Football Team:

Introduzione:

Il progetto "Football Team" costituisce l'architettura backend di un'applicazione destinata alla gestione completa di dati calcistici. La sua versatilità ne consente diverse implementazioni finali:

- **Piattaforma Pubblica di Consultazione:** Può servire come fondamento per un portale di informazioni calcistiche, simile a piattaforme quali *Transfermarkt*.
  In questo scenario, l'applicazione esporrebbe dati accessibili al pubblico su squadre, giocatori, leghe e contratti, come risorsa informativa globale.
- **Strumento Gestionale Professionale:** Alternativamente, può essere impiegato come un software gestionale ad uso interno per lo staff di una squadra di calcio (dirigenza, allenatori, scout).
  In questo scenario, offrirebbe funzionalità avanzate per l'amministrazione dettagliata del roster, la gestione contrattuale, il monitoraggio delle prestazioni dei giocatori e la partecipazione alle competizioni, supportando le decisioni operative del club

---

Una volta delineate le idee di implementazione del progetto "Football Team", è necessario capire le decisioni di fondo che ne hanno guidato la realizzazione.
Questa sezione esplora le scelte adottate per costruire un sistema robusto, sicuro ed efficiente, gettando le basi per le funzionalità che seguiranno.

1. Scelte Progettuali:

Le seguenti scelte progettuali sono state adottate per ottimizzare l'architettura e le funzionalità del sistema:

- **Disaccoppiamento Entità/DTO**: La separazione tra le entità di dominio (`model`) e i DTO in ingresso/uscita previene l'esposizione di dettagli sensibili del database e permette un controllo granulare sul formato dei dati scambiati via API. Questo contribuisce a una maggiore robustezza e sicurezza dell'applicazione.
- **Iniezione delle Dipendenze con Spring**: L'utilizzo del meccanismo di Iniezione delle dipendenze (tramite `@Autowired` e annotazioni come `@Service`, `@Repository`, `@Controller`) delegata a Spring riduce la complessità nella gestione delle dipendenze tra i componenti.
- **Persistenza Dati con Spring Data JPA**: Per l'interazione con il database, è stato scelto Spring Data JPA, un framework che semplifica notevolmente le operazioni di persistenza.
  Abilitando l'uso di interfacce
  `Repository` con metodi denominati e velocizzando lo sviluppo e migliorando la leggibilità.
- **Gestione della Sicurezza con Spring Security e JWT**: La sicurezza è implementata tramite Spring Security, con autenticazione basata su JWT.
  Questo approccio garantisce che le API siano protette, permettendo solo agli utenti autenticati e autorizzati di accedere alle risorse.
- **Integrazione con Database MySQL**: MySQL è stato selezionato come database relazionale per la persistenza dei dati, data la sua affidabilità e ampia adozione nell'ecosistema delle applicazioni web.

Stabilite le scelte progettuali che definiscono l'applicazione, il passo successivo è assicurare che possa essere distribuita ed eseguita in modo uniforme e affidabile su qualsiasi ambiente.
Per superare le sfide legate alle dipendenze e alle configurazioni, si è scelto di adottare Docker come ambiente ideale, come dettagliato in questa sezione.

2. Ambiente di Esecuzione: Docker

L'adozione di Docker offre benefici significativi in un contesto di sviluppo e produzione.
Permette di creare ambienti di esecuzione isolati e riproducibili, eliminando le problematiche legate alle differenze di configurazione tra diverse macchine.
Un membro dello staff di una squadra di calcio può avviare l'intero stack (App e DB) con pochi e semplici comandi, senza dover installare dipendenze e senza problemi di compatibilità, garantendo un comportamento coerente in ogni ambiente.

- **Container per l'Applicativo Spring Boot**: L'intera applicazione backend è pacchettizzata in un'immagine Docker. Questo include il codice, le dipendenze e l'ambiente di runtime, assicurando che l'applicazione sia autocontenuta.
- **Container per il Database MySQL**: Anche il database MySQL è eseguito come un container Docker separato.

Una volta definita e garantita la portabilità tramite Docker, è necessario illustrare come l'applicazione sia strutturata internamente per funzionare in modo efficiente.
Questa sezione descrive l'architettura a strati che organizza i vari componenti del sistema, evidenziando il ruolo e le responsabilità di ciascun pacchetto.

3. Architettura dell' Applicazione:

L'organizzazione del progetto segue i principi dell'architettura MVC, tipici delle applicazioni Spring Boot, per garantire modularità, manutenibilità e scalabilità.
Ogni "package" ha un ruolo ben definito:

- `controller`: Questo strato lavora come interfaccia esterna dell'applicazione.
  Riceve le richieste HTTP dai client (es. browser web, applicazioni mobili) e le delega ai servizi appropriati per l'elaborazione. È responsabile della mappatura degli endpoint URL e della gestione del flusso iniziale delle richieste e delle risposte.
  - **Esempi:** `AuthController`, `PlayerController`, `TeamController`, `LeagueController`, `ContractController`, `UserController`. Questi gestiscono le richieste RESTful per le rispettive entità.
- `service`: Rappresenta lo strato di logica di business.
  Contiene l'intelligenza applicativa e le regole che governano le operazioni sui dati. I servizi coordinano le interazioni tra i controller e i repository, eseguendo le validazioni e le trasformazioni necessarie.
  - **Esempi:** `PlayerService`, `TeamService`, `LeagueService`, `ContractService`, `UserService`. Implementano le operazioni CRUD (Create, Read, Update, Delete) e la logica specifica per ciascuna entità.
- `repository`: Questo strato è l'interfaccia con il database.
  I repository, tramite JPA, forniscono un'astrazione per le operazioni di persistenza dei dati.
  Consentono di effettuare query e manipolazioni del database con un codice minimo, traducendo le operazioni Java in istruzioni SQL.
  - **Esempi:** `PlayerRepository`, `TeamRepository`, `LeagueRepository`, `ContractRepository`, `UserRepository`. Ogni repository gestisce un'entità specifica.
- `model`: Contiene la definizione delle entità.
  Ovvero la rappresentazione Java delle tabelle e delle relazioni presenti nel database relazionale. Queste classi sono annotate con JPA per la mappatura oggetto-relazionale.
  - **Esempi:** `Player`, `Team`, `League`, `Contract`, `User`, `Role`. Definiscono la struttura dei dati persistenti.
- `dto` **(Data Transfer Object)**: Questo modulo definisce gli oggetti utilizzati per il trasferimento dei dati tra gli strati dell'applicazione e i client esterni.
  Abbiamo sia
  `RequestDto` (per i dati in ingresso) che `ResponseDto` (per i dati in uscita). L'uso dei DTO assicura una chiara separazione tra il modello interno e il formato dei dati scambiati via API, migliorando la sicurezza e la validazione dei dati.
  - **Esempi:** `AuthRequestDto`, `AuthResponseDto`, `PlayerRequestDto`, `PlayerResponseDto`, `TeamRequestDto`, `TeamResponseDto`, `LeagueRequestDto`, `LeagueResponseDto`, `ContractRequestDto`, `ContractResponseDto`, `UserRequestDto`, `UserResponseDto`.
- `security`: Questo modulo è dedicato alla gestione della sicurezza dell'applicazione.
  Comprende le configurazioni per l'autenticazione (verifica identità utente) e l'autorizzazione (permessi). Utilizza Spring Security e i JWT per implementare un meccanismo di accesso sicuro.

---

4. Considerazioni Finali:

Il progetto "Football Team" mira a realizzare gli obiettivi prefissati: essere una piattaforma di consultazione completa per gli appassionati e uno strumento gestionale professionale indispensabile per i club.
L'attenzione alla modularità, alla sicurezza e alla portabilità tramite Docker garantisce che questa applicazione sia non solo tecnicamente solida, ma anche pronta a creare un valore tangibile nel mondo del calcio, sia per il pubblico che per gli addetti ai lavori.