# Relazione Tecnica - Project work Java

## Sistema di gestione della Biblioteca

### 1. Introduzione:

Il mio progetto “Sistema_Biblioteca” è un’applicazione backend in **Java con Spring Boot, con architettura MVC (Controller, Service, Repository, Model, DTO), con infrastruttura di containerizzazione in Docker, che ho sfruttato per la gestione dell’ambiente del database MySQL, includendo anche phpMyAdmin per una visualizzare in tempo reale il database;**

Per il mio progetto L’obiettivo è stato quello di creare un **gestionale per biblioteche**, che consenta:

- La gestione degli utenti, con ruoli **ADMIN** e **CLIENT**.
- La gestione del catalogo libri.
- Il tracciamento dei prestiti e delle restituzioni.

---

### 2. Struttura dell’applicativo:

Docker e Docker Compose contiene questa struttura di container:

- **Spring Boot App** – Fornisce API REST per la gestione di libri, utenti e prestiti.
- **MySQL** – Database relazionale per persistenza dati.
- **phpMyAdmin** – Interfaccia web per consultare e gestire il database.

L’architettura logica ha un’architettura MVC, composta con la seguente suddivisione:

- **Controller**: Gestisce le richieste HTTP in arrivo, coordina il flusso dell'applicazione e chiama i servizi necessari per elaborare i dati.
- Service: Contiene la **logica di business** dell'applicazione, ovvero le regole e le operazioni specifiche che non sono direttamente legate alla gestione

delle richieste HTTP o all'accesso ai dati. Il controller si interfaccia con il service per eseguire le operazioni.

- Repository: Si occupa dell'**accesso ai dati**, interagendo direttamente con il database. Il repository nasconde la complessità della persistenza dei dati agli altri componenti.
- Model: Rappresenta i dati e la logica associata. Sono le classi che mappano le tabelle del database.
- DTO e Mapper: Un oggetto semplice usato per trasferire dati tra i diversi strati dell'applicazione, riducendo la quantità di dati scambiati e disaccoppiando le classi interne dal mondo esterno.

---

# 3. Scelte Progettuali

**Architettura MVC**, per mantenere separazione tra logica di business, gestione delle richieste e persistenza.

**DTO e Mapper**, per evitare di esporre direttamente le entità e migliorare la sicurezza (non mostrando la password).

**Spring Security con JWT,** per l'autenticazione e autorizzazione basate su token, con ruoli utente e admin (CLIENT e ADMIN).

Spring Data JPA, per ridurre il codice e nella gestione database.

Docker e Docker Compose, per una configurazione automatica del database con tabelle e dati di esempio.

**Script SQL di inizializzazione,** per avere tabelle e dati pre-caricati ad ogni avvio.

**Gestione eccezioni,** per risposte API coerenti in caso di errori.

---

# 4. Funzionalità Principali

- **Gestione Utenti**
  - Registrazione e autenticazione con **password criptate (BCrypt)**.
  - Ruoli differenziati: **CLIENT** (prestito libri) e **ADMIN** (gestione catalogo e utenti).
- **Gestione Libri**
  - Operazioni CRUD (creazione, lettura, aggiornamento, eliminazione).

    - Ricerca per titolo, autore o categoria.
    - Aggiornamento stato di disponibilità in base ai prestiti.
- **Gestione Prestiti**
    - Creazione di un prestito solo se il libro è disponibile.
    - Registrazione data di prestito e data prevista di restituzione.
    - Segnalazione restituzione e aggiornamento disponibilità libro.
- **Sicurezza**
    - Autenticazione tramite token **JWT**.
    - Filtri di accesso basati sul ruolo utente.
    - Protezione delle password con hashing.

---

# 5. Funzionalità delle classi

- **Controller**
    - `LibraryController`: gestione CRUD libri e ricerca.
    - `PrestitoController`: gestione prestiti (creazione, restituzione, elenco prestiti attivi).
    - `AuthController`: login e gestione token JWT.
    - `UserController`: registrazione e gestione CRUD e ricerca degli utenti
- **Service**
    - `LibraryService` e `PrestitoController`: logica di gestione libri e prestiti, verifica disponibilità.
    - `UserService`: validazione credenziali e generazione token.
- **Repository**
    - `BookRepository`, `PrestioRepository`, `UserRepository`: Accesso alle entità.
- **Model**
    - `Book`: rappresenta un libro con titolo, autore, categoria, anno e disponibilità.
    - `User`: utente con credenziali, ruolo e dati personali.

- `Prestito`: relazione libro-utente con data prestito e restituzione.

- **DTO e Mapper**
  - `BookDTO`: Rappresenta un libro in formato semplificato come risposta al client.
  - `BookMapper`: converte oggetti `Book` in `BookDTO` e liste di `Book` in liste di `BookDTO`.
    sfrutta i metodi:
    - `toDTO(Book book)`: converte un singolo oggetto.
    - `toDTOList(List<Book> books)`: converte una lista di oggetti.
  - `LoginResponse`: invia dei dati come risposta al login per fornire al client un token e i dati dell'utente autenticato (email, role, token).
  - `PrestitoDTO`: rappresenta le informazioni relative a un prestito di un libro. Il costruttore riceve un oggetto `Prestito` e popola i campi, controllando che le entità `User` e `Book` siano presenti nel database.
  - `UserDto`: rappresenta i dati necessari per la creazione o l'aggiornamento di un utente. (name, surname, age, email, password, role (default "CLIENT").)
  - `UserResponseDto`: rappresenta i dati di un utente in risposta alle API per la consultazione degli utenti.

- **Security**:
  - `CustomUserDetailsService`: Recupera i dati dell'utente dal database a partire dall'email. Fornisce a Spring Security le informazioni necessarie per verificare credenziali e ruolo.
  - `JwtAuthenticationFilter`: Controlla ogni richiesta per verificare la presenza di un token JWT valido. Se il token è corretto, l'utente viene considerato autenticato.
  - `JwtUtil`: Si occupa di creare e leggere i token JWT.
    Contiene funzioni per generare il token dopo il login, estrarre informazioni come username e ruolo, e verificare se il token è scaduto.
  - `SecurityConfig`: Definisce le regole di accesso alle API:
    - Alcune rotte sono pubbliche (login, registrazione, elenco libri).
    - Altre sono riservate agli utenti autenticati o con ruolo **ADMIN**.

    - Utilizza la codifica **BCrypt** per proteggere le password.

---

# 6. Conclusione

Il progetto rappresenta una **base solida** per un sistema di gestione bibliotecaria, con tecnologie moderne e un'architettura facilmente estendibile.

Grazie a **Spring Boot**, **JWT**, **Docker** e **Spring Data JPA**, il sistema è:

- **Scalabile** – Facile da espandere con nuove funzionalità.
- **Sicuro** – Gestione robusta di autenticazione e autorizzazione.
- **Portabile** – Replicabile in qualsiasi ambiente grazie ai container.