TISM

TECH

Hotel Booking System

Project Team Information

PREPARED BY :
22i-1291 Mishal Ali
22i-0861 Ayaan Mughal
22i-0832 Ayaan Khan

Hotel Booking System

TISM Tech: Ayaan Khan, Ayaan Mughal, Mishal Ali

Software Engineering, Spring 2025

System Overview

Web-based Platform

Microservices-based using MERN stack for hotel reservations

Guest Experience

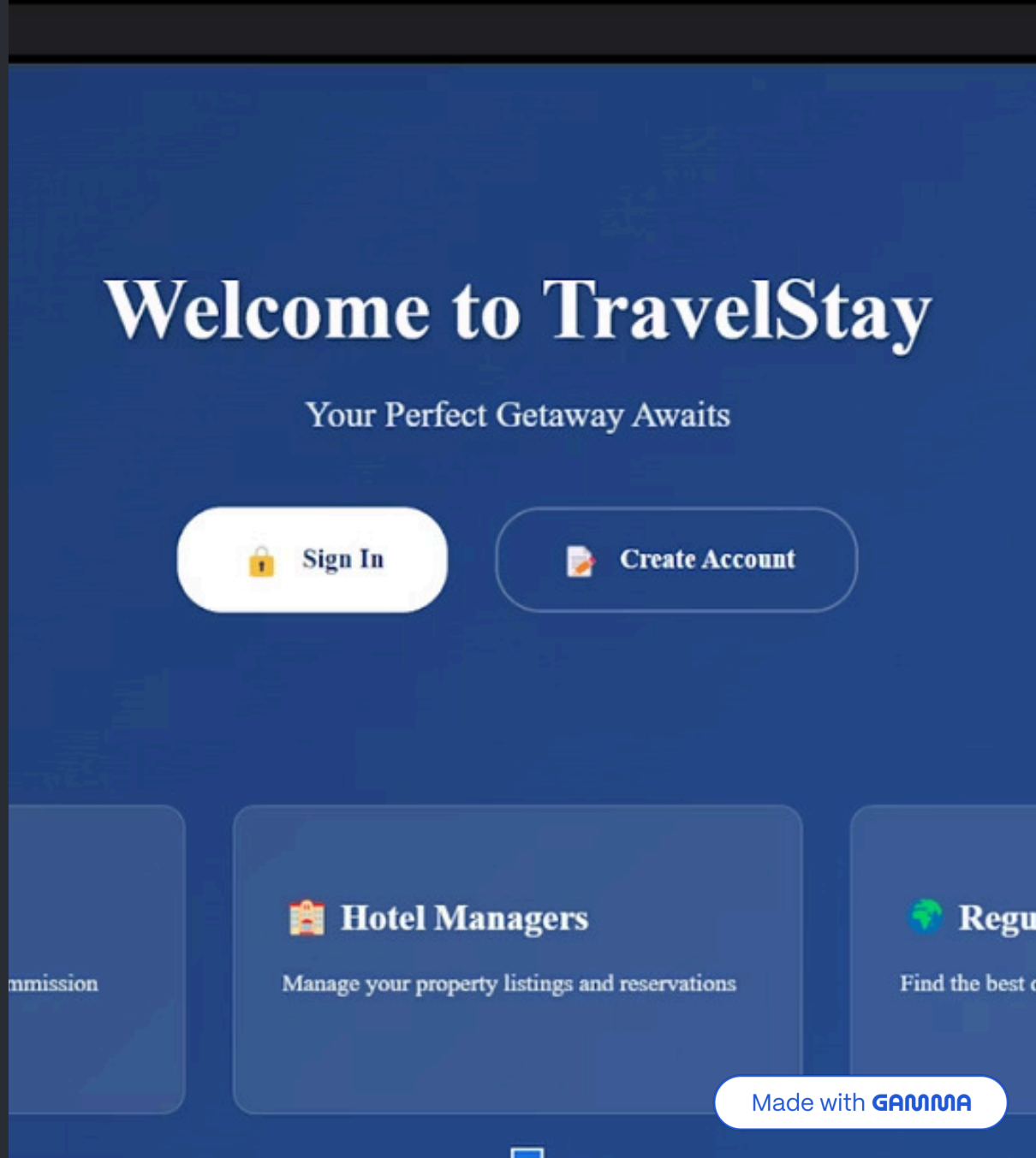
Simplified search, book, manage workflow

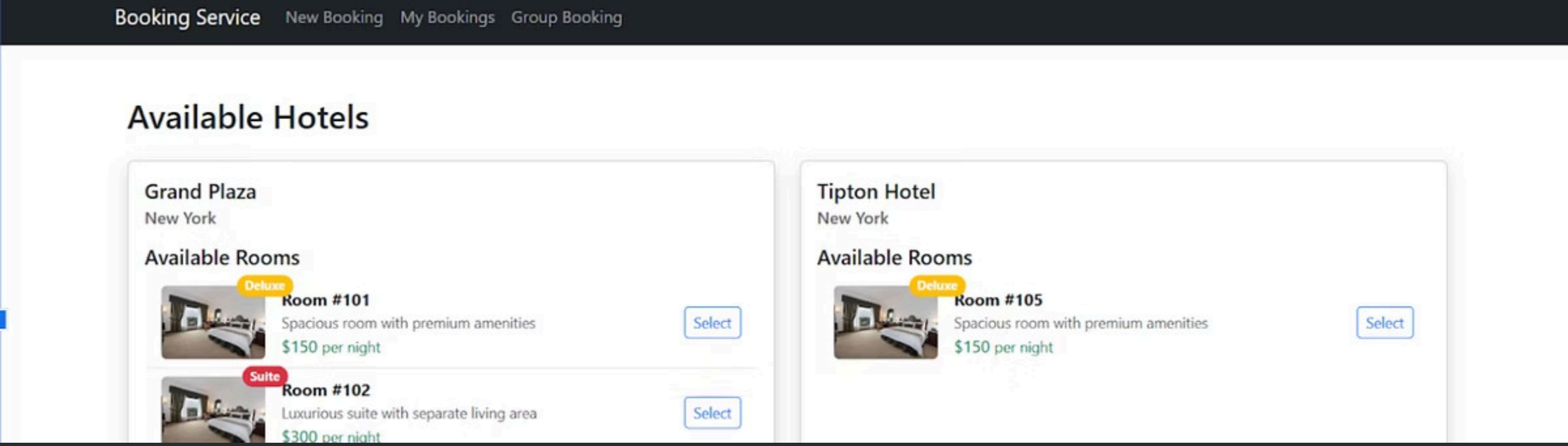
Hotel Management

Real-time inventory and pricing controls

Travel Agents

Support for bulk and group bookings





Functional Requirements

Administer Rooms/Hotels

Add/Update Room and Hotel prices and capacities

Book a Room

With optional add-ons

Manage Bookings

View, modify, cancel with notifications

User Accounts

Register/login, loyalty program

Group Bookings

For travel agents

Non-Functional Requirements

Scalability

Independent microservices, containerized

Maintainability

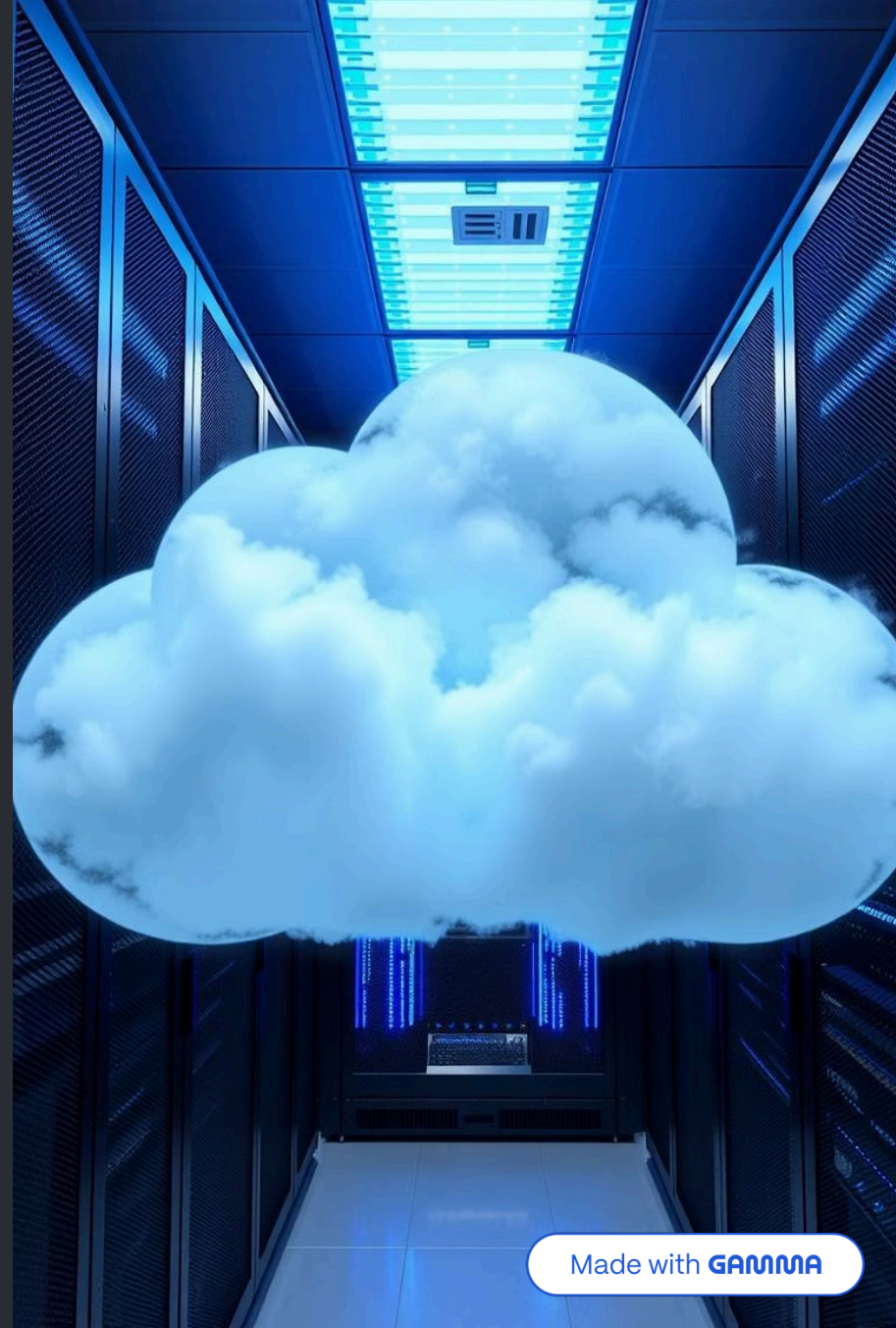
Code must be modular, well documented, and easy to update

Usability

Provide an intuitive, accessible interface on Desktops

Performance

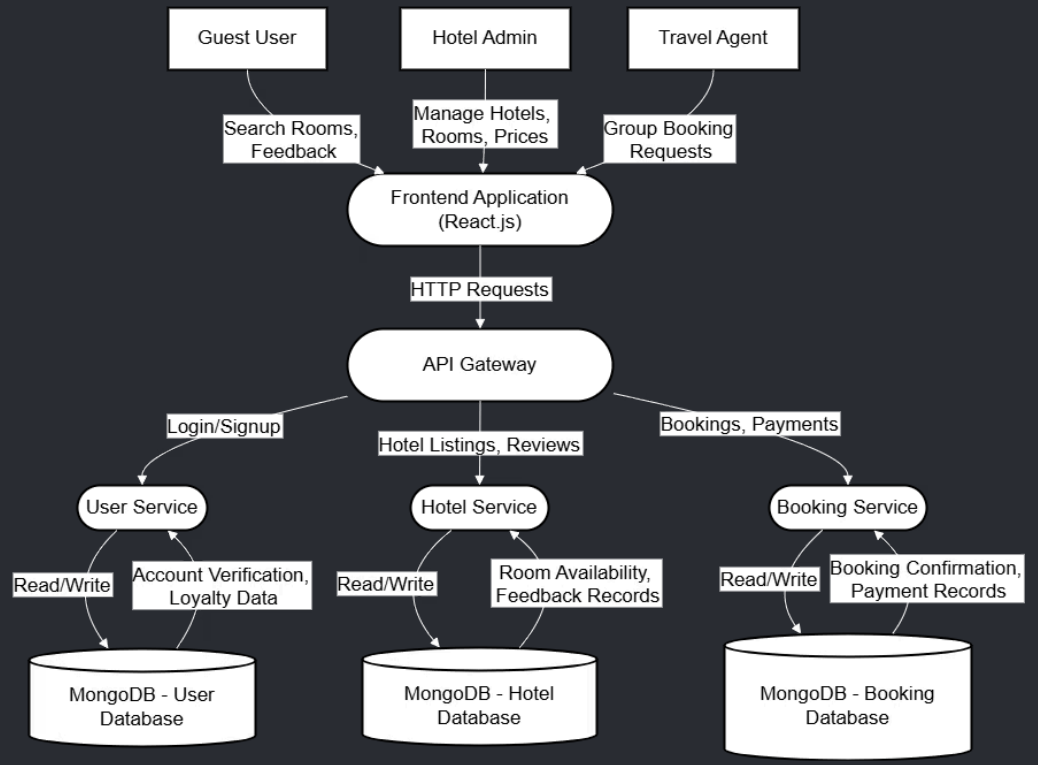
Page-load ≤ 5s under normal load



Data-flow Diagram

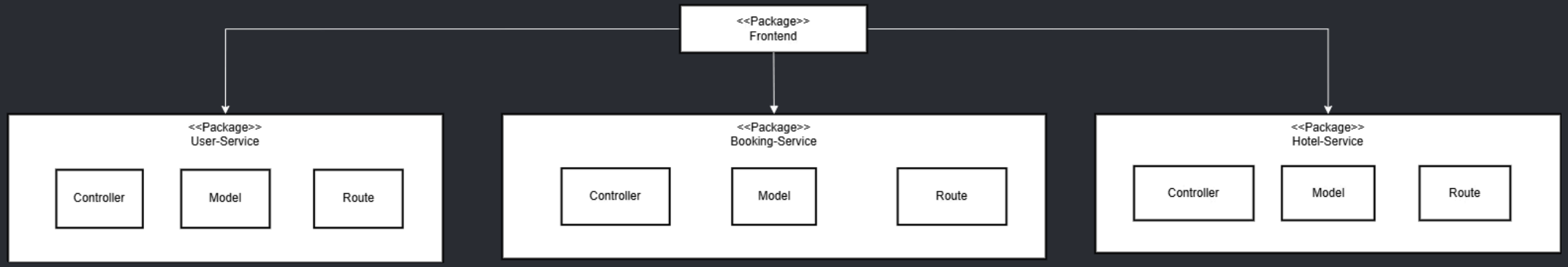
Class Diagram not suitable since program not based on OOP Principles

Data-flow Diagram instead chosen to represent it



Architecture (Package Diagram)

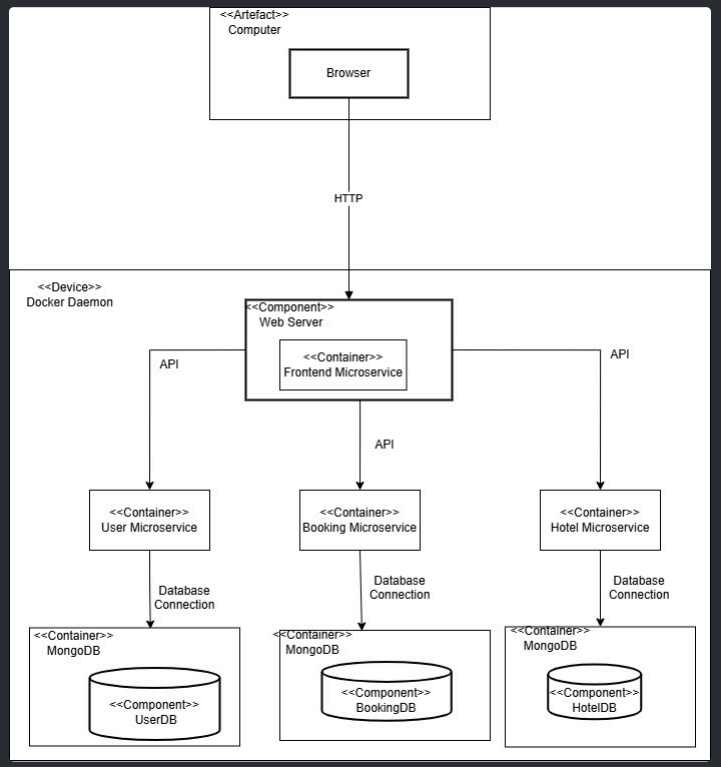
Microservices Based Architecture



Each Micro-service follows a **layered structure**:

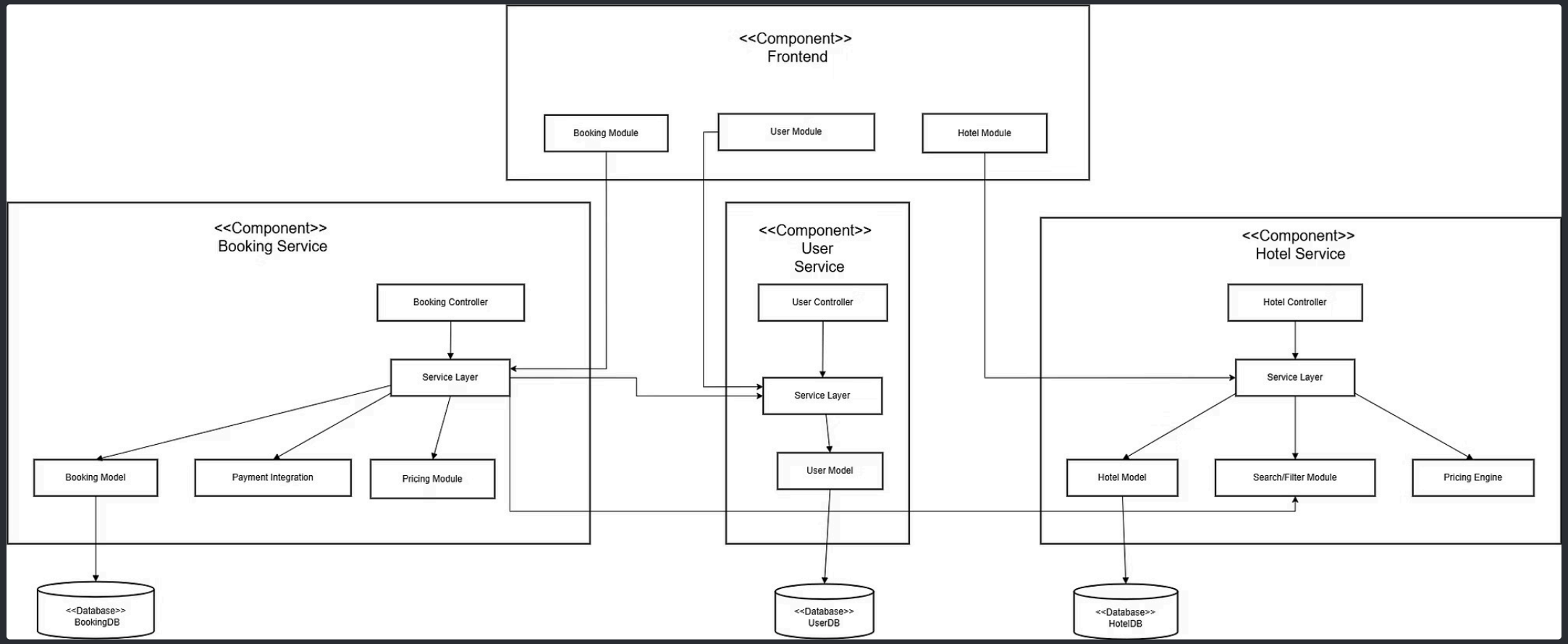
- **Controller**: Handles logic and validation
- **Route Layer**: Manages request routing
- **Model Layer**: Defines and interacts with MongoDB collections

Deployment Diagram



To further understand the workings of the program architecture

Component Diagram



Project Planner

Planning & Setup

Feb 12–28

Sprint 1 & 2

Mar 6–23

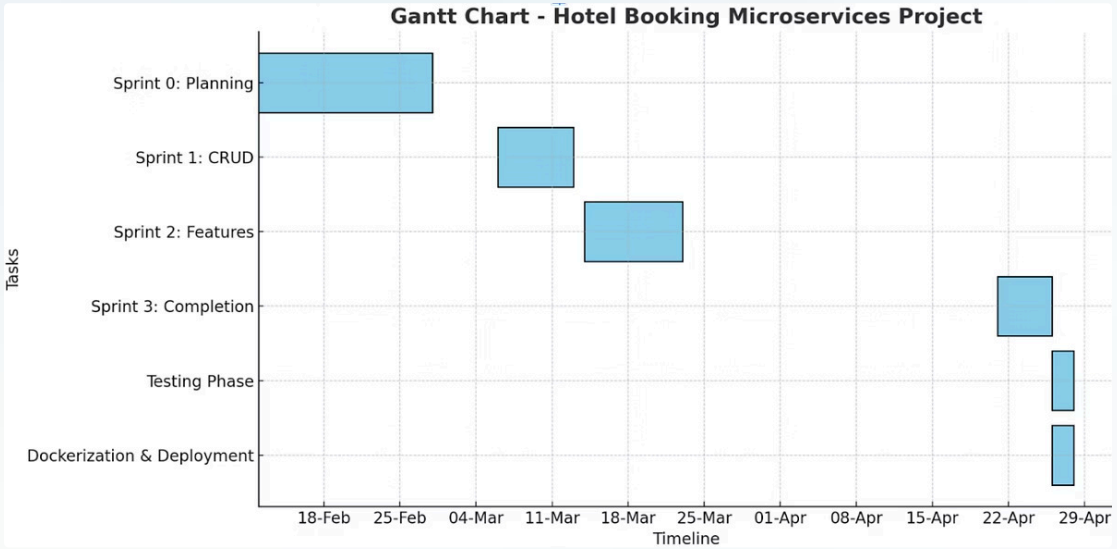
User login, basic booking

Search, pricing, manage bookings

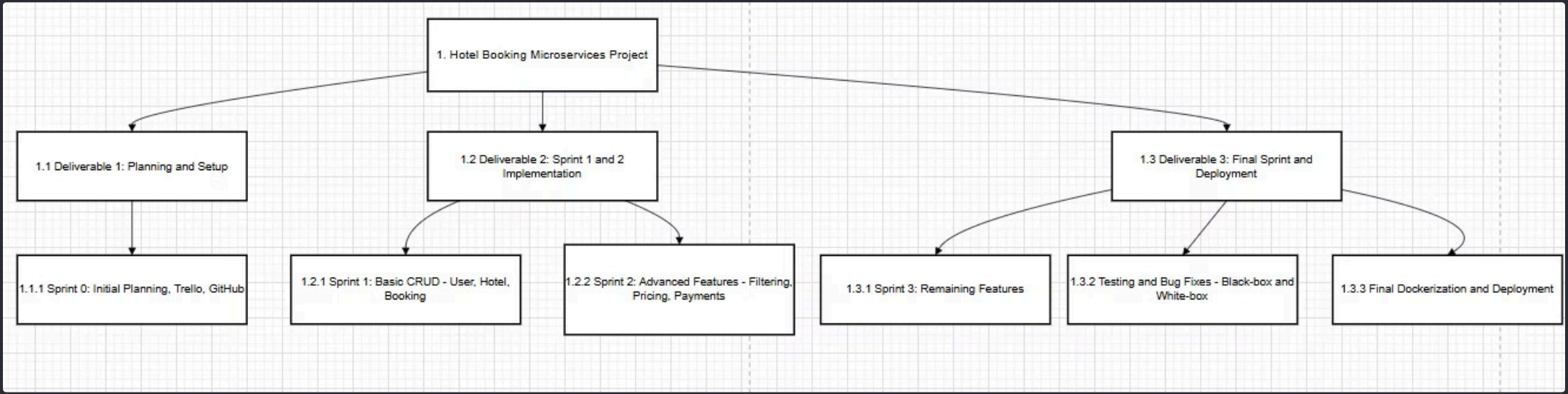
Sprint 3 & Testing

Apr 21–28

Payments, group bookings, reporting



Work Breakdown Structure

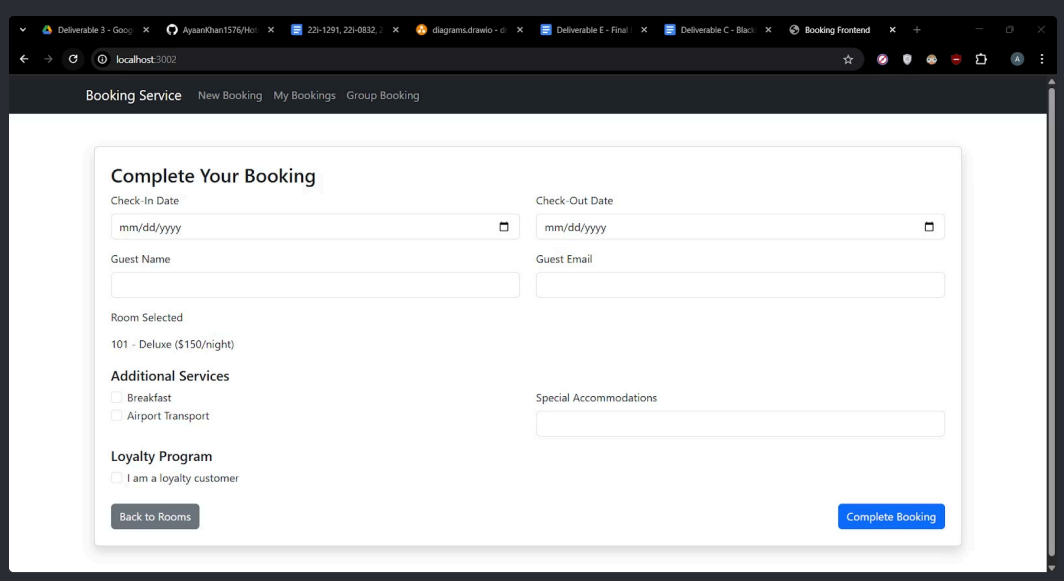


User Stories

US02

Booking a Room

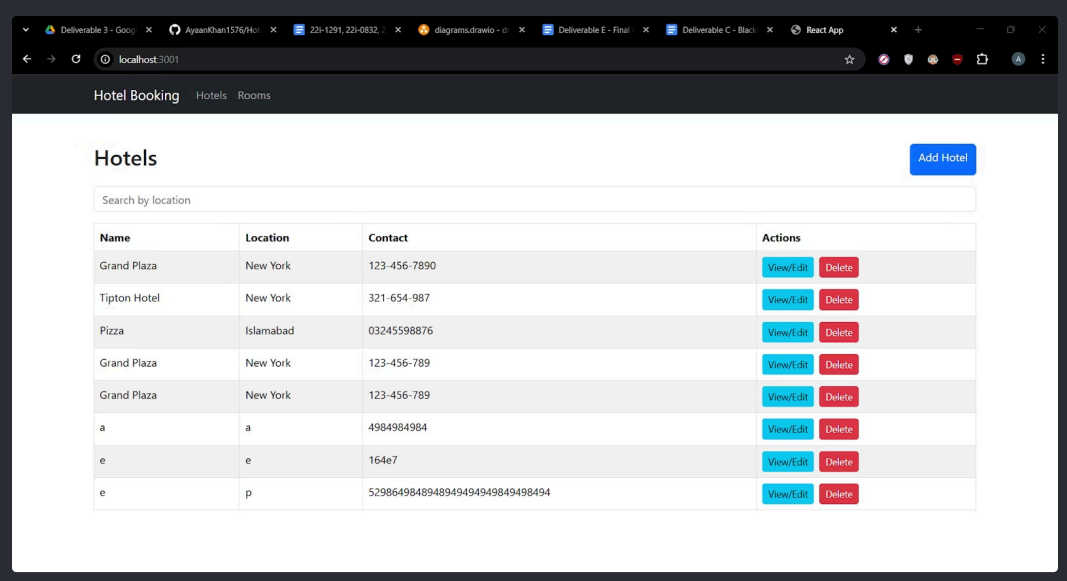
Secure, real-time with add-ons



US11

Manage Hotels

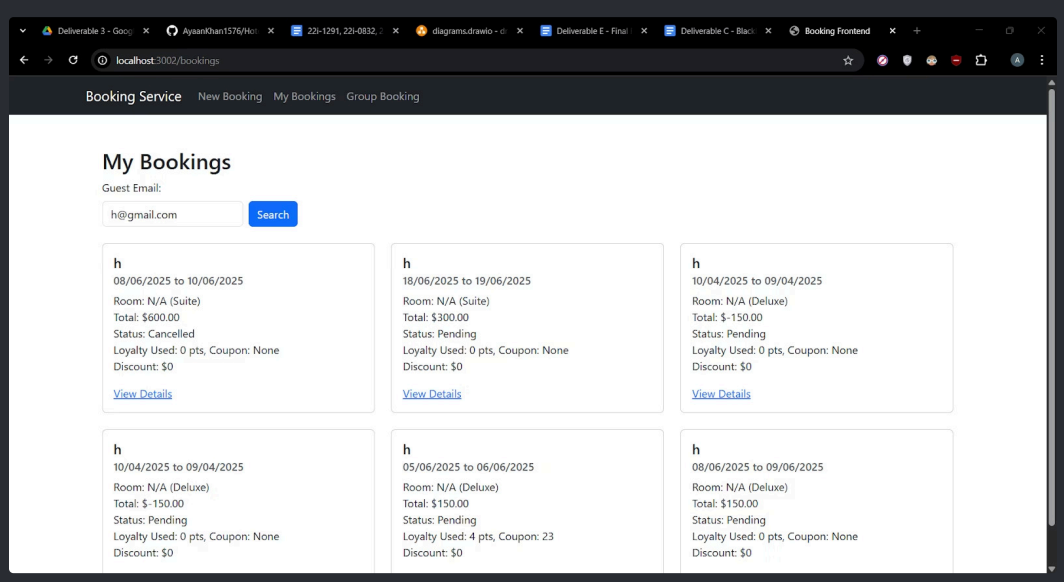
CRUD operations on Hotels to maintain catalogue



US01

Search & Filter

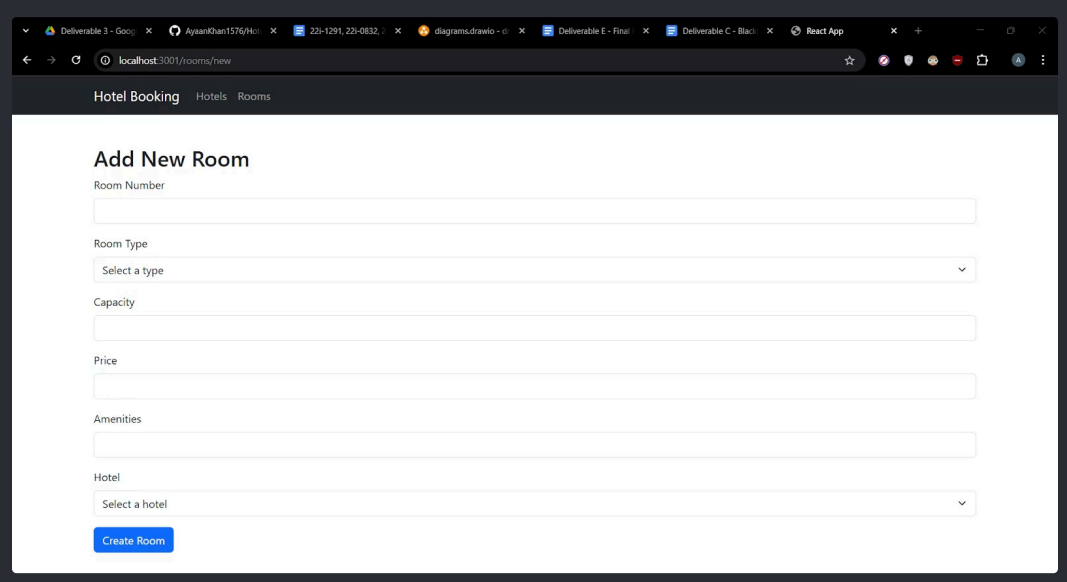
Location, dates, amenities, sort



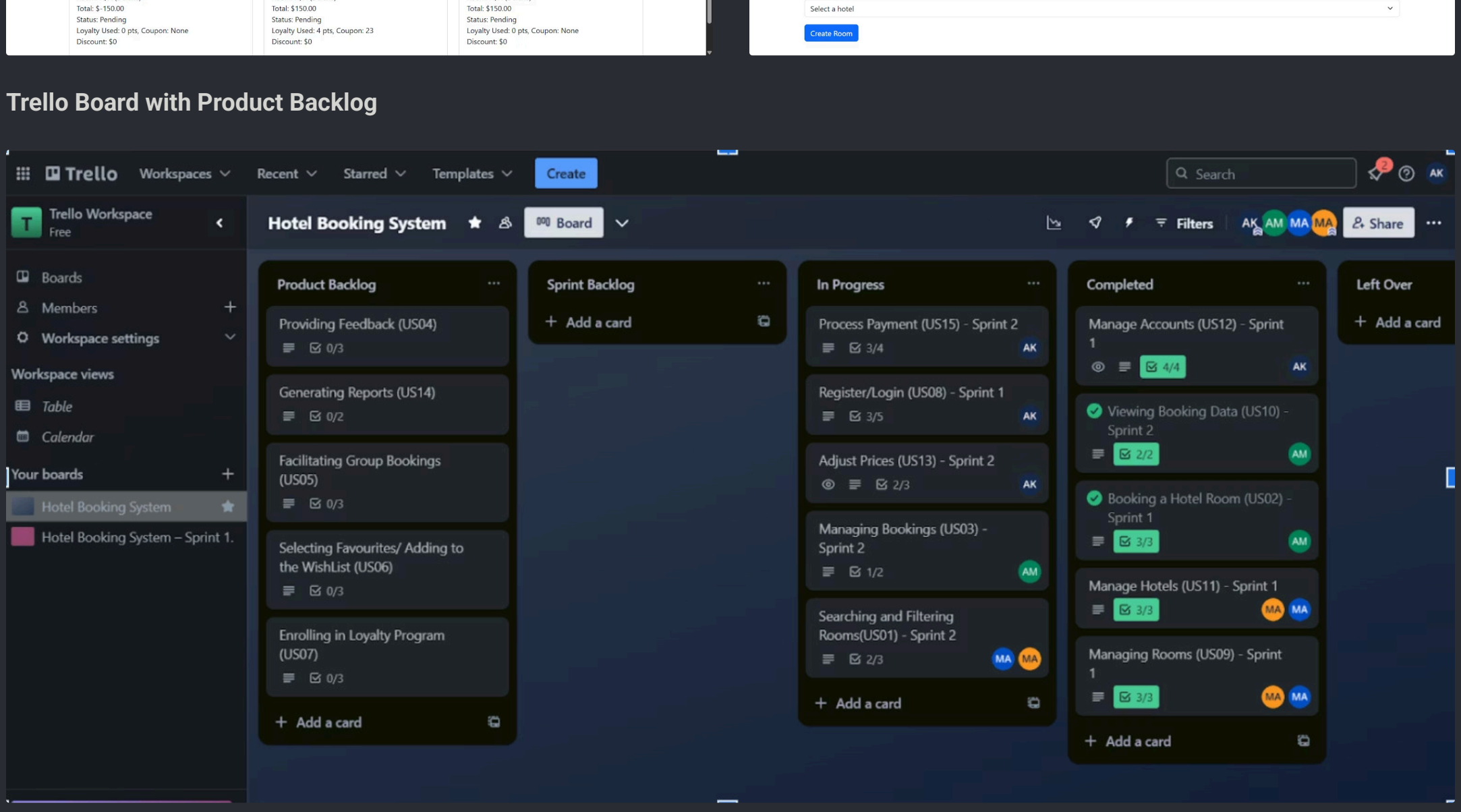
US13

Adjust Prices

Real-time pricing rules updates



Trello Board with Product Backlog



Examples of User Story Format

High Priority

- **US11: Manage Hotels:** As an Admin, I want to add and manage hotels within the system so that hotel information is accurate and reflective of available inventory
    - **US11.1:** Provide an intuitive interface for admins to add new hotels with necessary details like name, location, contact information, and amenities.
    - **US11.2:** Enable updating and editing of existing hotel information to ensure accuracy.
    - **US11.3:** Allow removal of hotel listings to maintain an up-to-date inventory.

- **US02: Booking a Hotel Room:** As a Guest, I want to securely book a hotel room in real-time so that I have my accommodation confirmed to my liking
    - **US02.1:** Display comprehensive room details, including images, descriptions, pricing, and available dates.
    - **US02.2:** Offer guests the ability to select additional services such as breakfast, airport transportation, or special accommodations.
    - **US02.3:** Issue immediate booking confirmations and receipts upon successful payment transactions to assure guests of their reservations.

Black-Box Testing

Equivalence Class Partitioning

- Divide the input space into “equivalence classes” where any one value in the class should behave the same

E.g US02 Booking a Hotel Room:

- **Valid email format**: "name@email.com"
- **Invalid email format**: Any email not in correct syntax

Test Case ID: TC-US02-03

- **Test Case Name**: Booking with invalid email
- **User Story**: US02: Booking a Hotel Room
- **Input**: Valid guest information, email doesn’t follow format “hgmail.com”
- **Expected Output**: System rejects transaction with "Invalid email syntax" message
- **Actual Output**: Rejects booking and gives error
- **Status**: Passed
- **Testing Method**: Equivalence Class Partitioning
- **Steps to Execute**:
    1. Select a hotel and a room.
    2. Enter valid guest details.
    3. Enter invalid credit card number (e.g., 1234).
    4. Attempt to book.
    5. Verify that payment is rejected.

Boundary Value Analysis

- we focus on the “edges” of each valid interval:

E.g US02 Booking a Hotel Room:

Check-in / Check-out dates:

- Just inside availability (e.g. first available date and last available date)
- Just outside availability (day before first or day after last)

Test Case ID: TC-US02-05

- **Test Case Name**: Booking on current date (minimum valid check-in)
- **User Story**: US02: Booking a Hotel Room
- **Input**: Check-in date = Today's date
- **Expected Output**: Booking is allowed
- **Actual Output**: Room not available
- **Status**: Failed
- **Testing Method**: Boundary Value Analysis
- **Steps to Execute**:
    1. Book hotels with today's date.
    2. Select room.
    3. Complete booking process.
    4. Verify booking success.

White-Box Testing

Performed on all backend microservices (User, Hotel, Booking)

- Used Jest for unit tests and Supertest for endpoint/integration tests
- All Services achieved MORE than 80% Coverage
- Focused on exercising internal logic, data flows, and error branches

What IS Covered	What is NOT Covered
Core Business Logic	Third-Party Libraries
Branch Coverage	Frontend
Function & Statement Coverage	External Service Failures
API Endpoints	Cron Jobs

User Service:

All files

81.39% Statements 175/215 **63.01%** Branches 46/73 **84.21%** Functions 16/19 **82.26%** Lines 167/203

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
controllers	83.76%	129/154	71.18%	42/59	93.75%	15/16	85.21%	121/142
middleware	85%	17/20	50%	4/8	100%	1/1	85%	17/20
models	52%	13/25	0%	0/6	0%	0/2	52%	13/25
routes	100%	16/16	100%	0/0	100%	0/0	100%	16/16

Hotel Service

All files

83.92% Statements 94/112 **75%** Branches 21/28 **100%** Functions 11/11 **86.79%** Lines 92/106

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
models	100%	9/9	100%	0/0	100%	0/0	100%	9/9
routes	82.52%	85/103	75%	21/28	100%	11/11	85.56%	83/97

Booking Service

All files

80.85% Statements 245/303 **63.44%** Branches 59/93 **82.75%** Functions 24/29 **80.95%** Lines 238/294

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
config	100%	8/8	100%	0/0	100%	1/1	100%	8/8
controllers	78.11%	207/265	63.44%	59/93	82.14%	23/28	78.12%	200/256
models	100%	10/10	100%	0/0	100%	0/0	100%	10/10
routes	100%	20/20	100%	0/0	100%	0/0	100%	20/20

Work Division

All Version Control done via Github and Project Management done via Trello

Ayaan Khan	Ayaan Mughal	Mishal Ali
Scrum Master, Tester	**UI Designer, Developer**	**Requirements Architect, Developer**
• 5 User-Service User Stories • Blackbox and Whitebox Testing • Setting up Trello board and structure • Documentation	• 5 Booking-Service User Stories • Responsible for Frontend Development • Undertook Services Integration • Documentation	• 5 Hotel-Service User Stories • Responsible for Frontend Development • Undertook Requirements analysis • Documentation

Further Improvements

- Project security can be improved so that users cannot abuse bugs to access out of role functions
- More functionality can be added to enhance user experience
- Deployed via Kubernetes and incorporate CI/CD pipelines for Scalability and Maintainability
- Add support for other devices
- Test Frontend via End-to-end (Cypress) automated UI tests

Conclusions and Lessons Learned

What Went Well:

- Clear sprint goals, rapid prototyping with React & Express
- Major User Stories implemented on the backend
- Work Division was seamless and well coordinated
- Version control implemented well so no problem with rolling back or updating

Challenges:

- Integration testing across microservices
- Frontend integration with backend
- Ensuring > 70% white-box coverage under time pressure
- Figuring out right diagrams or schemas to use for a non-OOP project

Takeaways:

- Importance of automated tests early on
- Start work early so to not be overburdened near deadlines
- Learned the importance of requirements analysis in figuring out what to program
- Found that mocking external calls (Mongoose, Axios) speeds up tests and keeps them deterministic—but real integration tests against a staging DB are still invaluable for end-to-end confidence.
- Testing while development would aide more in bug fixing instead of testing the entire program at once at the end