**Application Overview**

This application was developed as an online platform for viewing and purchasing tickets to sports matches. It allows users to browse upcoming matches, view match details, and buy tickets for specific events. The application is designed for sports enthusiasts who are interested in staying informed about upcoming matches and purchasing tickets online.

**Features**

* Displays a list of upcoming matches.
* Users can view match details, including participating teams, location, date, and ticket price.
* Only authenticated users can purchase tickets; however, match details are available to all users.
* After purchasing a ticket, the application simulates sending an email with the invoice by logging the invoice details in the console.

**Authentication and Authorization**

* Users must create an account and log in to purchase tickets.
* Logged-in users can access additional features, such as purchasing tickets and receiving invoices.

**Backend Architecture**

The backend consists of multiple microservices, each with a specific responsibility:

1. **Authentication Service (auth-service)**
   * Manages user registration, login, and authentication.
   * Issues tokens for secure user sessions.
2. **Load Balancer (Nginx)**
   * Distributes incoming requests between the authentication service, match-service, and ticket-service for efficient load handling.
3. **Match Service (match-service)**
   * Retrieves and manages match information.
   * Provides an API to fetch match lists and details.
4. **Ticket Service (ticket-service)**
   * Handles ticket purchases for matches.
   * Communicates with the invoice service via a Kafka queue to generate invoices.
5. **Invoice Service (invoice-service)**
   * Generates invoices for ticket purchases.
   * Sends invoice data to the email service via a RabbitMQ queue.
6. **Email Service (email-service)**
   * Simulates sending an email with the invoice details.
   * Currently logs the invoice content to the console instead of actually sending an email.

**Workflow**

1. A user visits the website and views upcoming matches.
2. If they want to buy a ticket, they must log in or register.
3. Once logged in, they can purchase a ticket for a match.
4. The ticket-service generates the ticket and sends a request to the invoice-service through Kafka.
5. The invoice-service processes the request, generates an invoice, and sends it to the email-service through RabbitMQ.
6. The email-service logs the invoice details, simulating the email-sending process.

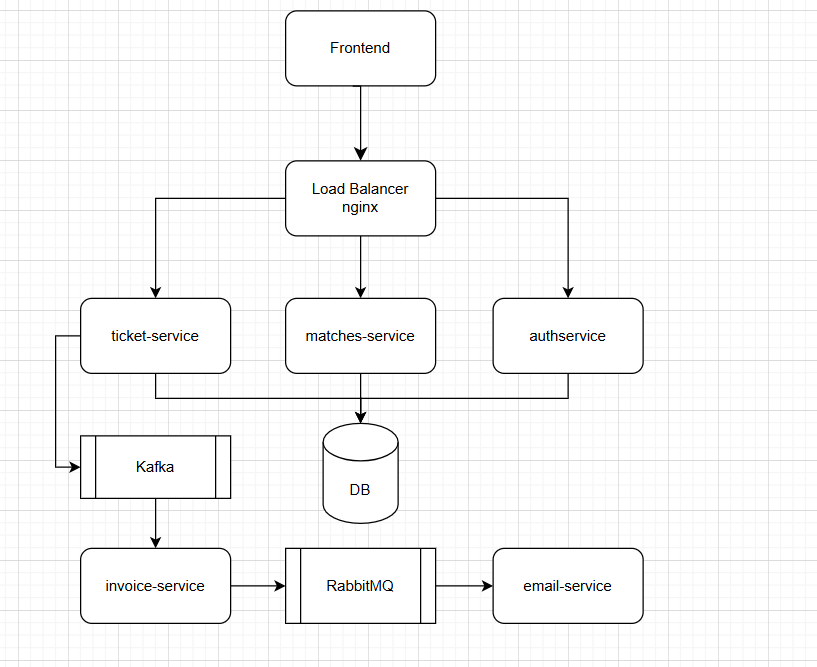
This architecture ensures scalability, modularity, and efficient event-driven communication between services.

**User Actions**

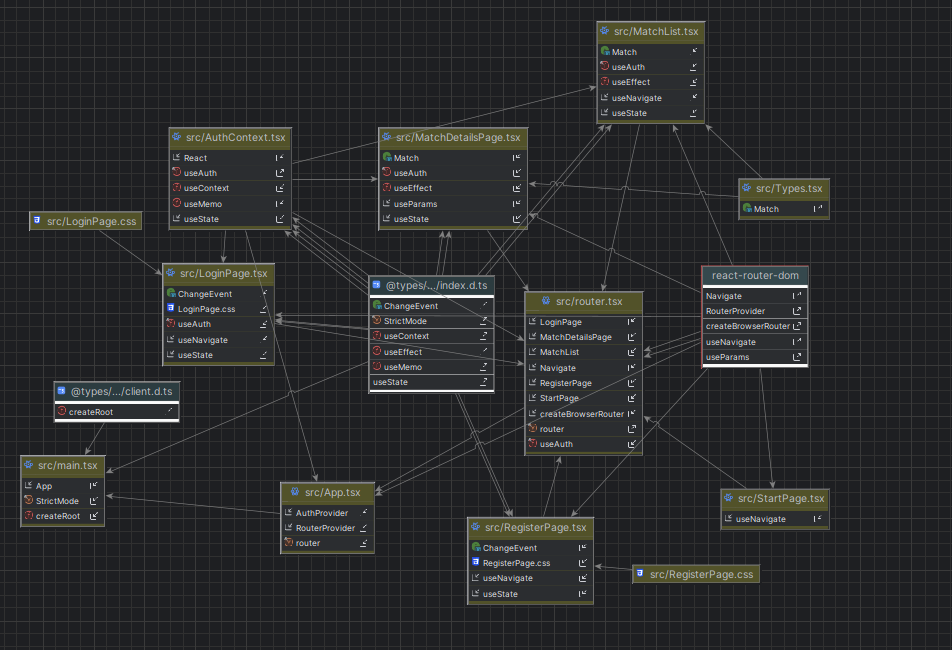
* Users browse matches (Match Service).
* Users log in/register (Auth Service).
* Authenticated users buy tickets (Ticket Service).

**Backend Communication Flow**

* The **Auth Service** issues tokens and verifies user authentication.
* The **Match Service** retrieves match details.
* The **Ticket Service** processes ticket purchases and triggers invoice generation via **Kafka**.
* The **Invoice Service** creates invoices and pushes them to **RabbitMQ**.
* The **Email Service** logs invoice details to simulate email sending.
* The **Nginx Load Balancer** distributes traffic between services.



Frontend Diagram:



Context Diagram:  
O imagine care conține text, captură de ecran, diagramă, linie

Conținutul generat de inteligența artificială poate fi incorect.

