ROOMIELAH

DESIGN REPORT ON SOFTWARE MAINTAINABILITY

Version 1.2
03/29/2022

By: Arora Srishti, Pratyush Kumar Pandey, Rajgopal Iyer, Sanath Surawar, Atul Acharya, Aks Tayal, Gopal Agarwal

Document Change Record

Revision	Description of Change	Approved by	Date
1.0	Initial Design Report	Srishti Arora	03/21/2022
1.1	Added Software Configuration Management Tools and Software Architecture	Srishti Arora	03/25/2022
1.2	Final review, proofreading and formatting	Atul Acharya	02/29/2022

Contents

1. Design Strategies

1.1. The Planning Phase Before Development

Initiating the planning phase before the development, RoomieLah has been designed to be made extensible and scalable as the user base grows. Any improvements required in the future are anticipated and analyzed earlier in the life cycle, to ensure a flexible design of the application. As the application footfall increases, it will be scaled up to handle higher user requests. Hence, we targeted scalability as one of the factors that we would have to investigate in the future.

RoomieLah is an application targeting university students. For our current release, we target primarily the university students living in campus housing in Singapore. Once the application becomes popular, we might aim to extend our application to more users from other countries as well. This would cause our application to be able to handle more types of inputs based on residence countries and also handle country-wise matching criteria, thus making the app extensible, which is why we have added extensibility as an important development criteria.

We would also be adding new features such as machine learning models to make more accurate recommendations for users. Hence we also considered Flexibility as one of the criteria while designing the system architecture. Flexibility and Scalability promote each other.

Considering the above constraints, we decided to develop RoomieLah based on the Microservice architecture. This system architecture allows the application to be broken into smaller independent components. This makes the application more flexible as any change to a component will not break the whole application, and also makes it scalable, since it is easier to add more server instances running in this architecture as compared to a Monolithic style.

1.2 The Process of Developing

The RoomieLah application is tested using a test-driven development approach. Due to the COVID-19 pandemic, safe distancing measures and contact minimization have been enforced. This has restricted us to approach more users from our target base for testing out our application. However, to follow quality assurance, team members have taken up the role of tester. They provide continuous feedback on the design and usability of the product which are taken into consideration for improving the software application.

Moreover, during the development phase itself, the application continued to evolve, and the design was revised throughout the software development lifecycle. Using a variety of tests, reviews, and usability analysis the application has been designed to achieve high scalability, usability and maintainability. During the development phase, the various components of the application have been developed to be as generic and re-usable as possible to ensure easy extensibility, reusability and flexibility. The use of decoupled and single use application components also allows for easy maintenance of the application.

1.3 Correction by Nature

We will correct our application while testing the application. This is what we will look out for:

- **Corrective Maintainability**
  This will involve fixing bugs detected through testing.
- **Preventive Maintainability**
  Features shall be implemented in an atomic manner with each feature tested independently leading to easy error detection and making the overall application more reliable.

○ **1.4 Enhancement by Nature**

We will enhance our application while testing the application. This is what we will look out for:

- **Adaptive Maintenance**
  This ensures that the app can be easily adapted to a new operational environment.
- **Perfective Maintenance**
  This ensures that after product delivery, an error can be quickly detected and corrected, reducing maintenance costs and time required.

1.5 Maintainability Practices

To uphold quality in both process and product, we have implemented the following maintainability practices over the course of our project RoomieLah:

- Readable Code
  - When RoomieLah interacts with the real world, with the feedback and response from users, new changes and modifications in the application would be inevitable
  - In such a case, readable source code facilitates better and easier understanding of the already written codebase.
  - Readable code saves future developers' time and effort and is easy to understand ("write programs for people").

- Version Control
  - Version Control, also called Version Management, or Revision Control, allows to effectively track and control changes to a collection of related entities. It enables tracking and control changes to source code.
  - For development of RoomieLah, Git has been chosen as the Version Control System, hosted on the web application GitHub.
  - All the development code has been hosted on GitHub and all the changes, branches and various versions of the application have tracked, and controlled on the platform.

- Standardize Documentation
  - All the documents prepared for documentation of RoomieLah have been developed using standardized Industry templates, guidelines, and frameworks, for instance the IEEE practices.
  - This ensures and allows better tracking and maintainability of the

product overall and does not cause confusion if there is a change in the team members.

- Continuous Integration
  - Automated build make the code easy to compile
  - Automated tests make it easy to validate changes

- Modularity
  - The software is divided into various components that work together to form a single functioning item but sometimes they can perform as a complete function if not connected with each other. This process of creating software modules is known as Modularity.
  - The principle of System Modularity has been kept in mind during the development of RoomieLah. This will help us effectively maintain it in the future as new changes have to be rolled out.

- Design for maintainability from the outset

2. Architectural Design Patterns

○ Client - Server Architectural Design Pattern

The system architecture follows a Client-Server model. The Client-server model is a distributed application structure that partitions tasks or workload between the providers of a resource or service, called server, and service requesters called clients. It has the following components:

1. **Client:** A computer capable of receiving information or using a particular service from the service providers. In our case, the front end mobile application serves as the client.
2. **Servers**: A remote computer which provides information (data) or access to particular services. In our case, the backend application, containing modules for database CRUD operations and recommendation algorithm, forms the server side.

Several clients request and receive service from a centralized server. 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 architecture diagram for the same is given below:

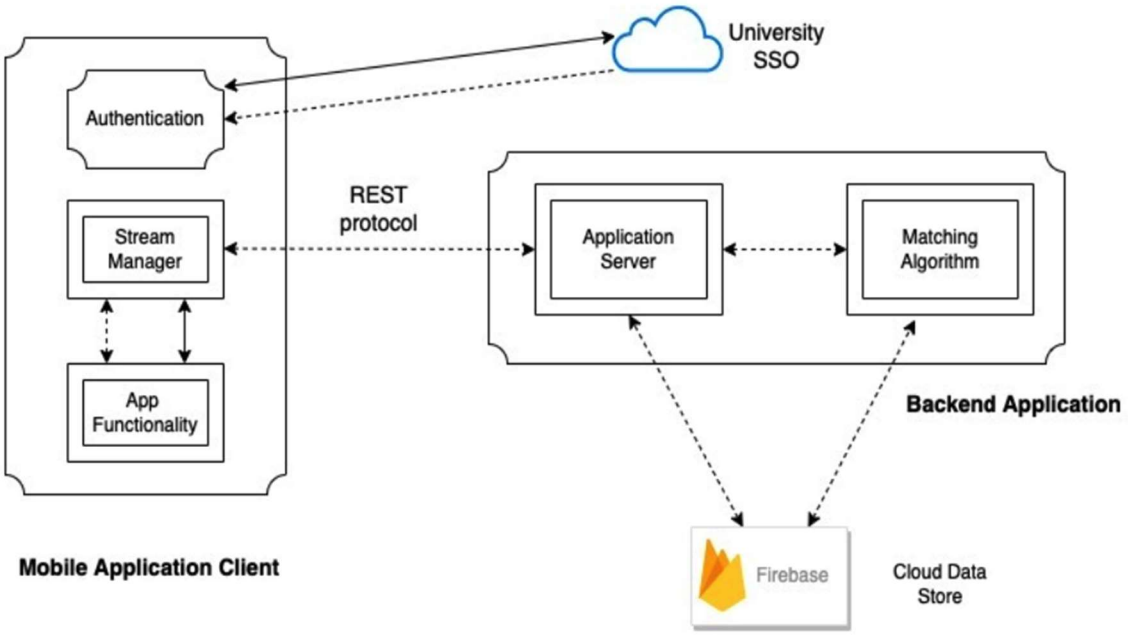

Figure 1. Architecture Diagram *for RoomieLah*

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.

The above architecture ensures that the following **Quality Concerns** are addressed:

- Modifiability
    - Changes in System Usage
    - Changes in data representation for input/output
    - Enhancement to system function
- Performance
- Reusability

    - **Observer Design Pattern (Behavioral)**
      Observer pattern has been adopted to implement RoomieLah. The UI screens for the front end application follows Model-View-Controller pattern which in turn follows the observer design pattern. The following strengths make it the optimal choice for our project:
    - It supports the principle of loose coupling between objects that interact with each other
    - Dynamic rendering of UI which improves user experience.

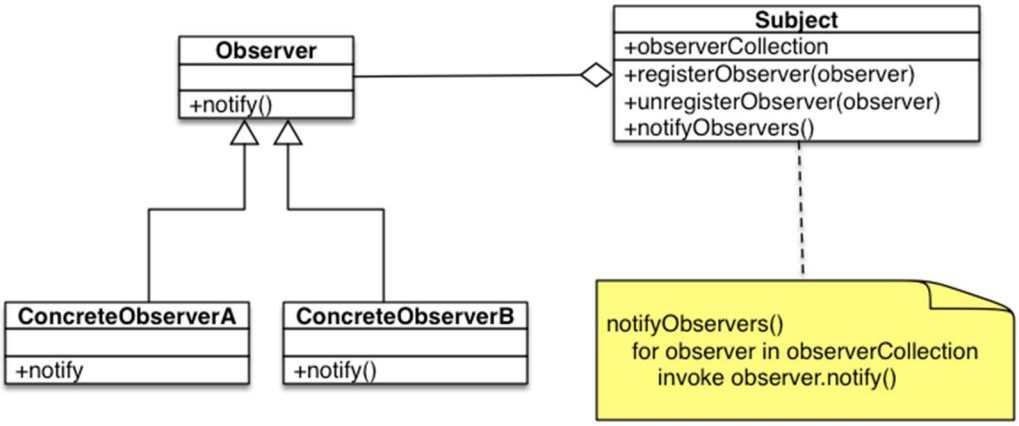

***Figure 2**. Class Diagram for Observer Design Pattern*

- **Facade Structural Design Pattern**
  The Facade design pattern compliments our object oriented design approach and enhances the quality of the code as explained below. In our front end application, the controller for user details invokes other controllers to retrieves and store user preferences, profile, matches, swipes and chat history.

- Makes code easier to use and understand
- Reduces dependencies on classes
- Decouples a client from a complex system

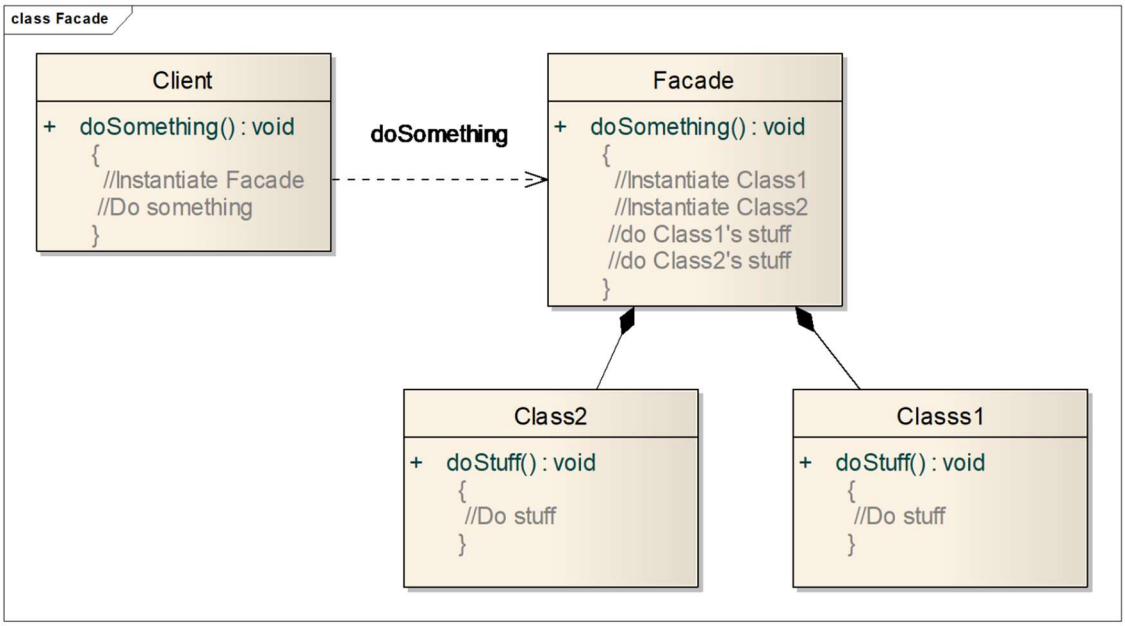

***Figure 3**. Class Diagram for Facade Design Pattern*

- **Singleton Design Pattern**

This pattern involves a single class which is responsible for creating an object while making sure that only a single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class. The front end application stores the user details in a currentUser object which belongs to the singleton class: User. This helps us to store

and retrieve the static details easily without having to make multiple read requests to the database.

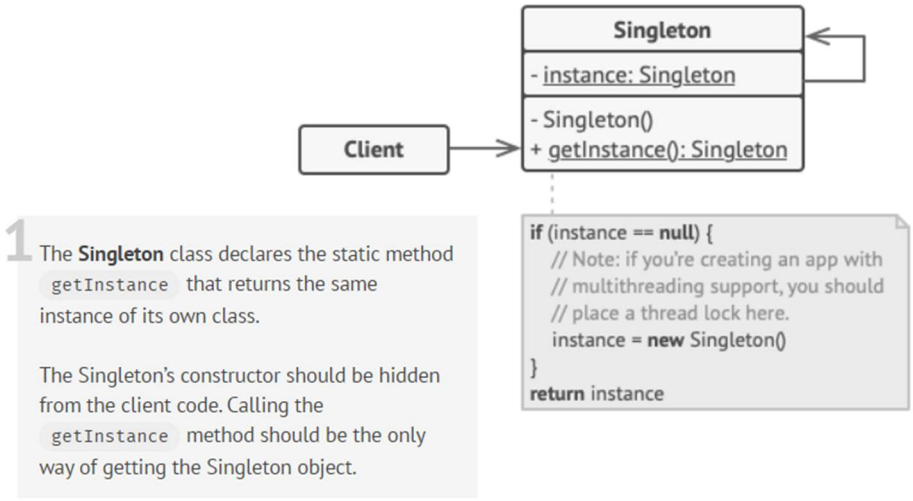

Figure 4. *Class Diagram for Structural Design Pattern*

3. **Software Configuration Management Tools**

This is where we will discuss version control management and tracking on who made what changes and when.

1. **MediaWiki**
   MediaWiki is a free and open-source application. This service is used as it is easy for beginners to pick up. There are many FAQs provided which can teach users the functions required by the users. There is a wide range of functions which allows users to create their information in different styles. It also allows users to concurrently edit the page at the same time. Hence, editing of the page will not result in a loss of information

2. **GitHub**
   GitHub is a source code hosting platform using the distributed version control and source code management Git. GitHub is chosen for its familiarity and support provided by various IDE applications. GitHub also supports issue tracking similar to a ticketing system. Whether it's a software bug, code enhancement or documentation, users can open an issue, label them appropriately and assign them for other team members to resolve. All users involved will receive timely updates on the progress of the issue.

3. **OneDrive**
   OneDrive service is used as file storage and for the backup of documents initially created. This service allows users to share and store files within the group easily. Furthermore, OneDrive allows users to edit documents using the full functionality of Microsoft Office desktop applications. This service allows users to edit documents concurrently and supports version control.

4. **SVN**
   All the documentation documents have been tracked and recorded using SVN. This allows for easy maintainability of all the documentation items. Further different versions of the documents are also tracked, and the history is maintained in the SVN for easy roll back in case of any issue.