

StayMate

A Centralized Hostel & Mess Booking Marketplace

Semester Project Report

Submitted By

Mustafa Fawwaz	BSSE23036
Hafiz M. Saad Irfan	BSSE23082

Presented To

Dr. Zunnurain Hussain
Sir Umair Makhdoom

Course Name

Cloud Computing

Semester & Year

Fall 2025

Department of Software Engineering
Information Technology University (ITU)

Executive Summary

The search for reliable student accommodation and meal services remains a significant hurdle for university students. **StayMate** addresses this by providing a centralized marketplace for hostel and mess bookings with built-in trust mechanisms. This report details the architectural design and implementation of StayMate using an **AWS Serverless Architecture**.

By leveraging services such as AWS Lambda, Amazon API Gateway, Amazon S3, and Amazon RDS (PostgreSQL), the platform ensures high availability and data consistency. Key functionalities implemented include a **Booking Approval Workflow** (allowing owners to accept/reject requests), a **Student Review System** for transparency, and dynamic image management for both hostels and mess services. The system utilizes custom secure authentication (bcrypt) to ensure data privacy. The result is a robust, secure, and user-centric platform that streamlines the housing search process for students and listing management for property owners.

Contents

Executive Summary	i
1 Introduction	1
1.1 Background	1
1.2 Motivation	1
1.3 Domain Relevance	2
2 Problem Statement	3
2.1 Definition	3
2.2 Justification	3
3 Aim and Objectives	5
3.1 Aim	5
3.2 Objectives	5
4 Proposed Solution	7
4.1 Functional Overview	7
4.2 Why Serverless?	7
5 System Architecture	9
5.1 Architectural Design	9
5.2 Ideal vs. Implemented Architecture	9
6 AWS Services Used	11
6.1 Amazon S3	11
6.2 Amazon API Gateway	11
6.3 AWS Lambda	11
6.4 Amazon RDS (PostgreSQL)	11
7 Implementation Details	12
7.1 MVP Architecture (Implemented)	12
7.2 AWS Service Configuration (Step-by-Step)	12

7.2.1	S3 Bucket Setup	13
7.2.2	RDS Database Configuration	13
7.2.3	Lambda Function Deployment	14
7.2.4	API Gateway Integration	14
7.2.5	Monitoring Logging	15
7.3	Frontend Implementation	15
7.4	Backend and API Flow	16
7.5	Database Schema	16
7.6	Image Upload via Pre-signed URLs	16
8	Security	17
8.1	Data Security	17
8.2	Network Security	17
8.3	Access Control	17
9	Results and Output	18
9.1	Functional Results	18
9.2	Live Deployment	18
9.3	User Interfaces	18
9.3.1	Student Dashboard Search	18
9.3.2	Hostel Owner Dashboard	19
9.3.3	Mess Service Dashboard	20
10	Conclusion and Future Work	22
10.1	Summary	22
10.2	Future Enhancements	22
References		23

List of Figures

1.1	Official Logo of StayMate	1
5.1	StayMate Serverless Architecture	9
7.1	Implemented MVP Architecture (Learner Lab Environment)	12
7.2	Creation of S3 Buckets for Frontend and Media	13
7.3	PostgreSQL Database Instance in RDS	13
7.4	Node.js Backend Function in AWS Lambda Console	14
7.5	API Gateway Resources and Methods	14
7.6	CloudWatch Logs showing successful API requests	15
9.1	Student Landing Page	19
9.2	Accommodation Detail View with Reviews	19
9.3	Hostel Owner Dashboard - Listings Management	20
9.4	Booking Approval Workflow for Owners	20
9.5	Mess Owner Dashboard	21
9.6	Mess Owner Dashboard showing Active Subscribers	21

List of Tables

2.1 Stakeholders and Their Needs	3
2.2 Existing System vs Proposed System	4
3.1 Objectives Mapping with AWS Services	6
4.1 Functional Requirements	8

Chapter 1

Introduction

1.1 Background

In the modern educational landscape, student mobility is at an all-time high. A significant portion of the student population migrates to different cities for higher education, necessitating reliable hostel and mess (meal) facilities. Traditionally, this process is managed through word-of-mouth or fragmented offline directories, leading to a lack of trust and verification.



Figure 1.1: Official Logo of StayMate

1.2 Motivation

The primary motivation for StayMate is to bridge the gap between service providers and students. Existing solutions often lack transparency in pricing, peer reviews, and centralized booking management. By digitizing this ecosystem, StayMate provides a trustworthy environment where students can view verified images, read peer reviews, and request bookings seamlessly.

1.3 Domain Relevance

This project sits at the intersection of **Cloud Computing** and **Full-stack Web Development**. It demonstrates the practical application of serverless computing paradigms to solve real-world logistical problems, utilizing relational databases in a cloud environment.

Chapter 2

Problem Statement

2.1 Definition

University students face extreme difficulty in finding verified hostels and mess services that fit their budget. Conversely, small-scale property owners struggle to reach their target audience without expensive marketing and lack tools to manage booking requests efficiently.

2.2 Justification

Manual booking systems are prone to errors and do not offer a centralized way to compare multiple facilities. There is a clear need for a cloud-native platform that offers searchability, transparency via reviews, and an automated approval workflow.

Table 2.1: Stakeholders and Their Needs

Stakeholder	Problems Faced	System Solution
Students	Unverified hostels, lack of reviews	Centralized listings with Peer Reviews
Property Owners	Manual tracking of requests	Dashboard for Booking Approvals
University Admin	Lack of transparency	Structured data and monitoring

Table 2.2: Existing System vs Proposed System

Aspect	Existing System	StayMate System
Booking Method	Manual / WhatsApp	Online Request & Approval
Verification	None	Verified Listings & Reviews
Scalability	Low	High (AWS Serverless)
Availability	Limited	24/7 Access

Chapter 3

Aim and Objectives

3.1 Aim

To design and implement a scalable and cost-effective centralized marketplace for hostel and mess facility management using AWS Serverless Architecture.

3.2 Objectives

- Develop a responsive frontend using **React.js** hosted on Amazon S3.
- Implement a **RESTful API** using Amazon API Gateway to handle client-server communication.
- Utilize **AWS Lambda** for serverless compute logic (Node.js/Express).
- Implement persistent data storage using **Amazon RDS (PostgreSQL)** with tables for Users, Hostels, Messes, and Reviews.
- Implement an **Approval Workflow** where owners can Accept or Reject booking requests.
- Enable **Review and Rating** functionality for students to provide feedback.
- Enable secure image uploading for both Hostels and Messes using **S3 Pre-signed URLs**.

Table 3.1: Objectives Mapping with AWS Services

Objective	AWS Service Used
Static frontend hosting	Amazon S3
Backend business logic	AWS Lambda
RESTful API communication	Amazon API Gateway
Database	Amazon RDS (PostgreSQL)
Monitoring and logging	Amazon CloudWatch

Chapter 4

Proposed Solution

4.1 Functional Overview

StayMate allows students to filter properties based on specific criteria (price, location). It provides specific dashboards for three user roles: Students, Hostel Owners, and Mess Owners. The solution uses a decoupled architecture where the frontend interacts with the backend strictly through REST API calls.

New functionality includes a Modal-based Product View allowing students to see detailed room types and reviews before sending a booking request. Owners receive these requests on their dashboard to approve or reject.

4.2 Why Serverless?

Serverless architecture was chosen to eliminate the overhead of managing virtual servers. Benefits include:

- **No Server Management:** No need to patch or scale servers manually.
- **Cost Optimization:** Costs are only incurred when code is executed (Lambda) or stored (S3/RDS).
- **Modularity:** Separation of concerns between frontend, backend logic, and database storage.

Table 4.1: Functional Requirements

Feature	Description
Search	Filter by city, price range, and type
Listing Details	Detailed Modals with Room Types and Images
Booking Workflow	Request-based system (Pending → Confirmed/Cancelled)
Reviews	5-Star rating system with comments for Hostels/Messes
Dashboard	Owner dashboard for managing listings and requests
Image upload	Upload/Remove images using pre-signed S3 URLs

Chapter 5

System Architecture

5.1 Architectural Design

The architecture follows a **Serverless 3-Tier Web Application** pattern.

1. **Presentation Layer:** A React application hosted on S3 serves as the client.
2. **Application Layer:** Amazon API Gateway routes requests to AWS Lambda functions running an Express.js app.
3. **Data Layer:** An Amazon RDS (PostgreSQL) instance stores relational data for users, hostels, rooms, bookings, and reviews.

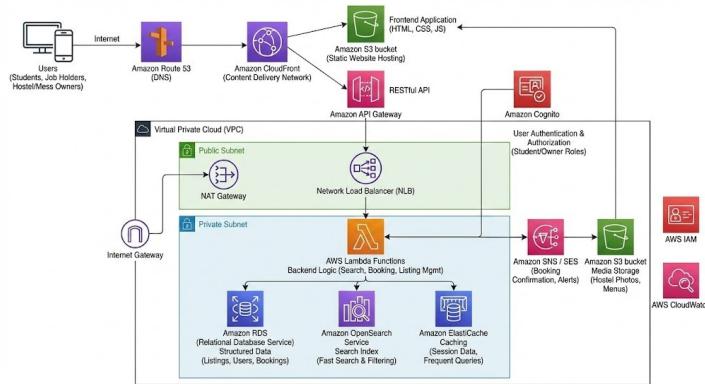


Figure 5.1: StayMate Serverless Architecture

5.2 Ideal vs. Implemented Architecture

While the ideal production architecture would include services like Amazon Cognito for identity management, Amazon CloudFront for content delivery, and VPC NAT Gateways for secure networking, the implementation for this semester project was adapted to fit within the constraints of the *AWS Academy Learner Lab*.

The system utilizes a monolithic Lambda function approach to handle all API routes, connecting to an RDS instance via public access (secured with Security Groups) to avoid the complexity and cost of VPC NAT Gateways which are often restricted in Learner Labs.

Chapter 6

AWS Services Used

6.1 Amazon S3

Used for hosting the static React website (Frontend) and storing user-uploaded property images (Assets).

6.2 Amazon API Gateway

Acts as the entry point for the backend, routing HTTP requests to the Lambda function via proxy integration.

6.3 AWS Lambda

Executes the Node.js backend logic, handling routing, authentication, and database queries.

6.4 Amazon RDS (PostgreSQL)

A managed relational database service used to store structured data including Users, Hostels, Rooms, Mess Services, Reviews, and Bookings.

Chapter 7

Implementation Details

7.1 MVP Architecture (Implemented)

The actual deployed architecture focuses on a lean, cost-effective serverless model that maximizes functionality within the Learner Lab limits. The diagram below illustrates the flow of data from the React Frontend to the AWS Cloud components.

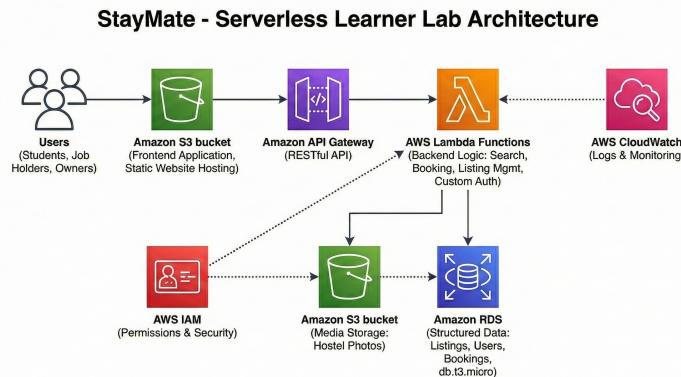


Figure 7.1: Implemented MVP Architecture (Learner Lab Environment)

7.2 AWS Service Configuration (Step-by-Step)

This section details the configuration of AWS services used in the StayMate MVP.

7.2.1 S3 Bucket Setup

The screenshot shows the AWS S3 console for the 'stay-mate-bucket'. The 'Objects' tab is selected, displaying a list of files: favicon.ico (ico), index.html (html), logo.png (png), and vite.svg (svg). The 'Block public access (bucket settings)' section has 'Off' selected. The 'Bucket policy' section shows a JSON policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::stay-mate-bucket/*"
    }
  ]
}
```

Figure 7.2: Creation of S3 Buckets for Frontend and Media

Steps Taken:

1. Created a bucket: **stay-mate-bucket**.
2. Enabled "Static Website Hosting" on the frontend bucket.
3. Configured CORS policies on the media bucket to allow uploads from the frontend.

7.2.2 RDS Database Configuration

The screenshot shows the AWS RDS console for the 'stay-mate-db' PostgreSQL instance. The 'Summary' tab is active, displaying basic information like DB identifier, CPU usage (2.98%), and engine version (17.6). The 'Configuration' tab provides detailed settings for the instance, including instance class (db.t4g.micro), vCPU (2), RAM (1 GB), and storage type (General Purpose SSD (gp2)). The 'Monitoring' tab shows monitoring type (Database Insights - Standard), performance insights (Enabled), and retention period (7 days).

Figure 7.3: PostgreSQL Database Instance in RDS

Steps Taken:

1. Launched a db.t4g.micro PostgreSQL instance.
2. Configured Security Groups to allow inbound traffic on port 5432.
3. Connected via *dbeaver* to initialize the schema tables (Users, Bookings, Reviews).

7.2.3 Lambda Function Deployment

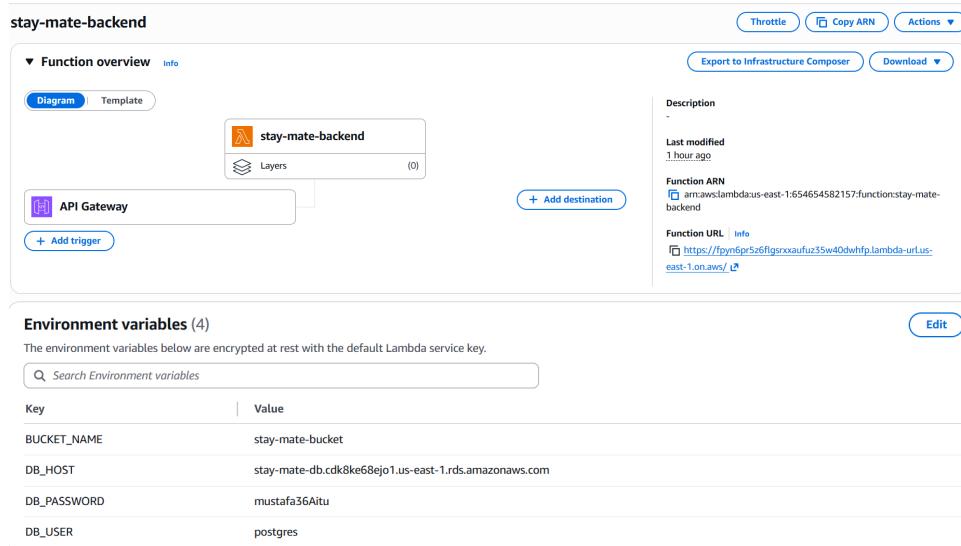


Figure 7.4: Node.js Backend Function in AWS Lambda Console

Steps Taken:

1. Created a new function **stay-mate-backend** using Node.js 18.x runtime.
2. Uploaded the backend code zip file.
3. Configured Environment Variables for DB_HOST, DB_USER, and S3_BUCKET.

7.2.4 API Gateway Integration

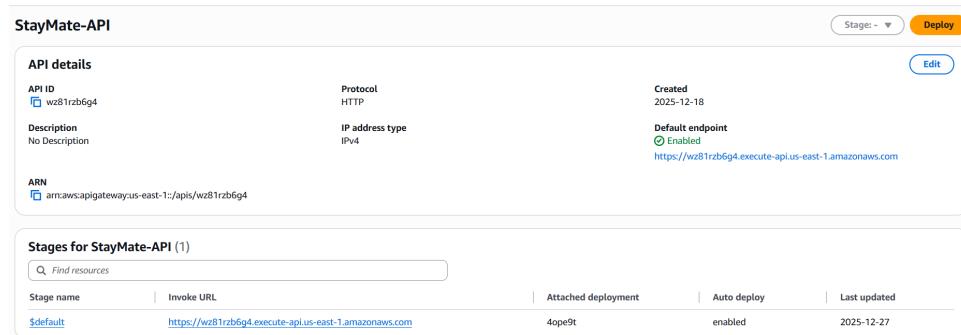


Figure 7.5: API Gateway Resources and Methods

Steps Taken:

1. Created a REST API and defined resources for `/hostels`, `/book`, and `/reviews`.
2. Enabled CORS for all resources.
3. Deployed the API to a `prod` stage.

7.2.5 Monitoring Logging

The figure consists of three vertically stacked screenshots of the AWS CloudWatch Logs interface, specifically for the log group `/aws/lambda/stay-mate-backend`. Each screenshot shows a list of log streams with their last event times.

Screenshot 1 (Top):

Log Stream	Last Event Time
2026/01/03/[SLATEST]ac9ad4a513b44c8d9da0ca28ffd98442	2026-01-03 00:50:49 (UTC)
2026/01/03/[SLATEST]8da0846515524904b47cff8e260a4c63	2026-01-03 00:49:29 (UTC)
2026/01/03/[SLATEST]f58eb7ab166d408ba642d02f728b510c	2026-01-03 00:33:49 (UTC)
2026/01/03/[SLATEST]0b2b90c867e34aac8f18031f4951afa5	2026-01-03 00:31:50 (UTC)
2026/01/02/[SLATEST]b7aaaf52339804bb893fa81257c3b7016	2026-01-03 00:10:51 (UTC)
2026/01/03/[SLATEST]c382a3c8dc78483085ce26fa25702a94	2026-01-03 00:02:00 (UTC)
2026/01/02/[SLATEST]aa3dfb33ae6d4d58b78fb1f94ab98384	2026-01-02 23:53:56 (UTC)
2026/01/02/[SLATEST]ada41444e5ee05798f7126e93e7c7308	2026-01-02 23:52:10 (UTC)
2026/01/02/[SLATEST]de12881e9d22431cbf56a0398cffddc	2026-01-02 23:49:24 (UTC)
2026/01/02/[SLATEST]d1d7672a96444566b135e97bbdd70822	2026-01-02 23:48:45 (UTC)

Screenshot 2 (Middle):

Log Stream	Last Event Time
2026/01/03/[SLATEST]ac9ad4a513b44c8d9da0ca28ffd98442	2026-01-03 00:50:49 (UTC)
2026/01/03/[SLATEST]8da0846515524904b47cff8e260a4c63	2026-01-03 00:49:29 (UTC)
2026/01/03/[SLATEST]f58eb7ab166d408ba642d02f728b510c	2026-01-03 00:33:49 (UTC)
2026/01/03/[SLATEST]0b2b90c867e34aac8f18031f4951afa5	2026-01-03 00:31:50 (UTC)
2026/01/02/[SLATEST]b7aaaf52339804bb893fa81257c3b7016	2026-01-03 00:10:51 (UTC)
2026/01/03/[SLATEST]c382a3c8dc78483085ce26fa25702a94	2026-01-03 00:02:00 (UTC)
2026/01/02/[SLATEST]aa3dfb33ae6d4d58b78fb1f94ab98384	2026-01-02 23:53:56 (UTC)
2026/01/02/[SLATEST]ada41444e5ee05798f7126e93e7c7308	2026-01-02 23:52:10 (UTC)
2026/01/02/[SLATEST]de12881e9d22431cbf56a0398cffddc	2026-01-02 23:49:24 (UTC)
2026/01/02/[SLATEST]d1d7672a96444566b135e97bbdd70822	2026-01-02 23:48:45 (UTC)

Screenshot 3 (Bottom):

Log Stream	Last Event Time
2026/01/03/[SLATEST]ac9ad4a513b44c8d9da0ca28ffd98442	2026-01-03 00:50:49 (UTC)
2026/01/03/[SLATEST]8da0846515524904b47cff8e260a4c63	2026-01-03 00:49:29 (UTC)
2026/01/03/[SLATEST]f58eb7ab166d408ba642d02f728b510c	2026-01-03 00:33:49 (UTC)
2026/01/03/[SLATEST]0b2b90c867e34aac8f18031f4951afa5	2026-01-03 00:31:50 (UTC)
2026/01/02/[SLATEST]b7aaaf52339804bb893fa81257c3b7016	2026-01-03 00:10:51 (UTC)
2026/01/03/[SLATEST]c382a3c8dc78483085ce26fa25702a94	2026-01-03 00:02:00 (UTC)
2026/01/02/[SLATEST]aa3dfb33ae6d4d58b78fb1f94ab98384	2026-01-02 23:53:56 (UTC)
2026/01/02/[SLATEST]ada41444e5ee05798f7126e93e7c7308	2026-01-02 23:52:10 (UTC)
2026/01/02/[SLATEST]de12881e9d22431cbf56a0398cffddc	2026-01-02 23:49:24 (UTC)
2026/01/02/[SLATEST]d1d7672a96444566b135e97bbdd70822	2026-01-02 23:48:45 (UTC)

Figure 7.6: CloudWatch Logs showing successful API requests

7.3 Frontend Implementation

The frontend is built with **React.js**. It consumes the backend API to display listings. Key UI components include:

- **Home/Search:** Hero section with filters and cards.

- **Detail Modals:** Popups displaying Room types, Prices, and Reviews side-by-side.
- **Owner Dashboards:** Tabbed interfaces for managing "Listings" vs "Pending Requests".

7.4 Backend and API Flow

API Gateway receives a request → Triggers the Lambda function (Serverless-Express) → Lambda connects to the PostgreSQL database via a connection pool → Returns JSON response to the client. Specific logic was implemented to handle the *Pending state* for bookings, preventing bed count deduction until Owner confirmation.

7.5 Database Schema

The system relies on a relational schema with tables for:

- `users`: Stores Students and Owners.
- `hostels & mess_services`: Stores property details (including `main_image_url`).
- `rooms`: Stores room types and available beds.
- `room_bookings & mess_subscriptions`: Links students to services with a `status` column.
- `reviews`: Links students to properties with ratings and comments.

7.6 Image Upload via Pre-signed URLs

To allow secure image uploads, the backend generates a short-lived **S3 Pre-signed URL**. The frontend uses this URL to upload images directly to the S3 bucket. This logic was extended to Mess Services as well, allowing mess owners to showcase food menus/facilities.

Chapter 8

Security

8.1 Data Security

User passwords are never stored in plain text. The application uses **bcrypt** to salt and hash passwords before storing them in the PostgreSQL database.

8.2 Network Security

The database connection utilizes SSL (Secure Sockets Layer) to encrypt data in transit between the Lambda function and the RDS instance.

8.3 Access Control

The application enforces Role-Based Access Control (RBAC) logic. For example, the review submission endpoint validates that the user is a registered student, and the booking approval endpoint validates that the user owns the specific property.

Chapter 9

Results and Output

9.1 Functional Results

- Successful user registration and secure login for three distinct roles.
- Dynamic searching of hostels and mess services.
- **Booking Approval Workflow:** Owners successfully receive, accept, or reject pending requests.
- **Review System:** Students can post and view real-time reviews for properties.
- Real-time image upload to S3 for both Hostels and Messes.
- Persistent booking records in PostgreSQL.

9.2 Live Deployment

The StayMate platform is currently deployed and accessible via the following public URL:

<http://stay-mate-bucket.s3-website-us-east-1.amazonaws.com/>

9.3 User Interfaces

This section demonstrates the functional user interfaces for the three primary user roles: Students, Hostel Owners, and Mess Owners.

9.3.1 Student Dashboard Search

Figure 9.1 shows the landing page where students can filter accommodations by city and price. Figure 9.2 illustrates the detailed booking modal including the new review section.

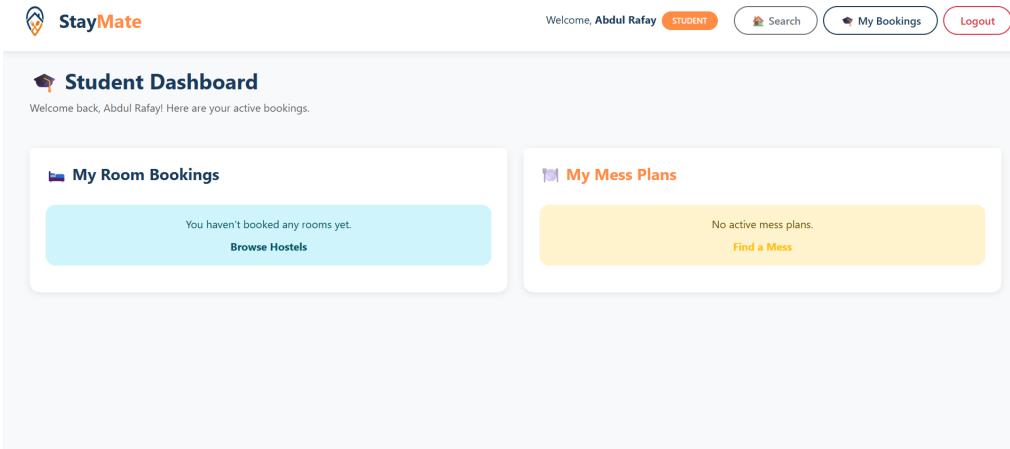


Figure 9.1: Student Landing Page

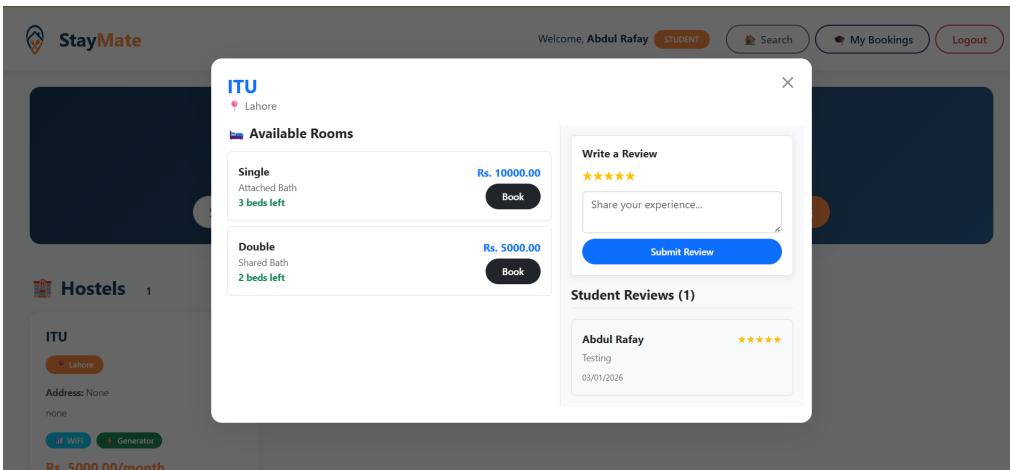


Figure 9.2: Accommodation Detail View with Reviews

9.3.2 Hostel Owner Dashboard

The Hostel Owner Dashboard (Figure 9.3) allows owners to manage their property listings. Figure 9.4 demonstrates the "Requests" tab where owners can approve or reject incoming booking applications.

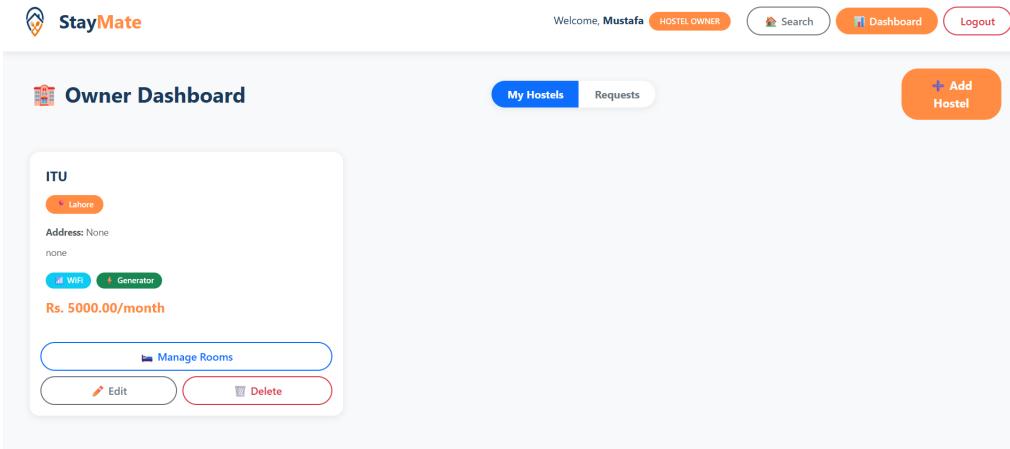


Figure 9.3: Hostel Owner Dashboard - Listings Management

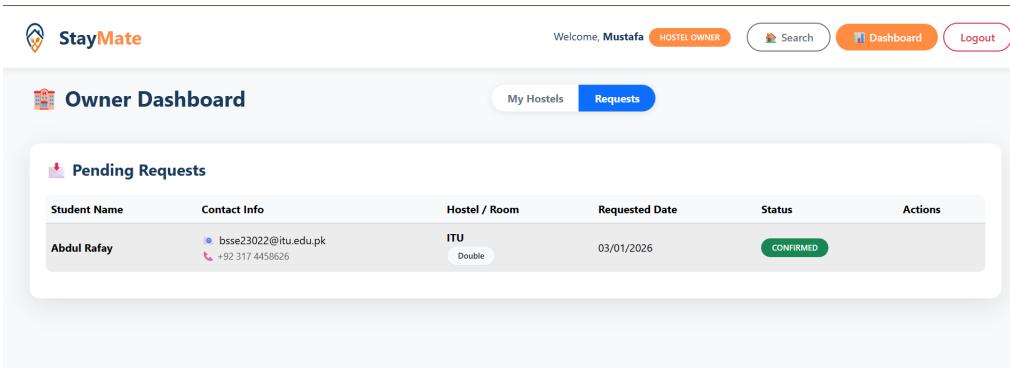


Figure 9.4: Booking Approval Workflow for Owners

9.3.3 Mess Service Dashboard

The Mess Owner Dashboard is designed with a specialized workflow distinct from hostel management. Recognizing that mess services operate on a subscription basis rather than a reservation basis, the system implements an *Instant Subscription Model*.

The interface consists of two primary tabs:

- **My Mess Services:** A CRUD interface allowing owners to manage service details such as monthly pricing, delivery radius, and menu images (Figure 9.5) .
- **Current Subscribers:** Unlike the hostel approval workflow, this tab displays a real-time list of all students with **active** subscriptions. Since student requests are auto-approved for immediate access, this interface focuses on monitoring active users and provides an administrative control to **End Subscription** (cancel access) if necessary (Figure 9.6).

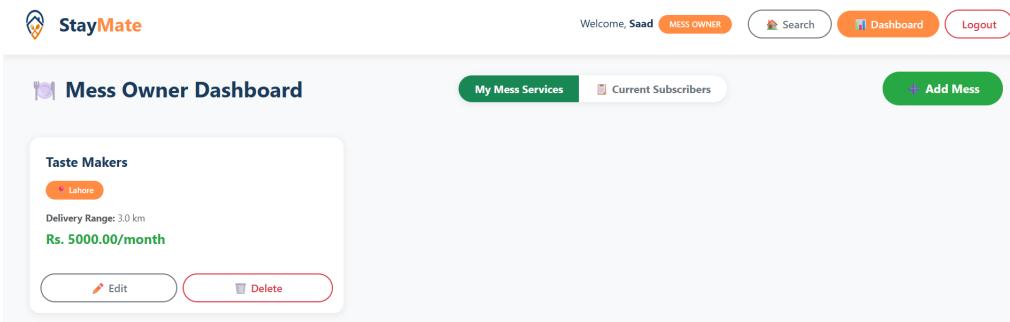


Figure 9.5: Mess Owner Dashboard

This screenshot shows the "Active Subscribers" section of the StayMate Mess Owner Dashboard. The title is "Active Subscribers". A table lists one subscriber: Abdul Rafay, with contact info "+92 317 4458626" and email "bsse23022@itu.edu.pk". The status is "ACTIVE" with an "End Sub" button next to it. Navigation tabs "My Mess Services" and "Current Subscribers" are at the top.

Student Name	Contact	Start Date	Status	Action
Abdul Rafay	+92 317 4458626 bsse23022@itu.edu.pk	03/01/2026	<input checked="" type="checkbox"/> ACTIVE	End Sub

Figure 9.6: Mess Owner Dashboard showing Active Subscribers

Chapter 10

Conclusion and Future Work

10.1 Summary

StayMate successfully demonstrates the efficacy of using AWS Lambda and RDS to build a scalable marketplace. The project meets all functional requirements, providing a seamless experience for students to find accommodation, read reviews, and book services, while empowering owners to manage their businesses digitally.

10.2 Future Enhancements

- Integration of a payment gateway (e.g., Stripe) for deposit collection.
- Implementation of an AI-based recommendation engine for hostels.
- Real-time chat functionality between Students and Owners.
- Migration to a fully VPC-isolated architecture for enhanced enterprise security.

Bibliography

- [1] Amazon Web Services, “AWS Lambda Documentation,” [Online]. Available: <https://docs.aws.amazon.com/lambda/>
- [2] Amazon Web Services, “Amazon RDS for PostgreSQL,” [Online]. Available: <https://aws.amazon.com/rds/postgresql/>
- [3] Brianc, “node-postgres (pg) Documentation,” [Online]. Available: <https://node-postgres.com/>
- [4] Meta Open Source, “React Documentation,” [Online]. Available: <https://react.dev/>