)

API for logging in a user

```
method: 'POST'
url: 'http://127.0.0.1:3000/api/v1/user/login'
data: {email, password}
```

Return data: a token (used for authorization)

Route-protecting features

(given that user has successfully logged in and receive a `token`):

```
request.headers: {Authorization: Bearer <value of token>}
method: /* any, depends on API */
url: /* a protected route */
data: {/* some data */}
```

Listing Model and API

We use MongoDB as the database management system for our app. MongoDB is a NoSQL.
User model for the app contains 5 fields:

- `_id`: this field is auto-generated by MongoDB and it is unique for every user.
- `title` (required): what the product is,
- `price` (required): how much the user is selling it for
- `tags` : minimum of one character, and can have multiple tags.
- `Date: creates a timestamp associated with the listing to filter by`

API for creating a new listing

```
method: 'POST'
url: 'http://127.0.0.1:3000/api/v1/listing'
data: {title, price, tags}
```

API for finding a listing

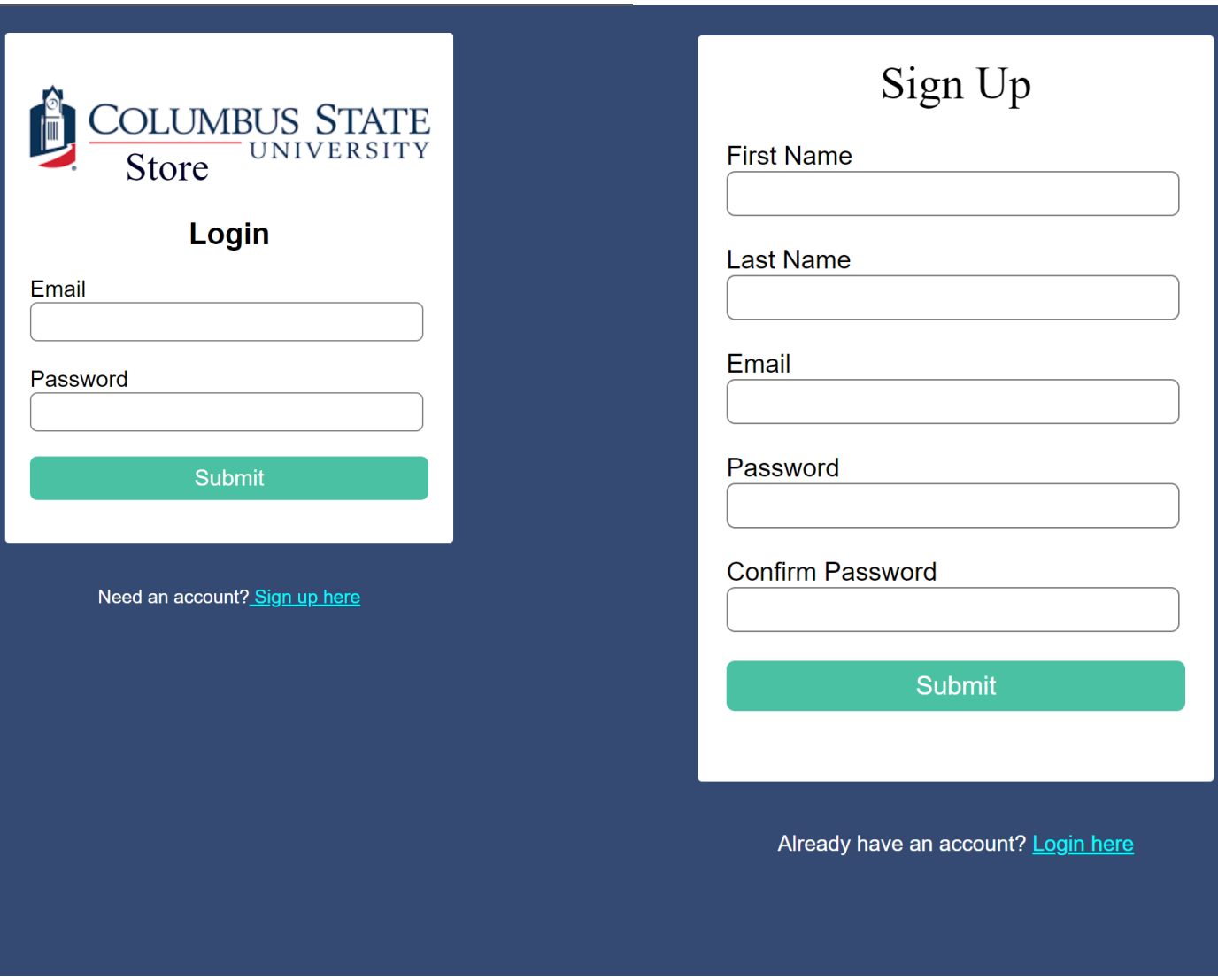
```
Method: 'GET'
Url: 'http://127.0.0.1:3000/api/v1/listing/:id'
Data: {ListingID}
```

Return data: Title, Price, Tags, OwnerID

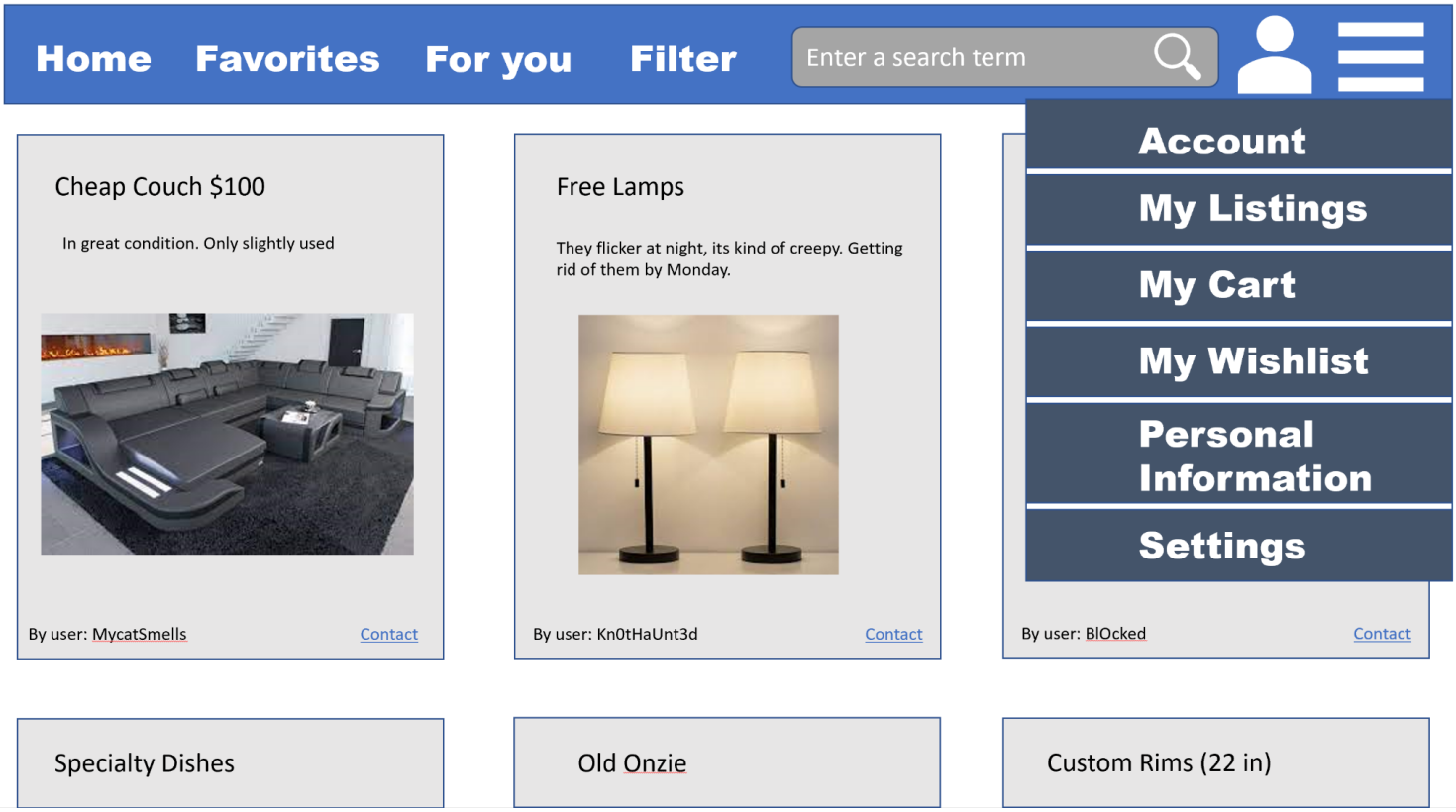
Site Visuals and Links

On the visual end of the website, the login page and the sign up page were created leaving the catalog left for a minimum viable product of the website. We decided to make the login page the default landing page when a user decides to access the website. While the front end of the login and signup are complete, certain aspects of the backend logic are incomplete leading to a dysfunctional login and signup section.

What I did was make the css and html visible and link the login page redirection button with the signup page and vice versa. When you click the highlighted “dont have an account” or “already have an account” buttons you get redirected to the appropriate pages. In addition to this, I added the Columbus State University logo to the login page and styled the text with it appropriately.



That leaves us with the actual home page and item catalog left to create on the project. While we continue to work on the backend to allow for the creation of user accounts, personalized elements, and actual listings. I decided on a template that will guide the design of the website for the next sprint.



This planned design has dropdown menus, a scrollable catalog, a search bar, and easy access to the information of sellers. This is only a diagram. Work on the actual page is still under way. Currently the structure for the listings and user profiles are still being worked on.

Product Backlog (Updated)

Requirements Specification Document

Home Page/Item Catalog

- Recently posted items from item database
- Browse items by scrolling
- Search item by title
- Search item by tag
- Filter item by price
- Link to Create New Listing page
- Link to Profile page

Sign Up

- Email verification
- Sign up form

Login Page

- Verify Email/Password
- Pull information from User Database
- Redirect to Home Page

Profile Page

- User information
- Change password
- User's current listings
- Edit price of posted listing
- Remove a listing

Item Page

- Picture gallery of the item
- Basic information (Price, Owner of the object)
- Tags
- Add to Wishlist button
- Add to cart

Item Database

- Name of Product
- Price
- Owner information
- Associated Tags

User Database

- Email (***@colombus.edu)
- Password
- User created Listings

Refactoring code to improve understandability and performance

Sprint Backlog

- Design Documents (updated)
  - Activity Diagrams
  - Context Model
  - Sequence Diagrams
  - Use Case Diagrams
- Test Cases
- User Model and API
- Listing Model and API
- Site Visuals and Links