Project - High Level Design

on

Dockerized Entertainment React Service

Course Name: DevOPS Function

Institution Name: Medicaps University – Datagami Skill Based Course

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

Sr no	Student Name	Enrolment Number
1	Yuvraj Jaiswal	EN22CS303060
2	Divyansh Parmar	EN22CS306017
3	Suyash Jain	EN22CS3011005
4	Bhumika Gwala	EN22CS304021
5	Rohit Marmat	EN22CS303045
6	Aayush Yadav	EN22CS303002

Group Name: Group 08D11

Project Number: DO-34

Industry Mentor Name : Vaibhav Sir

University Mentor Name : Prof. Shyam Patel

Academic Year: 2026

Table of Contents

1. Introduction

The Entertainment Service Web Application is a full-stack web platform designed to provide entertainment-related services such as user registration, authentication, content browsing, and booking management. The system follows a microservice-ready architecture using FastAPI for backend services, MySQL as the relational database, Docker for containerization, and a modern frontend framework for user interaction.

The objective of this project is to implement a scalable, secure, and containerized application aligned with DevOps best practices.

1.1 Scope of the Document

This document defines the high-level design (HLD) of the Entertainment Service Web Application. It covers :

- System architecture
- Component design
- API structure
- Database design
- Security and performance considerations
- Deployment approach using Docker

It does not include low-level implementation details.

1.2 Intended Audience

This document is intended for :

- Developers
- DevOps Engineers
- Academic Evaluators

This document is intended for:

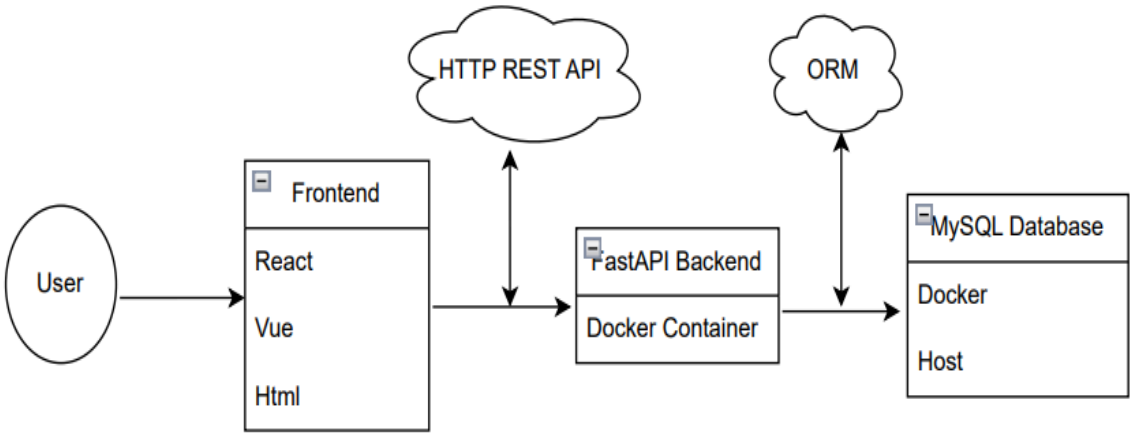
- Project Mentors
- System Architects

1.3 System Overview

The system consists of :

- Frontend (Client Application)
- Backend (FastAPI REST API)
- MySQL Database
- Docker-based container environment

Users interact through the frontend, which communicates with backend APIs. The backend handles business logic, authentication, and database operations.



2. System Design

2.1 Application Design

Backend developed using :

FastAPI

Uvicorn ASGI server

SQLAlchemy ORM

PyMySQL driver

JWT-based authentication

Frontend using :

SPA (Single Page Application)

Communicates with backend via REST APIs

2.2 Process Flow

User accesses frontend.

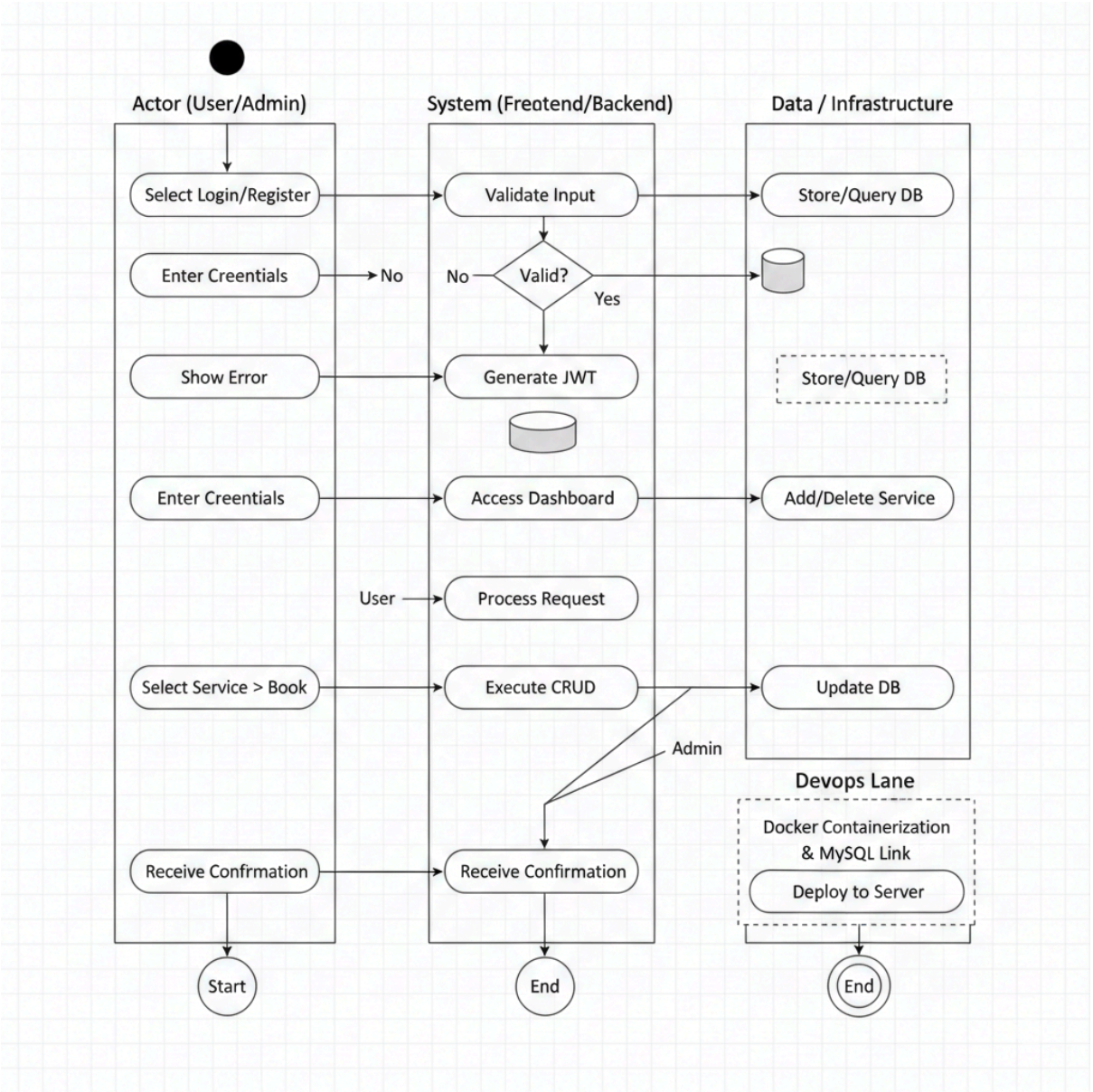
User registers or logs in.

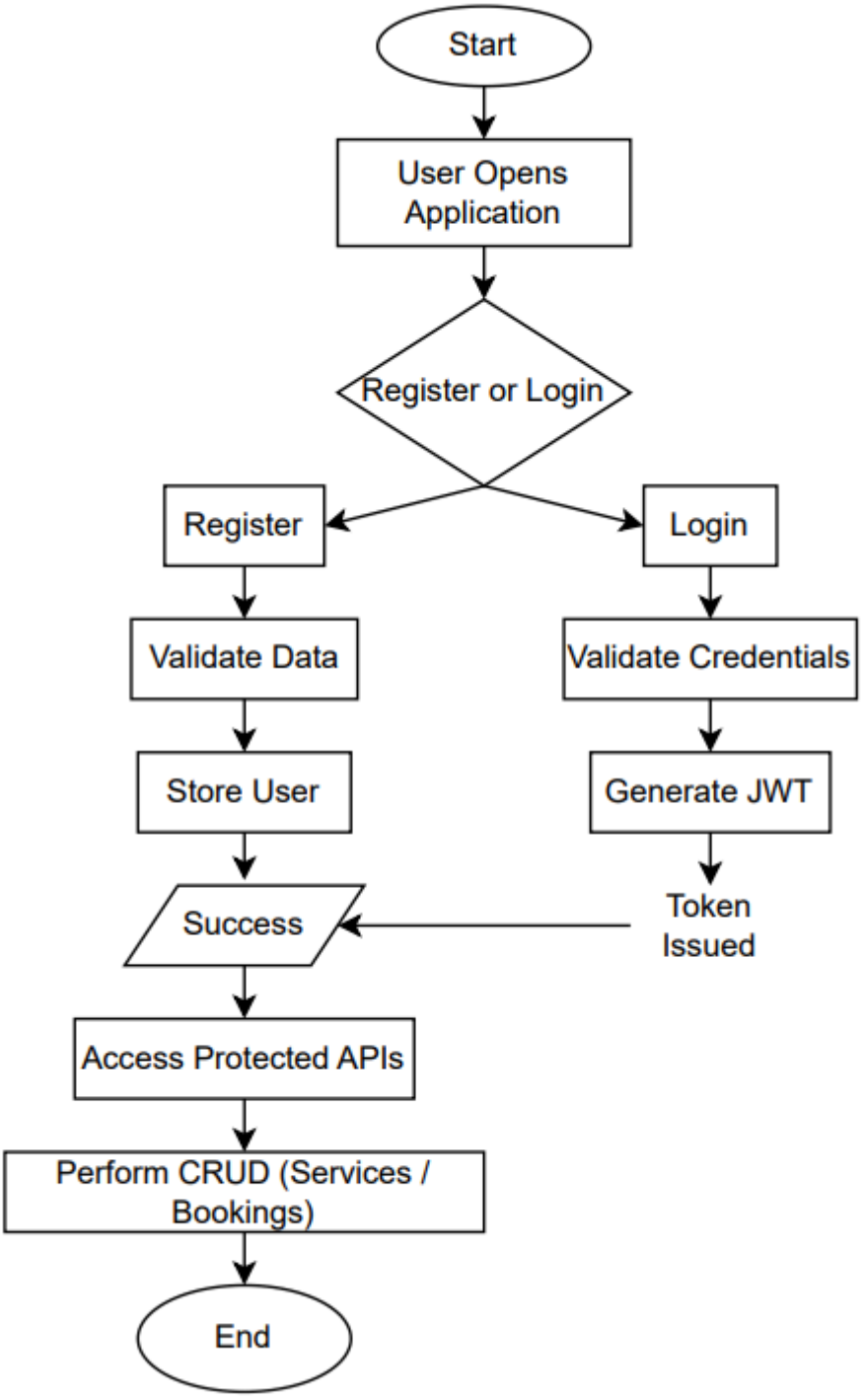
Backend validates credentials.

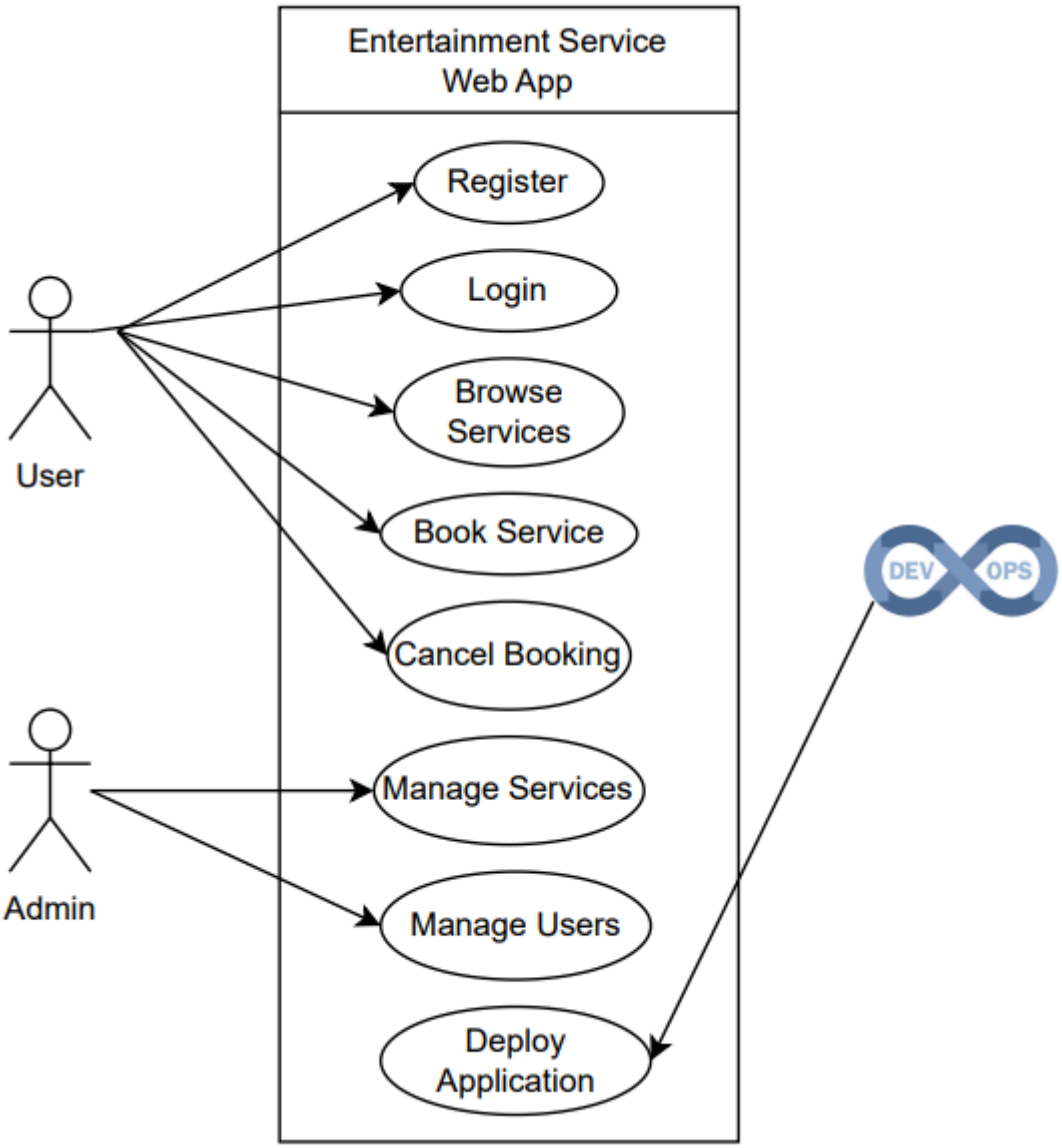
JWT token generated upon successful authentication.

Token used for protected API access.

Backend performs CRUD operations via database.


Actor (User/Admin)
System (Frontend/Backend)
Data / Infrastructure
Select Login/Register
Validate Input
Store/Query DB
Enter Creentials
No
No
Valid?
Yes
Show Error
Generate JWT
Store/Query DB
Enter Creentials
Access Dashboard
Add/Delete Service
User
Process Request
Select Service > Book
Execute CRUD
Update DB
Admin
Devops Lane
Docker Containerization
& MySQL Link
Deploy to Server
Receive Confirmation
Receive Confirmation
Start
End
End


Start
User Opens Application
Register or Login
Register
Login
Validate Data
Validate Credentials
Store User
Generate JWT
Success
Token Issued
Access Protected APIs
Perform CRUD (Services / Bookings)
End



2.3 Information Flow

All sensitive communication uses secure tokens.

User

↓

Frontend

↓

Backend API

↓

Backend

↓

Frontend

↓

User

2.4 Components Design

Backend Components :

Authentication Module

User Management Module

Booking/Service Module

Database Module

Token Generation & Validation Module

Infrastructure Components:

Docker Engine

MySQL Server

Uvicorn Server

2.5 Key Design Considerations

- Scalability through containerization
- Stateless API design
- Secure token-based authentication
- Clean modular structure

2.6 API Catalogue

Authentication APIs

POST /register

POST /login

User APIs

GET /users

GET /users/{id}

PUT /users/{id}

DELETE /users/{id}

Service APIs

GET /services

POST /services

PUT /services/{id}

DELETE /services/{id}

Swagger Documentation available at : /docs

---

3. Data Design

Relational database (MySQL)

Main Entities :

Users

Services

Bookings

Roles

3.1 Data Model

Data includes :

- Users
- id (Primary Key)
- name
- email
- password_hash
- role
- created_at
- Services
- Id
- Name
- Description
- Price
- availability_status
- Bookings

- Id
- user_id (Foreign Key)
- service_id (Foreign Key)
- booking_date
- status

Relationships :

One user → Many bookings

One service → Many bookings

3.2 Data Access Mechanism

SQLAlchemy ORM

Connection string via environment variable :

DATABASE_URL

Managed using session-based transactions

3.3 Data Retention Policies

- User data retained until account deletion
- Booking records stored permanently unless manually removed
- Logs retained based on server configuration

3.4 Data Migration

Database schema managed via migrations (Alembic recommended)

Manual migration scripts supported

---

4. Interfaces

External Interfaces:

- REST APIs (JSON)
- HTTP Protocol

Internal Interfaces:

- ORM to Database
- Backend modules communication

5.State and Session Management

Stateless API

JWT tokens used for session management

No server-side session storage

---

6. Caching

Currently not implemented.

Future improvements may include:

- Redis for API response caching
- Token blacklist caching

---

7. Non-Functional Requirements

System must support concurrent users.

Scalability

Container-based deployment enables horizontal scaling.

Reliability

Graceful error handling and validation mechanisms.

Maintainability

Modular code structure.

7.1 Security Aspects

- JWT Authentication
- Password hashing (bcrypt)
- Environment variables for secrets
- Role-based access control
- Input validation
- CORS configuration

7.2 Performance Aspects

- Async FastAPI framework
- Lightweight Docker container (python:3.11-slim)
- Optimized dependency installation
- Database indexing recommended

---

8. References

- FastAPI Documentation
- Docker Documentation
- MySQL Documentation
- SQLAlchemy Documentation
- JWT Authentication Standards
