CO4353 Distributed Systems
Mini Project

Visitor Management System

Group No: 07

20/ENG/014 - BATANGALA D.R.A.M.M.K.
20/ENG/016 - BUWANEKA R.I.
20/ENG/061 - JAYASINGHE M.A.R.D.

Table of Contents

Introduction

Visitor access in a majority of apartment buildings remains managed with outdated manual processes like word of mouth between security personnel and residents, and logbooks. Not only do these procedures take much time, but they are also susceptible to mistakes and pose a large security risk. Without verification procedures or ways of real-time communication, unverified individuals can enter, compromising the security of the residents. In addition, lack of role-based access and systematic record-keeping makes it impossible to track visitor activity or impose rigorous security protocols. With residential communities growing larger and more complex, there is a growing need for a secure, automated, and scalable visitor management system.

This project offers a Visitor Management System (VMS) for apartment complexes on the basis of microservices architecture. It allows residents to make digital invites in the form of QR codes or temporary access codes. Security personnel can validate these codes in real-time using a particular mobile application, permitting or denying entry on the basis of guidelines provided by residents. All access attempts are tracked with particular information such as timestamp, visitor name, and verification status, making traceability and accountability possible.

The system has three interface-related responsibilities: a mobile application for the residents to manage invitations, an independent mobile application for the security staff to verify guests, and a web interface for the administrators to oversee system behavior and handle users. The backend consists of four loosely coupled microservices—Admin, Visitor, Security, and Notification Services—each handling particular responsibilities and sending requests to each other via APIs. This microservices architecture provides flexibility, scalability, and ease of maintenance.

The chief objective of the project is to improve visitor safety, streamline verification procedures, and create an auditable record of all interactions with the visitor. By making use of current technologies and design practices, this system provides a realistic and reliable solution to the intricacies of visitor management in apartment buildings.

Architecture

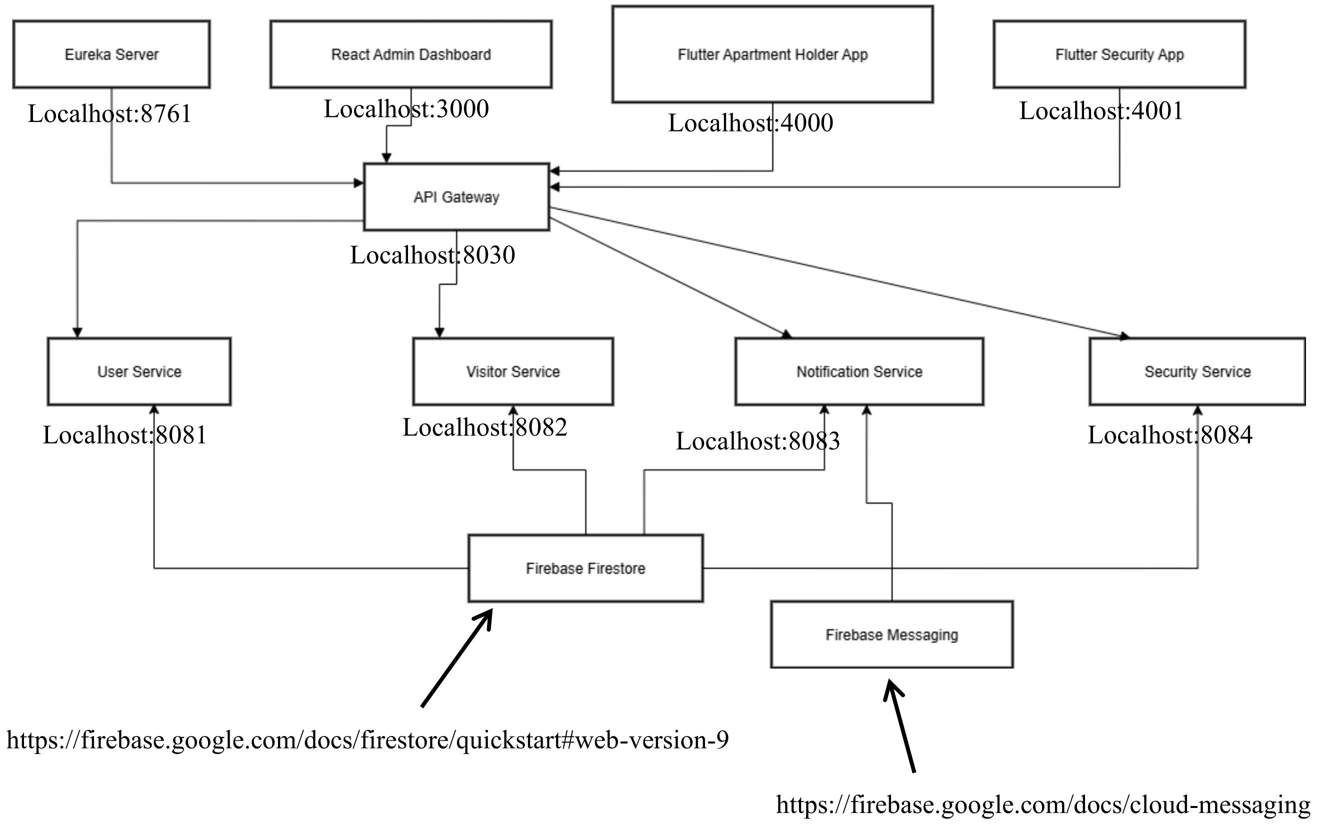


Figure 1 – Architecture Diagram

Component	**Local Address (Binding)**
Eureka Server	http://localhost:8761/eureka
API Gateway	http://localhost:8030
React Admin Dashboard	http://localhost:3000
Flutter Apartment Holder	http://localhost:4000
Flutter Security App	http://localhost:4001
User Service	http://localhost:8081
Visitor Service	http://localhost:8082
Notification Service	http://localhost:8083
Security Service	http://localhost:8084
Firebase Firestore	localhost:MeSK (placeholder – not a real port)
Firebase Messaging	localhost:MeSK (placeholder – not a real port)

Micro-services

The implementation methods used (Netflix software stack) Core services.

1. Admin Service

The Admin Service manages all user accounts and roles within the system, including apartment holders and security personnel. It is responsible for user registration, role assignment, account activation/deactivation, and system-wide configuration. This service ensures that access control is maintained based on roles and responsibilities.

2. Visitor Service

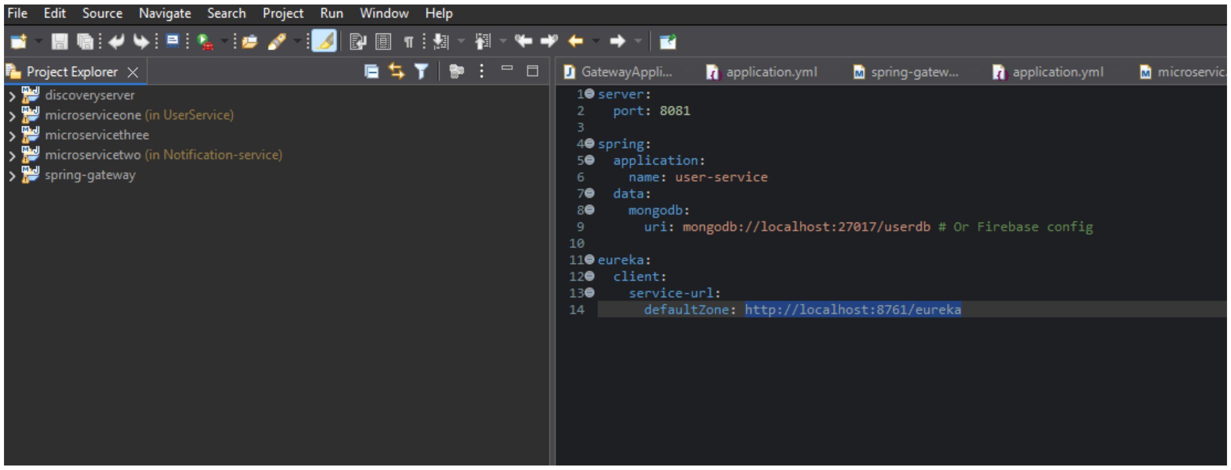
The Visitor Service handles the creation, modification, and retrieval of visitor invitations. It allows apartment holders to enter visitor details such as name, reason for visit, and time window. Once the information is submitted, this service generates a unique QR code or access code that is sent to the visitor. It also maintains a log of all invitations for tracking and transparency.

3. Security Service

The Security Service validates visitor access at the entry point. When a visitor presents a QR code or access code, the service checks the invitation's status, verifies time restrictions and permissions, and records the outcome. It also stores logs of all scanned entries, along with relevant visitor details and timestamps, helping enforce security policies effectively.

4. Notification Service

The Notification Service is responsible for sending communications such as QR codes or access links to visitors and entry notifications to apartment holders. It integrates with Firebase Messaging and/or email/SMS providers to deliver real-time updates, ensuring all stakeholders are informed promptly about visitor activities.

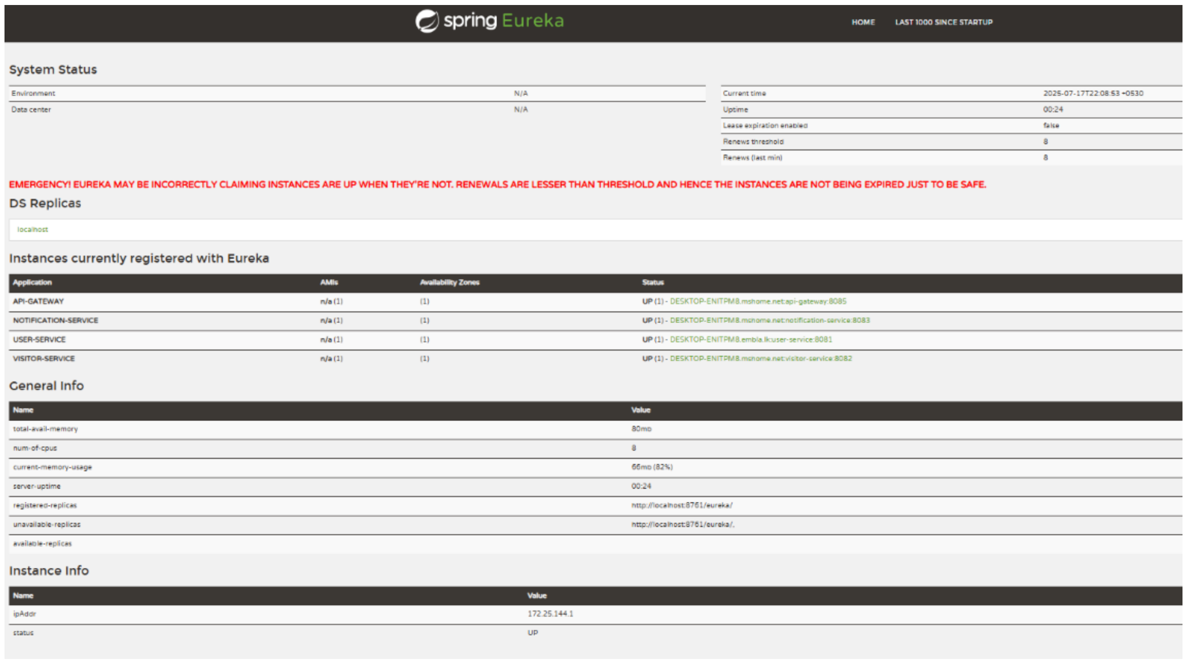


```yaml
server:
  port: 8081

spring:
  application:
    name: user-service
  data:
    mongodb:
      uri: mongodb://localhost:27017/userdb # Or Firebase config

eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka
```

Figure 2 – Implementation of microservices in Eclipse

Discovery Server

The Discovery Server, built using Netflix Eureka in the Visitor Management System, is an intermediate registry for microservices built using Spring Boot and Java. The server supports dynamic service discovery through the capability to register their location by microservices (e.g., user management, invitation, and access validation). The Eureka Server works in conjunction with the Spring Cloud Gateway (API Gateway) to route the client requests to the desired microservice without hardcoding service locations. This setup maximizes flexibility, scalability, and good communication in the system.



System Status

Environment	N/A
Data center	N/A

Current time	2025-07-17T22:08:53 +0530
Uptime	00:24
Lease expiration enabled	false
Renews threshold	8
Renews (last min)	8

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - DESKTOP-ENITPM8.mshome.net:api-gateway:8085
NOTIFICATION-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-ENITPM8.mshome.net:notification-service:8083
USER-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-ENITPM8.embla.lk:user-service:8081
VISITOR-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-ENITPM8.mshome.net:visitor-service:8082

General Info

Name	Value
total-avail-memory	80mb
num-of-cpus	8
current-memory-usage	66mb (82%)
server-uptime	00:24
registered-replicas	http://localhost:8761/eureka/
unavailable-replicas	http://localhost:8761/eureka/,
available-replicas	

Instance Info

Name	Value
ipAddr	172.25.144.1
status	UP

Figure 2 – Eureka Discovery Server Interface

API Gateway

The API Gateway serves as the central entry point for all client interactions in the Visitor Management System, facilitating secure, efficient, and scalable communication between clients (e.g., residents, security personnel, or administrators) and the system's microservices. The API Gateway is implemented using Spring Cloud Gateway within a Spring Boot and Java-based architecture. It handles critical functions such as request routing, authentication, authorization, load balancing, and response aggregation. This ensures seamless operation of the system's core functionalities, including invitation creation, QR code validation, and user management.

The API Gateway is integrated with a Eureka Server for service discovery, and it dynamically locates and routes requests to the appropriate microservices (e.g., user management, invitation generation, and access validation) based on their registered instances. For example, when a resident creates an invitation via a mobile application, the API Gateway authenticates the request, queries the Eureka Server to identify the invitation generation microservice, and forwards the request accordingly. Similarly, when security personnel validate a visitor's QR code, the API Gateway retrieves and aggregates relevant data, such as visitor details and access permissions, to provide a unified response.

The use of Spring Cloud Gateway enhances the system's modularity and scalability by enabling features like rate limiting, path-based routing, and security enforcement. It ensures role-based access control, allowing only authorized users (e.g., residents for invitation creation, security for validation) to perform specific actions. Additionally, the API Gateway centralizes monitoring and logging, providing traceability for all visitor access events, which is critical for maintaining security in the apartment complex.

User Interface

<u>**Apartment Holder Application**</u>

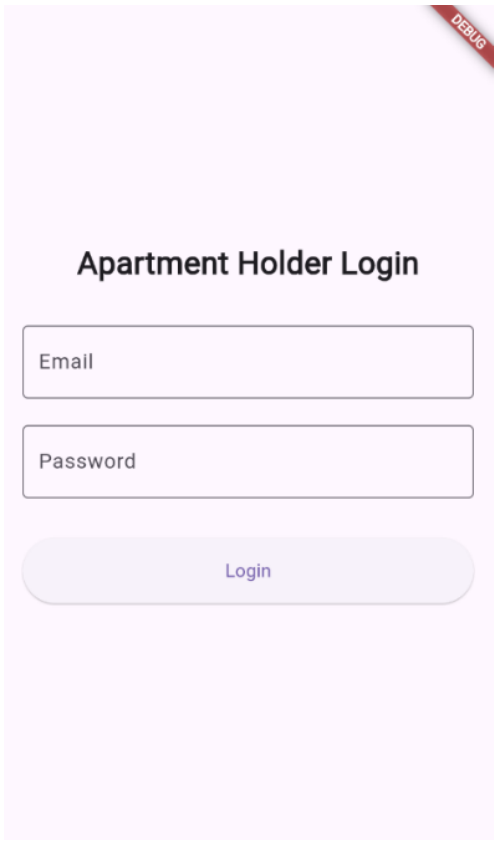


Figure 3 – Apartment Holder Login

The Apartment Holder app features a clean, user-friendly login screen that allows residents to securely sign in using their email and password. Its minimal design ensures easy access and usability for users of all age groups.

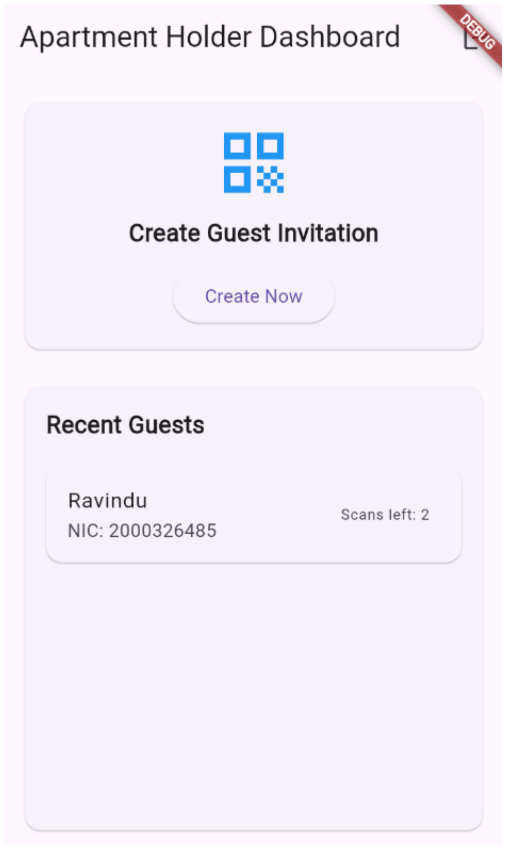
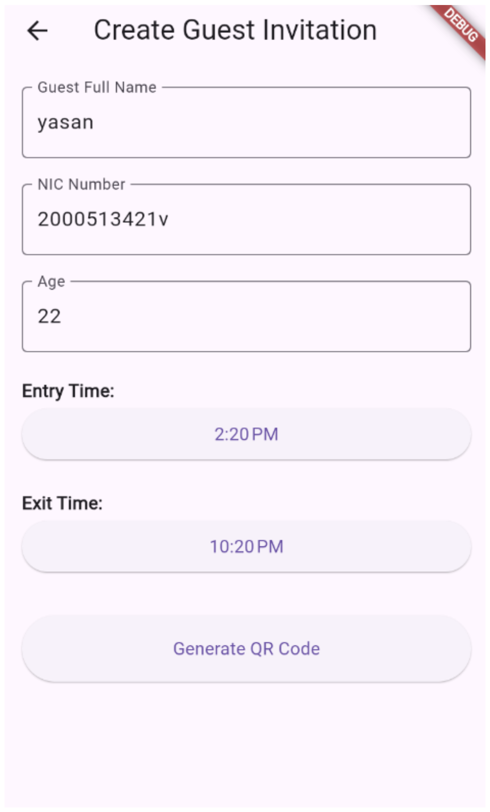


Figure 4 – Recent invitation view

Users access a dashboard after login to create and manage visitor invitations. They can generate digital invites with visitor details, receiving a shareable QR code or link. A "Recent Invites" section helps track past entries and avoid duplicates.



Create Guest Invitation

Guest Full Name: yasan

NIC Number: 2000513421v

Age: 22

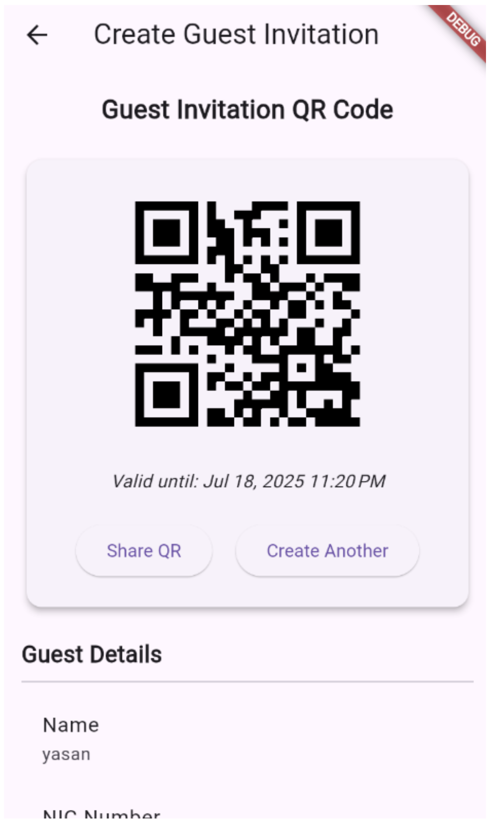
Entry Time:

2:20 PM

Exit Time:

10:20 PM

Figure 5 – Visitor Invitation creation



Create Guest Invitation

Guest Invitation QR Code

Valid until: Jul 18, 2025 11:20 PM

Guest Details

Name
yasan

NIC Number

Figure 6 – Generated QR

The Apartment Holder app offers a secure and digital solution for managing visitors, replacing physical passes with QR invitations. Its intuitive interface and invite history feature give residents full control over access, ensuring a safer and more convenient experience.

Security Person Application

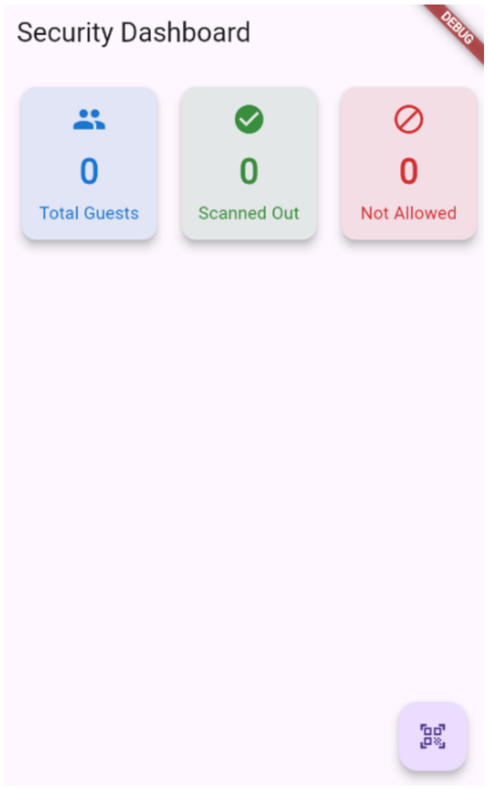


Figure 7 – Security Dashboard

The Security App features a simple dashboard that gives security personnel a real-time overview of visitor activity, including the total number of guests scanned and how many were granted or denied entry. This helps staff stay informed and maintain effective access control throughout their shift.

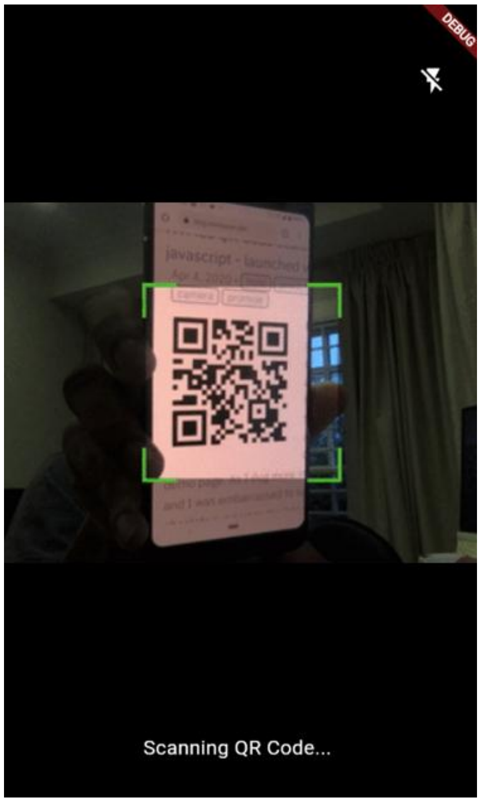
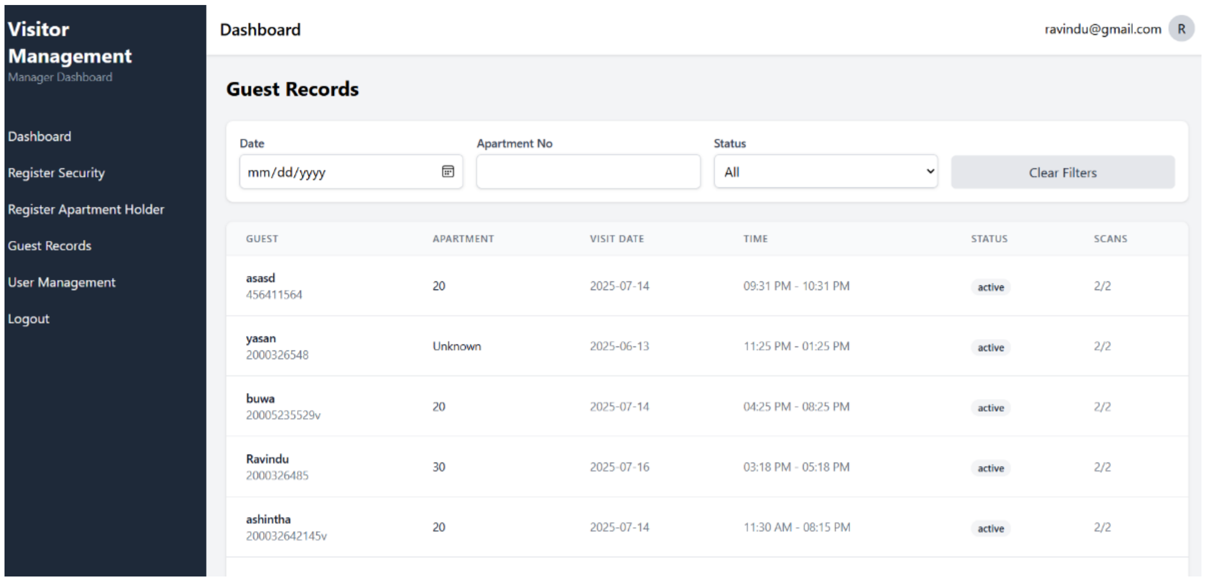


Figure 8 – Scanning the QR

Beyond the dashboard, the app allows security personnel to scan QR codes to verify invitation status and view guest details like name, visit reason, and apartment number. This ensures that only authorized visitors are granted entry, streamlining the check-in process.

Manager Application

The Manager Dashboard displays important statistics such as the total number of visitors, active users, registered apartments, and security personnel. This real-time overview provides a quick summary of system usage and helps managers identify patterns or issues. It acts as a control center, giving the manager both insight and authority to maintain security and flow in the residential complex.

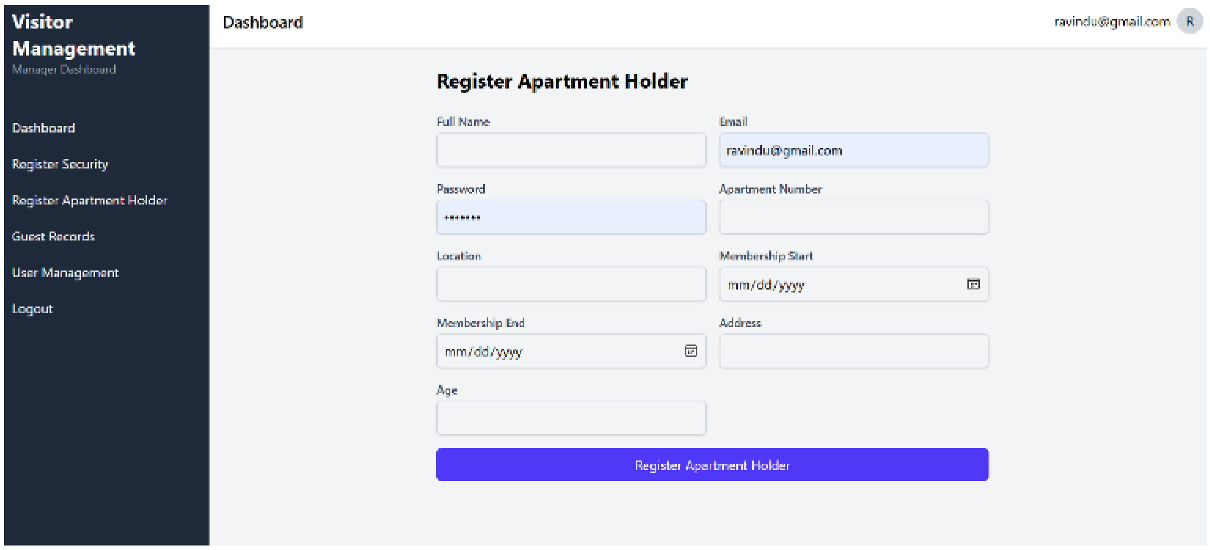


Guest Records

Date: mm/dd/yyyy
Apartment No:
Status: All

GUEST	APARTMENT	VISIT DATE	TIME	STATUS	SCANS
asasd 456411564	20	2025-07-14	09:31 PM - 10:31 PM	active	2/2
yasan 2000326548	Unknown	2025-06-13	11:25 PM - 01:25 PM	active	2/2
buwa 20005235529v	20	2025-07-14	04:25 PM - 08:25 PM	active	2/2
Ravindu 2000326485	30	2025-07-16	03:18 PM - 05:18 PM	active	2/2
ashintha 200032642145v	20	2025-07-14	11:30 AM - 08:15 PM	active	2/2

Figure 9 – Guest Records View

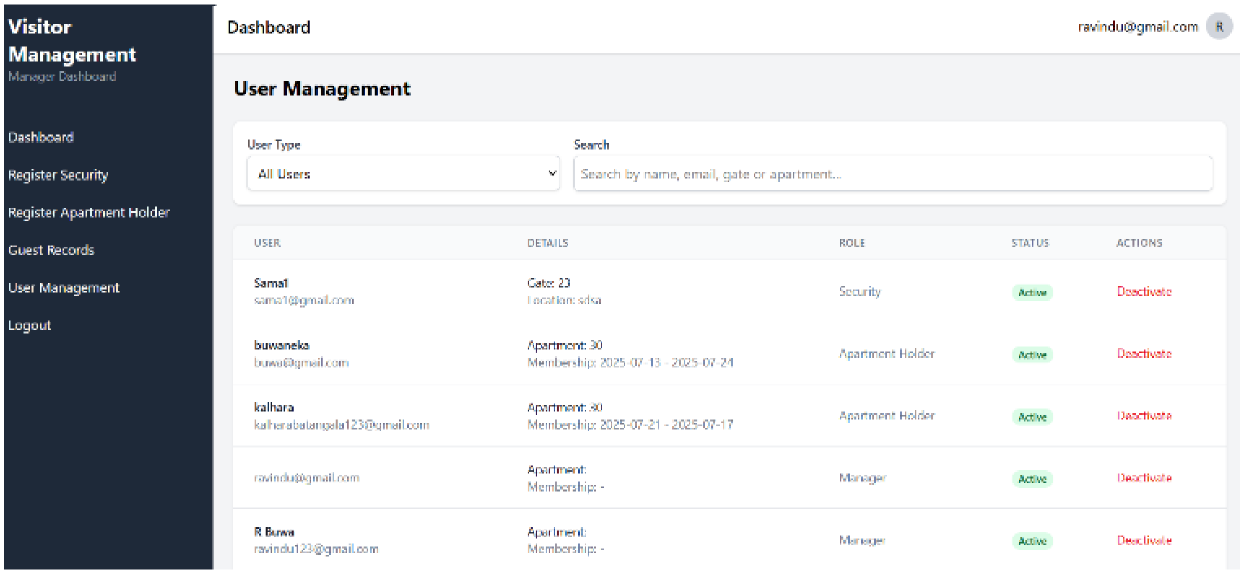


Register Apartment Holder

Full Name:
Email: ravindu@gmail.com
Password: •••••••
Apartment Number:
Location:
Membership Start: mm/dd/yyyy
Membership End: mm/dd/yyyy
Address:
Age:

Figure 10 – Registering Apartment Holders

9



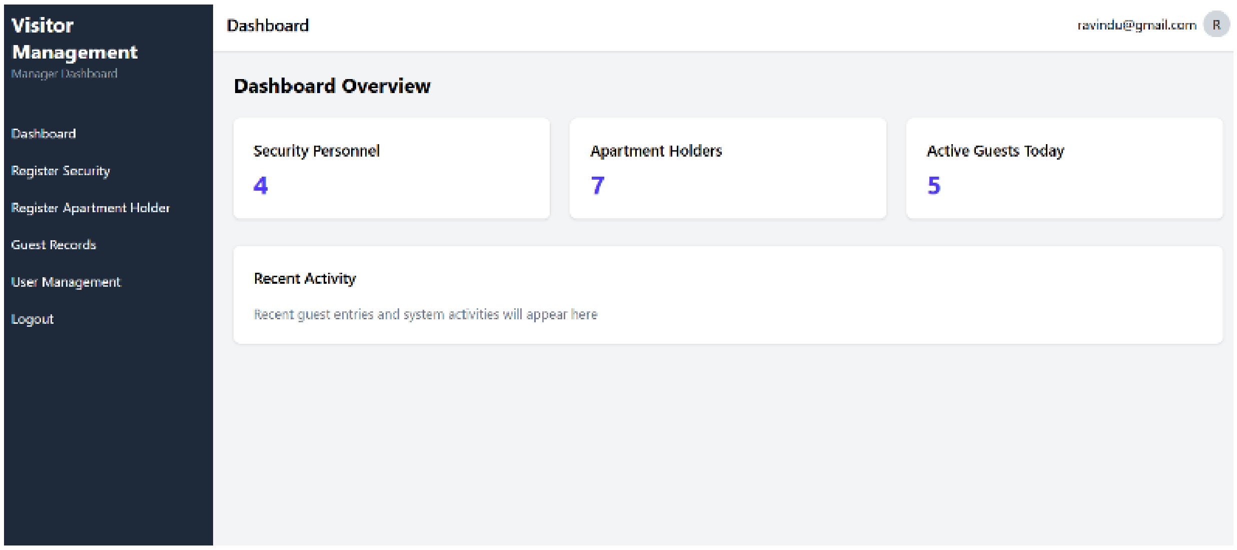
User Management

User Type: All Users

Search: Search by name, email, gate or apartment...

USER	DETAILS	ROLE	STATUS	ACTIONS
Samal samal@gmail.com	Gate: 23 Location: sdsa	Security	Active	Deactivate
buwaneka buwa@gmail.com	Apartment: 30 Membership: 2025-07-13 - 2025-07-24	Apartment Holder	Active	Deactivate
kalhara kalharabalangala123@gmail.com	Apartment: 30 Membership: 2025-07-21 - 2025-07-17	Apartment Holder	Active	Deactivate
ravindu@gmail.com	Apartment: Membership: -	Manager	Active	Deactivate
R Buwa ravindu123@gmail.com	Apartment: Membership: -	Manager	Active	Deactivate

Figure 11 – Managing Users



Dashboard Overview

Security Personnel	Apartment Holders	Active Guests Today
4	7	5

Recent Activity

Recent guest entries and system activities will appear here

Figure 12 – Overall Summary

The Manager Portal is the central admin dashboard for managing the visitor system. It allows managers to register apartment holders and security personnel, assign them to specific areas, and control user access by activating or deactivating accounts. This ensures proper role assignment and system security.

Deployment

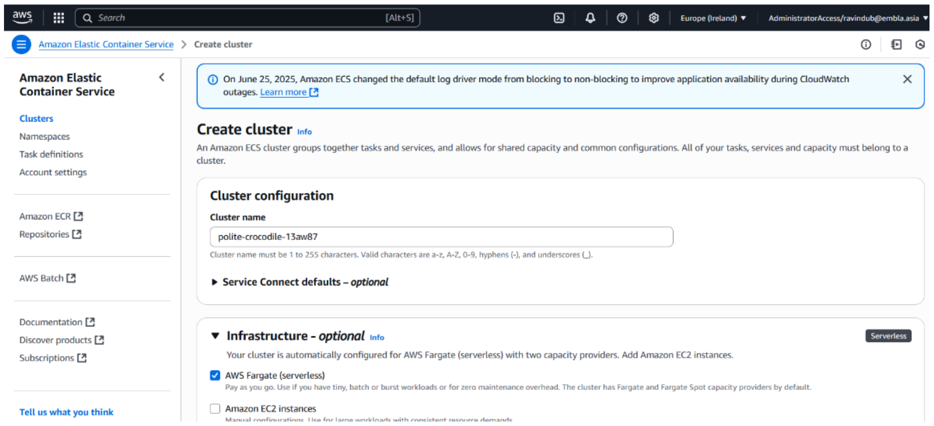
Proposed Deployment Method

Containerization with Docker

- Each microservice is containerized using Docker. This ensures that the application and its dependencies are packaged together, making the services portable, scalable, and consistent across different environments. Docker images for each service will be built and pushed to Amazon Elastic Container Registry (ECR).

Orchestration with AWS ECS Fargate

- Instead of managing EC2 instances manually, the deployment uses **AWS ECS with Fargate**, a serverless container management service. This allows you to run containers without managing servers, where each microservice runs as an independent ECS task or service. ECS handles task placement, scaling, and networking automatically.



On June 25, 2025, Amazon ECS changed the default log driver mode from blocking to non-blocking to improve application availability during CloudWatch outages. Learn more

Create cluster Info

An Amazon ECS cluster groups together tasks and services, and allows for shared capacity and common configurations. All of your tasks, services and capacity must belong to a cluster.

Cluster configuration

Cluster name: polite-crocodile-13aw87

Cluster name must be 1 to 255 characters. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Service Connect defaults – *optional*

Infrastructure - *optional* Info — Serverless

Your cluster is automatically configured for AWS Fargate (serverless) with two capacity providers. Add Amazon EC2 instances.

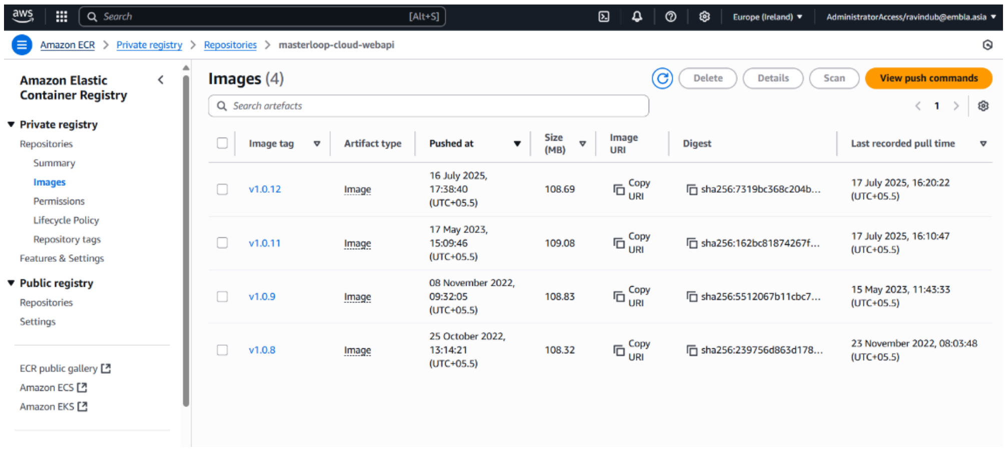
[✓] AWS Fargate (serverless)
Pay as you go. Use if you have tiny, batch or burst workloads or for zero maintenance overhead. The cluster has Fargate and Fargate Spot capacity providers by default.

[ ] Amazon EC2 instances

Figure 12 – Creating ECS Cluster

Cloud Infrastructure

- The entire infrastructure is hosted on **AWS**, ensuring high availability and scalability. Core services include **ECR** for image storage, **ECS (Fargate)** for container execution, **CloudWatch** for monitoring logs and metrics, and **IAM** for secure access control. Optionally, services like **RDS**, **S3**, or **DynamoDB** can be integrated depending on the backend needs.



Amazon ECR > Private registry > Repositories > masterloop-cloud-webapi

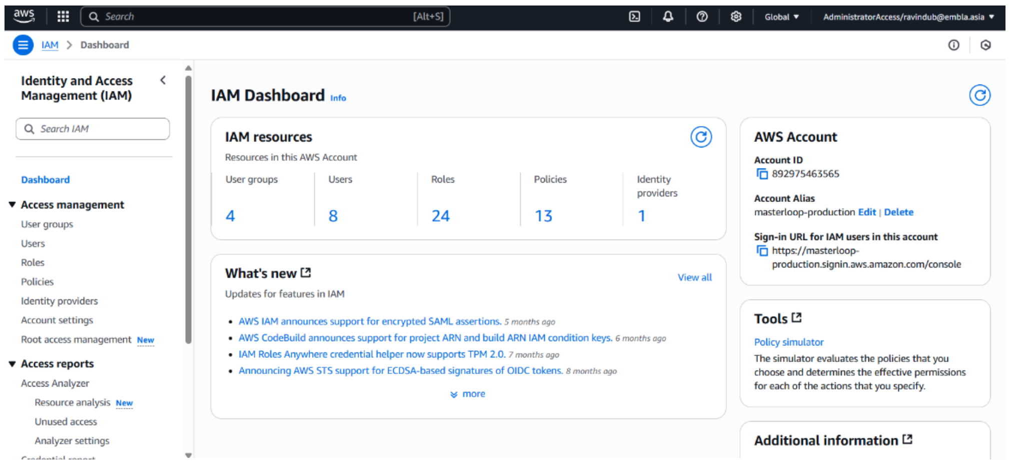
Images (4)

Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest	Last recorded pull time
v1.0.12	Image	16 July 2025, 17:38:40 (UTC+05.5)	108.69	Copy URI	sha256:7319bc368c204b...	17 July 2025, 16:20:22 (UTC+05.5)
v1.0.11	Image	17 May 2023, 15:09:46 (UTC+05.5)	109.08	Copy URI	sha256:162bc81874267f...	17 July 2025, 16:10:47 (UTC+05.5)
v1.0.9	Image	08 November 2022, 09:32:05 (UTC+05.5)	108.83	Copy URI	sha256:5512067b11cbc7...	15 May 2023, 11:43:33 (UTC+05.5)
v1.0.8	Image	25 October 2022, 13:14:21 (UTC+05.5)	108.32	Copy URI	sha256:239756d863d178...	23 November 2022, 08:03:48 (UTC+05.5)

Figure 13 – Adding Docker Images to ECR

Security

- Security is enforced through **IAM roles and policies**, ensuring services have the minimum required permissions. **Secrets Manager** or **SSM Parameter Store** can be used to manage sensitive configuration like database credentials or API keys. Network-level security is managed via **VPC**, **security groups**, and **load balancers** with HTTPS.



IAM Dashboard Info

IAM resources
Resources in this AWS Account

User groups	Users	Roles	Policies	Identity providers
4	8	24	13	1

What's new
Updates for features in IAM

- AWS IAM announces support for encrypted SAML assertions. *5 months ago*
- AWS CodeBuild announces support for project ARN and build ARN IAM condition keys. *6 months ago*
- IAM Roles Anywhere credential helper now supports TPM 2.0. *7 months ago*
- Announcing AWS STS support for ECDSA-based signatures of OIDC tokens. *8 months ago*

AWS Account
Account ID
892975463565
Account Alias
masterloop-production Edit | Delete
Sign-in URL for IAM users in this account
https://masterloop-production.signin.aws.amazon.com/console

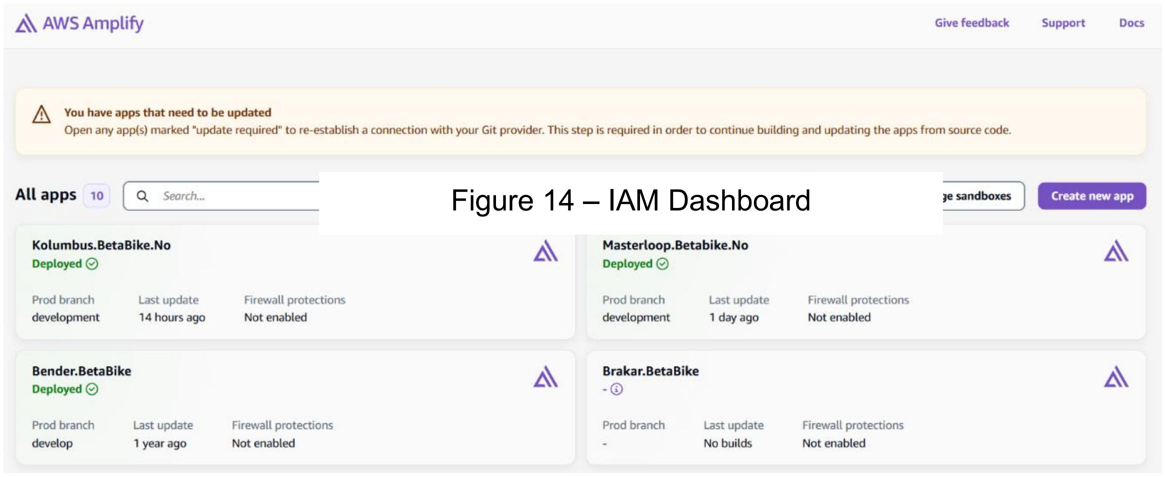
Tools
Policy simulator
The simulator evaluates the policies that you choose and determines the effective permissions for each of the actions that you specify.

Additional information

Figure 14 – IAM Dashboard

CI/CD Pipeline

- A CI/CD pipeline (using **GitHub Actions**, **GitLab CI**, or **AWS CodePipeline**) automates the build and deployment process. Whenever code is pushed, the pipeline builds the Docker image, pushes it to ECR, and updates the ECS service with the new image, ensuring smooth and fast delivery of updates with minimal manual intervention.



AWS Amplify — Give feedback · Support · Docs

You have apps that need to be updated
Open any app(s) marked "update required" to re-establish a connection with your Git provider. This step is required in order to continue building and updating the apps from source code.

All apps 10 · Search... · ge sandboxes · Create new app

Figure 14 – IAM Dashboard

App	Status	Prod branch	Last update	Firewall protections
Kolumbus.BetaBike.No	Deployed	development	14 hours ago	Not enabled
Masterloop.Betabike.No	Deployed	development	1 day ago	Not enabled
Bender.BetaBike	Deployed	develop	1 year ago	Not enabled
Brakar.BetaBike	-	-	No builds	Not enabled

Figure 15 – Setting up AWS Amplifier

<u>Summarized workflow</u>

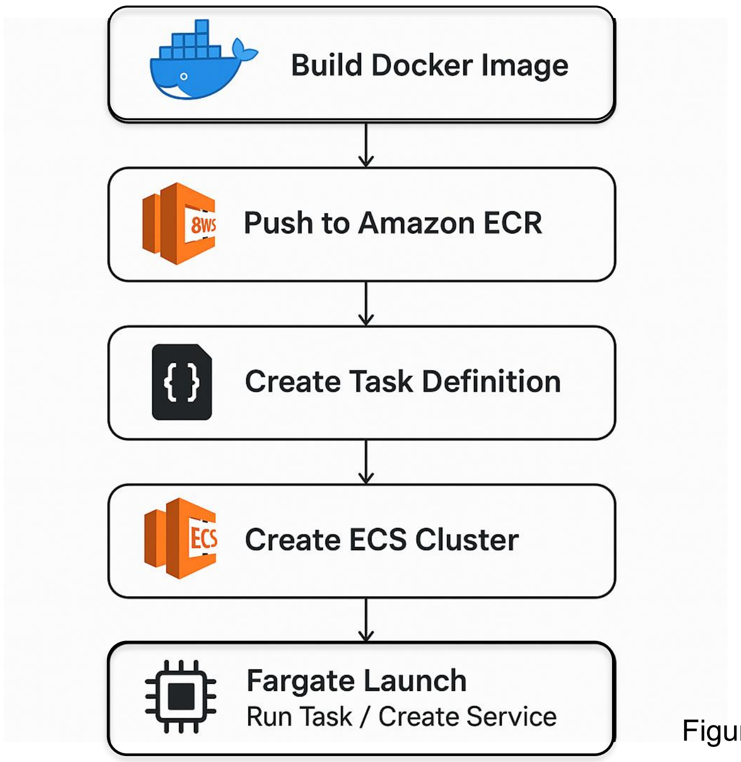


Figure 16 – Data Flow Diagram

Source Code

- Github link

Difficulties faced during development

- Designing role-based access with Firestore collections and rules was complex.
- Implementing smooth generation and scanning across platforms took several iterations.
- It is difficult to simulate microservice behavior using Firebase's serverless backend and NoSQL structure.

References

[1] "Visitor Management System Design and Implementation during the Covid-19 Pandemic," *Information Sciences Letters*, vol. 11, no. 4, pp. 1059–1067, Jun. 2022, doi: 10.18576/isl/110406.

[2] O. Muchamad, "Design and development of visitor management system," Bunghatta, Jan. 2019, doi: 10.15282/MEKATRONIKA.V1I1.152.

[3] "How to implement Firebase Firestore with a microservices architecture? | Bootstrapped Firebase Guides." https://bootstrapped.app/guide/how-to-implement-firebase-firestore-with-a-microservices-architecture