SW Engineering CSC 648/848 - Milestone 2

SecondChance: The Eco-Friendly Rental Platform!

CSC 648 - Section 01 - Team 05
10/19/2024

Team Members & Roles

Student Name	Student's Role
Parth Desai	Team Lead / PM
Pedro Grande	Front-end Developer
Charvi Sharma	Scrum Master
Josue Hernandez	Git Master
Andre Velarde	Back-end Lead
Preet Vithani	Front-end Lead
Hsin-Ying Tsai	Back-end Developer

Data Definitions V2

1. User Data

This table defines all the data items related to users, including metadata and privileges. These are critical to the platform as they help in managing access, listings, and transactions.

Field	Data Type	Description
`user_id`	UUID	A unique identifier for each user. This is the primary key.
`first_name`	String	The user's first name.

Field	Data Type	Description
`last_name`	String	The user's last name.
`email`	String	The user's email address. Used for login and notifications. Should be unique across the platform.
`password_hash`	String	A hashed version of the user's password for security.
`avatar_url`	String	URL to the user's profile picture. The picture formats allowed are `JPEG`, `PNG`, and the maximum size is 5MB.
`user_privilege`	Enum	Defines the user role: `admin`, `owner`, `renter`.
`created_at`	DateTime	The date and time the user account was created.
`updated_at`	DateTime	The date and time the user's details were last updated.
`last_login`	DateTime	The last time the user logged in.

2. Item Data

These fields define the data associated with items that users list on the platform for rent.

Field	Data Type	Description
`item_id`	UUID	A unique identifier for each item. This is the primary key.
`user_id`	UUID (FK)	Foreign key referring to the `user_id` of the owner.
`title`	String	Title of the item being listed (e.g., "Sofa for rent").

Field	Data Type	Description
`description`	Text	Detailed description of the item, including its features and condition.
`category`	Enum	Item category (e.g., `furniture`, `electronics`, `clothing`, `books`, `others`).
`rental_price`	Float	The rental price of the item per day in USD.
`condition`	Enum	Condition of the item: `new`, `like new`, `used`.
`quantity`	Integer	The number of units available for rent.
`location`	String	Location of the item for rent (city, country).
`image_url`	String[]	URLs to item images (multiple images allowed). Max of 5 images per item, formats: `JPEG`, `PNG`, max size 5MB.
`created_at`	DateTime	The date and time when the item was listed.
`updated_at`	DateTime	The date and time when the item was last updated.
`availability_status`	Enum	Status of the item: `available`, `rented`.

3. Chat Message Data

Defines the structure for messages exchanged between users in the platform's chat functionality.

Field	Data Type	Description
`message_id`	UUID	A unique identifier for each message. This is the primary key.
`conversation_id`	UUID	Foreign key referring to a conversation. Each conversation may have multiple messages.
`sent_by`	UUID (FK)	The user ID of the person sending the message.
`sent_to`	UUID (FK)	The user ID of the person receiving the message.
`body`	Text	The text content of the message.
`created_at`	DateTime	The timestamp when the message was sent.

4. Search Filters Data

Data used for filtering items when users search for listings.

Field	Data Type	Description
`category`	Enum	Filter by item category (e.g., `furniture`, `electronics`, `clothing`).
`location`	String	Filter by location (city, country).
`condition`	Enum	Filter by item condition: `new`, `like new`, `used`.
`price_min`	Float	Minimum price filter.
`price_max`	Float	Maximum price filter.

5. Transaction Data

Defines the details of transactions when users rent items.

Field	Data Type	Description
`transaction_id`	UUID	A unique identifier for each transaction. This is the primary key.
`item_id`	UUID (FK)	Foreign key referring to the `item_id` of the rented item.
`renter_id`	UUID (FK)	Foreign key referring to the `user_id` of the user renting the item.
`owner_id`	UUID (FK)	Foreign key referring to the `user_id` of the owner of the item.
`start_date`	Date	The start date of the rental period.
`end_date`	Date	The end date of the rental period.
`total_price`	Float	The total price for the rental period, calculated as `rental_price * number_of_days`.
`transaction_status`	Enum	Status of the transaction: `pending`, `completed`, `cancelled`.

6. Review Data

Defines data related to reviews for items and transactions.

Field	Data Type	Description
`review_id`	UUID	A unique identifier for each review.

Field	Data Type	Description
`transaction_id`	UUID (FK)	Foreign key linking the review to a specific transaction.
`reviewer_id`	UUID (FK)	Foreign key referring to the `user_id` of the person leaving the review.
`rating`	Integer	A rating between 1 and 5 stars.
`comment`	Text	The user's written feedback on the transaction or item.
`created_at`	DateTime	The timestamp when the review was submitted.

7. Notification Data

Defines data for notifications sent to users about updates, messages, or transactions.

Field	Data Type	Description
`notification_id`	UUID	A unique identifier for each notification.
`user_id`	UUID (FK)	Foreign key referring to the `user_id` of the recipient of the notification.
`message`	String	The content of the notification (e.g., "Your item has been rented").
`read_status`	Boolean	Indicates whether the notification has been read (`true`/`false`).
`created_at`	DateTime	The timestamp when the notification was sent.

8. Admin Data

Defines the structure for administrative users and their privileges.

Field	Data Type	Description
`admin_id`	UUID	A unique identifier for each admin user.
`user_id`	UUID (FK)	Foreign key referring to the `user_id` of the admin.
`role`	String	The role of the admin (e.g., `Super Admin`, `Moderator`).
`created_at`	DateTime	The date and time the admin account was created.
`updated_at`	DateTime	The date and time the admin's details were last updated.

Functional Requirements V2

1. User Management

R.1: User Registration and Authentication

- **R.1.1**: A new user must be able to register with a valid email, password, and name. (Priority 1)
- **R.1.2**: The system should verify the user's email during the registration process. (Priority 1)
- **R.1.3**: Users must be able to log in using their email and password. (Priority 1)
- **R.1.4**: Implement password hashing for security. (Priority 1)
- **R.1.5**: Users must be able to reset their password via email. (Priority 2)
- **R.1.6**: Users can update their profile details such as name, phone number, and avatar. (Priority 2)
- **R.1.7**: Admin users should have elevated privileges, allowing them to manage content on the platform. (Priority 1)

R.2: User Roles & Privileges

- **R.2.1**: Different roles (admin, renter, owner) must be clearly defined, with appropriate access controls in place. (Priority 1)
- **R.2.2**: Admins should be able to suspend or delete user accounts. (Priority 2)
- **R.2.3**: Owners should be able to list items for rent and manage those listings. (Priority 1)

- **R.2.4**: Renters can rent items and leave reviews. (Priority 1)

2. Item Management

R.3: Item Listings

- **R.3.1**: Owners must be able to create a listing for an item, including title, description, category, price, location, and upload images. (Priority 1)
- **R.3.2**: Listings must have a status to indicate if an item is available or rented. (Priority 1)
- **R.3.3**: Users must be able to update or delete their listings. (Priority 1)
- **R.3.4**: Each item must display detailed information, including reviews and rental history. (Priority 1)
- **R.3.5**: Items must be categorized into predefined categories: furniture, electronics, clothing, etc. (Priority 1)

3. Search & Filtering

R.4: Item Search and Filters

- **R.4.1**: Users must be able to search for items by category, location, and condition. (Priority 1)
- **R.4.2**: Users must be able to filter items by price range, availability, and condition. (Priority 1)
- **R.4.3**: Implement advanced search using Elasticsearch to make queries fast and efficient. (Priority 1)
- **R.4.4**: Users should be able to save their search preferences for future use. (Priority 2)

4. Renting and Transactions

R.5: Rental Process

- **R.5.1**: Renters must be able to select a start and end date for renting an item. (Priority 1)
- **R.5.2**: The system must calculate the total rental cost based on the duration and rental price. (Priority 1)
- **R.5.3**: Owners should receive notifications when their items are rented. (Priority 2)
- **R.5.4**: Renters must be able to cancel or extend their rentals, depending on availability. (Priority 2)
- **R.5.5**: Owners must be able to set limits for rental periods (min and max duration). (Priority 2)
- **R.5.6**: A secure payment gateway must be implemented for rental transactions. (Priority 1)

R.6: Transaction Management

- **R.6.1**: The system must store a history of all transactions for both renters and owners. (Priority 1)
- **R.6.2**: Users should be able to view their past rentals and active rentals. (Priority 1)
- **R.6.3**: Implement refunds or adjustments in the event of cancellations or issues. (Priority 3)
- **R.6.4**: All payments must be logged with details such as transaction date, amount, and status (pending, completed, or canceled). (Priority 1)

5. Chat Functionality

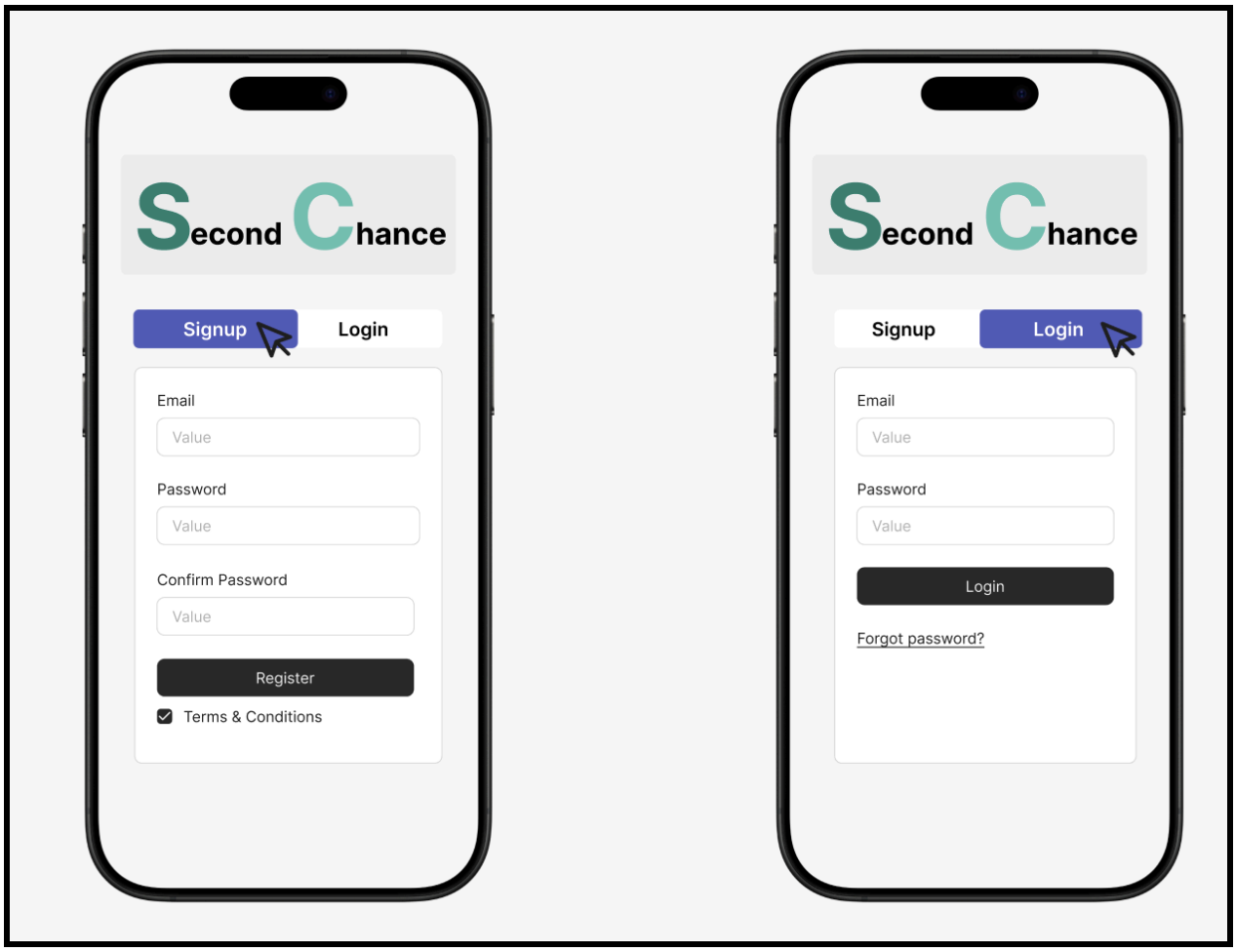
R.7: Real-Time Messaging

- **R.7.1**: Renters and owners must be able to send and receive messages in real-time via a chat interface. (Priority 1)
- **R.7.2**: The chat must display the user's avatar and timestamp for each message. (Priority 2)
- **R.7.3**: Unread messages should trigger notifications to users. (Priority 2)
- **R.7.4**: Users should be able to report inappropriate behavior or messages. (Priority 3)
- **R.7.5**: Messages must be stored in the database and retrievable at any time. (Priority 1)

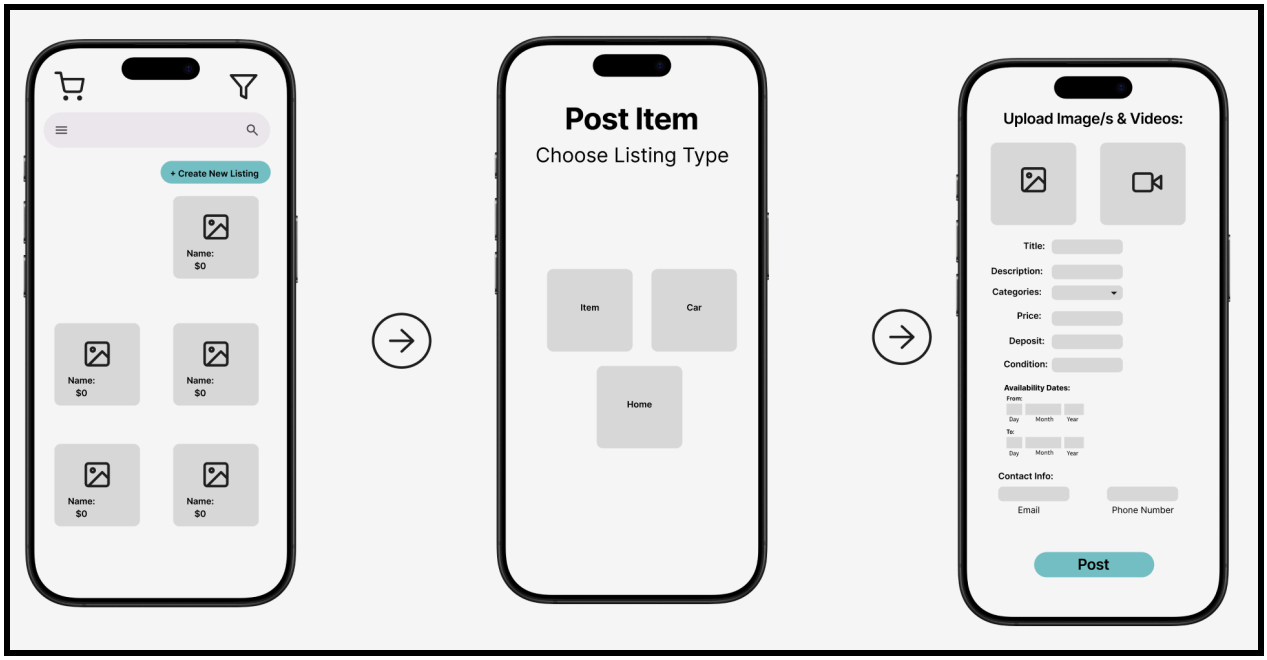
UI Mockups and UX Flows

Figma Link:
https://www.figma.com/design/Tu660uQUgxM8cPqHWkNbEl/Milestone-2-Worklfow?node-id=0-1&m=dev&t=w2ZQbERi83bcnlVY-1

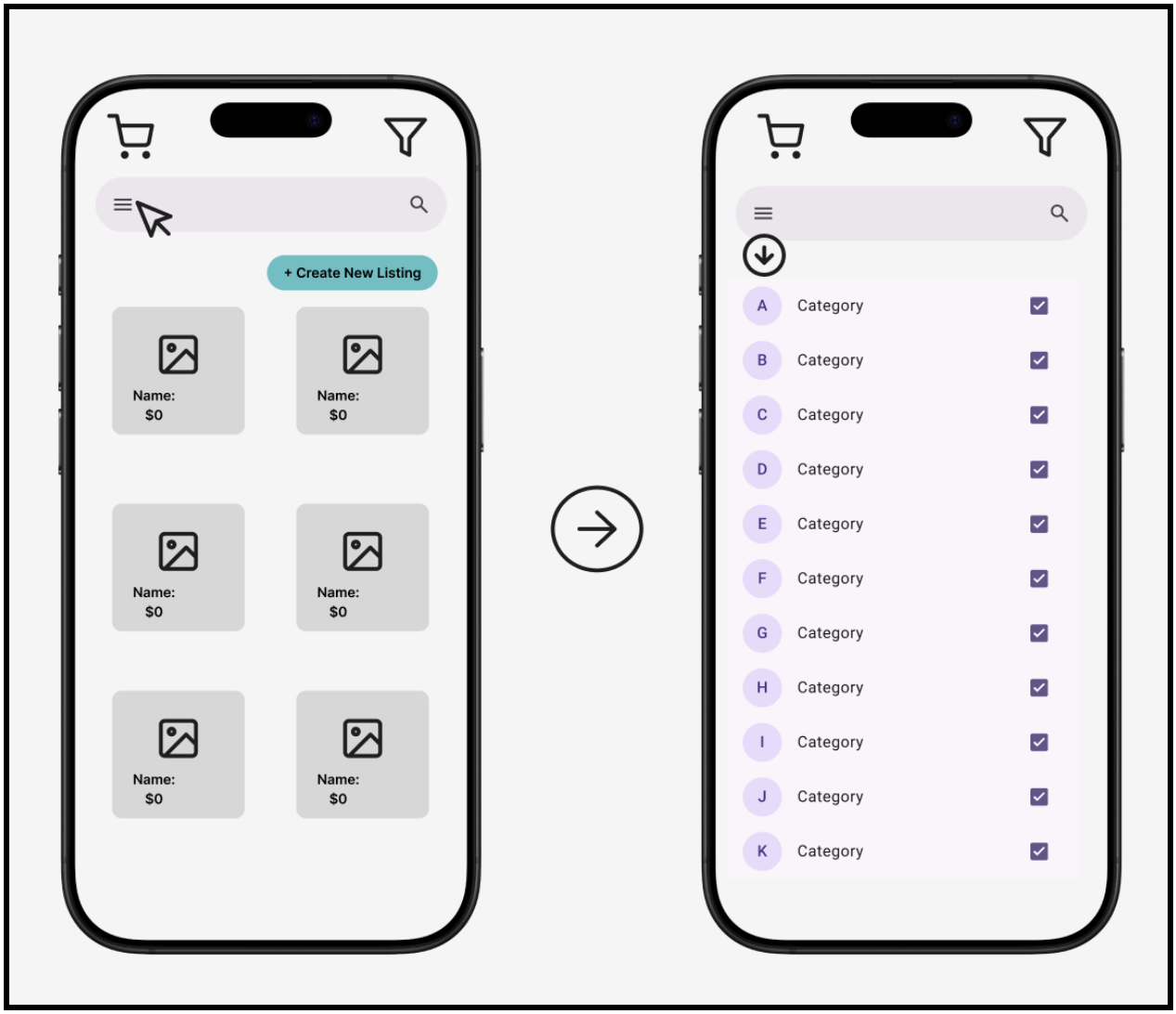
1. User Can Create Account & Login

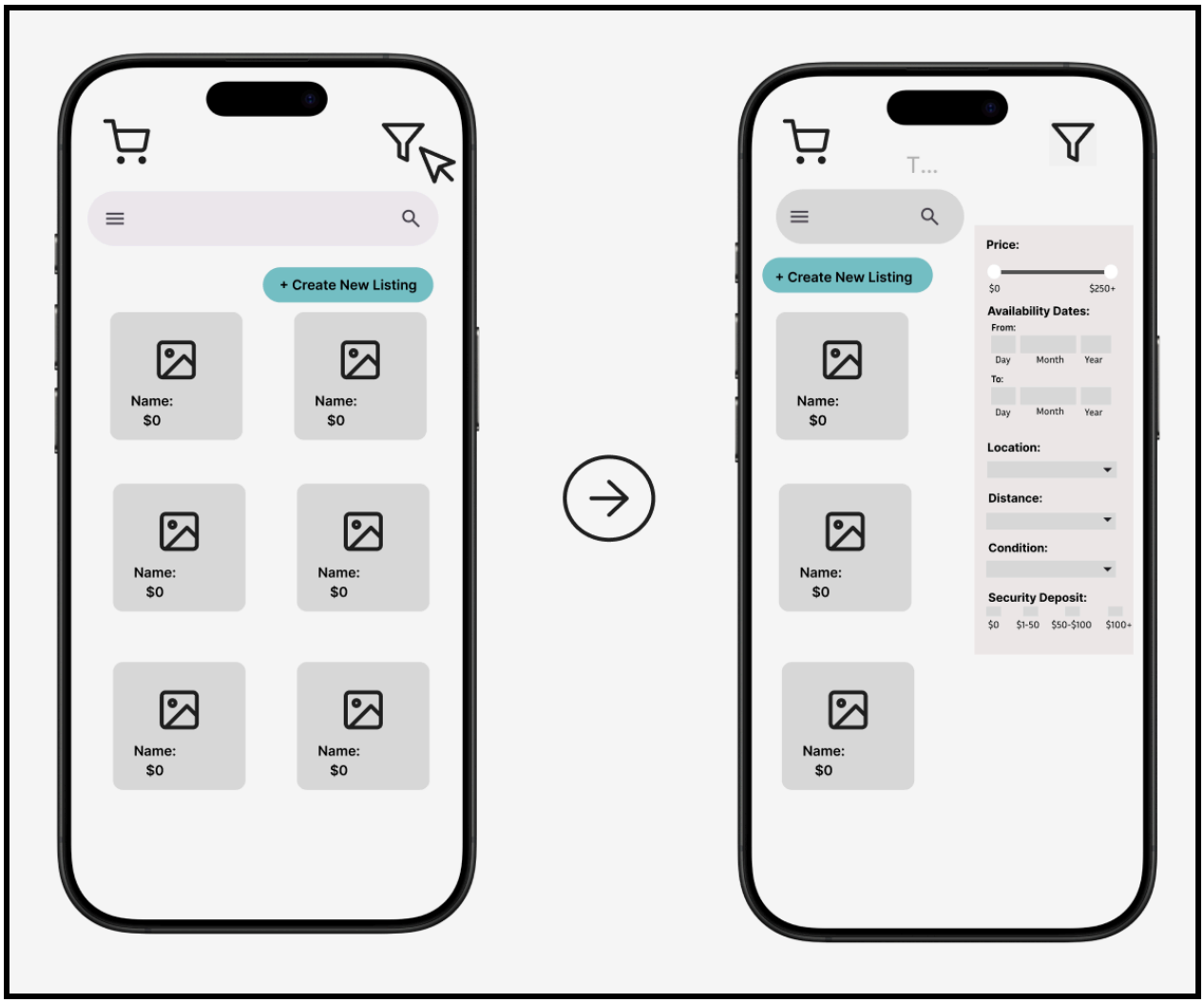


2. Post item/s for Rent

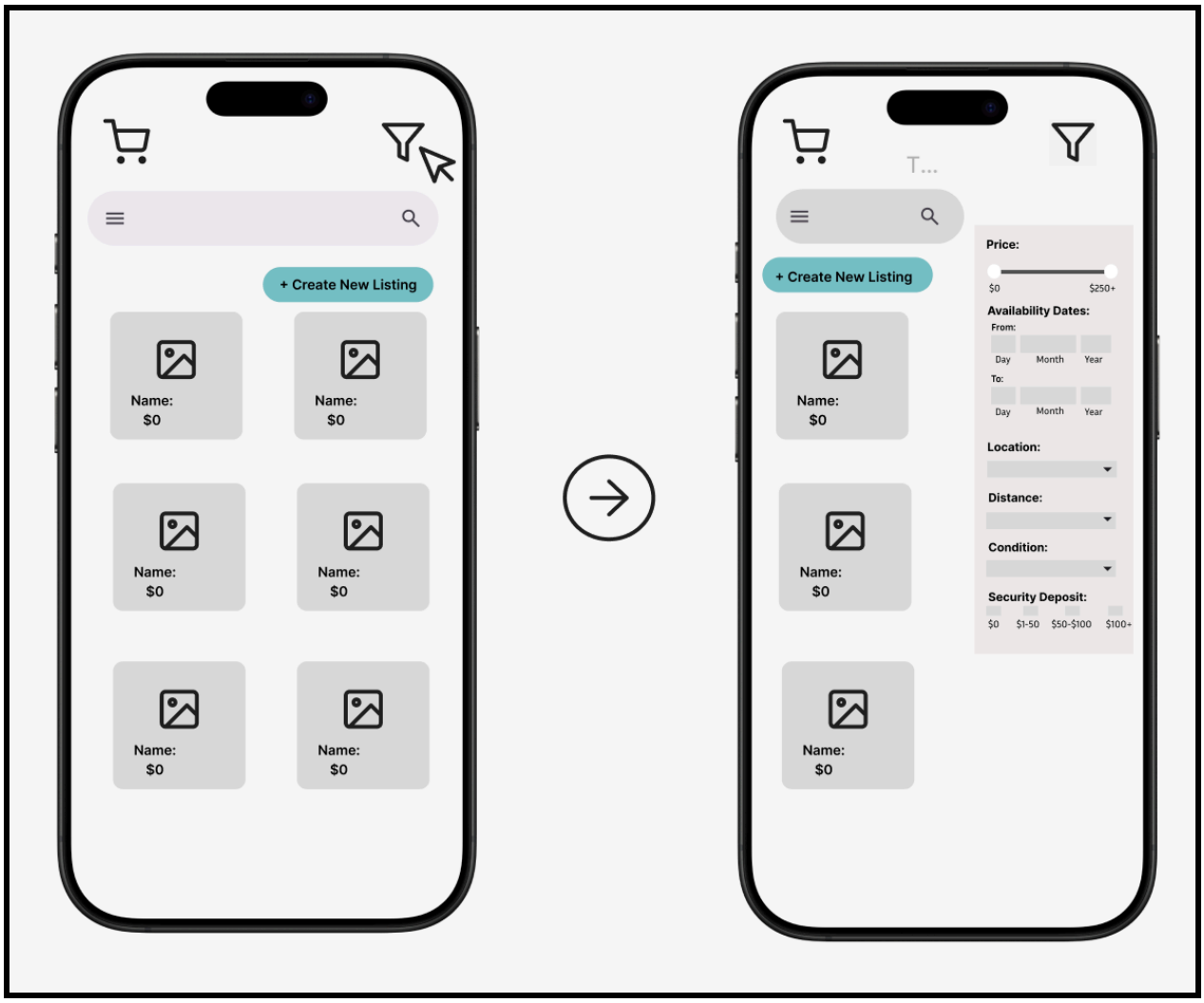


3. Search Item/s

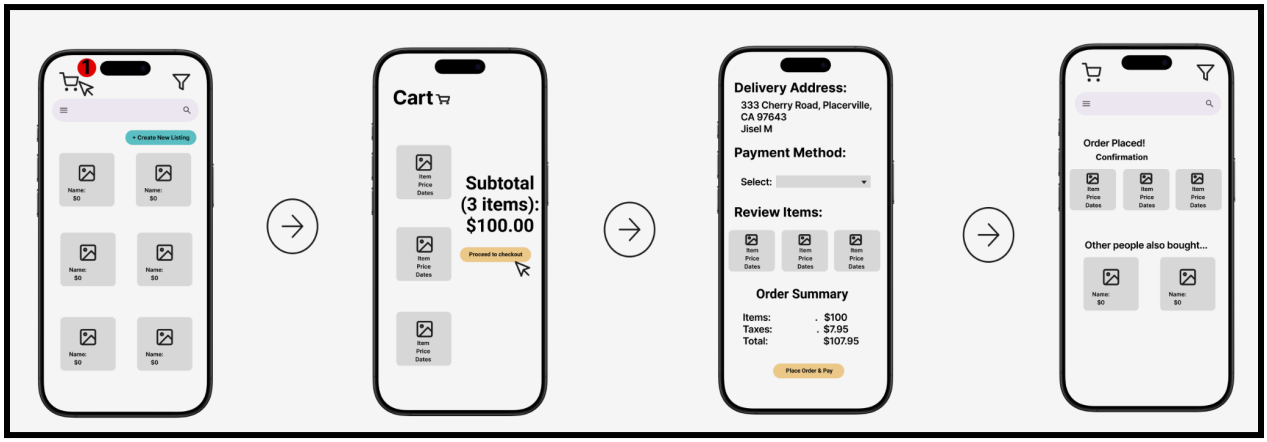

+ Create New Listing
Name:
$0
Name:
$0
Name:
$0
Name:
$0
Name:
$0
Name:
$0
A Category
B Category
C Category
D Category
E Category
F Category
G Category
H Category
I Category
J Category
K Category



4. View Item Info/Details


+ Create New Listing
Name:
$0
Price:
$0
$250+
Availability Dates:
From:
Day
Month
Year
To:
Location:
Distance:
Condition:
Security Deposit:
$0
$1-50
$50-$100
$100+

5. Book Item/s


Cart
Subtotal
(3 items):
$100.00
Delivery Address:
333 Cherry Road, Placerville, CA 97643
Jisel M
Payment Method:
Select:
Review Items:
Order Summary
Items: $100
Taxes: $7.95
Total: $107.95
Order Placed!
Confirmation
Other people also bought...

Database Organization

For SecondChance, we will use a SQL database (PostgreSQL) for structured data storage. Below is a high-level schema for the primary SQL tables. You can find the entire Database Schema here: https://lucid.app/lucidchart/22b57e12-8c58-4092-b651-a344ffb43560/edit?viewport_loc=-1361%2C934%2C1612%2C736%2C0_0&invitationId=inv_c1002da0-b325-4362-8952-79f67f6d8310

1. **Users Table**
    - **Columns**:
        - `user_id`: Primary Key, Auto-increment
        - `email`: String, Unique
        - `password`: String (hashed)
        - `name`: String
        - `role`: String (values: "renter", "owner", "admin")
        - `profile_picture`: URL to user avatar
        - `created_at`: Timestamp
        - `updated_at`: Timestamp
2. **Items Table**
    - **Columns**:
        - `item_id`: Primary Key, Auto-increment
        - `owner_id`: Foreign Key (references Users.user_id)
        - `title`: String
        - `description`: Text
        - `category`: String (e.g., furniture, electronics)
        - `rental_price`: Decimal
        - `quantity`: Integer
        - `condition`: String (e.g., New, Like New, Used)
        - `location`: String (City, State)
        - `status`: String (values: "available", "rented")
        - `created_at`: Timestamp
        - `updated_at`: Timestamp
3. **Rentals Table**
    - **Columns**:
        - `rental_id`: Primary Key, Auto-increment
        - `renter_id`: Foreign Key (references Users.user_id)
        - `item_id`: Foreign Key (references Items.item_id)
        - `start_date`: Date

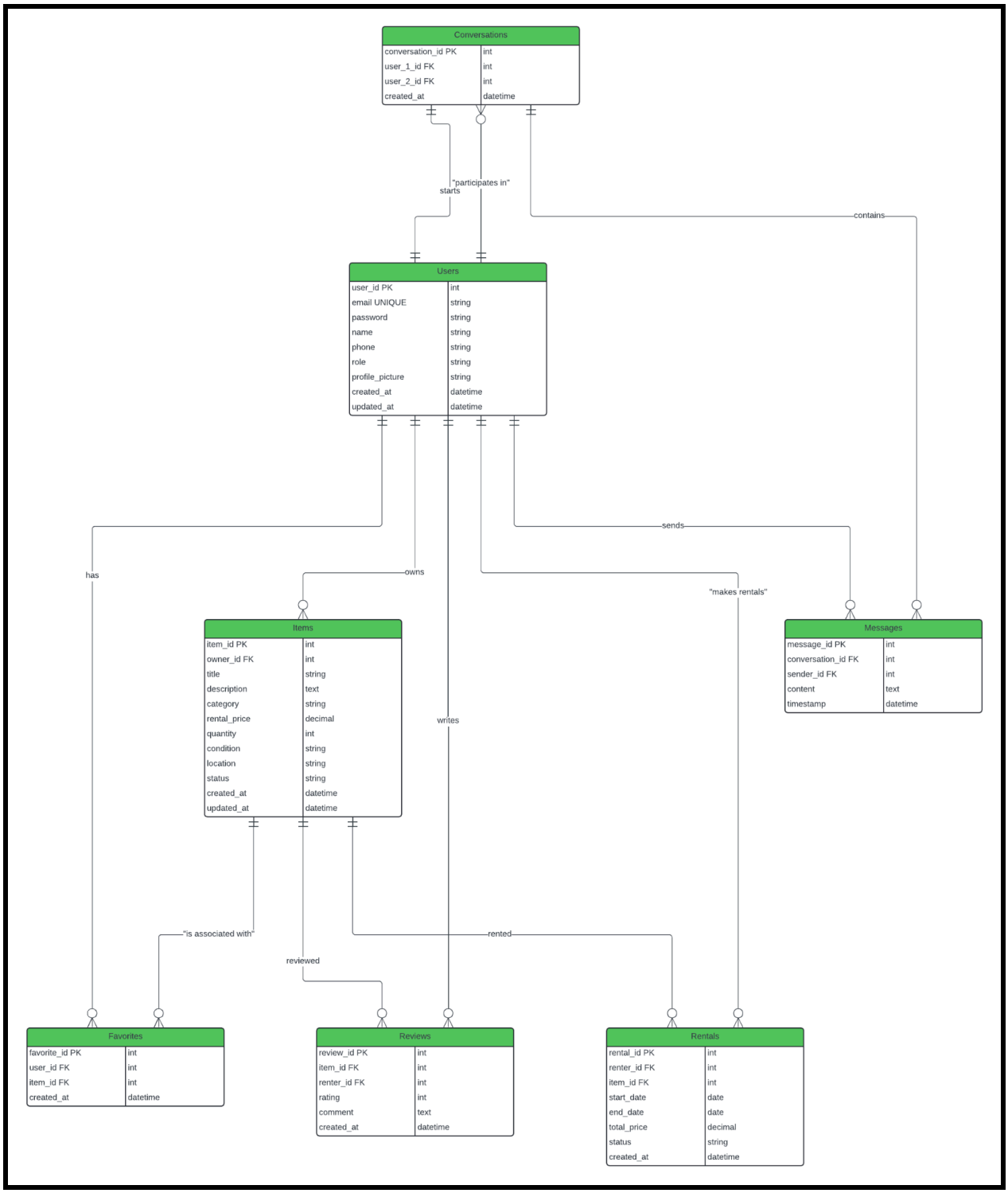
        - `end_date`: Date
        - `total_price`: Decimal
        - `status`: String (values: "active", "completed", "cancelled")
        - `created_at`: Timestamp

4. **Reviews Table**
    - **Columns**:
        - `review_id`: Primary Key, Auto-increment
        - `item_id`: Foreign Key (references Items.item_id)
        - `renter_id`: Foreign Key (references Users.user_id)
        - `rating`: Integer (1-5)
        - `comment`: Text
        - `created_at`: Timestamp

5. **Favorites Table**
    - **Columns**:
        - `favorite_id`: Primary Key, Auto-increment
        - `user_id`: Foreign Key (references Users.user_id)
        - `item_id`: Foreign Key (references Items.item_id)
        - `created_at`: Timestamp

6. **Messages Table**
    - **Columns**:
        - `message_id`: Primary Key, Auto-increment
        - `conversation_id`: Foreign Key (references Conversations.conversation_id)
        - `sender_id`: Foreign Key (references Users.user_id)
        - `content`: Text
        - `timestamp`: Timestamp

7. **Conversations Table**
    - **Columns**:
        - `conversation_id`: Primary Key, Auto-increment
        - `user_1_id`: Foreign Key (references Users.user_id)
        - `user_2_id`: Foreign Key (references Users.user_id)
        - `created_at`: Timestamp



Add/Delete/Search Architecture

Each table will support the following CRUD (Create, Read, Update, Delete) operations:

1. **Users Table**:
    - **Create**: Add new users upon registration.
    - **Read**: Search users by email or `user_id`.
    - **Update**: Allow users to update their profiles, such as name, phone, or profile picture.
    - **Delete**: Admins can deactivate or delete user accounts.
2. **Items Table**:
    - **Create**: Owners can list new items for rent.
    - **Read**: Users can search items by `category`, `location`, `price range`, and `condition`.
    - **Update**: Owners can update item details such as description, status, and quantity.
    - **Delete**: Owners can remove items from the marketplace.
3. **Rentals Table**:
    - **Create**: A rental record is created when a renter checks out an item.
    - **Read**: Retrieve rental history for both owners and renters.
    - **Update**: Modify rental status (e.g., extend rental duration or cancel).
    - **Delete**: N/A (rental records remain for historical reference).
4. **Reviews Table**:
    - **Create**: Renters can leave reviews for items they've rented.
    - **Read**: Display reviews on item pages.
    - **Update**: Renters can edit their reviews within a specified time period.
    - **Delete**: Admins can delete reviews in case of disputes.
5. **Favorites Table**:
    - **Create**: Users can add items to their favorites list.
    - **Read**: Display user's favorite items on their dashboard.
    - **Update**: N/A
    - **Delete**: Users can remove items from their favorites list.
6. **Messages & Conversations**:
    - **Create**: New conversations and messages are created when users chat.
    - **Read**: Fetch chat history for both users in a conversation.
    - **Update**: N/A
    - **Delete**: Admins can delete inappropriate conversations.

Backend Endpoint APIs

Below are the major backend APIs we will use:

1. **User APIs**:
    - **POST /api/auth/register**: Register a new user.
    - **POST /api/auth/login**: Authenticate a user and generate an access token.
    - **PUT /api/users/:user_id**: Update user profile details.
    - **GET /api/users/:user_id**: Retrieve user details.
    - **DELETE /api/users/:user_id**: Deactivate or delete user.
2. **Item APIs**:
    - **POST /api/items/create**: Create a new item listing.
    - **GET /api/items**: Fetch a list of items with filters like `category`, `location`, `priceMin`, `priceMax`.
    - **GET /api/items/:item_id**: Retrieve details of a specific item.
    - **PUT /api/items/:item_id**: Update an item's details.
    - **DELETE /api/items/:item_id**: Remove an item from the platform.
3. **Rental APIs**:
    - **POST /api/rentals/create**: Create a new rental record.
    - **GET /api/rentals/:rental_id**: Retrieve details of a rental.
    - **PUT /api/rentals/:rental_id**: Update rental details (e.g., extend rental).
    - **GET /api/users/:user_id/rentals**: List a user's rental history.
4. **Review APIs**:
    - **POST /api/reviews/create**: Leave a review for an item.
    - **GET /api/items/:item_id/reviews**: Fetch all reviews for an item.
    - **DELETE /api/reviews/:review_id**: Delete a review.
5. **Chat APIs**:
    - **POST /api/chat/start/:user_id**: Start a new conversation with another user.
    - **GET /api/chat/:conversation_id**: Retrieve all messages for a conversation.
    - **POST /api/chat/:conversation_id/message**: Send a new message in a conversation.
6. **Favorites APIs**:
    - **POST /api/favorites/:item_id**: Mark an item as favorite.
    - **DELETE /api/favorites/:item_id**: Remove an item from favorites.
    - **GET /api/users/:user_id/favorites**: List all favorite items for a user.

3rd Party APIs & Open-Source Components

1. **3rd Party APIs**:

- **Stripe**: For handling payments and transactions, we will integrate **Stripe**. APIs like `POST /charge` and `GET /payment-status` will be utilized to process rental payments and refunds.
- **Google Maps API**: To enable location-based searches and to display the location of rental items on the map.

2. **Open-Source Components**:
    - **TailwindCSS**: A utility-first CSS framework for building custom designs easily.
    - **React.js**: Used for building interactive UI components.
    - **Next.js**: Server-side rendering framework to improve SEO and load times.
    - **Zustand**: For state management within React.

High Level UML Diagram

You can find the UML Diagram here: https://lucid.app/lucidchart/800525de-2e28-43ea-8f49-37ad9a60cbba/edit?viewport_loc=158%2C-777%2C1574%2C972%2C0_0&invitationId=inv_67b0df62-1ea6-4d51-a863-139787785a2c

Key Classes:

1. **User**
    - Attributes:
        - `userId`: String
        - `email`: String
        - `password`: String
        - `name`: String
        - `phone`: String
        - `profilePicture`: String
        - `role`: String
    - Methods:
        - `register()`: Creates a new user account
        - `login()`: Logs the user into the system
        - `updateProfile()`: Updates user profile information
        - `viewRentals()`: Lists all rentals made by the user
        - `sendMessage()`: Initiates a conversation with another user

2. **Item**
    - Attributes:
        - `itemId`: String
        - `title`: String
        - `description`: String
        - `category`: String

        - `rentalPrice`: Decimal
        - `quantity`: Integer
        - `condition`: String
        - `location`: String
        - `owner`: User (owner of the item)
    - Methods:
        - `createListing()`: Creates a new item listing
        - `updateListing()`: Updates an existing item
        - `removeListing()`: Deletes an item listing
        - `getAvailableItems()`: Retrieves all available items for rent
        - `rentItem()`: Rent an item to a renter

3. **Rental**
    - Attributes:
        - `rentalId`: String
        - `renter`: User
        - `item`: Item
        - `startDate`: Date
        - `endDate`: Date
        - `totalPrice`: Decimal
        - `status`: String
    - Methods:
        - `createRental()`: Creates a new rental transaction
        - `updateRental()`: Modifies rental details (e.g., extend rental)
        - `completeRental()`: Marks rental as complete

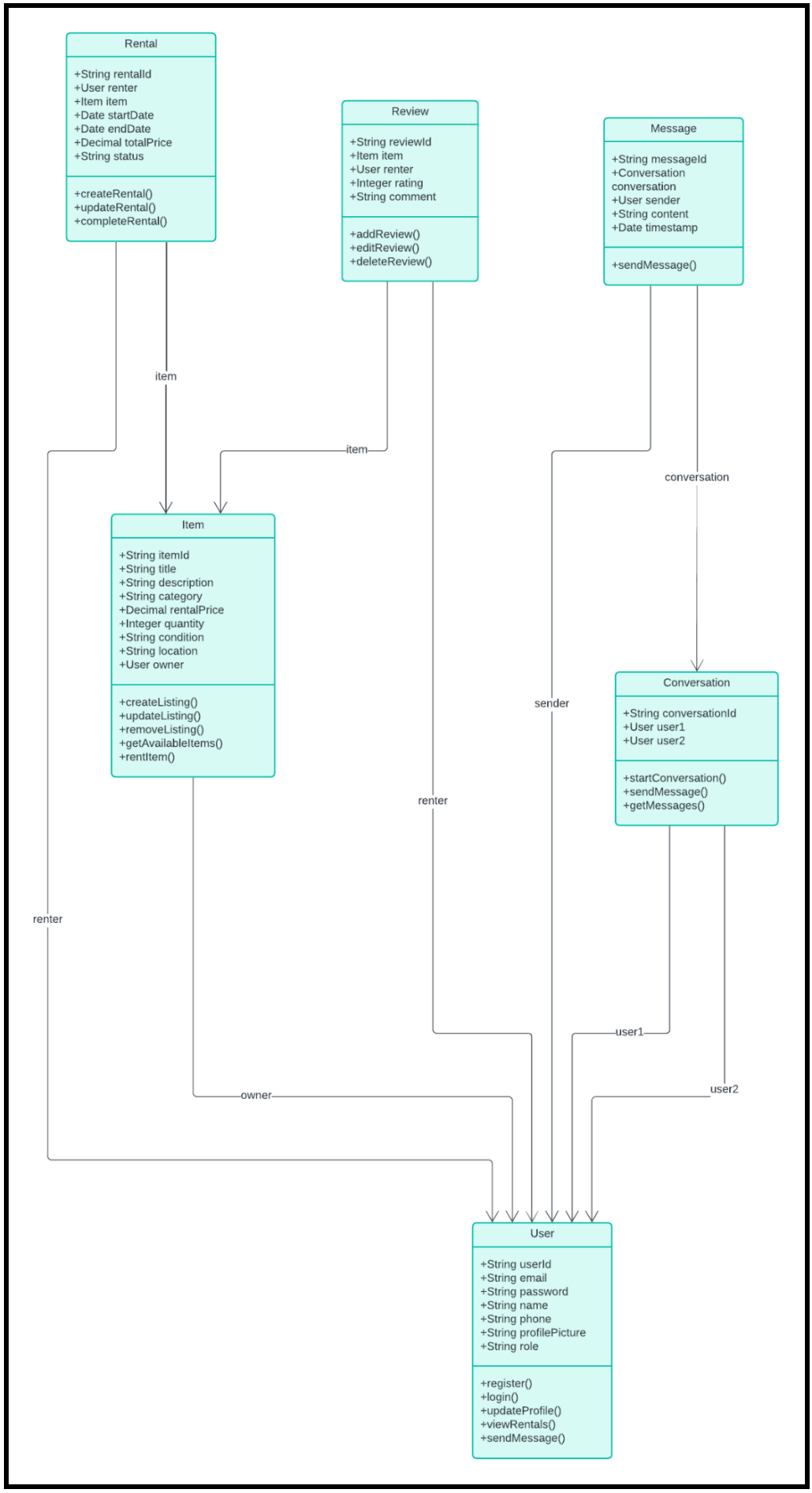
4. **Review**
    - Attributes:
        - `reviewId`: String
        - `item`: Item
        - `renter`: User
        - `rating`: Integer
        - `comment`: String
    - Methods:
        - `addReview()`: Adds a review for a rented item
        - `editReview()`: Edits an existing review
        - `deleteReview()`: Deletes a review

5. **Conversation**

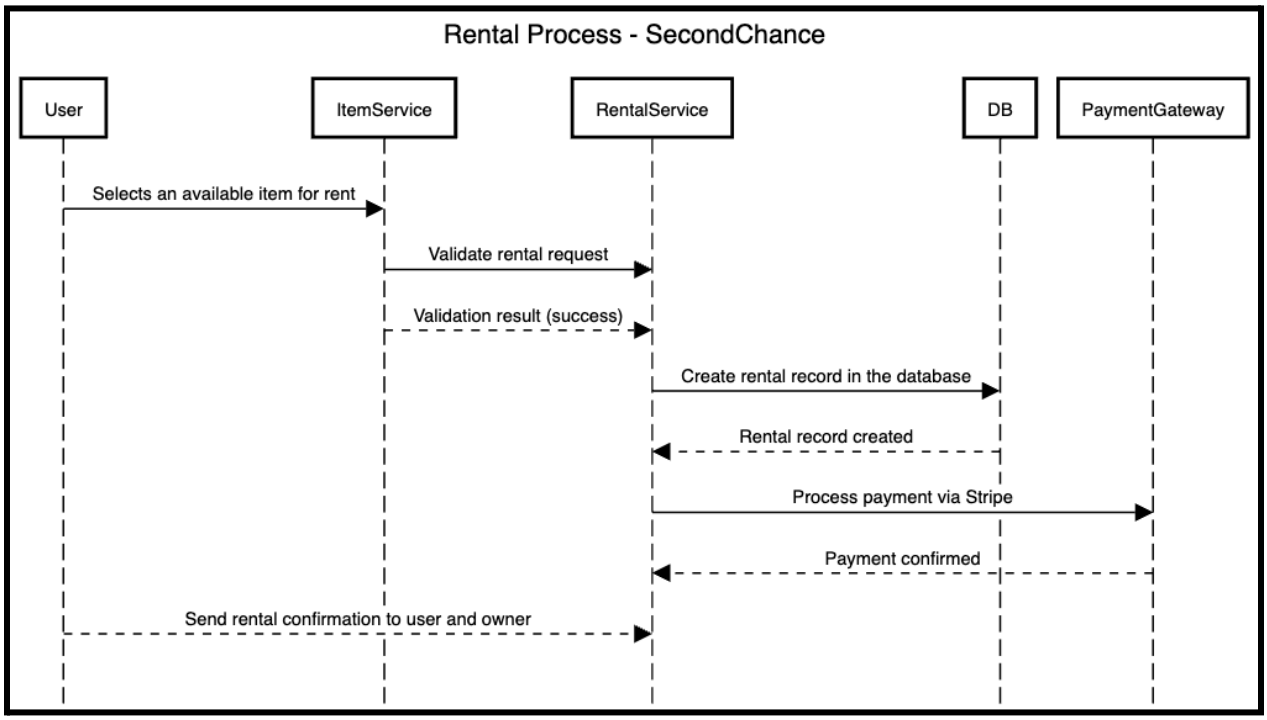
- Attributes:
  - `conversationId`: String
  - `user1`: User
  - `user2`: User
- Methods:
  - `startConversation()`: Starts a new conversation
  - `sendMessage()`: Sends a message to another user
  - `getMessages()`: Retrieves all messages in the conversation

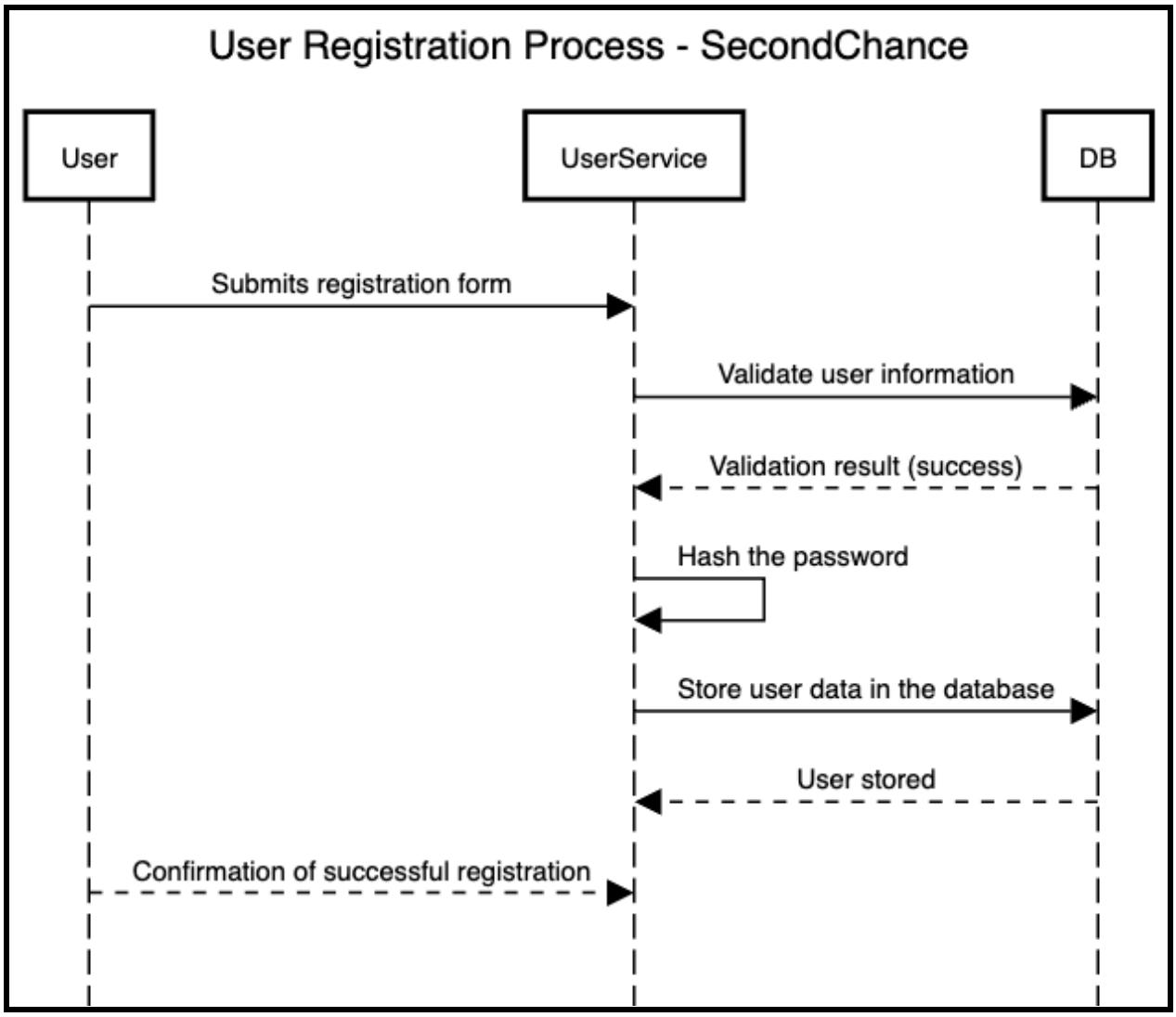
6. **Message**

- Attributes:
  - `messageId`: String
  - `conversation`: Conversation
  - `sender`: User
  - `content`: String
  - `timestamp`: Date
- Methods:
  - `sendMessage()`: Sends a new message in a conversation


Rental
+String rentalId
+User renter
+Item item
+Date startDate
+Date endDate
+Decimal totalPrice
+String status
+createRental()
+updateRental()
+completeRental()
Review
+String reviewId
+Item item
+User renter
+Integer rating
+String comment
+addReview()
+editReview()
+deleteReview()
Message
+String messageId
+Conversation conversation
+User sender
+String content
+Date timestamp
+sendMessage()
item
item
conversation
Item
+String itemId
+String title
+String description
+String category
+Decimal rentalPrice
+Integer quantity
+String condition
+String location
+User owner
+createListing()
+updateListing()
+removeListing()
+getAvailableItems()
+rentItem()
Conversation
+String conversationId
+User user1
+User user2
+startConversation()
+sendMessage()
+getMessages()
sender
renter
renter
user1
user2
owner
User
+String userId
+String email
+String password
+String name
+String phone
+String profilePicture
+String role
+register()
+login()
+updateProfile()
+viewRentals()
+sendMessage()

High-Level UML Sequence Diagram





Risk Identification

1. Skills Risks and Mitigation Plan

- **Risk**: Some team members are lacking familiarity with certain technologies, especially in React and NodeJS.
- **Mitigation Plan**: We are ensuring that every member is up to speed by organizing knowledge-sharing sessions, which will cover specific tools like NodeJS, React, and Django.

2. Schedule Risks

- **Risk**: Our timelines have started to lag as certain team members are becoming unresponsive, except for Charvi and Pedro, which affects team efficiency and overall progress.

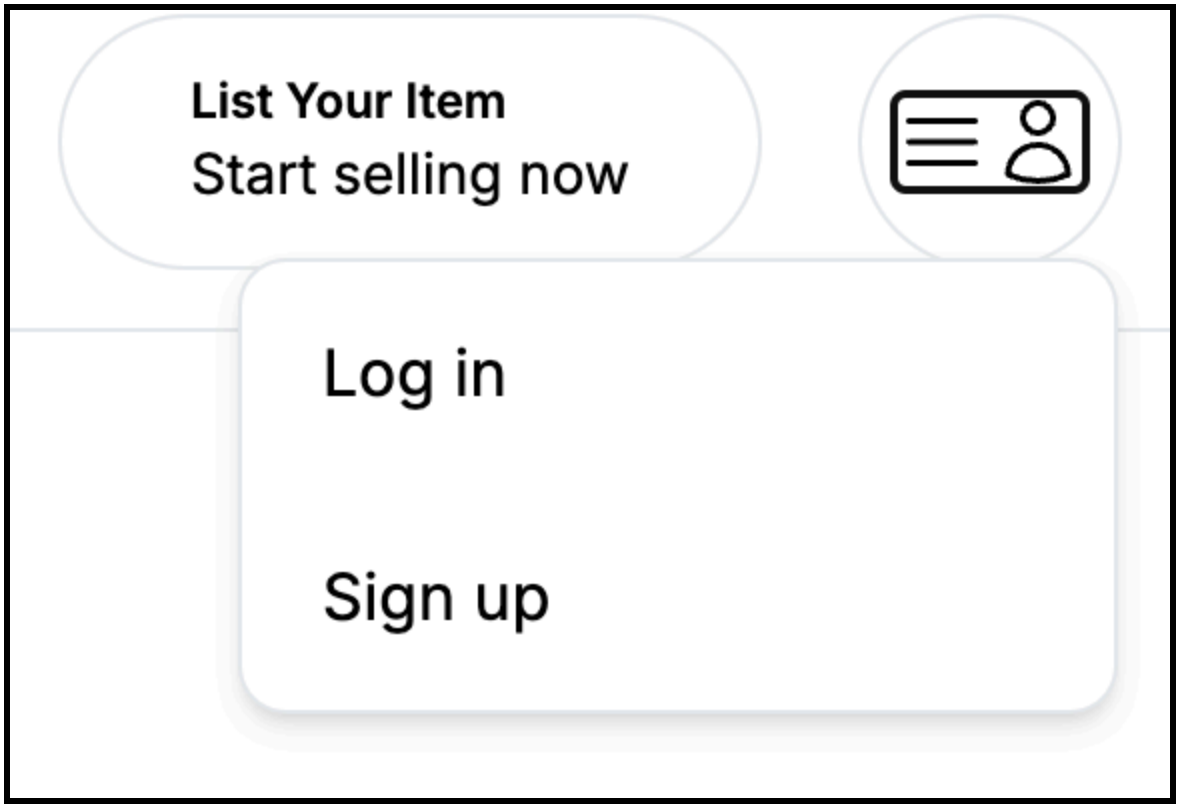
- **Mitigation Plan**: Charvi (Scrum Master) and I will conduct more frequent check-ins with each member to ensure they are on track and meeting deadlines. If unresponsiveness continues, tasks will be reassigned to active team members or split into smaller subtasks for easier completion. We will reinforce using our project management tool (Trello/Jira) to update schedules and deadlines transparently. Immediate intervention will be implemented if further delays occur.

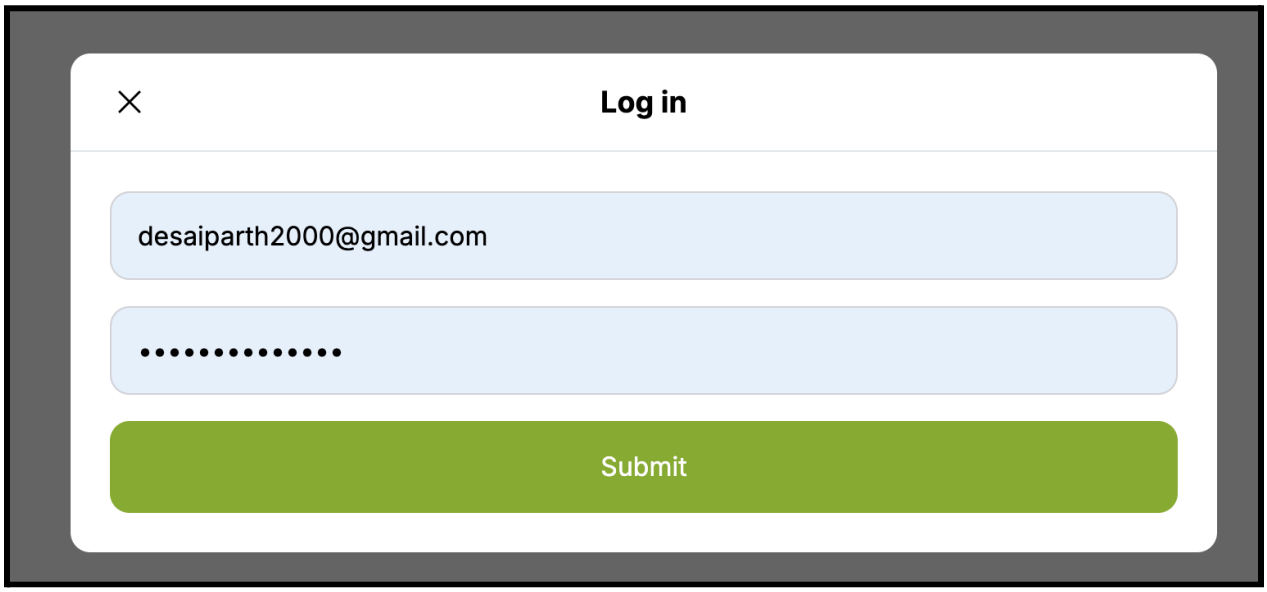
Project Management

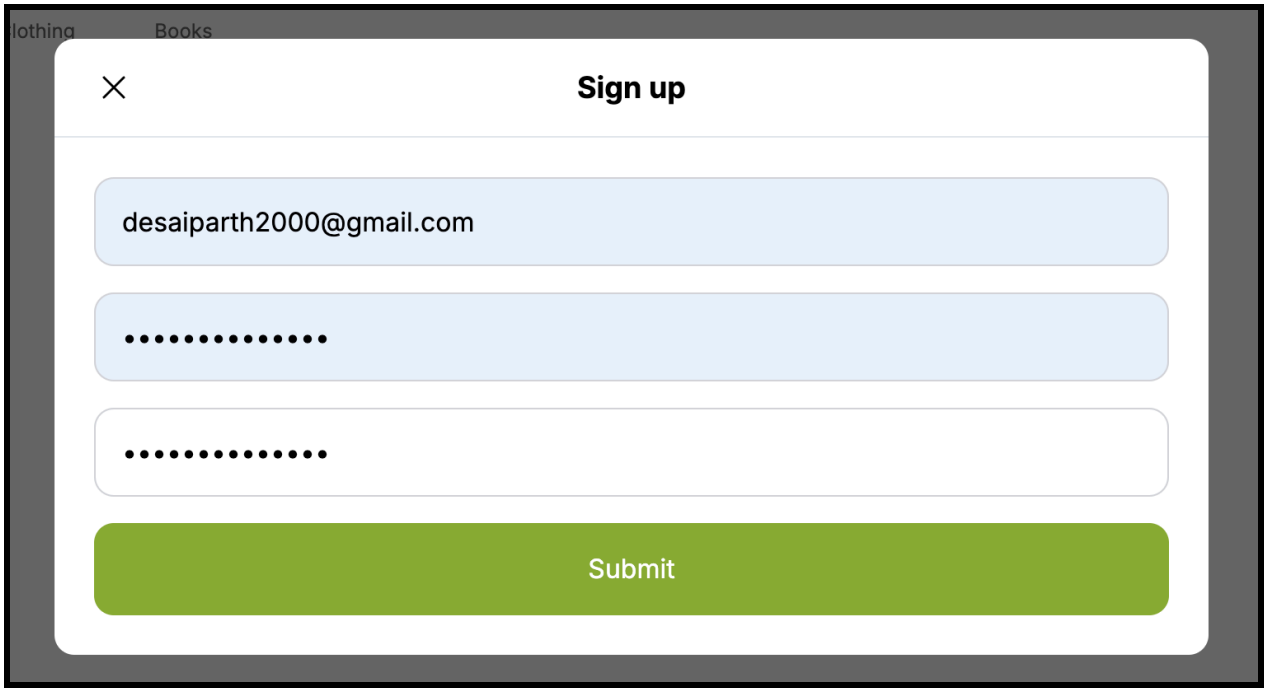
Our team is managing the tasks for Milestone 2 primarily through weekly **SCRUM meetings** held every Wednesday. During these meetings, each member provides an update on their progress, and we discuss any blockers or dependencies affecting task completion. The team shares progress transparently in these meetings, and each member is held accountable for their respective tasks.

We are currently tracking our progress using a shared **Google Docs document**, where each team member logs their updates and tasks. While this method helps with transparency, we are considering migrating to a more structured project management tool like **Notion** to streamline task assignments.

Vertical SW Prototype


List Your Item
Start selling now
Log in
Sign up


Log in
desaiparth2000@gmail.com
Submit


lothing
Books
Sign up
desaiparth2000@gmail.com
Submit