System Requirement Specifications (SRS)

RoomieLah

A roommate matching application for university students

Submitted to—
Shi Hanyu
School of Computer Science and Engineering
Nanyang Technological University

Delivered By:
StrawHats

Arora Srishti
Pandey Pratyush Kumar
Surawar Sanath
Iyer Rajagopal Mahadevan Acharya Atul
Tayal Aks
Agarwal Gopal

1 Table of Contents

2 Problem Statement

Singapore is home to hundreds of thousands of students who are studying in one of the thirty-four universities present. This includes both – local students and international students. Students usually stay in either university halls of residence or co-living apartments. For students living in a double room in university hall or co-living apartments, their roommates are significant to their living experience. This is because they spend a considerable amount of time with them and influence each other’s lives strongly – both positively and negatively. This could also result in them affecting each other’s mental health.

At present, if a student applies for a double room or a co-living apartment, they end up living with someone random. This leads to a very high chance of two not very compatible students ending up living together which might lead to unpleasant experiences. This could be a result of differences in four variables – personality, values/attitude, religious and socioeconomic background, and living habits. To elaborate, the roommates could have different sleeping patterns which could end up affecting their quality of sleep, or one of them is not considerate about the cleanliness of the room which bothers the other, or one of them is an extrovert who has guests over often resulting in disturbing the other, or in general, one of them doesn’t respect the roommate’s cultural or socio-economic background.

As a result of differences in some of these variables, conflict may arise which would affect an individual in multiple ways. Firstly, it would negatively affect one’s mental health as there is a significant correlation between roommate conflict and stress levels. Secondly an experiment conducted by Dr. J.K. Coleman concludes that a student with a random roommate has a lower GPA than a student with a roommate whom they met online which in turn, is lower than a student with a roommate whom they have a prior relationship with. Also, a study by Michigan State University identified roommate conflicts as one of the major factors for students dropping out of college. Therefore, it is imperative to develop an innovative and effective solution to mitigate this problem as soon as possible.

Team StrawHats consists of seven students, all of whom have faced a similar problem in university. This will be an added advantage to the ample amount of experience the team has in software development, resulting in building of an easily maintainable and user-friendly solution.

3 Overview

3.1 Background

Room allocation has always been a tedious task for both universities as well as students. All universities give students the option to apply with a friend for a room, and if they don’t, then the university randomly assigns a roommate to them. Since most of the incoming freshmen (and even some seniors) may not know anyone on campus, they end up living with a random roommate who they have never met before, which obviously leads to a multitude of cases where the students aren’t compatible with each other due to differing lifestyle habits.

With increasing dissent regarding the allocation system and a rise in cases of shifting rooms mid-semester due to domestic problems, there is an urgent need for a social media service dedicated to help students find potential matches for roommates.

3.2 Overall Description

RoomieLah is essentially a social media application for all university students in Singapore, where they can set their personal preferences, based on which the matching algorithm finds users to match them with. The preferences are stored in the database, and the matches are updated whenever they are changed. Once users are successfully matched with each other, they are allowed to use the in-app chat feature to further decide compatibility. User data is always encrypted to ensure maximum user privacy, and only university students are allowed to register as users on the application using their university email-ids.

4 Investigation & Analysis Methodology

4.1 System Investigation

RoomieLah’s system follows state-of-the-art practices and industry standards for the entire flow of the application, to ensure the user has an enriching experience. When the user opens the application, he/she has to login with their university credentials. Upon successful authentication, the user is led to a user profile page where he/she can fill in their personal details, upon the completion of which, they are asked tailored preferential questions that help the application better gauge the roommate that they are searching for. These details are securely stored in the backend database service in Cloud Firestore, provided by Google Firebase, so they can be retrieved as and when needed in the future.

The user is then redirected to the main matching page of the application, where potential roommates are displayed to the student based on his/her preferences and a corresponding match with other app users. The matching is carried out in the backend and on the frontend, each prospective roommate is displayed on the whole screen, to which app users can choose to either swipe left (indicating non-likeability towards listed match) or swipe right (indicating likeability towards listed match). If the person that the current application user has swiped right on would also swipe right back, a match notification is displayed to both users. Upon matching, the chat feature of the app is invoked wherein newly matched prospective roommates are allowed to chat to better understand their needs, preferences, and way of living to make a rational decision. This data of matches is also stored in Firebase, which updates real-time to provide the user with up-to-date information.

The user also has the provision to update their profile via edit profile and preferences tab that allows them to update their initial responses and biodata to optimize results that the matching algorithm can provide them with. This interfacing is meant to provide the users with the most accurate and recent results to provide a more well-rounded application.

4.2 Analysis Methodology

4.2.1 Feasibility study and requirements elicitation

To make RoomieLah relevant to users, detailed surveying and social research has been carried out. The UX and Research Team reached out to our audience i.e., university students on the lookout for roommates with the target audience being university freshmen who were asked if they would have preferred to have a say in their roommate allocation. This informational data serves as meta-data to implement and realize the application that the team is committed to building.

The Requirements Elicitation will also include specific data from interviews the team conducts with the target audience to understand underlying problems and how the application interface with its functionalities can hope to solve it. The team would also understand the volume of prospective international students who would require housing upon arriving at universities in Singapore.

The application's recommendation system would ultimately innovate its matching algorithm based on intricate responses and pain points that the target audience introduces, which can serve as probing questions in the preferences subsegment of our application, that would allow the algorithm to get more parameters to determine a best fit for each person's living needs.

4.2.2 System analysis and requirements specification

Scope

1. Users use assigned university email ID for authentication and validation to link with their RoomieLah account.

2. Construct secure, reliable, and efficient communication between the frontend, backend, and database components in RoomieLah.

3. Provide accurate and personalized recommendations to each user based on their input preferences and matching algorithm.

4. Enable users to chat with people they match with and provide for additional resources to connect with prospective roommates in a foreign university.

5. Enable users to rate the application and app workflow to better gauge likeability and reactions from the audience.

Limitations

1. RoomieLah's system could be prone to crashes in case there are several users concurrently using the application. Load testing could help overcome the issue.

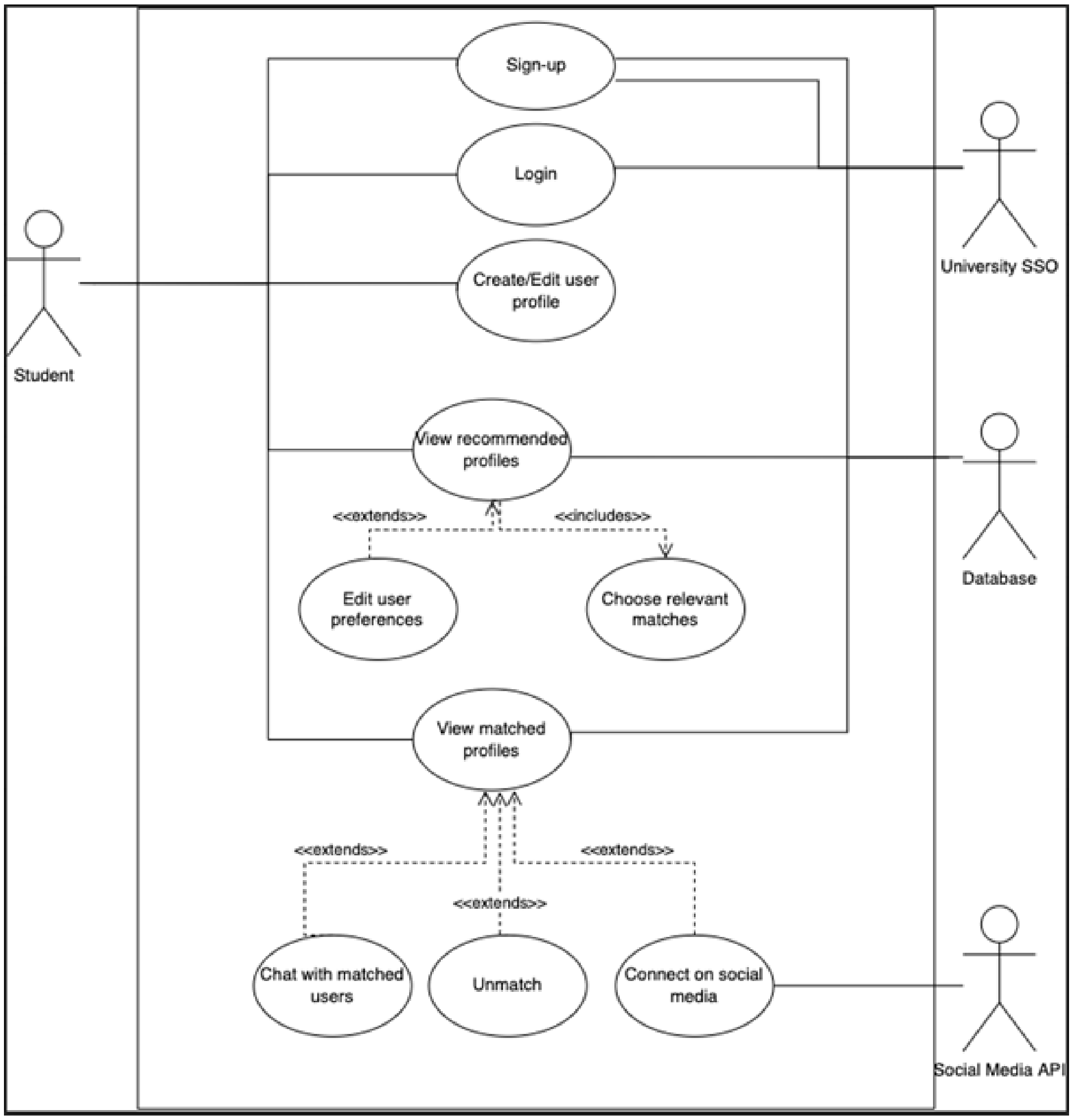
2. If the database communication is not working properly, any changes will not reflect when the user logs in on a new session. This can lead to repetitive work as the user might have to update their preferences continuously until it reflects on updating.

3. The database is susceptible to race conditions when two users simultaneously attempt to perform similar actions that update the same section of the database. Appropriate alternatives like using semaphores, locks etc. will be needed to overcome this issue.

4.2.3 Object-oriented design using UML

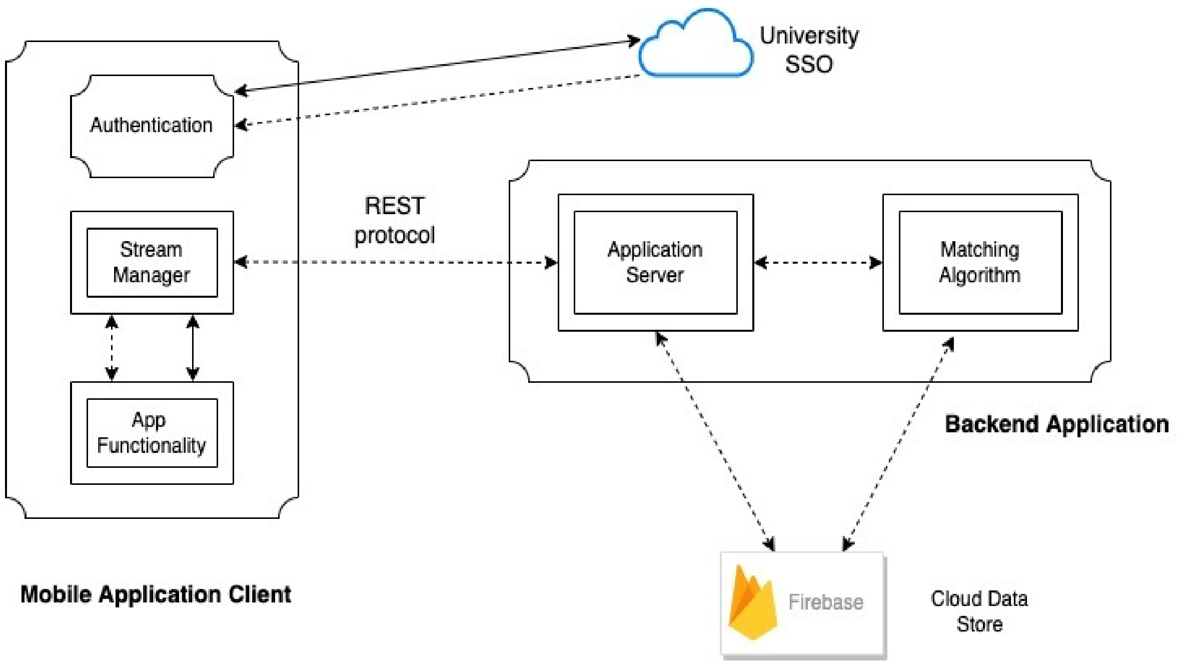
UML would be the language chosen to design an intricate and specific object-oriented design. The system is primarily a personalized recommendation system that works on user’s personal preferences to assign to them someone, also registered on the application, who is matched to the user based on the matching algorithm. The system fetches data from a MySQL server which stores the entire data for all user’s preferences, which serves as the basis for matching. The system is secured with a university email verification account of the user. All industry-standard object-oriented practices such as modularization, the MVC (Model-View-Controller) structure for the code, low coupling using concepts of inheritance etc. will be employed to maintain high quality of the code making it easier to add, remove and reuse certain sections of the code for implementing additional functionalities and updating existing ones. The system will overarchingly consist of objects, each for a different functionality, that communicate with each other for achieving a particular requirement.

4.2.3.1 Use Case 1



4.2.4 Prototyping

Object Oriented Evolutionary Prototyping will be used to provide a robust prototype for RommieLah. It will be thoroughly reviewed by Developers, the Quality Assurance manager and Project Manager to ensure the final product has a fine quality. The prototype will be fully functional and would include most of the planned features. It will also be used for an interim/beta release before the final product is launched into the market. This beta version will be used to collect feedback from potential users to improve the product and catch any unnoticed bugs.



The system architecture follows a Client-Server model. It is the architecture in which many clients (remote processors) request and receive service from a centralized server (host computer). Client computers provide an interface to allow a computer user to request services of the server and to display the results the server returns. Servers wait for requests to arrive from clients and then respond to them.

The mobile application client includes the authentication feature that is implemented with a university SSO to ensure only valid university students can login. The stream manager works in sync with the application functionality, which communicates with the backend application through Representational State Transfer protocols. The backend architecture broadly comprises the application server and the proprietary matching algorithm, both of which store data in Firebase, a cloud data store that represents a real-time database.

Initial UI Mockups

Figure 1: Start Screen

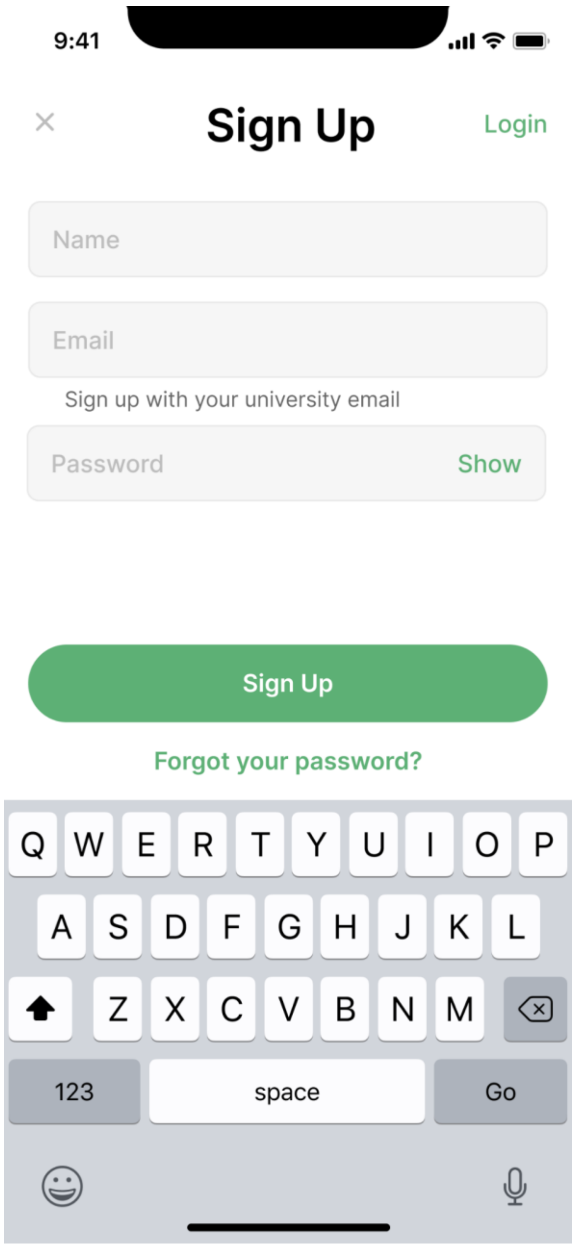


Figure 2: Sign Up Screen

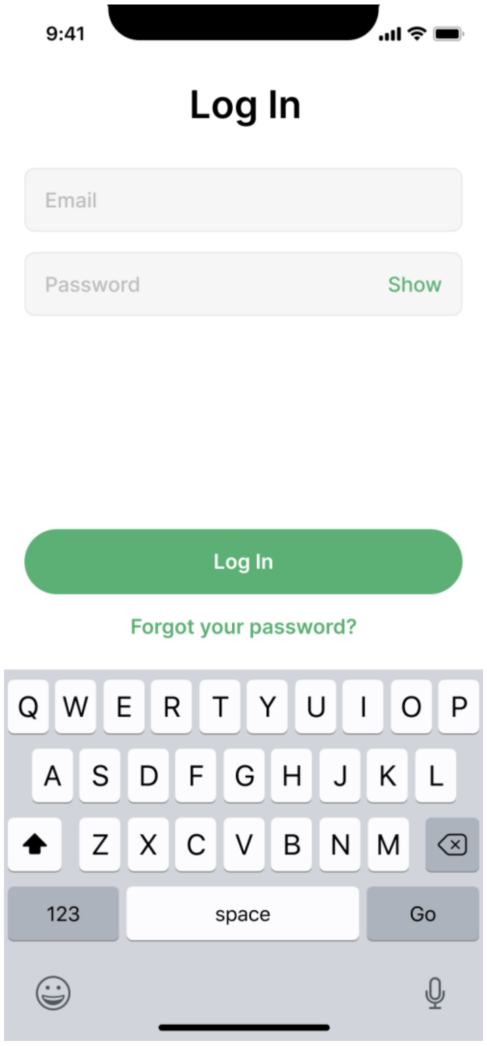


Figure 3: Log in Screen

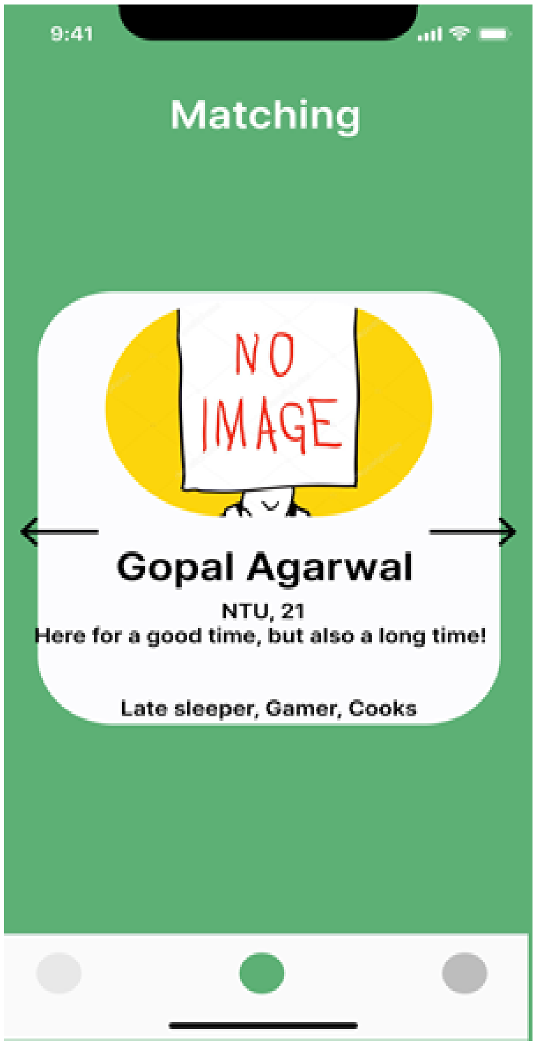
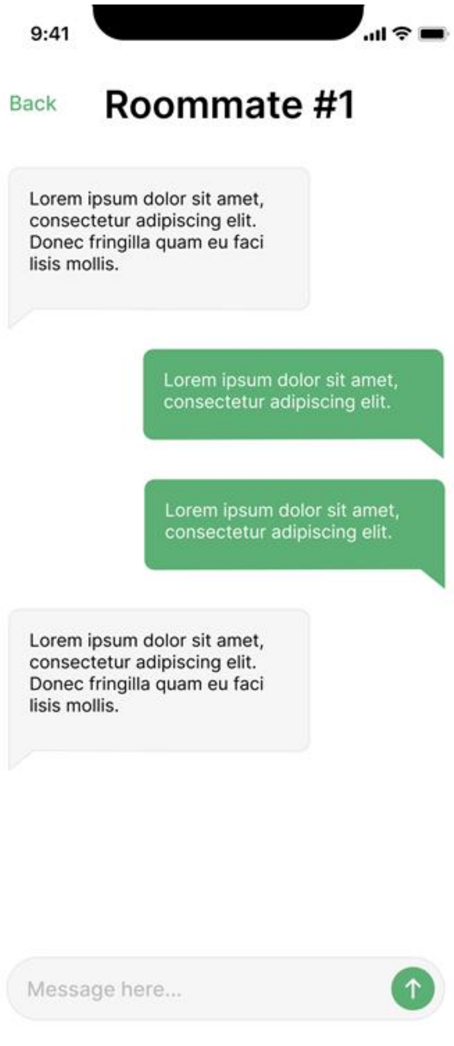

9:41
Matching
NO IMAGE
Gopal Agarwal
NTU, 21
Here for a good time, but also a long time!
Late sleeper, Gamer, Cooks

Figure 4: Viewing different profiles



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec fringilla quam eu faci lisis mollis.

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec fringilla quam eu faci lisis mollis.

Figure 5: Chat Screen

5 Constraints

5.1 Scalability

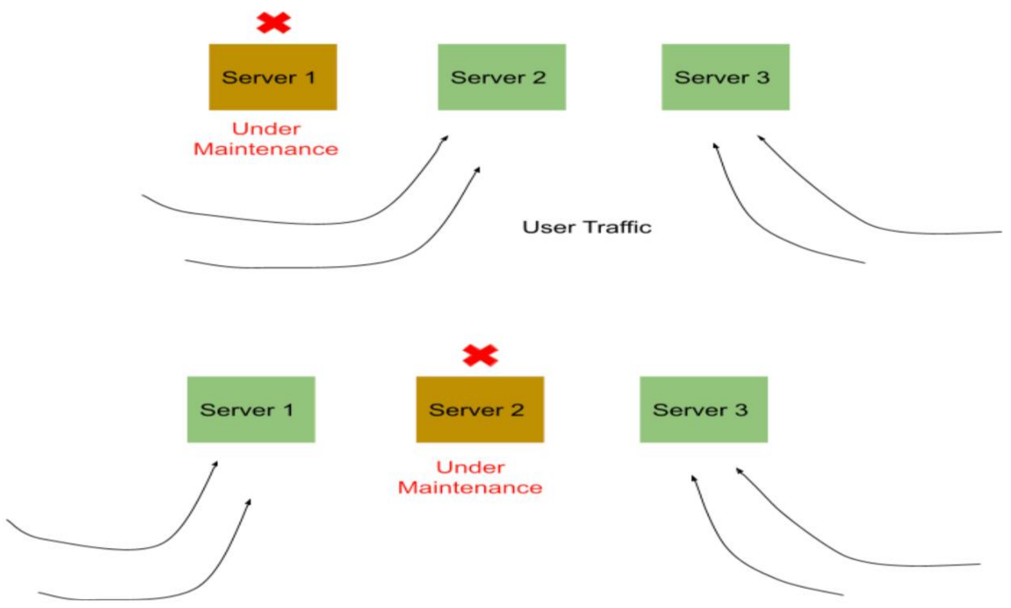
Database: To minimize and prevent future complications, an Object Store Database - Google Firebase will be used. Using an object store database will enable us to make quick changes. The APIs of Firebase are designed to scale linearly with the size of data being synchronized. As the number of users using RoomieLah increases, the object store database will help us scale the software without complicating the data.

Code: Writing thoughtful and clean code will be prioritized because a high-quality code input produces longer-lasting output. Additionally, it is easier to duplicate and test in the future. The code quality will be improved by using SOLID principles with emphasis on Single Responsibility Principle. This means, each module or class within the application is only responsible for a single functionality. This allows low coupling between each module and therefore, increasing the flexibility of the application significantly. Flexibility and modularity results in high scalability, allowing new functionalities to be added into the application with relative ease in the future.

Maintenance: Automated testing will set across functionalities to efficiently illuminate the bugs in our system and will make the process much faster and manageable. This will ensure that our software is convenient for testing and maintenance even when it undergoes countless improvements and destructions.

5.2 Data and Function Mapping

The software will be deployed on docker which will create multiple replicas. Normally, the user traffic will be randomly directed to one of the different replicas. When a new version needs to be deployed, the service of replicas will be stopped one at a time and the user traffic will be routed to the remaining servers. Meanwhile, the latest program will be loaded into the server, which is not functioning, and recompilation of code will take place. Service would then be resumed on Server 1 and monitored for any crashes or potential bugs. It will be gradually released to the remaining replicas in a similar fashion. This would ensure that the service is always available to the customers.



5.3 Proprietary hardware and software

The proprietary hardware tools for the application includes standard mobile devices for implementation and testing of the application. As the application is to be run on users' devices locally, there will be no proprietary software tools involved in this project.

5.4 Batch updates vs. (close) Real-time updates

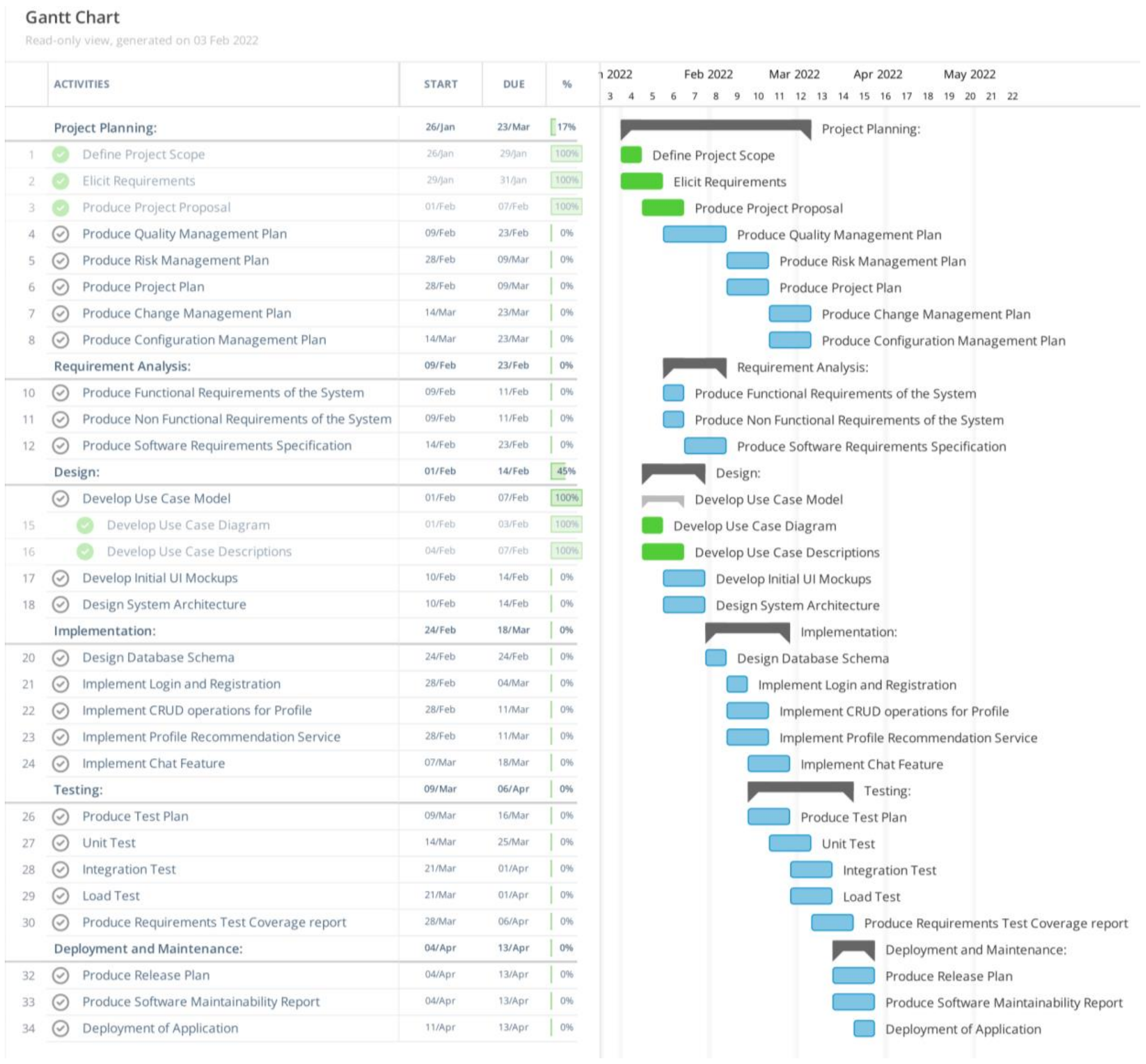
RoomieLah requires registration of all users for authentication purposes. All user data is synchronized via Cloud Firestore database system on a close to real-time basis. Other data such as potential matches, their profiles, and chats are displayed and updated in real time as well with the help of event listeners.

Regardless of any manual backups, the system is configured to perform a daily backup of the database, assigned to a specific hour each day. The data is transferred to our Google Cloud Storage bucket and timestamped in ISO 8601 standard with specific naming convention via a scheduled job.

5.5 Project Schedule

There is a four-month timeframe to implement a production system of a roommate matching system from project commencement in time for AY 2022-23 Hall Applications. Prototypes of the UI will be launched before the final product is released to receive user feedback and testing results. This prototype is scheduled to be released after 1 month, where it will include basic functionalities of the application to showcase the application’s interface and demonstrate how the application will interact with the users.

A Quality Management Plan is produced to help guide the Program Manager (PM) and project personnel to execute quality management and quality assurance activities for the project. All development activities are commenced only after the Quality Management Plan is produced. A Project plan and Risk Management Plan is developed during the early stages of the implementation phase. A comprehensive Test Plan is produced towards the end of the implementation phase. Testing of each feature is carried out only after the complete implementation of the feature. Testing will be carried out in a bottom-up approach where integration tests and load tests are performed only after the completion of unit tests. A test convergence report is submitted once the system has met all the test requirements. A Change and Configuration Management Plan is developed after the implementation of the system. A Release Plan and Software Maintainability Report is produced after the System has been thoroughly tested. The final application is then deployed after passing Quality assurance and user acceptance tests.


Gantt Chart

Read-only view, generated on 03 Feb 2022

	ACTIVITIES	START	DUE	%
	Project Planning:	26/Jan	23/Mar	17%
1	Define Project Scope	26/Jan	29/Jan	100%
2	Elicit Requirements	29/Jan	31/Jan	100%
3	Produce Project Proposal	01/Feb	07/Feb	100%
4	Produce Quality Management Plan	09/Feb	23/Feb	0%
5	Produce Risk Management Plan	28/Feb	09/Mar	0%
6	Produce Project Plan	28/Feb	09/Mar	0%
7	Produce Change Management Plan	14/Mar	23/Mar	0%
8	Produce Configuration Management Plan	14/Mar	23/Mar	0%
	Requirement Analysis:	09/Feb	23/Feb	0%
10	Produce Functional Requirements of the System	09/Feb	11/Feb	0%
11	Produce Non Functional Requirements of the System	09/Feb	11/Feb	0%
12	Produce Software Requirements Specification	14/Feb	23/Feb	0%
	Design:	01/Feb	14/Feb	45%
	Develop Use Case Model	01/Feb	07/Feb	100%
15	Develop Use Case Diagram	01/Feb	03/Feb	100%
16	Develop Use Case Descriptions	04/Feb	07/Feb	100%
17	Develop Initial UI Mockups	10/Feb	14/Feb	0%
18	Design System Architecture	10/Feb	14/Feb	0%
	Implementation:	24/Feb	18/Mar	0%
20	Design Database Schema	24/Feb	24/Feb	0%
21	Implement Login and Registration	28/Feb	04/Mar	0%
22	Implement CRUD operations for Profile	28/Feb	11/Mar	0%
23	Implement Profile Recommendation Service	28/Feb	11/Mar	0%
24	Implement Chat Feature	07/Mar	18/Mar	0%
	Testing:	09/Mar	06/Apr	0%
26	Produce Test Plan	09/Mar	16/Mar	0%
27	Unit Test	14/Mar	25/Mar	0%
28	Integration Test	21/Mar	01/Apr	0%
29	Load Test	21/Mar	01/Apr	0%
30	Produce Requirements Test Coverage report	28/Mar	06/Apr	0%
	Deployment and Maintenance:	04/Apr	13/Apr	0%
32	Produce Release Plan	04/Apr	13/Apr	0%
33	Produce Software Maintainability Report	04/Apr	13/Apr	0%
34	Deployment of Application	11/Apr	13/Apr	0%

Figure 6: Gantt Chart of schedule

6 Operational Requirements

6.1 User Support

There must be a support section in the app which will cater to users who have any issues/concerns such as account lock-out assistance, application errors, server downtime enquiries etc. This section must have a Frequently Asked Questions (FAQ) to address the most common issues/concerns. In case this doesn't provide a satisfactory solution, then the user will be able to send an email explaining the issue which will be solved by the official support team as soon as possible.

6.2 Application Services and Technical support

The application must request for user feedback through which issues can be shared by the user. When the user logs into the application for the first time, there must be a pop-up asking whether the user will be willing to share crash reports. If the user opts in, it can enable the developers to identify issues and further improve the end-user experience.

It is very important to ensure that the application is always up and running. So, the support of the network engineer as well as the database administrator is imperative to ensure maximum availability.

6.3 System hardware fail over and routine back up

An IT operations team must be set up, whose primary task is to ensure business continuity in case of unforeseen incidents. They must have a clear disaster recovery and business continuity plan in place. They must also take backups of all the data every week on a fixed day. On top of taking backup, they must also ensure that the backup is verified and can be successfully used to get back the complete data without any issues.

6.4 Audit Trail

An audit trail will be maintained which will have all the actions taken by users with a timestamp. Actions include, but not limited to, account creation, profile modification, preference modification, liking a profile, matching with another profile, unmatching a profile etc.

6.5 Security

The app has a one-to-one messaging feature. All the messages sent in the chat must be end-to-end encrypted which will ensure that any eavesdropper would not be able to view or modify the messages. The messages sent from one user will be temporarily stored in the server until the second user has received it, after which it must be deleted from the server side. However, it is important to note that only the actual message will be deleted, and the audit log will reflect no such changes.

Code signing must be implemented so that any unauthorized changes are immediately identified and notified. This will ensure that users don’t download a modified version of the application which might contain some malicious code. The only official place from which a user can download this application are the Google Play Store and the Apple App Store.

7 Functional Requirements

7.1 User Registration

7.1.1 The system must allow the user to register when the user opens the application for the first time.

7.1.1.1 The user must be able to click on the ‘Register’ button.

7.1.1.2 The user must enter his/her university email ID and a password.

7.1.1.3 The system must verify that the provided email ID is a university email ID (based on the domain) and the email ID is not already registered in the platform.

7.1.1.3.1 If valid, then the system must send a verification link to the registered email ID.

7.1.1.3.2 If invalid, then there must be an "Invalid Email ID" error displayed.

7.1.1.4 User must verify his/her university email.

7.1.1.3.1 User must click on the verification link sent to the registered email ID.

7.1.1.5 The system must finish the registration process after the user clicks on the verification link sent to the registered email ID.

7.2 User Login

7.2.1 The system must allow the user to login when the user has valid credentials.

7.2.2 The user must enter the registered email ID and the password to log into the application.

7.2.3 The system must verify the provided credentials and ensure it is valid.

7.2.3.1 If the given credentials are valid, then the system must log in the user into the application.

7.2.3.2 If the given credentials are invalid, then there must be an “Invalid Credentials” error message displayed.

7.2.4 The system must allow the user to change the password in case the user does not remember it.

7.2.4.1 The user must enter the registered email ID to receive a password reset link.

7.2.4.2 The system will verify whether the provided email ID is already registered.

7.2.4.2.1 If it is registered, then a password reset link is sent to the email ID to change the password.

7.2.4.2.2 If it is not registered, the system must not send any reset link. There is no explicit message shown to ensure that the information that there is or isn't a user with that email ID registered in the application accessible to everyone.

7.2.4.3 The system will display the message "Reset link sent if your account exists"

7.3. Create User Profile

7.3.1 The system prompts the user to create a user profile for identity.

7.3.1.1 The user must enter certain credentials to make a personalized user profile (biodata and personal preferences)

7.3.1.2 The user must be able to click on the Create Profile Button.

7.3.1.3 The system must be able to verify if the entered credentials are valid.

7.3.1.3.1 If the entered credential is valid, say age is entered as 18, the system must validate the input.

7.3.1.3.2 If the entered credential is invalid, say invalid age is input, an error message must be displayed.

7.3.2 The system displays a generated user profile based on user inputs.

7.4 Edit User Profile

7.4.1 The system must allow the user to edit his/her user profile.

7.4.1.1 The user must click on the Edit User Profile button.

7.4.1.2 The system allows the user to edit initial profile details while performing the same validation checks to inputs.

7.4.1.3 The system must be able to verify if the entered credentials are valid.

7.3.1.3.1 If the entered credential is valid, say age is entered as 18, the system must validate the input.

7.3.1.3.2 If the entered credential is invalid, say invalid age is input, an error message must be displayed.

7.4.2 The system generates a newly updated user profile based on user inputs.

7.5 Edit Profile Preferences

7.5.1 The system must allow the user to change their preferences for their roommate.

7.5.1.1 The user must click on the “Edit Profile Preferences” button.

7.5.1.2 The system allows the user to edit their responses to questions concerning their roommate preferences.

7.5.1.3 The user is allowed to edit initial responses to any preference question.

7.5.1.4 The user clicks on the “Confirm Changes” button or on the back button if no change is to be done.

7.5.1.5 The system updates the profile preferences of the user with the necessary changes

7.5.2 The system displays appropriate profiles to users reflecting the changes to their preferences.

7.6 Recommended Profiles

7.6.1 The system must display profiles that match the user's profile and preferences.

7.6.1.1 The displayed profiles can contain the following information: profile picture, name, age, gender, bio, hobbies, preferences, and social media links.

7.6.1.2 The system should not display information marked as sensitive or hidden by a user.

7.6.1.2 The user must be displayed profiles dynamically by the matching algorithm in case of any change in his/her preferences.

7.6.2 The user must be given the option to “like” profiles (potential roommate matches) or “skip” displayed profiles.

7.6.2.1 The user must be able to click a button or swipe right to express his/her interest.

7.6.2.2 The user must be able to click scroll through, or swipe left if they are not interested in a profile.

7.6.2.3 The system must keep checking if 2 users have expressed interest in one another as roommates.

7.6.2.3.1 The system should send a push notification to the offline users displaying the message “You have matched with <Name of the matched user>”.

7.6.2.3.2 The system should display a pop-up message: “It's a match” to the online users

7.6.2.3.3 The matched users must be able to chat with one another using the application.

7.6.2.3.4 The matched users must be able to discontinue chatting or delete conversations (un-match).

7.8 Chat

7.8.1 The system must allow the user to view a list of all users who they match with.

7.8.1.1 The system must allow the user to select one user from that list and open a chat page.

7.8.2 The system must allow the user to exchange real time messages with other users who are a potential match.

7.8.2.1 The system must display an onscreen keyboard and a send button on the chat screen.

7.8.2.2 The system must display “typing…” when the other user is typing a message.

7.8.2.3 The system must display “seen” under the text bubble, if the other user has read the bubble.

7.8.3 The system must store the chat history with all users and load it whenever the chat page is loaded

7.8.4 The system must delete chat history and remove a user from the list of potential matches if the user is unmatched.

7.9 Connect on Social Media Platforms

7.9.1 The system must allow the user to view a list of all users who they match with.

7.9.1.1 The system must allow the user to select one user from that list and view their profile.

7.9.1.2 The system must display the URL to the social media account of the user, if it exists.

7.9.1.3 The system must allow the user to click on that URL.

7.9.2 The system must open the URL on the default web browser of the device or on the social media app if it is already installed on the device.

7.9.2.1 In case the URL provided on the application does not exist, the system must display an error message.

7.10 Unmatch with users

7.10.1 The system must allow the user to view a list of all users who they match with.

7.10.1.1 The system must allow the user to select one user from that list and open a chat page.

7.10.1.2 The chat page must have an unmatch button at the top of the chat page.

7.10.2 The user must be able to unmatch with the matched user on clicking on the unmatch button.

7.10.2.1 The system must prompt the user for confirmation before performing the unmatching.

7.10.3 The system must remove the unmatched user from the matched users list once the unmatch button is clicked.

7.10.4 The system must clear the chat history between the users once they are unmatched.

7.10.5 The system must not display the unmatched users in their respective recommended profiles in the future.

7.1 Student Self-service

Student can make changes to his/her courses that are about to be taken for a semester in the future. All system (browser) interfaces are based ISO accepted industry standards for the WWW. Among others the online registration system will have the following functionalities:

7.1.1 Personal Profile

- Student Address
- Student Authentication/Change PIN
- Email/Fax Address
- Stops

7.1.2 Registration

- Registration Status
- Course Status
- Student’s Current Schedule
- Register for a course
- Add or drop a course
- Course Evaluation Guide
- Registration Schedule

7.1.3 Grades

- View past grades earned from each course taken up to the last completed semester.
- View and print non-official records of grades
- Keep a cumulative count of credits finished
- Display a computed value of current GPA

7.1.4 Registration Assistance

Stop a registration request course for error conditions:

- Courses have scheduling conflict
- Course does not exist
- Course requires a prerequisite that is not met
- Course has already been registered and or completed

8 Input Requirements

8.1 Student Email ID

Every student is given a unique email ID when they enroll into a university. The user must use this email ID while signing up and logging into the application. This is because the intended audience of this application are university students. So only university students should be able to use it and the domain of the email ID will be used to ensure that. There is also a verification link sent to that email ID to ensure that the user is using his/her own university email ID. Only after this verification process will the users finish the registration process and proceed to use the application.

8.2 Profile Information

When the user is creating a new account, he will be asked to provide more information about himself. Basic information like name, age, course of study etc. must be given. There will be more questions pertaining to personality, values/attitude, religious and socioeconomic background and living habits which are optional. However, providing the answer will enable the system to provide a better recommendation

8.3 Personal Preference

The user will be able to provide more information about his preferences or traits and qualities that he wishes his ideal roommate to have. The application will utilize this information to show the user the best possible roommates.

9 Process Requirements

The following are among the inherent requirements that our application RoomieLah must be able to handle.

9.1 Database transaction

A database transaction is a unit of work done against a database system and it follows the ACID properties:

- Atomicity - Each transaction is either complete in entirety or have no effect whatsoever
- Consistency - Any transaction will bring the database from one valid state to another

- Isolation - Executing transactions concurrently has the same results as executing them serially
- Durability - Once a transaction has been committed, it will remain so

The system must be able to send, receive and trigger database transactions to the Firebase Database system.

9.2 Data integrity

Data stored in database systems can lose its integrity due to several reasons: Human error, Transfer error etc. The following measures will be taken to minimize the risk of integrity loss:

1. Validation layer on the application server side to handle human errors in database transactions
2. Audit trails and logs to trace other errors and track modification of data in the database

Additionally, in our system, transactions that are completed must be committed to the database. Unfinished or timed-out transactions must be handled in the following way:

1. Rollback mechanism on the client side to restore the system state to the initial state in case of unfinished or timed-out transactions.
2. Dead letter queues to store unfinished or timed out transactions for future retry or error tracing

9.3 Data validation

Data error from the user's end and from the back-end database-processing end must be gracefully handled. There will be data validation and error-handling layers on the backend application server to ensure correctness and integrity of database transactions. This takes utmost importance when the user creates a profile or edits personal details.

9.4 Performance

The system must be able to handle concurrent use on a 24x7 basis and consequently resolve any locking issues. In cases where requests may take some considerable time to display the results, a user-friendly message or progress bar shall be displayed to the user. In case of failed write transactions, the user should be informed that their request has been currently queued due to system unavailability. The backend application server should at least maintain a 99% uptime. To ensure the system is scalable and available, the backend server should maintain memory cache and back-up read replica shards in the database.

9.5 Data repository

The system shall store all the relevant information related to users matched/unmatched, selections and other personalized data in an existing Firestore instance (main repository) in the database. This will be extremely beneficial in fine tuning or improving the matching algorithm (roommate recommender system) for a much more personalized user experience.

10 Output Requirements

10.1 Transaction summary and confirmation

Each database transaction or sequence of actions taken by the user must be summarized and confirmed where needed. For example, when the user wants to un-match with a person, a confirmation pop up must be displayed before carrying out the action/transaction. The user will also be allowed to undo actions wherever possible

10.2 Profile Matching

The main screen of the application (once the user logs in) displays a suggested user's profile. The user can choose to match or unmatch with the suggested user. A new suggested user's profile will be displayed if the user matches or unmatched with the current suggested user profile.

10.3 Chat Messages

From the Chat Page a user can send text messages to other users they have matched with by clicking on the matched user’s profile and send text messages using the inbuilt keyboard of their mobile device.

10.4 User Profile

From the Main Page a user can view their profile by clicking ‘Profile’ where they can view or edit their details

10.5 User Preferences

From the Main Page a user can view their preferences by clicking ‘Preferences’ where they can view or edit their preferences by editing their responses to a series of questions. Once the user clicks ‘Confirm Changes’ they will be taken to the main screen where they can see new suggested profiles based on their new preferences.

11 Hardware Requirements

11.1 Network

A stable and high-speed wireless connection is required for stable functioning of the application. The recommended speed is to ensure smooth functioning of the application is 10 Mbps. The bandwidth used by the application will be optimized for the best experience based on the participant’s network. It will automatically adjust for 3D, 4G or Wi-Fi environments.

11.2 Client Computers

The mobile hardware capability is as follows – iOS devices (iPhone SE, iPhone 6S or newer) and all ARM based Android devices.

Processor and RAM requirements:

	Minimum	Recommended
Processor	Single-core 1GHZ or higher	Dual-core 2GHZ or higher
RAM	N/A	2 GB

11.3 Production support systems

RoomieLah uses Google's FIrebase (Backend Cloud Infrastructure) and Firestore (Cloud System Database) as the primary production support system. All cloud systems comprising hardware, virtualization and storage components are fully managed and maintained by Google. No infrastructure management is required. The data centers hosting our application's data consist of backup tapes, redundant drives and are spread across multiple regions for low latency delivery.

12 Software Requirements

12.1 Client Operating Systems

- Android - Jelly Bean, API level 13, 4.1.x or newer

  Android is a mobile operating system on a modified version of LINUX kernel and other open-source software, designed primarily for touchscreen mobile devices such as smartphones or tablets.

- Apple - iOS 13.4 or newer

iOS (formerly iPhone OS) is a mobile operating system created and developed by Apple Inc. exclusively for its hardware. It is the operating system that powers many of the company's mobile devices, including the iPhone and iPod Touch.

12.2 Client Application

The application has been built using Flutter, open-source UI development, which uses Dart as the programming language. It is a product of Google used to develop cross-platform applications from a single code base.

It compiles automatically into self-contained, native-executable code due to which it does not require any support from the client apart from the installation of the application APK (for Android) and application IPA (for iOS).

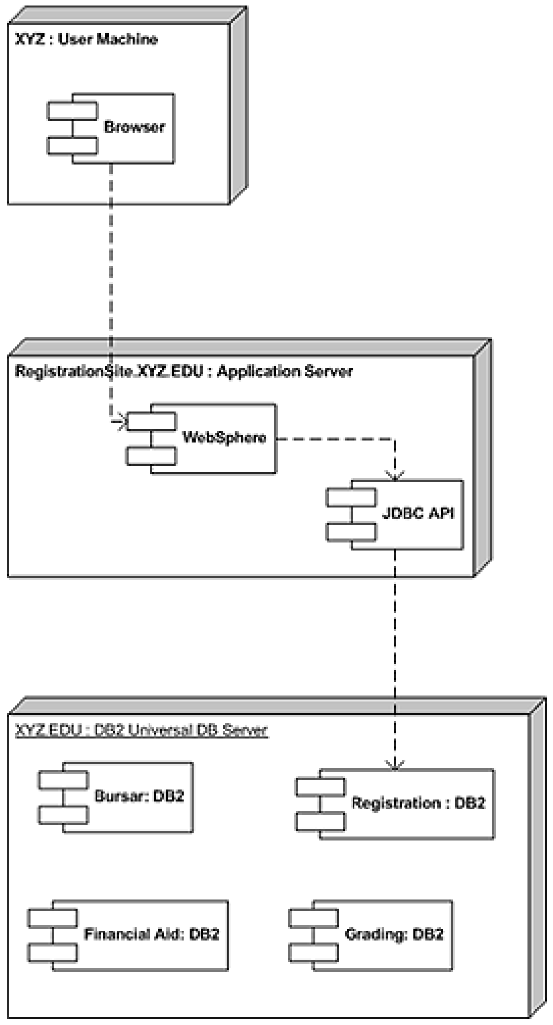
12.3 Licenses

Valid licenses are required to run software from third party vendors:

Application Development Tools	Android Studio	Apache License 2.0
	Visual Studio Code	MIT License Agreement
	Github	MIT License Agreement
	Figma	Organizational License
	Flutter	New BSD License
Backend and Database	Firebase Firestore	Apache Software Foundation

Application Marketplace	Google Play Store	Google Play Developer Distribution Agreement
	Apple App Store	Apple Developer Enterprise Program

13 Deployment Requirements



14. Bibliography

1. 9 Typical Roommate Issues. Retrieved from https://stevebrownapts.com/2018/07/02/research-shows-25-students-experience-college-roommate-problems/
2. How to be a good roommate. Retrieved from https://www.thespruce.com/how-to-be-a-good-roommate-1216908
3. The Rise of Roommate Households, The Atlantic. Retrieved from https://www.theatlantic.com/family/archive/2018/08/the-strange-unique-intimacy-of-the-roommate-relationship/567296/
4. Finding The Perfect Room and Roommates. Retrieved from https://bootcamp.uxdesign.cc/finding-the-perfect-rooms-and-roommates-a-ui-ux-case-study-63ca14666821
5. The System Design Primer. Retrieved from https://github.com/donnemartin/system-design-primer/blob/master/README.md