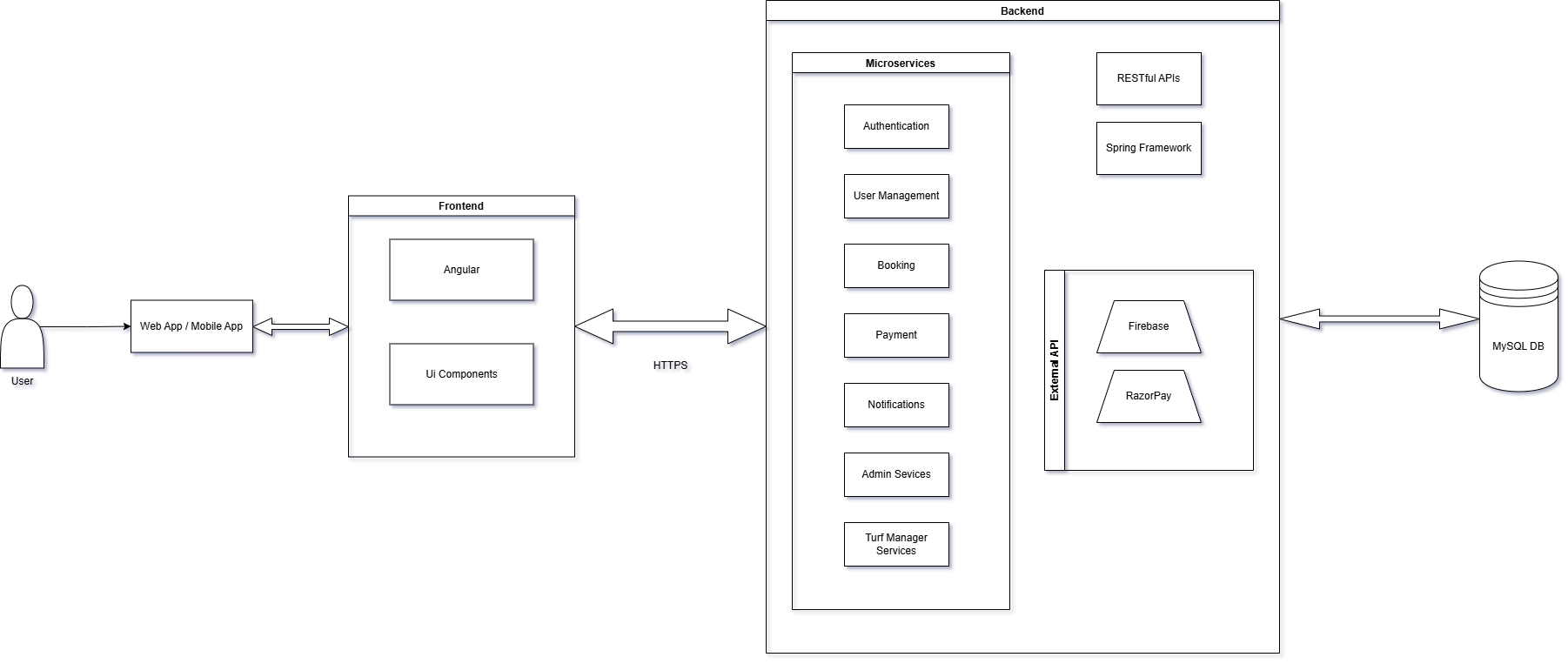
## **Turf Booking Management System Design Document**

### **1. Project Overview**

**Project Name**: Turf Booking Management  
**Version**: 1.0  
**Date**: 25th May 2024  
**Author**: Akhil Aduvala  
**Contact**: [aduvalaakhil@gmail.com](mailto:aduvalaakhil@gmail.com)

### **2. High-Level Architecture**



**Clients**: Mobile app, Web app  
  
**Backend Microservices**:

* Authentication
* UserManagement
* Booking
* Payment
* Notifications
* LocationandTurf
* AdminServices
* ManagerServices

**Database**: MySQL

**External Services**:

* Payment Gateway: PayPal or Razorpay
* Notification Service: Firebase

**Components**:

* Backend: Controller, Service, DAO, DAOImpl, DTO, Mapper, Task
* Frontend: Header, Footer, Login, Register, UserProfile, Booking, Payment, Notification, AdminDashboard, ManagerDashboard

### **Core Microservices and Their Responsibilities**

1. **Authentication Service**
   * **Purpose**: Handle user authentication, issuing and validating tokens.
   * **Endpoints**: Login, Register, Token Refresh, Logout.
2. **User Management Service**
   * **Purpose**: Manage user profiles and roles.
   * **Endpoints**: CRUD operations for users, roles assignment, profile management.
3. **Booking Service**
   * **Purpose**: Manage bookings, scheduling, and related operations.
   * **Endpoints**: Create Booking, View Booking, Update Booking, Cancel Booking.
4. **Payment Service**
   * **Purpose**: Handle payment processing and integration with payment gateways.
   * **Endpoints**: Process Payment, Payment Status, Payment History.
5. **Notification Service**
   * **Purpose**: Manage sending notifications to users.
   * **Endpoints**: Send Notification, Notification Status, Notification History.
6. **Location and Turf Service**
   * **Purpose**: Manage locations and turfs, including availability and pricing.
   * **Endpoints**: CRUD operations for locations, CRUD operations for turfs, Update Turf Availability and Pricing.
7. **Admin Service**
   * **Purpose**: Perform administrative tasks, manage system configurations, oversee all entities.
   * **Endpoints**: Manage Users, Manage Roles, System Configuration, View All Bookings, Generate Reports.
   * **Roles**: Admin can have overarching access across services.
8. **Manager Service**
   * **Purpose**: Manage specific turfs, handle bookings for assigned turfs, adjust availability and pricing for turfs they manage.
   * **Endpoints**: View Managed Turfs, Update Turf Details, View and Manage Bookings for assigned turfs.
   * **Roles**: Turf Manager with access limited to their turfs.

### **3. Detailed System Design**

#### **3.1 Database Schema**

CREATE TABLE User (

user\_id INT PRIMARY KEY AUTO\_INCREMENT,

userFName VARCHAR(50),

userLName VARCHAR(50),

mobile VARCHAR(15),

email VARCHAR(100) UNIQUE NOT NULL,

usergroupId INT,

password VARCHAR(100) NOT NULL

);

CREATE TABLE UserGroup (

userGroupId INT PRIMARY KEY AUTO\_INCREMENT,

userGroupName VARCHAR(50) UNIQUE NOT NULL,

FOREIGN KEY (locationId) REFERENCES Locations(locationId)

);

CREATE TABLE Turf (

turfId INT PRIMARY KEY AUTO\_INCREMENT,

turfName VARCHAR(100) NOT NULL,

locationId INT,

sizeDetails VARCHAR(100),

pricing DECIMAL(10, 2),

availability BOOLEAN,

minAdvance INT,

FOREIGN KEY (locationId) REFERENCES Locations(locationId)

);

CREATE TABLE Location (

locationId INT PRIMARY KEY AUTO\_INCREMENT,

locationName VARCHAR(100) UNIQUE NOT NULL,

address TEXT,

contactInfo VARCHAR(100)

);

CREATE TABLE Booking (

bookingID INT PRIMARY KEY AUTO\_INCREMENT,

userId INT,

turfId INT,

bookingDate DATE,

slot VARCHAR(50),

bookingstatus VARCHAR(50),

FOREIGN KEY (userId) REFERENCES User(user\_id),

FOREIGN KEY (turfId) REFERENCES Turf(turfId)

);

CREATE TABLE Payment (

paymentId INT PRIMARY KEY AUTO\_INCREMENT,

bookingId INT,

amountPaid DECIMAL(10, 2),

amountDue DECIMAL(10, 2),

paymentStatus VARCHAR(50),

FOREIGN KEY (bookingId) REFERENCES Booking(bookingID)

);

CREATE TABLE Notification (

notificationId INT PRIMARY KEY AUTO\_INCREMENT,

userId INT,

bookingId INT,

type VARCHAR(50),

notificationStatus VARCHAR(50),

FOREIGN KEY (userId) REFERENCES User(user\_id),

FOREIGN KEY (bookingId) REFERENCES Booking(bookingID)

);

#### **3.2 API Endpoints**

**Authentication API**:

* **POST /api/auth/register**: Register a new user (public access).
* **POST /api/auth/login**: Login an existing user.
* **POST /api/auth/change-password**: Change user password.

**User Management API**:

* **GET /api/users/profile**: Get user profile.
* **PUT /api/users/update-profile**: Update user profile.
* **GET /api/users/get-bookings**: Get user bookings.

**Bookings API**:

* **POST /api/bookings/new-booking**: Create a new booking.
* **GET /api/bookings/bookings/{bookingId}**: Get booking details by ID.
* **GET /api/bookings/get-slots/{locationId, turfId}**: Get available slots for a specific location and turf.

**LocationandTurf API**:

* **GET /api/locationandturf/get-location**: Get locations.
* **GET /api/locationandturf/get-turfs/{locationId}**: Get turfs for a location.
* **PUT /api/locationandturf/updatePrice/{turfId, price}**: Update turf price.

**Notification API**:

* **POST /api/notification/send-notification/{BookingId, type}**: Send notification.

**Admin API**:

* **POST /api/admin/add-users**: Add a new user (admin access).
* **PUT /api/admin/get-users/{userId}**: Get user details.
* **DELETE /api/admin/delete-users/{userId}**: Delete a user.
* **POST /api/admin/add-usergroup**: Add a new user group.
* **PUT /api/admin/get-usergroup/{usergroupid}**: Get user group details.
* **DELETE /api/admin/delete-usergroup/{usergroupid}**: Delete a user group.
* **POST /api/admin/add-locations**: Add a new location.
* **PUT /api/admin/get-locations/{locationId}**: Get location details.
* **DELETE /api/admin/locations/{locationId}**: Delete a location.
* **POST /api/admin/new-turf**: Add a new turf.
* **PUT /api/admin/update-turf/{turfId}**: Update turf details.
* **DELETE /api/admin/delete-turf/{turfId}**: Delete a turf.
* **GET /api/admin/booking-analytics**: Get booking analytics.

**Manager API**:

* **GET /api/manager/turfs**: Get turfs managed by the manager.
* **PUT /api/manager/turfs-availability/{TurfDTO}**: Update turf availability.
* **GET /api/manager/bookings/{locationId}**: Get bookings for managed locations.
* **PUT /api/manager/updatebookings/{bookingId}**: Update booking details.

#### **3.3 Service Design**

* **Auth Service**: Handles registration, login, and password management.
* **User Service**: Manages user profiles and bookings.
* **Booking Service**: Handles turf booking operations.
* **Payment Service**: Manages payment processing.
* **Notification Service**: Handles notifications for bookings.
* **Location and Turf Service**: Location and turf details, pricing, and availability.
* **Admin Service**: High-level administrative functions (manage users, roles, configurations).
* **Manager Service**: Turf-specific management (manage bookings, availability, and pricing for assigned turfs).

#### **3.4 Security Design**

* **Spring Security**: Implemented for authentication and authorization.
* **OAuth2.0 and JWT**: Used for secure API access.
* **Firewall**: Cloud-provided firewall or WAF.
* **SSL/TLS**: Certificates from Let's Encrypt for secure communication.

#### **3.5 Logging and Monitoring**

* **Spring Boot Actuator**: For health checks and monitoring.
* **Logback**: For centralized logging.
* **ELK Stack**: For log aggregation and analysis.
* **Prometheus and Grafana**: For metrics and monitoring.

#### **3.6 Messaging Queue**

* **RabbitMQ**: For handling asynchronous messaging.
* **Apache Kafka**: As an alternative for high throughput requirements.

#### **3.7 API Gateway**

* **Kong API Gateway**: For managing and routing API requests.

### **4. Deployment**

* **Hosting**: AWS
* **CI/CD**: Jenkins
* **Load Balancer**: AWS Application Load Balancer (if needed)
* **Auto Scaling**: AWS EC2 Auto Scaling Groups (if needed)

### **5. Docker and Kubernetes**

#### **5.1 Docker**

* **Purpose**: Containerize the microservices for consistent environments across development, testing, and production.
* **Dockerfile**: Each microservice will have its own Dockerfile to define its container.
* **Docker Compose**: Used for defining and running multi-container Docker applications for local development.

**Example Dockerfile for a Spring Boot Microservice**:

dockerfile

# Use the official image as a parent image

FROM openjdk:11-jre-slim

# Set the working directory

WORKDIR /app

# Copy the jar file

COPY target/microservice.jar microservice.jar

# Expose the port the app runs on

EXPOSE 8080

# Run the jar file

ENTRYPOINT ["java", "-jar", "microservice.jar"]

#### **5.2 Kubernetes**

* **Purpose**: Orchestrate Docker containers to ensure high availability, scalability, and management.
* **Kubernetes Cluster**: Deployed on AWS using EKS (Elastic Kubernetes Service).
* **Components**:
  + **Pods**: The smallest deployable units, representing a single instance of a running process in a cluster.
  + **Services**: Expose the microservices running in Pods.
  + **Deployments**: Manage the deployment of Pods, ensuring the desired number of replicas.
  + **Ingress**: Manage external access to the services, typically HTTP.

**Example Kubernetes Deployment YAML**:

yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: user-management

spec:

replicas: 3

selector:

matchLabels:

app: user-management

template:

metadata:

labels:

app: user-management

spec:

containers:

- name: user-management

image: user-management:latest

ports:

- containerPort: 8080

env:

- name: SPRING\_PROFILES\_ACTIVE

value: prod

---

apiVersion: v1

kind: Service

metadata:

name: user-management

spec:

type: ClusterIP

ports:

- port: 80

targetPort: 8080

selector:

app: user-management

---

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: user-management-ingress

spec:

rules:

- host: user-management.example.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: user-management

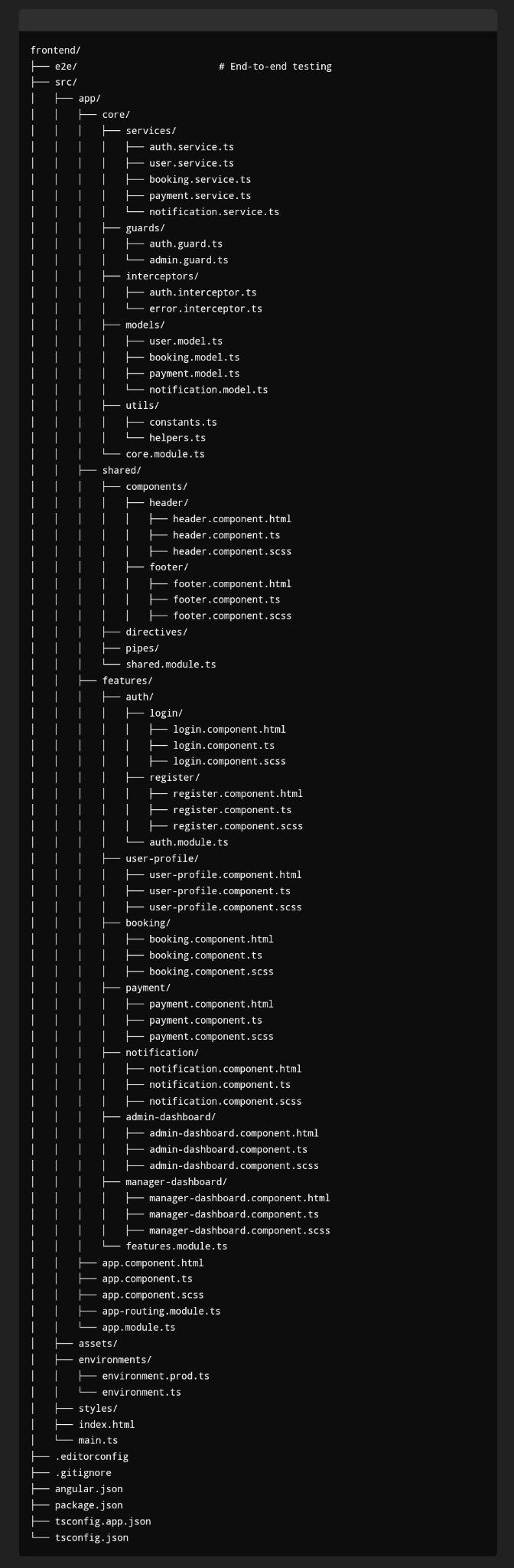
port:

number: 80

### 6. **User Roles and Permissions:**

* **Admin**: Access to all services. Can manage users, roles, locations, turfs, bookings, payments, and notifications.
* **Turf Manager**: Limited access to manage specific turfs and their bookings.
* **User**: Limited access to manage their profile, make bookings, and view notifications.

### **7. Project Structure**



#### **Core Module**

* **services/**: Contains all the service classes like auth.service.ts, user.service.ts, etc.
* **guards/**: Contains route guards like auth.guard.ts, admin.guard.ts, etc.
* **interceptors/**: Contains HTTP interceptors for handling authentication, errors, etc.
* **models/**: Contains TypeScript interfaces or classes for the application data models.
* **utils/**: Contains utility functions and constants.
* **core.module.ts**: The core module that imports and provides core services and features used throughout the app.

#### **Shared Module**

* **components/**: Contains shared components like header, footer, etc.
* **directives/**: Contains shared directives.
* **pipes/**: Contains shared pipes.
* **shared.module.ts**: The shared module that imports and exports shared components, directives, and pipes.

#### **Features Module**

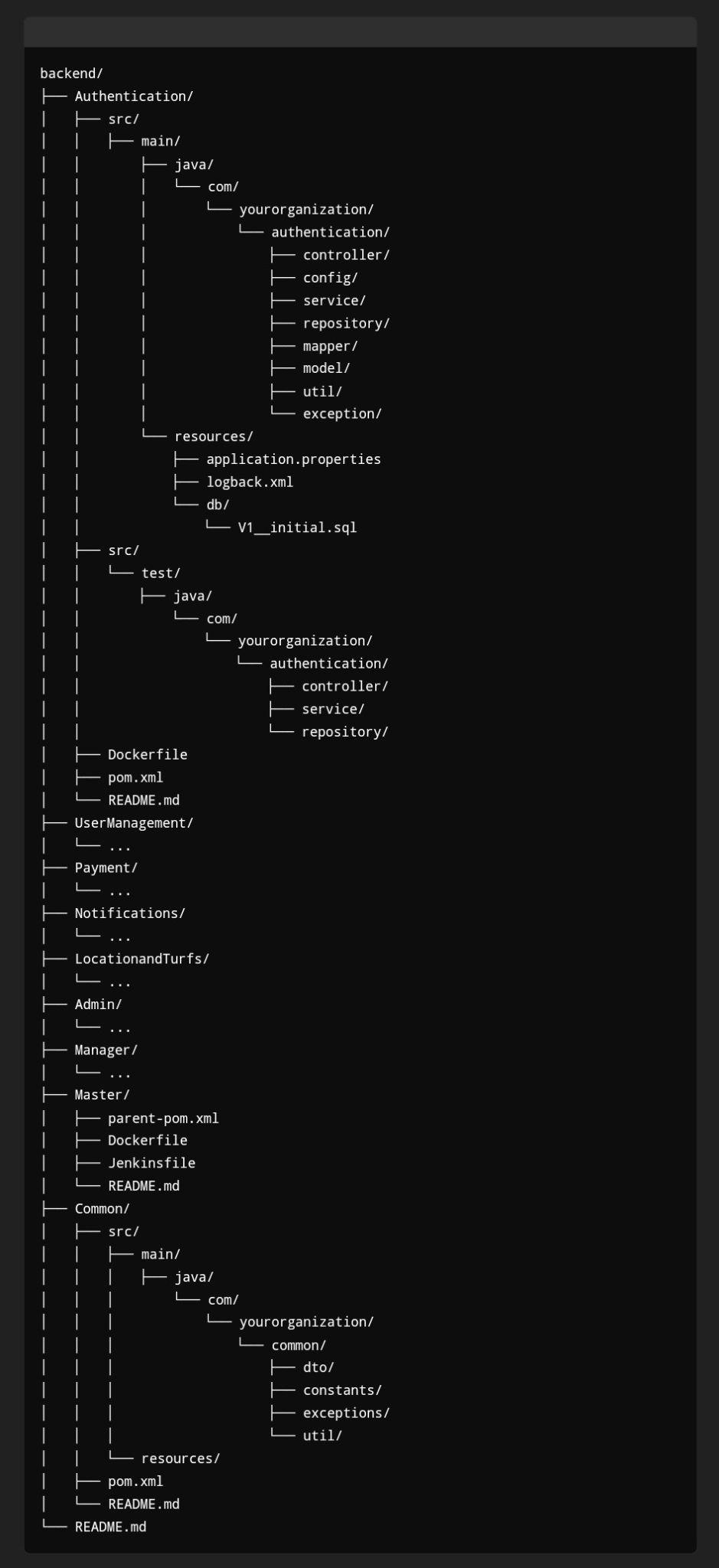
* **auth/**: Contains authentication-related components like login and register.
  + **login/**: Contains files for the login component.
  + **register/**: Contains files for the register component.
* **user-profile/**: Contains files for the user profile component.
* **booking/**: Contains files for the booking component.
* **payment/**: Contains files for the payment component.
* **notification/**: Contains files for the notification component.
* **admin-dashboard/**: Contains files for the admin dashboard component.
* **manager-dashboard/**: Contains files for the manager dashboard component.
* **features.module.ts**: The features module that imports and exports feature components.

#### **App Module**

* **app.component.html, app.component.ts, app.component.scss**: Main application component files.
* **app-routing.module.ts**: Routing configuration for the application.
* **app.module.ts**: The root module that imports core, shared, and feature modules.

#### **Other Directories**

* **assets/**: Contains static assets like images, fonts, etc.
* **environments/**: Contains environment-specific configuration files.
* **styles/**: Global styles for the application.
* **index.html**: The main HTML file for the application.
* **main.ts**: The main TypeScript file that bootstraps the Angular application.



#### **Microservices (e.g., Authentication, UserManagement, etc.)**

Each microservice folder follows the structure as discussed, with directories for,

**Controller:** Contains REST controllers that handle incoming HTTP requests and send responses.

**Config:** Contains configuration classes, such as security configurations, bean definitions, and other settings.

**Service:** Contains the business logic of the application. Services typically interact with repositories and other services.

**Repository:** Contains Spring Data JPA repositories for database operations, replacing traditional DAO and DAOImpl patterns.

**Mapper:** Contains classes and interfaces for mapping between different object models (e.g., DTOs and entities).

**Model:** Contains domain model classes, such as JPA entities.

**Util:** Contains utility classes and helper methods used across the service.

**Exception:** Contains custom exception classes and global exception handlers.

**resources/db:** Contains database migration scripts, queries, typically managed by Flyway or Liquibase00000000000000000

**test:** Contains unit and integration tests organized by functionality.

Each microservice also includes its own,

**Dockerfile:** Defines the Docker image for the microservice.

**pom.xml:** Maven configuration file.

**README.md:** Documentation for the microservice.

#### **Master**

* **parent-pom.xml**: Parent POM file for managing dependencies and plugin configurations common to all microservices.
* **Dockerfile**: Docker configuration for building and deploying the entire application.
* **Jenkinsfile**: Configuration for CI/CD pipeline.
* **README.md**: Documentation for the master project.



#### **Common**

* **dto**: Data Transfer Objects shared across all microservices.
* **constants**: Application-wide constants.
* **exceptions**: Common exception classes used across microservices.
* **util**: Utility classes and methods shared across services.