**CS673 Software Engineering** 

**Team 2 - Rental Ninja**

**Project Proposal and Planning**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Yongjing Wu | Team Lead | *Yongjing wu* | 09/08/2024 |
| Jiacheng Ding | Security Lead | *Jiacheng Ding* | 09/08/2024 |
| Yueyang He | Design and Implementation Lead | *Yueyang He* | 09/08/2024 |
| Xueqi Zhou | Requirement Lead | *Xueqi Zhou* | 09/08/2024 |
| Rundong Zhong | Configuration Lead | *Rundong Zhong* | 09/08/2024 |
| Xiang Zhang | QA Lead | *Xiang Zhang* | 09/09/2024 |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **0.1** | **Yongjing Wu, Xueqi Zhou, Xiang Zhang** | 09/09/2024 | **Initial draft** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

[Overview](#_heading=h.gjdgxs)

[Related Work](#_heading=h.30j0zll)

[Proposed High level Requirements](#_heading=h.1fob9te)

[Management Plan](#_heading=h.3znysh7)

[Objectives and Priorities](#_heading=h.2et92p0)

[Risk Management (need to be updated constantly)](#_heading=h.tyjcwt)

[Timeline (need to be updated at the end of each iteration)](#_heading=h.3dy6vkm)

[Configuration Management Plan](#_heading=h.1t3h5sf)

[Tools](#_heading=h.4d34og8)

[Deployment Plan if applicable](#_heading=h.2s8eyo1)

[Quality Assurance Plan](#_heading=h.17dp8vu)

[Metrics](#_heading=h.3rdcrjn)

[Code Review Process](#_heading=h.26in1rg)

[Testing](#_heading=h.lnxbz9)

[Defect Management](#_heading=h.35nkun2)

[References](#_heading=h.1ksv4uv)

[Glossary](#_heading=h.44sinio)

# Overview

We are developing a rental website specifically designed for international students. The aim of this platform is to create an easy-to-use online platform that connects international students with verified rental properties, which can satisfy their needs. Like cost-effective and close to their university, internship, workplace, etc. The platform will offer specialized services such as browsing function, property filtering, property details, property post and so on. This will help alleviate the common challenges international students face when finding reliable and affordable housing in foreign countries.

# Related Work

Our platform is specifically designed for international students, offering a tailored rental experience with features like flexible lease terms, making it more suited to the unique needs of international students.

Unlike existing platforms such as Uniplaces and HousingAnywhere, which focus on broader student or expat markets. Student.com is another leading accommodation platform, offering property listings near universities globally. While it excels in offering a wide range of choices, it does not verify all properties, which may lead to safety concerns for international students unfamiliar with local housing markets. Our platform not only focuses on international student rentals, but we can also help international students with vacant homes, such as those that are vacant for the summer. Subletting vacant summer houses can minimize the loss of international students, and can also provide shorter rental periods for some summer internship students, while bringing convenience to both landlords and tenants.

# Proposed High level Requirements

## Functional Requirements

* + 1. User Registration and Authentication
       1. As a user, I want to create an account and log in securely so that I can access personalized features and saved listings.
    2. Property Search and Filter Capabilities
       1. As a student, I want to search for properties using various filters (e.g., location, price, furnished/unfurnished, property type, budget, proximity to educational institutions, and amenities.), so that I can find a place that matches my needs.
       2. Country and City-specific searches.
    3. Property post
       1. Allows renters to post new properties on the platform and the property’s details.
       2. Property Title: A brief title for the listing (e.g., "Spacious 2-Bedroom Apartment").
       3. Description: A detailed description of the property, including amenities, features, and any special requirements.
       4. Location: The full address or general area of the property.
       5. Rental Price: The monthly or weekly rent, including options for different pricing models (e.g., fixed, negotiable).
       6. Property Type: Select the type of property (e.g., apartment, house, studio, etc.).
       7. Number of Bedrooms/Bathrooms: Specify the number of bedrooms and bathrooms.
       8. Images/Videos: Upload multiple images or videos of the property.
       9. Availability Dates: Specify the start and end dates for when the property is available.
    4. Property list
       1. Allows users to edit or update existing property details.
       2. Access: The user accesses a list of their existing listings via a dashboard.
       3. Editing Interface: The user can update fields such as price, description, availability, or upload new images.
       4. Save Changes: After making changes, the rentuserer can save the listing, which updates the listing in real-time on the platform.
       5. Status Changes: The user can also change the listing’s status from “available” to “unavailable” or temporarily deactivate it.
    5. User Posted List (Desirable)
       1. 1. Allows users to create, update, and manage their personal profiles on the platform, including name, email, profile picture. etc.
    6. Favorites (Desirable)
       1. The Favorite Function allows users to bookmark or save properties they are interested in for easy access later. This feature enhances the user experience by allowing them to create personalized lists of properties without needing to search for them repeatedly.
       2. The User can see their collection rental list, just in case they want to find the exact one in the massive rental information
    7. Comments(Optional)
       1. Users can comment on any rental posts, and these posters also can comment.
       2. Users can see all the comments they made.
    8. Message (Optional)
       1. Users can message the poster to chat about the rental information.
       2. The poster can message back to the user who is interested in their rental information.

## Nonfunctional Requirements

* + 1. Security requirements

1. Data Encryption: All user data (personal details, payment information, etc.) must be encrypted using AES-256 encryption at rest and TLS 1.2 or higher for data in transit.
2. User Authentication: The platform must implement two-factor authentication (2FA) for all users logging in to their accounts.
3. Data Privacy: The platform must comply with GDPR regulations, ensuring users can request data deletion and control over their personal data.
   * 1. Performance
4. Response Time: The platform must load search results within 2 seconds of a query, even under a heavy load of 10,000 concurrent users.
5. Throughput: The system should handle up to 500 property listings being added per minute without degradation of performance.
   * 1. Usability
6. Ease of Use: New users should be able to complete the sign-up and profile setup process easily, without external assistance.
   * 1. Reliability
7. System Uptime: The platform must have 99.9% availability over a calendar year, allowing for no more than 8.76 hours of downtime annually.
8. Error Recovery: In case of failure, the system must automatically recover and resume normal operations within 60 seconds, with no data loss.
   * 1. Scalability
9. Horizontal Scalability: The system must be able to scale horizontally, adding more servers to handle increased traffic without requiring downtime or significant changes to the codebase.
10. Load Balancing: The platform must distribute incoming traffic across multiple servers using a load balancer, ensuring efficient resource usage and avoiding server overloads.
    * 1. Maintainability
11. Code Modularity: The platform’s codebase should be modular, allowing for independent updates and maintenance of various components without affecting the entire system.
12. Deployment Frequency: The platform must support continuous integration and deployment (CI/CD), allowing for frequent and safe updates at least twice a week without downtime.
    * 1. Availability
13. Backup and Disaster Recovery: Data backups must occur at least every 24 hours, and in the case of a disaster, the system must be able to recover and restore all services within 4 hours.
    * 1. Portability
14. Cross-Browser Compatibility: The platform must function properly on the latest versions of major web browsers (Chrome, Firefox, Safari, Edge).
15. Device Compatibility: The platform must be fully responsive and work seamlessly on a variety of devices, including smartphones, tablets, and desktops with varying screen sizes.

# Management Plan

## Objectives and Priorities

1. Deliver a fully functional and reliable Project Management Service that meets all essential requirements.

2. Maintain high code quality and test coverage for the Project Management Service.

3. Ensure proper integration and communication with other services, such as User Management, Collaboration, and Notification.

## Risk Management (need to be updated constantly)

The main risks identified for the Project Management Service include:

1. Complexity in handling project dependencies and concurrent updates from multiple users.

2. Challenges in integrating with other services, such as ensuring consistent data and communication protocols.

3. Potential performance issues or scalability concerns as the number of projects and users grows.

4. Security risks related to data breaches, unauthorized access, or vulnerabilities in the service.

**To manage these risks, we will:**

* Implement proper locking mechanisms and conflict resolution strategies for concurrent updates.
* Establish clear communication protocols and data contracts between services, and conduct thorough integration testing.
* Optimize the service for performance and scalability, and implement load testing and monitoring.
* Follow secure coding practices, conduct security testing, and regularly update dependencies and libraries.

**Risk Management Sheet Link: https://docs.google.com/spreadsheets/d/1SPGFYGi\_iIYRyxFFdpYmrjeRC2P4Ven5/edit?gid=1631798556#gid=1631798556**

## Timeline (this section should be filled in iteration 0 and updated at the end of each later iteration)

| Iteration | Functional Requirements(Essential/Disable/Option) | Tasks (Cross requirements tasks) | Estimated/real person hours |
| --- | --- | --- | --- |
| 0 | Initial | Environment Setup,  Initialisation,  Architecture Design,  Tech Stack,  Timeline,  Iteration Plan,  Document & Presentation | 7-8 |
| 1 | Essential:  1. User Register & Login  2. User Post & Browse  3. Rental Information Search | Design, Implementation, Testing,  Fixing,  Review,  Document & Presentation | 15-20 |
| 2 | Desirable:  1. Collection  2. User Posted List  Testing & Review:  1. Testing all Essentials 2.Requirements | Design, Implementation, Testing,  Fixing,  Review,  Document & Presentation | 15-20 |
| 3 | Optional:  1. Comment  2. Feedback  Testing & Check:  1. Security Testing  2. Requirements check | Fixing,  Review,  Document & Presentation,  Last Check,  Deployment | 7-8 |

# Configuration Management Plan

## Tools

| **Category** | **Tools** |
| --- | --- |
| **CI/CD Tools** | GitHub Actions |
| **Cloud Deployment** | AWS EC2, AWS S3, AWS DynamoDB, AWS ECR |
| **Containerization** | Docker |
| **Testing & Code Quality** | JUnit, Jest, Linters |
| **Monitoring & Error Tracking** | AWS CloudWatch, Sentry or AWS X-Ray |
| **Backup & Security** | AWS IAM, DynamoDB Backups |
| **Project Management** | Jira: https://rentandhousing.atlassian.net/jira/software/projects/SCRUM/boards/1 |

## Deployment Plan if applicable

### Deployment Plan Overview

* CI/CD Tools: GitHub Actions for Continuous Integration (CI) and Continuous Deployment (CD)
* Cloud Platform: AWS Cloud (EC2, S3, RDS, etc.)
* Technology Stack:
  + Frontend: Vue.js
  + Backend: Java (AWS Lambda)
  + Database: DynamoDB
  + Cache: Redis

### 1. Infrastructure Setup (AWS)

* AWS EC2 (Elastic Compute Cloud): Set up EC2 instances to host the backend services (Java application).
* AWS S3 (Simple Storage Service): Use S3 to host static content (images, frontend assets).
* AWS DynamoDB: Used for storing data, set up tables based on your application's data structure.
* AWS Redis (Elasticache): Set up Redis for caching to improve application performance.

### 2. Environment Configuration

* Development Environment:
  + AWS EC2 instance with security groups configured to allow access for developers.
  + Local Docker containers for testing changes before committing.
* Staging Environment:
  + EC2 instances for pre-production testing.
  + S3 for static content.
  + DynamoDB and Redis (Elasticache) replication.
* Production Environment:
  + Multi-AZ EC2 instances.
  + S3 for static content.
  + DynamoDB for high availability.

### 3. CI/CD Pipeline (GitHub Actions)

* CI Pipeline:
  + Linting and Formatting:
    - Set up linters for Java, Vue.js, and any other languages to ensure code quality.
    - Use GitHub Actions to automatically run these checks on pull requests.
  + Unit Testing:
    - Write unit tests for backend (JUnit for Java) and frontend (Jest for Vue.js).
    - Configure GitHub Actions to run these tests on every push and pull request to ensure code functionality.
* CD Pipeline:
  + Build the Application:
    - Use GitHub Actions to build the frontend (Vue.js) and backend (Java) code.
    - For the frontend, compile the Vue.js app and upload the assets to S3.
    - For the backend, build the Java app using Maven/Gradle and create a Docker image.
  + Dockerization:
    - Create Dockerfiles for the frontend and backend.
    - Use GitHub Actions to build Docker images for both frontend and backend.
  + Push Docker Images to AWS ECR (Elastic Container Registry):
    - Authenticate with AWS ECR.
    - Push the Docker images to ECR for use in deployment.
  + Deploy to AWS:
    - Use AWS Elastic Beanstalk or EC2 instances to deploy the backend services.
    - Deploy the frontend by syncing the built Vue.js files to the S3 bucket.
    - Use GitHub Actions to automatically deploy to the AWS environment (staging or production) when a branch is merged.

### 4. Branching Strategy

* Git Branch:
  + Dev: the development branch for integration of both frontend and backend
  + Staging: The staging branch which is used to do pre-release and integration testing
  + Main: main branch, used for release
  + Frontend-dev: used for frontend development
  + Backend-dev: used for backend development
* Git Flow:
  + Dev->Staging->Main
* PR Policy:
  + The main branch and staging branch are protected. You cannot force merge these branches locally. You can only submit a pull request from git. The pull request needs to be approved by at least 2 people.
  + The dev branch has the same policy, the only difference is the pull request needs to be approved by at least 1 person.

### 5. Deployment Strategy

* Staging Deployment:
  + Automatically deploy to the staging environment upon merging pull requests into the develop branch.
  + Run integration tests in the staging environment to ensure everything works as expected.
* Production Deployment:
  + Use the main branch for production deployment.
  + Manually trigger deployments from GitHub Actions or set up automated triggers upon merging into the main branch.
  + Deploy frontend assets to S3 and the backend to EC2 instances or Elastic Beanstalk.

### 6. Monitoring and Rollbacks

* Monitoring:
  + Set up AWS CloudWatch to monitor the health of EC2 instances, DynamoDB, and Redis.
  + Set up alarms for high CPU usage, latency, or downtime.
* Error Tracking:
  + Integrate Sentry or AWS X-Ray for error tracking.
* Rollback Mechanism:
  + Use GitHub Actions to maintain version history, and allow for rollbacks if a deployment fails.
  + Leverage AWS Elastic Beanstalk for easy version control of deployments.

### 7. Security and Access Control

* Access Control:
  + Use AWS IAM roles to control access to AWS services.
  + Restrict GitHub Actions access with security keys to deploy only from trusted branches (e.g., main, develop).
* Data Encryption:
  + Ensure that sensitive data in DynamoDB and Redis is encrypted at rest.
  + Use HTTPS for all frontend/backend communication.

# Quality Assurance Plan

## Metrics

(Describe the metrics to be used in the project to measure the quality of your software. Each metric should be measurable and quantifiable. Examples of metrics include product complexity (LOC, # of files, # of classes, # methods, cyclomatic complexity, etc.) , defect rate (# of defect per KLOC), # of test cases, test case pass rate, cost (# of person hours used), # of user stories completed, etc. **The result of these metrics should be reported in the progress report/ iteration summary sheet.**)

| Metric Name | Description |
| --- | --- |
| Number of Test Cases | The total number of test cases created for the project. |
| Test Case Pass Rate | The percentage of test cases that pass successfully during testing. |
| Lines of Code | The total number of lines of code written, including the number of files, classes, and methods. |
| Defect Rate | Number of defects found per KLOC. |
| Test Coverage | Percentage of code covered by tests (target: 30%). |

* 1. Coding Standard

(Describe any coding standard to be used)

**Frontend (Vue)**

1. Naming: camelCase for variables/methods, PascalCase for components.

2. Comments: Public methods only. Inline for complex logic.

3. Error Handling: Use Vue error handling. User-friendly messages.

4. Formatting: 2 spaces for indentation. No trailing spaces.

5. Structure: Separate components, services, Vuex store.

**Backend (Java)**

1. Naming: camelCase for methods/variables, PascalCase for classes.

2. Comments: Public methods only. Inline for complex logic.

3. Error Handling: Use try-catch. Log meaningful messages.

4. Formatting: 4 spaces for indentation. No trailing spaces.

5. Structure: Separate controllers, services, models.

## Code Review Process

(Everyone should review all documents to be submitted. Here you will mainly describe how the code review will be done. Who will review the code, e.g. design or implementation leader will review all code or team members review each other’s code. Do you use pull requests for the code review? Is there a checklist to help review? What feedback should the reviewer provide?)

**Reviewer:**

Team members review each other’s code. The design or implementation leader reviews critical sections.

**Review Method:**

Code reviews are done via pull requests. The reviewer ensures that the code follows coding standards and meets functionality requirements.

**Checklist:**

1. Adheres to naming conventions.

2. Proper error handling.

3. Test cases provided.

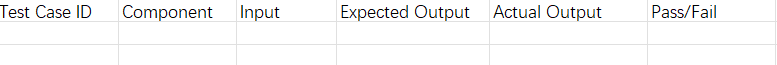
4. No duplicate or unused code.

**Feedback:**

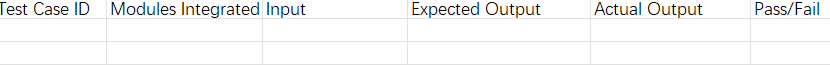
The reviewer provides feedback on logic, readability, and optimization. Any requested changes should be specific and actionable.

## Testing

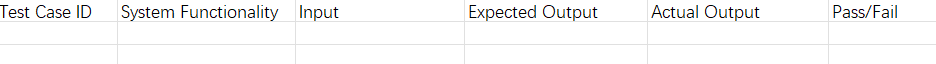
* + 1. Unit Testing
       1. Description: Test individual components or modules of the code to ensure they function correctly.
       2. Tools: JUnit



* + - 1. Objective: Ensure each part of the system works independently.
    1. Integration Testing
       1. Description: Test the interactions between different modules or components to ensure they work together as expected.
       2. Tools: Postman



* + - 1. Objective: Verify that different components of the software function together correctly.
    1. System Testing
       1. Description: Test the complete system as a whole to ensure it meets the specified requirements.
       2. Tools: Selenium



* + - 1. Objective: Ensure that the system behaves as expected under real-world scenarios.
         1. Functional Testing:

Ensures that the system's features work according to the specified requirements by testing each function of the software application.

* + - * 1. User Interface (UI) Testing:

Verifies that the interface is intuitive, responsive, and works correctly across all elements, ensuring users can navigate and interact with the system easily.

* + - * 1. Performance Testing:

Measures how the system performs under expected workloads, ensuring it can handle high volumes of users, transactions, or data without performance degradation.

* + - * 1. Compatibility Testing:

Ensures the software works across various environments, such as different browsers, devices, operating systems, or network configurations, to maintain broad usability.

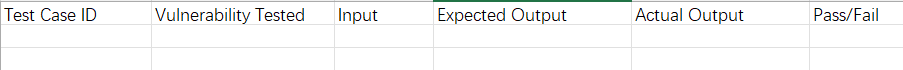
* + - * 1. Security Testing:

Identifies potential vulnerabilities or weaknesses in the system to ensure data protection, user privacy, and overall system security against threats or unauthorized access.

* + 1. Regression Testing
       1. Description: Re-run previous tests after changes have been made to ensure that no new bugs have been introduced.
       2. Tools: Selenium



* + - 1. Objective: Ensure that existing functionalities still work after updates or changes.
    1. Security Testing
       1. Description: Check for vulnerabilities in the system, ensuring that data and resources are protected from potential threats.
       2. Tools: OWASP ZAP, Burp Suite.



* + - 1. Objective: Ensure the system is secure and protect it from malicious attacks.

## Defect Management

* + 1. Tool for Defect Management:
       1. We will use Jira Issues to track, document, and manage defects throughout the project. This tool will allow us to assign defects to team members, prioritize issues, and link them to relevant code changes or commits for easier tracking and resolution.
    2. Types of Defects
       1. **Functional Defects**: These occur when a feature or functionality does not work as intended, such as broken user flows or incorrect calculations.
       2. **UI/UX Defects**: Issues related to the user interface, such as layout problems, poor responsiveness, or confusing navigation.
       3. **Performance Defects**: These refer to any issues that affect the speed, scalability, or overall performance of the system.
       4. **Security Defects**: Vulnerabilities in the system, such as weak authentication, data leakage, or susceptibility to attacks like SQL injection.
       5. **Integration Defects**: Problems that arise when different modules or systems do not interact as expected, leading to data inconsistency or failure in communication.
       6. **Compatibility Defects**: Issues that occur when the system does not function properly across different browsers, devices, or operating systems.

# References

1. Sommerville, I. (2016). Software Engineering (10th ed.). Pearson Education.

# Glossary

(Any acronym used in the document should be explained here)