Advanced Technique And Tools For Software Development

Sviluppo di una web application di un sistema di gestione di libri

Jacopo Vezzosi Mat. 7103062

Prof. Lorenzo Bettini

Dipartimento di Matematica e Informatica "Ulisse Dini"
Università degli Studi di Firenze
A.A. 2024/2025

1 Applicazione

Ho sviluppato un sistema completo per la gestione di libri utilizzando Spring Boot, con l'obiettivo di creare un'applicazione modulare ed estendibile. Il progetto si basa sull'entità principale Book, che rappresenta i libri presenti nel sistema e contiene informazioni fondamentali come id, titolo, autore, categoria e prezzo.

Per la gestione dei dati ho implementato una repository JPA, mentre la logica di business è stata incapsulata in un service layer, che gestisce operazioni come l'aggiunta di nuovi libri, l'aggiornamento dei dati esistenti e la ricerca dei libri in base a vari criteri.

L'applicazione è accessibile sia tramite API REST, utili per integrazioni e test automatici, sia tramite interfaccia web, che consente agli utenti di consultare e gestire i libri attraverso il browser.

Nel complesso, il sistema è progettato per essere chiaro, modulare e facilmente estendibile, con componenti separate che permettono di aggiungere facilmente nuove funzionalità o nuovi tipi di entità in futuro.

2 Tecniche e Framework

Per lo sviluppo dell'applicazione ho seguito un approccio Test-Driven Development (TDD), scrivendo prima i test unitari e poi implementando il codice necessario per farli passare. Questo metodo ha permesso di garantire la correttezza della logica di business e di progettare l'applicazione in modo modulare e facilmente manutenibile.

Per i test end-to-end (E2E), ho adottato invece un approccio Behavior-Driven Development (BDD) utilizzando Cucumber, definendo scenari chiari in linguaggio naturale che descrivono il comportamento atteso dell'applicazione. In questo modo, i test non solo verificano il funzionamento dell'applicazione nella sua interezza, ma fungono anche da documentazione eseguibile dei requisiti funzionali.

Di seguito si riportano i framework utilizzati:

- Eclipse (Spring Tools for Eclipse)
- Maven
- spring-boot-starter-test (JUnit, AssertJ, Json Path, Mockito, Hamcrest, JSONAssert, Spring Test and Spring Boot Test, HTMLUnit, Selenium, H2)
- JUnit Vintage Engine
- Thymeleaf
- DataJPA
- PIT
- Jacoco, Coveralls
- Docker
- Postgresql
- Git, Github, Github Actions
- SonarQube, SonarCloud
- Cucumber
- Spring Profiles

3 Design, Sviluppo e Testing

Nella progettazione dell'applicazione ho privilegiato modularità, chiarezza e manutenibilità. La scelta di organizzare il codice secondo il modello Controller → Service → Repository nasce dalla volontà di separare chiaramente le responsabilità: i controller gestiscono l'interazione con gli utenti e le richieste esterne, il service layer incapsula la logica di business, e le repository si occupano esclusivamente della persistenza dei dati. Questo approccio facilita l'estensione futura del sistema, ad esempio introducendo nuovi servizi, nuove entità o ulteriori modalità di accesso ai dati, senza compromettere il codice già esistente.

L'entità centrale Book è stata progettata per rappresentare in modo completo e coerente le informazioni principali di un libro, includendo attributi come id, titolo, autore, categoria e prezzo. La gestione dei dati tramite JPA consente di astrarre il livello di persistenza, semplificando l'interazione con il database e riducendo il rischio di errori legati alla gestione manuale delle query.

Per l'esposizione delle funzionalità, ho previsto due tipologie di controller: il REST controller permette l'interazione programmata con l'applicazione, facilitando integrazioni e automazioni, mentre il Web controller è dedicato all'interfaccia utente, collegando i dati dei libri alle pagine HTML. In particolare, l'applicazione presenta una pagina principale dove vengono visualizzati tutti i libri disponibili, con possibilità di modificarli direttamente quando presenti, e un'ulteriore pagina, dedicata all'aggiunta di nuovi libri. Questa distinzione migliora la flessibilità dell'applicazione e consente di mantenere separate le logiche di presentazione e di business, offrendo agli utenti un'interfaccia chiara e facilmente navigabile.

Dal punto di vista del testing, ogni componente dell'applicazione è stata testata in isolamento seguendo la tecnica del Test-Driven Development (TDD). In particolare, gli unit test sono stati progettati sfruttando la dependency injection, che permette di fornire a ciascun componente solo le dipendenze necessarie, sostituendo eventualmente le altre con mock o versioni semplificate. In questo modo, ogni classe viene testata indipendentemente dal comportamento delle altre, garantendo che la logica interna funzioni correttamente senza interferenze esterne.

Gli Integration Test, invece, verificano la corretta collaborazione tra i diversi componenti, interagendo con un database reale PostgreSQL. Il database viene avviato automaticamente prima dell'esecuzione dei test e fermato al termine, consentendo di controllare che repository, service e layer di persistenza lavorino in modo integrato e coerente.

Infine, i test end-to-end (E2E) sono stati implementati seguendo un approccio Behavior-Driven Development (BDD) con Cucumber. In questo caso, l'applicazione web viene avviata automaticamente, permettendo di eseguire scenari completi che simulano l'interazione reale dell'utente con l'interfaccia, verificando così il corretto funzionamento dell'intero sistema dal punto di vista dell'esperienza finale.

3.1 Difficoltà Riscontrate

Durante lo sviluppo dell'applicazione, ho incontrato alcune difficoltà legate principalmente all'aggiornamento alle ultime versioni di Spring Boot. Questo ha comportato la necessità di adattare le dipendenze, alcune delle quali avevano versioni più recenti o metodi deprecati, richiedendo modifiche al codice esistente e verifiche aggiuntive per garantire la compatibilità e il corretto funzionamento dell'applicazione.

Un'altra sfida significativa è stata la configurazione di Cucumber per i test end-to-end (E2E). In particolare, eseguire i test con un profilo separato in cui l'applicazione si avvia automaticamente non è stato semplice. Un problema specifico riguardava l'interazione con PostgreSQL: durante l'esecuzione dei test, venivano avviati due container sulla stessa porta — uno generato automaticamente dalla build e uno dall'avvio dell'applicazione — causando conflitti e impedendo l'esecuzione corretta dei test. È stato necessario progettare la configurazione in modo accurato per evitare il conflitto tra i container e garantire che i test E2E potessero eseguire scenari realistici senza interferenze.

3.2 Spring Profiles

Per quanto riguarda gli spring profiles, sono stati creati alcuni file application-xxx.yml contenenti ciascuno configurazioni per l'utilizzo di diversi database secondo i propri bisogni. Di seguito, le differenti configurazioni:

- application-dev.yml per l'utilizzo di postgresql
- application-prod.yml per l'utilizzo di mysql

Per utilizzare i profili, è necessario settarli nella configurazione della run: click destro sul progetto, Run As → Run configuration, Spring Boot App → bookmanager - BookmanagerApplication, → Arguments, in program arguments incollare –spring.profiles.active=xxx, dove "xxx" è il profilo. Inoltre, è necessario settare anche il file docker compose da utilizzare, al fine di avere un automatismo completo dove docker runna il container contente il database corretto, quindi in VM arguments incollare: -Dspring.docker.compose.file=compose-xxx.yml, dove "xxx" è il file docker

compose che si vuole utilizzare e che deve corrispondere con il profilo di spring. Ad esempio, se si vuole utilizzare postgre:

–**spring.profiles.active=dev** (Due "-" iniziali prima di spring)
-Dspring.docker.compose.file=compose-dev.yml

3.2.1 Profilo Maven e2e-tests

Nel profilo maven per l'esecuzione degli e2e-tests, dato che è configurato per far partire automaticamente l'applicazione, è necessario configurare il pom opportunamente cambiando gli argomenti del programma e della VM come mostrato sopra.

```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <executions>
        <execution>
            <id>pre-integration-test</id>
            <goals>
                <goal>start</goal>
            </goals>
            <configuration>
                <arguments>
                    <argument>--server.port=${tomcat.http.port}</argument>
                    <argument>--spring.profiles.active=postgresql</argument>
                </arguments>
                <jvmArguments>
                    -Dspring.docker.compose.file=compose-postgresql.yml
                </jvmArguments>
            </configuration>
        </execution>
        <execution>
            <id>post-integration-test</id>
            <goals>
                <goal>stop</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

4 Istruzioni Riproduzione Build

Per l'esecuzione della build, posizionarsi nella directory principale dove è presente il file pom.xml, avviare docker (i container partono automaticamente durante la build) e da linea di comando digitare:

mvn verify -Pjacoco,mutation-testing

Per l'esecuzione degli e2e tests con cucumber:

mvn verify -Pe2e-tests