Event Booking System - Design Document

Hoda Samir Touny Mohammed

Project Overview

Momentix is a full-stack event booking platform built to make discovering, booking, and managing events effortless for users — and powerful for admins.

The backend was designed to be clean, secure, and fully tested with unit and integration tests, following industry best practices.

Features:

- **Authentication & Roles**
  - Secure user login and registration using JWT, with role-based access control for admin and regular users.
- **Event Management**
  - Admins can create, edit, and delete events, while users can browse events with pagination and filters.
- **Event Booking**
  - Authenticated users can book available events, view confirmation, and avoid duplicate bookings.
- **Admin Dashboard**
  - An interactive dashboard displays total users, events, bookings, and revenue, with charts by category.
- **Multilingual Support**
  - Full support for English and Arabic, including translated UI elements and event categories.
- **Auto-Expire Old Events**
  - A scheduled cron job runs automatically to mark past events as expired based on date.
- **UX & Feedback**
  - Toast notifications, loading spinners, and confirmation modals enhance user experience.
- **Code Quality & Testing**
  - Clean modular structure with unit tests.

Backend Overview

Technology Stack

To build **Momentix**, I focused on using reliable and modern technologies that are widely adopted in production environments.

- **Backend:** Node.js with Express.js for building fast and scalable REST APIs.
- **Database:** PostgreSQL, managed through Prisma ORM to simplify database modeling and querying.
- **Authentication:** JWT combined with secure HTTP-only cookies for safe user sessions.
- **Cloud Storage:** Cloudinary was used to handle image uploads, providing fast and secure image hosting.
- **Localization:** i18n library for static translations and Gemini AI integration for dynamic, high-quality event translations.
- **Security:** Helmet.js for HTTP security headers and Express Rate Limiter to prevent abuse and DDoS attacks.
- **Testing:** Jest for unit tests covering middlewares, services, and controllers, and Postman for route integration testing.

Throughout development, I used AI tools like ChatGPT and Copilot to speed up coding, debug faster, and explore better architectural decisions.

System Architecture

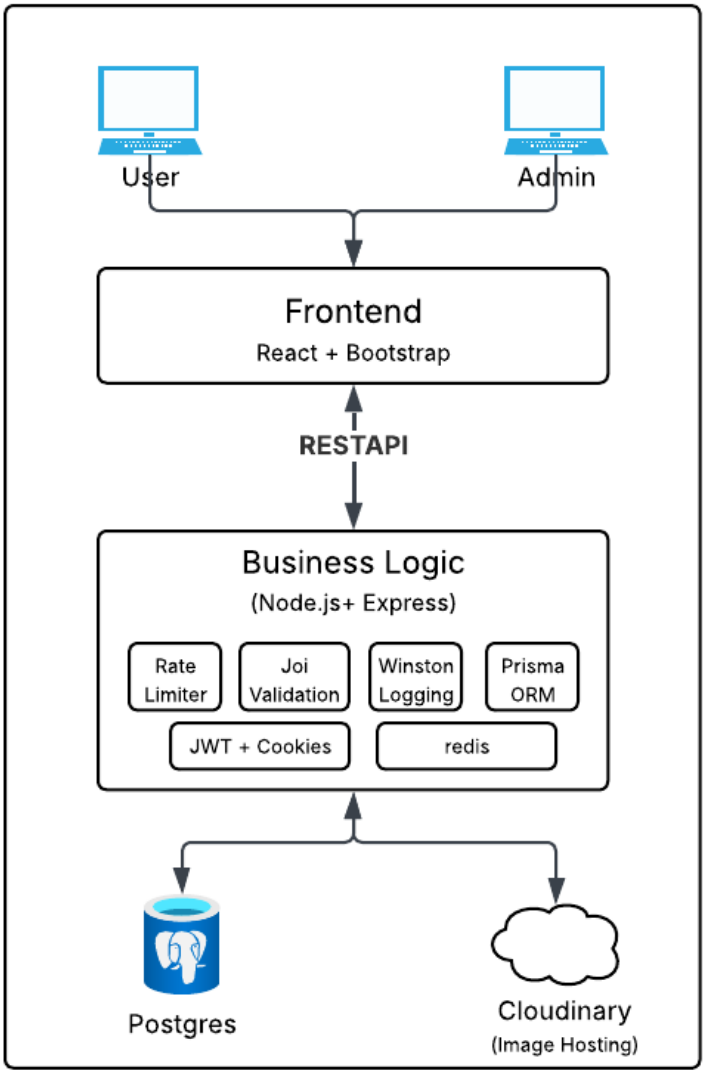
Momentix follows a clean and layered architecture designed for clarity, maintainability, and scalability.

- **Routes:** Handle incoming HTTP requests and delegate to the correct controllers.
- **Middlewares:** Handle validation, authentication, localization, and error interception before requests reach controllers.
- **Controllers:** Process requests, call the needed services, and return standardized responses.
- **Services:** Contain the core business logic and interact with the database.
- **Database Layer (Prisma):** Prisma ORM interacts with PostgreSQL, managing models, relationships, and queries.
- **Utilities and Helpers:** Manage repetitive tasks like parsing queries, handling custom errors, managing tokens, and logging.

Gemini AI are integrated directly into the translation flow for dynamic multi-language event content.

Error handling is centralized, ensuring consistent messages and status codes across the API.

Architecture Diagram



Database Design

For the database layer, I chose **PostgreSQL**, a powerful and reliable relational database system.

PostgreSQL was my choice because it handles structured data and relational models very well, which matches the event booking system's needs — where users, events, and bookings are tightly connected.

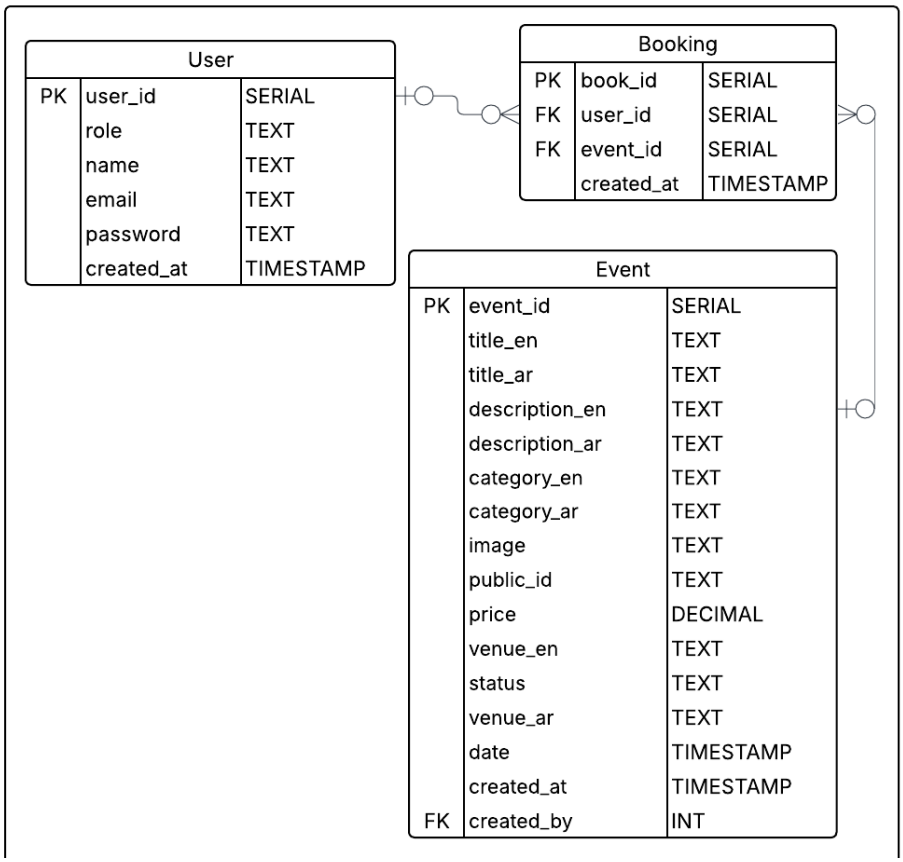
To simplify database management, I used **Prisma ORM**. Prisma made it easy to define models, handle migrations, and build clean queries without manually writing SQL.

The main relationships in the system are:

- **User ➔ Booking (One-to-Many):** A user can book multiple events.
- **Event ➔ Booking (One-to-Many):** An event can have multiple users booking it.

This relational structure keeps the system organized, easy to query, and scalable for future features like ticket management or event categories.

Entity-Relationship Diagram



Caching Strategy

To enhance performance and minimize redundant database queries, Redis was introduced as a lightweight caching layer.

- Dashboard summaries and analytics are cached for 15 minutes.
- Event listings are cached based on page, size, category, and sort order.
- Any changes to events or bookings automatically clear relevant caches to maintain consistency.

This improves frontend response times and reduces backend load during high-traffic periods.

System Components

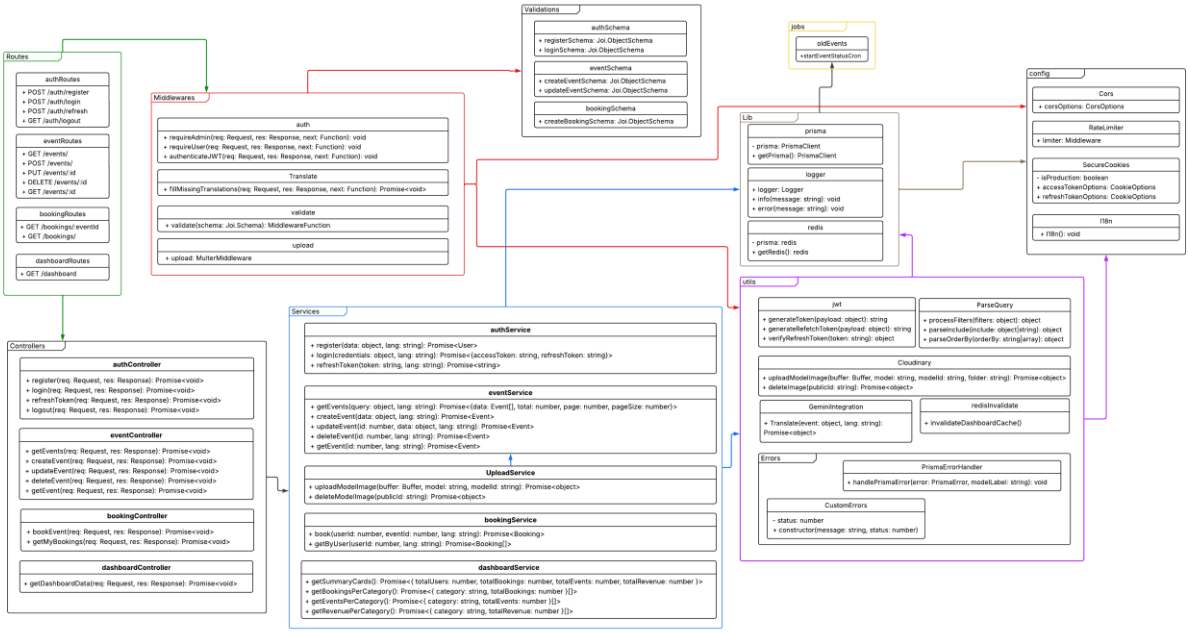
Momentix is organized into clear and well-separated components to keep the project clean, easy to scale, and easy to maintain.

The system is divided into the following main parts:

- **Routes:**
  Each feature (auth, events, bookings) has its own routes. Routes handle incoming HTTP requests and call the correct controller functions.
- **Middlewares:**
  Reusable middlewares handle tasks like validating user input, authenticating users, checking user roles (admin/user), rate limiting, setting language preferences, and translating missing fields dynamically.
- **Controllers:**
  Controllers are responsible for receiving validated requests, calling the needed services, and returning appropriate responses. They do not contain business logic.
- **Services:**
  Services contain the core business logic of the system. They interact with Prisma ORM, handle complex operations like booking an event, uploading images to Cloudinary, or translating event fields using Gemini AI.
- **Prisma ORM:**
  Prisma acts as the bridge between services and the database. It manages all queries, relationships, and migrations cleanly.
- **Utilities:**
  Utility modules manage common tasks like parsing filters for database queries, generating and verifying tokens, logging, and handling custom errors.

- **Config Files:**
  Configuration files manage reusable settings for CORS, cookies, localization (i18n), security options, and rate limiting.

Class Diagram



Security Implementation

Security was a core focus during the development of **Momentix**. I made sure the backend followed best practices to protect user data, sessions, and system stability.

The security measures included:

- **Authentication:**
  Users log in and receive an **access token** (short-lived) and a **refresh token stored** securely inside **HTTP-only cookies**.
  This prevents JavaScript from accessing the tokens and protects against XSS attacks.
- **Authorization:**
  Role-based access control (RBAC) ensures that only users with the "Admin" role can create, update, or delete events.
  Middleware functions (`requireAdmin`, `requireUser`) check user roles before allowing access.

- **Input Validation:**
  All incoming requests are validated using Joi schemas to avoid invalid or malicious data reaching the services or database.
- **Helmet.js:**
  HTTP security headers are automatically applied to protect against common vulnerabilities like clickjacking, MIME sniffing, and cross-site scripting.
- **Rate Limiting:**
  An express rate limiter restricts how many requests a client can send, helping to prevent abuse and protect against basic denial-of-service attacks.
- **Error Handling:**
  Centralized error handling catches and formats Prisma database errors, authentication errors, and general server errors, returning safe and user-friendly messages.

Together, these layers create a secure and robust backend that protects both user information and the integrity of the system.

Testing Strategy

To ensure that **Momentix** is reliable and behaves as expected, I followed a clear and layered testing strategy.

The approach combined **unit testing** for core logic with **integration testing** for full API flows:

- **Unit Testing:**
  Using **Jest**, I tested critical parts of the backend, including:
  - Middlewares (like `authenticateJWT`, `requireAdmin`)
  - Services (authService, bookingService, eventService)
  - Controllers (handling different success and error scenarios)
- **Integration Testing:**
  Using **Postman**, I tested the complete API endpoints, including:
  - User registration and login
  - Event creation, update, retrieval, and deletion
  - Booking flow (book an event, view bookings)

These tests made sure the full system flow — from the route through middleware, controller, service, and database — behaves as expected.

Frontend Overview

The frontend of Momentix is a responsive, bilingual (English–Arabic), and dynamic web application built with React.js and modern tooling. It follows a clean architecture based on **component-based design**, **context management**, and **query-based data fetching** to ensure scalability, maintainability, and excellent performance.

Color Scheme and Color Theory

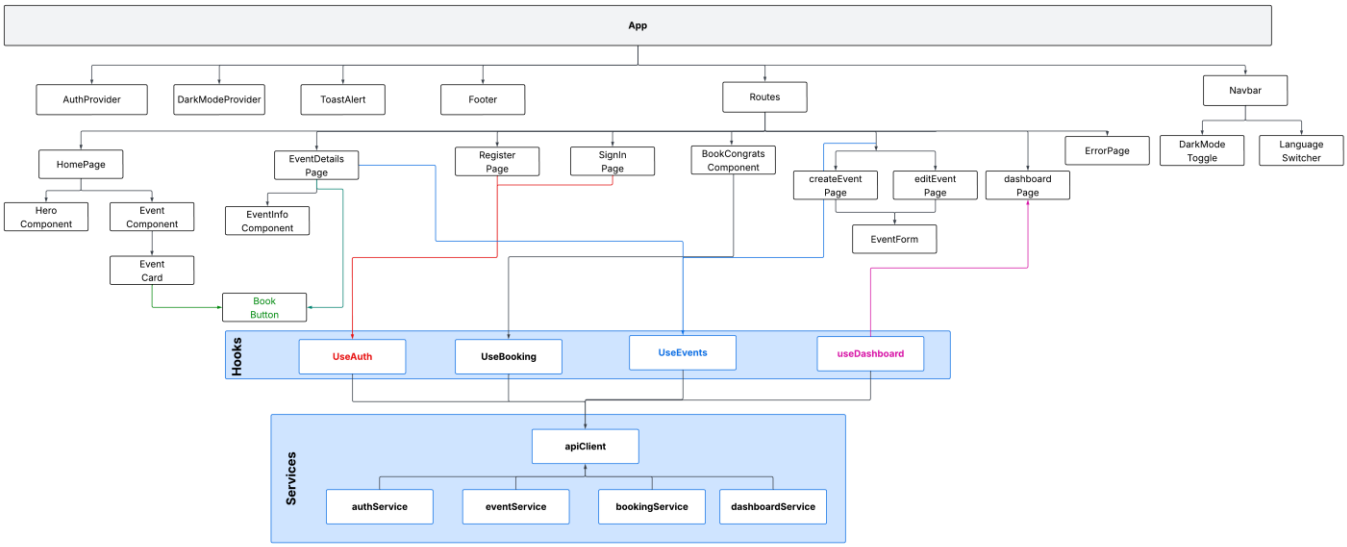
The **Momentix** color scheme is built on purposeful choices to enhance trust, clarity, and usability.

- **Primary Color (Blue Shades)**
  - Shades of blue were selected to convey **trust, security, and calmness,** essential for a booking system.
  - Only medium-light blues like #3B82F6 and #60A5FA were used avoiding **dark** or **neon** blues to keep the interface friendly, clean, and easy on the eyes.
- **Secondary Color (Red)**
  - Red is used selectively for alerts and destructive actions, drawing attention without overwhelming the user.
- **Backgrounds and Cards**
  - Light mode uses soft grays and whites for high readability.
  - Dark mode uses smooth, muted darks to reduce eye strain and modernize the interface.
- **Accent and Button Text**
  - High contrast between buttons and backgrounds improves **clickability** and ensures **accessibility** for all users.

Technology Stack:

- React.js
- React Router DOM
- React Query
- i18next (Internationalization)
- Bootstrap
- Custom Dark/Light Theme Context
- Toastify

Component Diagram



AI Integration

AI tools were an important part of developing **Momentix**, helping to speed up the development process and add smart features inside the system itself.

- **Development Assistance:**
  I used **ChatGPT** and **Windsurf** to:
  - Discuss and explore architectural decisions.
  - Speed up coding by generating repetitive tasks.
  - Teach me unit testing concepts and assist me in applying them effectively using Jest.
  - Generating component code based on image references.
  - Suggest the best structure for frontend components and state management.
- **Feature Integration:**
  I integrated **Gemini AI** directly into the event management workflow.
  When admins create or edit events, Gemini dynamically generates translations for missing English or Arabic fields, ensuring smooth multi-language support without relying only on static text files.

Diagrams Link

For better view here's lucid chart link ACT_01060584671