Relazione Tecnologie e Servizi Web

Componenti del gruppo:

* [Raffaele Melluso](mailto:raffaelemelluso01@gmail.com), matr. 1045495;
* [Dario Mangini](mailto:dario.mangini@edu.unito.it), matr. 1047397.

[Repository github solo per la parte di TWEB](https://github.com/RaffaeleMelluso/TWEB-PROGETTO_Mangini-Melluso)

**Introduzione**

Nel seguente report abbiamo suddiviso l’esposizione del lavoro in base alle macro task affrontate durante l’implementazione. Inizialmente, ci siamo concentrati sul front-end, seguito dall’integrazione di Socket.IO per la chat in tempo reale. Infine, abbiamo dedicato la maggior parte del nostro impegno alla parte più corposa, ovvero il back-end, che ha richiesto il maggior sforzo in termini di progettazione e implementazione. Per la parte del back-end abbiamo suddiviso il lavoro in base alle singole API per ogni pagina.

**Technical Tasks**

1. ***Sviluppo del frontend:***

**Solution:** per sviluppare l’interfaccia dell’applicazione abbiamo inizialmente creato le pagine in HTML, per poi modificarle successivamente in Handlebars, che permetteva una migliore modularità e manutenibilità. Abbiamo potuto così creare delle componenti riutilizzabili in più pagine del sito (ad esempio la navbar o le card per la ricerca dei film), permettendo così nel caso di modifiche di fare un unico intervento sul template.

Per la parte di stile e layout abbiamo usato una combinazione di Bootstrap e CSS personalizzato: Bootstrap ci ha permesso di creare un’interfaccia responsiva e ha permesso una rapida gestione degli elementi da disporre in griglia e delle componenti più comuni, CSS ci ha permesso di personalizzare ulteriormente la nostra pagina e gestire in maniera puntuale alcune componenti.

Abbiamo usato inoltre JavaScript per migliorare l’interattività, come l’invio di form senza refresh e aggiungere dinamicità nel caricamento dei dati.

**Issues:** abbiamo avuto alcuni problemi nell’interazione tra CSS e Bootstrap. Ci sono stati alcuni elementi che hanno prodotto dei bug dal punto di vista grafico, come il carousel, con i quali abbiamo integrato l’uso di intelligenza artificiale per capirne le cause e poter così risolvere.

**Requirements:** i requisiti sono stati rispettati in quanto abbiamo usato i linguaggi richiesti per creare pagine per il front-end, ossia Handlebars, CSS, Bootstrap, Javascript e HTML (quest’ultimo solo in fase iniziale).

**Limitations:** il front end carica interamente alcune liste di dati, il ché può rallentare il rendering delle pagine.

1. ***Socket.IO e chat:***

**Solution:** per implementare la chat in tempo reale tra fan ed esperti abbiamo usato la libreria Socket.IO, che consente comunicazioni bidirezionali via WebSockets. Nello specifico il modulo *socket\_io.js* gestisce la creazione e la gestione della stanza, l’invio e la ricezione dei messaggi e la gestione delle disconnessioni; il file *chat.js* connette l’utente alla chat (dopo aver compilato il form con nome, ruolo, argomento della stanza e numero della stanza che può anche essere generato), mostra i messaggi in tempo reale, e gestisce ulteriormente l’invio e la ricezione dei messaggi.

**Issues:** l’integrazione delle routes è stata all’inizio lenta a causa di nomi duplicati, successivamente sistemati con refactor. Collisione di attributi tra bootstrap e css che rendevano difficile nascondere la chat se non si aveva già compilato il form per l’accesso.

**Requirements:** utilizzo della libreria Socket.IO per realizzare la chat in tempo reale.

**Limitations:** i messaggi sono temporanei, potrebbero venire salvati in modo che siano persistenti.

1. ***Routes e query per l’ homepage (index):***

**Solution:** l'homepage dell’applicazione è progettata per offrire una piccola panoramica dei film disponibili sul sito, mostrando nel carosello i 5 film più recenti, mentre invece negli elementi sottostanti mostra i 3 film dal rating più alto. Le richieste vengono mandate dal main server tramite Axios al server Spring Boot (server statico), che effettua le query nel database. Nel primo caso il server statico controlla si assicura che i dati siano consistenti, ad esempio controllando che la data di rilascio sia precedente alla data odierna e restituisce una lista contenente l’id e il poster del film per riempire il carosello. Nella seconda route il server statico restituisce una lista contenente l’id, il nome, la descrizione, il rating e il poster dei 3 film che hanno il rating maggiore.

**Issues:** le difficoltà maggiori sono state riscontrate nella creazione della query, per avere certezza di consistenza tra i dati e una buona velocità di esecuzione di esse.

**Requirements:** il main server gestisce l’invio e la ricezione delle richieste e delle risposte, mentre è il server statico a interrogare il database e fornire i dati. Inoltre il server spring boot è ottimizzato per gestire grandi quantità di dati.

**Limitations:** ogni volta che si carica la pagina viene effettuata la query, per cui si potrebbe implementare una cache per ridurre il carico delle query frequenti.

1. ***Routes e query per la pagina “ultime uscite” (latest):***

**Solution:** la pagina delle ultime uscite permette agli utenti di vedere gli ultimi 10 film usciti negli U.S.A. Il main server invia la richiesta al server Spring Boot che effettua la query e si assicura che i dati restituiti siano consistenti, ad esempio gestendo il caso in cui il rating sia -1 (ossia il valore che abbiamo dato ai valori nulli durante la pulizia dei dati). Il server restituisce la lista con gli ultimi 10 film, che poi vengono posti in delle apposite card.

**Issues:** ci sono state alcune problematiche legate alla visualizzazione dei dati e alla loro renderizzazione all’interno delle card. Inoltre abbiamo avuto problemi per la pesantezza delle query che effettuavano join tra tabelle molto grandi.

**Requirements:** il client visualizza i dati, il main express si occupa delle richieste e il server Spring Boot effettua le query al database su Postgres.

**Limitations:** vengono visualizzate le ultime uscite negli Stati Uniti, si potrebbe applicare una maggiore scelta nel filtraggio dei dati facendo scegliere all’utente di quale paese visualizzare le ultime uscite o aggiungere altri parametri di ricerca.

1. ***Routes e query per la pagina dei film migliori (top):***

**Solution:** la pagina dei top film fornisce un elenco dei migliori film per genere, classificati in base al rating. Il main server invia la richiesta al server Spring Boot che effettua la query, ordinando i film per rating e suddividendoli nelle varie categorie. Il server restituisce una mappa contenente i dati dei 5 film migliori per ogni genere.

**Issues:** non ci sono state particolari complicazioni se non la necessità di comprendere meglio la comunicazione tra i server.

**Requirements:** il client visualizza i dati, il main express si occupa delle richieste e il server Spring Boot effettua le query al database su Postgres.

**Limitations:** i generi sono predefiniti nel controller di Spring Boot, avremmo potuto estrarre dinamicamente l’elenco dei generi dal database, ma non è stato fatto per non appesantire ulteriormente le richieste.

1. ***Routes e query per la pagina degli Oscar Awards (oscar):***

**Solution:** la pagina degli Oscar permette agli utenti di ricercare i vincitori e i candidati agli Oscar, filtrando i risultati per anno di cerimonia e nome del candidato o del vincitore. L’utente scrive nel form o l’anno o il nome della persona che vuole cercare, dopo di che il main server invia la richiesta inviando anche i parametri secondo cui fare la ricerca, e il server Spring Boot effettua la query considerando quali parametri ha ricevuto e il loro contenuto (questo poiché si può fare una ricerca mettendo solo l’anno e non il nome, e viceversa) e restituisce la lista dei risultati.

**Issues:** ci sono state alcune difficoltà nel capire come visualizzare i dati in maniera chiara per l’utente.

**Requirements:** il client visualizza i dati, il main express si occupa delle richieste e il server Spring Boot effettua le query al database su Postgres.

**Limitations:** nell’implementazione sono solo 2 i parametri di ricerca, avremmo potuto aggiungere altri parametri di ricerca o visualizzare grafici per mostrare informazioni sui dati.

1. ***Routes e query per la pagina della ricerca (search):***

**Solution:** la pagina di ricerca consente agli utenti di trovare film specifici filtrando i dati in base al nome del film o in base al genere. Anche in questo caso il main server invia i parametri della ricerca al server Spring Boot che interroga il database e restituisce la lista dei film che soddisfano i criteri di ricerca.

Quando si clicca sull'apposito bottone si passa alla pagina del film.

**Issues:** ci sono state alcune difficoltà legate alla libertà che il sito offre, dato che possono essere inseriti anche nomi parziali

**Requirements:** il client visualizza i dati, il main express si occupa delle richieste e il server Spring Boot effettua le query al database su Postgres.

**Limitations:** vengono eseguite ogni volta le query al database, si potrebbe incaricare un server per il mantenimento di una cache. Per questione di pesantezza si è limitato a un numero fisso di risultati per ricerca (20).

1. ***Routes e query per la pagina con i dettagli dei film (moviePage):***

**Solution:** nella maggior parte delle pagine quando vengono disposte le card con i film o per esempio nella homepage con il carosello si può passare con un click alla pagina del film in questione. In questo caso il main server invia le richieste a 2 server differenti: il Server Spring Boot effettua le query per restituire tutti i dettagli dei film, come gli attori, i registi, le lingue disponibili, lo studios, i paesi di produzione e le altre informazioni già presenti anche in molte card, mentre invece il secondo server Express interroga un database su MongoDB per cercare se sono presenti recensioni su quel film, cercando in base al nome del film, non essendoci un id che funge da chiave esterna. Inoltre è possibile scrivere una recensione, per cui il main server invia una POST al secondo server Express che in base ai parametri passati salva in MongoDB una nuova recensione.

**Issues:** ci sono state alcune problematiche nel visualizzare correttamente le recensioni, per cui è stato necessario approfondire meglio la comunicazione con tra i 2 server e l’interazione con MongoDB.

**Requirements:** il client visualizza i dati, il main express si occupa delle richieste,il server Spring Boot effettua le query al database su Postgres e il secondo server Express effettua le query sul database salvato su MongoDB.

**Limitations:** al momento i dati non sono aggiornati in tempo reale, se ci fosse qualche cambiamento sui dati dei film sarebbe necessario ricaricare la pagina, per cui si potrebbe implementare un meccanismo di aggiornamento automatico per i film.

**Conclusioni e divisione del lavoro**

Riteniamo che il lavoro sia stato diviso in maniera equa in base alle task da eseguire. Ci sono state alcune pagine dove un membro ha lavorato pressoché interamente a quella pagina: Mangini si è occupato quasi interamente della homepage, degli oscar e della pagina dei top film, Melluso della pagina della Chat , della pagina di ricerca e della pagina delle ultime uscite. In altri casi c’è stato un lavoro misto, ad esempio Mangini ha iniziato il front end delle ultime uscite ma poi ci ha lavorato principalmente Melluso per tutto ciò che riguardava la visualizzazione dei dati e la comunicazione con il database. Discorso simile per la pagina dei film, dove Mangini ha creato una prima versione e Melluso ha migliorato la visualizzazione e ha aggiunto la maggior parte dei dati presenti.

Nel lavorare sul database Spring Boot ci siamo divisi equamente la creazione delle entità e dei relativi controller, service e repository, andando a modificare i file in base alle query richieste dalle pagine. Ogni gestione delle routes assegnate come task è stata fatta dal membro del gruppo a cui era stata assegnata la sezione corrispondente.

Per il database Express che interagisce con MongoDB è stato creato da Melluso, Mangini ha completato la comunicazione con il main server per le recensioni e nuovamente Melluso si è poi occupato di bug fixing e la gestione dell’inserimento delle reviews.

Nel progetto abbiamo cercato di utilizzare il più possibile le cose imparate a lezione, mettendo finalmente insieme tutti i vari argomenti trattati durante il corso, come l’utilizzo di Express, di Axios, Spring Boot e via dicendo.

**Note**

Per la parte riguardante il database Postgres abbiamo utilizzato i dati puliti di Dario Mangini.

**Bibliografia**

<https://chatgpt.com/>

<https://www.w3schools.com>

<https://informatica.i-learn.unito.it/course/view.php?id=2950>