1. Project Structure

```
workspace-booking-system/
 ├── backend/
 │   ├── server.js
 │   ├── controllers/
 │   │   ├── booking.controller.js
 │   │   ├── analytics.controller.js
 │   ├── routes/
 │   │   ├── bookings.js
 │   │   ├── analytics.js
 │   ├── models/
 │   │   ├── rooms.js
 │   │   ├── bookings.js
 │   ├── utils/
 │   │   ├── time.js
 │   │   ├── pricing.js
 │  └── package.json
 ├── frontend/
 │   ├── src/
 │   │   ├── pages/
 │   │   ├── components/
 │   │   ├── api/
 │   │   │   ├── axios.js
 │   │   ├── styling/
 │   │  └── App.jsx
 │   ├── public/
 │  └── package.json
 ├── README.md
└── ARCHITECTURE.md
```

2. High-Level System Diagram

[User Browser] ↔ [Frontend (React + Vite)] ↔ [Backend (Node + Express)]

↕

[In-Memory Data Store: rooms[], bookings[]]

2.1 Users interact with the frontend

2.2 Frontend issues HTTP requests to backend for bookings, cancellations, analytics

2.3 Backend processes logic (conflict detection, pricing) and uses in-memory arrays to store state

2.4 No external database in this version

3. Core Components

3.1 Frontend Web App

Purpose: UI for booking rooms, viewing success, admin dashboard.

Technologies: React + Vite, Axios, React Router, custom CSS for layout and toast notifications.

Deployment: Vercel.

Key modules:

i. api/axios.js: sets baseURL and handles API calls

ii. BookingForm page: time input, calculates cost, submits booking

iii. AdminPage: tabs for bookings and analytics, cancel button, date filters, toast messages

3.2 Backend Service

Purpose: REST API handling bookings, cancellations, analytics, business logic.

Technologies: Node.js, Express, UUID, custom utilities for time and pricing.

Deployment: Render.

Key modules:

i. controllers/booking.controller.js: handles create and cancel logic

ii. controllers/analytics.controller.js: handles analytics queries

iii. models/rooms.js & models/bookings.js: in-memory data storage

iv. utils/time.js: overlapping logic

v. utils/pricing.js: calculates totalPrice based on peak hours

4. Data Stores

4.1 In-Memory Data

i. rooms[]: seeded room data (id, name, baseRate, capacity)

ii. bookings[]: records bookings with id, roomId, roomName, userName, startTime, endTime, totalPrice, status

Note: Data is lost on server restart; suitable for assignment environment rather than production.

5. External Integrations / APIs

None in this version — the system is self-contained.

All logic resides within the frontend and backend.

6. Deployment & Infrastructure

i. Frontend built with Vite and deployed to Vercel (static hosting + routing)

ii. Backend deployed on Render (Node environment)

iii. CORS configured to allow localhost (development) and the frontend's production domain

iv. Build & start commands:

Backend: npm install → node server.js

Frontend: npm install → npm run build (Vite)

7. Security Considerations

i. No authentication in this version (assignment scope)

ii. CORS set to limit allowed origins (frontend + localhost)

iii. Backend enforces business rules:

No overlapping bookings

No cancel within 2 hours

Max booking duration 12 hours

iv. HTTPS assumed (via hosting provider)

8. Development & Testing Environment

i. Local development:

Backend: http://localhost:5000/api

Frontend: http://localhost:5173

ii. Testing scenarios include:

Booking crossing peak hours

Overlapping booking attempts

Cancellation edge cases

Analytics date-range queries

9. Future Considerations / Roadmap

i. Use a persistent database (PostgreSQL, MongoDB) instead of in-memory storage

ii. Add user authentication & roles (admin, customer)

iii. Improve UI/UX with calendar visualization and drag-select times

iv. Add email confirmations/notifications

v. Handle timezones more robustly if deployment becomes global

10. Glossary

i. baseRate: standard hourly rate for a room

ii. Peak hours: 10 AM–1 PM and 4 PM–7 PM (Mon–Fri)

iii. totalPrice: calculated price for the booking (peak + non-peak)

iv. Confirmed: booking status when active

v. Cancelled: booking status when cancelled by admin