Project Specification: Item Rental SaaS

Course: CS-UY 4513 - Software Engineering
Professor: Dr. DePasquale
Project Title: Item Rental SaaS
Date: 09/28/2025

1.0 Project Overview

Item Rental SaaS is a web-based application powered by Ruby on Rails that allows users to rent everyday items to and from one another in a centralized platform.

On the platform, item owners will be equipped with the tools needed to list items such as cameras, bicycles, camping equipment, personal equipment, etc., and will allow interested renters to search, offer (or negotiate), request rental bookings, create, and view rental documents.

Item owners and renters are both able to report one another in the event of phishing or scams, which will be reviewed by administrators on the Item Rental SaaS team to ensure safety and security on the platform.

2.0 Core Requirements

The project must adhere to the mandatory characteristics outlined in the project specification template document.

2.1 User-Based System

Users are required to create an account and must be logged in to interact with the platform.

Authorization: Three roles—Renter, Owner, Admin—with role-scoped access:

Renter: browse/search listings, request bookings, manage own bookings.

Owner: create/manage their listings, set availability/pricing, approve/decline requests on those listings.

Admin: moderate users/listings, resolve disputes, view system metrics.

Administrative access is given to moderators and staff who operate and maintain the platform. Admins have the ability to switch into 3 different roles.

2.2 User Roles (Minimum 3)

ROLE 1: Item Owner

- Lists items with pricing, availability, photos, location, & descriptions.
- Has authority to accept/decline rental offers.
- Can leave reviews on renters.
- Rental agreement sent to renter on behalf of item owner after offer is accepted.

ROLE 2: Renter

- Can search & filter available items.
- Sends requests, signs rental agreement if owner agrees to offer.
- Required to pay for rental fee and deposit (if set by owner).
- Can leave reviews on items and its owner.

ROLE 3: Admin

- Manages all users on platform.
- Conducts reviews on flagged items, item owners, and renters.
- Has authority to mediate disputes between item owners and renters.
- Unrestricted access to all item owner and renter data.

2.3 Persistent Storage

The application will use a **PostgreSQL relational database** to persist all data, leveraging its strong support for relational integrity, indexing, and deployment compatibility with Heroku. The schema will be capped at **10 tables**, with clear normalization to avoid redundancy.

Proposed Database Schema (max 10 tables):

1. **users** – Stores user credentials and profile details.
    a. Fields: user_id (PK), username, first_name, last_name, email, phone, roles, account_created_at, report_count, account_status.
2. **items** – Represents rentable items.
    a. Fields: item_id (PK), owner_id (FK users), title, description, listed_at, item_image, price_per_day, category_id (FK categories), availability_status (available/pending/unavailable), views_count.
3. **categories** – Organizes items into categories.
    a. Fields: category_id (PK), name.

4. **bookings** – Tracks rental lifecycle between renter and owner.
    a. Fields: booking_id (PK), item_id (FK items), renter_id (FK users), start_date, end_date, price_total, terms, status (requested/approved/active/returned/cancelled).
5. **rental_agreements** – Stores formalized agreements tied to bookings.
    a. Fields: agreement_id (PK), booking_id (FK bookings), signed_owner_at, signed_renter_at, terms_text.
6. **messages** – Track communication between renter and owner.
    a. Fields: message_id (PK), booking_id (FK bookings), sender_id (FK users), body, sent_at.
7. **disputes** – Records and manages conflict resolution.
    a. Fields: dispute_id (PK), booking_id (FK bookings), created_by (FK users) , reason, details, status (open/resolved), resolved_by (FK users), created_at, resolved_at, resolution_notes.
8. **payments** – Keep track of payment information.
    a. Fields: payment_id (PK), booking_id (FK), payer_id (FK users), payee_id (FK users), dollar_amount, type (deposit/rental/adjustment/refund), method (simulated), status (pending/succeeded/failed/refunded), created_at, settled_at, reference_code

2.4 Modular Architecture (3-5 Modules)

The system must be logically divided into a minimum of 3 and a maximum of 5 modules. These modules must demonstrate clear dependencies and rely on each other for full functionality.

1. **User & Identity Management**
    a. Manages user registration, login, profiles, and role assignment (Renter, Owner, Admin).
    b. **Dependency:** Core dependency for all other modules.
2. **Listings & Search**
    a. Allows Owners to create, update, and delete listings (title, category, price, deposit, photos, availability).
    b. Allows Renters to browse listings with keyword/category filters
    c. **Dependency:** Depends on User & Identity Management to authenticate users and enforce role-based permissions.
3. **Booking & Agreements**
    a. Handles book requests/approvals, date-range validation, price/deposit calculation, and e-agreement signing.
    b. Tracks booking lifecycle (requested → approved → active → returned/cancelled) and pickup/return confirmations.

c. **Dependency:** Depends on Listings and Search for item/availability data and on User & Identity Management for renter/owner roles.

4. **Messaging & Notifications**
    a. Communication between renter ⇔ item owner, user ⇔ admin.
    b. Renter is notified that their request has been accepted by X item owner.
    c. Item owner is notified that renter X has sent a request for Y item.
    d. Users are notified that their report has been sent & been processed.
    e. Issue reporting to admins (damage/no-show), evidence uploads, and basic dispute workflow.
    f. **Dependency:** Depends on Booking & Agreements for booking context/data and on User & Identity Management for participants and access control.
5. **Admin & Reporting**
    a. Allows Admins to moderate user accounts and listings, oversee disputes, and manage system-wide settings.
    b. Generates dashboards and reports on usage trends, booking volume, cancellations, and issue resolution rates.
    c. **Dependency:** Depends on all other modules for aggregated data and administrative oversight.

2.5 API Interfaces

Each module must expose a RESTful API interface. This is crucial for enabling the modules to communicate with each other and for potential future integrations. The API should follow standard REST conventions.

Endpoints:

- **User/Auth API (User & Identity Module)**
    - **POST /api/v1/auth/register**
        - Register an account and log in as that account
        - Request body:
            - username, email, phone number, password
        - Response:
            - success: bool
            - token: string
    - **POST /api/v1/auth/login**
        - Login to an account
        - Request body:
            - username OR email, password
        - Response:
            - token: string

- **POST /api/v1/auth/id**
  - Get the account info given a token
  - Request body:
    - token
  - Response:
    - username: string
    - email: string
    - phone number: string

- **Items API (Listing & Search Module)**
  - **GET /api/v1/items/:page**
    - Get the listing of items available
    - Query Parameters:
      - sortOrder
      - filter
    - Response:
      - listing: Array[{
        - itemname: string
        - itemid: string
        - thumbnail_url: string }]
  - **GET /api/v1/items/:id**
    - Get info about an item
    - Response:
      - item: {
        - itemname: string
        - picture_url: string
        - description: string
        - owned_by: string
        - rented_by: string?
        - rental_terms: string }
  - **POST /api/v1/items/new**
    - Creates a new listing
    - Request body:
      - name (of the item)
      - picture of item
      - description
      - terms of rental
      - token (needs to be owner)
    - Response:
      - success: bool
  - **DELETE /api/v1/items/:id**

        - Deletes an item listing
        - Request body:
            - token (needs to be owner of item, or admin)
        - Response:
            - success: bool
- **Lender API (Booking & Agreements Module)**
    - **POST /api/v1/booking/:id/rent**
        - Rents the item
        - Request body:
            - token (needs to be renter)
        - Response:
            - success: bool
            - formal_agreement_url: string
    - **POST /api/v1/booking/:id/return**
        - Marks the item as returned
        - Request body:
            - token (needs to be owner of item)
        - Response:
            - success: bool
- **Messages API (Messaging & Notifications Module)**
    - **GET /api/v1/messages/conversations/**
        - Get a list of open conversations of a user
        - Request body:
            - token
        - Response:
            - conversations: Array[{ recipient: string, conversationID: string }]
    - **GET /api/v1/messages/conversations/:conversationID/:page**
        - Get a page of messages in the specified conversation
        - Request body:
            - token
        - Response:
            - messages: Array[{
                - from: string
                - text: string
                - timestamp: number
                - is_read: bool }]
    - **POST /api/v1/messages/send/:conversationID**
        - Sends a message to another user
        - Request body:
            - token (needs to be in the conversation, or admin)

                - message
            - Response:
                - success: bool
- **Admin API (Admin & Reporting Module)**
    - **POST /api/v1/admin/ban**
        - Disables or enables an account
        - Request body:
            - account_username
            - whether to disable or enable
            - token (needs to be admin)
        - Response:
            - success: bool
    - **POST /api/v1/admin/disputes/new**
        - Creates a new dispute
        - Request body:
            - itemID
            - token (needs to be renter or owner of item)
            - dispute details
        - Response:
            - success: bool
    - **GET /api/v1/admin/disputes**
        - Gets a list of all disputes
            - Request body:
                - token (needs to be admin)
        - Response:
            - disputes: Array[{ itemID, dispute_details }]

3.0 Technical Stack

- **Language:** Ruby
- **Framework:** Ruby on Rails
- **Database:** PostgreSQL
- **Testing:** RSpec for unit and integration testing

4.0 Deliverables

The final submission must include:

1. **Complete Source Code:** A well-structured and commented codebase, including all necessary configuration files.

2. **Project Documentation:** A `README.md` file that explains the system's architecture, setup instructions, and how to run the application.
3. **API Documentation:** A separate document or file detailing all API endpoints, their expected parameters, and response formats.
4. **Presentation:** A brief presentation and demonstration of the application's core features.