POE INSY7315 Senate Way Guest House

Web and Mobile Application

Team Members

Yadav Josh Maganbeharie ST10350794
Yadav Priaram ST10356506
Veeasha Pakirisamy ST10397833
Sahil Ramesar ST10356407

Links:

Main Website:
https://senatewayguesthouse.netlify.app/
Admin site: https://senatewayguesthouse.netlify.app/#admin
Snyk:
https://app.snyk.io/org/st10356506/project/e6659bc1-987c-40d8-b9c7-ee3308de162a/history/24115976-ef7b-4f1f-bab3-737c5ef37cd0
SonarQube:
https://sonarcloud.io/project/overview?id=senateway-guesthouse
Scrum:
https://senateway.atlassian.net/jira/software/projects/SCRUM/boards/1?atlOrigin=eyJpIjoiYjRmMGY1MzZkMDg5NGM3MmE4ODA1ODk0ZWUzYTQ1MDciLCJwIjoiaiJ9
Gantt Chart: https://magenta-syrniki-82a816.netlify.app/

Table of Contents

1. Introduction to the Project

1.1 Project Overview

The DIMEXL Guesthouse Application and Website project developed an online portal to book the DIMEXL Guesthouse a privately owned ten-bedroom guesthouse located in Kimberley. The system's primary goal is to improve guest engagement, streamline booking inquiries, and strengthen the guesthouse's online presence beyond third-party services such as Booking.com.

The app allowed guests to view available rooms, facility information and contact details, during which the property website mirrored the mobile application. The app was designed to be simple, responsive and easy to use across all Android devices; this gave the property greater online presence and improved SEO footprint. Both platforms are likewise branded, due to the experience and professionalism from DIMEXL.

Key Features

- **Home Page:** Welcoming interface showcasing the guesthouse, local attractions, and amenities.
- **Rooms Section:** Displays all ten rooms with detailed descriptions, high-quality photos, and pricing.
- **Gallery:** Highlights shared spaces such as the pool, garden, lounge, and braai area.
- **Location Integration:** Uses the **Google Maps API** to show the property and nearby landmarks.
- **Weather Forecast:** Implements **AccuWeather API** to display local weather in Kimberley.
- **Reviews & Contact:** Enables users to view, submit reviews, and contact management directly.
- **Booking Inquiry Form:** Allows users to submit booking requests; responses are managed by the SenateWay team.
- **AI Chatbot:** Integrates **Gemini AI** to answer user queries efficiently and handle FAQs.

API Integrations

- **Google Maps API** – Property location and points of interest.
- **EmailJS** – Automated email notifications for inquiries.
- **AccuWeather API** – Current weather display.
- **Gemini API** – AI chatbot integration for user assistance.

Booking Management

Currently, all reservations are processed through Booking.com. An alternative direct-inquiry system managed by DIMEXL through the app and website is part of the project's plans. Rather than payments being processed automatically, statuses are updated manually by the SenateWay team, allowing better communication.

Research into the Client Organisation

DIMEXL Guesthouse operates offline efficiently according to its owners' reports, but relies heavily on booking websites, which prevents establishments from promoting themselves as a brand and making direct contact with customers. The client wanted to reduce commission payments and take a meaningful role in customer communication and brand promotion while maintaining a high level of quality. Thus, the solution was designed to provide one integrated digital experience, in line with the guesthouse's aims of independent, convenience and professionalism.

Ethical and Privacy Considerations

Given the system's handling of personal and booking data, the team adhered to ethical and privacy principles:

- All personal data collected (e.g., names, emails, booking details) is handled in compliance with **POPIA** (Protection of Personal Information Act, South Africa) "This aligns with POPIA data protection guidelines (Staunton, Tschigg & Sherman, 2021)."
- No sensitive information is stored without encryption or consent.
- The AI chatbot avoids collecting or storing personal identifiers.
- Access to administrative data is restricted to authorized team members only.
- Regular code reviews and data handling audits were performed to maintain transparency and accountability.

1.2 Work Agreement

For the DIMEXL development team, adoption of an Agile Scrum model promoted constant team communication, iteration, and quality progress. The work agreement was the core of the team's discipline, communication, and accountability over the life of the project.

Team Responsibilities

- Each team member was assigned a role according to their expertise within: UI/UX design, database configuration, software documentation, testing, and configuration management. Tasks were divided into bi-weekly sprints as determined by story points and the dependencies between tasks.
- All members met with clients, reviewed sprints, and tested quality assurance to ensure the solution met both functional and visual requirements.

Communication

- The tea, communicated by WhatsApp, Microsoft Teams, and an Atlassian Scrum Board (Asproni, G., 2006)gus, a sprint management tool. Weekly meetings included reviews of completed sprints and progress in addition to planning more sprints. Immediate issues were discussed in the group chat in-between meetings to not disrupt each other's workflow.

Collaboration Tools

- **GitHub:** Version control and collaborative coding.
- **Atlassian Scrum Board:** Task management and sprint tracking.
- **Microsoft Excel:** Sprint tracking and performance recording.
- **Microsoft Word:** Shared documentation.

Accountability and Transparency

- During development, all members logged their work and checked-in their source code, which were all peer reviewed by other members. A task is only considered done when the entire team agrees that it has met quality, functionality, and standards that will satisfy the client.

1.3 Definition of Ready (DoR)

The Definition of Ready (DoR) is a checklist of conditions to be met for an user story or task to be considered ready for the development phase, including the understanding of its goals, requirements, and dependencies.

A feature is **Ready** when:

- The user story and acceptance criteria are clearly defined and approved.
- All dependencies, resources, and requirements are identified and documented.
- Sprint capacity is confirmed through story point estimation.
- UI/UX mockups and technical specifications are complete.
- All necessary configurations and API setups are available.
- The team collectively agrees the task is achievable within the sprint.

This process minimizes scope creep and confusion, ensuring that every sprint begins with fully prepared and validated work items (Power, K., 2014).

What the Definition of Ready Means to Us

For our team, the Definition of Ready represents **alignment and preparedness**. It ensures that before we start coding or designing, everyone understands *what needs to be done, why it matters,* and *how success will be measured*. Having a clear DoR prevents wasted time, misunderstandings, and rework. It gives us confidence that each feature has enough information, designs, and resources for smooth development. Essentially, DoR helps us begin every sprint with **clarity, shared purpose, and confidence** that our efforts will directly contribute to the project's success.

1.4 Definition of Done (DoD)

Definition of Done (DoD) is an agreed set of criteria or checklist, decided by a team, that must be met before declaring any work complete in terms of technical, ethical, and functional quality (Dalton, J., 2018).

A feature or user story is **Done** when:

- All acceptance criteria are met.
- Code is complete, peer-reviewed, and merged into the main branch.
- All tests (manual or automated) pass successfully with no major bugs.
- Documentation and deployment notes are updated.
- The feature has been reviewed and approved by the client or lecturer.
- Ethical and privacy standards (POPIA compliance, no data exposure) are upheld.

This means that all deliverables are production-ready and remain high quality and long-lived without any technical debt.

What the Definition of Done Means to Us

To us, the Definition of Done is our commitment to quality, accountability and pride in our work. It's the assurance that anything we ship is fully functional, tested to the highest standards, and meets the expectations set forth by the client. According to the DoD, a job is only done when it can be presented to the customer, when it has been done ethically and transparently, and when a visible, measurable requirement has been met. This is our approach to professionalism, excellence and user trust towards the DIMEXL Guesthouse system.

Ready is a checklist that lists acceptance criteria that the user stories need to meet, so that the development team has a shared understanding of them and is able to begin working on them in the sprint. To reduce duplication and scope creep, except in emergencies, the DoR requires all information, requirements, dependencies and resources to be available and documented pre-sprint.

SenateWay development team members are confident that all the features are clearly defined, documented and is prepared for developing, which enables a team to start working with the team having absolute confidence, gaining a clear understanding of requirements, scope, and expected outcomes for each feature. DoR is a very important factor.

It is considered *Ready* when the following conditions are met:

- There is a very clear description of and has an acceptance criterion in place to ensure that the teams goals and objectives are clearly defined so that the team collectively understands what needs to be done and how to meet goals and objectives to ensure success and how success will be measured
- The team agrees collectively on a set priority for the set project and its various tasks within each sprint
- The team will identify any important dependencies required and handle them withing the projects sprint timelines
- Sprint capacity planning is determined via the team estimating the work using story points
- There are UI/UX mockups in place for features that need to be implemented
- All integrations such as APIs, Authentication and configurations are all set up

1.5 Roadmap and Scrum Board

Link to the Scrum Board providing our timeline, access to our sprints and backlogs as well as the dashboard:

https://senateway.atlassian.net/jira/software/projects/SCRUM/boards/1?atlOrigin=eyJpIjoiYjRmMGY1MzZkMDg5NGM3MmE4ODA1ODk0ZWUzYTQ1MDciLCJwIjoiaiJ9

(Asproni, G., 2006)

“Roadmaps are strategic planning tools that guide organizational progress (Kappel, 2001).”

1.6 Project Risks and Mitigations

Risk	**Impact**	**Mitigation Strategy**
API service downtime	Medium	Implement fallback messages and cached data.
Data privacy breach	High	Enforce encryption, restrict access, and comply with POPIA.
Time management issues	High	Use strict sprint deadlines and weekly progress meetings.
Team communication delays	Medium	Maintain daily check-ins via WhatsApp.
Client scope changes	Medium	Record all changes in backlog and manage via sprint review.
Technical bugs during deployment	High	Perform pre-release testing and rollback planning.

2. Requirements

2.1 User Roles

The user experience of Senate Way Guest House was designed for two types of users, a guest user and an admin (the Owner). A different user interface was used for both of these user types to satisfy their needs in interacting with the application.

1. Guest

Description:
The target audience for the Senate Way application and website is guests looking to book or inquire about a room in the guest house, which guests can do through the mobile application or website.

Key Functions and Permissions:

- Guests can view room listings, descriptions, as well as images for each room or facilities.
- Guests can view the pricing of rooms and inquire about availability via a booking request on the contact page.
- View the photo gallery and property amenities to allow guests see all the images of the guest house and what it has to offer.
- Access location information via Google Maps API integration which will allow the guest to also open a route to the guest house.
- Submit contact forms or booking inquiries via the contact page and will receive an email regarding their booking request.
- Optionally leave feedback or reviews.
- Guests also can make use of the AI chat bot which is tailored to the Guest House and provides information on FAQ or any other questions the user may have.

Goal:
To deliver a seamless, informative, and user-friendly digital experience that enables guests to easily explore the Senate Way Guest House, access essential information, make booking inquiries, and engage directly through interactive tools ultimately fostering convenience, trust, and increased direct bookings.

2. Administrator (Owner – *Ms. Vanessa Packirisamy*)

Description:
DIMEXL Guesthouse is owned by Ms. Vanessa Packirisamy, who is the system administrator, the highest level of user. This user defines and maintains the content, updates property information, recognizes the issues in the system, and provides a proper guest experience. She also performs communications within the workspace of the system.

Key Functions and Permissions:

- Monitor guest inquiries and manage responses through the integrated contact system powered by EmailJS. This system automatically notifies guests about the status of their booking inquiries, and upon admin approval, sends a confirmation email to finalize the process.
- They can View a set of analytics that display how many bookings were made, how many reviews there are, how many pending and confirmed booking there is, however on the website they can access more analytics such as user engagement rate, average views per user, total users on the site, pages views and lastly interactions (Kujala et al. 2005).

Goal:
To give the administrator a full set of tools to manage the site, including options for reviewing booking inquiries, managing communication with guests through automated email notifications, and providing data-driven perceptions such as booking statistics, reviews monitoring, user activity tracking, and thorough website performance monitoring.

Use Case Diagram:

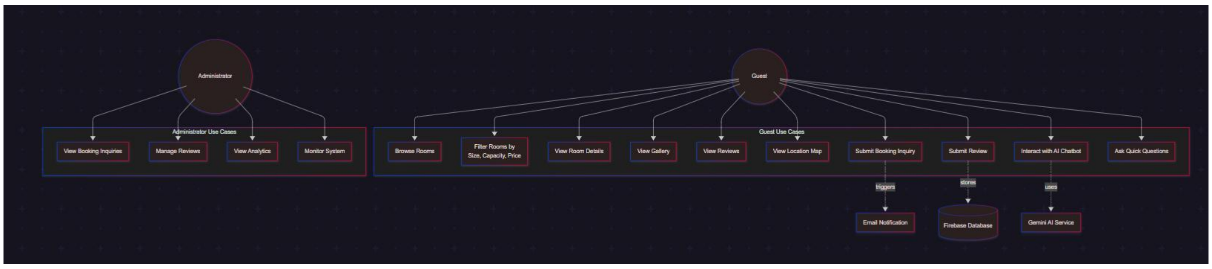


Figure 1: Use Case Diagram

In Figure 1 this diagram defines what each type of user can do and outlining their interactions with the system and related external services

2.2 User Stories

1. Owner (Vanessa Packirisamy)

As the owner of the DIMEXL Guesthouse, I want to access a dashboard view for booking inquiries, guest reviews, website and mobile application statistics and analytics.

This helps me track business performance. This helps me identify trends. This helps me make informed decisions regarding pricing, marketing strategies, and areas for improvement.

- **Business Priority:** High — Central to guesthouse management and decision-making
- **Team Estimation:** 8 Story Points (Complex – requires data aggregation and dashboard visualization).
- **Sprint Assigned:** Sprint 3 (Implementation and Testing Phase).
- **Implementation Status:** In Progress – Dashboard structure designed; data integration in development.

User Story Grooming and Reprioritization

The user stories that include Admin Dashboard plus Guest Booking had review, grooming, and reprioritization during the development process during sprint planning and backlog refinement sessions.

User stories were sequenced by business value and technical dependency, so that essential features such as the guest booking system and contact integration were developed in the first sprints.

The team during development found technical debt and bugs that needed fixing or updating. The priority of these stories shifted within sprints; for example, tweaks to the front-end and data visualizations on the dashboard were moved to Sprint 3 to allow for further integration testing.

During each grooming session, stories were:

- Reviewed for clarity and completeness to meet the Definition of Ready (DoR).
- Estimated using Story Points to determine development effort.
- Reprioritized based on feedback, testing outcomes, and dependencies identified in previous sprints.
- Updated to reflect any changes or enhancements requested by the product owner.

Grooming and reprioritization of the backlog iteratively ensured that the product kept evolving in line with project objectives and with stakeholders (Lucassen et al. 2016).

2. Guest

As a Guest,
I want to browse available room options, view amenities, and submit booking inquiries easily,
so that I can plan my stay conveniently and communicate directly with the guesthouse.

- **Business Priority:** High — Directly supports user experience and engagement.
- **Team Estimation:** 5 Story Points (Medium – front-end functionality and form integration).
- **Sprint Assigned:** Sprint 1 (Initial Development Phase).
- **Implementation Status:** Completed – Room listings, inquiry form, and API integration functional.

User Story Grooming and Reprioritization

These user stories were groomed and improved during sprint planning. Guest room browsing and booking inquiry were prioritized based on their potential impact to the users and core functionality of the system and scheduled for sprint 1.

The story was frequently revisited and reordered during development when minor updates and bug fixes were discovered, for example:

- Adjustments were made to room listings and amenity displays to ensure clarity and accuracy.
- The booking inquiry form was updated to improve usability and error handling.
- Story priorities were shifted slightly in subsequent sprints to accommodate front-end enhancements and integration fixes.

This iterative process of grooming, reprioritization, and updating ensured that the guest-facing functionalities remained intuitive, reliable, and aligned with project goals.

3. Non-Functional Requirements

The non-functional requirements define the overall quality standards and operational expectations of the DIMEXL Senate Way Guesthouse system. These requirements apply to both the **Website** and **Mobile Application**, ensuring consistent functionality, performance, and user experience across devices and platforms.

It is a priority for DIMEXL Guesthouse that users can interact with the platform seamlessly on both Web and Mobile. These requirements ensure that the platform not only functions correctly but also provides an efficient, secure, and enjoyable experience.

By maintaining these standards, the platform delivers high **performance**, **reliability**, and **usability** while ensuring **security**, **maintainability**, and **scalability** for future growth (Glinz, M., 2007, October.).

3.1 Performance Requirements

Performance requirements ensure that both the DIMEXL Guesthouse Web Application and Mobile App deliver a fast, smooth, and responsive experience. (Okubo, S., McCann, K. and Lippmann, A., 1995.)

Requirements:

- All screens (Web + Mobile) must load within 3 seconds on a standard broadband or 4G connection.
- Image files (e.g. gallery and room photos) must be optimised for speed without losing quality.
- The system must support at least 50 concurrent users with minimal performance degradation.
- Server response time must remain below 500 ms under normal usage.
- Frequently accessed data (e.g. images, room listings) should use local caching to reduce server load.

Goal:

To ensure a fast, fluid experience for all users, regardless of connection or device, even during high-traffic periods.

3.2 Scalability Requirements

Scalability ensures that the DIMEXL platform can grow as the business expands, without major redevelopment (Duboc et al., 2008).

Requirements:

- The architecture must support future integrations (e.g., online payments, staff management, or booking automation).
- The database must allow adding new rooms, features, and amenities without breaking existing data.
- The system must be deployable on **cloud-based infrastructure** (e.g., Azure, AWS) with scalable bandwidth and storage.
- Both the Web and Mobile App should share a common backend to avoid data duplication.

Goal:

To ensure the system can expand seamlessly in line with future business and technical requirements.

3.3 Reliability Requirements

Reliability ensures that the system consistently performs its functions without failure. (Hassine, J., 2015.)

Requirements:

- Maintain **99% uptime** under normal operating conditions.
- Implement **daily automated data backups** to prevent data loss.
- If server errors occur, the app must display user-friendly error messages and retry automatically.
- Admins must receive automated **email alerts** for critical system or database errors.
- Conduct periodic maintenance checks to ensure system stability.

Goal:

To provide continuous and dependable service to guests and administrators without interruptions or data loss.

3.4 Maintainability Requirements

Maintainability defines how easily the system can be updated, debugged, and improved over time. (ISO/IEC 9126-1:2001.)

Requirements:

- Code should follow **clean architecture principles** (MVVM for Android; modular and reusable components).
- Use **consistent coding conventions** and proper naming standards across Kotlin and XML files.
- Maintain **well-documented code** using comments and README files for all modules.
- All dependencies (libraries, APIs) must be version-controlled and regularly updated.
- Errors and bugs must be logged systematically (e.g. via Logcat or Firebase Crashlytics).
- The system should allow **feature updates and UI adjustments** without rewriting the entire app.

Goal:

To ensure future developers can easily maintain, modify, or upgrade the system with minimal downtime and maximum efficiency.

3.5 Security Requirements

Security ensures data confidentiality, integrity, and protection from unauthorised access (Almogahed et al., 2022).

Requirements:

- All communication between users and the system (Web + Mobile) must use HTTPS (SSL/TLS) encryption.
- Input validation must prevent SQL injection, XSS, and form spam.
- User passwords must be hashed and salted before storage.
- Admin accounts must include session timeouts after inactivity.
- Regular security audits should be conducted (e.g. vulnerability scans via Snyk).
- Sensitive information (e.g. booking details, emails) must be encrypted at rest.

Goal:

To maintain user trust and ensure that all data, personal and booking-related, remains fully protected at all times.

3.6 Usability Requirements

Usability ensures the system is intuitive and accessible to all users, regardless of technical ability. (Juristo, N., 2006.)

Requirements:

- The design must follow **Material Design Guidelines** for Android.
- Maintain **consistent navigation** across screens with clear labels and icons.
- Use **responsive layouts** that adapt to different screen sizes and resolutions.
- All text and visual elements must follow **WCAG 2.1 accessibility standards**.
- Provide clear validation messages and success/failure feedback for form submissions.
- Core actions (e.g., booking, viewing rooms) should be accessible within **three taps**.

Goal:

To deliver an intuitive, visually appealing, and easy-to-use interface that enhances the guest experience.

3.7 Interoperability Requirements

Interoperability ensures the DIMEXL system can communicate effectively with other services and platforms. (Koziolek, H., 2011.)

Requirements:

- The system must integrate with third-party APIs such as Google Maps, EmailJS, AccuWeather, and Gemini AI.
- All integrations should use RESTful APIs for standardised communication.
- The backend must support JSON data format for mobile and web data exchange.
- Ensure cross-platform compatibility between Android Studio (Kotlin) and web technologies.
- The app should be able to interface with future third-party tools such as analytics, or CRM solutions.

Goal:

To maintain seamless data exchange and compatibility between DIMEXL's digital platforms and any future integrated services.

3.8 Internationalisation / Localisation Requirements

Internationalisation ensures the app can support multiple languages and cultural formats. (Esselink, B., 2000.)

Requirements:

- The mobile app must be designed to support **multiple languages** through Android's strings.xml file.
- All text and content must be stored as **externalised strings**, not hardcoded, to simplify translation.
- Dates, currencies, and units should follow **user locale settings** (e.g., South African Rand displayed as "R").
- The system should allow easy translation updates without changing source code.
- The app must display UTF-8 encoded characters to support global languages.

Goal:

To make the DIMEXL platform adaptable for future international guests, ensuring inclusivity and global accessibility.

4. User experience (UX)

4.1 User journey map and explanation

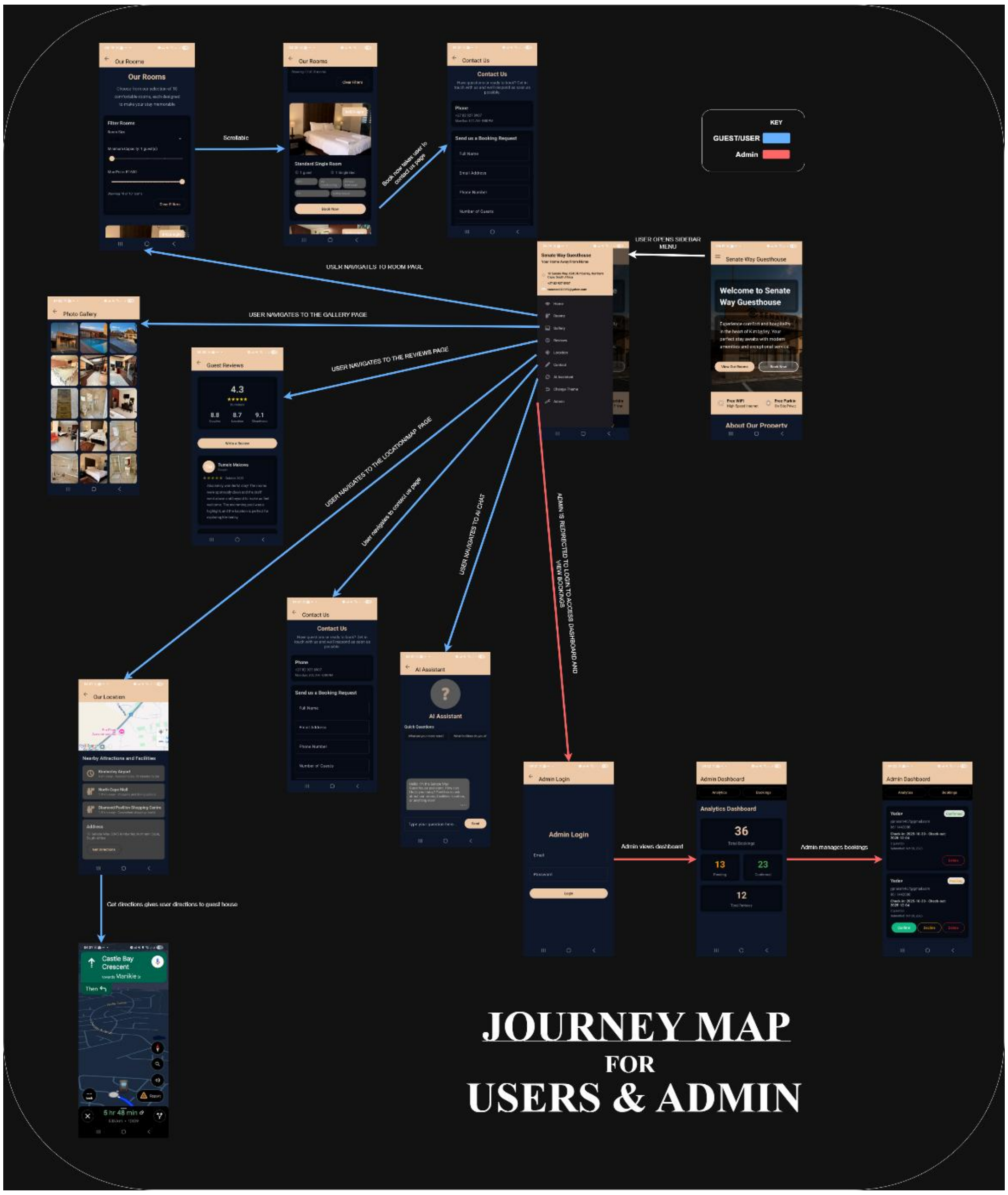

KEY
GUEST/USER 🟦
Admin 🟥
Scrollable
Book now takes user to contact us page
USER OPENS SIDEBAR MENU
USER NAVIGATES TO ROOM PAGE
USER NAVIGATES TO THE GALLERY PAGE
USER NAVIGATES TO THE REVIEWS PAGE
USER NAVIGATES TO THE LOCATION/MAP PAGE
User navigates to contact us page
USER NAVIGATES TO AI CHAT
ADMIN IS REDIRECTED TO LOGIN TO ACCESS DASHBOARD AND VIEW BOOKINGS
Admin views dashboard
Admin manages bookings
Get directions gives user directions to guest house
JOURNEY MAP
FOR
USERS & ADMIN

Figure 2: Mock-up Journey Map

The above image is a Journey Map of the Information Architecture and Navigation Flow for the Senate Way Guesthouse application.

Dual User Paths: Clearly defined application paths for two classes of users:

- **Guest/User** (Blue Paths): Focuses on browsing public information, such as Rooms, the Photo Gallery, Contact details/Map, AI ChatBot, Rooms and bookings and the Welcome Page.
- **Admin** (Red Path): Requires an Admin Login to access a secure area, primarily the Analytics Dashboard and the Admin Panel for managing guesthouse operations such as bookings

Its goal is to give developers and designers a visual guide as to what screens will be needed by the application, and how they will link together, and to what extent public users and admin will have access.

5. Understanding and Analysis

Bounded Contexts

While analysing the project domain several bounded contexts were elicited. A bounded context is the area of responsibility and the boundary around a model within the problem domain. These contexts separate concerns and can lead to a modular design that is easier to develop, maintain, and scale.

1. **User Management Context**
   - **Responsibility:** Handles all user-related operations such as registration, authentication, profile management, and user roles.
   - **Key Entities:** User, Role, Permissions.
   - **Purpose:** Ensures secure and structured management of users, aligning with the system's access control requirements.
2. **Booking Management Context**
   - **Responsibility:** Manages guesthouse bookings, availability checking, and reservations.
   - **Key Entities:** Booking, Room, Schedule.
   - **Purpose:** Provides a dedicated space for booking operations, ensuring correct reservation handling without affecting other system functionalities.
3. **Payment Context**
   - **Responsibility:** Handles payment processing, billing, and transaction history.
   - **Key Entities:** Payment, Invoice, Transaction.
   - **Purpose:** Encapsulates all financial operations, maintaining integrity and security of financial data.
4. **Notification Context**
   - **Responsibility:** Sends notifications to users about booking confirmations, payment receipts, and reminders.
   - **Key Entities:** Notification, Message Template, User Preferences.
   - **Purpose:** Provides a focused service for communication with users without mixing with core booking or user management logic.

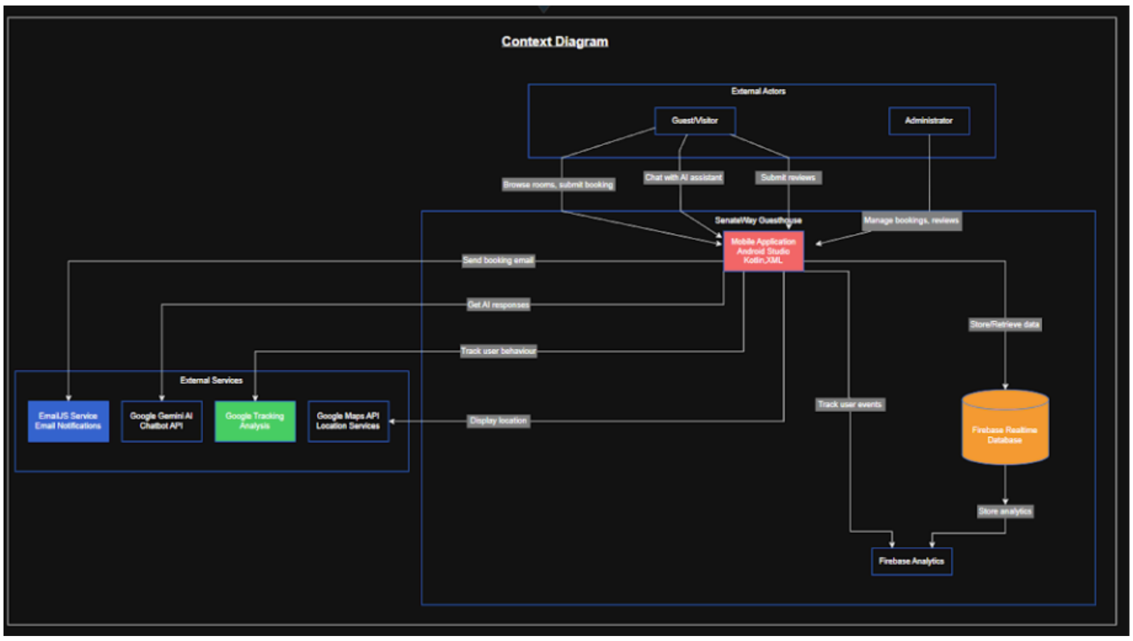


Figure 3: Context Diagram

6. Implementation Documentation

6.1 Sequence diagram

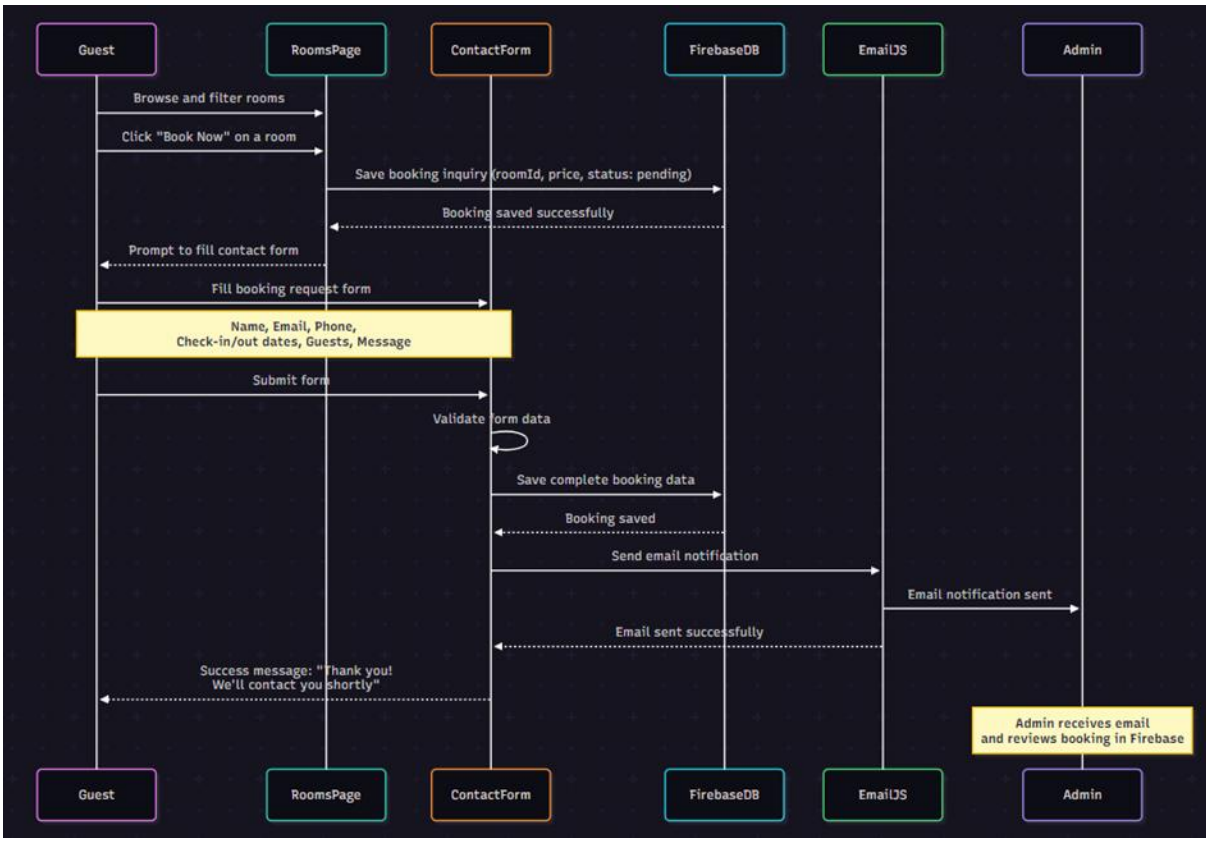

Guest
RoomsPage
ContactForm
FirebaseDB
EmailJS
Admin
Browse and filter rooms
Click "Book Now" on a room
Save booking inquiry (roomId, price, status: pending)
Booking saved successfully
Prompt to fill contact form
Fill booking request form
Name, Email, Phone, Check-in/out dates, Guests, Message
Submit form
Validate form data
Save complete booking data
Booking saved
Send email notification
Email notification sent
Email sent successfully
Success message: "Thank you! We'll contact you shortly"
Admin receives email and reviews booking in Firebase
Guest
RoomsPage
ContactForm
FirebaseDB
EmailJS
Admin

Figure 4: Sequence Diagram

The above flow diagram shows the steps that are followed upon an user (Guest) sending an inquiry for a room booking. The flow starts when an user lands on the RoomsPage, finds the room the user wants and clicks on "Book Now". This sends a request to the FirebaseDB to set up a "pending" booking request. The system then redirects the Guest to the ContactForm to fill in their personal details name, email, dates, and message. After a form is submitted, the following steps happen:

- Validation: The ContactForm first validates the submitted data to ensure accuracy and completeness.
- Data Persistence: The validated, complete booking information is then securely saved, updating the existing record in the FirebaseDB.
- Notification: The ContactForm triggers an email service (EmailJS) to send a notification to the Admin regarding the new booking request.
- Confirmation: Once the email is successfully sent, the Guest receives a final success message: "Thank you! We'll contact you shortly."

Afterward, the Admin receives a notification, which allows them to log into Firebase and check the booking to either approve or deny it.

6.2 State diagram

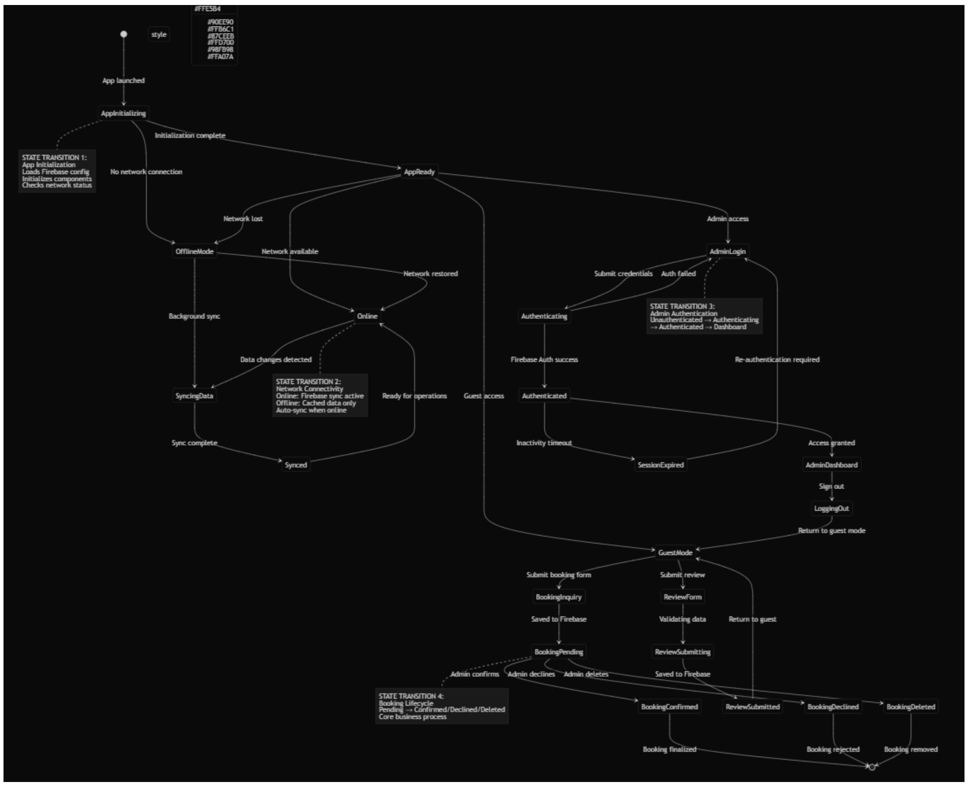


Figure 5: State Diagram

Explanation of the 4 Critical State Transitions:

The SenateWay Guesthouse application uses four state transitions to carry out actions, manage security, and improve the user experience.

1. **App Initialization States:**
   During the AppInitializing to GuestMode/AdminLogin steps, the Firebase system is initialized and the network status is grabbed. Then, the app config is handled. Depending on whether the user is allowed to use the app or not, they will then be taken either into guest mode or the login page.
2. **Network Connectivity States:**
   In the Online ↔ OfflineMode ↔ SyncingData → Synced state flow, the app toggles between online and offline modes, and may synchronize the data with Firebase upon a change in data while in the background to ensure data consistency.
3. **Admin Authentication States:**
   The flow AdminLogin → Authenticating → Authenticated → AdminDashboard → SessionExpired/LoggingOut manages the administration section of the application, including login, session handling, access permission, and the log out or timeout of admin users.
4. **Booking Lifecycle States:**
   The BookingInquiry -> BookingPending -> BookingConfirmed / BookingDeclined / BookingDeleted pattern represents the main booking flow within the system. It is used upon creation of a booking inquiry, processing, and the final status of the booking being saved in the system and email notifications sent to the guests.

Together, these four transitions cover the application's startup, network handling, admin security, and the guest booking process, ensuring reliable and secure operation throughout the app.

7.Understanding of Architecture

7.1 The application lives with a hybrid cloud architecture. It includes BaaS (Backend-as-a-Service) and RESTful APIs. The application uses Firebase for its primary backend technology and the application accesses bookings and reviews with Realtime Database, it authenticates admin access with Authentication, and it analyzes user behaviour with Analytics (Varia, 2008; Albini & Rajnai, 2018)..

The database URL is https://senateway-f2c37-default-rtdb.firebaseio.com. The project ID of the app is senateway-f2c37.

The third-party services used are EmailJS for sending emails, AccuWeather for acquiring weather data, and Google Gemini AI for chatbot functionality. The architecture follows a client-server architecture with the Android app directly communicating with the web services over HTTPS protocol.

Firebase SDK is responsible for maintaining Web Socket connections for real-time data and for synchronization, while data is also kept locally to allow the application to work in offline mode.

7.2 Cloud Architecture Considerations:

The Firebase Realtime-Database synchronizes in real-time because no one had to implement a server side. It also persists data and supports offline work in a built-in way, and this would ease development. The team focused on selecting the best solutions to problems. EmailJS, AccuWeather and Gemini were selected. For the HTTP request, the team chose the OkHttp client, explaining that it provides connection pooling, timeout management and an industry-standard, reliable API.

The singleton pattern used in FirebaseConfig allows a single instance of the database and authentication service to be accessed, allowing for quick development without sacrificing scalability or maintainability using managed cloud services.

7.3 Cloud Networking:

Firebase SDK makes use of WebSocket connections for its networking, taking care of reconnections and synchronization. The communication is split across three layers. OkHttp client is used for calls to EmailJS and the AccuWeather API. It uses 10 second connection and reads timeouts, provides reliable HTTP operations, and handles TLS/SSL automatically. HttpsURLConnection is used for Google Gemini API calls.

HTTPS over port 443 is used for all requests to external servers. The Firebase Realtime Database URL is resolved into Google Cloud Platform server infrastructure and third-party APIs are contacted via their cloud endpoints. Network requests are asynchronous and non-blocking using Kotlin coroutines.

7.4 HTTPS/TLS 1.2+ is the transport protocol used for all external API traffic, which is encrypted end-to-end to protect it from eavesdropping or man-in-the-middle attacks.

JSON (JavaScript Object Notation) is the data serialization format used in all API requests and responses, a human-readable and lightweight format. Application/json is the constant value for the Content-Type request header of every POST request.

Firebase Realtime Database utilizes the WebSocket Secure protocol for transmission and synchronization of data bidirectionally in real-time. Firebase Realtime Database does this to avoid polling.

TCP/IP protocols provide reliable packet delivery within the network layer. HTTPS is used for protection of sensitive data in transit, such as API keys or booking and user credentials. JSON was chosen for its popularity as well as being easily parsed into Kotlin data classes, without executable code like XML.

System and Cloud Architecture Diagram

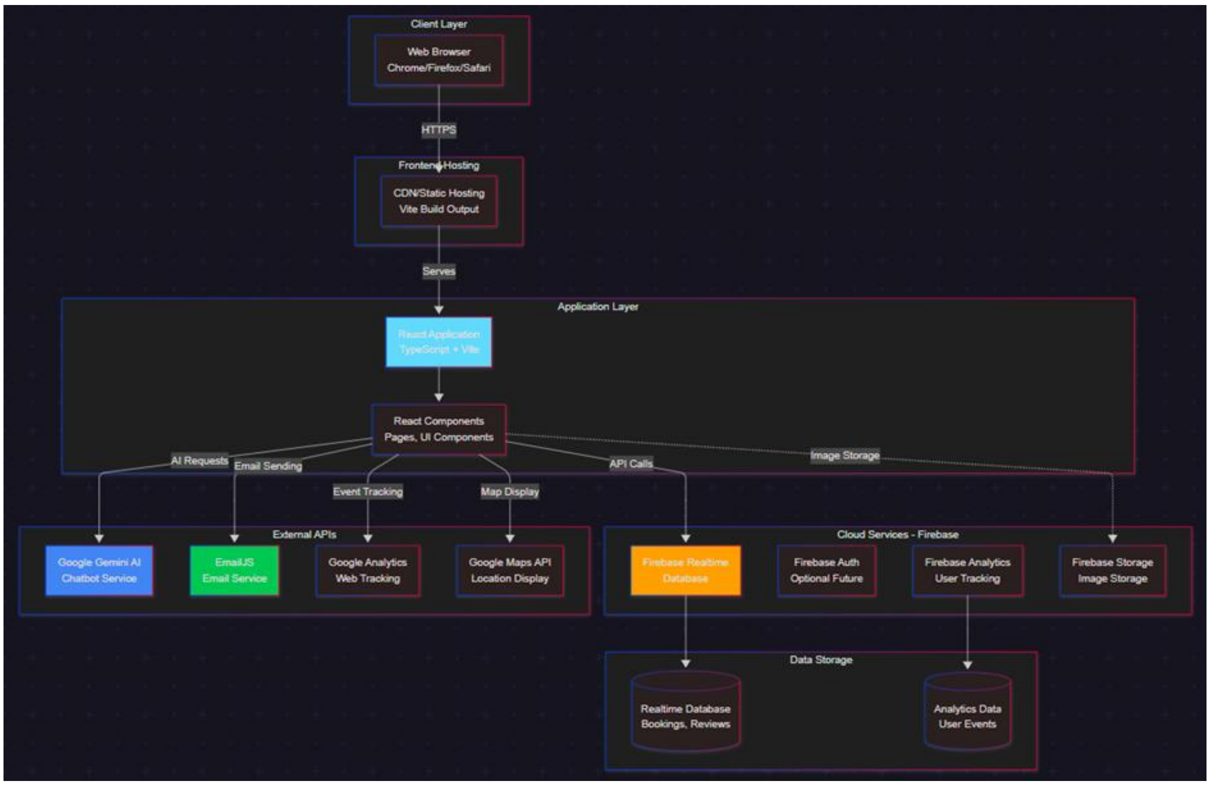

Client Layer
Web Browser Chrome/Firefox/Safari
HTTPS
Frontend Hosting
CDN/Static Hosting Vite Build Output
Serves
Application Layer
React Application TypeScript + Vite
React Components Pages, UI Components
AI Requests
Email Sending
Event Tracking
Map Display
API Calls
Image Storage
External APIs
Google Gemini AI Chatbot Service
EmailJS Email Service
Google Analytics Web Tracking
Google Maps API Location Display
Cloud Services - Firebase
Firebase Realtime Database
Firebase Auth Optional Future
Firebase Analytics User Tracking
Firebase Storage Image Storage
Data Storage
Realtime Database Bookings, Reviews
Analytics Data User Events

Figure 6: System and Cloud Architecture Diagram

Figure 5 illustrates how the entire system operates, from the user’s browser or mobile app, through the React frontend, to external APIs and Firebase storage

8. Design and Architecture Patterns

8.1 Documented Data Structures and Well-Known Algorithms:

Several data structures are used, including mutableListOf<Message> for chat messages, listOf<Room> for rooms, and listOf<String> for amenities. On Room lists, the filters included the capacity filter { it.capacity >= 2 } and price filter { it.price in minPrice..maxPrice }.

HashMap<String, Any> dictionary class is used in working with Firebase to send booking details as a List of key-value pairs with keys such as "name", "email", "checkIn".

The JSONObject class is an object used for the POX API POST requests to EmailJS and Gemini. RecyclerView's setHasStableIds(true) uses sets implicitly. Sets stop multiple views from forming for one piece of data.

Sets are used in Firebase push IDs. For example, the recursive function getAllChildren recursively traverses a ViewGroup hierarchy applying calendar styling, showing tree traversal. The data structures in use are well-chosen lists, for ordered collections maps, for key-value pairs and sets, for uniqueness. The code is clean in its design, "Consistent documentation ensures reproducibility and transparency (Slaughter et al., 2015).". It uses meaningful names. It uses strict typing.

8.2 Inter-service communication between activities is achieved using Kotlin data classes. Parcelable-compatible immutable data classes are WeatherData, Room, GalleryImage, Message and Booking which can be used between activities. WeatherData uses data classes (TemperatureUnit, WindData, and MetricValue) inside a data class to parse the unstructured AccuWeather API response.

Booking data is represented as HashMap<String, Any> to be serialized by Firebase in AdminDashboardActivity. The Room entity uses a List<String> to model composite data structures. The GalleryImage implements Parcelable interface to ease Activity-to-Activity data transfer via Intent extras.

All classes following data class conventions with properties declared as val to ensure immutability, default property values set, and meaningful naming. Code is well-read and the package is organized as data/model. These classes are directly mapped to the domain models, so that storage, transfer, and user interface are consistent.

8.3 It applies multiple design patterns. For instance, the Singleton Pattern applies to the FirebaseConfig object, which lazily holds the Firebase instance, with only one database and authentication instance existing throughout the application lifecycle.

The Adapter pattern is implemented within the GalleryAdapter, RoomsAdapter, ChatbotAdapter, and ReviewsAdapter classes to convert lists of data into ViewHolder-based user interfaces. The Observer pattern is implemented via LiveData and ValueEventListener. HomeViewModel exposes MutableLiveData<Float> for rating modification.

Firebase listeners utilize addValueEventListener to observe real-time database changes. MVVM Architecture separates business logic and UI through ViewModel (HomeViewModel). LiveData observation within activities maintains separation of concerns and testability. The Repository Pattern with FirebaseConfig serves as the repository layer between the UI and the database. All patterns are implemented properly and follow Android best practices with separated concerns.

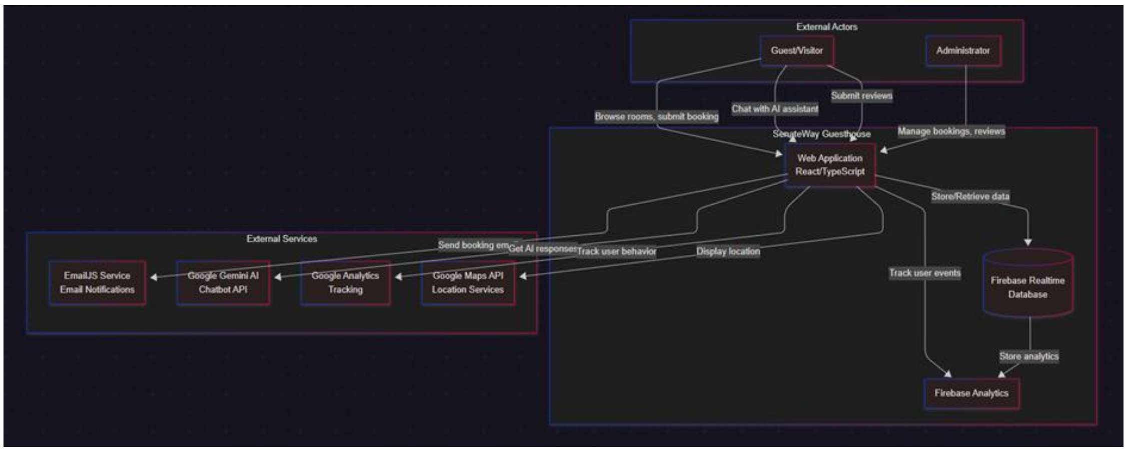


Figure 7: Domain Model

8.4 For frontend-backend communication, RESTful APIs are used following industry best practices. The EmailJS and AccuWeather APIs are called from the frontend using the OkHttp client. JSON serializes with Content-Type in application/json through JSONObject plus JSONArray.

The Firebase Realtime Database API relies upon WebSockets to provide real-time synchronization. ValueEventListener callbacks use the observer pattern. They give reactive data changes. FirebaseConfig implements the repository pattern to abstract data access.

The ViewModel pattern (HomeViewModel) mediates between the UI and data sources. LiveData objects are exposed to observe data changes. Activities observe for the LiveData instead of querying directly, in an observer pattern. All network calls are done using coroutines to avoid blocking the UI thread and try-catch statements together with onFailure callbacks are used. The communication layer separates the UI, business logic, and data access layers and utilizes the Android Architecture Components framework for this purpose.

8.5 The application integrates through documentation with multiple third-party APIs (application programming interfaces). The Google Maps SDK integration uses the OnMapReadyCallback interface pattern. The GoogleMap instance allows one to access location services and map rendering. AccuWeather API is executed via REST with GET requests.

The JSON response is parsed into a nested structure of data classes WeatherData. Google Gemini AI API is invoked via HttpsURLConnection with POST requests, sending prompt data in JSON. EmailJS service integration is done by sending email requests in JSON format as POST requests using OkHttp with template parameters. All APIs are served over HTTPS/TLS upon port 443.

API keys are stored in strings.xml. This storage should have been a move to BuildConfig for production. Geolocation is handled with the Google Maps SDK, guesthouse coordinates are -28.7674381, 24.7497489. There is good error handling including timeouts (10 seconds), response codes, and fallbacks.

The integrations are also well done. Error handling, timeout handling, and separation of concerns are achieved through service classes and utility functions.

9.Data schemas

9.1 Data Storage Description

The project uses a relational database to store all necessary data, which allows for data integrity and protection against data corruption, as well as for ease of querying and reading. Each type of data that the project uses is stored in a different entity (table) and is usually associated with a domain model. The relationship between entities is established via primary and foreign keys to maintain referential integrity.

For example, the main entities in the system include:

- **User**: Stores information about system users such as name, email, and login credentials.
- **Booking**: Contains details of guest bookings including dates, room selection, and status.
- **Room**: Represents available rooms with details such as type, price, and capacity.
- **Payment**: Records transactions made by users, including payment method and amount.

Each entity is designed to minimize redundancy while ensuring that all required information is stored efficiently.

10. Digital law and ethics

10.1 Declaration

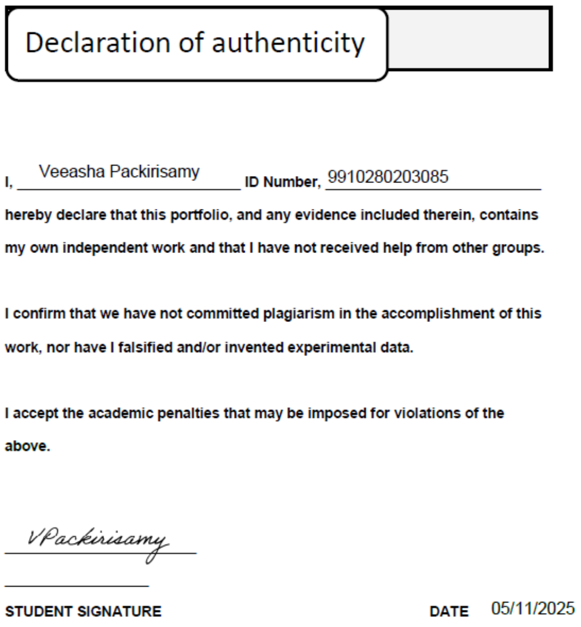
We, the undersigned, hereby declare that the work submitted in this Portfolio of Evidence (PoE) is our own original work and that all sources of information and help that we have used in completing this project have been properly acknowledged and referenced.

This PoE and the accompanying reports, documents, diagrams, source code, and designs were all created during the Work Integrated Learning (WIL) Module (INSY7315) as part of the DIMEXL Guesthouse System project our team did during the module. This project was done in an academic context, simulating the workplace, and presents our ability to apply software engineering concepts, Agile methodologies, and technical knowledge.

We confirm that this assignment has not been copied or plagiarized and has not been submitted for any other module or assessment. This applies to all members of the group who made a meaningful contribution to planning, developing, testing and documenting.

We understand that any plagiarism or academic dishonesty will render us liable to disciplinary action in accordance with the IIE's academic integrity policy.

Veeasha Packirisamy, ST10397833



Declaration of authenticity

I, Veeasha Packirisamy ID Number, 9910280203085

hereby declare that this portfolio, and any evidence included therein, contains my own independent work and that I have not received help from other groups.

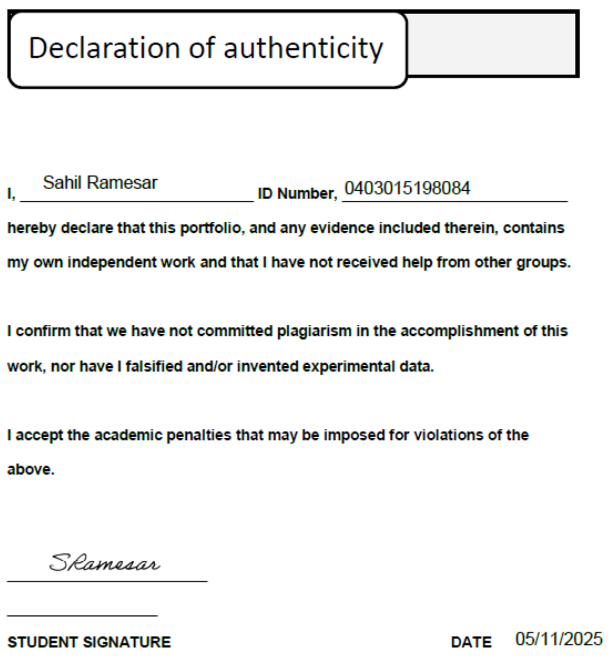
I confirm that we have not committed plagiarism in the accomplishment of this work, nor have I falsified and/or invented experimental data.

I accept the academic penalties that may be imposed for violations of the above.

VPackirisamy

STUDENT SIGNATURE DATE 05/11/2025

Sahil Ramesar, ST10356407



Declaration of authenticity

I, Sahil Ramesar ID Number, 0403015198084

hereby declare that this portfolio, and any evidence included therein, contains my own independent work and that I have not received help from other groups.

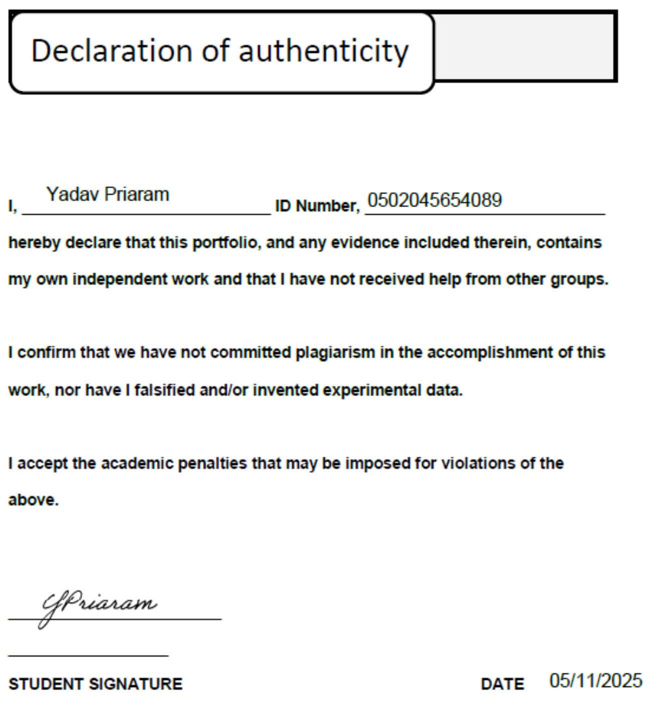
I confirm that we have not committed plagiarism in the accomplishment of this work, nor have I falsified and/or invented experimental data.

I accept the academic penalties that may be imposed for violations of the above.

SRamesar

STUDENT SIGNATURE DATE 05/11/2025

Yadav Priaram, ST10356506



Declaration of authenticity

I, Yadav Priaram ID Number, 0502045654089

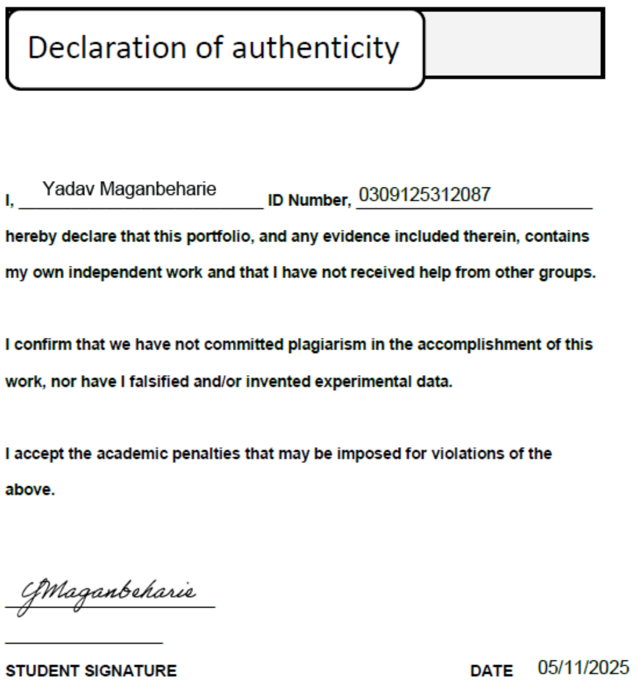
hereby declare that this portfolio, and any evidence included therein, contains my own independent work and that I have not received help from other groups.

I confirm that we have not committed plagiarism in the accomplishment of this work, nor have I falsified and/or invented experimental data.

I accept the academic penalties that may be imposed for violations of the above.

YPriaram

STUDENT SIGNATURE DATE 05/11/2025



Declaration of authenticity

I, Yadav Maganbeharie ID Number, 0309125312087

hereby declare that this portfolio, and any evidence included therein, contains my own independent work and that I have not received help from other groups.

I confirm that we have not committed plagiarism in the accomplishment of this work, nor have I falsified and/or invented experimental data.

I accept the academic penalties that may be imposed for violations of the above.

YMaganbeharie

STUDENT SIGNATURE DATE 05/11/2025

10.2 Software license

The source code of the DIMEXL Guesthouse website and application will be licensed under a Custom Proprietary License for the exclusive use of the DIMEXL Guesthouse and will not be distributed or modified outside of the DIMEXL business.

Justification:
The proprietary license gives DIMEXL Guesthouse exclusive rights to the system and prevents any other company or individual from copying, using, or altering the system. It also protects the brand, data and system. Although the developer holds the intellectual property rights, DIMEXL has full usage rights over their guesthouse.

What It Means to Us:
For us as developers, it means we control the intellectual property. DIMEXL Guesthouse's operations are characterized by exclusivity and discretion.

11. Security

The area of security was a key factor in the design of the DIMEXL Guesthouse mobile application and website, because of the need to protect user data, administrative information, and the overall functionality of the system (Graff & Van Wyk, 2003). The application collected sensitive information such as guest contact details, booking queries, and admin credentials, requiring strong security protocols.

11.1 Potential Threat Actors

Threat actors are individuals or entities that may attempt to exploit vulnerabilities in the system. For the DIMEXL Guesthouse system, possible threat actors include:

- **External Hackers:** Attempting to gain unauthorized access to the website or database for data theft or sabotage.
- **Malicious Users:** Guests or outsiders who submit harmful scripts or malicious inputs through forms (e.g., SQL injection).
- **Internal Threats:** Employees or former staff misusing admin credentials or sensitive information.
- **Automated Bots:** Web crawlers or spam bots attempting to overload the system with fake inquiries or login attempts.

Understanding these actors allows the development team to design appropriate security controls.

11.2 Potential Threat Vectors

Threat vectors are the paths through which attacks may occur. For this project, key vectors include:

- **Unsecured Input Fields:** Forms that could be exploited for SQL injection or cross-site scripting (XSS).
- **Unencrypted Communication:** Data transmitted without SSL/TLS encryption could be intercepted.
- **Weak Authentication:** Simple or reused passwords could allow unauthorized logins.
- **Insecure APIs:** Improperly configured Firebase or backend APIs may expose sensitive data.
- **Third-Party Dependencies:** External libraries or plugins may introduce vulnerabilities if not updated.

These vectors inform the system's mitigation strategies and continuous monitoring approach.

11.3 Mitigations for Threats

Mitigations were designed using the **principle of economy of mechanism**, meaning security mechanisms remain simple, efficient, and easy to maintain.

Threat	**Mitigation Strategy**	**Economy of Mechanism Applied**
SQL Injection & XSS	Use of **input whitelisting** and **parameterized queries** in all form submissions.	Simple regex validation and built-in sanitization minimize complexity.
Unauthorized Access	**Hashed and salted passwords** using Firebase Authentication.	Uses native Firebase security features to reduce custom logic.
Data Interception	**HTTPS (SSL/TLS)** enforced for all traffic.	Centralized SSL setup reduces redundancy.
Brute Force / Bots	**Rate limiting** and **CAPTCHA** on forms.	Lightweight CAPTCHA prevents overload without impacting user experience.
Insider Threats	**Role-based access control (RBAC)** for admin and user levels.	Simple permission tiers limit access while staying easy to manage.

These mitigations balance simplicity with effectiveness, ensuring strong protection without unnecessary system complexity.

11.4 Balancing Security with Usability

Security and usability can be hard to balance. If security is excessive, the controls will be avoided. If too few, they will not be effective.

For the DIMEXL Guesthouse system:

- The **login process** for admin users includes secure authentication but uses **Firebase's built-in login**, ensuring speed and familiarity.
- **Guests** can browse, inquire, and contact the guesthouse without mandatory registration, preserving usability.
- **Form validation** prevents malicious input but provides clear, user-friendly error messages.
- **SSL encryption** operates in the background, maintaining data security without affecting user experience.

This balance ensures security mechanisms protect users transparently, without frustrating or deterring engagement.

11.5 Security in Data Access (Complete Mediation)

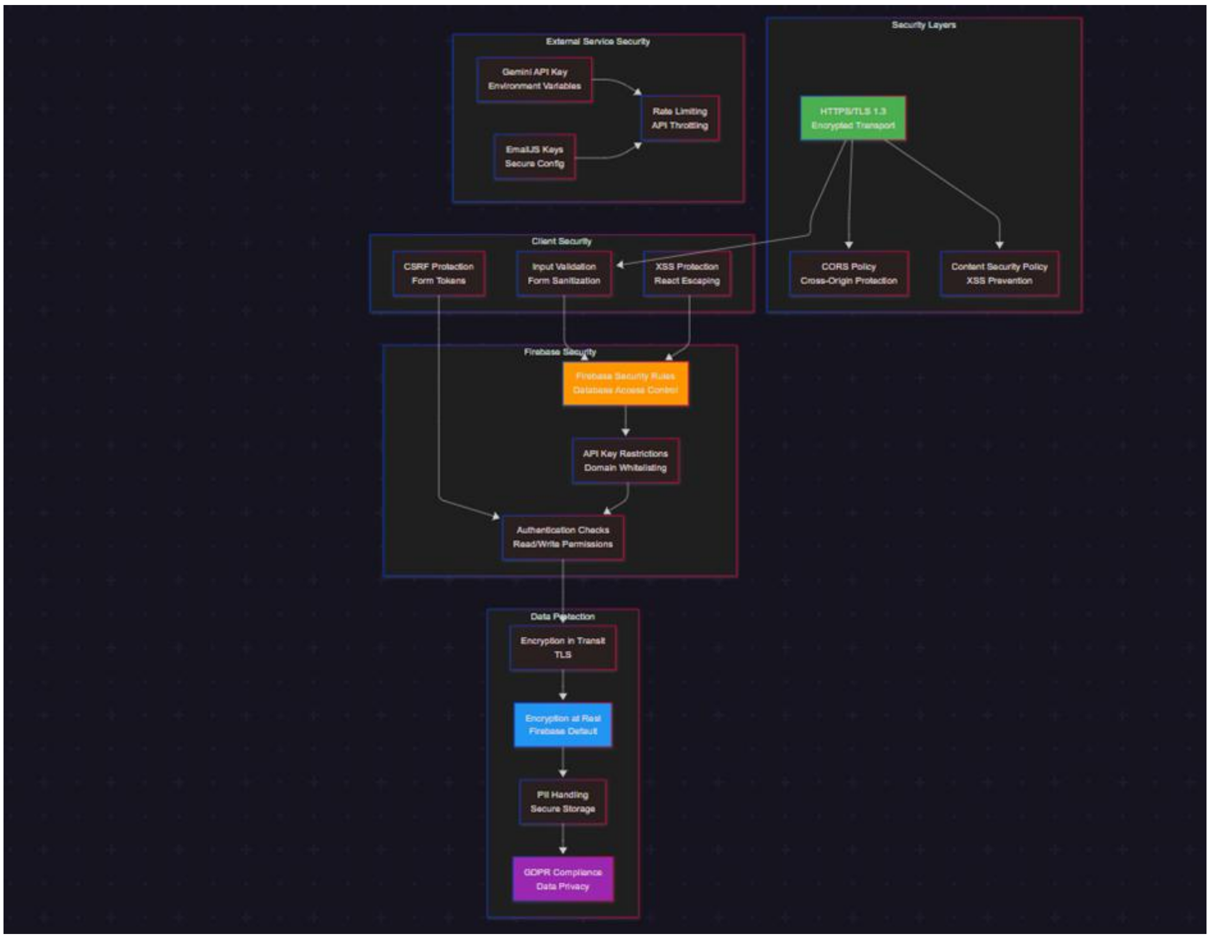
Complete mediation ensures every access to a data object (e.g., booking inquiry, user message, or image upload) is verified before being granted.

In the DIMEXL system:

- All databases read/write operations go through Firebase Authentication and Rules, ensuring that only authorized users (e.g., admin) can modify data.
- Guest data submissions (inquiry forms) are verified, sanitized, and validated before being stored.
- Admin actions (like updating images or room details) are authenticated at each request — even after login — to prevent session hijacking.
- No data object is accessed directly from the client side; all operations are mediated through secure API calls.

This ensures consistent enforcement of access controls and minimizes the risk of unauthorized data exposure or manipulation.

Security Architecture Diagram



External Service Security
Gemini API Key Environment Variables
Rate Limiting API Throttling
EmailJS Keys Secure Config
Security Layers
HTTPS/TLS 1.3 Encrypted Transport
Client Security
CSRF Protection Form Tokens
Input Validation Form Sanitization
XSS Protection React Escaping
CORS Policy Cross-Origin Protection
Content Security Policy XSS Prevention
Firebase Security
Firebase Security Rules Database Access Control
API Key Restrictions Domain Whitelisting
Authentication Checks Read/Write Permissions
Data Protection
Encryption in Transit TLS
Encryption at Rest Firebase Default
PII Handling Secure Storage
GDPR Compliance Data Privacy

Figure 8: Security Architecture Diagram

In Figure 7 it explains how the applications protect data to the internal and external, using layered security.

Operations

12. DevOps

12.1 Description of the use of DevOps in the project is included DevOps in the Senate Way Guesthouse Android Project

The Senate Way Guesthouse Android app integrates DevOps to automate builds, testing, deployment, and ensure code quality and security (Thota, 2020).

1. Version Control and Collaboration:
Git with GitHub handles version control using feature branches with pull requests, a main branch for stable code, and specific commit conventions.

2. Build Automation:
Git with GitHub is used for version control, employs feature branches and pull requests, has a stable branch within, and follows specific conventions for commit messages.

3. Continuous Integration (CI):
Automated GitHub Actions builds have static code analysis with Android Lint, ktlint, and Detekt, dependency scanning with Dependabot, unit and UI testing with JUnit and Espresso, and package build artifacts for APK/AAB.

4. Continuous Deployment (CD):
Internally, it is tested with Firebase App Distribution and through beta channels. Automated publishing to the Google Play Store tracks happens in stages. Firebase backend is configured using the Firebase console.

5. Environment Management:
Debug/release builds, emulator suites, and API keys secured for each environment separate development, staging, and production environments.

6. Code Quality and Security:
Static analysis, code review, obfuscation via ProGuard/R8, Firebase Security Rules application, and HTTPS enforcement keep the code base high-quality and secure.

7. Monitoring and Analytics:
Firebase Analytics is used to analyze user interactions. CI/CD pipelines are used to monitor builds.

8. Infrastructure as Code:
Gradle build scripts, dependency versions, and Firebase configuration files are all versioned to ensure reproducibility.

9. Automated Testing Strategy:
Unit, integration and UI tests cover both critical booking submission and admin dashboard paths, and API/database actions.

10. Deployment Automation:
CI/CD pipelines automate testing, versioning, artifact generation, uploading to the Play Console, and generating release notes.

Benefits: Releases happen fast. Builds are repeatable. Source code is of good quality. Code is secure. People collaborate. Monitoring is iterative. Infrastructure is scalable.

Tools & Technologies: Git, GitHub, Gradle, Android Gradle Plugin, GitHub Actions, JUnit, Espresso, Android Lint, ktlint, Detekt, Dependabot, Snyk, Firebase including Realtime DB, Auth, Analytics, Crashlytics, Google Play Console.

A DevOps toolchain that enables the development, safe deployment and quality operation of the Senate Way Guesthouse Android app.

12.2 CI/CD Pipeline Diagram

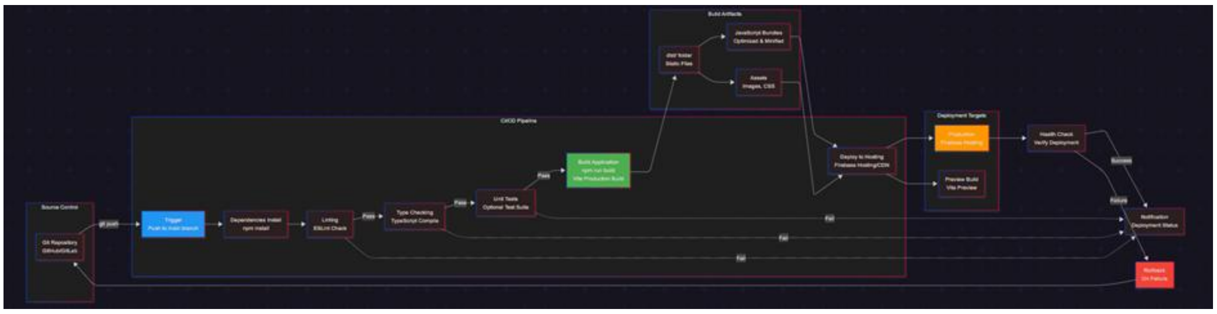
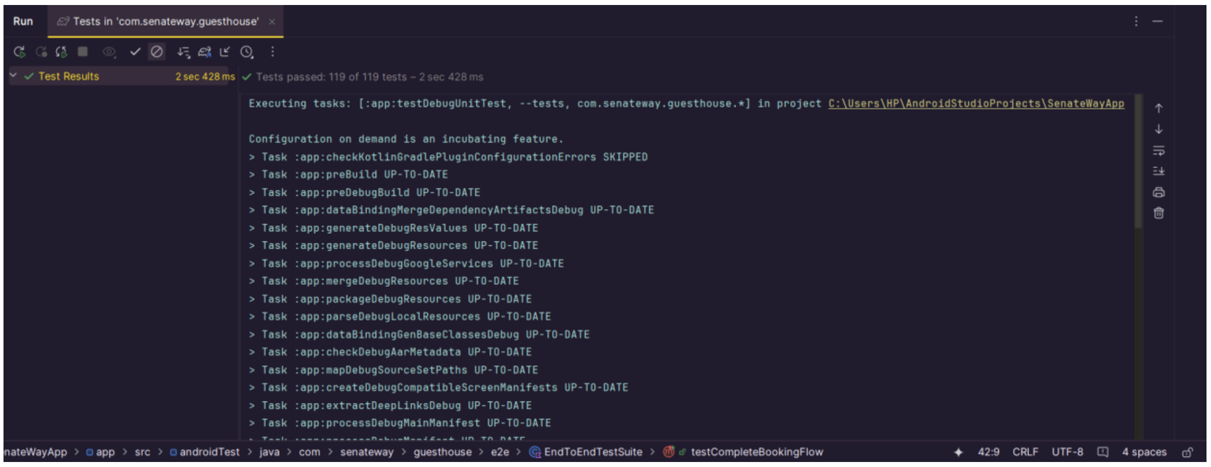

Source Control
Git Repository GitHub/GitLab
git push
CI/CD Pipeline
Trigger Push to main branch
Dependencies Install npm install
Linting ESLint Check
Pass
Type Checking TypeScript Compile
Pass
Unit Tests Optional Test Suite
Pass
Build Application npm run build Vite Production Build
Build Artifacts
dist/ folder Static Files
JavaScript Bundles Optimized & Minified
Assets Images, CSS
Deploy to Hosting Firebase Hosting/CDN
Deployment Targets
Production Firebase Hosting
Preview Build Vite Preview
Health Check Verify Deployment
Success
Failure
Fail
Fail
Fail
Notification Deployment Status
Rollback On Failure

Figure 9: CI/CD Pipline Diagram

12.3 Unit Test Report (Pipeline)



```
✓ Tests passed: 119 of 119 tests – 2 sec 428 ms

Executing tasks: [:app:testDebugUnitTest, --tests, com.senateway.guesthouse.*] in project C:\Users\HP\AndroidStudioProjects\SenateWayApp

Configuration on demand is an incubating feature.
> Task :app:checkKotlinGradlePluginConfigurationErrors SKIPPED
> Task :app:preBuild UP-TO-DATE
> Task :app:preDebugBuild UP-TO-DATE
> Task :app:dataBindingMergeDependencyArtifactsDebug UP-TO-DATE
> Task :app:generateDebugResValues UP-TO-DATE
> Task :app:generateDebugResources UP-TO-DATE
> Task :app:processDebugGoogleServices UP-TO-DATE
> Task :app:mergeDebugResources UP-TO-DATE
> Task :app:packageDebugResources UP-TO-DATE
> Task :app:parseDebugLocalResources UP-TO-DATE
> Task :app:dataBindingGenBaseClassesDebug UP-TO-DATE
> Task :app:checkDebugAarMetadata UP-TO-DATE
> Task :app:mapDebugSourceSetPaths UP-TO-DATE
> Task :app:createDebugCompatibleScreenManifests UP-TO-DATE
> Task :app:extractDeepLinksDebug UP-TO-DATE
> Task :app:processDebugMainManifest UP-TO-DATE
```

nateWayApp > app > src > androidTest > java > com > senateway > guesthouse > e2e > EndToEndTestSuite > testCompleteBookingFlow
42:9 CRLF UTF-8 4 spaces

Figure 10: Unit Test Report

12.4 Sonar Qube

https://sonarcloud.io/project/overview?id=senateway-guesthouse

12.5 Snyk Unit Test

https://app.snyk.io/org/st10356506/project/e6659bc1-987c-40d8-b9c7-ee3308de162a/history/24115976-ef7b-4f1f-bab3-737c5ef37cd0

13. Running Costs and Predicted Growth

The running costs of the DIMEXL Guesthouse Web and Mobile Application are based on sustainable hosting and scalable backend solutions. The current architecture uses Netlify (Free Plan) for hosting, Firebase for the real-time database and authentication, and Google Maps & EmailJS APIs for integrations. These services provide free or low-cost tiers initially but can scale automatically with increased usage (Berrocal et al., 2020).

The following sections include:

- Documented user growth predictions,
- Defined scaling points for each technology,
- Predictive cost models (best, mean, and worst cases), and
- Analysis of alternative technologies for future scalability.

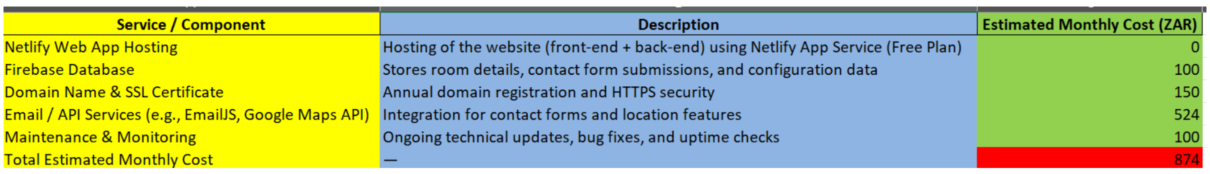
13.1 Predicted User Growth

The projected user base is based on an expected increase as the guesthouse gains visibility through its website and mobile app.

Period	Estimated Monthly Users	Description
Month 1–6	100–250	Initial launch phase; mostly local customers using mobile app and website.
Month 7–12	250–500	Gradual increase through word-of-mouth and local tourism exposure.
Month 13–18	500–800	New returning users and online visitors from SEO & social marketing.
Month 19–24	800–1200	Increased seasonal bookings and repeat customers.

13.2 Scaling Points by Technology

Technology / Service	Scaling Threshold	Impact of Scaling	Current Plan
Netlify Hosting	>10,000 monthly page views	Upgrade to paid plan for higher bandwidth and build minutes	Currently on Free Tier
Firebase (Database & Auth)	>5 GB storage / >50k reads per month	Move to Firebase Blaze (Pay-as-you-go) plan	Currently on Free Plan
Google Maps API	>2,500 API calls/day	API costs will increase per 1,000 requests	Basic API usage
EmailJS / Email API	>200 emails/month	Move to paid tier for increased monthly sends	Free Plan
Monitoring & Maintenance	Increased error reports or uptime issues	Manual checks and automated alerts	Managed internally



Service / Component	Description	Estimated Monthly Cost (ZAR)
Netlify Web App Hosting	Hosting of the website (front-end + back-end) using Netlify App Service (Free Plan)	0
Firebase Database	Stores room details, contact form submissions, and configuration data	100
Domain Name & SSL Certificate	Annual domain registration and HTTPS security	150
Email / API Services (e.g., EmailJS, Google Maps API)	Integration for contact forms and location features	524
Maintenance & Monitoring	Ongoing technical updates, bug fixes, and uptime checks	100
Total Estimated Monthly Cost	—	874

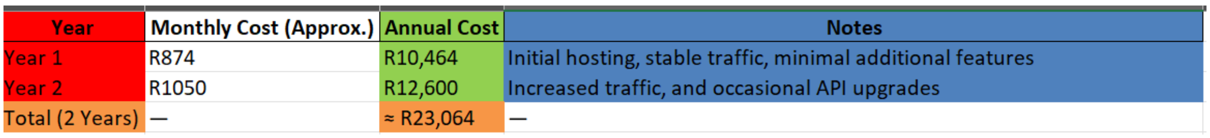
These scaling thresholds are estimated based on Firebase and Netlify documentation and user benchmarks (Berrocal et al., 2020).

13.3 Predictive Cost Models (Monthly Basis)

Scenario	Description	Estimated Monthly Cost (ZAR)	Notes
Best Case (Low Growth)	Traffic grows slowly; services remain mostly on free tiers	**R750–R900**	Minimal API usage, stable user base
Mean Case (Expected Growth)	10% growth every 6 months; slight increase in Firebase & API use	**R1000–R1100**	Matches your Year 2 projection
Worst Case (High Growth)	Heavy traffic from tourism seasons; paid Firebase & API tiers	**R1300–R1600**	Upgrades to paid hosting & API limits exceeded

13.4 Two-Year Projection Summary

The two-year cost projection assumes moderate growth in site traffic and storage needs—approximately 10% increase every six months due to more users, higher image uploads, and potential new features such as a booking system



Year	Monthly Cost (Approx.)	Annual Cost	Notes
Year 1	R874	R10,464	Initial hosting, stable traffic, minimal additional features
Year 2	R1050	R12,600	Increased traffic, and occasional API upgrades
Total (2 Years)	—	≈ R23,064	—

Figure 11: Cost Calculation for Years

The projected total cost of approximately R23,064 over two years is considered cost-effective for maintaining a reliable, secure, and scalable online presence for DIMEXL Senate Way Guesthouse.

Included Benefits:

- **High Availability:** Global CDN provided by Netlify ensures fast and reliable access for users (Berrocal, J., Garcia-Alonso, J., Fernandez, P., Perez-Vereda, A., Hernandez, J., Canal, C., Murillo, J.M. and Ruiz-Cortes, A., 2020.).
- **Secure Hosting:** Automatic SSL certificates, HTTPS enforcement, and Firebase security rules protect user data.
- **Scalability:** Hosting and backend services can scale with traffic growth and new features, including a future booking system.
- **Professional Monitoring and Maintenance:** Continuous deployment via GitHub, automated build pipelines, and Firebase monitoring ensure optimal performance.

These measures ensure that the guesthouse website remains accessible, trustworthy, and capable of handling growing customer engagement without compromising performance or security.

13.5 Technology Scaling and Alternatives

Over time, as the user base grows beyond the thresholds of free or basic plans, the following alternatives may be more cost-effective and scalable:

Current Technology	Alternative Option	When to Consider	Benefits
Firebase Database	AWS DynamoDB or Supabase	When data exceeds 5 GB or 100k reads/month	More control over data, predictable cost scaling
Netlify Hosting	AWS Amplify or Vercel	If traffic surpasses 10k monthly page views	Automatic scaling with integrated CI/CD
EmailJS API	SendGrid / AWS SES	When email traffic exceeds 200 emails/month	High deliverability, API reliability

13.6 Summary

Thus, the DIMEXL Guesthouse system is continually kept cost-efficient and scalable, focusing on users' needs regarding security and privacy. In the worst-case scenario, this amount is less than R1,600 per month, which is affordable given the growing number of establishments, especially with online booking systems. The reason for utilizing Firebase, Netlify, and cloud APIs is to scale the features or infrastructure without having to redeploy or redesign the code.

The table below shows the estimated monthly cost to host and maintain the DIMEXL Guesthouse website on Netlify and Firebase. If there is a mobile app, it will be hosted on the PlayStore. These costs included the cost of web hosting, domain name registration, storage, API calls, and maintenance over two years, considering their growing use and feature additions (Berrocal et al., 2020).

14. Change Management

In order to achieve this, change management needs to be implemented to successfully deploy the change from the current booking channel, whether they be manual or third-party channels like Booking.com, or simple email inquiries to the DIMEXL Guesthouse Web and Mobile Application. Management, staff, and guests need to be confident and comfortable with the application, without there being a substantial impact to operations (Lauer, T., 2010).

This addresses both organizational adoption (by management and employees) and user adoption (by guests), by promoting the system's smooth integration into business processes and improving guest satisfaction as part of the total guest experience.

14.1 Organisational Adoption and Integration into Operations

The implementation of this system for DIMEXL Guesthouse will allow this organisation and company to digitalise the daily operations and processes through the use of the mobile app and website. This will allow the organisation managers & employees to monitor, update and interact with their clients in real time. "Formal onboarding practices, as Frögéli, Jenner and Gustavsson (2023) note, improve employee adaptation during digital transitions."

Key Benefits for the Organisation:

- **Centralised Operations:**
  All booking details, guest inquiries, and contact information are synchronised through Firebase, ensuring that data is up to date across both the mobile app and website.
- **Efficiency:**
  Admin users can log in from either a desktop or mobile device to manage room listings, gallery images, and guest messages without technical assistance.
- **Reduced Costs:**
  Direct bookings through the DIMEXL platform minimise third-party commission fees (e.g., Booking.com), allowing greater control over pricing and availability.
- **Scalability:**
  The Firebase backend allows seamless expansion when the guesthouse grows or adds more properties.

What It Means for the Organisation:
The system replaces the previous manual, paper-based coordination of the operation of the DIMEXL Guesthouse and provides an efficient, transparent, and reliable automated solution that can be accessed from any device and that is sustainable over the long term.

14.2 User Adoption (Guests and Staff)

For Staff and Management:

Employee adoption is kept to a minimum because both the mobile app and the associated admin portal are designed to be intuitive and responsive, so the owner's admin staff can handle most changes without needing external technical support.

Adoption Strategies:

- Conduct **brief on-site demonstrations** of both the website and mobile app functions.
- Provide **simple training materials** with screenshots for tasks like editing room details or uploading photos.
- Implement a **"train-the-owner" model**, where the guesthouse owner can teach new staff as needed.

For Guests:

Guest adoption is anticipated to be rapid, as the application is easy to use and available on mobile devices. The DIMEXL mobile application provides a responsive experience for browsing room details and images, along with a direct contact method to the guesthouse, all on the guest's mobile device.

Guest Experience Improvements:

- Quick and direct contact through mobile-friendly forms.
- Integrated Google Maps navigation to the guesthouse.
- Smooth browsing experience with optimised loading on mobile devices.

What It Means for Guests:
With such additional services, guests can communicate directly with the guesthouse, surf with their smartphone, and be answered more quickly so that customer satisfaction is improved.

14.3 Adoption Strategy

To successfully gain adoption from both the **organisation** and **users**, a multi-step rollout and engagement plan will be implemented:

Phase	**Description**	**Purpose**
1. Awareness & Introduction	Introduce staff and guests to the benefits of the app through meetings, printed guides, and signage.	Build understanding and excitement.
2. Training & Onboarding	Conduct hands-on demonstrations for the owner and admin staff. Provide digital and printed manuals.	Build confidence and reduce resistance.
3. Pilot Testing (Soft Launch)	Release the app and website internally before public launch. Collect feedback and resolve usability issues.	Ensure readiness and stability.
4. Full Deployment	Officially launch the app to the public with social media promotion.	Encourage guest engagement and usage.
5. Continuous Improvement	Gather ongoing feedback from staff and guests to improve functionality.	Maintain adoption and satisfaction.

This strategy encourages both organisational ownership and user trust, ensuring that adoption is natural and sustained over time.

14.4 Feedback Collection and Continuous Improvement

To maintain long-term engagement and system reliability, feedback and performance monitoring will be continuous (Fabijan et al., 2015).

Feedback Mechanisms:

- **Staff feedback sessions** every quarter to identify operational improvements.
- **Guest feedback forms and quick surveys** on the website and app.
- **Google Analytics and Firebase Analytics** to measure engagement, load times, and user activity.
- **Post-deployment support channel** for reporting technical issues directly to the developer.

This ensures the system evolves alongside the guesthouse's needs, maintaining performance and relevance as technology and user expectations change.

14.5 Maintenance and Support Strategy

Maintaining and supporting the DIMEXL Guesthouse system is essential for continued functionality, reliability, and user trust.

Support Structure:

- **Monthly Monitoring:** Regular uptime checks, error monitoring, and database backups.
- **Quarterly Updates:** Apply security patches, SDK updates (Android, Firebase), and performance optimisations.
- **Version Control:** All updates are managed via GitHub, with rollback options for safety.
- **Technical Support:** Direct email or WhatsApp contact between the guesthouse owner and developer for urgent fixes.

Long-Term Plan:
If the guesthouse expands or internal staff gain technical skills, maintenance can gradually be transferred in-house, supported by the provided documentation and cloud-based administrative tools.

What It Means for DIMEXL:
This ensures that the system remains **sustainable, reliable, and self-sufficient**, supporting both mobile and web platforms with minimal external dependency.

References

(Asproni, G., 2006. An introduction to scrum. *Software Developer's Journal*, *6*(1), pp.1-10.)

(Fabijan, A., Olsson, H.H. and Bosch, J., 2015, June. Customer feedback and data collection techniques in software R&D: a literature review. In *International Conference of Software Business* (pp. 139-153). Cham: Springer International Publishing.)

(Lauer, T., 2010. *Change management* (pp. 3-1). Springer Berlin Heidelberg.).

B. and Gustavsson, P., 2023. Effectiveness of formal onboarding for facilitating organizational socialization: A systematic review. *PloS one*, *18*(2), p.e0281823.).

(Berrocal, J., Garcia-Alonso, J., Fernandez, P., Perez-Vereda, A., Hernandez, J., Canal, C., Murillo, J.M. and Ruiz-Cortes, A., 2020. Early evaluation of mobile applications' resource consumption and operating costs. *IEEE Access*, *8*, pp.146648-146665.).

(Thota, R.C., 2020. CI/CD Pipeline Optimization: Enhancing Deployment Speed and Reliability with AI and Github Actions. *International Journal of Innovative Research in Engineering & Multidisciplinary Physical Sciences*, *8*, pp.1-11.).

(Staunton, C., Tschigg, K. and Sherman, G., 2021. Data protection, data management, and data sharing: Stakeholder perspectives on the protection of personal health information in South Africa. *PLoS One*, *16*(12), p.e0260341.).

(Luk, M., Mezzour, G., Perrig, A. and Gligor, V., 2007, April. MiniSec: a secure sensor network communication architecture. In *Proceedings of the 6th international conference on Information processing in sensor networks* (pp. 479-488).

(Kim, H. and Lee, E.A., 2017. Authentication and Authorization for the Internet of Things. *IT Professional*, *19*(5), pp.27-33.)

(Graff, M. and Van Wyk, K.R., 2003. *Secure coding: principles and practices*. " O'Reilly Media, Inc.".)

(Varia, J., 2008. Cloud architectures. *White Paper of Amazon, jineshvaria. s3. amazonaws. com/public/cloudarchitectures-varia. pdf*, *16*.)

(Slaughter, S.E., Hill, J.N. and Snelgrove-Clarke, E., 2015. What is the extent and quality of documentation and reporting of fidelity to implementation strategies: a scoping review. *Implementation Science*, *10*(1), p.129.)

(Albini, A. and Rajnai, Z., 2018. General architecture of cloud. Procedia Manufacturing, 22, pp.485-490.)

(Hassine, J., 2015. Describing and assessing availability requirements in the early stages of system development. *Software & Systems Modeling*, *14*(4), pp.1455-1479.).

(Duboc, L., Letier, E., Rosenblum, D.S. and Wicks, T., 2008, September. A case study in eliciting scalability requirements. In *2008 16th IEEE International Requirements Engineering Conference* (pp. 247-252). IEEE.)

(Juristo, N., 2006. Impact of usability on software requirements and design. In *International Summer School on Software Engineering* (pp. 55-77). Berlin, Heidelberg: Springer Berlin Heidelberg.)

(Almogahed, A., Omar, M. and Zakaria, N.H., 2022. Refactoring codes to improve software security requirements. *Procedia Computer Science*, *204*, pp.108-115.)

(Glinz, M., 2007, October. On non-functional requirements. In *15th IEEE international requirements engineering conference (RE 2007)* (pp. 21-26). IEEE.)

(Lucassen, G., Dalpiaz, F., Werf, J.M.E.V.D. and Brinkkemper, S., 2016, March. The use and effectiveness of user stories in practice. In *International working conference on requirements engineering: Foundation for software quality* (pp. 205-222). Cham: Springer International Publishing.).

(Kujala, S., Kauppinen, M., Lehtola, L. and Kojo, T., 2005, August. The role of user involvement in requirements quality and project success. In *13th IEEE International Conference on Requirements Engineering (RE'05)* (pp. 75-84). IEEE.)

(Kappel, T.A., 2001. Perspectives on roadmaps: how organizations talk about the future. *Journal of Product Innovation Management: AN INTERNATIONAL PUBLICATION OF THE PRODUCT DEVELOPMENT & MANAGEMENT ASSOCIATION*, *18*(1), pp.39-50.)

(Dalton, J., 2018. Definition of Done. In *Great Big Agile: An OS for Agile Leaders* (pp. 159-161). Berkeley, CA: Apress.)**.**

Power, K., 2014, May. Definition of ready: An experience report from teams at cisco. In *International Conference on Agile Software Development* (pp. 312-319). Cham: Springer International Publishing.).