Suite Dreams

Architecture & Design Document

Kyren Stephenson, Jonathan Fasano, Alexes Pagan, John Doyle, Arman Hemmati Alamdari, Parsa Jafaripour

# Contents

## 1. Introduction

1.1 Purpose……………………………………………………………...3

## 2. System Overview

2.1 General Structure.…………………………………………………...3

2.2 MVC Architecture….……………………………………………….4

2.3 Pipe and Filter Architecture…………………………………………5

## 3. Design Patterns

3.1 Observer Design Pattern…………..……………………………..…6

3.2 Proxy Design Pattern…………………………………………….…7

3.3 Adapter Design Pattern……………………………………………..8

## 4. Technical Details

4.1 Implementation For Login Page……………………………………9

## 5. Architectural Views

5.1 Logical View ……………………………………………………….13

5.2 Development View…………………………………………………13

5.3 Physical View………………………………………………………13

## Introduction

1.1 Purpose

The purpose of this Architecture and Design document is to explain the necessary architecture and designs involved in Suite Dreams. It intends to explain thoroughly how the application is built and how it can be reproduced. The intention of Suite Dreams is to allocate dorm rooms to college students. This document includes the visual tools to understand how Suite Dreams is to be built such as the layers involved with the project where the interface accesses the data from the database and the diagram for how the user and the program interact with each other and how the matching algorithm assigns students to rooms and roommates. This document also includes the design patterns used. We would use designs for the user interface to update the database when a user inputs data and save it to the database and adapt it to the functions in the code. This document also contains the external software and languages used for this application.

## System Overview

2.1 General Structure

The Suite Dreams application integrates multiple architectural styles to facilitate user experience and a robust backend processing. This approach ensures each component operates both independently and collaboratively, enhancing maintainability and scalability.

2.2 MVC Architecture

Our application adopts the Model-View-Controller (MVC) architecture to organize its structure effectively. In this setup, the **Model** represents our data layer, which includes the Firebase/Firestore database where we store user information, application forms, and room assignments. This layer also encompasses our data handling modules, such as data.py, which performs operations like retrieving and updating data.

The **View** consists of the user interface elements that interact directly with our users. This includes HTML templates rendered by Flask, styled with CSS and enhanced with JavaScript and Bootstrap. These components present information to the users and capture their inputs through forms, such as login screens and application submissions.

The **Controller** serves as the intermediary between the Model and the View. Implemented using Flask routes and view functions, it handles incoming requests from the View, processes user inputs, interacts with the Model to perform necessary data operations, and determines what response to send back to the user. By separating concerns in this way, our application becomes more maintainable and scalable.



2.3 Pipe and Filter Architecture

Within our matching algorithm, we utilize the Pipe and Filter architecture to process student applications efficiently through a series of sequential stages. Each “filter” performs a specific function: collecting data, validating inputs, grouping students based on roommate preferences, calculating compatibility scores, and assigning rooms. The data flows through these filters via “pipes” which act as conduits for the data stream between each processing stage.

This architecture allows us to handle large volumes of application data systematically. By breaking down the processing into discrete steps, we can easily manage and modify each stage without affecting the others. For example, the data validation filter ensures all applications are complete and correct before they proceed to the grouping stage.



This architecture supports transformations of large datasets (student applications), enabling efficient data handling and dynamic matching.

## Design Patterns

In the overall architecture of our Suite Dreams Dorm Application Program, there are many different types of existing design patterns which are crucially important in creating a responsive and adaptable system.

3.1 Observer Design Pattern

We apply the Observer design pattern to keep various parts of our application in sync with the database. When changes occur in the Firebase/Firestore database, like new applications being submitted, the Observer pattern ensures that components such as the admin and student homepages are notified immediately. This allows the user interfaces to reflect the most current data without requiring manual refreshes, enhancing the user experience by providing timely and consistent information.



3.2 Proxy Design Pattern

The Proxy design pattern is employed to manage and secure access to our Firebase/Firestore database. By introducing a proxy layer in our data access module (data.py), we control all interactions with the database. This layer handles tasks like authentication checks, ensuring only authorized components can perform certain operations. The proxy acts as an intermediary, preventing direct access to the database from various parts of the application, which adds a layer of security.



3.3 Adapter Design Pattern

Our application integrates with external services like Google Forms and Sheets to collect student application data. However, the data formats provided by these services may not match what our system expects. To resolve this mismatch, we use the Adapter design pattern. The adapter transforms the data from Google Forms and Sheets into the appropriate format required by our application components, such as the matching algorithm. This allows us to seemingly incorporate external data sources into our system without altering the existing codebase.



## Technicals

4.1 Implementation for login page

The Suite Dreams app uses Firebase for user authentication and Flask for backend processing and session management. Therefore, through Firebase, the role-based access to the backend is guaranteed when the user uses the application.

1. HTML Form for User Login:

The user interface consists of an HTML form where users can enter their email and password. Users can access the program using this form. This form is designed to securely record user login information and start the authentication process by sending them to firebase through flask.

2. Firebase Authentication:

Firebase Authentication, offered by Google Firebase, allows users to authenticate through various methods like email/password and social logins. In this app, when you type in your login email and password, Firebase's authentication service checks if it matches with what's stored in its database. Once Firebase confirms the accuracy of the information, it proceeds to verify the user's identity and continue to the next stage.

3. Token Retrieval:

After successful authentication, Firebase will generate an ID code. This token is in JSON form which is called Web Token (JWT). JWT is employed as a concise and safe way to transfer claims between two parties which includes encrypted details about the user, such as a unique identifier (UID) and email address. The token ID is securely signed by Firebase, which allows the backend to verify it and confirm that it hasn't changed.

4. Sending the ID Token to the Backend:

The token ID is sent from the frontend to the Flask backend using a POST request. Enabling this step is necessary for the backend to authenticate the user without the direct handling of passwords.

4.1.2. Backend (Flask) Session Management and Protected Routes

1. Verifying the Firebase ID Token:

After receiving the token ID, the Flask backend authenticates the token using the Firebase Admin SDK. This SDK acts as a server-side library that facilitates the management of Firebase services. To verify, we need to confirm both the token's signature and its issuance by Firebase. By using this method, we can confirm the token's authenticity and the user's identity.

2. Setting Up Flask Sessions:

After successful authentication, Flask creates a session for the authenticated user. A session acts as a mechanism to maintain user information across different requests. In Flask, session data is stored in a signed cookie on the client side. So, once you're logged in, the server can grant you access without having to log in again every time you want to do something. This session contains the user's UID and role, which can be used to customize the user's experience and permissions in the application.

3. Middleware for Protecting Routes:

Flask uses middleware, often implemented as a decorator, to protect routes that require authentication. A decorator is a function that wraps another function to extend its behavior. The login required decorator checks if the user is logged in by validating session data. If the session is valid, the user can access the protected path. Otherwise, they will be redirected to the login page.

4. Addressing Errors:

Not every error is an external factor, the backend has to deal with scenarios where the token is either invalid or expired. This happens when the token has been changed or has simply run its course. On these occasions, the backend utilizes and forwards a suitable error message to the frontend and also guarantees that the user is restricted from some areas.

5.Session Management in Flask:

Cookies are small, a kind of data interchanged between the client and the server to store a certain amount of information about the user’s activity to which the user can be automatically logged back. These cookies are validated with the use of the Flask’s secret key and as a result, the client cannot alter the content of the information without informing the user. This measure is helpful in retaining the login status of the client and facilitates a secure approach in the management of client’s sessions.

6. Recording Events by Middleware:

Logging is a critical function of any application as it is vital in ensuring the performance of the application, troubleshooting , and more importantly security of the application by keeping track of user activities and events of the application. This software implements middleware to capture and record the attributes of each incoming request and outgoing response. Of these particulars, the request method used, a UID and email address are some hidden attributes about the user that were in the form of requests.

## Architectural Views

5.1 Logical View

The Logical view of the Suite Dreams focuses on the functionalities which are offered to users, especially those related to dorm assignments and roommate matching. This view includes user interface classes that handle user inputs during the application process, user authentication and session management, a dorm application service to process and store application form responses, a matching algorithm that matches roommates based on compatibility scores, and dorm management that oversees occupancy.

5.2 Development View

The development view organizes the system from a software development standpoint. It includes a Frontend package containing all User Interface (UI) of Suite Dreams using Bootstrap, HTML, CSS and JavaScript for an intuitive and clean User Interface. Bootstrap gives the UI its core design and sets a solid foundation for the interface, while CSS and JavaScript ensure interactivity. A Backend package with modules for Authentication, Data Management, and Business Logic, and Database Scripts for managing the database is also included for a seamless interaction between the End-User and the Backend.

5.3 Physical View

The physical view addresses the deployment of the Suite Dreams system on both physical and virtual hardware. It encompasses a Web Server that hosts the application, an Application Server that executes backend logic and handles requests, a Database Server for data storage which is Firebase Firestore, and interfaces with external services like Google Forms, Google Sheets for data collection.