CineMagazine - Università di Torino

Progetto TWEB - Anno 2024/2025

Damiano Cannizzaro

Introduzione

Questo report descrive lo sviluppo di **CineMagazine**, una single page web application realizzata per il corso di TWEB nell'anno accademico 2024/2025. L'applicazione permette la visualizzazione di informazioni su film, attori e premi Oscar attraverso un'architettura multi-server che integra database relazionali e NoSQL per fornire un'esperienza utente completa e interattiva nell'esplorazione del mondo cinematografico.

Il progetto rappresenta un esempio concreto di come diverse tecnologie moderne possano essere integrate efficacemente per creare una soluzione scalabile e performante. Durante lo sviluppo ho affrontato sfide significative che mi hanno permesso di approfondire concetti architetturali avanzati e best practices dello sviluppo full-stack.

Architettura del Sistema e Scelte Progettuali

Visione d'Insieme

Ho progettato CineMagazine seguendo un approccio a microservizi, suddividendo le responsabilità tra tre componenti principali. Questa scelta, inizialmente sembrata eccessiva per un progetto accademico, si è rivelata vincente per la modularità e manutenibilità del sistema. L'architettura implementata segue il pattern BFF (Backend for Frontend) con il MainServer che funge da orchestratore centrale, coordinando le comunicazioni tra i diversi servizi.

La separazione logica delle componenti riflette la natura diversificata dei dati gestiti dall'applicazione. I dati cinematografici strutturati, con le loro complesse relazioni tra film, attori, generi e premi, richiedevano la robustezza di un database relazionale. Al contrario, contenuti dinamici come recensioni, commenti degli utenti e messaggi di chat beneficiavano della flessibilità offerta da MongoDB.

MainServer - Il Cuore dell'Applicazione

Il MainServer, sviluppato con Node.js ed Express, rappresenta il punto di ingresso dell'applicazione e gestisce l'interfaccia utente attraverso Handlebars come template engine. Questa scelta mi ha permesso di implementare server-side rendering garantendo tempi di caricamento ottimali e una migliore indicizzazione SEO.

L'integrazione di Socket.io per le funzionalità real-time è stata una delle sfide più interessanti del progetto. Il sistema di chat permette agli utenti di discutere in tempo reale sui film, creando un senso di comunità attorno ai contenuti cinematografici. Ho implementato stanze tematiche per discussioni su film specifici e chat su canale comune generico, con persistenza dei messaggi .

JavaServer - Gestione dei Dati Strutturati

Il JavaServer, costruito con Spring Boot e JPA, gestisce tutti i dati cinematografici nel database PostgreSQL. Le entità principali includono Movies, Actors, Crew, Genres, Countries, Languages, Posters, Releases, Studios e Themes, tutte interconnesse attraverso relazioni complesse che riflettono la realtà dell'industria cinematografica.

L'implementazione del pattern Repository mi ha permesso di mantenere una chiara separazione tra la logica di business e l'accesso ai dati. L'utilizzo di JPA per l'ORM semplifica la gestione delle relazioni complesse tra le entità, mentre l'integrazione di Swagger fornisce una documentazione interattiva delle API REST esposte.

Una delle sfide più significative è stata ottimizzare le query per gestire grandi dataset senza impattare negativamente sulle performance. Ho implementato strategie di lazy loading e utilizzato indici appropriati per garantire tempi di risposta accettabili anche con migliaia di film e relazioni associate.

MongoDBServer - Flessibilità per i Dati Dinamici

Il MongoDBServer, sviluppato con Node.js e Mongoose, gestisce le collections principali: theOscarAwards per le informazioni sui premi, rottenTomatoesReviews per le recensioni, users per la gestione utenti e chatMessages per la chat real-time.

La scelta di MongoDB per questi dati è motivata dalla loro natura semi-strutturata e dalla necessità di evolverli rapidamente durante lo sviluppo. Le recensioni degli utenti, ad esempio, possono contenere campi variabili e metadata aggiuntivi che sarebbero difficili da modellare efficacemente in un database relazionale.

L'implementazione di controller specifici per ogni collection ha permesso di ottimizzare le query in base alle caratteristiche specifiche di ciascun tipo di dato, utilizzando proiezioni mirate per minimizzare il trasferimento dati non necessari.

Sviluppo del Frontend e User Experience

Approccio Tecnologico

Il frontend è stato sviluppato come Single Page Application utilizzando un mix di tecnologie moderne. Handlebars come template engine mi ha permesso di creare un sistema di rendering dinamico efficiente, mentre Bootstrap 5 ha fornito una base solida per componenti responsive e accessibili.

Ho scelto di utilizzare JavaScript vanilla piuttosto che framework più complessi per mantenere il controllo completo sulle performance e ridurre la complessità del bundle finale. Questa decisione si è rivelata azzeccata considerando la natura dell'applicazione e i requisiti prestazionali.

Struttura e Organizzazione

L'organizzazione delle views segue una struttura modulare con layout principale, pagine specifiche e partial riutilizzabili. Questo approccio ha facilitato la manutenzione del codice e permesso una rapida iterazione durante lo sviluppo delle diverse funzionalità.

Le pagine principali includono la homepage con carousel di film in evidenza, la sezione films per la ricerca e navigazione dei contenuti, le schede dettaglio per ogni film, la sezione actors per l'esplorazione del cast, e le pagine dedicate agli Oscar con ricerca per anno e film vincitori.

I partial componenti come navbar, footer, searchbar e le diverse tipologie di card (orizzontali, verticali, recensioni) sono stati progettati per essere riutilizzabili e personalizzabili, garantendo coerenza visiva e facilitando future espansioni dell'interfaccia.

Funzionalità Implementate

La ricerca avanzata rappresenta una delle funzionalità più complesse implementate. Permette agli utenti di filtrare i film per genere, anno di uscita, durata e rating, con risultati aggiornati dinamicamente e paginazione lato server. L'implementazione gestisce casi edge come query vuote, filtri multipli e ordinamenti personalizzati.

Il sistema di visualizzazione dettagli integra informazioni provenienti da entrambi i database, mostrando dati strutturati del film (da PostgreSQL) insieme alle recensioni dinamiche degli utenti (da MongoDB). Questo ha richiesto una coordinazione attenta tra i diversi servizi per evitare problemi di latenza e inconsistenza dei dati.

La chat real-time utilizza Socket.io client per fornire un'esperienza di messaggistica fluida e immediata. Gli utenti possono partecipare a discussioni pubbliche sui film o avviare una conversazione sul canare genrico degli Oscar, con notifiche push e indicatori di presenza in tempo reale sul terminale.

Sfide Tecniche e Soluzioni Implementate

Gestione delle Comunicazioni Inter-Server

Una delle sfide più complesse è stata coordinare le comunicazioni tra i tre server mantenendo la coerenza dei dati e gestendo situazioni di errore. Ho implementato un sistema di retry automatico per le chiamate HTTP fallite e meccanismi di circuit breaker per evitare cascading failures quando un servizio è temporaneamente non disponibile.

La gestione CORS tra i diversi domini ha richiesto configurazioni specifiche in ciascun server, con particolare attenzione alla sicurezza e alla prevenzione di richieste non autorizzate. Le configurazioni sono state centralizzate in file dedicati (`CorsConfig.java` e `corsConfig.js`) per facilitare la manutenzione.

Performance e Scalabilità

L'ottimizzazione delle performance è stata una preoccupazione costante durante lo sviluppo. Ho implementato diverse strategie per gestire efficacemente grandi volumi di dati: paginazione server-side per limitare il trasferimento dati, lazy loading per componenti non critici e query ottimizzate con indici appropriati.

La gestione della memoria JavaScript è stata ottimizzata attraverso tecniche di object pooling per le connessioni Socket.io e garbage collection manuale per oggetti di grandi dimensioni. Queste ottimizzazioni hanno permesso di mantenere performance accettabili anche con centinaia di utenti simultanei nella chat.

Sincronizzazione e Consistenza dei Dati

Mantenere la consistenza tra PostgreSQL e MongoDB ha rappresentato una sfida architetturale significativa. Ho implementato un sistema di eventi asincroni per sincronizzare dati correlati, come l'aggiornamento delle statistiche film quando vengono aggiunte nuove recensioni.

La gestione delle transazioni distribuite è stata semplificata utilizzando il pattern Saga, dove ogni operazione che coinvolge multiple basi dati viene suddivisa in passi compensabili, permettendo rollback parziali in caso di errori.

Integrazioni con API Esterne

Ho utilizzato bcrypt per l'hashing delle password e implementato meccanismi di refresh token per il
Per arricchire l'esperienza utente e fornire contenuti visivi di alta qualità, CineMagazine integra due

servizi API esterni che completani i dati dei Database locali.
la prima integrazione riguarda TMDB, che mi ha permesso di avere accesso alle immagini degli attori per dare più specificità alle ricerche.

La seconda integrazione coinvolge DiceBear, che mi ha permesso di generare avatar casuali ed unici per ogni utente connesso alla chat.

Risultati e Valutazione del Progetto

Requisiti Soddisfatti

Il progetto ha raggiunto tutti gli obiettivi tecnici prefissati: database PostgreSQL con JPA/Spring Boot per dati relazionali complessi, database MongoDB con Mongoose per contenuti dinamici, main server Express.js con template engine Handlebars, architettura a microservizi con comunicazione REST, integrazione Socket.io per funzionalità real-time, documentazione API completa con Swagger.

L'implementazione ha dimostrato la capacità di integrare efficacemente tecnologie diverse mantenendo performance accettabili e un'architettura scalabile. La documentazione del codice e delle API facilita la manutenzione e future espansioni del sistema.

Limitations e Possibili Miglioramenti

Nonostante il successo complessivo del progetto, ho identificato diverse aree di miglioramento. L'architettura attuale non implementa un sistema di cache distribuita come Redis, che potrebbe migliorare significativamente le performance in scenari di alto carico. Inoltre, manca un sistema di message queue per la comunicazione asincrona tra servizi, che renderebbe il sistema più resiliente.

Il sistema di autenticazione, non supporta l'uso di password, autenticazione multi-fattore né login tramite provider esterni. Il sistema di raccomandazione film è limitato e potrebbe beneficiare di algoritmi di machine learning per personalizzare meglio l'esperienza utente.
La ricerca è attualmente basata su corrispondenze testuali semplici e potrebbe essere migliorata con funzionalità di ricerca semantica avanzata. Inoltre, manca un sistema automatico di import di nuovi dati cinematografici da fonti esterne come TMDB o IMDB.

Apprendimenti e Riflessioni

Competenze Tecniche Acquisite

Lo sviluppo di CineMagazine mi ha permesso di approfondire significativamente la comprensione delle architetture distribuite moderne. Ho acquisito esperienza pratica con pattern architetturali come microservizi, BFF e Repository, comprendendo vantaggi e svantaggi di ciascun approccio.

L'esperienza con tecnologie diverse (Node.js, Spring Boot, MongoDB, PostgreSQL) mi ha fornito una visione completa dello stack tecnologico moderno e delle best practices specifiche per ciascuna piattaforma. La gestione delle comunicazioni real-time con WebSocket ha ampliato la mia comprensione dei protocolli di rete e delle sfide legate alla scalabilità.

Metodologie di Sviluppo

Il progetto ha sottolineato l'importanza di una pianificazione architetturale solida fin dalle prime fasi di sviluppo. La decisione di utilizzare tre server separati, inizialmente questionabile per la complessità

aggiuntiva, si è rivelata fondamentale per la modularità e testabilità del sistema.

Ho appreso l'importanza di implementare logging completo e monitoraggio delle performance fin dall'inizio, piuttosto che aggiungerli successivamente. Questi strumenti si sono rivelati invaluabili durante il debugging di problemi complessi che coinvolgevano multiple componenti.

La documentazione continua del codice e delle decisioni architetturali ha facilitato enormemente la manutenzione e l'aggiunta di nuove funzionalità durante le fasi avanzate del progetto.

Conclusioni

CineMagazine rappresenta un esempio completo di applicazione web moderna che integra efficacemente diverse tecnologie e pattern architetturali. Il progetto ha dimostrato che è possibile costruire sistemi complessi e performanti utilizzando un approccio modulare e ben progettato.

L'esperienza ha rafforzato la mia convinzione che le scelte architetturali iniziali abbiano un impatto fondamentale sulla manutenibilità e scalabilità a lungo termine del software. La separazione delle responsabilità tra i diversi servizi ha permesso di sviluppare, testare e deployare ciascuna componente indipendentemente.

Il successo del progetto deriva dalla combinazione di una progettazione attenta, implementazione incrementale e testing continuo. Ogni sfida tecnica affrontata ha rappresentato un'opportunità di apprendimento che ha contribuito al mio sviluppo professionale nel campo dell'ingegneria del software.

La natura distribuita dell'architettura ha evidenziato l'importanza di considerare aspetti come latenza di rete, gestione degli errori e resilienza del sistema fin dalle prime fasi di progettazione. Questi concetti sono fondamentali per lo sviluppo di applicazioni moderne scalabili e affidabili.

Setup e Configurazione Tecnica

Requisiti di Sistema

Per eseguire CineMagazine è necessario avere installati Node.js versione 14 o superiore, Java 17 o superiore, MongoDB 4.4 o superiore e PostgreSQL 13 o superiore. È inoltre consigliato avere almeno 4GB di RAM disponibili per garantire performance ottimali durante l'esecuzione simultanea di tutti i servizi.

Procedura di Avvio

L'avvio del sistema richiede una sequenza specifica per garantire la corretta inizializzazione delle dipendenze. Prima è necessario avviare i servizi di database (MongoDB e PostgreSQL), poi il MongoDBServer con `npm install && npm start` sulla porta 3001, seguito dal JavaServer con `mvn spring-boot:run` sulla porta 8081, e infine il MainServer con `npm install && npm start` sulla porta 3000.

Configurazioni Importanti

Il JavaServer richiede la modifica del file `application.properties` con le credenziali corrette del database PostgreSQL. Il MongoDBServer necessita della configurazione della connection string MongoDB nel file `database.js`. Le configurazioni CORS sono critiche e devono essere verificate nei file `CorsConfig.java` e `corsConfig.js` per garantire la corretta comunicazione inter-server.

È fondamentale creare manualmente il database PostgreSQL "CineMagazine" prima del primo avvio. Le credenziali di default sono configurate per username "postgres" e password "0000", ma possono essere modificate nei file di configurazione appropriati.

Bibliografia e Risorse

Documentazione e Riferimenti Tecnici

Durante lo sviluppo ho fatto riferimento extensively alla documentazione ufficiale di Spring Boot, Express.js, MongoDB Manual, Socket.io Documentation e Handlebars.js. Queste risorse sono state fondamentali per comprendere best practices e pattern di implementazione specifici per ciascuna tecnologia.

Le librerie utilizzate includono Axios per le chiamate HTTP asincrone, Mongoose come ODM per MongoDB, Spring Data JPA per l'ORM PostgreSQL, Bootstrap 5 per il framework CSS e Swagger UI per la documentazione interattiva delle API.

Fonti di Apprendimento

Oltre al materiale del corso di TWEB, ho utilizzato Stack Overflow per la risoluzione di problemi specifici, GitHub per esempi di implementazione e best practices, e strumenti di AI (ChatGPT/Copilot) per la comprensione di errori complessi, l'ottimizzazione delle query, l'apprendimento di best practices architetturali e il debugging di configurazioni CORS particolarmente complesse.

Dataset e Fonti Dati

I dataset sono stati processati e normalizzati per garantire la coerenza e l'integrità referenziale tra i diversi file.

La qualità e completezza dei dati ha rappresentato un fattore critico per il successo del progetto, richiedendo significativi sforzi di data cleaning e validazione durante le fasi iniziali di popolamento dei database.
