Progetto IUM-Tweb

Sommario

1. Introduzione

Il progetto, nella sua parte di Tweb, ha riguardato la creazione di un sito web a partire dai dati forniti che riguardasse film e serie TV. È stato svolto organizzando in tre grandi componenti: due server Express e uno Springboot.

In particolare, abbiamo due server limitati alla gestione di dati statici o dinamici (uno Express e l'altro Springboot) e il server principale adibito alla raccolta dati e alla visualizzazione della pagina Web.

2. Soluzione

2.1 Server Express Principale

Il server centrale è stato sviluppato utilizzando Express.js, un framework web per Node.js. Il design segue un approccio modulare, separando le responsabilità in diverse route.

In particolare, abbiamo strutturato il lavoro in maniera tale da avere un file javascript per ognuna delle pagine che vogliamo gestire, con relativo controller che si occupa di gestire le richieste al server Springboot nel caso di richiesta di dati statici. Il server principale gestisce esclusivamente l'instradamento delle richieste API.

2.1.1 Problematiche

Durante lo sviluppo, le principali difficoltà incontrate sono state:

- **Gestione dei carichi elevati**: Il server centrale potrebbe diventare un punto di congestione se riceve troppe richieste simultaneamente. Questo può rallentare il sistema se non gestito correttamente.
- **Gestione di dati ricevuti da entrambi i server per costruire una pagina web**: il server centrale deve raccogliere e combinare i dati provenienti dal server Express per i dati dinamici e dal server Spring Boot per i dati statici, creando una pagina web coerente. Questo processo può essere complesso e richiede la sincronizzazione tra le risorse di ciascun server. La difficoltà cresce se i dati non sono allineati o se ci sono differenze nei tempi di risposta dei server.

2.1.2 Requisiti

- Deve essere in grado di gestire un gran numero di richieste simultanee. (Soddisfatto grazie alla divisione modulare delle richieste)
- Deve supportare la gestione delle comunicazioni tra server differenti in modo efficiente. (Soddisfatto tramite uso di async/await e promise.all)

- Deve essere in grado di integrare correttamente i dati ricevuti da entrambi i server, creando una risposta unificata per il client. (Soddisfatto e testato → sezione Recensioni)

2.1.3 Limitazioni

Una volta richiesti i dati, in alcune situazioni il client filtra i dati ottenuti per renderizzare poche informazioni, ma potrebbe essere gestito tutto quanto lato server.

2.2 Creazione secondo server Express

La funzione del secondo server Express è quella di gestire dati dinamici presenti nel database MongoDB. Comunica con il server centrale per fornire i dati richiesti provenienti dalle tabelle relative alle recensioni o alle ultime uscite.

2.2.1 Problematiche

Nel processo di sviluppo delle funzionalità per l'inserimento e il recupero dei dati dalla tabella "*rotten_tomatoes_review_data*", sono emerse diverse difficoltà legate alla gestione delle richieste HTTP nel server Express, in particolare nell'interazione con il database. Tuttavia, identificando e affrontando queste problematiche, è stato possibile implementare soluzioni che migliorano la stabilità e l'efficienza del sistema.

2.2.2 Requisiti

Per far si che tutto giri correttamente, MongoDB deve essere configurato correttamente, in particolare è stato necessario un accesso forzato nei model per ogni tabella presente all'interno del database, specificandone il nome esatto.

2.2.3 Limitazioni

Alcune delle limitazioni riguardano sicuramente il numero di funzioni che il nostro sito web implementa, appunto limitato in quanto non è stato realizzato con lo scopo di essere reso visibile e messo online, ma appunto in quanto sito web che deve dimostrare le competenze e conoscenze acquisite seguendo il corso di IUM-Tweb (12 crediti).

2.3 Creazione server Spring Boot

Il server backend è stato implementato utilizzando **Spring Boot** per garantire una gestione efficiente e scalabile dei dati. Questo server si occupa della gestione delle informazioni statiche, come dati relativi ai film, agli attori e alla crew, basandosi sui dataset forniti dall'assignement.

Per l'interazione con il database, viene utilizzato **PostgreSQL** in combinazione con **JPA (Java Persistence API)**, facilitando l'accesso e la manipolazione dei dati.

La struttura del backend è stata organizzata secondo un'architettura ben definita, suddivisa in più livelli:

- **Entità**: Definisce le classi che rappresentano le tabelle del database, contenendo costruttori, metodi *getter* e *setter*.
- **Repository**: Implementa l'interazione diretta con il database utilizzando JPA, consentendo il recupero efficiente dei dati.
- **Service**: Contiene la logica applicativa e i metodi che richiamano il repository per gestire le richieste.
- **Controller**: Espone le API REST, gestisce le richieste HTTP e interagisce con il livello service.

Non tutte le tabelle seguono esattamente questa struttura, poiché alcune operazioni vengono centralizzate nella tabella **movies**, che funge da punto principale per le JOIN con altre tabelle. In particolare, è stato implementato un **DTO (Data Transfer Object)** che permette di combinare dati provenienti da più tabelle in un'unica risposta. Questo approccio consente, ad esempio, di ottenere in una sola chiamata le informazioni di un film e l'URL del relativo poster, ottimizzando le query e migliorando l'efficienza del trasferimento dati tra il database e il backend.

2.3.1 Problematiche

Durante lo sviluppo, sono emerse alcune difficoltà nell'esecuzione di **JOIN** tra tabelle con relazioni **uno-a-molti**. Un caso specifico riguarda l'associazione tra la tabella **movies** e le tabelle "**countries"** o "**themes"**, in cui un singolo ID di film doveva essere collegato a più righe delle rispettive tabelle. Per risolvere questa problematica, sono state adottate strategie di mappatura con JPA, tra cui l'uso di **@OneToMany** e **@ManyToOne**, garantendo una gestione ottimale delle relazioni.

2.3.2 Requisiti

- Strutturare e ottimizzare le API REST per una gestione efficiente delle richieste. (Soddisfatto)
- Garantire un accesso rapido e sicuro ai dati tramite il livello service (Soddisfatto).
- Ottimizzare le query per ridurre il numero di richieste al database. (Soddisfatto, abbiamo più query specifiche)

2.3.3 Limitazioni e Considerazioni

- L'implementazione richiede diverse configurazioni per gestire correttamente le relazioni tra le entità e migliorare le prestazioni.
- Il DTO gestisce solamente alcuni dei dati che vengono richiesti più spesso (come i poster), mentre il resto è gestito attraverso chiamate singole, che in future implementazioni potrebbero essere gestite direttamente nel DTO.

2.4 Creazione del sito web

Il sito web è stato sviluppato utilizzando **Handlebars** per la gestione dei template, consentendo un rendering dinamico e strutturato delle pagine. Per la comunicazione con il backend, viene utilizzata la libreria **Axios**, che permette di effettuare chiamate API in modo efficiente e asincrono. I dati vengono organizzati in sezioni per garantire una navigazione chiara e intuitiva.

2.4.1 Problematiche

Uno dei principali ostacoli riguarda la **quantità di dati da visualizzare**, che potrebbe rallentare il caricamento delle pagine. Abbiamo gestito il problema cercando di limitare al minimo i dati richiesti per le pagine iniziali di ogni sezione, in modo da rendere il sito il più fluido possibile

2.4.2 Requisiti

In caso di problemi il sito deve restituire pagine di errore dedicate.

2.4.3 Limitazioni

L'interfaccia potrebbe diventare lenta se non viene ottimizzato il caricamento dei dati o la gestione delle risposte da parte del backend.

3. Conclusioni

La suddivisione dei server in diverse componenti consente di gestire in modo ottimale i dati statici e dinamici, ma richiede un'accurata gestione della comunicazione e dei carichi di lavoro tra di essi. L'integrazione di Express e Spring Boot assicura una gestione separata e specializzata, ma può comportare sfide in termini di scalabilità e coerenza dei dati. Il lavoro è stato diviso cercando di rispettare il più possibile la divisione dei compiti, portando il lavoro avanti insieme e in contemporanea. Reputiamo dunque che la divisione risulti in:

- Agnese Andriani 50%
- Davide Giordano 50%

4. Informazioni aggiuntive

Nessuna informazione aggiuntiva.

5. Bibliografia

Abbiamo utilizzato le slide su moodle, i siti Bootstrap, W3School, JavaDoc e chatGPT per la generazione automatica di codice. In seguito alla ricezione del codice da parte dell'intelligenza artificiale è stato eseguito il seguente procedimento: lettura e comprensione del codice, valutazione della correttezza e infine adattamento al progetto, al fine di un utilizzo il più didattico possibile.