**Design Doc – BookMySport**

Design Document: User Registration and Authentication System

**1. Introduction**

This document outlines the design for the **User Registration and Authentication** system within the sports management platform. The system handles user registration, authentication, and role-based access control (RBAC) to ensure that users (admins, court owners, coaches, and regular users) can interact with the system securely and according to their roles.

**2. Objectives**

* Provide a secure method for user registration and login.
* Implement role-based access control (RBAC) to restrict certain operations based on user roles.
* Ensure secure password storage and management.
* Facilitate password recovery via email-based password reset.
* Provide JWT-based authentication for stateless user sessions.

**3. System Components**

**3.1. API Endpoints**

The system will provide a set of RESTful APIs to handle registration, authentication, and role management.

**4. API Specifications**

**4.1. User Registration**

* **Description**: Allows a new user to register an account by providing basic details (name, email, password) and selecting a role.
* **Endpoint**: POST /auth/register
* **Method**: POST
* **Request Body**:

{

"name": "John Doe",

"email": "john.doe@example.com",

"password": "securepassword123",

"role": "user"

}

**Response**:

* 201 Created for success:

{

"message": "User registered successfully",

"userId": "123456"

}

400 Bad Request for errors (e.g., email already exists, invalid role):

{

"message": "Email is already registered"

}

**Validations**:

* Ensure email is unique and valid.
* Password should meet minimum security standards (e.g., length, complexity).
* Role must be one of: user, admin, coach, court\_owner.

**4.2. User Login**

* **Description**: Authenticates a user by verifying email and password. Returns a JWT token on successful login.
* **Endpoint**: POST /auth/login
* **Method**: POST
* **Request Body**:

{

"email": "john.doe@example.com",

"password": "securepassword123"

}

**Response**:

* 200 OK for success:

{

"message": "Login successful",

"token": "jwt\_token\_here"

}

401 Unauthorized for invalid credentials:

{

"message": "Invalid email or password"

}

**JWT Details**:

* A stateless JWT token will be returned on successful login, allowing users to authenticate further requests without re-login.

**4.3. User Profile**

* **Description**: Allows a logged-in user to view their own profile details. Requires a valid JWT token in the request header.
* **Endpoint**: GET /auth/profile
* **Method**: GET
* **Request Headers**:
  + Authorization: Bearer {jwt\_token}
* **Response**:
  + 200 OK for success:

json

Copy code

{

"name": "John Doe",

"email": "john.doe@example.com",

"role": "user",

"created\_at": "2024-09-20"

}

* + 401 Unauthorized for missing/invalid token.

**4.4. Update User Profile**

* **Description**: Allows users to update their profile (e.g., name or password). Requires a valid JWT token.
* **Endpoint**: PUT /auth/profile
* **Method**: PUT
* **Request Headers**:
  + Authorization: Bearer {jwt\_token}
* **Request Body**:

json

Copy code

{

"name": "John Doe",

"password": "newsecurepassword123"

}

* **Response**:
  + 200 OK for success:

json

Copy code

{

"message": "Profile updated successfully"

}

* + 400 Bad Request for invalid input or errors.

**4.5. Role Assignment (Admin Only)**

* **Description**: Allows admins to assign or update roles for existing users.
* **Endpoint**: PUT /admin/users/{userId}/role
* **Method**: PUT
* **Request Headers**:
  + Authorization: Bearer {admin\_jwt\_token}
* **Request Body**:

json

Copy code

{

"role": "court\_owner"

}

* **Response**:
  + 200 OK for success:

json

Copy code

{

"message": "User role updated successfully"

}

* + 403 Forbidden for unauthorized role change attempts.

**4.6. User Logout**

* **Description**: Logs out the current user by invalidating their JWT token. This can be implemented client-side or with a token blacklist mechanism.
* **Endpoint**: POST /auth/logout
* **Method**: POST
* **Request Headers**:
  + Authorization: Bearer {jwt\_token}
* **Response**:
  + 200 OK for success:

json

Copy code

{

"message": "Logout successful"

}

* + 401 Unauthorized for missing/invalid token.

**4.7. Forgot Password**

* **Description**: Sends a password reset link to the user's registered email address.
* **Endpoint**: POST /auth/forgot-password
* **Method**: POST
* **Request Body**:

json

Copy code

{

"email": "john.doe@example.com"

}

* **Response**:
  + 200 OK for success:

json

Copy code

{

"message": "Password reset link sent"

}

* + 404 Not Found for unregistered emails.

**4.8. Reset Password**

* **Description**: Allows users to reset their password using the token sent in the password reset email.
* **Endpoint**: POST /auth/reset-password
* **Method**: POST
* **Request Body**:

json

Copy code

{

"token": "reset\_token\_here",

"password": "newpassword123"

}

* **Response**:
  + 200 OK for success:

json

Copy code

{

"message": "Password reset successful"

}

* + 400 Bad Request or 401 Unauthorized for invalid/expired token.

**5. Role-Based Access Control (RBAC)**

The system will enforce **role-based access control** using middleware to protect certain routes. Each user will have a role (e.g., admin, user, coach, court owner), and only users with specific roles will be allowed to access certain endpoints.

**Roles:**

* **Admin**: Can manage user roles, approve court listings, and view all system data.
* **User**: Can book courts and coaching sessions.
* **Court Owner**: Can list courts, manage availability, and view bookings.
* **Coach**: Can list availability, accept bookings, and manage coaching sessions.

**6. Security Considerations**

* **Password Hashing**: All passwords will be hashed using bcrypt before storage in the database.
* **JWT Authentication**: All protected routes will require a valid JWT token. Tokens should have an expiration time (e.g., 24 hours) to limit the lifespan of any compromised token.
* **HTTPS**: Ensure that all API requests are made over HTTPS to encrypt sensitive data during transmission.
* **Token Revocation**: Implement a method for token invalidation upon logout, using either a token blacklist or token expiry strategy.

**7. Database Schema**

**Users Table:**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | UUID | Unique user ID |
| name | String | User's name |
| email | String | User's email (unique) |
| password | String | Hashed password |
| role | Enum | Role (admin, user, coach, etc.) |
| created\_at | Timestamp | Timestamp of user creation |
| updated\_at | Timestamp | Timestamp of last update |

**Password Resets Table (if necessary for token-based resets):**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| id | UUID | Unique reset ID |
| user\_id | UUID | ID of the user requesting reset |
| token | String | Password reset token |
| created\_at | Timestamp | Timestamp of token creation |

**8. Conclusion**

This design document provides a detailed outline for building a robust and secure user authentication and role management system. Once implemented, it will serve as the foundation for managing different user roles and ensuring secure access to the various features of the sports management platform.