CommuneKit

307 Design Document

CommuneKit

Team 35
Edward Bi
Rohan Gaddam
Nikhil Pisolkar
Ash Wang
Mingfan Xie

Index

Purpose

Over the course of a month, I might expect to need a large fold-out table once, maybe twice. My neighbors probably do the same, which begs the question: do both of us need our own folding tables? There are a variety of goods whose patterns of use make individual ownership unnecessarily expensive, redundant, and inefficient. Lawnmowers, steam cleaners, party supplies, tools, and a variety of other items. CommuneKit provides a platform for members of the same community to list items that would otherwise be collecting dust. Users can create an account and list items, and register prices of things, borrow from friends, and search within a radius to find items near them. Unlike Craigslist/Facebook Marketplace, CommuneKit is tailored towards use within smaller communities, so there is not a focus on shipping/long range borrowing, and works on reviews and community consensus to work as a primarily a free to use site.

Requirements

Functional Requirements

As a User, I would like to:

1. **Login and Account Management**
    - Land on a screen prompting for login or account creation.
    - Create a new account.
    - Log in to the website with a password-protected account.
    - Log in and use the site concurrently with other users.
    - Edit my user profile, including username, email, profile picture, address, and bio.
    - Upload a profile picture via file upload.
    - Have a default profile picture if none is uploaded.
    - Delete my account.
    - Log out of my account.
2. **Profile Features**
    - Upload profile pictures of any size with compression (optional).
    - Upload a profile picture using the camera (optional).
    - View another user's profile by clicking on it.
    - Send messages to other users.
    - Receive messages from other users.
    - See the message history between me and another user.
    - View reviews that other users have given me.
    - Block messages from other users (optional).
    - Rate and review other users.

   - Report other users for abusive behavior.
   - View admin warnings about my account or items.
   - Be notified when suspended or blocked, and have my account impacted accordingly.
3. **Item Browsing and Interaction**
   - View a general map area where items or users are located (optional).
   - View items available within my community (optional).
   - Click on a home button to navigate to the homepage.
   - See a menu at the top of the screen for site navigation.
   - Click a back button to return to the previous page.
   - Click on a specific item to view more details.
   - Refresh the page to view any updated information.
   - After clicking on a specific item, save my place on the page/scrollbar when returning to the previous page.

As a Borrower, I would like to:

1. **Search and Sort Items**
   - View all items within a 1-10 mile radius.
   - Sort items by map distance.
   - Sort items by exact keyword match.
   - Sort items by 'best' match.
   - Sort items by ratings.
   - View how many times an item has been borrowed.
   - Sort by driving distance/drive time (optional).
   - Filter items by category.
   - Scroll easily through a list of items that match my search criteria.
2. **View Item Details**
   - See ratings for a listed item.
   - See images for a listed item.
   - Read descriptions for a listed item.
   - See when the listed item was posted.
   - Read reviews for a listed item.
   - Read reviews for the lender of a listed item.
   - See the delivery/pickup preferences of the lender.
3. **Save and Manage Items**
   - Save items of interest.
   - Unsave items.
   - View saved items in a 'saved' folder.
   - Remove all items from my saved folder with one button press.
4. **Borrowing Management**

- See the items I am currently borrowing.
- See the date when I started borrowing an item.
- View items I have borrowed in the past.

5. **Communication and Reviews**
    - Send messages to the lender (optional).
    - Receive messages from the lender (optional).
    - Review items after I borrow them.
    - Review lenders after borrowing from them.
    - Report accounts that engage in inappropriate behavior.
    - Report inappropriate items.
6. **Notifications and Reimbursements**
    - Reimburse the lender for damages (optional).
    - Receive a notification if an item I am looking for is listed (optional).
    - Receive a notification if an item I am looking for has been borrowed by another user (optional).

As a Lender, I would like to:

1. **Item Posting and Management**
    - Post items that I'd like to lend.
    - Upload images for my posts.
    - Create descriptions for my posts.
    - Edit the descriptions of my posts.
    - Delete my posts.
    - Edit or update a listing for an item.
    - Upload multiple images for an item at once.
    - Upload images of any size, compressing when necessary (optional).
    - Add a security deposit requirement for an item (optional).
2. **Borrower and Item Tracking**
    - See a current list of users borrowing my items.
    - See which user is borrowing which item.
    - View the history of items I have lent in the past.
    - View items I am currently lending out.
3. **Borrower Interaction and Reviews**
    - Review borrowers after lending them an item.
    - Report borrowers who interact inappropriately.
    - Receive notifications when a report I filed has been received.
    - Receive notifications when a report I filed has been processed or acted on.
4. **Item Organization and Sorting**
    - View all my listed items with their information and status on one page.
    - Sort lent items by lend date.

  - Sort lent items by expected return date.
  - Sort items in search by listing date.
  - Sort items in search by last lend date.
  - Sort items in search by last return date.
5. **Notifications and Reimbursement**
   - Receive notifications if my item has been reported or taken down.
   - Arrange for reimbursement for damages to my items (optional).
   - Receive payment for deposits or reimbursements (optional).
   - Receive automated reminders when an item is due to be returned soon.
   - Send automated reminders to borrowers if the return date is approaching or has passed.
   - Receive a notification if an item has been requested for extension by the borrower (optional).
   - Be notified if an item is marked as unavailable temporarily or during specific time frames.
   - Be notified of any user feedback or ratings on my items, including suggestions for improving listings.

As an Admin, I would like to:

1. **Account and Report Management**
   - Log in to an admin account.
   - View a list of all reports.
   - Sort the report list by date and time (ascending and descending).
   - Take down inappropriate posts.
   - Suspend user accounts.
   - Permanently ban user accounts.
   - Delete user accounts.
   - Create and send a message to a user when their account is removed.
   - Create and send a message to a user when an item is removed.
   - Be notified when an account or item is reported.
   - Receive email notifications for critical reports.
   - Look up a user by their username.
   - View messages sent by reporters explaining the reasons for reports.
   - Search for specific users.
   - View all reports submitted by a specific user.
   - See a history of reports made against a user or an item.
2. **Post and User Moderation**
   - Temporarily disable user accounts pending investigation, with an option to restore.

As a Project Owner/Manager, I would like to:

1. **Admin Management**
    - Log in with an owner username and password.
    - View a list of all admin accounts.
    - Create new admin accounts.
    - Update admin account information.
    - Delete admin accounts.
    - View the actions taken by all admin accounts.

As a Developer, I would like to:

1. **Code Testing and Version Control**
    - Have unit tests for critical features and components.
    - Establish secure and consistent communication protocols between the server and client-side of the application.
    - Maintain consistent version control for the codebase.
    - Use encryption for storing sensitive user information.
    - Implement input validation to prevent security vulnerabilities.

Non-functional Requirements

Architecture

- The backend will be constructed using **Java** with **JDBC** to connect to a **MySQL** database. The frontend will be built with **React** and **HTML/CSS**. The system architecture will be designed to allow the frontend and backend to work independently, with clear API contracts ensuring seamless communication between the two.
- The codebase will be maintained in a private **GitHub** repository, ensuring proper version control. **Branching** strategies like **feature branches** and **pull requests** will be used to manage code updates efficiently.

Security

- **Auth0** will be used for handling authentication and securing user sign-in information such as username, email, and password. It will also manage secure API access for authenticated users.
- **Prepared statements** and **input validation** will be utilized throughout the application to prevent security risks like **SQL injection** and **XSS attacks**.
- **Database permissions** will be properly set to control access to sensitive data, ensuring that only authorized users can view or edit specific data fields.
- A **password hashing** mechanism (e.g., bcrypt or similar) will be used to securely store user passwords in the database, making sure no plain-text passwords are saved.

Performance

- The web application should be able to handle multiple concurrent queries, with a target of up to **50 queries per second** under normal load conditions.
- When an item is officially borrowed, deleted, or updated, the system should reflect these changes on newly loaded pages within **2000ms** to ensure a responsive user experience.
- A performance monitoring plan will be in place to periodically test and optimize system response times, ensuring consistent performance as the platform scales.

User Experience

- The web app will focus on providing a simple, **easy-to-navigate** interface. Key actions, such as listing, borrowing, and searching for items, will be intuitive, even for users with limited technical experience.
- **Buttons** and other interactive elements will be clearly labeled and easily accessible. Form inputs will provide clear feedback in case of errors or missing information, improving the overall user experience.
- **Consistency in design** will be ensured throughout the platform to avoid user confusion. The flow from one page to another will be logically structured to minimize user friction.
- If time allows, the design will be refined for a more visually appealing interface, though the focus remains on functionality over aesthetics.

Hosting

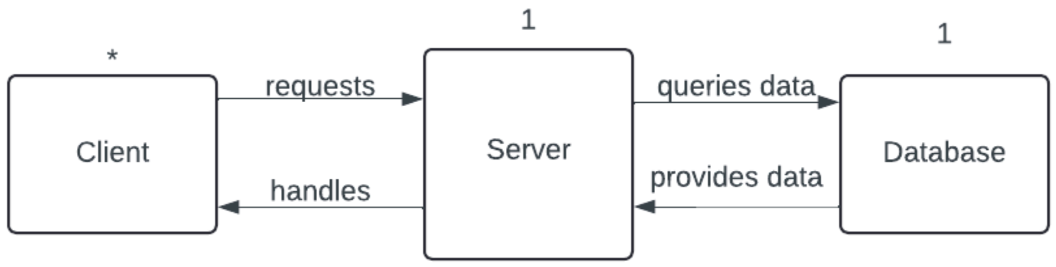
- Initially, the system will be hosted **locally** to support development and testing. Detailed instructions will be provided for setting up and running the local environment, including configuration files and environment variables.
- **Environment-specific configurations** will be maintained to easily switch between local development and production environments when scaling to a cloud platform (such as Google Cloud App Engine) in the future, if required.

- The system will include basic **backup procedures** to ensure that data is regularly saved and can be restored in the event of an unexpected system failure or crash.

Design Outline

This project will follow the client-server model and allow clients to use a web app to access the databases, in which user data and images are stored, through a server. The server should be able to support concurrent use from multiple clients. Data access and modification should be synchronized for all users.

High-Level Overview



Client

- The client consists of a web interface that will allow users to create and log in to password-protected accounts, and perform all the necessary user interactions
- The client will send authentication and data requests to the database through the server, also communicating with Auth0 authentication API for secure token-verified communication
- The client will retrieve data from server responses and convert it to a user-friendly format

Server

- The server will accept requests and forward them to the database
- The server will send the data retrieved from the database response back to the client

Database

- The database will store all data, including that pertaining to user accounts and items
- The database will handle queries from the server and return the appropriate data

Detailed Overview

Design Issues

Functional Issues

1. Should users be required to list an item for lending before borrowing?
    a. Option 1: Yes
    b. Option 2: No

CommuneKit is intended as a way to facilitate sharing within a community. While it is best for everyone in the site to engage equally in lending and borrowing, it may not be best to enforce any strict rules on mandating item listing before lending, as this may decrease the size of the potential user base.

2. Should users be able to request payment for borrowing?
    a. Option 1: Always
    b. Option 2: Never
    c. Option 3: Sometimes

As CommuneKit may grow in size and scope, payment for items may be a feature that gets added on eventually. However, for now the basis of the website is on borrowing. There are other places that people can look for renting items from people. In the case of CommuneKit, for now we enable people to lend as much as they are able and willing to. The return or incentive here is moreso in community involvement and the ability to borrow from others.

3. Should users be able to borrow without explicit owner approval?
    a. Option 1: Never
    b. Option 2: Only on items that owners explicitly designate
    c. Option 3: **Always**

Unlike Amazon or Ebay, CommuneKit generally prioritizes sharing within communities. As a result, shipping and handling is generally arranged by the lenders and borrowers. Also, items are always returned to the original owner. As a result, it makes most sense to enforce that both lender and borrower must mutually wish to share the item. A lender will always have the ability to decide whom they are lending their item to.

4. Should an admin role exist? And what privileges will they have?

In this case, an admin should exist for moderation purposes. Admin accounts exist separate from the normal user. Admins are not users with extra privileges. Rather, they are a unique entity that after logging in, has the single function of handling and managing accounts and items in response to reports from users. CommuneKit does have a large amount of user-user interaction as well as user-generated content like item descriptions and pictures. Thus, there is a need for moderation of this content and interaction.

5. How should location data be queried?
    a. **Option 1: Self-reported**
    b. Option 2: Periodically
    c. Option 3: On server refresh

CommuneKit relies on borrowers looking for items within their community. As a result, we choose to have people enter a permanent address and use that to ground locations of items. In the future, it may make sense to add functionality to distribute a user's items in different places if, for example, they have multiple houses or storage spaces separate from their permanent address.

Non-Functional Issues

1. What language will be used to implement the server?
    a. **Java**
    b. C++
    c. Node

Based on our experience with different platforms and languages, it makes sense for us to implement the backend in Java. There will be a solid class architecture that will fit well into Java's strictly object-oriented structure

2. How will user authorization be implemented
    a. Proprietary
    b. **auth0**

For CommuneKit, we choose to use a third-party user and transaction authentication API, Auth0. This will allow us to ensure secure communication between client and server, and keep user location data safe. This level of security will help isolate authentication, which is a complex, dynamic environment better left to software maintained by trusted, verified services.

3. What frontend framework will we use?
    a. Flask
    b. React
    c. Angular

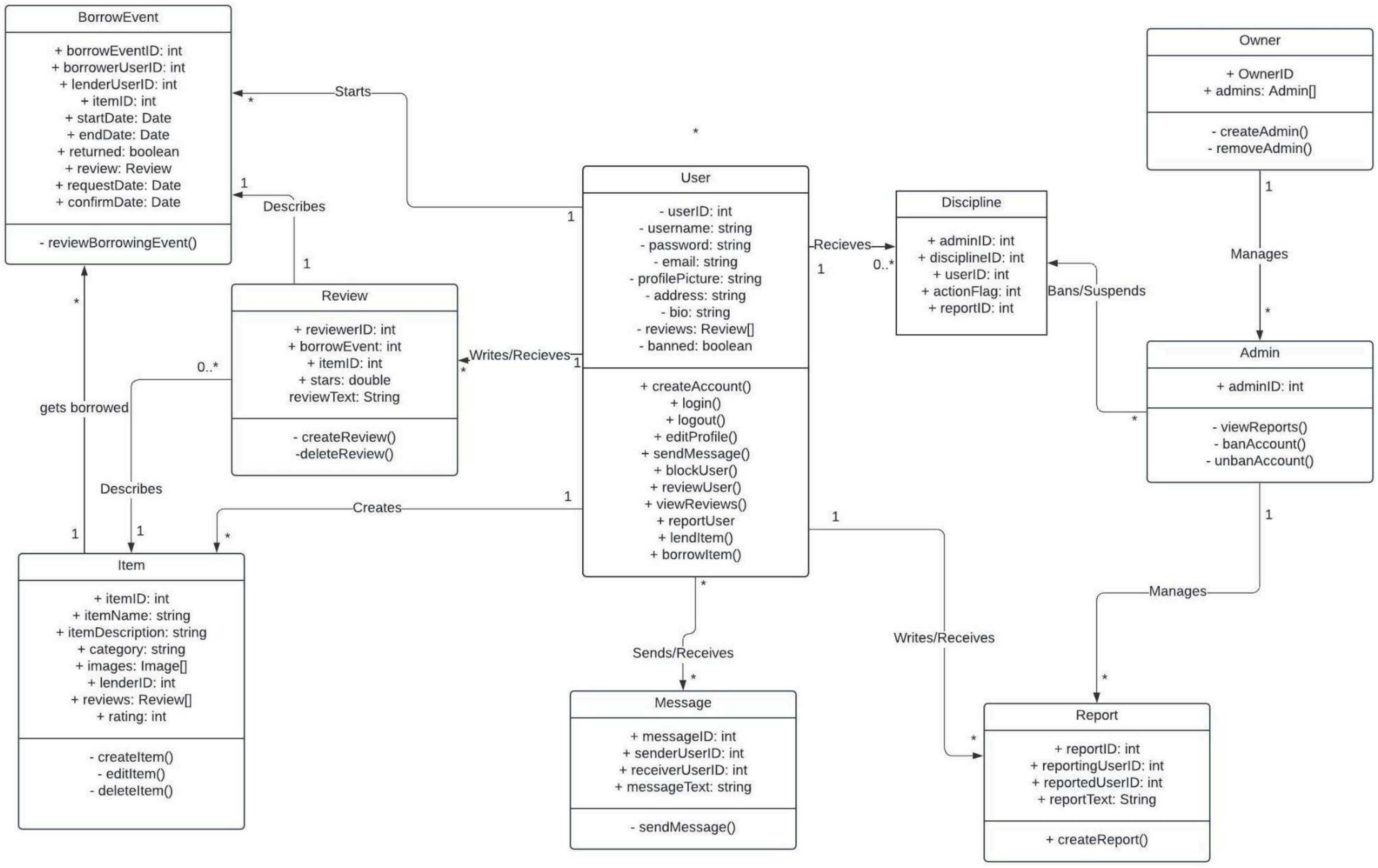
Given our personal levels of experience, we did not want to mess with raw html and JavaScript. React provides a clean, simple framework to arrange components and is relatively easy to learn. Building components like chatting, profile and user blurbs, as well as item listings will make portions of the website more reusable and repeatable.

4. What database will we use?
    a. Option 1: MySQL
    b. Option 2: SQLite
    c. Option 3: MongoDB
    d. Option 4: Graph DB

Here, we choose to stick with a relational database. MySQL will allow us to use stored procedures and prepared statements, which aren't options that are available to us in SQLite. We will be able to set up indexes to optimize processes like item and photo retrieval to speed execution of what would otherwise be costly processes.

Design Details

Class-Level Breakdown



Significant Classes

User

- An user object is created when someone signs up for an account
- The user will have an unique userID
- The user must have an username, password, and email
- The user may have a profilePicture, address, and bio. If any are not filled in, a default value will be assigned instead
- Each user can receive Reviews which will be collected in an array. Upon creation, the array is empty
- Users can view each other's profile, including username, bio, and address
- Users can edit their bio, address, email, and profile picture upon viewing their own profile
- Users can view the reviews of other users or items
- Depending on the severity, users can be banned from certain interactions
- Users can send and receive messages from other users
- Users can file reports against other users or items

Item

- When a user posts an item to be lent out, an item object is created
- Each item will have an unique itemID
- Each item will have an item name
- Each item will belong to a category, which is selected when the user lists it
- Each item may have a description and up to five images
- Each item will also have a status: whether it's lent out or not
- When an user selects an item, they will be able to view the user that lists it
- Each item can receive reviews, and a rating will be calculated based on the reviews
- An item has no rating when it is first listed
- Items contain references to users owned by them as well as location data

Admin

- Admin account can only be created by owner
- Admin accounts will each have an unique admin ID

- Admin accounts can view items and profiles same as regular users
- They can also access a list of all reports that are filed
- Admin accounts can ban and delete user accounts
- Admin account can unban accounts
- Admins also have the ability to Discipline users by removing some of their items

BorrowEvent

- Whenever an item is borrowed, an unique borrowID is created
- Lender's userID and borrower's userID are both recorded
- ItemID is also recorded
- Item is marked as unavailable
- Time borrowed is marked down, as is expected return date
- Item will be marked as returned once lender receives the item back
- Lender will be able to view the borrowerID, and vice versa
- Item can only be borrowed after lender has confirmed the transaction
- Once item is marked as returned, both users may leave reviews
- Admins can access list of all borrow events

Review

- Each review will have an unique reviewID
- Review can be for either user or item
- If for user, target userID will be recorded
- Otherwise, itemID will be recorded
- Each review will have a stars value (the rating)
- Optionally, the review can contain a description/explanation

Report

- Users may leave reports on other users/items at any time
- Every report will have an unique report ID
- Reports are only visible by admins
- UserID/ItemID are recorded
- Optional text for explanations may be included
- User ID of the user filing the report is also recorded

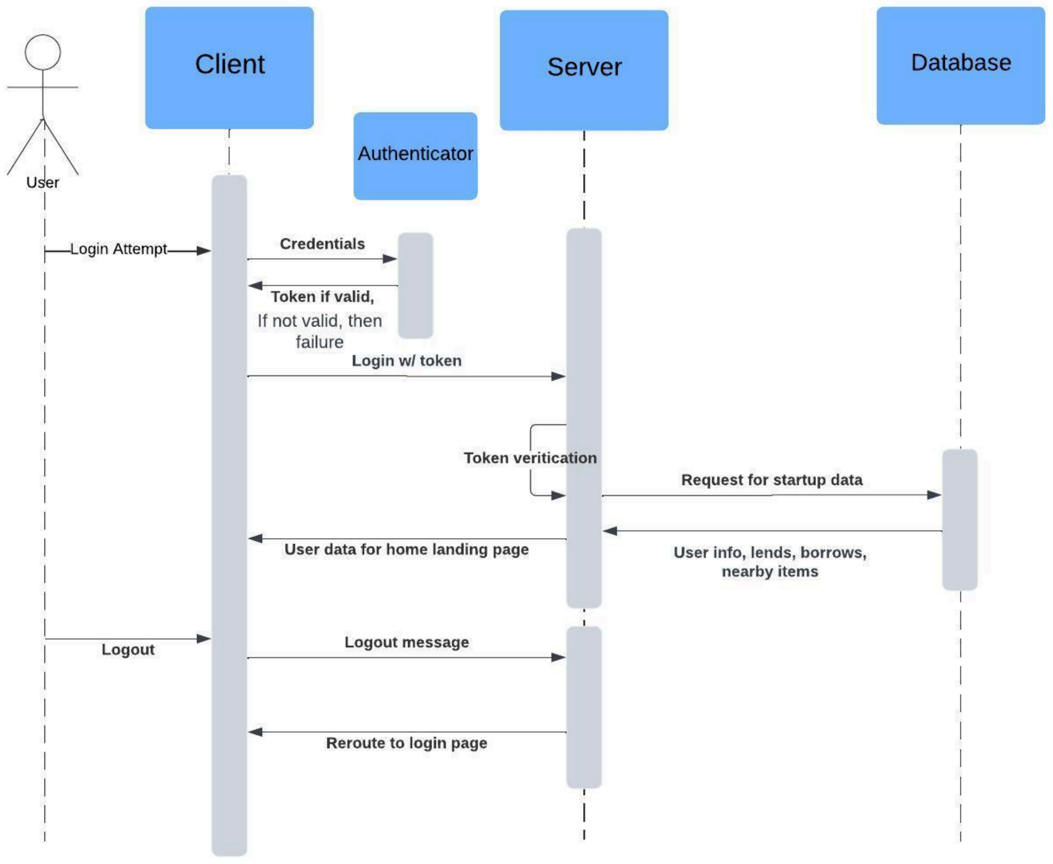
Message

- A message is created whenever an user sends a new message to someone else

- Each message has an unique message ID
- The sender’s userID and receiver’s userID are both recorded

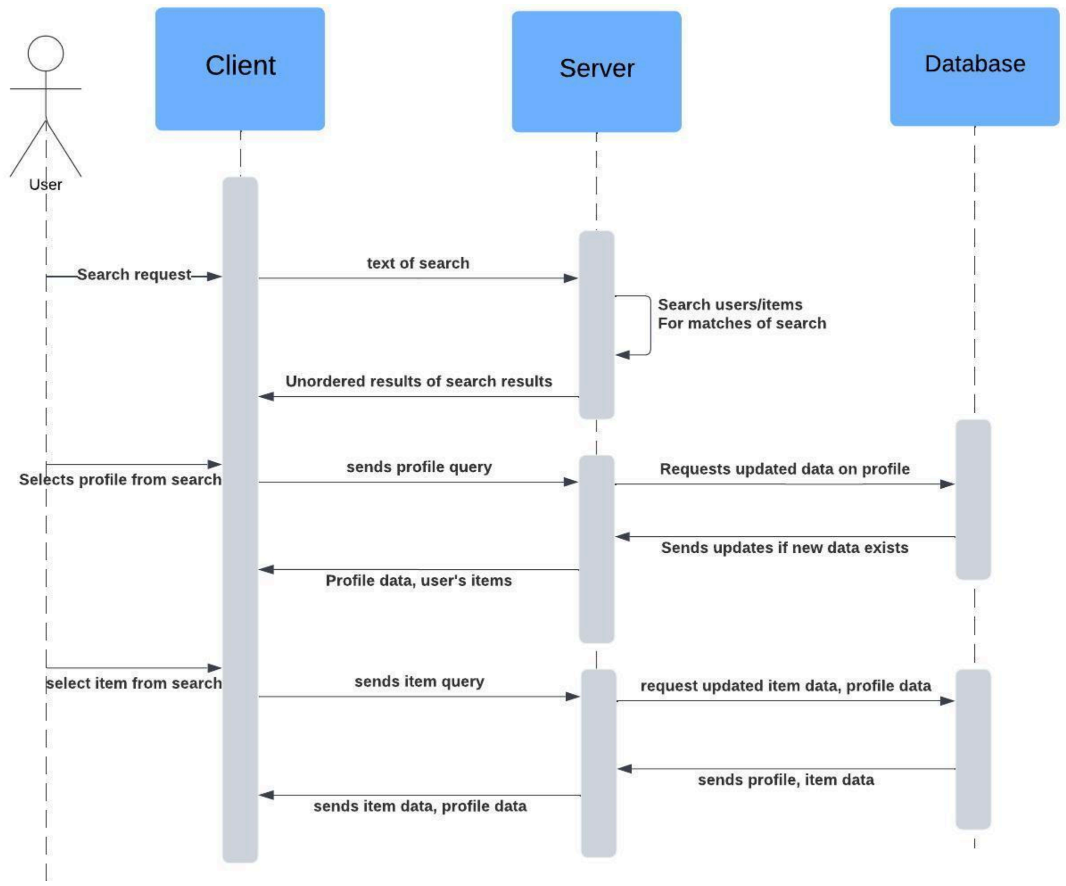
Sequences of Events

Account creation/login, logout



When a user lands on the CommuneKit page, they are directed to the login screen. They have the option of creating an account or logging in to an account. Either way, they will have their request sent to the Auth0 API, which will either authenticate the login request or verify the new login. Once that happens, the client will send that login information with a token. The server will use hash verification to verify the token sent by the client. After verification, the server will send the login/account data to the database for storage and then retrieve all relevant data for that user and their location. Then, the home page is sent to the client for display.

Search for item → select item

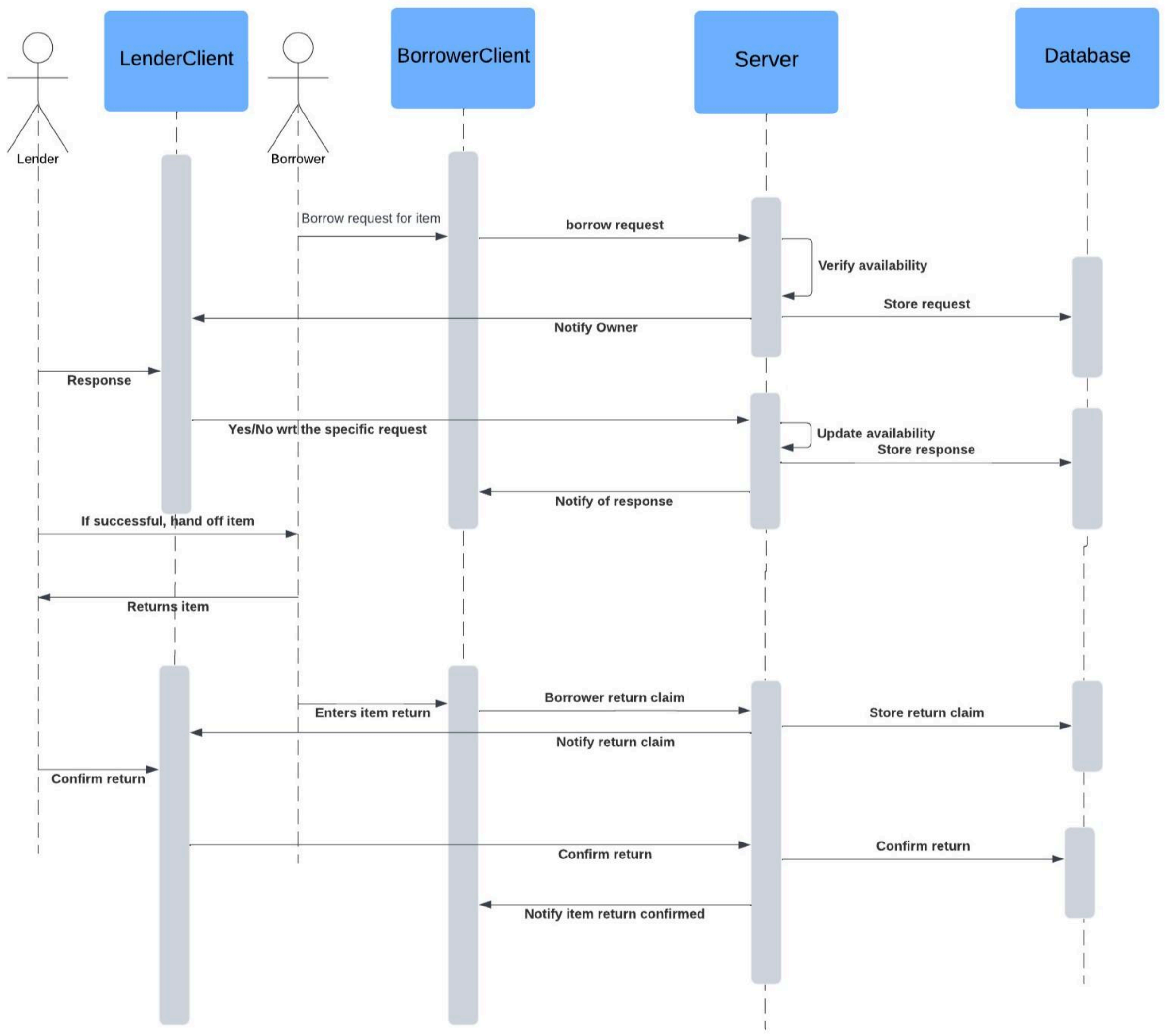

Client
Server
Database
User
Search request
text of search
Search users/items For matches of search
Unordered results of search results
Selects profile from search
sends profile query
Requests updated data on profile
Sends updates if new data exists
Profile data, user's items
select item from search
sends item query
request updated item data, profile data
sends profile, item data
sends item data, profile data

This sequence diagram outlines the process of a user searching for an item on the CommuneKit platform, selecting a profile associated with the search results, and finally choosing a specific item. It shows the interactions between the **User**, **Client**, **Server**, and **Database** during the search and item selection process.

1. **Search Request**: The user initiates a search query by entering the text of the item they want to borrow.
2. **Search Processing**: The server searches the database for users and items matching the search criteria and retrieves unordered search results.
3. **Display Search Results**: The server sends the search results to the client, displaying them to the user.
4. **Profile Selection**: The user selects a profile from the search results. The client sends a request to the server to fetch the profile data and the items listed by that user.
5. **Profile Data Retrieval**: The server queries the database for the profile data and listed items. If there are updates, the server retrieves the latest profile information.
6. **Display Profile and Items**: The server sends the profile and item data to the client, allowing the user to view the items listed by the selected profile.

7. **Item Selection**: The user selects an item from the profile. The client sends a query to the server to fetch specific item details.
8. **Retrieve Item Data**: The server requests the item data from the database and sends it back to the client.
9. **Display Item Data**: The client displays the specific item data and the profile information to the user.

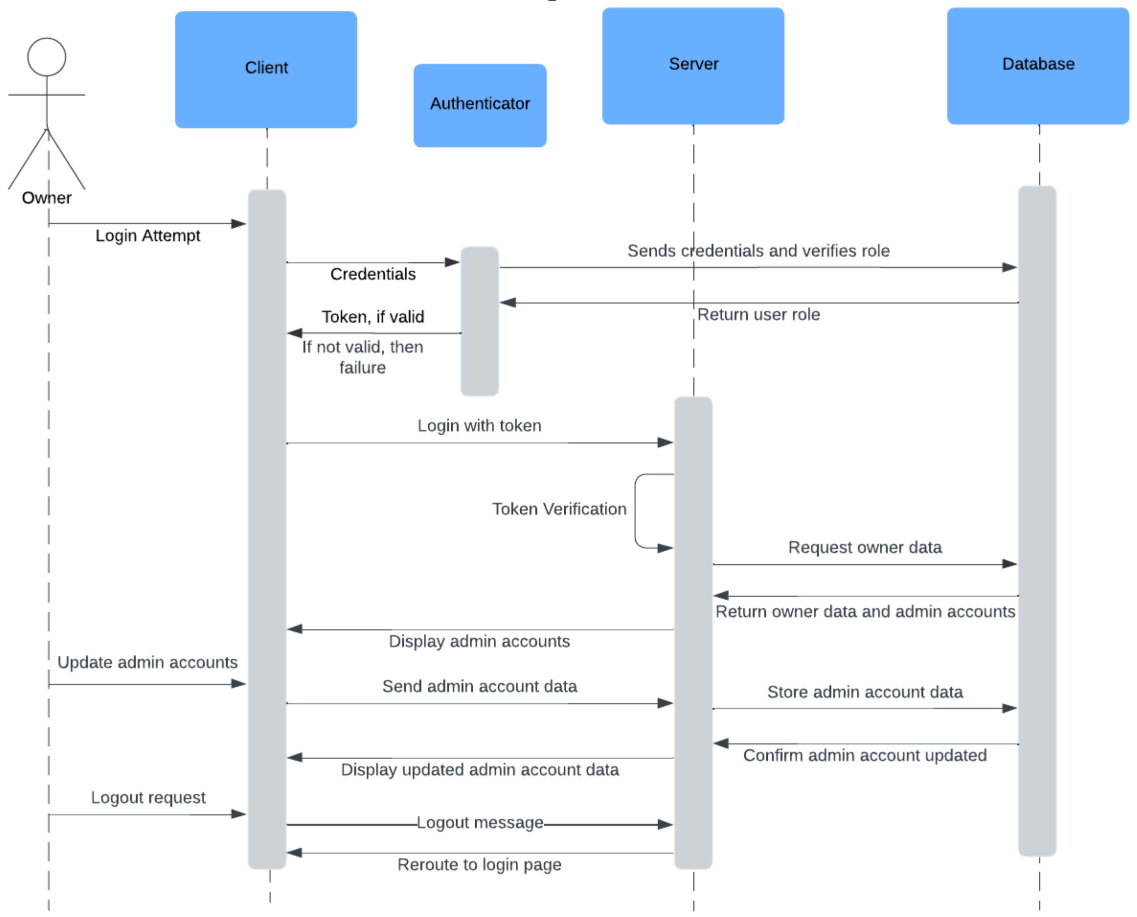
Borrowing an item



This sequence diagram illustrates the process of borrowing an item through the CommuneKit platform. It details the interactions between the borrower, lender, server, and database when a borrower requests to borrow an item from a lender.

1. **Borrow Request**: The borrower sends a borrow request for a specific item to the system.
2. **Verification**: The server verifies the item's availability by checking the database.
3. **Notify Lender**: If the item is available, the server notifies the lender of the request.
4. **Lender Response**: The lender can accept or reject the borrow request.
   - If accepted, the system updates the item's availability in the database.
5. **Borrower Notification**: The borrower is notified of the lender's decision.
   - If successful, arrangements are made to hand over the item.
6. **Item Return**: Once the borrowing period is over, the borrower initiates the return process.
7. **Return Confirmation**: The system verifies the return, and the lender confirms the item has been returned.

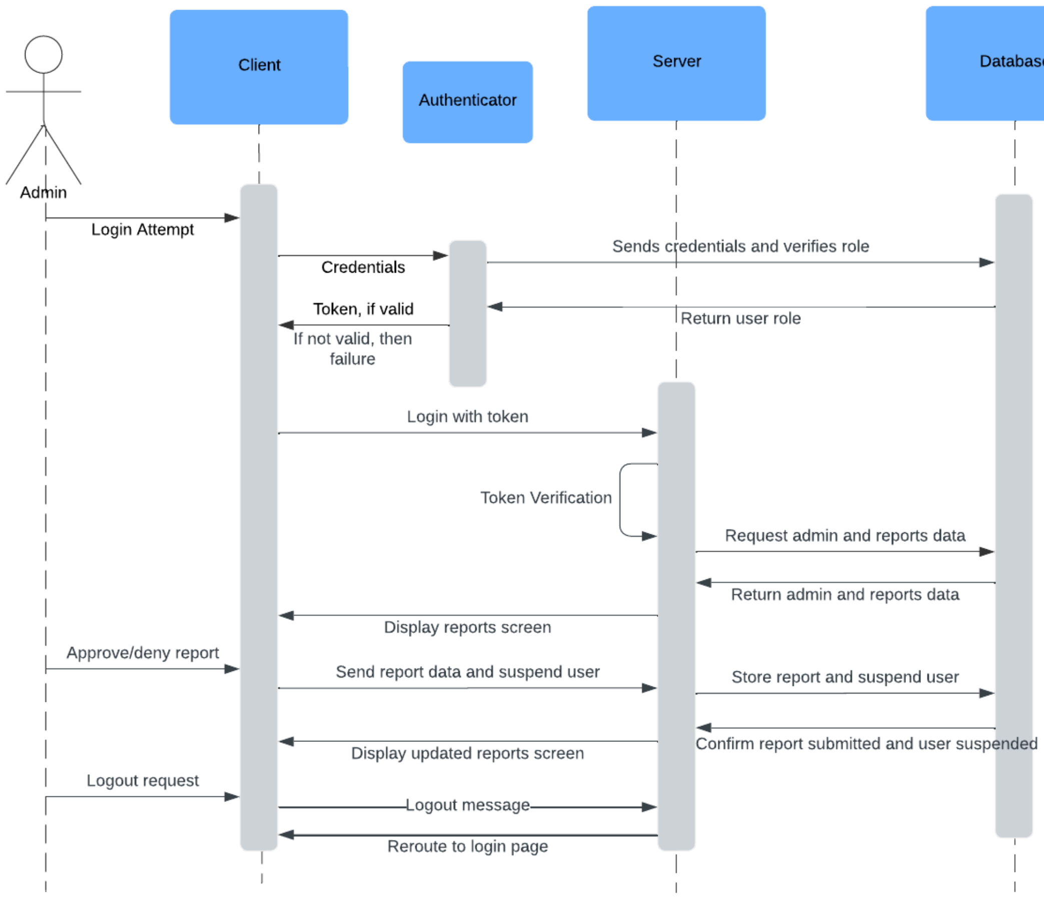
8. Owner updates admin accounts



This sequence diagram shows how the **owner** of the CommuneKit platform updates admin accounts. It demonstrates the interactions between the **Client**, **Authenticator**, **Server**, and **Database** during the process of logging in, retrieving admin account data, making updates, and confirming those changes.

1. **Login Attempt**: The owner initiates a login request through the client, providing credentials to the system.
2. **Authentication**: The **Authenticator** verifies the credentials. If they are valid, a token is issued and returned to the client.
3. **Token Verification**: The **Server** receives the token and verifies the owner's role by checking the token's validity.
4. **Retrieve Admin Accounts**: The **Server** sends a request to the **Database** for the owner's account data and admin accounts associated with the system.
5. **Display Admin Accounts**: Once the admin account data is retrieved, the server sends the data back to the client for the owner to view and update.
6. **Update Admin Accounts**: The owner updates the admin account data through the client interface, and the changes are sent back to the server.
7. **Store Updates**: The server stores the updated admin account data in the database, confirming the changes.
8. **Logout**: Once the update is complete, the owner can send a logout request, and the system reroutes them back to the login page.

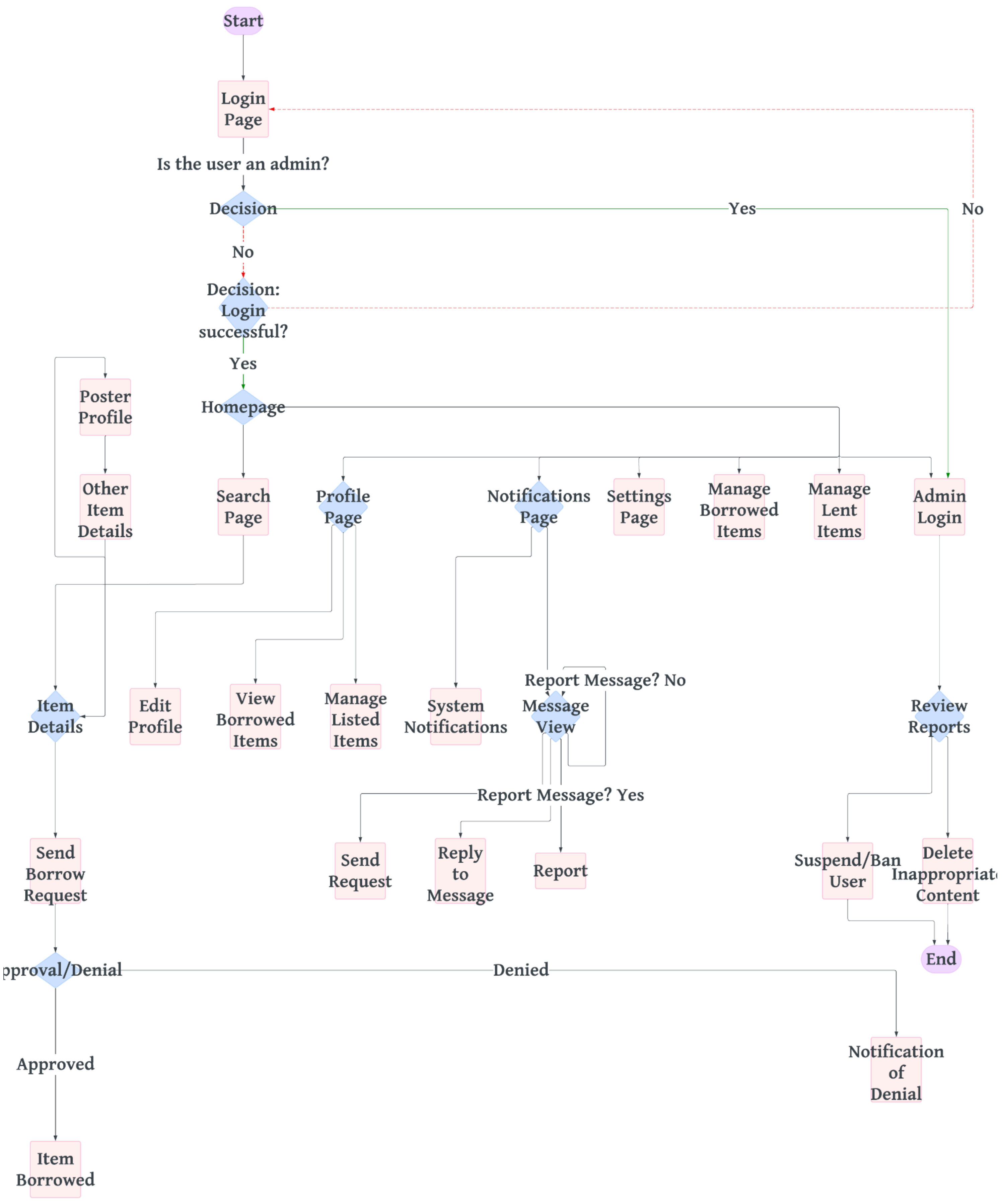
Admin confirms a report


Client
Authenticator
Server
Database
Admin
Login Attempt
Credentials
Sends credentials and verifies role
Token, if valid
Return user role
If not valid, then failure
Login with token
Token Verification
Request admin and reports data
Return admin and reports data
Display reports screen
Approve/deny report
Send report data and suspend user
Store report and suspend user
Confirm report submitted and user suspended
Display updated reports screen
Logout request
Logout message
Reroute to login page

This sequence diagram illustrates the process in which an **admin** confirms or denies a report on the CommuneKit platform. It outlines how the admin logs into the system, retrieves report data, and takes action on a specific report, either by approving or denying it. If approved, the admin can also choose to suspend the user involved in the report.

1. **Login Attempt**: The admin initiates a login attempt by providing credentials through the client interface.
2. **Authentication**: The **Authenticator** verifies the credentials. If valid, the system issues a token, which the server uses to authenticate the admin's session.
3. **Token Verification**: The server verifies the admin's role and retrieves report data from the **Database**.
4. **Display Reports**: The retrieved report data is displayed to the admin, who can review the details.
5. **Approve/Deny Report**: The admin decides to approve or deny the report. If approved, the system may also suspend the reported user.
6. **Store Report and Suspend User**: If the admin approves the report, the **Server** stores the report decision and, if applicable, sends a request to suspend the user.
7. **Update Reports Screen**: After the decision is made, the updated report status is displayed to the admin.
8. **Logout Request**: The admin can log out once the action is complete, and the system reroutes them back to the login page.

Activity/State Diagrams



Description:

This flowchart represents the overall navigation and decision flow for both users and admins on the CommuneKit platform. The diagram starts from the **Login Page** and outlines the various paths users and admins can take based on their actions, including item borrowing, profile management, notification handling, and admin-specific functions.

User Flow:

1. **Login Process**:
    - The user begins at the **Login Page**.
    - **Decision Node**: "Is the user an admin?"
        - If **No**, the user continues with the standard login flow.
        - If **Yes**, the user is redirected to the **Admin Login** flow (explained below).
    - After login credentials are validated, the user proceeds to the **Homepage**.
2. **Homepage Navigation**: From the **Homepage**, the user has multiple options:
    - **Search Page**: The user can search for items to borrow.
    - **Profile Page**: The user can view and edit their profile.
    - **Notifications Page**: Users can check system notifications and messages.
    - **Settings Page**: Manage account settings.
    - **Manage Borrowed Items**: View and manage items they are currently borrowing.
    - **Manage Lent Items**: Manage the items they have lent out to others.
3. **Search and Borrowing Flow**:
    - From the **Search Page**, the user can select an item to view its details.
    - The user can also view the **Poster Profile**, which includes other items listed by the same user.
    - After selecting an item, the user sends a **Borrow Request**.
    - **Approval/Denial Node**: If the request is approved, the item is marked as borrowed. If denied, the user is notified of the denial.
4. **Messaging and Notifications**:
    - From the **Notifications Page**, the user can either check system notifications or view messages.
    - If a message is inappropriate, the user can report it.
        - **Report Message? Node**: If **Yes**, the report is submitted. If **No**, the user can reply or send a request.

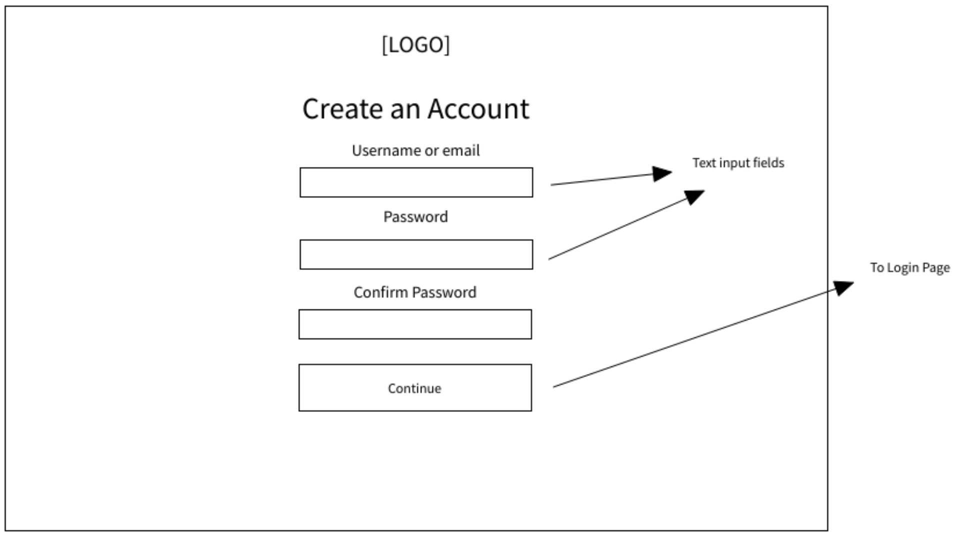
Admin Flow:

1. **Admin Login**:
    - If the user is an admin, they are directed to the **Admin Login** page.
    - After logging in, the admin has access to their dashboard, where they can:
        - **Review Reports**: View and manage user-submitted reports.

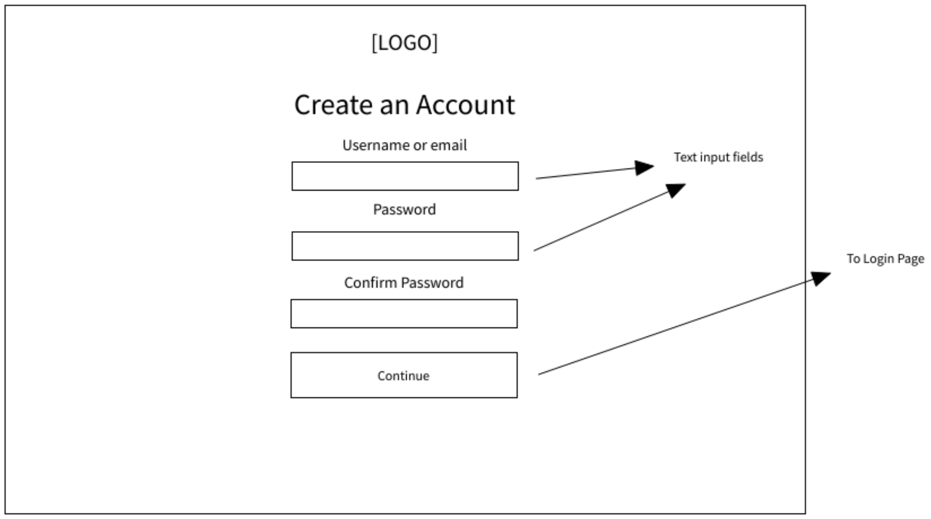
2. **Admin Actions**:
   - Admins can take actions on user reports, including:
     - **Suspend or Ban User**: If a user is found violating rules.
     - **Delete Inappropriate Content**: Remove any reported content that violates platform policies.
3. **End Process**:
   - After completing actions, both users and admins can log out, concluding the session.

UI Mockups

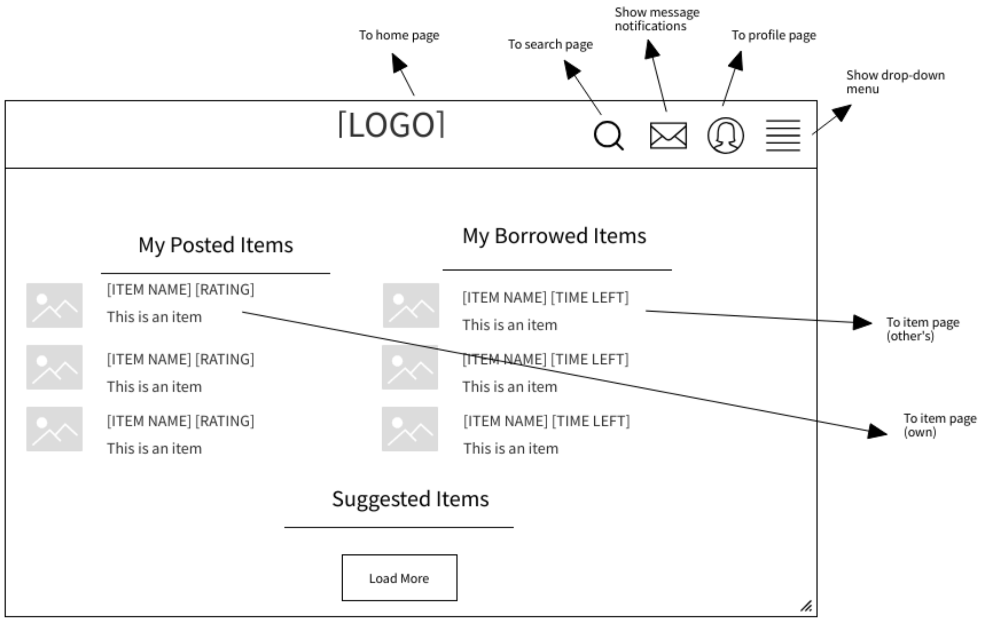
Login Page



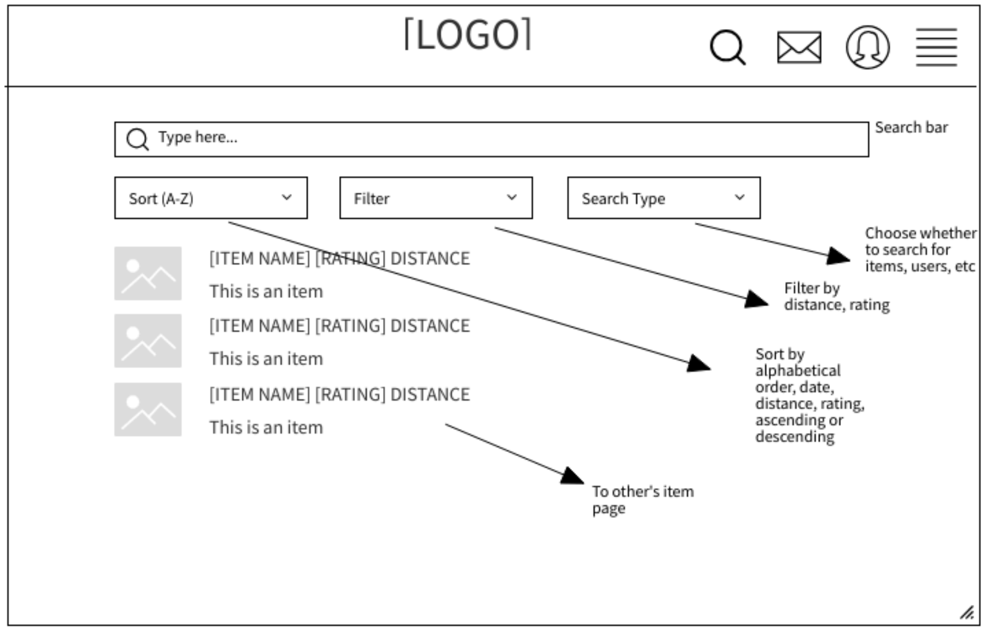
Create Account Page



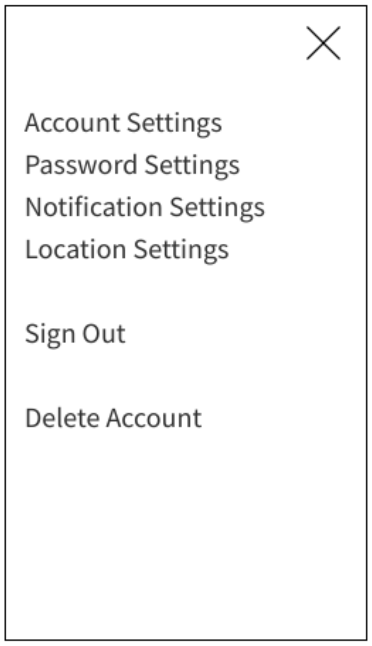
Home Page



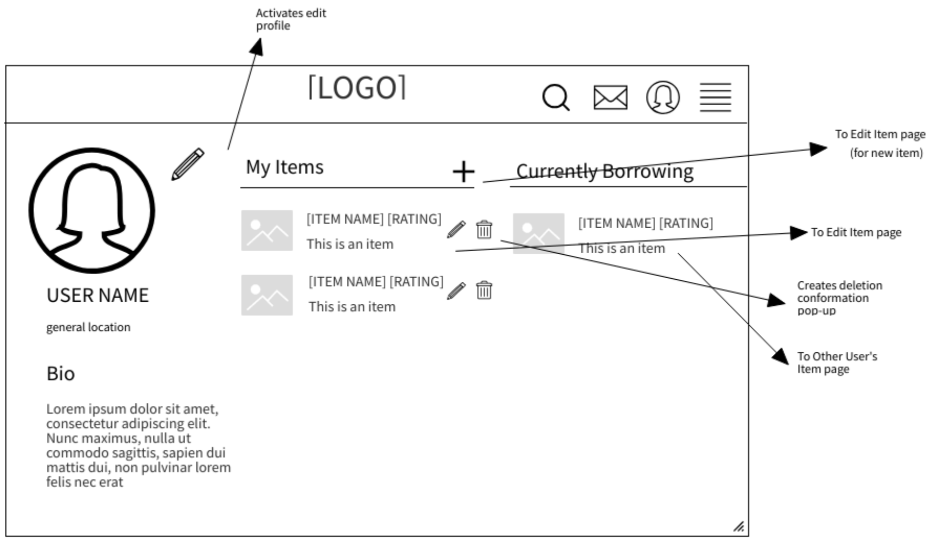
Search Page


[LOGO]
Type here...
Search bar
Sort (A-Z)
Filter
Search Type
Choose whether to search for items, users, etc
[ITEM NAME] [RATING] DISTANCE
This is an item
Filter by distance, rating
[ITEM NAME] [RATING] DISTANCE
This is an item
Sort by alphabetical order, date, distance, rating, ascending or descending
[ITEM NAME] [RATING] DISTANCE
This is an item
To other's item page

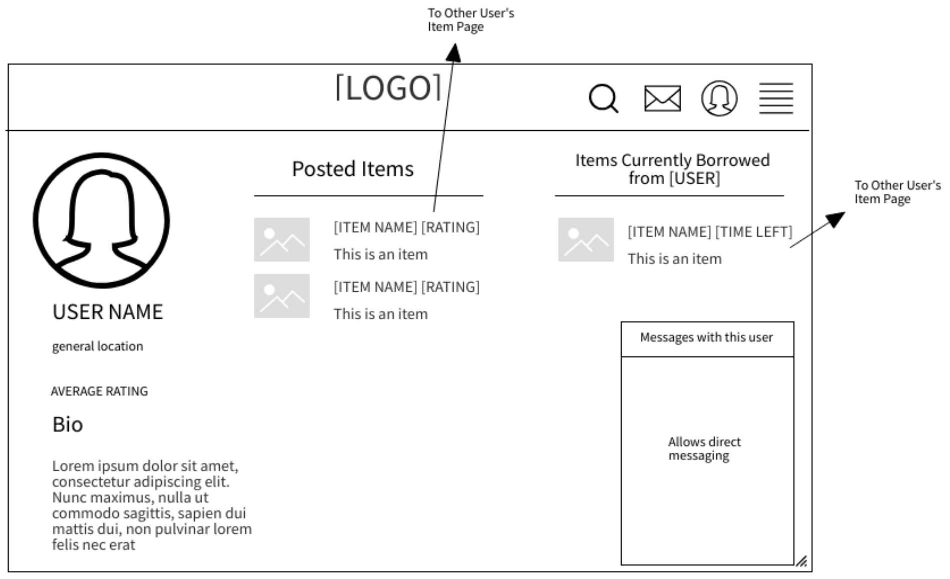
Menu Dropdown



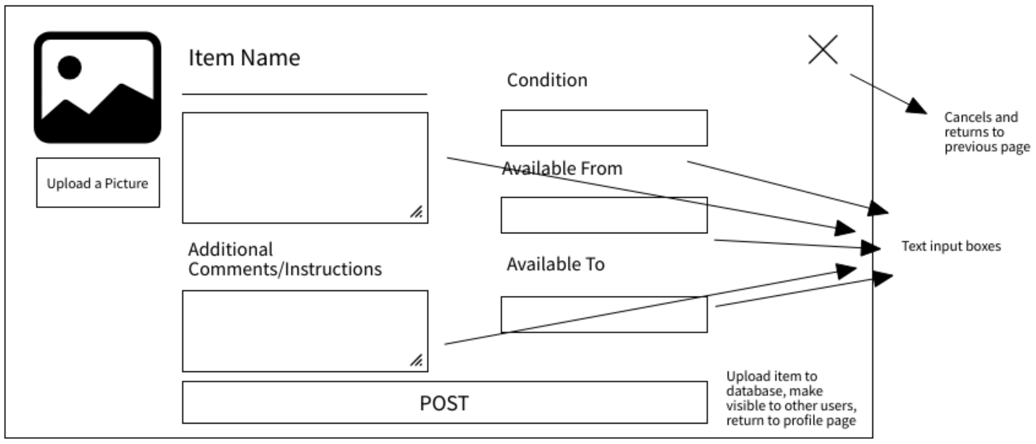
Own Profile Page


Activates edit profile
[LOGO]
To Edit Item page (for new item)
My Items
Currently Borrowing
[ITEM NAME] [RATING]
This is an item
[ITEM NAME] [RATING]
This is an item
[ITEM NAME] [RATING]
This is an item
To Edit Item page
Creates deletion conformation pop-up
To Other User's Item page
USER NAME
general location
Bio
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc maximus, nulla ut commodo sagittis, sapien dui mattis dui, non pulvinar lorem felis nec erat

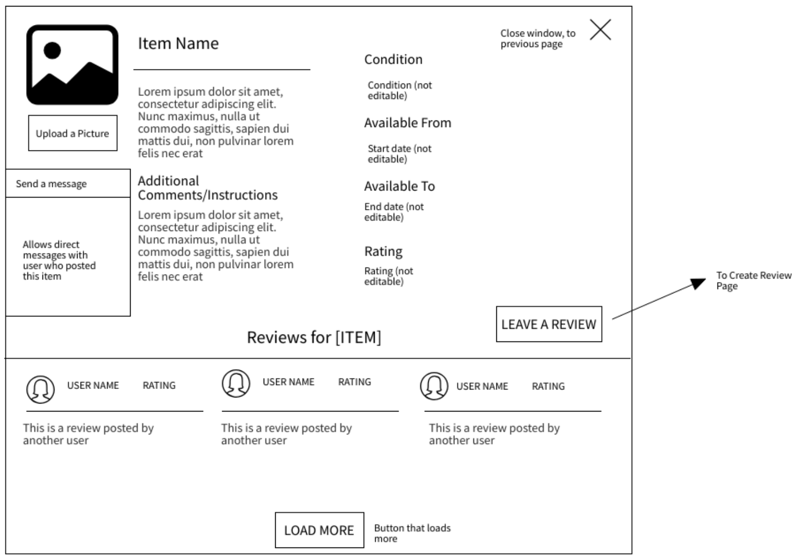
Other User's Profile Page


To Other User's Item Page
[LOGO]
USER NAME
general location
AVERAGE RATING
Bio
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc maximus, nulla ut commodo sagittis, sapien dui mattis dui, non pulvinar lorem felis nec erat
Posted Items
[ITEM NAME] [RATING]
This is an item
[ITEM NAME] [RATING]
This is an item
Items Currently Borrowed from [USER]
[ITEM NAME] [TIME LEFT]
This is an item
To Other User's Item Page
Messages with this user
Allows direct messaging

Create/Edit Item Page

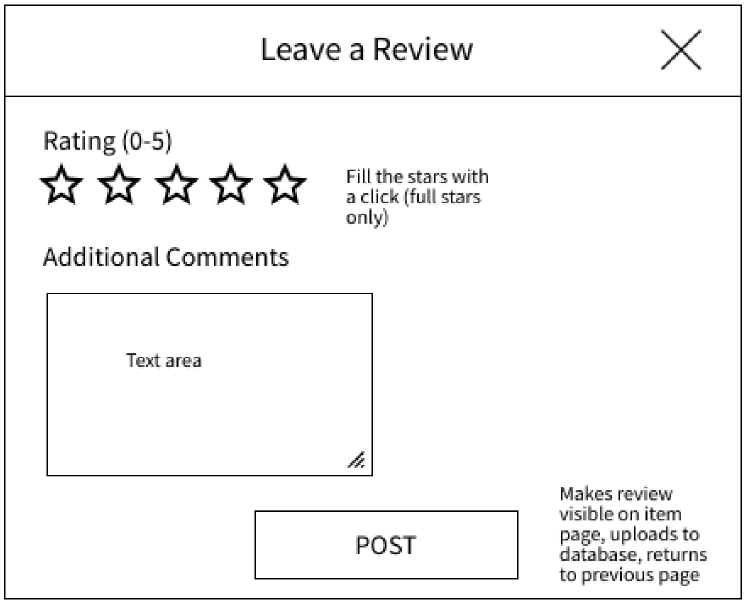

Upload a Picture
Item Name
Additional Comments/Instructions
Condition
Available From
Available To
POST
Cancels and returns to previous page
Text input boxes
Upload item to database, make visible to other users, return to profile page

Item Page

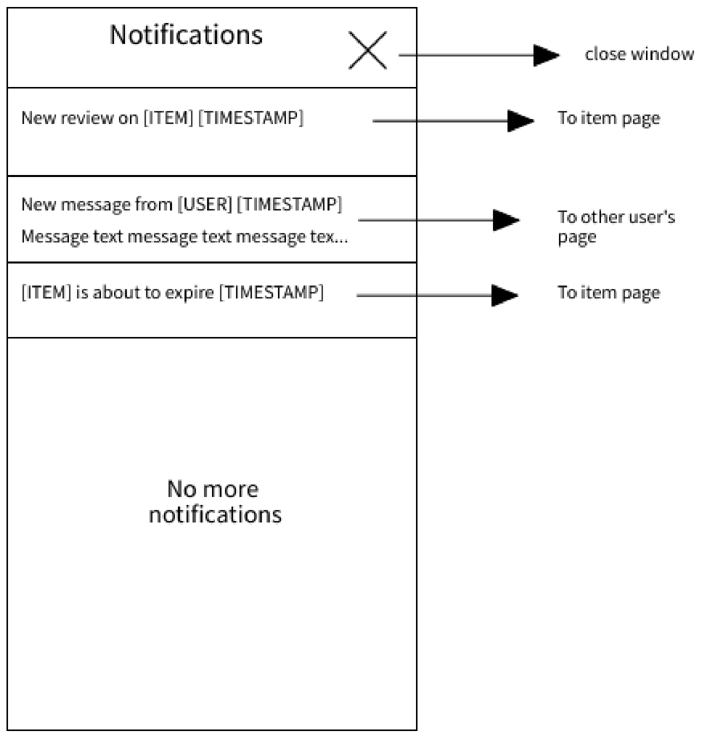

Item Name
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc maximus, nulla ut commodo sagittis, sapien dui mattis dui, non pulvinar lorem felis nec erat
Upload a Picture
Send a message
Allows direct messages with user who posted this item
Additional Comments/Instructions
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc maximus, nulla ut commodo sagittis, sapien dui mattis dui, non pulvinar lorem felis nec erat
Condition
Condition (not editable)
Available From
Start date (not editable)
Available To
End date (not editable)
Rating
Rating (not editable)
Close window, to previous page
LEAVE A REVIEW
To Create Review Page
Reviews for [ITEM]
USER NAME
RATING
This is a review posted by another user
LOAD MORE
Button that loads more

Label

Create a Review Page


Leave a Review
Rating (0-5)
Fill the stars with a click (full stars only)
Additional Comments
Text area
POST
Makes review visible on item page, uploads to database, returns to previous page

Notifications Dropdown


Notifications
close window
New review on [ITEM] [TIMESTAMP]
To item page
New message from [USER] [TIMESTAMP]
Message text message text message tex...
To other user's page
[ITEM] is about to expire [TIMESTAMP]
To item page
No more notifications