Project - High Level Design

On

Hospitality App

Course Name: DevOps Foundation

Institution Name: Medicaps University – Datagami Skill Based Course

Student Name(s) & Enrolment Number(s):

Sr no	Student Name	Enrolment Number
01	Sumit Nigade	EN22CS301999
02	Vedang Rajoriya	EN22CS3011066
03	Suryapratap Sisodiya	EN22CS3011004
04	Vandana Patel	EN22CS3011057

Group Name: Group 08 D9

Project Number: DO-08

Industry Mentor Name: Mr. Vaibhav Sir

University Mentor Name: Prof. Dr. Ritesh Joshi

Academic Year: 2026

Table of Contents

1. Introduction

1.1 Scope of the Document

This document provides a detailed high-level design (HLD) of the Hospitality Booking Web Application. It outlines the system architecture, design principles, component breakdown, data model, interface communication, and non-functional aspects of the application.

1.2 Intended Audience

- Technical architects and developers
- Project stakeholders
- Course instructors and evaluators
- DevOps and deployment teams

1.3 System Overview

The Hospitality Booking Web Application is a responsive and dynamic system that allows users to explore and book hotel rooms. It enables authentication, booking, and management functionalities for both users and admins. The application follows modern web standards using:

- **Frontend**: React + TypeScript
- **Backend**: Supabase Functions + API integration
- **Authentication & DB**: Supabase (PostgreSQL + Auth)
- **Deployment**: Docker, GitHub Actions, Kubernetes (k3s on AWS EC2)

2. System Design

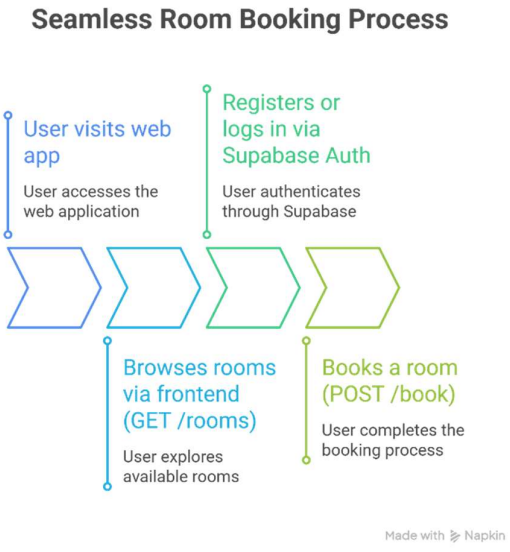
2.1 Application Design

The application is built with a modular architecture. Each component is responsible for a single function and interacts through APIs or shared state.

- **Frontend**: React app built using Vite. Includes user-facing components (Room Listings, Booking Forms, Login) and admin-facing dashboard.
- **Backend**: Supabase Edge Functions and APIs serve data and business logic.
- **Database**: Supabase PostgreSQL manages persistent data.
- **DevOps**: Docker for containerization, GitHub Actions for CI/CD, Kubernetes for orchestration

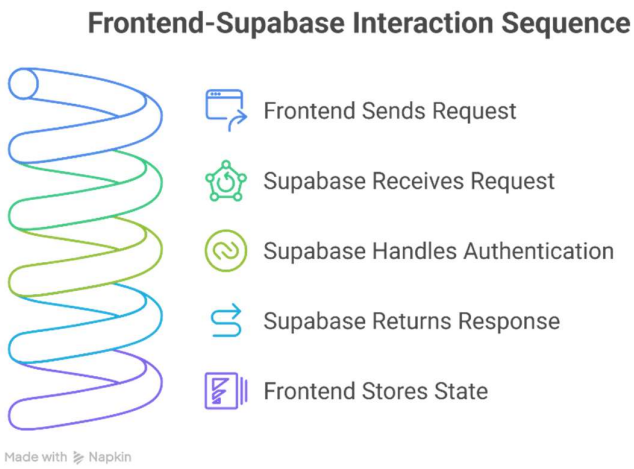
2.1 Architecture Diagram

2.2 Process Flow



2.3 Information Flow

- Frontend sends requests to Supabase (via client SDK or RESTful endpoints)
- Supabase returns responses and handles authentication
- State is stored in frontend using React Context and Query libraries



2.4 Components Design

Component	Description
Header/Footer	Common layout
RoomCard	Displays room preview
BookingForm	Allows date and guest selection
AuthContext	Maintains auth state
AdminDashboard	Admin controls for room management

2.5 Key Design Considerations

- Clean separation of logic between client and cloud backend
- Scalable deployment using Kubernetes and Docker
- Responsive UI for mobile and desktop
- Secure access via Supabase JWT sessions

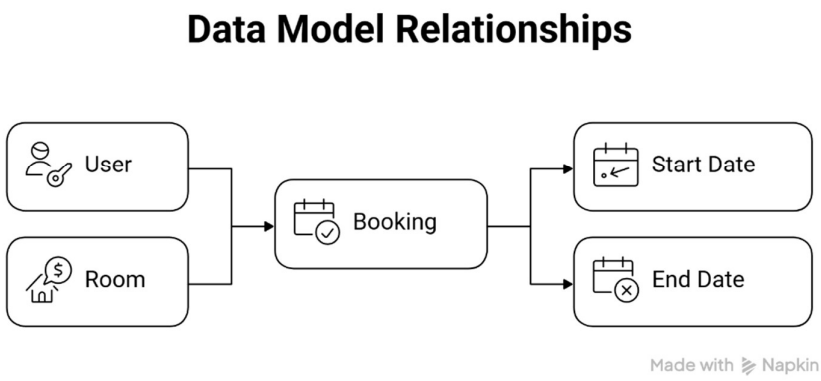
2.6 API Catalogue

Endpoint	Method	Purpose
/rooms	GET	List available rooms
/book	POST	Book a selected room
/bookings	GET	User booking history
/admin/rooms	POST	Add or update rooms
/auth	POST	User authentication

3. Data Design

3.1 Data Model

User(id, email, password, role)
Room(id, title, price, location, features)
Booking(id, user_id, room_id, start_date, end_date)



3.2 Data Access Mechanism

- Supabase client from frontend handles all DB access securely
- Supabase Role-Based Access Control (RBAC) ensures data visibility

3.3 Data Retention Policies

- Booking data retained indefinitely unless deleted
- Logs auto-purged after 6 months unless otherwise configured

3.4 Data Migration

- Initial schema created using Supabase SQL editor
- Changes managed via Supabase migrations or CLI

4. Interfaces

- **User Interface**: Web UI built in React
- **Admin Interface**: Admin dashboard protected via role checks
- **API Interface**: RESTful interaction with Supabase Edge Functions and SDK

5. State and Session Management

- Auth sessions maintained by Supabase using JWT tokens
- React Context API for state propagation
- React Query and Cache for stateful components

6. Caching

- Supabase offers smart caching with policies
- React Query maintains frontend cache with refetch logic
- Static content cached via browser and CDN if used

7. Non-Functional Requirements

7.1 Security Aspects

- JWT-based Supabase Auth
- HTTPS enforced across deployments
- Access control via Supabase Policies and Role logic

7.2 Performance Aspects

- Supabase optimized for PostgreSQL speed
- React optimizations: lazy loading, bundling
- Kubernetes autoscaling for load handling

8. References

1. ReactJS Documentation: https://react.dev
2. Supabase Docs: https://supabase.com/docs
3. Kubernetes Docs: https://kubernetes.io/docs/
4. GitHub Actions CI/CD: https://docs.github.com/en/actions
5. Docker Guide: https://docs.docker.com/
6. Vite JS: https://vitejs.dev
7. Tailwind CSS: https://tailwindcss.com