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	Abhishek Gupta	EN22EL301004
02	Tanish Israni	EN22IT301112
03	Sourabh Patel	EN22IT301106
04	Samarth Patidar	EN22IT301087
05	Yash Dashore	EN22CS3011100
06	Ramakant Singh	EN22IT301077

Group Name: Group 10 D12

Project Number: DO-48

Industry Mentor Name: Mr. Vaibhav Sir

University Mentor Name: Prof. Akshay Saxena

Academic Year: 2026

Table of Contents

1. Introduction.

1.1. Scope of the document - This document provides the High-Level Design (HLD) for the Hospitality Application deployed using containerization and Kubernetes orchestration.

It covers: - Application architecture - Kubernetes deployment design - CI/CD pipeline integration - Rolling update strategy - Service exposure via LoadBalancer and Ingress - Non-functional requirements – Security and performance considerations.

1.2. Intended Audience - DevOps Engineers - Cloud Engineers - Kubernetes Administrators - Technical Review Panel - Development Team - System Architects.

1.3. System overview - The Hospitality Application is a containerized web application built using FastAPI (Python).

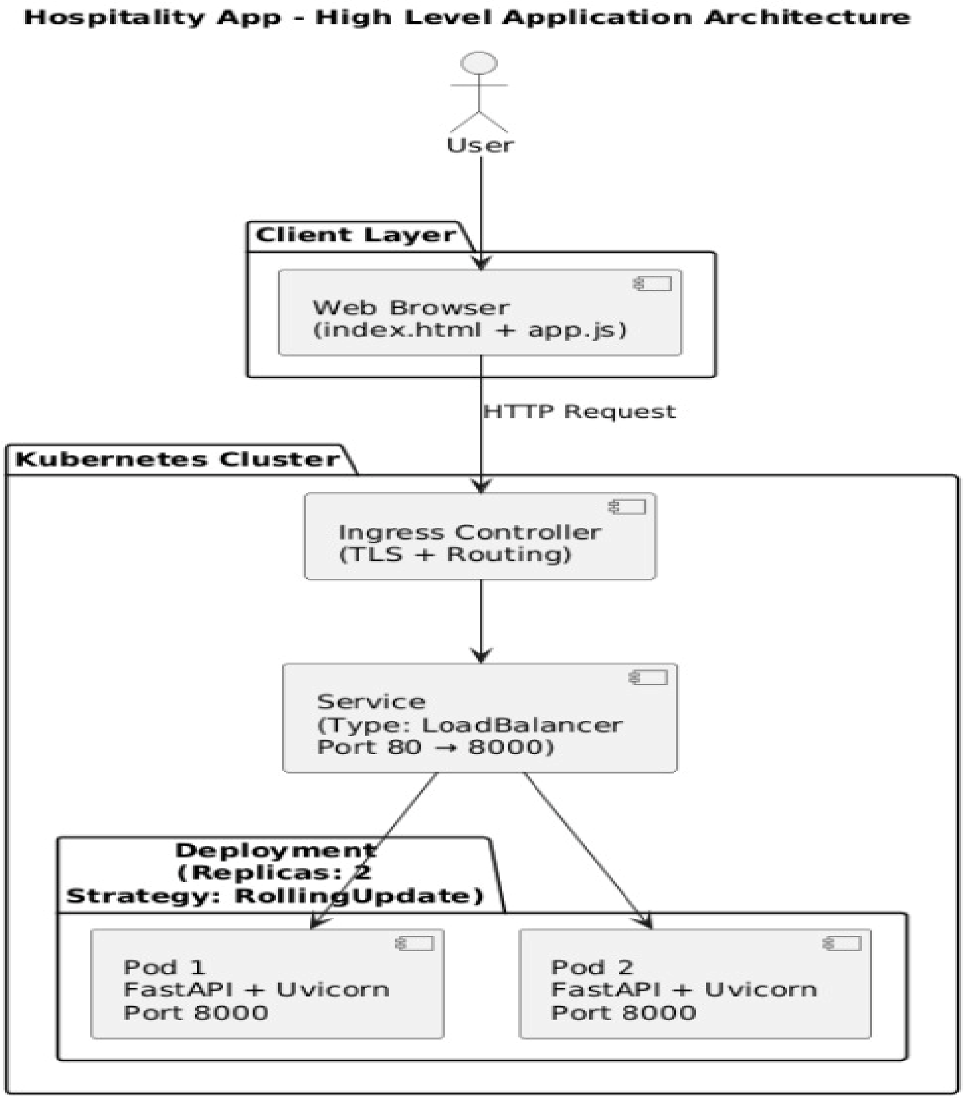
1.4. The system provides: - Static frontend (HTML, CSS, JS) - REST APIs: - /healthz - /availability - /book

The primary objective of the system is: - Containerize the application using Docker - Deploy it to Kubernetes (Minikube / AWS cluster) - Implement zero-downtime rolling updates - Automate CI/CD using GitHub Actions.

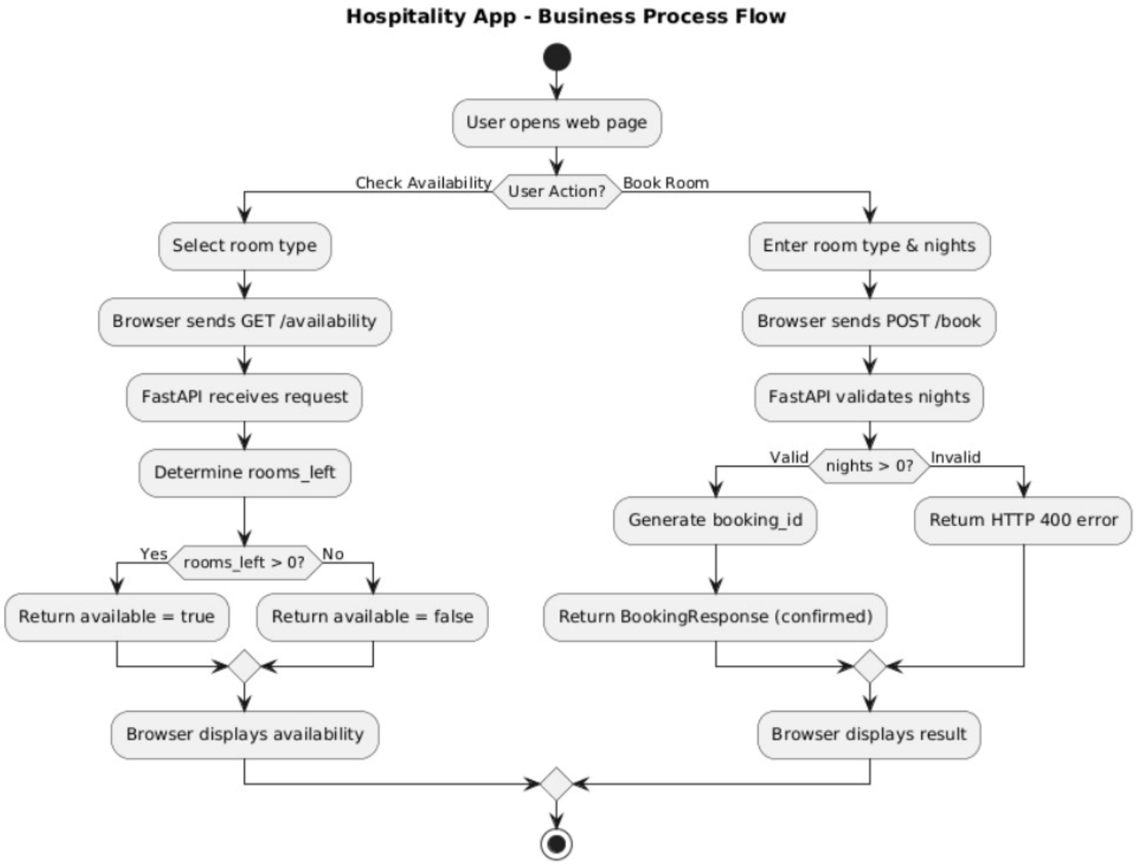
The application is stateless and does not use any external database.

2. System Design.

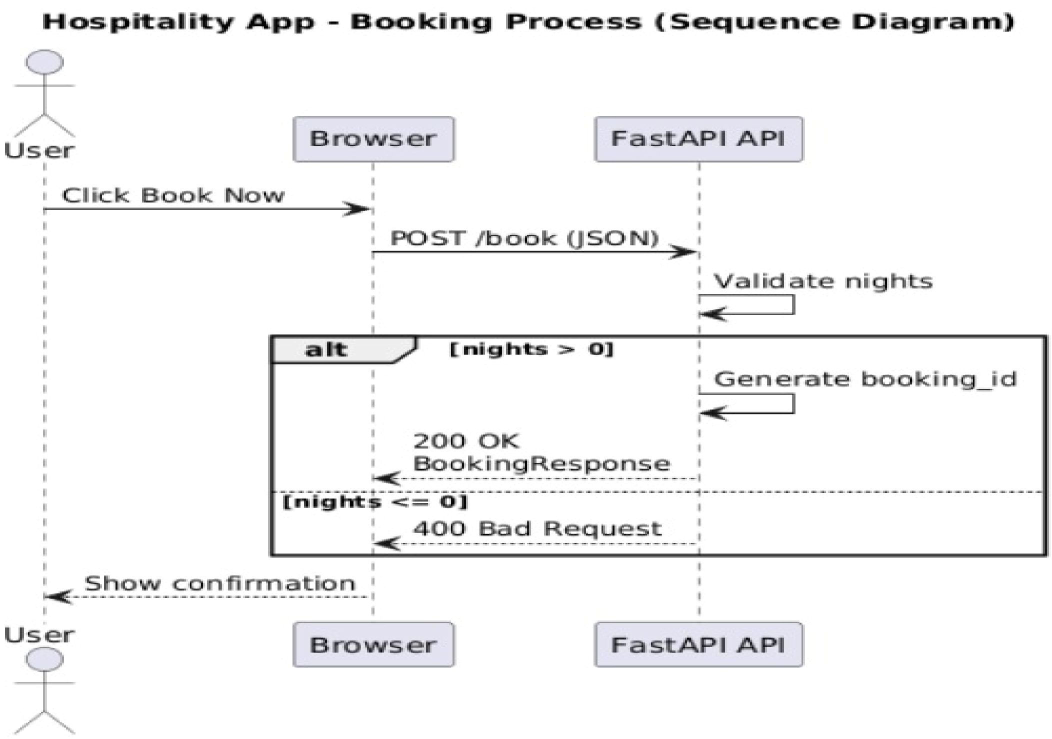
2.1. Application Design –



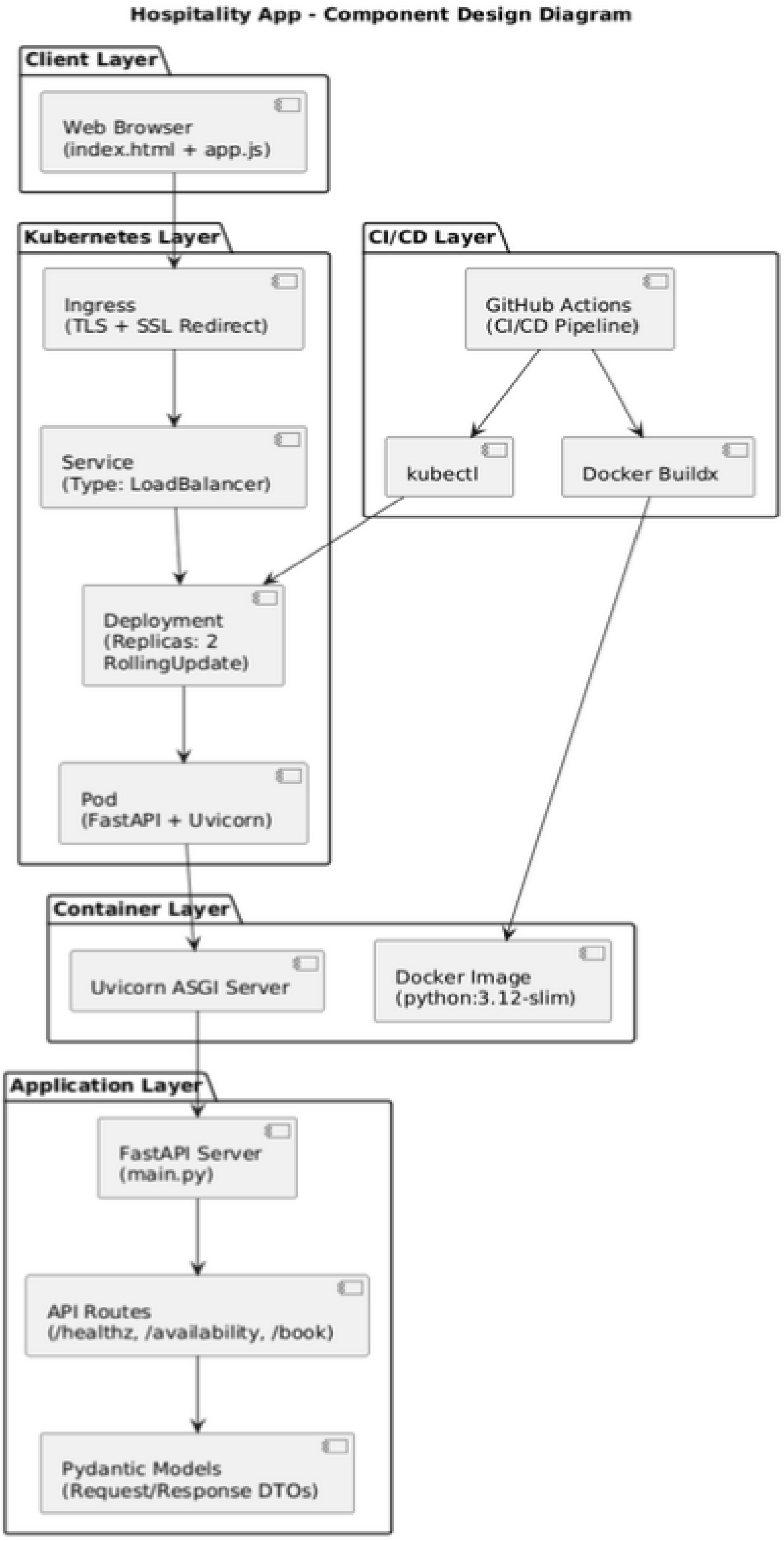
2.2. Process Flow –



2.3. Information Flow –



2.4. Components Design –



Application Layer: - FastAPI server - CORS enabled - Static file serving enabled - JSON-based REST APIs

Container Layer: - Dockerfile builds application image - Image tagged using commit SHA in CI/CD-Image pushed to container registry.
Kubernetes Layer: Deployment Configuration: - replicas: 2 -RollingUpdate strategy - maxSurge: 1 –

maxUnavailable: 0 - revisionHistoryLimit: 3
Service Configuration: - Type: LoadBalancer - Port 80 → TargetPort 8000
Ingress Configuration: - TLS enabled - SSL redirect enabled – Backend service routing
CI/CD Layer: GitHub Actions Workflow: Trigger: - Push to main branch

Pipeline Steps:
1. Checkout code
2. Setup Docker Buildx
3. Build Docker image
4. Save image artifact
5. Configure kubectl
6. Apply Kubernetes manifests
7. Update deployment image

Deployment Image Update Command: kubectl set image deployment/hospitality-api hospitality-api= <image-name>:<tag>

2.5. Key Design Considerations –

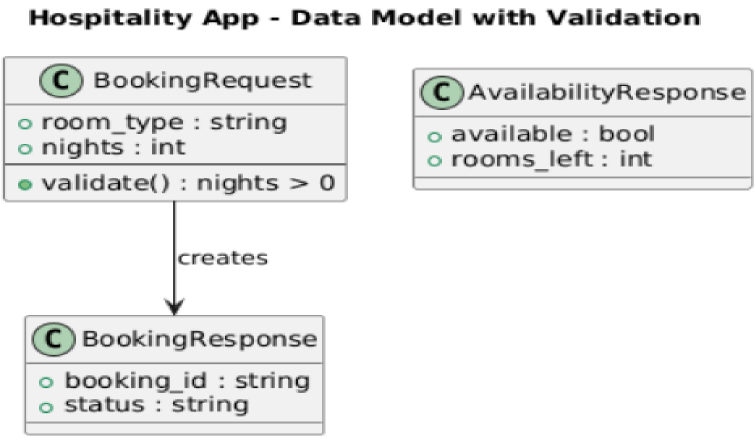
1. Zero Downtime:
- RollingUpdate strategy maxUnavailable: 0
- Readiness probe ensures traffic only to ready Pods
2. High Availability:
- Minimum 2 replicas
- Auto-restart of unhealthy Pods
3. Stateless Architecture:
- No session persistence
- Horizontal scaling supported
4. Environment Agnostic:
- Works on Minikube and AWS Kubernetes cluster

2.6. API Catalogue –

METHOD	**ENDPOINT**	**DESCRIPTION**	**REQUEST**	**RESPONSE**
GET	/healthz	Health Check	None	{status:"ok"}
GET	/availability	Check room availability	Query:room_type	{available,rooms_left}
POST	/book	Book a room	{room_type,nights}	{booking, status}

3. Data Design.

3.1. Data Model – The application uses DTO-based models defined using Pydantic.



3.2. Data Access Mechanism –
In-memory processing only
No database interaction
No external API dependency

3.3. Data Retention Policies –
No data persistence
No booking records stored
Stateless processing only

3.4. Data Migration - Not applicable currently (no database).

4. Interfaces –

4.1. External Interfaces –
REST APIs exposed over HTTP/HTTPS
JSON request/response
Swagger UI documentation available

4.2. Internal Interfaces –
Ingress to Service routing
Service to Pod communication
Kubernetes health probes

5. State and Session Management –
Stateless architecture.
No session storage
No cookies
Each request handled independently

6. Non-Functional Requirements

6.1. Security Aspects –
TLS termination at Ingress
Input validation via Pydantic

Kubernetes namespace isolation
Secrets used for Kubeconfig
Container isolation via Docker

6.2. Performance Aspects –
FastAPI ASGI-based asynchronous framework
Uvicorn high-performance server
Horizontal scaling with replicas
Load Balancing via Service
Rolling updates prevent downtime

7. References –

FastAPI Official Documentation
Docker Documentation
Kubernetes Documentation
GitHub Actions Documentation
CNCF Cloud Native Architecture Guidelines