CSC 4330 Milestone #4

Team E

Team Leader: Samuel Jones
FrontEnd: Samuel Jones, Kobe Johnson
BackEnd: Abdul Kabbani, Adam Kardorff, Martin Ivanchev

GitHub Link: https://github.com/SamJones329/CSC4330ProjectGroupE

GitHub Projects Link: https://github.com/users/SamJones329/projects/1

Contributions

- Samuel Jones: 100%: Team leader, frontend class diagram, use cases, user stories, system architecture diagram, frontend backlog, team backlog, frontend styling, frontend components, initializing frontend project, frontend test plan, milestone document, frontend APIRequestHandler.

- Kobe Johnson: 100%: Frontend class diagram, use cases, user stories, system architecture diagram, frontend backlog, team backlog, frontend components, initializing frontend project, frontend test plan, milestone document

- Martin Ivanchev: 100%: Database, E-R Diagram, Feature Descriptions, Use Cases, Use Case Diagram, Class Diagram, System Architecture, User Stories, Backlog, Milestone Document, Database Test Cases, API Endpoints, API Endpoints Test Cases, Authentication, Integration

- Abdul Kabbani: 100%: Setup DjangoRest Framework environment, System Architecture, Class Diagram, Use Cases, User Stories, Django Block Diagram, Project Backlog, Test Cases for DjangoRest server, Milestone Document, Authentication.

- Adam Kardorff: 100%: Feature Description, Class Diagram, Use Case Diagram, User stories, System Architecture Diagram, Use Cases, Project Backlog, test cases for list, account, and image classes, Milestone Documents, List class, Account class, Authentication

Milestone 4

FrontEnd Test Cases

- **Test case description:Sign Up**
    - Input: Valid email, username, and password
    - Condition or function under test:Sign Up
    - Expected Output:Confirm account was created and information reflects what was inputted
    - Output:Allows user to sign Up
- **Test case description:Sign In**
    - Input: Valid email or username and password
    - Condition or function under test:Sign In
    - Expected Output: Confirm user was signed in and info populates properly on profile page.
    - Output:Allows user to sign In
- **Test case description:Create Listing**
    - Input: Listing parameters
    - Condition or function under test:Create Listing
    - Expected Output: Confirm listing was created and info populates properly on listing page
    - Output:Allows user to create a listing
- **Test case description:Edit Listing**
    - Input: Listing changes
    - Condition or function under test:Edit listing
    - Expected Output: Confirm listing was updated and info populates properly on listing page
    - Output:Allows user to edit listings
- **Test case description:Search**
    - Input: keyword
    - Condition or function under test:Search
    - Expected Output: Listings containing the keyword
    - Output:Allows the user to search for a specific listing
- **Test case description:Change Search Parameters**
    - Input: New search parameters
    - Condition or function under test:Change Search Parameters
    - Expected Output: Updated search results
    - Output:Allows user to update their search
- **Test case description:Edit Profile**
    - Input: New profile settings
    - Condition or function under test:Edit Profile
    - Expected Output: Confirm profile settings updated and info populates properly on profile page
    - Output: Allows the user to update/edit their profile page
- **Test case description:Redirects**
    - **Not logged in**
        - Input: Home page URL
        - Condition or function under test:Redirects not logged in
        - Expected Output: No search bar, sign in and sign up buttons
        - Output:redirects the user to the home page with sign in or sign up button

        - Other Input: Profile page URL or search URL
        - Other Expected Output: Redirect to sign in
        - Other Output:redirects the user to sign in page
        - Other Input: Contact Us URL, Sign In URL, or Sign Up URL
        - Other Expected Output: No redirect
        - Other Output:
    - **Logged In**
        - Input: Home page URL
        - Condition or function under test:Redirects logged in
        - Expected Output: Search bar and no sign in/up buttons
        - Output:Redirect the user to the home page with featured listings
        - Other Input: profile page URL
        - Other Expected Output: Page populated with specified user info
        - Other Output:Redirects user to their specific profile page.
        - Other Input: Search page
        - Other expected Output: results based on search parameters
        - Other Output:Redirects user to a list with their search parameters.

API-related Tests

- **Endpoint /listings/ tests**
    - Reachability test
        Description: Verifies /listings/ endpoint is reachable
        Input: HTTP Request, username
        Function under test: GetListingsForUser
        Expected result: No HTTP_404_NOT_FOUND
        Result: <matches expected>
    - GET test
        Description: Verifies /listings/ returns all listings based on a set of query parameters
        Input: HTTP Request, query_params
        Function under test: GetListingsByQuery
        Expected result: All listings where <query> is contained (CI) in the listing title returned in JSON format
        Result: <matches expected>
    - PUT test
        Description: Verifies /listings/ updates a listing
        Input: HTTP Request, username, title
        Function under test: ModifyListingsForUser
        Expected result: Given Listing, if found, is correctly updated
        Result: <matches expected>
    - DELETE test
        Description: Verifies /listings/ deletes a listing
        Input: HTTP Request, username, title
        Function under test: ModifyListingsForUser
        Expected result: HTTP_202_NO_CONTENT
        Result: <matches expected>
- **Endpoint /wishlist/ tests**
    - Reachability test

Description: Verifies /wishlist/ endpoint is reachable
Input: HTTP Request, username
Function under test: GetWishlistForUser
Expected result: No HTTP_404_NOT_FOUND
Result: <matches expected>

- o GET test
  Description: Verifies /wishlist/ endpoint returns all wishlist listings belonging to <username>
  Input: HTTP Request, username
  Function under test: GetWishlistForUser
  Expected result: All wishlist listings belonging to <username> returned in JSON format
  Result: <matches expected>
- o POST test
  Description: Verifies /wishlist/ endpoint can create a wishlist listing
  Input: HTTP Request, username, WishlistListing object in JSON format
  Function under test: GetWishlistForUser
  Expected result: New WishlistListing created
  Result: <matches expected>

- **Endpoint /users/ tests**
  - o Reachability test
    Description: Verifies /users/ endpoint is reachable
    Input: HTTP Request
    Function under test: AllUsers
    Expected result: No HTTP_404_NOT_FOUND
    Result: <matches expected>
  - o GET test
    Description: Verifies /users/ returns all users
    Input: HTTP Request
    Function under test: AllUsers
    Expected result: All Users returned in JSON format
    Result: <matches expected>
  - o POST test
    Description: Verifies /users/ can create a user
    Input: HTTP Request, User object in JSON format
    Function under test: AllUsers
    Expected result: New User created
    Result: <matches expected>

Backend Test cases:

Django Rest framework Server Test Cases:

- **Test case description: Create a Superuser**
  - o Input: Valid username, email address and password
  - o Condition or function under test: create_superuser(self, email, password)
  - o Expected Result: If valid, Json Response confirmation
    If invalid, raise serializers.ValidationError as a Json Response
  - o Output: Json Response with the username

- **Test case description: Authentication Login**
  - Input: Valid username and password
  - Condition or function under test:

```
create_auth_token(sender, instance = None, created = False, **kwargs):
```

  - Expected Result: If valid, Json Response granting access

```
If invalid,
    if not email:
        raise ValueError('Users must have an email address')
    if not username:
        raise ValueError('Users must have a username')
    if not password:
        raise ValueError('User must have a password')
    a Json Response
```

  - Output: Confirmation Json Response granting access

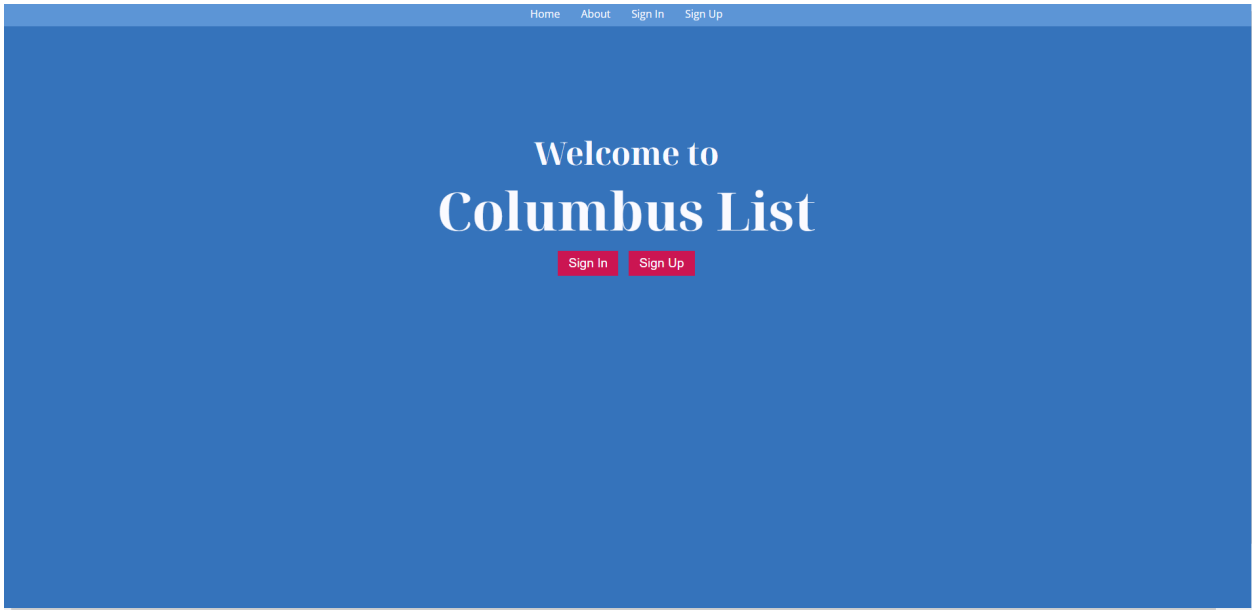
- **Test case description: Register User**
  - Input: Valid username, password, first name, last name, email address
  - Condition or function under test:

```
create_user(self, email, username, password=None)
```

  - Expected Result: If valid, Json Response confirmation

```
If invalid,
    if not email:
        raise ValueError('Users must have an email address')
    if not username:
        raise ValueError('Users must have a username')
    if not password:
        raise ValueError('User must have a password')
    a Json Response
```

  - Output: Json Response



For this milestone, we mainly focused on integrating the Frontend and Backend and polishing our overall product.

1	Create User Stories	akardorff, bkab1, ...	Done
2	Create Basic System Architecture	akardorff, bkab1, ...	Done
3	Create SQL database	mkimdd	Done
4	Create Template Django REST Framework Project	bkab1	Done
5	Add Node.js and React.js project in subfolder	EfficiencyIsKey and ...	Done
6	Create basic database component	mkimdd	Done
7	Create basic frontend router component	SamJones329	Done
8	Create basic UI components (search bar/filters, listing, profile display, nav bar)	EfficiencyIsKey	Done
9	Create class diagram	akardorff, bkab1, ...	Done
10	Create ER diagram for database	akardorff, bkab1, ...	Done
11	Create styling for basic UI components	EfficiencyIsKey and ...	Done
12	Create additional UI components and subcomponents	EfficiencyIsKey and ...	Done
13	Design documents	akardorff, bkab1, ...	Done
14	Create Requirements Specification documents	akardorff, bkab1, ...	Done
15	Design frontend test plans	EfficiencyIsKey and ...	Done
16	Design backend test plans	akardorff, bkab1, ...	Done
17	Create use cases	akardorff, bkab1, ...	Done
18	Initiate Django Rest Framework	bkab1	Done

19	Django Rest Framework - Data Base Migrations	bkab1 and mkimdd	Done
20	Create use case diagram	akardorff, bkab1, ...	Done
21	Design API	akardorff, bkab1, ...	Done
22	Create dummy frontend API handler to test UI components	SamJones329	Done
23	Create Listing Class	akardorff	Done
24	Create Account Class	akardorff	Done
25	Create Image Class	akardorff	Done
26	Pagination in DRF	bkab1	Done
27	Add end-points for our framework	bkab1 and mkimdd	Done
28	Flesh out UI to have full functionality	EfficiencyIsKey and ...	Done
29	Refine styling	EfficiencyIsKey and ...	Done
30	Start implementation of test cases	EfficiencyIsKey and ...	Done
31	Start implementation of authentication	EfficiencyIsKey and ...	Done
32	Start implementing real API handler	EfficiencyIsKey and ...	Done
33	Refine test cases	EfficiencyIsKey and ...	Done
34	Update class diagram	EfficiencyIsKey and ...	Done
35	Implement search functionality	EfficiencyIsKey and ...	Done
36	Implement search filtering functionality	EfficiencyIsKey and ...	Done

37	Implement listing preview	EfficiencyIsKey and ...	Done
38	Implement listing page	EfficiencyIsKey and ...	Done
39	Implement contact form	EfficiencyIsKey and ...	Done
40	Implement sign in form	EfficiencyIsKey and ...	Done
41	Implement sign up form	EfficiencyIsKey and ...	Done
42	Implement profile page	EfficiencyIsKey and ...	Done
43	Define Serializers for DjangoRest Framework	bkab1 and mkimdd	Done
44	Write Views for our DjangoRest Framework	bkab1 and mkimdd	Done
45	Write up API URLS in DjangoRest	bkab1 and mkimdd	Done
46	User Token Authentication	akardorff, bkab1, ...	Done
47	[DB] Implement tags	mkimdd	Done
48	[DB] Implement test cases for tags and images	mkimdd	Done
49	[Search/Filter] filter by tags	mkimdd	Done
50	[Search/Filter] filter by username	mkimdd	Done
51	[Search/Filter] combination filtering listings by title, tags, and username	mkimdd	Done
52	Merge authentication branch with API endpoints branch	akardorff, bkab1, ...	Done
53	Connect frontend and backend components for full system testing	EfficiencyIsKey and ...	Done
54	Implement functionality of forms	EfficiencyIsKey and ...	Done

55	integrate front-end with back-end api	EfficiencyIsKey and ...	Done
56	further implement front-end test cases	EfficiencyIsKey and ...	Done
57	further refine styling	EfficiencyIsKey and ...	Done
58	implement front-end session storing	EfficiencyIsKey and ...	Done
59	integrate image uploading into front-end	EfficiencyIsKey and ...	Done
60	work on pagination in search results	EfficiencyIsKey and ...	Done
61	[API] Edit views to take parameters from Request and not from URL	mkimdd	Done

Milestone 3

FrontEnd Test Cases

- **Test case description:Sign Up**
    - Input: Valid email, username, and password
    - Condition or function under test:Sign Up
    - Expected Output:Confirm account was created and information reflects what was inputted
    - Output:Allows user to sign Up
- **Test case description:Sign In**
    - Input: Valid email or username and password
    - Condition or function under test:Sign In
    - Expected Output: Confirm user was signed in and info populates properly on profile page.
    - Output:Allows user to sign In
- **Test case description:Create Listing**
    - Input: Listing parameters
    - Condition or function under test:Create Listing
    - Expected Output: Confirm listing was created and info populates properly on listing page
    - Output:Allows user to create a listing
- **Test case description:Edit Listing**
    - Input: Listing changes
    - Condition or function under test:Edit listing
    - Expected Output: Confirm listing was updated and info populates properly on listing page
    - Output:Allows user to edit listings
- **Test case description:Search**
    - Input: keyword
    - Condition or function under test:Search
    - Expected Output: Listings containing the keyword
    - Output:Allows the user to search for a specific listing
- **Test case description:Change Search Parameters**
    - Input: New search parameters
    - Condition or function under test:Change Search Parameters
    - Expected Output: Updated search results
    - Output:Allows user to update their search
- **Test case description:Edit Profile**
    - Input: New profile settings
    - Condition or function under test:Edit Profile
    - Expected Output: Confirm profile settings updated and info populates properly on profile page
    - Output: Allows the user to update/edit their profile page
- **Test case description:Redirects**
    - **Not logged in**
        - Input: Home page URL
        - Condition or function under test:Redirects not logged in
        - Expected Output: No search bar, sign in and sign up buttons
        - Output:redirects the user to the home page with sign in or sign up button

        - Other Input: Profile page URL or search URL
        - Other Expected Output: Redirect to sign in
        - Other Output:redirects the user to sign in page
        - Other Input: Contact Us URL, Sign In URL, or Sign Up URL
        - Other Expected Output: No redirect
        - Other Output:
    - **Logged In**
        - Input: Home page URL
        - Condition or function under test:Redirects logged in
        - Expected Output: Search bar and no sign in/up buttons
        - Output:Redirect the user to the home page with featured listings
        - Other Input: profile page URL
        - Other Expected Output: Page populated with specified user info
        - Other Output:Redirects user to their specific profile page.
        - Other Input: Search page
        - Other expected Output: results based on search parameters
        - Other Output:Redirects user to a list with their search parameters.

Backend Test cases:

Django Rest framework Server Test Cases:

- **Test case description: Create a Superuser**
    - Input: Valid username, email address and password
    - Condition or function under test: create_superuser(self, email, password)
    - Expected Result: If valid, Json Response confirmation
      If invalid, raise serializers.ValidationError as a Json Response
    - Output: Json Response with the username

- **Test case description: Authentication Login**
    - Input: Valid username and password
    - Condition or function under test:
      create_auth_token(sender, instance = None, created = False, **kwargs):
    - Expected Result: If valid, Json Response granting access
      If invalid,

```
if not email:
    raise ValueError('Users must have an email address')
if not username:
    raise ValueError('Users must have a username')
if not password:
```

raise ValueError('User must have a password')
a Json Response

- Output: Confirmation Json Response granting access

- **Test case description: Register User**
  - Input: Valid username, password, first name, last name, email address
  - Condition or function under test:
    create_user(self, email, username, password=None)
  - Expected Result: If valid, Json Response confirmation
    If invalid,
    if not email:
    raise ValueError('Users must have an email address')
    if not username:
    raise ValueError('Users must have a username')
    if not password:
    raise ValueError('User must have a password')
    a Json Response
  - Output: Json Response

- **Test case description: Page Request - View listing for user**
  - Input: (self, request, username)
  - Condition or function under test: get(self, request)
  - Expected Result: Json Response if valid
    Bad request, if invalid
  - Output: Json Response

- **Test case description: Post Request - Create listing for user**
  - Input: (self, request, username, title)
  - Condition or function under test: Post(self, request, username, title)
  - Expected Result: If valid, Json Response HTTP_201_Created
    If invalid, Json Response HTTP_400_BAD REQUEST
  - Output: Json Response HTTP_201_Created

- **Test case description: Put Request - Edit listing for user**
  - Input: (self, request, username, title)
  - Condition or function under test: put(self, request, username, title)
  - Expected Result: If valid, HTTP_Listing created
    If invalid, HTTP_400_BAD_REQUEST
  - Output: Json Response HTTP_201_Created

- **Test case description: Delete Request - Delete listing for user**
    - Input: (self, request, username, title)
    - Condition or function under test: delete(self, request, username, title)
    - Expected Result: If valid, HTTP_Listing Deleted
      If invalid, HTTP_400_BAD_REQUEST
    - Output: Json Response HTTP_DELETED

Database-related Tests

- **Setup Test**
    - Description: Verifies correct creation of database and initial tables.
    - Input: NONE
    - Function under test: Server setup
    - Expected Result: columbuslist database created with four tables (Users, Listings, Wishlists, Images)
    - Result: <matches expected>
- **addUser() Test**
    - Description: Verifies a user can be correctly entered in the Users table.
    - Input: username and password
    - Function under test: addUser
    - Expected Result: User entry created and correctly inserted into Users table
    - Result: <matches expected>
- **addListingForUser() Test**
    - Description: Verifies a listing can be corrected entered in the Listings table
    - Input: title, description, username, price, and contact
    - Function under test: addListingForUser
    - Expected Result: Listing entry correctly inserted into Listings table
    - Result: <matches expected>
- **addWishlistListingForUser() Test**
    - Description: Verifies a wishlist listing can be corrected entered in the Wishlists table
    - Input: listingID and username
    - Function under test: addWishlistListingForUser
    - Expected Result: Wishlist entry correctly inserted into Wishlists table
    - Result: <matches expected>
- **addImageForListing() Test**
    - Description: Verifies an image can be added to the Images table
    - Input: url and listingID
    - Function under test: addImageForListing
    - Expected Result: Image entry correctly inserted into Images table
    - Result: <unimplemented functionality>
- **getListingsForUser() Test**
    - Description: Verifies a specific user's listings can be retrieved
    - Input: username
    - Function under test: getListingsForUser
    - Expected Result: all listings with given username attribute returned
    - Result: <matches expected>
- **getWishlistListingsForUser() Test**
    - Description: Verifies a specific user's wishlist listings can be retrieved
    - Input: username
    - Function under test: getWishlistListingsForUser
    - Expected Result: all wishlist listings with given username attribute returned

- Result: <matches expected>

- **getImagesForListing() Test**
  - Description: Verifies a listing's images can be retrieved
  - Input: listingID
  - Function under test: getImagesForListing
  - Expected Result: all images with given listingID attribute returned
  - Result: <unimplemented functionality>

API-related Tests

- **Endpoint /listings/ tests**
  - Reachability test
    - Description: Verifies /listings/ endpoint is reachable
    - Input: HTTP Request
    - Function under test: AllListings
    - Expected result: No HTTP_404_NOT_FOUND
    - Result: <matches expected>
  - GET test
    - Description: Verifies /listings/ returns all listings
    - Input: HTTP Request
    - Function under test: AllListings
    - Expected Result: All listings returned in JSON format
    - Result: <matches expected>
  - POST test
    - Description: Verifies /listings/ creates a listing
    - Input: HTTP Request, Listing object in JSON format
    - Function under test: AllListings
    - Expected Result: New Listing created
    - Result: <matches expected>
- **Endpoint /listings/<username> tests**
  - Reachability test
    - Description: Verifies /listings/<username> endpoint is reachable
    - Input: HTTP Request, username
    - Function under test: GetListingsForUser
    - Expected result: No HTTP_404_NOT_FOUND
    - Result: <matches expected>
  - GET test
    - Description: Verifies /listings/<username> returns all listings belonging to <username>
    - Input: HTTP Request, username
    - Function under test: GetListingsForUser
    - Expected Result: All listings belonging to <username> returned in JSON format
    - Result: <matches expected>
- **Endpoint /listingsquery/<query>/ tests**
  - Reachability test
    - Description: Verifies /listingsquery/<query>/ endpoint is reachable
    - Input: HTTP Request, query
    - Function under test: GetListingsByQuery
    - Expected result: No HTTP_404_NOT_FOUND
    - Result: <matches expected>
  - GET test

Description: Verifies /listingsquery/<query>/ returns all listings where <query> is contained (case insensitive) in the listing title
Input: HTTP Request, query
Function under test: GetListingsByQuery
Expected result: All listings where <query> is contained (CI) in the listing title returned in JSON format
Result: <matches expected>

- **Endpoint /listings/<username>/<title>/ tests**
  - Reachability test
    Description: Verifies /listings/<username>/<title>/ endpoint is reachable
    Input: HTTP Request, username, title
    Function under test: ModifyListingsForUser
    Expected result: No HTTP_404_NOT_FOUND
    Result: <matches expected>
  - PUT test
    Description: Verifies /listings/<username>/<title>/ updates a listing
    Input: HTTP Request, username, title
    Function under test: ModifyListingsForUser
    Expected result: Given Listing, if found, is correctly updated
    Result: <matches expected>
  - DELETE test
    Description: Verifies /listings/<username>/<title>/ deletes a listing
    Input: HTTP Request, username, title
    Function under test: ModifyListingsForUser
    Expected result: HTTP_202_NO_CONTENT
    Result: <matches expected>
- **Endpoint /wishlist/<username>/ tests**
  - Reachability test
    Description: Verifies /wishlist/<username>/ endpoint is reachable
    Input: HTTP Request, username
    Function under test: GetWishlistForUser
    Expected result: No HTTP_404_NOT_FOUND
    Result: <matches expected>
  - GET test
    Description: Verifies /wishlist/<username>/ endpoint returns all wishlist listings belonging to <username>
    Input: HTTP Request, username
    Function under test: GetWishlistForUser
    Expected result: All wishlist listings belonging to <username> returned in JSON format
    Result: <matches expected>
  - POST test
    Description: Verifies /wishlist/<username>/ endpoint can create a wishlist listing
    Input: HTTP Request, username, WishlistListing object in JSON format
    Function under test: GetWishlistForUser
    Expected result: New WishlistListing created
    Result: <matches expected>
- **Endpoint /users/ tests**
  - Reachability test
    Description: Verifies /users/ endpoint is reachable
    Input: HTTP Request
    Function under test: AllUsers
    Expected result: No HTTP_404_NOT_FOUND
    Result: <matches expected>
  - GET test
    Description: Verifies /users/ returns all users

Input: HTTP Request
Function under test: AllUsers
Expected result: All Users returned in JSON format
Result: <matches expected>

- POST test
  Description: Verifies /users/ can create a user
  Input: HTTP Request, User object in JSON format
  Function under test: AllUsers
  Expected result: New User created
  Result: <matches expected>

Listing, Account, and Image class test cases:

Listing class tests:

- **Constructor**
  - **Input: listing title, price, contact required. Description, Images, Title Tags are optional inputs**
  - **Condition or function under test: creating listing object**
  - **Expected output: All inputs are properly saved, if a required field is blank there is an error, if an optional field is blank nothing happens**
- **getTitle()**
  - **input: none**
  - **Condition or function under test: get Title**
  - **expected output: title of listing**
  - **output: allows the title of a listing to be retrieved**
- **changeTitle(str)**
  - **input: new title of listing**
  - **Condition or function under test: change Title**
  - **expected output: confirmation that the title was changed**
  - **output: allows the title of a listing to be changed**
- **getDescription():**
  - **input: none**
  - **Condition or function under test: get Description**
  - **expected output: description of listing**

    - output: allows the description of a listing to be retrieved
- changeDescription(str):
    - input: new description of listing
    - Condition or function under test: change description
    - expected output: confirmation that the description was changed
    - output: allows the description of a listing to be changed
- getImages()
    - input: none
    - Condition or function under test: get images
    - expected output: image of listing
    - output: allows the images of a listing to be retrieved
- addImages(Listing[])
    - input: list of images to add to listing
    - Condition or function under test: add Images
    - expected output: confirmation that the images were added
    - output: allows images to be added to the listing
- removeImages(Listing[])
    - input: list of images to remove from listing
    - Condition or function under test: remove images
    - expected output: confirmation that the images were removed
    - output: allows images to be removed from the listing
- getPrices()
    - input: none
    - Condition or function under test: get Prices
    - expected output: price of listing
    - output: allows the price of a listing to be retrieved
- changePrices(str)
    - input: price of listing

    - Condition or function under test: change prices
    - expected output: confirmation that the price was changed
    - output: allows the price of a listing to be changed
- getContact()
    - input: none
    - Condition or function under test: get contact
    - expected output: contact info for seller of listing
    - output: allows the contact of a listing to be retrieved
- changeContact(str)
    - input: contact info for of listing
    - Condition or function under test: change contact
    - expected output: confirmation that the contact info was changed
    - output: allows the contact of a listing to be changed
- getTags()
    - input: none
    - Condition or function under test: get tags
    - expected output: tags of the listing
    - output: allows the tags of a listing to be retrieved
- addTags(str[])
    - input: tags to be added to listing
    - Condition or function under test: add Tags
    - expected output: confirmation that the tags were added
    - output: allows tags to be added to the listing
- removeTags(str[])
    - input: tags to be added to listing
    - Condition or function under test: remove Tags
    - expected output: confirmation that the tags were removed
    - output: allows tags to be removed from the listing

Account class tests:

- **Constructor**
    - **Input: Username and password required for account.**
    - **Condition or function under test: constructor for account object**
    - **Expected output: Username and password properly saved**
    - **output: allows the creation of account objects**
- **getUsername()**
    - **input: none**
    - **Condition or function under test: get username**
    - **Expected output: username of account**
    - **output: allows the username of an account to be retrieved**
- **getListings()**
    - **input: none**
    - **Condition or function under test: get Listings**
    - **Expected output: all active listings posted by account**
    - **output: allows the Listings of an account to be retrieved**
- **getWishlist()**
    - **input: none**
    - **Condition or function under test: get Wishlist**
    - **Expected output: All listings in the accounts wishlist**
    - **output: allows the wishlist of an account to be retrieved**
- **addToWishlist(Listing)**
    - **input: listing to be added to wishlist**
    - **Condition or function under test: add to wishlist**
    - **expected output: confirmation that listing was added**
    - **output: allows listings to be added to an accounts wishlist**
- **removeFromWishlist(Listing)**
    - **input: listing to be removed from wishlist**

- Condition or function under test: remove from wishlist
- expected output: confirmation that listing was removed
- output: allows listings to be removed from an accounts wishlist

- **changePassword(str)**
  - input: new password to account
  - Condition or function under test: change password
  - output: confirmation password was changed
  - output: allows the password of an account to be changed

Image class tests:

- **Constructor:**
  - Input: filename of image
  - Condition or function under test: Constructor
  - Expected output: same filename as input
  - output: allows the creation of an Image object

Use Case Diagram:

Feature Descriptions

Admin:

- As an admin, I want to be able to delete users.
- As an admin, I want to delete listing that violate school policies
- As an admin, I want to be able to edit tags, images, and the description of listings
- As an admin, I want to be able to delete users and listings to ensure that students follow the student code of conduct.

Buyer

- As a potential buyer, I want to be able to view all listings for sale
- As a potential buyer, I want to add listings to my wishlist if they interest me
- As a potential buyer, I want to view all listings in my wishlist
- As a buyer, I want to purchase items from a listing if it is still available
- As a buyer, I want to be notified of any price changes for listings in my wishlist
- As a buyer, I want to be able to view relatively high-resolution images of products to be able to tell the condition of the item.
- As a buyer, I want to be able to perform complex search and sort operations on available listings so I can find exactly what I am looking for.

Seller

- As a user wanting to sell an item, I want to create a listing for my item
- As a seller, I want to edit and customize the title, tags, and description of my listings
- As a seller, I want to post pictures of the item I am selling
- As a seller, I want to edit the price of my listing after its posted
- As a seller, I want to remove listings I have made
- As a seller, I want to choose what contact information is visible to potential buyers
- As a seller, I want to upload images with a relatively high-resolution so buyers can clearly see what I'm selling.

All users

- As a user, I want to log in if I know the correct password
- As a user, I want to view my account information
- As a user, I want to edit my account information
- As a current user, I want to delete my account

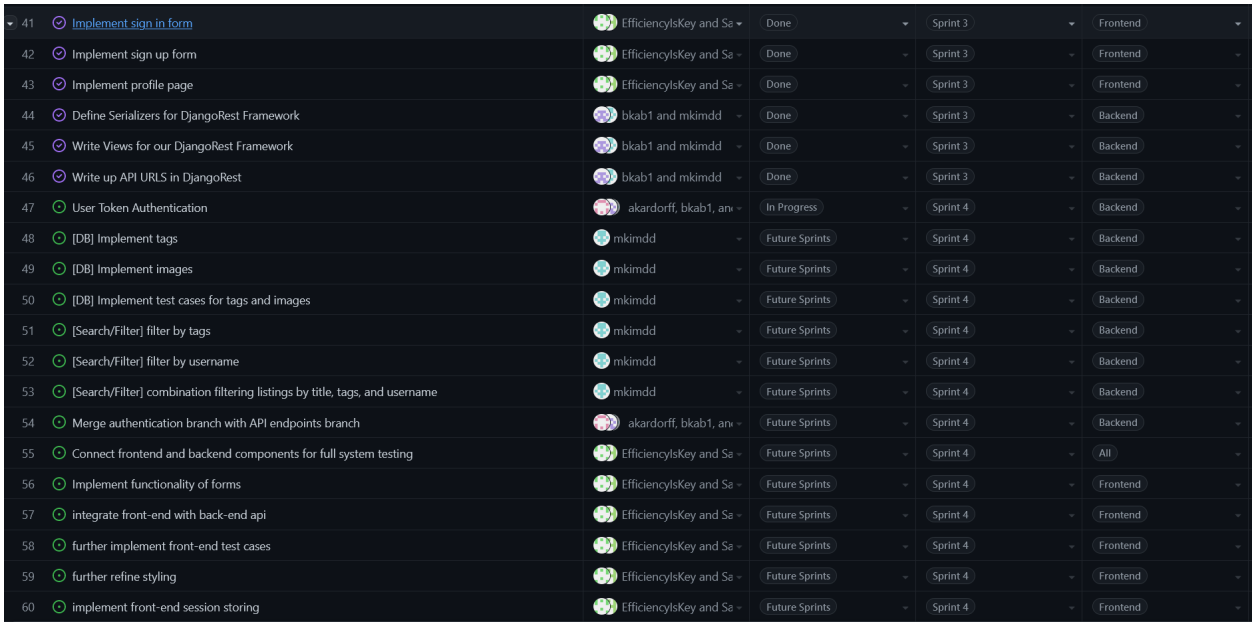
- As a new user, I want to create a new account
- As a user, I want to be able to visit this website from any modern browser (Firefox, Chromium based browsers, etc.) so I don’t have to worry about browser compatibility.
- As a user, I want to be confident that the other users I interact with are not looking to scam me.
- As a user, I want a reliable website that won’t crash constantly
- As a user, I want to be able to use this website from any modern computer without issues.

Project Backlog

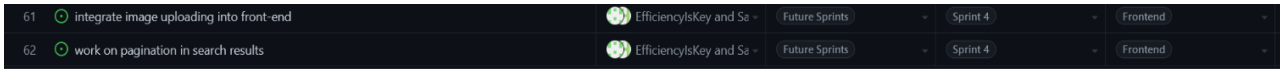
CSC 4330 Team E

Sprint Backlog | Board View | Sprint 1 | Sprint 2 | Sprint 3 | Sprint 4 | Frontend | Backend | + New view

	Title	Assignees	Status	Sprint	Subteam
1	Create User Stories	akardorff, bkab1, Eff	Done	Sprint 1	All
2	Create Basic System Architecture	akardorff, bkab1, Eff	Done	Sprint 1	All
3	Create SQL database	mkimdd	Done	Sprint 2	Backend
4	Create Template Django REST Framework Project	bkab1	Done	Sprint 2	Backend
5	Add Node.js and React.js project in subfolder	EfficiencyIsKey and Sa	Done	Sprint 2	Frontend
6	Create basic database component	mkimdd	Done	Sprint 2	Backend
7	Create basic frontend router component	SamJones329	Done	Sprint 2	Frontend
8	Create basic UI components (search bar/filters, listing, profile display, nav bar)	EfficiencyIsKey	Done	Sprint 2	Frontend
9	Create class diagram	akardorff, bkab1, Eff	Done	Sprint 2	All
10	Create ER diagram for database	akardorff, bkab1, an	Done	Sprint 2	Backend
11	Create styling for basic UI components	EfficiencyIsKey and Sa	Done	Sprint 2	Frontend
12	Create additional UI components and subcomponents	EfficiencyIsKey and Sa	Done	Sprint 2	Frontend
13	Design documents	akardorff, bkab1, Eff	Done	Sprint 2	All
14	Create Requirements Specification documents	akardorff, bkab1, Eff	Done	Sprint 2	All
15	Design frontend test plans	EfficiencyIsKey and Sa	Done	Sprint 2	Frontend
16	Design backend test plans	akardorff, bkab1, an	Done	Sprint 2	Backend
17	Create use cases	akardorff, bkab1, Eff	Done	Sprint 2	All
18	Initiate Django Rest Framework	bkab1	Done	Sprint 2	Backend
19	Django Rest Framework - Data Base Migrations	bkab1 and mkimdd	Done	Sprint 3	Backend
20	Create use case diagram	akardorff, bkab1, an	Done	Sprint 3	All
21	Design API	akardorff, bkab1, Eff	Done	Sprint 3	Backend
22	Create dummy frontend API handler to test UI components	SamJones329	Done	Sprint 3	Frontend
23	Create Listing Class	akardorff	Done	Sprint 2	Backend
24	Create Account Class	akardorff	Done	Sprint 2	Backend
25	Create Image Class	akardorff	Todo	Sprint 3	Backend
26	Pagination in DRF	bkab1	Done	Sprint 3	Backend
27	Add end-points for our framework	bkab1 and mkimdd	Done	Sprint 3	Backend
28	Flesh out UI to have full functionality	EfficiencyIsKey and Sa	Done	Sprint 3	Frontend
29	Refine styling	EfficiencyIsKey and Sa	Done	Sprint 3	Frontend
30	Start implementation of test cases	EfficiencyIsKey and Sa	Done	Sprint 3	Frontend
31	Start implementation of authentication	EfficiencyIsKey and Sa	Future Sprints	Sprint 3	Frontend
32	Start implementing real API handler	EfficiencyIsKey and Sa	Done	Sprint 3	Frontend
33	Refine test cases	EfficiencyIsKey and Sa	Done	Sprint 3	Frontend
34	Update class diagram	EfficiencyIsKey and Sa	Done	Sprint 3	Frontend
35	Create data flow diagram	EfficiencyIsKey and Sa	Future Sprints	Sprint 3	Frontend
36	Implement search functionality	EfficiencyIsKey and Sa	Done	Sprint 3	Frontend
37	Implement search filtering functionality	EfficiencyIsKey and Sa	Done	Sprint 3	Frontend
38	Implement listing preview	EfficiencyIsKey and Sa	Done	Sprint 3	Frontend
39	Implement listing page	EfficiencyIsKey and Sa	Done	Sprint 3	Frontend
40	Implement contact form	EfficiencyIsKey and Sa	Done	Sprint 3	Frontend



41	Implement sign in form	EfficiencyIsKey and Sa	Done	Sprint 3	Frontend
42	Implement sign up form	EfficiencyIsKey and Sa	Done	Sprint 3	Frontend
43	Implement profile page	EfficiencyIsKey and Sa	Done	Sprint 3	Frontend
44	Define Serializers for DjangoRest Framework	bkab1 and mkimdd	Done	Sprint 3	Backend
45	Write Views for our DjangoRest Framework	bkab1 and mkimdd	Done	Sprint 3	Backend
46	Write up API URLS in DjangoRest	bkab1 and mkimdd	Done	Sprint 3	Backend
47	User Token Authentication	akardorff, bkab1, an	In Progress	Sprint 4	Backend
48	[DB] Implement tags	mkimdd	Future Sprints	Sprint 4	Backend
49	[DB] Implement images	mkimdd	Future Sprints	Sprint 4	Backend
50	[DB] Implement test cases for tags and images	mkimdd	Future Sprints	Sprint 4	Backend
51	[Search/Filter] filter by tags	mkimdd	Future Sprints	Sprint 4	Backend
52	[Search/Filter] filter by username	mkimdd	Future Sprints	Sprint 4	Backend
53	[Search/Filter] combination filtering listings by title, tags, and username	mkimdd	Future Sprints	Sprint 4	Backend
54	Merge authentication branch with API endpoints branch	akardorff, bkab1, an	Future Sprints	Sprint 4	Backend
55	Connect frontend and backend components for full system testing	EfficiencyIsKey and Sa	Future Sprints	Sprint 4	All
56	Implement functionality of forms	EfficiencyIsKey and Sa	Future Sprints	Sprint 4	Frontend
57	integrate front-end with back-end api	EfficiencyIsKey and Sa	Future Sprints	Sprint 4	Frontend
58	further implement front-end test cases	EfficiencyIsKey and Sa	Future Sprints	Sprint 4	Frontend
59	further refine styling	EfficiencyIsKey and Sa	Future Sprints	Sprint 4	Frontend
60	implement front-end session storing	EfficiencyIsKey and Sa	Future Sprints	Sprint 4	Frontend



61	integrate image uploading into front-end	EfficiencyIsKey and Sa	Future Sprints	Sprint 4	Frontend
62	work on pagination in search results	EfficiencyIsKey and Sa	Future Sprints	Sprint 4	Frontend

Milestone 2

Feature Description:

Buyer

- As a potential buyer, I want to be able to view all listings for sale
- As a potential buyer, I want to add listings to my wishlist if they interest me
- As a potential buyer, I want to view all listings in my wishlist
- As a potential buyer, I want to be able to filter and search for listings
- As a buyer, I want to purchase items from a listing if it is still available
- As a buyer, I want to be notified of any price changes for listings in my wishlist

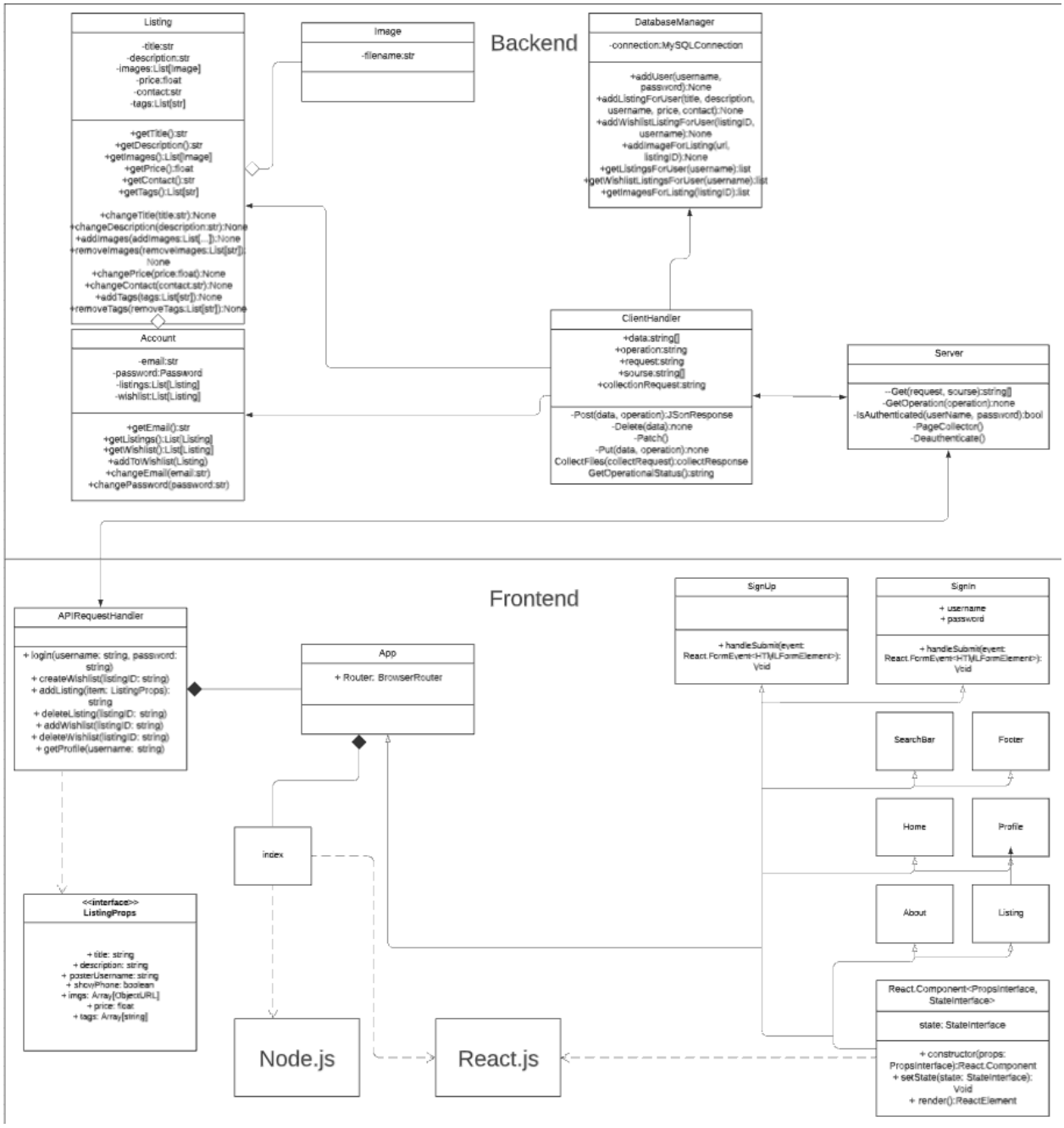
Seller

- As a user wanting to sell an item, I want to create a listing for my item
- As a seller, I want to edit and customize the title, tags, and description of my listings
- As a seller, I want to post pictures of the item I am selling
- As a seller, I want to edit the price of my listing after its posted
- As a seller, I want to remove listings I have made
- As a seller, I want to choose what contact information is visible to potential buyers

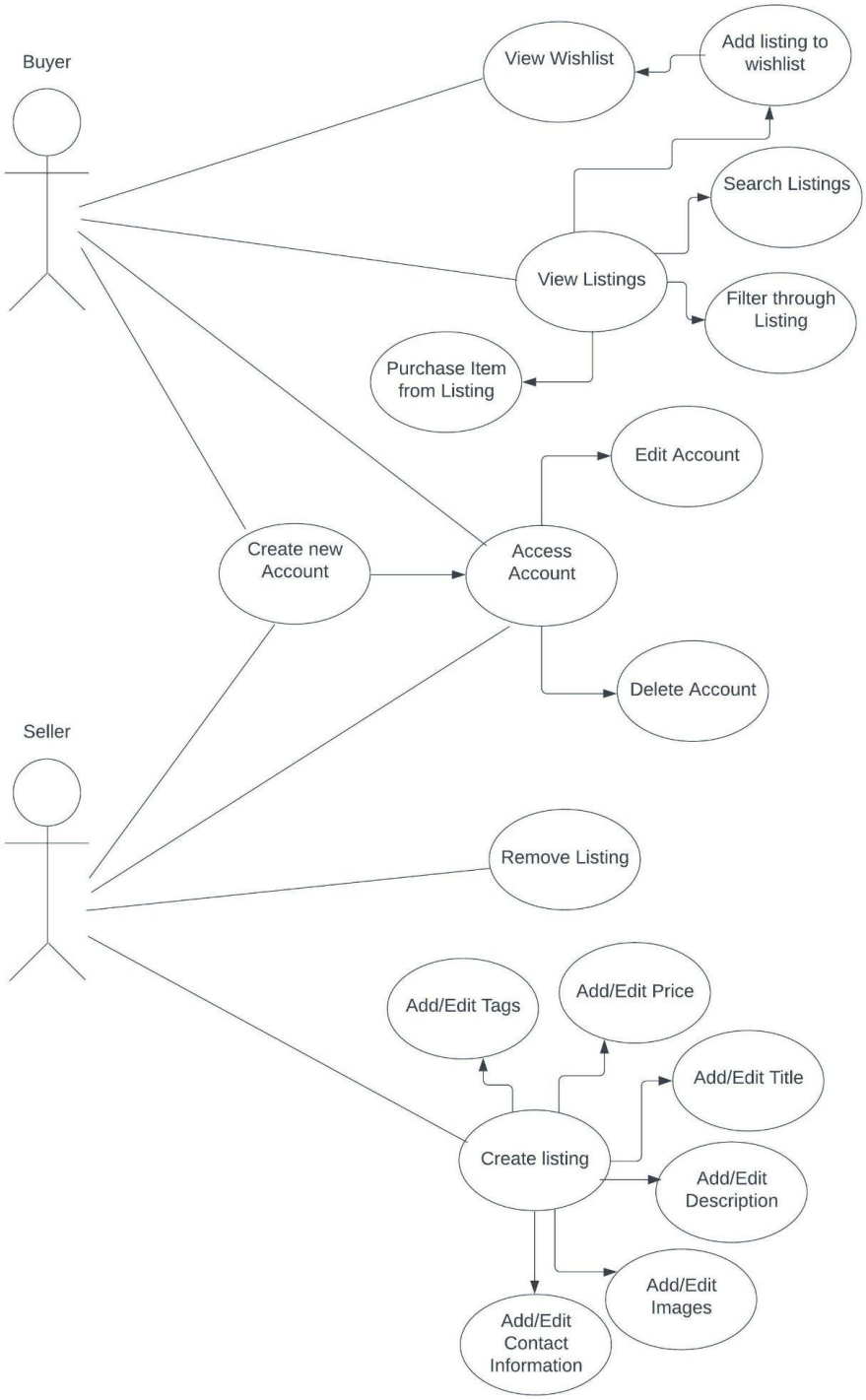
All users

- As a user, I want to log in if I know the correct password
- As a user, I want to view my account information
- As a user, I want to edit my account information
- As a current user, I want to delete my account
- As a new user, I want to create a new account

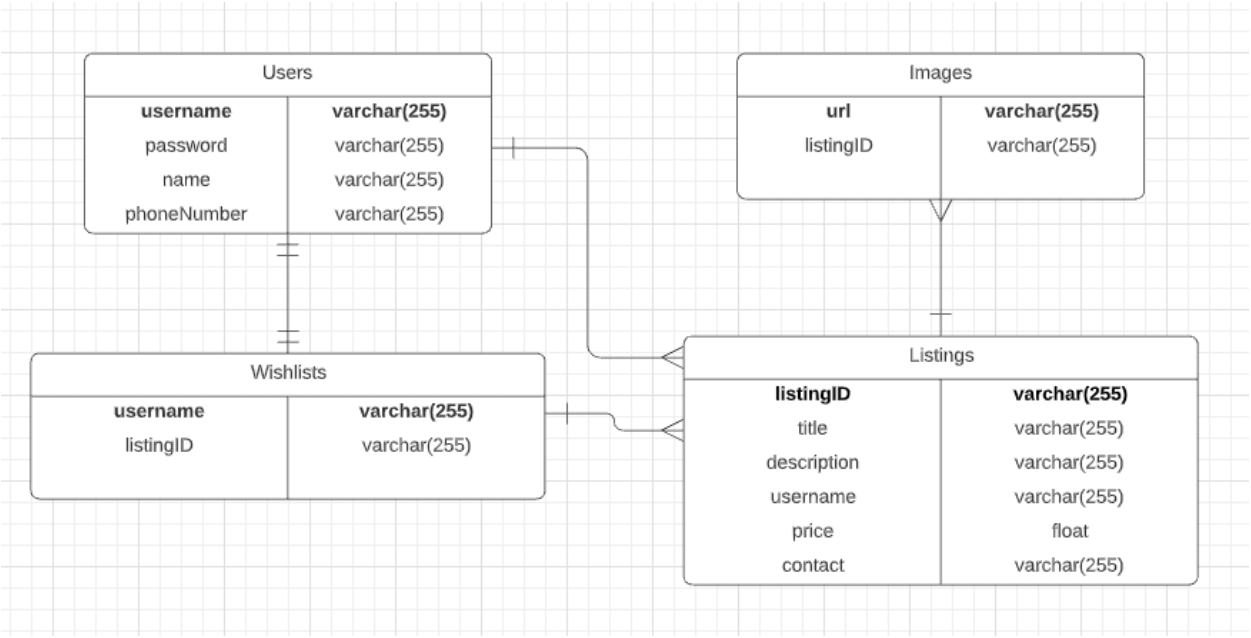
Class Diagram:



Use Case Diagram:



Entity-Relationship Diagram:



Preliminary Database Test Cases:

Database-related Tests

- **Setup Test**
  - Input: NONE
  - Expected Result: columbuslist database created with four tables (Users, Listings, Wishlists, Images)
- **addUser() Test**
  - Input: username and password
  - Expected Result: User entry correctly inserted into Users table
- **addListingForUser() Test**
  - Input: title, description, username, price, and contact
  - Expected Result: Listing entry correctly inserted into Listings table
- **addWishlistListingForUser() Test**
  - Input: listingID and username
  - Expected Result: Wishlist entry correctly inserted into Wishlists table
- **addImageForListing() Test**
  - Input: url and listingID
  - Expected Result: Image entry correctly inserted into Images table
- **getListingsForUser() Test**
  - Input: username
  - Expected Result: all listings with given username attribute returned
- **getWishlistListingsForUser() Test**
  - Input: username
  - Expected Result: all wishlist listings with given username attribute returned
- **getImagesForListing() Test**
  - Input: listingID
  - Expected Result: all images with given listingID attribute returned

Preliminary Django Rest framework Server Test Cases:

- **Test case description: Authentication Login**
  - Input: Valid username and password
  - Condition or function under test: IsAuthenticated(username, password)
  - Expected Result: Access Granted or Access Denied
  - Output: Confirmation Json Response

- **Test case description: Page Request**
  - Input: Valid username and password
  - Condition or function under test: get(self, request)
  - Expected Result: Json Response if valid
    Bad request, if invalid
  - Output: Json Response

- **Test case description: Page Request - View listing for user**
  - Input: (self, request, username)
  - Condition or function under test: get(self, request)
  - Expected Result: Json Response if valid
    Bad request, if invalid
  - Output: Json Response

- **Test case description: Post Request - Create listing for user**
  - Input: (self, request, username, title)
  - Condition or function under test: Post(self, request, username, title)
  - Expected Result: If valid, Json Response HTTP_201_Created
    If invalid, Json Response HTTP_400_BAD REQUEST
  - Output: Json Response HTTP_201_Created

- **Test case description: Put Request - Edit listing for user**
  - Input: (self, request, username, title)
  - Condition or function under test: put(self, request, username, title)
  - Expected Result: If valid, HTTP_Listing created
    If invalid, HTTP_400_BAD_REQUEST
  - Output: Json Response HTTP_201_Created

- **Test case description: Delete Request - Delete listing for user**
  - Input: (self, request, username, title)
  - Condition or function under test: delete(self, request, username, title)
  - Expected Result: If valid, HTTP_Listing Deleted

If invalid, HTTP_400_BAD_REQUEST

- Output: Json Response HTTP_DELETED

Preliminary Frontend Test Cases

- **Test case description:Sign Up**
  - Input: Valid email, username, and password
  - Condition or function under test:Sign Up
  - Expected Output:Confirm account was created and information reflects what was inputted
- **Test case description:Sign In**
  - Input: Valid email or username and password
  - Condition or function under test:Sign In
  - Expected Output: Confirm user was signed in and info populates properly on profile page.
- **Test case description:Create Listing**
  - Input: Listing parameters
  - Condition or function under test:Create Listing
  - Expected Output: Confirm listing was created and info populates properly on listing page
- **Test case description:Edit Listing**
  - Input: Listing changes
  - Condition or function under test:Edit listing
  - Expected Output: Confirm listing was updated and info populates properly on listing page
- **Test case description:Search**
  - Input: keyword
  - Condition or function under test:Search
  - Expected Output: Listings containing the keyword
- **Test case description:Change Search Parameters**
  - Input: New search parameters
  - Condition or function under test:Change Search Parameters
  - Expected Output: Updated search results
- **Test case description:Edit Profile**
  - Input: New profile settings
  - Condition or function under test:Edit Profile
  - Expected Output: Confirm profile settings updated and info populates properly on profile page
- **Test case description:Redirects**
  - **Not logged in**
    - Input: Home page URL
    - Condition or function under test:Redirects not logged in
    - Expected Output: No search bar, sign in and sign up buttons
    - Other Input: Profile page URL or search URL
    - Other Expected Output: Redirect to sign in
    - Other Input: Contact Us URL, Sign In URL, or Sign Up URL
    - Other Expected Output: No redirect

- **Logged In**
  - Input: Home page URL
  - Condition or function under test:Redirects logged in
  - Expected Output: Search bar and no sign in/up buttons
  - Other Input: profile page URL
  - Other Expected Output: Page populated with specified user info
  - Other Input: Search page
  - Other expected Output: results based on search parameters

Milestone 1

User Stories:

Buyer

Epic: As a Columbus University student, I want to buy items from fellow students so that I can get cheap stuff locally and with accountability.

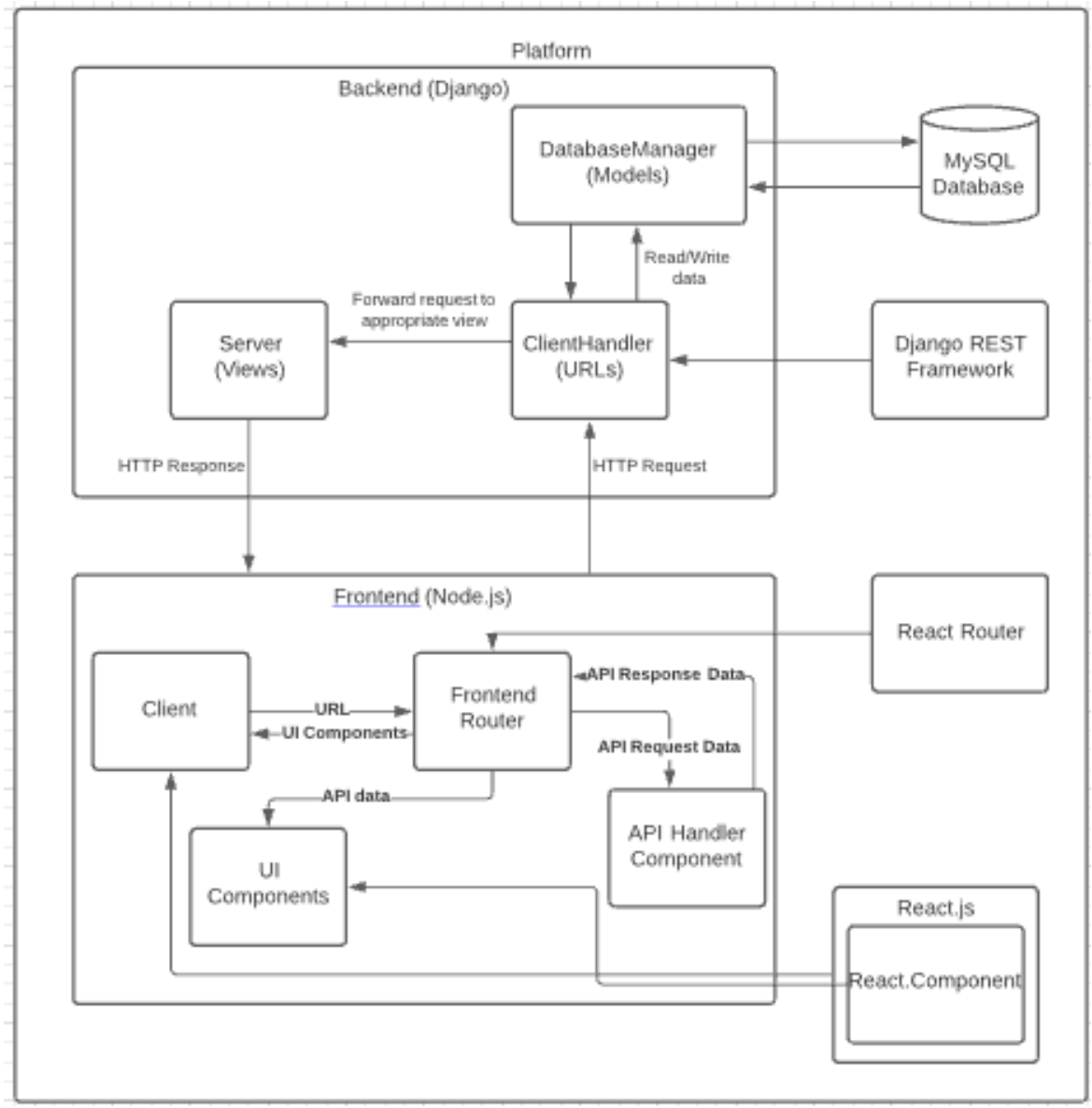
- As a ColumbusList buyer, I want the ability to save listings to a wishlist for quick access at a later time.
- As a user with a wishlist, I want to be notified of any price changes to listings in my wishlist.
- As a buyer, I want to be able to search listings for specific terms so that I can look for particular items.
- As a buyer, I want to be able to see recent listings when I go on the site so that I can easily see if something I want has been posted to be the first to try to purchase it.
- As a buyer, I want to be able to filter my searches by price and tag so that I can minimize time wasted looking at irrelevant listings and find what I want more quickly.

Seller

Epic: As a Columbus University student, I want to sell items to fellow students so I can get rid of things I don't want and make money safely.

- As a Columbus University student looking to sell items and services, I want the ability to add a sell listing advertising my item or service.
- As a user making a sell listing, I want the ability to add a title to my listing to quickly describe the item or service I want to sell.
- As a user making a sell listing, I want the ability to add a description to my listing to describe in detail the item or service I want to sell.
- As a user making a sell listing, I want the ability to add images to my listing to visually advertise my item or service.
- As a user making a sell listing, I want the ability to add a price to my listing to inform potential buyers of my required compensation.
- As a user making a sell listing, I want the ability to add contact information to my listing so that potential buyers can let me know of their interest in my item or service.
- As a user making a sell listing, I want the ability to add tags to my listing to improve the visibility of my listing and help potential buyers find what they are looking for quicker.
- As a ColumbusList user making a listing, I want to be able to edit all aspects of my listing after it has been posted.

System architecture:



Project and Sprint Backlogs

CSC 4330 Team E

Sprint Backlog | Board View | Sprint 1 | Sprint 2 | Sprint 3 | Frontend | Backend | + New view

	Title	Assignees	Status	Sprint	Subteam
1	Create User Stories	akardorff, bkab1, Eff	Done	Sprint 0	All
2	Create Basic System Architecture	akardorff, bkab1, Eff	Done	Sprint 0	All
3	Create SQL database	mkimdd	Done	Sprint 1	Backend
4	Create Template Django REST Framework Project	bkab1	Done	Sprint 1	Backend
5	Add Node.js and React.js project in subfolder	EfficiencyIsKey and Sa	Done	Sprint 1	Frontend
6	Design API	akardorff, bkab1, Eff	Todo	Sprint 3	Backend
7	Create basic database component	mkimdd	Done	Sprint 1	Backend
8	Create basic frontend router component	SamJones329	Done	Sprint 1	Frontend
9	Create basic UI components (search bar/filters, listing, profile display, nav bar)	EfficiencyIsKey	Done	Sprint 1	Frontend
10	Create dummy frontend API handler to test UI components	SamJones329	Todo	Sprint 3	Frontend
11	Create class diagram	akardorff, bkab1, Eff	Done	Sprint 1	All
12	Create ER diagram for database	akardorff, bkab1, an	Done	Sprint 1	Backend
13	Create styling for basic UI components	EfficiencyIsKey and Sa	Done	Sprint 1	Frontend
14	Create additional UI components and subcomponents	EfficiencyIsKey and Sa	Done	Sprint 1	Frontend
15	Design documents	akardorff, bkab1, Eff	Done	Sprint 1	All
16	Create Requirements Specification documents	akardorff, bkab1, Eff	Done	Sprint 1	All
17	Design frontend test plans	EfficiencyIsKey and Sa	Done	Sprint 1	Frontend
18	Design backend test plans	akardorff, bkab1, an	Done	Sprint 1	Backend
19	Create use cases	akardorff, bkab1, Eff	Done	Sprint 1	All
20	Initiate Django Rest Framework	bkab1	Done	Sprint 2	Backend
21	Django Rest Framework - Data Base Migrations	bkab1 and mkimdd	Done	Sprint 2	Backend
22	Modify MVT and eliminate Templates	bkab1	Done	Sprint 2	Backend
23	Define Serializers for DjangoRest Framework	bkab1	In Progress	Sprint 3	Backend
24	Define Serializers for DjangoRest Framework	bkab1	In Progress	Sprint 3	Backend
25	Write Views for our DjangoRest Framework	bkab1	In Progress	Sprint 3	Backend
26	Create Listing Class	akardorff	Todo	Sprint 2	Backend
27	Create Account Class	akardorff	Todo	Sprint 2	Backend
28	Write up API URLS in DjangoRest	bkab1	In Progress	Sprint 3	Backend
29	Create Image Class	akardorff	Todo	Sprint 2	Backend
30	Create use case diagram	akardorff, bkab1, an	Done	Sprint 2	All
31	Pagination in DRF	bkab1	Todo	Sprint 3	Backend
32	Add end-points for our framework	bkab1	Todo	Sprint 3	Backend
33	Flesh out UI to have full functionality	EfficiencyIsKey and Sa	Future Sprints	Sprint 2	Frontend
34	Refine styling	EfficiencyIsKey and Sa	Future Sprints	Sprint 2	Frontend
35	Start implementation of test cases	EfficiencyIsKey and Sa	Future Sprints	Sprint 2	Frontend
36	Start implementation of authentication	EfficiencyIsKey and Sa	Future Sprints	Sprint 2	Frontend
37	Start implementing real API handler	EfficiencyIsKey and Sa	Future Sprints	Sprint 2	Frontend
38	Refine test cases	EfficiencyIsKey and Sa	Future Sprints	Sprint 2	Frontend
39	Update class diagram	EfficiencyIsKey and Sa	Future Sprints	Sprint 2	Frontend
40	Create data flow diagram	EfficiencyIsKey and Sa	Future Sprints	Sprint 2	Frontend

41	Implement search functionality	EfficiencyIsKey and Sa	Future Sprints	Sprint 2	Frontend
42	Implement search filtering functionality	EfficiencyIsKey and Sa	Future Sprints	Sprint 2	Frontend
43	Implement listing preview	EfficiencyIsKey and Sa	Future Sprints	Sprint 2	Frontend
44	Implement listing page	EfficiencyIsKey and Sa	Future Sprints	Sprint 2	Frontend
45	Implement contact form	EfficiencyIsKey and Sa	Future Sprints	Sprint 2	Frontend
46	Implement sign in form	EfficiencyIsKey and Sa	Future Sprints	Sprint 2	Frontend
47	Implement sign up form	EfficiencyIsKey and Sa	Future Sprints	Sprint 2	Frontend
48	Implement profile page	EfficiencyIsKey and Sa	Future Sprints	Sprint 2	Frontend

Sprint Backl... | Board View | Sprint 1 | Sprint 2 | Sprint 3 | Frontend | Backend | + New view

sprint:"Sprint 1" 15

	Title	Assignees	Status	Sprint	Subteam
3	Create SQL database	mkimdd	Done	Sprint 1	Backend
4	Create Template Django REST Framework Project	bkab1	Done	Sprint 1	Backend
5	Add Node.js and React.js project in subfolder	EfficiencyIsKey and Sa	Done	Sprint 1	Frontend
7	Create basic database component	mkimdd	Done	Sprint 1	Backend
8	Create basic frontend router component	SamJones329	Done	Sprint 1	Frontend
9	Create basic UI components (search bar/filters, listing, profile display, nav bar)	EfficiencyIsKey	Done	Sprint 1	Frontend
11	Create class diagram	akardorff, bkab1, Eff	Done	Sprint 1	All
12	Create ER diagram for database	akardorff, bkab1, an	Done	Sprint 1	Backend
13	Create styling for basic UI components	EfficiencyIsKey and Sa	Done	Sprint 1	Frontend
14	Create additional UI components and subcomponents	EfficiencyIsKey and Sa	Done	Sprint 1	Frontend
15	Design documents	akardorff, bkab1, Eff	Done	Sprint 1	All
16	Create Requirements Specification documents	akardorff, bkab1, Eff	Done	Sprint 1	All
17	Design frontend test plans	EfficiencyIsKey and Sa	Done	Sprint 1	Frontend
18	Design backend test plans	akardorff, bkab1, an	Done	Sprint 1	Backend
19	Create use cases	akardorff, bkab1, Eff	Done	Sprint 1	All

Tech details:

- ➔ The DatabaseManager interacts with a MySQL database. MySQL was chosen as the database technology due to its fast performance, programmer-friendly connection capability in Python through the mysql.connection framework, and external access capability for easy verification and modification of database state. Using this database, we can store the accounts of ColumbusList users, their listings, their wishlist listings, and the images associated with those listings.
- ➔ MySQL - A SQL database management system for implementing a relational database.
- ➔ MySQL Connector/Python - A self contained Python driver for communicating with MySQL servers.
- ➔ Django - A python web framework based on the model-view-template architecture.
- ➔ Django REST Framework - A python toolkit for building web APIs in Django.
- ➔ React.js - A declarative, component-based front-end library/framework for providing views based on application state and which is meant to be integrable into any project.
- ➔ React Router - A client side router for use with React.js.
- ➔ Node.js - A JavaScript runtime built on Chrome's V8 JavaScript Engine. Used for development of Frontend.
- ➔ TypeScript - Strict syntactical superset of JavaScript and adds optional static typing to the language. TypeScript is designed for the development of large applications and trans-compiles to JavaScript.

- ➔ Sassy CSS - CSS language extension
- ➔ Font Awesome - Icon font

Tools:

- ➔ Google Drive - Online storage for we use for storing milestone deliverables
- ➔ Discord - VoIP, direct messaging, and content distribution platform we use for sharing information and virtual meetings/correspondence
- ➔ Figma - Web-based design software we use for frontend design and whiteboarding
- ➔ Lucid Chart - Online tool for creating diagrams such as ER, use case, and class
- ➔ GitHub - Remote service for software development and version control we use for storing code repository
- ➔ VS Code - Editor/IDE used for development