# RoomMate Project Specification

Course: CS-UY 4513 - Software Engineering
Professor: Dr. DePasquale
Project Title: RoomMate
Date: September, 28, 2025

## 1.0 Project Overview

The goal of this project is to design and develop a SaaS platform that helps students and newcomers to a city find compatible roommates to share housing costs.

The platform is named **RoomMate** and will serve as a reliable and user-friendly solution for individuals seeking to split rent and reduce living expenses.

The system will support a robust user management system with:

- Profile creation and verification for identity, budget, lifestyle preferences, and housing needs;
- Matching algorithms to connect users based on compatibility factors such as budget, location, habits, and schedules;
- A way for users to coordinate with potential roommates;
- Persistent data storage to maintain user preferences, search history, and active matches.

## 2.0 Core Requirements

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

### 2.1 User-Based System - Drastansh

RoomMate is designed as a user-centered platform, meaning that everything starts with the individual. Each person who signs up will create their own profile, where they can share important details such as their budget, housing preferences, lifestyle habits, and even their daily schedule. This information will help the system understand what kind of roommate they are looking for and make better matches.

To make the experience smooth and trustworthy, users will be able to register and log in securely, either with a simple email and password or through third-party options like Google or university accounts. We will also include basic verification steps, such as confirming email or phone numbers, and may expand to options like student ID or government ID checks for extra safety.

Once a profile is created, users can update or change their information whenever needed. All preferences, search history, and active matches will be stored safely in the system, so users can pick up right where they left off, even if they switch devices. Privacy will also be a key focus—people will have control over what information they want to share publicly and what stays private.

By building the system around these features, RoomMate ensures that everyone using the platform has a secure, reliable, and personalized experience while searching for a compatible roommate.

## 2.2 User Roles (Minimum 3) - Eric

- **ROLE 1:** **Member**: the core participant on the platform, who can act as either a room seeker or a room lister. Including students currently enrolled in a college or university, recent graduates, young professionals, or newcomers to the city for work.
    - Responsible for creating and managing their own profile (e.g., lifestyle details, budget, preferences).
    - Students can obtain a "Verified Student" badge through a specific process (e.g., using a .edu email) to increase their credibility.
    - Can receive, review, and manage a list of potential matches generated by the matching algorithm.
    - In addition to reviewing system-generated matches, as a room seeker, can search and filter for listings and communicate with room listers.
    - As a room lister, can create a public housing listing for a property they occupy and communicate with seekers.
    - Can send a request to a "Community Verifier" to validate a listing they created.
    - After a successful match, they can rate their roommate(s), contributing to the community credit system.
    - Can rate a "Community Verifier" on their service.
    - Can report suspicious behavior on the platform.
    - Cannot self-verify a listing they have posted.
    - Cannot modify another user's profile, listing, or any submitted ratings and their own public roommate credit score.

- **ROLE 2:** **Community Verifier**: a trusted, platform-vetted role (e.g., a landlord, property manager, or a reputable prior tenant) whose sole purpose is to provide an authenticity verification service for listings created by "Members", in order to enhance platform security (a verified listing receives a critical trust signal that significantly boosts its visibility and priority within the matching algorithm).
    - Maintains a public verifier profile and can edit their own information.
    - Can add a "Verified" badge to a listing after accepting a verification request from a "Member".
    - Can answer property-specific questions from other "Members" regarding a listing they have verified.
    - Can provide optional public endorsements for users they have had a positive history with.
    - Cannot create their own housing listings.

- Cannot search for roommates or initiate contact for housing-related purposes like a “Member”.
- Cannot modify a user’s listing content, nor their own aggregated public rating received from others.
- Cannot modify administrative settings.

- **ROLE 3:** **Administrator**: a superuser responsible for ensuring the safety, integrity, and smooth operation of the entire platform.
  - Has full access to a dedicated administrative dashboard to oversee all platform activities.
  - Oversees the performance, fairness, and tuning of the matching algorithm to ensure it does not produce biased or harmful results.
  - Responsible for the vetting, approval, and revocation of “Community Verifier” status.
  - Manages all user accounts (“Members” and “Community Verifiers”), including suspension and deletion for policy violations.
  - Investigates and resolves user-submitted reports regarding fraud, harassment, or other misconduct.
  - Moderates all user-generated content (including listings, profiles, and photos) to ensure compliance with community standards.
  - Oversees the integrity of the platform’s reputation system and has the authority to investigate and mediate disputes (e.g., retaliatory ratings or persistent mismatches caused by the algorithm) in exceptional cases.
  - Cannot view private user messages unless required for an investigation into a reported violation, to protect user privacy.
  - Cannot participate in the platform market as a “Member” or “Community Verifier” using their administrative account.

## 2.3 Persistent Storage - Terry

**Proposed Database Schema (max 10 tables):**

- **users**: Stores core user accounts with information including login credentials and basic personal details such as name, email, phone number, etc.
- **profiles**: Stores extended profile data for users including budget, location, lifestyle/roommate preferences, etc.
- **listings**: Stores housing options posted by users.
- **user_listings**: Links listings to users.
- **potential_matches**: Records suggested roommate matches based on users’ profiles and the application’s compatibility algorithms.
- **active_matches**: Records active and successful matches between users.
- **messages**: Stores chat messages between matched users.
- **search_history**: Keeps track of user’s past searches, such information could include location, budget, preferences, etc.
- **verifications**: Stores user verification records.
- **reports**: Logs issues raised by users, such issues could include spams, harassment, fake profiles, etc.

## 2.4 Modular Architecture (3-5 Modules) - Kevin

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-service
    - user login, registration, and verification (email/password, Google, SSO)
    - session management (jwt-based authentication)
    - account management (password reset, account deletion, other settings.)
    - stores and manages profile info (budgets, habits, lifestyle, location, etc.)
    - dependencies: none
2. matcher-service
    - generate potential matches for users based on budget, lifestyle, location, and preferences
    - CRUD operations for potentials matches and active matches
    - track matching history
    - determine compatibility score for listings (percent\points)
    - dependencies:
        - lister-service (for listings)
        - user-service (for profile data, jwt tokens)
3. lister-service
    - CRUD operations for housing listings (title, rent, location, description, images)
    - link listings to owners
    - handle favorite or save actions by users.
    - show if a listing is verified
    - dependencies:
        - user-service (to see if a place is verified, jwt tokens)
4. messenger
    - One-to-one messaging between matched users.
    - Send/receive messages (text, media, files).
    - Block or report other users.
    - Log conversation history for active matches.
    - dependencies:
        - user-service (to identify users, jwt tokens)
        - matcher-service (ensure that only matched users communicate)

## 2.5 API Interfaces - Steven

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.

```
# Authentication
POST /auth/register # Register a new user
POST /auth/login # User login
POST /auth/refresh # Refresh authentication token
POST /auth/logout # Logout user
POST /auth/reset-password # Reset user password

# User Management
GET /users/me # Get current user info
DELETE /users/me # Delete current user
PUT /users/me # Update current user
POST /users/me # Create current user
GET /users/{id} # Get user by ID

# Profile
GET /users/{id}/profile # Get user profile info
DELETE /users/{id}/profile # Delete user profile
PUT /users/{id}/profile # Update user profile (partial)
POST /users/{id}/profile # Create user profile
GET /users/{id}/picture # Get user profile picture
DELETE /users/{id}/picture # Delete user profile picture
POST /users/{id}/picture # Upload user profile picture

# Matching
GET /matches/potential # Get potential matches
GET /matches/{id} # Get match details
GET /matches/{id}/like # Get like status for match
POST /matches/{id}/like # Like/save a match
DELETE /matches/{id}/like # Remove like/save from match
GET /matches/history # Get match history

# Communication
GET /conversations # Get all conversations
GET /conversations/{conversationId}/messages # Get messages in conversation
POST /conversations/{conversationId}/messages # Send message in conversation
POST /conversations/{conversationId}/files # Send file or media in conversation
POST /conversations/{conversationId}/block # Block user in conversation
POST /conversations/{conversationId}/report # Report user in conversation

# Housing Listings
GET /listings # Get all listings
GET /listings/{listingId} # Get listing by ID
POST /listings # Create new listing
DELETE /listings/{listingId} # Delete listing by ID
PUT /listings/{listingId} # Update listing by ID
GET /listings/{listingId}/images # Get images for listing
POST /listings/{listingId}/images # Upload image to listing
DELETE /listings/{listingId}/images/{imageId} # Delete image from listing
```

```
GET /listings/favorites # Get saved/favorite listings
POST /listings/{listingId}/favorite # Save/favorite a listing
DELETE /listings/{listingId}/favorite # Remove favorite from listing

# Verification System
GET /verification/email/status # Get email verification status
GET /verification/email/verify-code # Get email verification code status
POST /verification/email/verify-code # Verify email with code

# Search & Filters
GET /search/users # Search for users
GET /search/listings # Search for listings
GET /search/history # Get search history

# Notifications
GET /notifications # Get notifications
POST /notifications/{notificationId} # Mark notifications as read
DELETE /notifications/{notificationId} # Delete notification

# Admin Operations
GET /admin/users # Get all users (admin)
DELETE /admin/users/{userId} # Delete user (admin)
DELETE /admin/listings/{listingId} # Delete listing (admin)
DELETE /admin/conversations/{conversationId} # Delete conversation (admin)
DELETE /admin/pictures/{pictureId} # Delete picture (admin)

# Error Handling
GET /error # Get error response

# Rate Limiting
GET /rate-limiting # Get rate limiting info
```

## 3.0 Technical Stack - Tianyi

- **Language:** Ruby; HTML, CSS
- **Framework:** Ruby on Rails 7
- **Database:** PostgreSQL
- **Testing:** For Unit Tests: RSpec: To test our models
  For System Tests: Cucumber: To describe user behavior in .feature files; Capybara: To execute browser actions inside your Cucumber step definitions.

## 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.