**Cricket Team Management System - Full Stack Challenge**

**Objective:**

Design and implement a Cricket Team Management System with both front-end and back-end components. The application should allow users to view, create, update, and delete players using RESTful API endpoints.

**Front-End (2 hours):**

Build the front-end of the Cricket Team Management system using a modern JavaScript framework (e.g., React, Angular, or Vue.js). The front-end should have the following features:

1. Display a list of Players with playerId, playerName, jerseyNumber and role.

2. Allow users to add new players, specifying the playerId, playerName, jerseyNumbre and role

refer below for detailed information.

3. Implement the ability to update players details.

4. Provide a user-friendly interface for managing players.

**Back-End (2 hours):**

Create the back-end of the Cricket Team Management System using a suitable technology stack (e.g., Java + Spring Boot or .NET Core). The back-end should have RESTful API endpoints for Team Management, including the following:

* **GET /api/players** : Retrieve a list of players.
* **POST /api/players**: Create a new player.
* **GET /api/players/:playerId**: Retrieve a specific Player by ID.
* **PUT /api/players/:playerId**: Update Player details.
* **DELETE /api/players/:playerId**: Delete a Player.

Implement validation and error handling for Player creation and updates.

**Player Information are:**

1. **Player ID :** A unique Identity for the Player.
2. **Player Name:** The Name of the Player.
3. **Jersey Number:**  The Number on Player’s Uniform/Jersey.
4. **Role :**  The Role of the Player in team (ex: Batsman , Bowler , Keeper , All Rounder)
5. **Total Matches** : Number of matches played by the Player
6. **Team Name** : The Name of the team
7. **Country/State Name :** The Name of the Country/State team belongs to.
8. **Description:** A brief description or summary of the player.

Sample Project Structure







|  |  |  |
| --- | --- | --- |
| 1 | Backend  Development | * Use Rest APIs (Springboot/ASP.Net Core WebAPI) to develop the services. * Use Java/C# latest features. * Use ORM with database. * Perform backend data validation. * Message input/output format should be in JSON (Read the values from the property/input files, wherever applicable). Input/output format can be designed as per the discretion of the participant. * Database connections and web service URLs should be configurable. * Follow Coding Standards with proper project structure. |

**Backend Constraints**

**Frontend Constraints**

|  |  |  |
| --- | --- | --- |
| 1 | **Layout and Structure** | Create a clean and organized layout for your pages. You can use a responsive grid system (e.g., Bootstrap or Flexbox) to ensure your design looks good on various screen sizes. |
| 2 | **Visual Elements** | |  | | --- | | **Form Fields:** Include input fields for gather information from the user. | | **Buttons/Link:** Design attractive and easily distinguishable buttons / Link | | **Error Messages:** Provide clear error messages for incorrect data entered in any form in application | | **Hover Effects:** Change the appearance of buttons and links when users hover over them. | |
| 3 | **Color Scheme and Typography** | Choose a color scheme that reflects your brand and creates a visually pleasing experience. Ensure good contrast between text and background colors for readability. Select a legible and consistent typography for headings and body text. |
| 4 | **Add form page** | |  | | --- | | **Form Fields:** Include fields for users to enter required information as given and any other relevant information. Use placeholders and labels to guide users. | | **Validation:** Implement real-time validation for fields (e.g., check email format) and provide immediate feedback for any errors.  **Form Validation:** Implement client-side form validation to ensure required fields are filled out correctly before submission. | |
| 5 | **Common to React/Angular** | * Use Angular/React to develop the UI. * Implement Forms, data binding, validations, error message in required pages. * Implement Routing and navigations. * Use JavaScript to enhance functionalities. * Implement External and Custom JavaScript files. * Implement Typescript for Functions Operators. * Any error message or exception should be logged and should be user-readable (and not technical). * Follow coding standards. * Follow Standard project structure. * Design your pages to be responsive so they adapt well to different screen sizes, including mobile devices and tablets |

**Frontend Constraints**