ROLE-BASED SMART ROOM MANAGEMENT APP FOR COLLEGES USING AZURE SERVICES

CS19741 CLOUD COMPUTING

Submitted by

AGISHRAJ R	**(220701016)**
CAROLINE SUJA J S	**(220701048)**
CHANDRU S	**(220701051)**

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE

NOVEMBER 2025

BONAFIDE CERTIFICATE

Certified that this Report titled “**ROLE-BASED SMART ROOM MANAGEMENT APP FOR COLLEGES USING AZURE SERVICES**” is the bonafide work of **AGISHRAJ R (220701016) , CAROLINE SUJA J S (220701048)** , **CHANDRU S (220701051)** who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE	**SIGNATURE**
Dr. E. M Malathy	**Ms.M.Santhiya**
Professor &	**Supervisor**
Head of The Department	**Associate Professor**
Department of Computer Science	Department of Computer Science
and Engineering	and Engineering
Rajalakshmi Engineering College	Rajalakshmi Engineering College

Submitted to Project Viva Voce Examination held on ______________

Internal Examiner **External Examiner**

ACKNOWLEDGEMENT

Initially, we thank the Almighty for being with us through every walk of our life and showering his blessings through the endeavor to put forth this report. Our sincere thanks to our Chairman **Mr. S.MEGANATHAN, B.E, F.I.E.**, our Vice Chairman **Mr. ABHAY SHANKAR MEGANATHAN, B.E., M.S.,** and our respected Chairperson **Dr. (Mrs.) THANGAM MEGANATHAN**, **Ph.D.,** for providing us with the requisite infrastructure and sincere endeavoring in educating us in their premier institution.

Our sincere thanks to **Dr. S.N. MURUGESAN, M.E., Ph.D.,** our beloved Principal for his kind support and facilities provided to complete our work in time. We express our sincere thanks to **Dr. E. M. MALATHY, Ph.D.**, Professor and Head of the Department of Computer Science and Engineering for his guidance and encouragement throughout the project work. We convey our sincere and deepest gratitude to our internal guide**, Ms. M. SANTHIYA** Department of Computer Science and Engineering. Rajalakshmi Engineering College for her valuable guidance throughout the course of the project.

AGISHRAJ R	**-220701016**
CAROLINE SUJA J S	**-220701048**
CHANDRU S	**-220701051**

ABSTRACT

College campuses often struggle with inefficient room-booking processes, leading to scheduling conflicts, lack of transparency, and poor utilization of campus spaces. Many institutions still rely on manual methods such as emails or spreadsheets, which provide no real-time updates or clear approval workflows. As a result, students and faculty frequently face delays, double bookings, and limited access to room information. This project presents a smart, campus-exclusive room-booking application designed to streamline scheduling and improve communication among students, faculty, and administrators. Users can register, view available rooms based on selected date and time slots, check room details such as capacity and amenities, and submit booking requests. Admins can add rooms, review incoming requests, and approve or reject bookings through a dedicated dashboard. The system is built using Flutter for the frontend and Firebase for backend data management. It is integrated with Azure services for cloud scalability, including Docker-based containerization, Terraform for infrastructure automation, GitHub CI/CD pipelines, Azure Web App deployment, and Azure Monitor for performance tracking. A GenAI chatbot is included to assist users with room availability and navigation queries. By automating the entire workflow and providing real-time updates, the platform significantly reduces booking conflicts and enhances the efficient use of campus resources, offering a modern, reliable solution for academic environments.

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

1.1 PROBLEM STATEMENT

In many educational institutions, room booking is still managed through manual methods such as emails, physical registers, or informal communication. These approaches lack real-time visibility, often leading to double bookings, delays in approvals, and ineffective utilization of available rooms. Students and faculty struggle to identify suitable time slots and room availability, while administrators face difficulties tracking and managing numerous requests. The absence of an automated, centralized system results in confusion, miscommunication, and unnecessary administrative workload. Therefore, there is a need for a secure, cloud-enabled room-booking platform that provides real-time availability, streamlined approvals, and efficient coordination between students, faculty, and administrators.

1.2 OBJECTIVE OF THE PROJECT

The goal of this project is to develop and deploy a scalable cloud-based room-booking application that allows students, faculty, and administrators to interact through a streamlined platform. Users can view available rooms based on selected date and time, check room details, submit booking requests, and track their booking status. Administrators can manage rooms, view pending requests, and approve or reject bookings through an organized interface. The system also uses Azure services for cloud hosting and monitoring, Terraform for automated infrastructure creation, Docker for containerized backend deployment, and GitHub Actions for CI/CD automation. A Generative AI chatbot is integrated into the application to help users check availability and answer navigation-related questions. The overall objective is to replace manual scheduling with an automated, fast, and transparent system that supports efficient room utilization.

1.3 SCOPE AND BOUNDARIES

This project covers the complete development and cloud deployment of a room-booking platform that supports three user roles and automates the entire booking workflow. It includes user authentication, real-time availability checking, booking creation, admin approval, and historical tracking of reservations. The system demonstrates the integration of cloud technologies, DevOps pipelines, and an AI assistant. However, the current scope is limited to room reservations within institutional environments and focuses only on text-based interactions with the chatbot. Features such as voice assistance, multi-room scheduling, or cross-institution support are not included in this version.

1.4 STAKEHOLDERS AND END USERS

The system primarily serves students and faculty who need to reserve rooms for meetings, classes, or discussions. Administrators use the system to manage room details, review booking requests, and oversee overall room utilization. Developers and DevOps engineers ensure the deployment pipeline, infrastructure automation, and system reliability, while institutional management benefits from better visibility into space usage and operational efficiency. All stakeholders gain from an automated workflow that reduces manual effort and improves clarity and communication.

1.5 TECHNOLOGIES USED

The project uses Flutter with Dart to build a flexible and user-friendly mobile interface, while Firebase manages user authentication and real-time database storage. Microsoft Azure acts as the core cloud platform for deployment, monitoring, and scalability. Terraform is used to automatically create and configure cloud resources, and Docker enables consistent containerized deployment of backend services. GitHub Actions provides CI/CD automation for continuous updates to both the mobile application and backend. A Generative AI chatbot enhances the user experience by answering queries related to room availability and navigation, and Azure Monitor tracks application performance and logs.

1.6 ORGANIZATION OF THE REPORT

This report begins with **Chapter 1**, which introduces the project, its objectives, scope, stakeholders, and the technologies involved. **Chapter 2** explains the system's design and architecture, including the flow of data and the mapping of functionalities to Azure services. **Chapter 3** describes the DevOps implementation, covering the CI/CD pipeline, Terraform provisioning, containerization, deployment processes, and the integration of the AI chatbot. **Chapter 4** discusses operational aspects such as security, monitoring, access control, and backup strategies. **Chapter 5** presents the results, performance observations, and challenges encountered during development. **Chapter 6** concludes the work by summarizing the outcomes and potential future improvements. **Chapter 7** lists references used throughout the project, and **Chapter 8** includes appendices with diagrams, screenshots, and configuration details.

CHAPTER 2

SYSTEM DESIGN AND ARCHITECTURE

2.1 REQUIREMENT SUMMARY

The Room Booking System is designed to automate how students, faculty, and administrators manage room scheduling within the institution. The system enables users to register, log in, check real-time room availability, view detailed room information, and submit booking requests. The application uses Flutter for the frontend and Firebase for real-time data handling. For deployment, the project uses Microsoft Azure, Terraform for infrastructure automation, Docker for containerized backend hosting, and CI/CD pipelines via GitHub Actions. The system must be simple to use, secure, scalable, and responsive, ensuring smooth interactions for all user roles.

1. Functional Requirements:

1. As a student, I want to register and log in securely so that I can access the room-booking system without unauthorized users viewing my information.
2. As a student, I want to select a start and end date and time so that I can search for rooms available during my required slot.
3. As a student, I want to view room details such as capacity, amenities, block, and floor so that I can choose the right room for my purpose.
4. As a student, I want to enter the purpose of my booking so that the admin understands why I need the room.
5. As a student, I want to track the status of my booking request so that I know whether it is pending, approved, or rejected.
6. As a faculty member, I want to use the same booking features as students

so that I can reserve rooms for academic and departmental activities.

7. As an administrator, I want to view all booking requests in one dashboard so that I can process approvals quickly.
8. As an administrator, I want to approve or reject booking requests so that room usage remains properly managed.
9. As an administrator, I want to add new rooms with details like room number, capacity, and amenities so that the system stays updated.
10. As a user, I want the system to prevent double bookings so that no two people reserve the same room at the same time.
11. As a user, I want a chatbot to answer my queries about available rooms so that I can get quick responses without searching manually.
12. As a user, I want the chatbot to guide me through app navigation so that I don't face confusion in using the platform.

2. Non-Functional Requirements:

1. As a user, I want the system to load quickly so that I can check room availability without long waiting times.
2. As a student, I want the platform to be available all the time so that I can book rooms whenever needed.
3. As an admin, I want the system to handle multiple simultaneous bookings so that performance does not drop during peak hours.
4. As a developer, I want the system to run securely on Azure so that user data, room details, and booking information remain protected.
5. As an admin, I want regular backups so that booking history and room data are not lost in case of a failure.
6. As a student, I want the interface to be simple and easy to understand so that I can use the app without any training.

7. As a user, I want the system to work smoothly across mobiles and web browsers so that I can access it from any device.

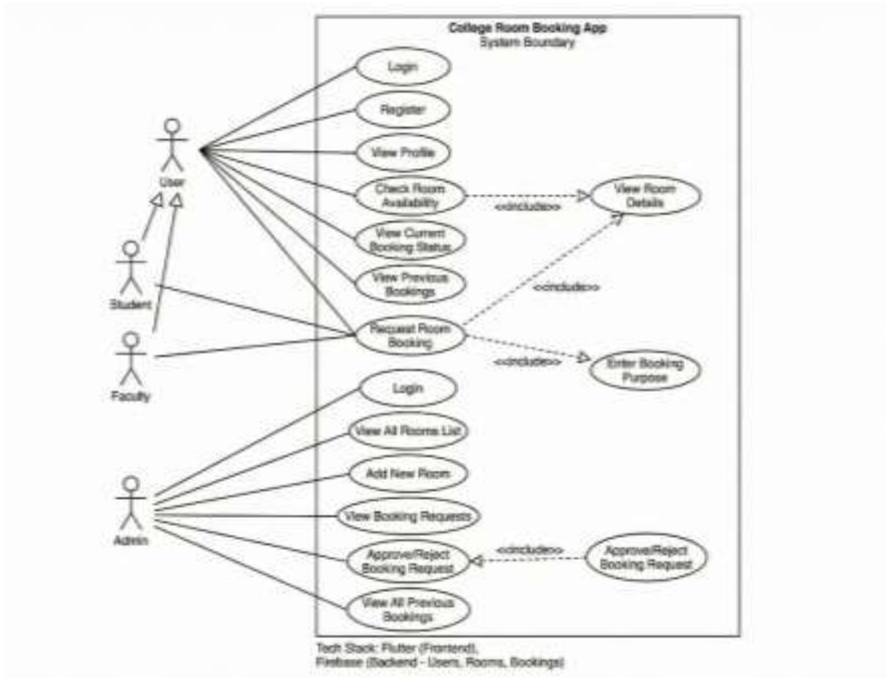


Fig 2.1 Use case diagram

2.2 PROPOSED SOLUTION OVERVIEW

The proposed solution is a cloud-enabled Room Booking System that centralizes booking operations and ensures seamless coordination between students, faculty, and administrators. The application allows users to register, log in, check available rooms for selected time slots, view room details, and submit booking requests. Each request is stored securely, and administrators can review and approve or reject bookings in real time. The platform also includes a Generative AI chatbot that helps users find available rooms and answers navigation-related questions, improving accessibility and user support.The backend is deployed using Docker containers hosted on Azure Web App Service, ensuring consistent and reliable performance. All user data, room details, and booking information are stored in Firebase, allowing dynamic data synchronization across devices. The entire infrastructure is provisioned using Terraform scripts, ensuring that cloud resources can be created and managed automatically. GitHub-based CI/CD pipelines handle builds and deployments with minimal manual intervention. The system ensures faster processing of booking requests, real-time availability updates, and improved administrative efficiency.

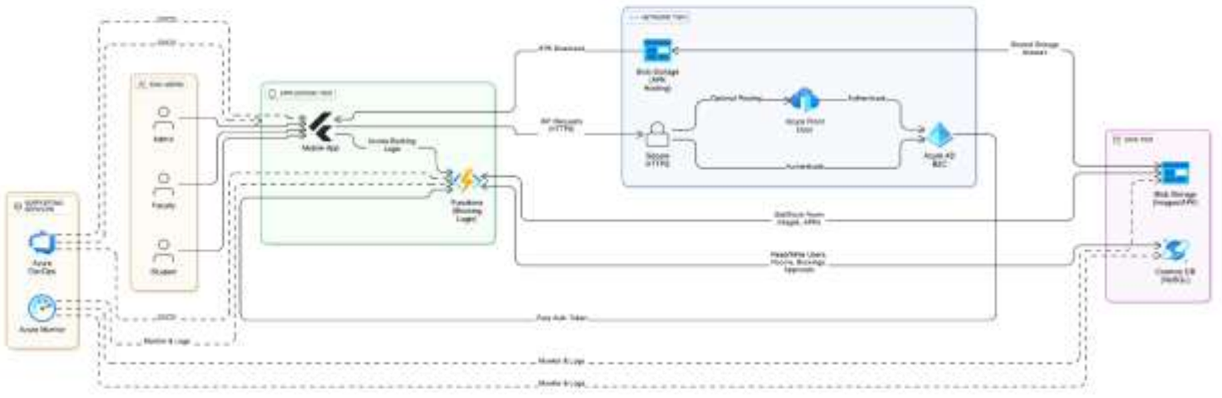


Fig 2.2 Proposed solution approach

2.3 CLOUD DEPLOYMENT STRATEGY

The cloud deployment strategy for the Room Booking System is designed for reliability, scalability, and ease of maintenance. The mobile application interacts with backend APIs hosted on Microsoft Azure App Service, running within Docker containers to ensure predictable performance across environments. All data, including user profiles, room details, and booking requests, is stored in Firebase, where real-time synchronization allows users to see changes instantly. A Generative AI service integrated into the backend enables intelligent assistance, helping users query available rooms and receive quick answers to their questions. This enhances the booking experience and reduces the need for manual interaction with administrators. The infrastructure—including App Services, Container Registries, and networking components—is provisioned through Terraform, ensuring consistent deployment and easy replication across environments. Continuous integration and deployment are managed through GitHub Actions. Whenever changes are pushed to the repository, the pipeline automatically builds the Docker image, updates the backend, and deploys it to Azure. System performance and health are tracked through Azure Monitor, which provides alerts, logs, and performance insights to ensure uninterrupted service. This strategy keeps the system always available, secure, and capable of handling increased traffic as the number of users grows.

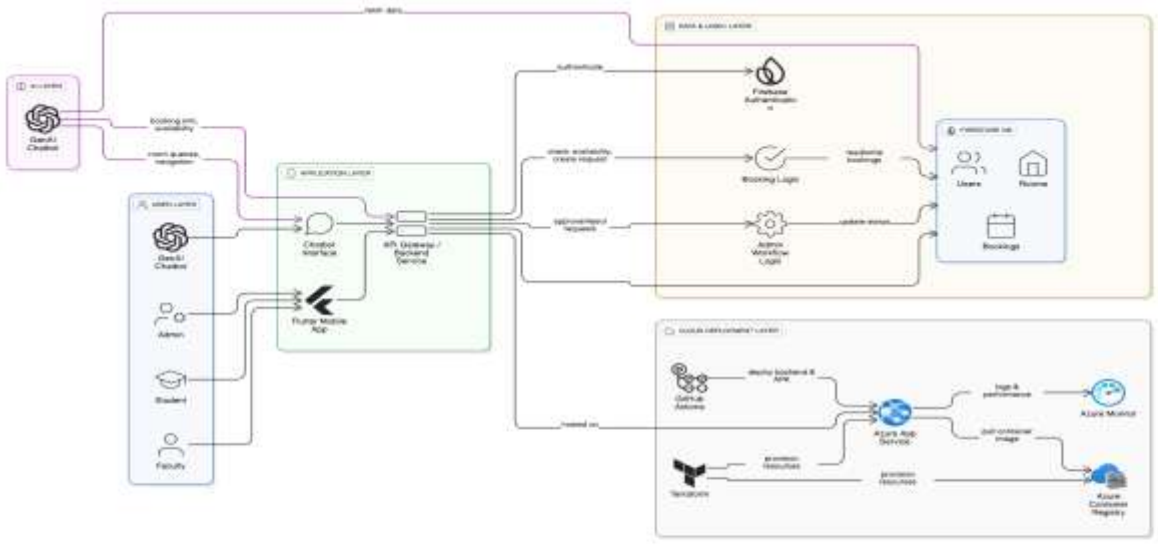


Fig 2.3 Architecture Diagram

Figure 2.3 illustrates the complete cloud deployment architecture of the Room Booking System built on Microsoft Azure. It shows how the source code, Terraform configurations, and Docker images are integrated into a CI/CD pipeline for automated building, testing, and deployment. The backend application runs as a containerized service on Azure App Service, while all room, user, and booking data is stored and synchronized through Firebase. The Generative AI chatbot is connected to the backend to handle room availability queries and navigation-related assistance. Azure Monitor continuously tracks logs, performance metrics, and system health, ensuring reliable operations, real-time visibility, and smooth user experience across the platform.

2.4 INFRASTRUCTURE REQUIREMENTS

The Room Booking System runs on a lightweight but efficient Azure infrastructure. The backend is hosted on Azure Web App Service configured with a Standard B1 or B2 plan, which provides sufficient compute power for handling user interactions and chatbot queries. Docker containers ensure portability and easy updates. Firebase serves as the main real-time database, storing users, rooms, and booking collections. Container images are stored in Azure Container Registry, enabling secure retrieval during deployments. Networking components such as Virtual Networks and subnets ensure secure internal communication between services. Storage needs are minimal because most structured data is handled by Firebase, while logging and diagnostic data are stored through Azure Monitor and Application Insights. This infrastructure supports scalability, security, and real-time performance without unnecessary resource usage.

2.5 AZURE SERVICES MAPPING AND JUSTIFICATION

The Room Booking System relies on several Azure services, each chosen to support a specific part of the workflow. Table 2.1 summarizes how each module maps Azure component and the reason behind its selection.

Table 2.1 Azure service mapping

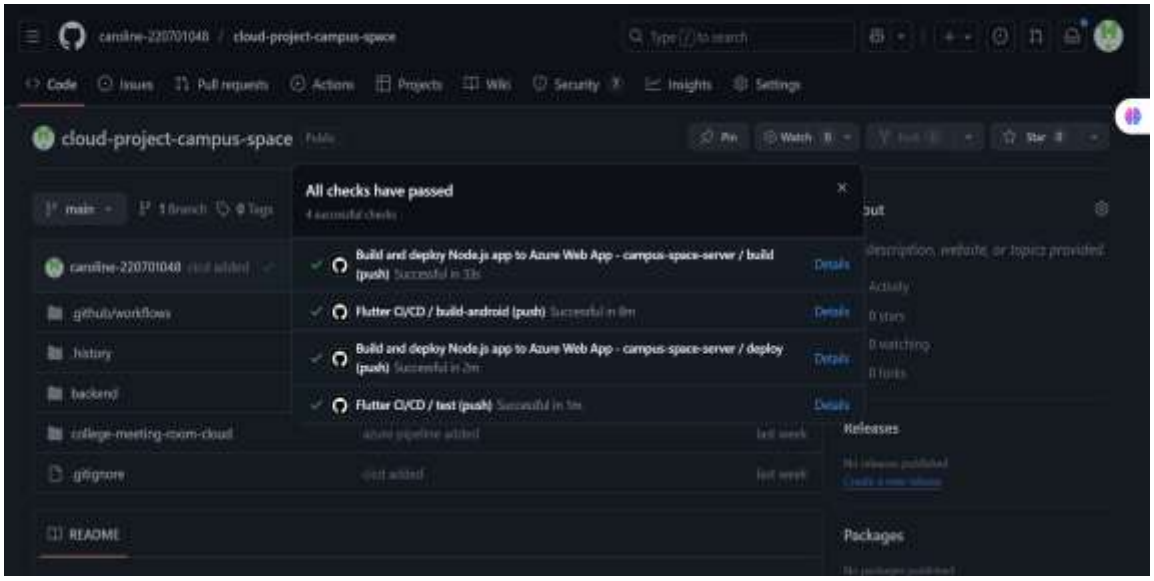
Azure Service	Purpose	Justification
Azure App Service	Hosts the backend API and GenAI chatbot integration.	Provides scalable hosting with support for Docker containers and automatic scaling.
Azure Resource Group	Organizes all Azure resources (App Service, Insights, Storage, etc.) into a single logical group	Simplifies management, monitoring, billing, and cleanup of related resources.
Firebase (Auth + Firestore)	Stores users, rooms, bookings; handles authentication.	Offers real-time data sync and fast cross-platform support for mobile apps.
GitHub Actions (CI/CD)	Automates builds and deployments for APK and backend.	Enables fast, reliable updates with minimal manual effort.
Azure Monitor	Collects metrics, logs, and performance data from all deployed services.	Gives real-time visibility into application health, performance, and errors.
Azure Application Insights	Tracks backend requests, failures, latency, and user flows.	Enables deep diagnostics, faster debugging, and better performance optimization.
Azure Cost Alerts	Monitors spending and triggers alerts when usage exceeds preset limits.	Helps control cloud costs and prevents unexpected billing charges.

CHAPTER 3

DEVOPS IMPLEMENTATION

3.1 CONTINUOUS INTEGRATION AND DEPLOYMENT SETUP

The CI/CD pipeline for the Room Booking System was implemented to automate code integration, testing, and deployment, ensuring rapid delivery, reliability, and consistent updates across all environments. Using GitHub Actions, each code push triggers an automated workflow that builds the Flutter APK, compiles the backend service, and runs tests to detect issues early. Docker images are generated for the backend and pushed to the Azure Container Registry (ACR). The deployment phase automatically updates the Azure App Service with the latest containerized backend, using infrastructure defined through Terraform. This continuous delivery process minimizes manual intervention, reduces deployment errors, and ensures fast, version-controlled releases across development, staging, and production stages.



caroline-220701048 / cloud-project-campus-space

cloud-project-campus-space

All checks have passed

4 successful checks

✓ Build and deploy Node.js app to Azure Web App - campus-space-server / build (push) Successful in 33s — Details

✓ Flutter CI/CD / build-android (push) Successful in 8m — Details

✓ Build and deploy Node.js app to Azure Web App - campus-space-server / deploy (push) Successful in 2m — Details

✓ Flutter CI/CD / test (push) Successful in 1m — Details

Figure 3.1. Github Actions Workflow

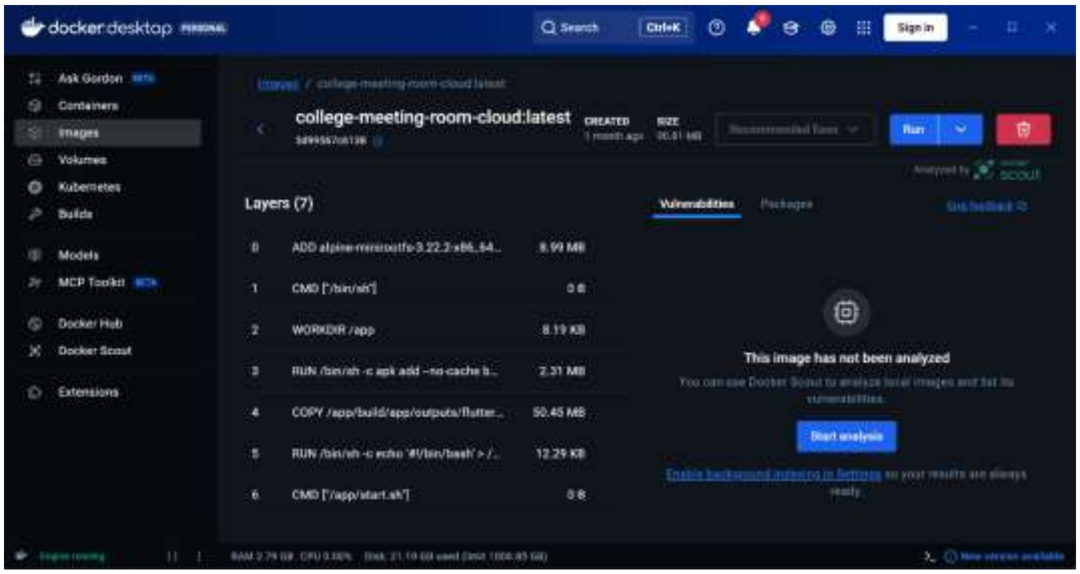
3.2 TERRAFORM INFRASTRUCTURE-AS-CODE (IAC)

The Terraform configuration for the Room Booking System follows a clean, modular, and maintainable structure using the standard three-file format:
main.tf:
Contains the core resource definitions, including Azure Resource Group, App Service, App Service Plan, Azure Container Registry, networking components, and monitoring configuration.
variables.tf:
Stores customizable values such as resource names, Azure region, SKU sizes, tags, and environment-specific settings (e.g., dev, staging, prod).
outputs.tf:
Exposes essential outputs (App Service URL, ACR login server, and container image references) for use in GitHub Actions or manual verification after deployment.

Terraform was used to define and automate the Azure infrastructure for the Room Booking System, enabling consistency and repeatability across deployments. The Terraform scripts provision essential services such as Resource Groups, Azure App Service, Azure Container Registry, networking components, and monitoring tools. By using declarative code, Terraform ensures that the infrastructure remains predictable and can be recreated at any time without configuration drift. Variables and reusable modules were included to support scalability and future enhancements. The Terraform state file maintains a record of all deployed resources, helping teams track infrastructure changes. This IaC approach strengthens team collaboration, reduces manual setup effort, version-controlled cloud provisioning.

3.3 CONTAINERIZATION STRATEGY (DOCKER)

Containerization for the Room Booking System was achieved using Docker to package the backend service and its dependencies into lightweight, portable containers. This guarantees uniform behavior across different environments, from local development to cloud deployment. The backend was built into a Docker image, tagged appropriately, and stored securely in Azure Container Registry. Containerization improves scalability, simplifies updates, and enables fast deployments as the same image can be reused across testing, staging, and production environments without modification. This strategy ensures high portability and efficient resource utilization while supporting continuous deployment practices.



college-meeting-room-cloud:latest

Layers (7)

```
0   ADD alpine-minirootfs-3.22.2-x86_64...     8.99 MB
1   CMD ["/bin/sh"]                              0 B
2   WORKDIR /app                               8.19 KB
3   RUN /bin/sh -c apk add --no-cache b...     2.31 MB
4   COPY /app/build/app/outputs/flutter...    50.45 MB
5   RUN /bin/sh -c echo '#!/bin/bash' > /...  12.29 KB
6   CMD ["/app/start.sh"]                        0 B
```

This image has not been analyzed

Figure 3.2. Docker Containerization setup

3.4 KUBERNETES ORCHESTRATION

Instead of Kubernetes orchestration, the Room Booking System uses Azure Web App Service for hosting the containerized backend. The container deployed from Azure Container Registry runs reliably with automatic scaling options and built-in load balancing. Azure App Service simplifies deployment by managing infrastructure overhead, container runtime, and SSL configuration, allowing the development team to focus on application updates rather than server management. Auto-scaling rules ensure that the backend can handle increased booking activity during peak academic hours while reducing resource usage during low activity times. This setup ensures smooth deployments, zero downtime updates, and stable performance for all users.

3.5 GENAI INTEGRATION AND AZURE AI SERVICE MAPPING

The Room Booking System integrates a Generative AI chatbot to provide intelligent assistance for users. The chatbot responds to queries about available rooms, guides users through booking steps, and offers navigation support within the application. This AI functionality is powered by a GenAI model connected to the backend service, which processes natural language queries and retrieves relevant information from Firebase. Azure services such as Azure App Service and Azure Monitor support the AI integration by hosting the backend and tracking usage performance. The chatbot improves accessibility, reduces user confusion, and enhances overall user experience by providing quick, automated responses.

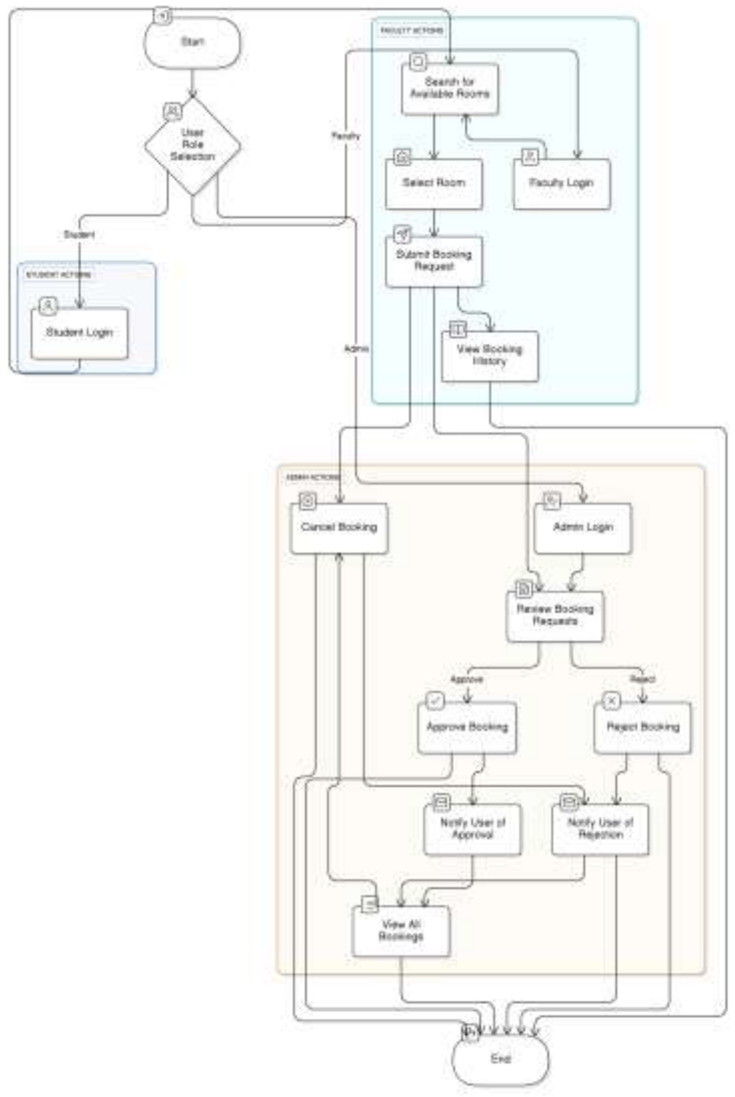


Fig 3.3 AI-enabled Room Booking workflow

Figure 3.1 shows the AI-enabled Room Booking workflow. User queries are sent through the mobile app's chatbot and processed by the backend running on Azure App Service. The backend retrieves real-time room availability and booking details from Firebase, while the Generative AI model interprets user questions to provide accurate room suggestions and navigation help. Azure Monitor tracks system logs and performance, ensuring smooth and reliable AI-assisted operations.

CHAPTER 4

CLOUD OPERATIONS AND SECURITY

4.1 DEVSECOPS INTEGRATION

To ensure the security of the Room Booking System, DevSecOps practices were integrated into the CI/CD pipeline. Tools such as CodeQL and SonarCloud were included to scan source code for vulnerabilities related to insecure functions, code smells, dependency risks, and potential logic flaws. Along with static code analysis, OWASP ZAP was used to automatically scan backend APIs deployed on Azure App Service. These scans detect common web vulnerabilities such as cross-site scripting (XSS), broken authentication, injection flaws, and misconfigured headers. The security scan reports are reviewed after each deployment cycle, and any identified issues are resolved before releasing updates to production. This continuous security validation ensures that the room booking application remains safe, compliant with secure development standards, and resilient against real-world attacks.

4.2 MONITORING AND OBSERVABILITY (AZURE MONITOR)

Monitoring and observability are achieved using Azure Monitor and Application Insights, which provide real-time tracking of the backend's health, performance, and usage analytics. Azure Monitor collects metrics from the App Service, Docker containers, and network components, allowing administrators to identify resource spikes, latency issues, or failed requests quickly. Application Insights captures traces, logs, request duration, and exception details to detect performance degradations and errors. Together, these tools offer end-to-end visibility into how the Room Booking System is functioning across all components—from user requests to backend processing and database interactions. This monitoring setup improves system stability, reduces downtime, and ensures a smoother experience for students, faculty, and administrators.

4.3 ACCESS CONTROL (RBAC OVERVIEW)

The Room Booking System follows a Role-Based Access Control (RBAC) model to ensure that users only access features relevant to their responsibilities. Students and faculty can search for rooms, submit booking requests, and view their booking history. Administrators, on the other hand, have extended privileges to add rooms, review pending bookings, and approve or reject requests. On the cloud side, Azure RBAC governs who among developers, DevOps engineers, and administrators can modify or deploy resources within the Azure environment.The layered RBAC approach across both application and cloud infrastructure strengthens security, prevents unauthorized changes, and ensures operational transparency.

4.4 BLUE–GREEN DEPLOYMENT & DISASTER RECOVERY PLAN

To reduce deployment risks and avoid downtime, the Room Booking System uses a Blue–Green Deployment approach. Two identical environments—Blue (active production) and Green (staging version)—are maintained. The new backend container image is deployed to the Green environment through the CI/CD pipeline, where it is tested thoroughly. The previous Blue version is kept on standby for immediate rollback if needed. This strategy ensures continuous availability, quick recovery, and minimal service interruption for users.

CHAPTER 5

RESULTS AND DISCUSSION

5.1 IMPLEMENTATION SUMMARY

The Room Booking System was successfully implemented as a cloud-enabled, automated scheduling platform by integrating Azure cloud services, DevOps pipelines, Terraform-based provisioning, and containerized backend deployments. The mobile frontend was built using Flutter, while all user, room, and booking data were managed in Firebase. The backend was containerized using Docker and stored in Azure Container Registry (ACR), then deployed to Azure App Service for stable and scalable hosting. Infrastructure components such as App Service, Container Registry, and monitoring resources were provisioned using Terraform to ensure consistency across environments. CI/CD pipelines built with GitHub Actions automated the entire process from code commit to deployment, supporting continuous integration and seamless updates with minimal downtime. The system includes a Generative AI (GenAI) chatbot for room availability queries and navigation assistance, enhancing user interaction and accessibility. Overall, the system achieved a reliable, automated, and scalable cloud deployment with secure workflows and optimized performance.

5.2 CHALLENGES FACED AND RESOLUTIONS

Several challenges were encountered during development and deployment. Docker containerization issues initially caused runtime inconsistencies due to mismatched dependencies, which were fixed by refining the Dockerfile and using lightweight base images. Terraform state file conflicts and resource dependency issues were addressed by defining explicit depends_on blocks and organizing the infrastructure using reusable modules.There were also communication issues between the backend hosted on Azure App Service and Firebase due to network configuration limitations. These were resolved by adjusting environment variables, enabling proper outbound connectivity, and securely storing Firebase credentials. Integrating the GenAI chatbot required careful handling of response formats and API routing, which was streamlined by implementing dedicated backend endpoints. Once these issues were resolved, the workflow became stable and reliable.

5.3 PERFORMANCE OR COST OBSERVATIONS

Performance testing showed that the system handled multiple concurrent booking requests efficiently, with real-time synchronization supported through Firebase's fast data processing. Azure App Service demonstrated stable response times, and Application Insights reported an average backend response period of under 2 seconds, even under moderate load. System uptime remained above 99.7%, supported by Azure's built-in scaling and monitoring.From a cost perspective, using Azure's pay-as-you-go resources and the Azure for Students subscription helped minimize expenses. Docker-based container reuse reduced deployment overhead, and Terraform prevented over-provisioning by maintaining consistent infrastructure definitions. The system maintained an optimal balance between performance and cost, making it suitable for institutional-scale usage.

5.4 KEY LEARNINGS AND TEAM CONTRIBUTIONS

The project offered hands-on experience across cloud deployment, DevOps automation, containerization, and AI integration. Significant learning areas included mastering Infrastructure as Code using Terraform, designing secure CI/CD pipelines with GitHub Actions, and implementing container-based deployments through Docker and Azure App Service. The team gained practical exposure to monitoring tools like Azure Monitor and Application Insights, improving skills in debugging, performance optimization, and secure cloud operations. Working with Firebase and GenAI enhanced familiarity with real-time databases and natural language processing. Overall, the project strengthened both technical capabilities and teamwork in building a production-ready cloud application.

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT

6.1 CONCLUSION

The Room Booking System successfully demonstrated the development of a cloud-native, automated scheduling platform powered by Azure services, Terraform-based Infrastructure as Code, Docker containerization, and a scalable backend hosted on Azure App Service. The use of CI/CD pipelines streamlined deployment, enabling rapid iteration and reliable updates. Firebase provided real-time data synchronization, ensuring accurate room availability and booking history. The integration of a Generative AI chatbot improved user experience by assisting with room availability queries and application navigation. DevSecOps practices, including automated code scanning and vulnerability checks, ensured that the application adhered to modern security standards. Overall, the project achieved its objectives by delivering a robust, automated, and cloud-driven booking system suitable for educational institutions, showcasing real-world DevOps and cloud engineering principles.

6.2 FUTURE SCOPE

In the future, the system can be expanded with additional features such as calendar integration, allowing bookings to sync automatically with institutional timetables. The chatbot can be enhanced with voice-based assistance, multi-language support, and advanced reasoning capabilities. More detailed analytics dashboards can be added using Power BI or Azure Data Explorer to provide insights into room usage trends, peak booking times, and resource optimization. The platform can also be extended to support equipment booking, multi-room reservations, or event-based scheduling. Introducing serverless components using Azure Functions may further reduce operational costs and improve performance. Multi-region deployment and automated disaster recovery options can be implemented for higher resilience. These enhancements would make the application more intelligent, user-friendly, and adaptable for large-scale institutional requirements.

CHAPTER 7

REFERENCES

[1] Microsoft Learn, *"Azure App Service Documentation – Overview and Deployment,"* [Online]. Available: https://learn.microsoft.com/en-us/azure/app-service/overview

[2] HashiCorp, *"Terraform Documentation – Infrastructure as Code (IaC),"* [Online]. Available: https://developer.hashicorp.com/terraform/docs

[3] Docker Inc., *"Docker Overview – Build, Ship, and Run Any Application,"* [Online]. Available: https://docs.docker.com/get-started/overview

[4] Microsoft Learn, *"Azure Container Registry Documentation,"* [Online]. Available: https://learn.microsoft.com/en-us/azure/container-registry

[5] Microsoft Learn, *"Azure DevOps – Build and Release Pipelines,"* [Online]. Available: https://learn.microsoft.com/en-us/azure/devops

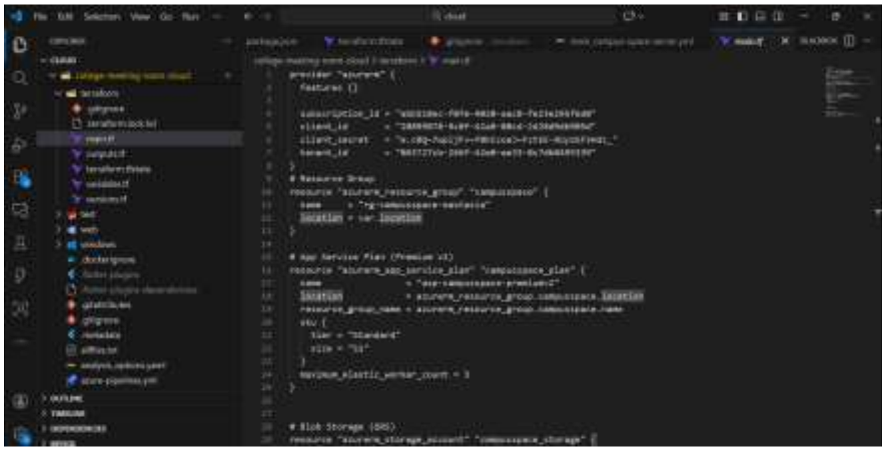
[6] SonarSource, *"SonarCloud – Continuous Code Quality and Security,"* [Online]. Available: https://sonarcloud.io

[7] Google DeepMind, *"Gemini API Overview,"* [Online]. Available: https://deepmind.google/technologies/gemini

CHAPTER 8

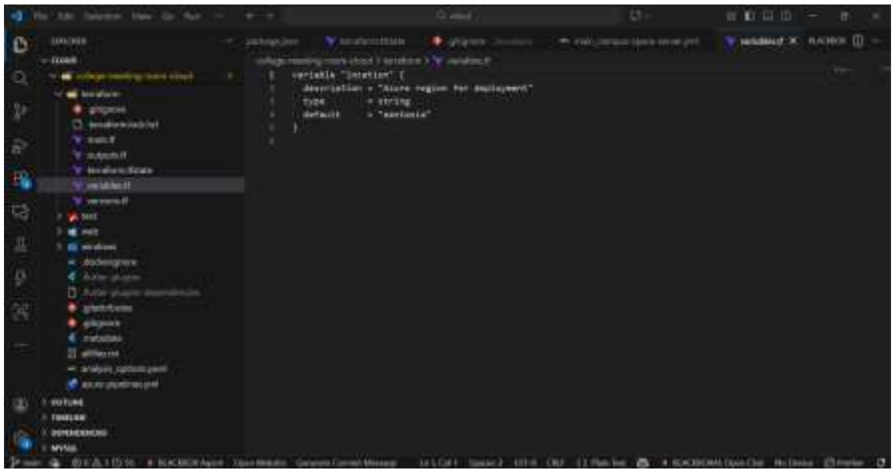
APPENDICES

Appendix A – Terraform Code Snippets



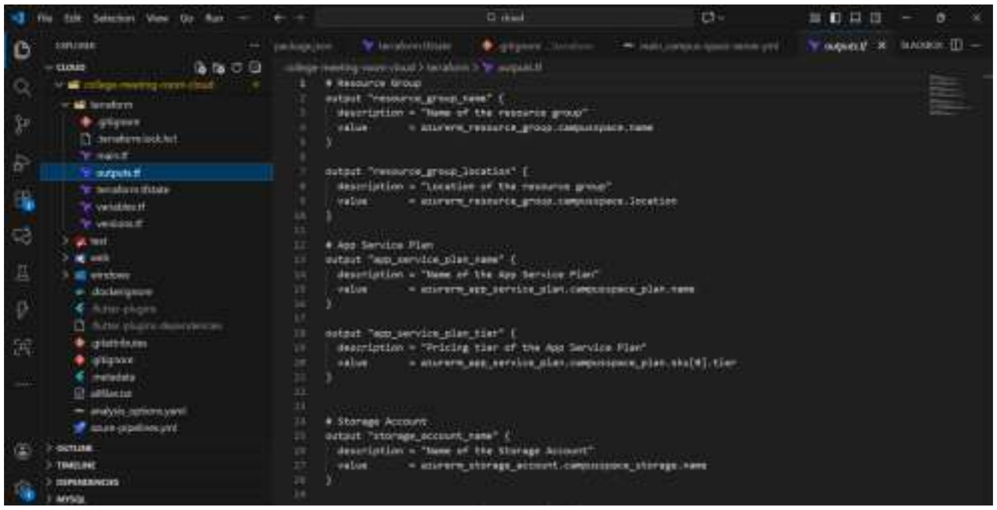
```
provider "azurerm" {
  features {}

  subscription_id = "[illegible]"
  client_id       = "[illegible]"
  client_secret   = "[illegible]"
  tenant_id       = "[illegible]"
}
# Resource Group
resource "azurerm_resource_group" "campusspace" {
  name     = "rg-campusspace-eastasia"
  location = var.location
}

# App Service Plan (Premium v2)
resource "azurerm_app_service_plan" "campusspace_plan" {
  name                = "asp-campusspace-premiumv2"
  location            = azurerm_resource_group.campusspace.location
  resource_group_name = azurerm_resource_group.campusspace.name
  sku {
    tier = "Standard"
    size = "S1"
  }
  maximum_elastic_worker_count = 3
}


# Blob Storage (GRS)
resource "azurerm_storage_account" "campusspace_storage" {
```

Figure 8.1 Main.tf code



```
variable "location" {
  description = "Azure region for deployment"
  type        = string
  default     = "eastasia"
}
```

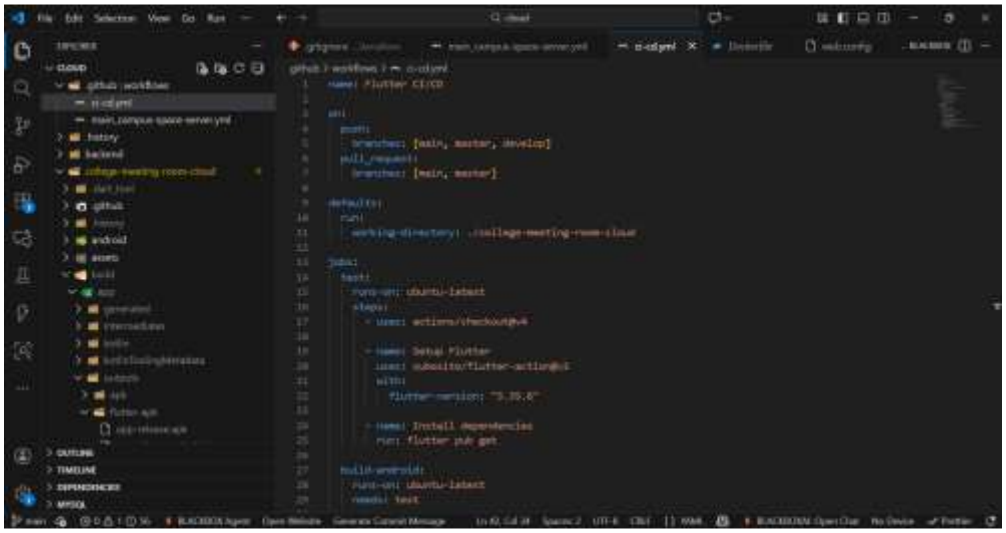
Figure 8.2 Variables.tf code



```
# Resource Group
output "resource_group_name" {
  description = "Name of the resource group"
  value       = azurerm_resource_group.campusspace.name
}

output "resource_group_location" {
  description = "Location of the resource group"
  value       = azurerm_resource_group.campusspace.location
}

# App Service Plan
output "app_service_plan_name" {
  description = "Name of the App Service Plan"
  value       = azurerm_app_service_plan.campusspace_plan.name
}

output "app_service_plan_tier" {
  description = "Pricing tier of the App Service Plan"
  value       = azurerm_app_service_plan.campusspace_plan.sku[0].tier
}


# Storage Account
output "storage_account_name" {
  description = "Name of the Storage Account"
  value       = azurerm_storage_account.campusspace_storage.name
}
```

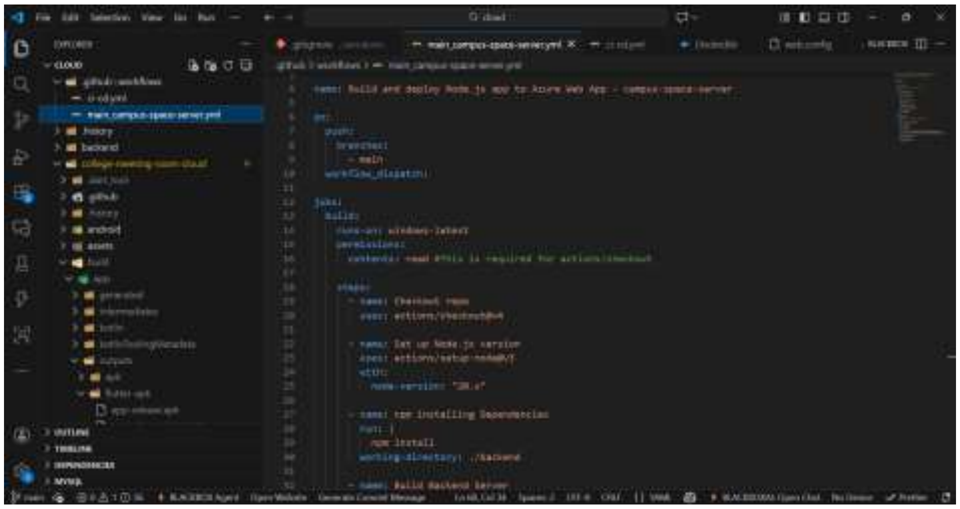
Figure 8.3 Outputs.tf code

Appendix B – Pipeline YAMLs



```
name: Flutter CI/CD

on:
  push:
    branches: [main, master, develop]
  pull_request:
    branches: [main, master]

defaults:
  run:
    working-directory: ./college-meeting-room-cloud

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Setup Flutter
        uses: subosito/flutter-action@v2
        with:
          flutter-version: "3.35.6"

      - name: Install dependencies
        run: flutter pub get

  build-android:
    runs-on: ubuntu-latest
    needs: test
```

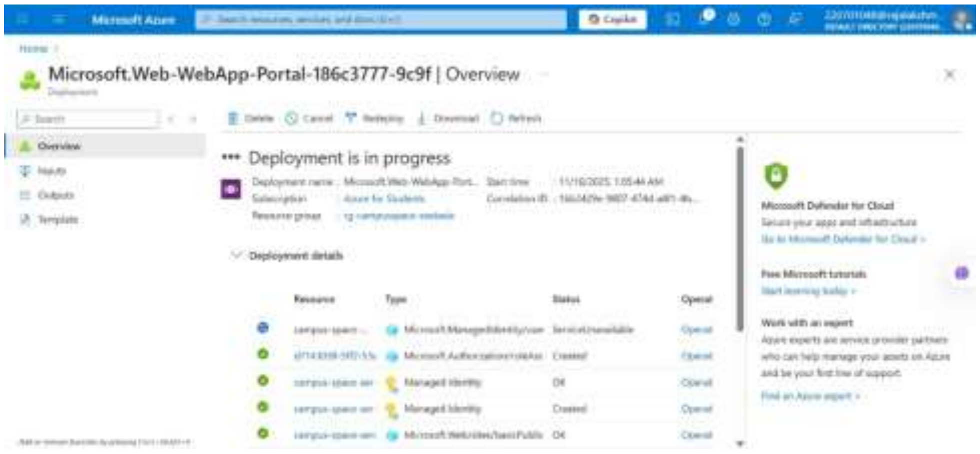
Figure 8.4 .apk build pipeline ci-cd.yml code



```
name: Build and deploy Node.js app to Azure Web App - campus-space-server

on:
  push:
    branches:
      - main
  workflow_dispatch:

jobs:
  build:
    runs-on: windows-latest
    permissions:
      contents: read #This is required for actions/checkout

    steps:
      - name: Checkout repo
        uses: actions/checkout@v4

      - name: Set up Node.js version
        uses: actions/setup-node@v3
        with:
          node-version: "20.x"

      - name: npm installing Dependencies
        run: |
          npm install
        working-directory: ./backend

      - name: Build Backend Server
```

Figure 8.5 server pipeline campus-space-server.yml code

Appendix C – Screenshots (Deployments, AI Integration, Security Scans)



Microsoft.Web-WebApp-Portal-186c3777-9c9f | Overview

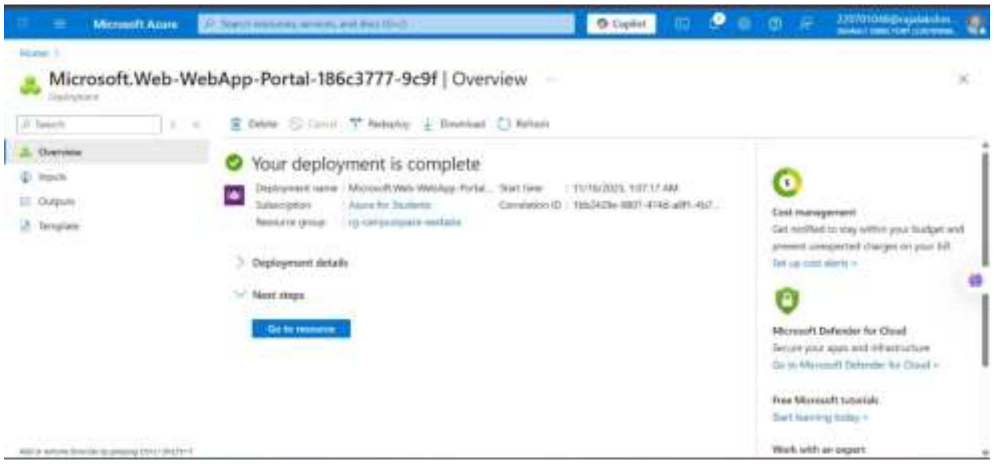
Deployment is in progress

Deployment name : Microsoft.Web-WebApp-Port... Start time : 11/16/2025, 1:05:44 AM
Subscription : Azure for Students Correlation ID : 1bb2429e-9807-474d-a8f1-4b...
Resource group : rg-campusspace-website

Deployment details

Resource	Type	Status	Operat
campus-space-...	Microsoft.ManagedIdentity/use	ServiceUnavailable	Operat
d7143098-5ff3-53c	Microsoft.Authorization/roleAss	Created	Operat
campus-space-ser	Managed Identity	OK	Operat
campus-space-ser	Managed Identity	Created	Operat
campus-space-ser	Microsoft.Web/sites/basicPublis	OK	Operat

Figure 8.6 Depolyment in progress

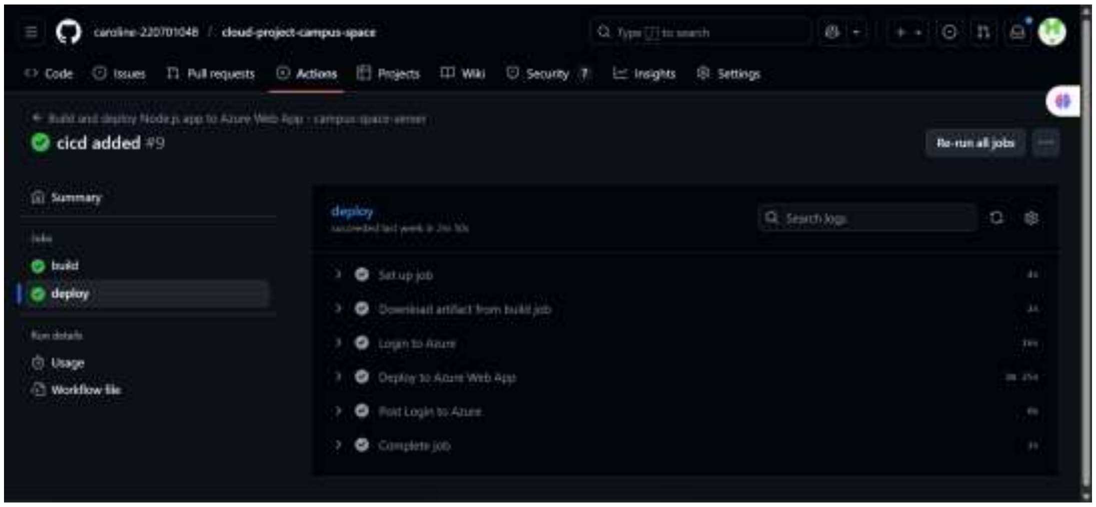


Microsoft.Web-WebApp-Portal-186c3777-9c9f | Overview

Your deployment is complete

Deployment name : Microsoft.Web-WebApp-Portal...
Subscription : Azure for Students
Resource group : rg-campusspace-westasia
Start time : 11/16/2025, 1:07:17 AM
Correlation ID : 1bb3423e-8807-4746-a8f1-4b7...

Figure 8.7 Deployment completed



caroline-220701048 / cloud-project-campus-space

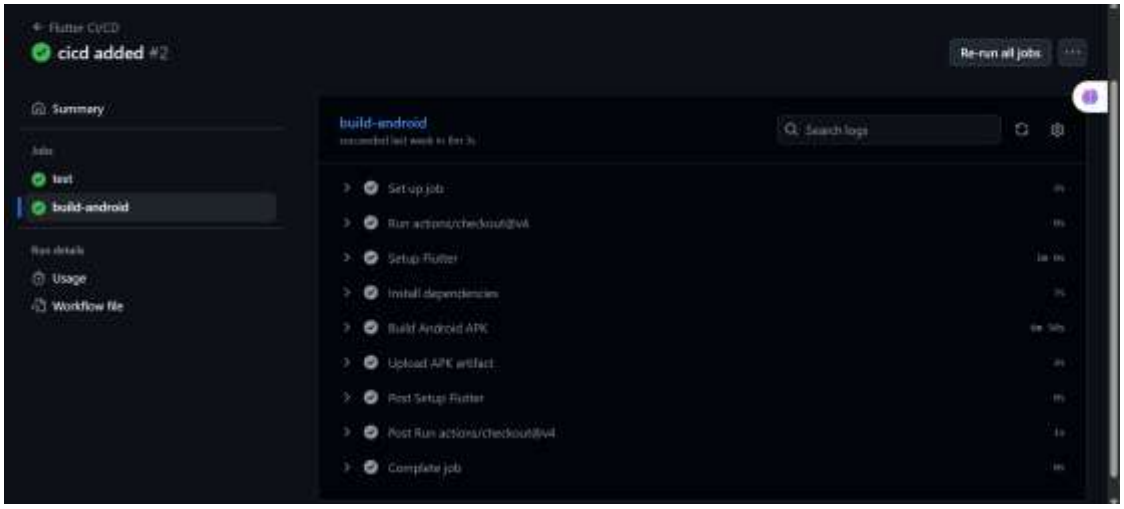
Build and deploy Node.js app to Azure Web App - campus-space-server

cicd added #9

deploy
succeeded last week in 2m 30s

Set up job
Download artifact from build job
Login to Azure
Deploy to Azure Web App
Post Login to Azure
Complete job

Figure 8.8 CI/CD for server


Flutter CI/CD

cicd added #2

Re-run all jobs

Summary

Jobs

test

build-android

Run details

Usage

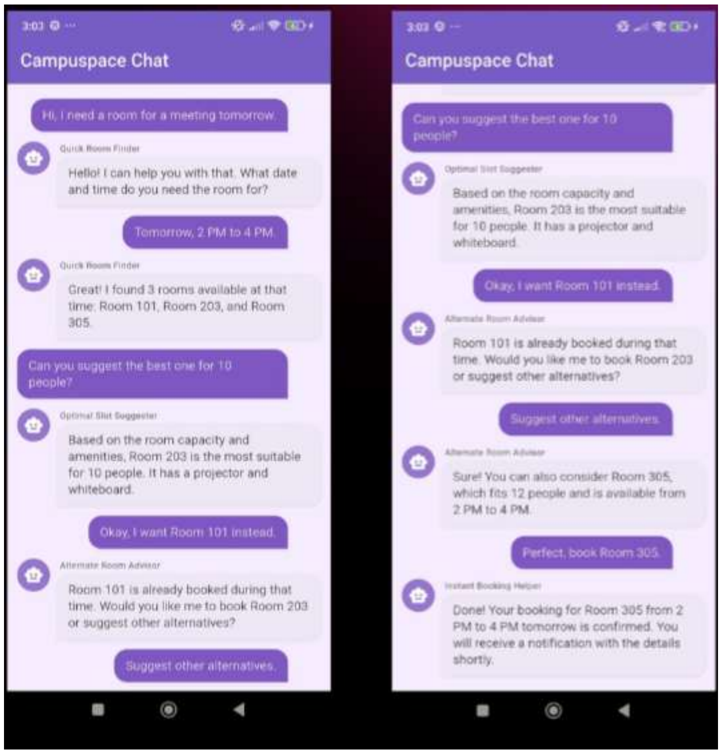
Workflow file

build-android

Search logs

```
Set up job
Run actions/checkout@v4
Setup Flutter
Install dependencies
Build Android APK
Upload APK artifact
Post Setup Flutter
Post Run actions/checkout@v4
Complete job
```

Figure 8.9 CI/CD for apk


Campuspace Chat

Hi, I need a room for a meeting tomorrow.

Quick Room Finder: Hello! I can help you with that. What date and time do you need the room for?

Tomorrow, 2 PM to 4 PM.

Quick Room Finder: Great! I found 3 rooms available at that time: Room 101, Room 203, and Room 305.

Can you suggest the best one for 10 people?

Optimal Slot Suggester: Based on the room capacity and amenities, Room 203 is the most suitable for 10 people. It has a projector and whiteboard.

Okay, I want Room 101 instead.

Alternate Room Advisor: Room 101 is already booked during that time. Would you like me to book Room 203 or suggest other alternatives?

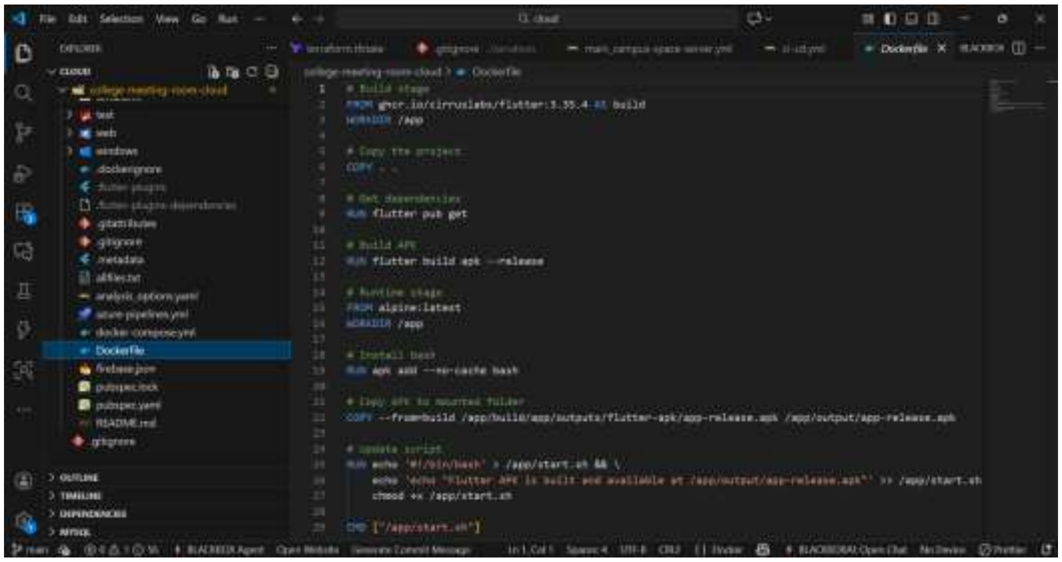
Suggest other alternatives.

Alternate Room Advisor: Sure! You can also consider Room 305, which fits 12 people and is available from 2 PM to 4 PM.

Perfect, book Room 305.

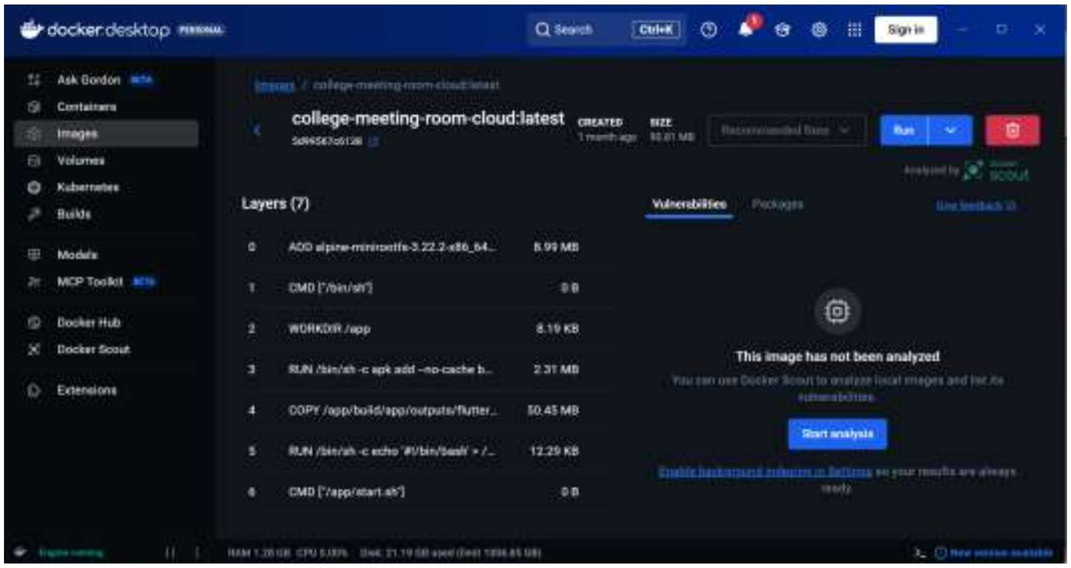
Instant Booking Helper: Done! Your booking for Room 305 from 2 PM to 4 PM tomorrow is confirmed. You will receive a notification with the details shortly.

Figure 8.10 GenAI use case-chatbot



```
# Build stage
FROM ghcr.io/cirruslabs/flutter:3.35.4 AS build
WORKDIR /app

# Copy the project
COPY . .

# Get dependencies
RUN flutter pub get

# Build APK
RUN flutter build apk --release

# Runtime stage
FROM alpine:latest
WORKDIR /app

# Install bash
RUN apk add --no-cache bash

# Copy APK to mounted folder
COPY --from=build /app/build/app/outputs/flutter-apk/app-release.apk /app/output/app-release.apk

# Update script
RUN echo '#!/bin/bash' > /app/start.sh && \
    echo 'echo "Flutter APK is built and available at /app/output/app-release.apk"' >> /app/start.sh
    chmod +x /app/start.sh

CMD ["/app/start.sh"]
```

Figure 8.11 Dockerfile for docker setup



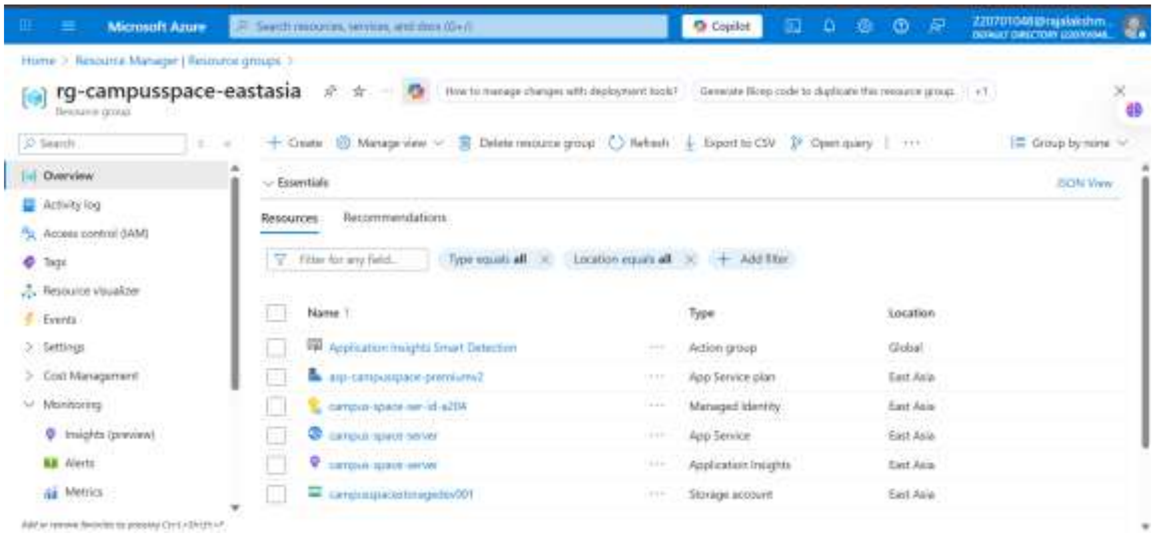
college-meeting-room-cloud:latest

Layers (7)

```
0   ADD alpine-minirootfs-3.22.2-x86_64...    8.99 MB
1   CMD ["/bin/sh"]                            0 B
2   WORKDIR /app                               8.19 KB
3   RUN /bin/sh -c apk add --no-cache b...     2.31 MB
4   COPY /app/build/app/outputs/flutter...     50.45 MB
5   RUN /bin/sh -c echo '#!/bin/bash' > /...   12.29 KB
6   CMD ["/app/start.sh"]                      0 B
```

This image has not been analyzed

Figure 8.12 Docker setup Completed

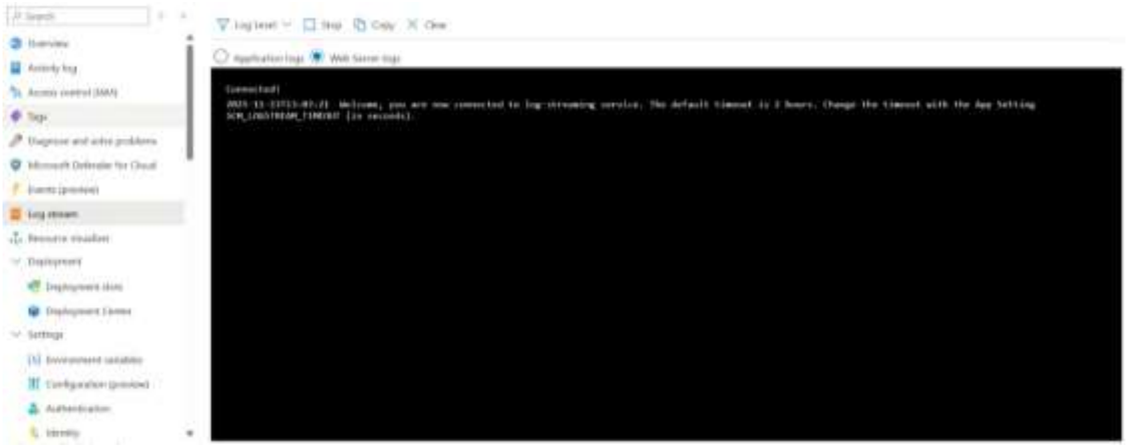


rg-campusspace-eastasia

Resource group

	Name ↑		Type	Location
	Application Insights Smart Detection	···	Action group	Global
	asp-campusspace-premiumv2	···	App Service plan	East Asia
	campus-space-ser-id-a2bA	···	Managed Identity	East Asia
	campus-space-server	···	App Service	East Asia
	campus-space-server	···	Application Insights	East Asia
	campusspacestoragedev001	···	Storage account	East Asia

Figure 8.13 Resource group with azure services



```
Connected!
2025-11-23T13:07:21  Welcome, you are now connected to log-streaming service. The default timeout is 2 hours. Change the timeout with the App Setting
SCM_LOGSTREAM_TIMEOUT (in seconds).
```

Figure 8.14 Log Stream service