Report

Silvio Caprara, Yun Qing Zhou

Progetto TWEB IUM 2023-2024

Contents

1 Introduzione

Nel presente report verranno elencate le soluzioni adottate e le problematiche riscontrate durante lo sviluppo del progetto. L'implementazione è iniziata creando la repository Github e organizzando quest'ultima mediante le corrette directory come da consegna. Prima ancora di realizzare il prototipo per la piattaforma, è stato necessario uno studio dei dataset i quali sono stati divisi in "dataset frequentemente modificati" e "dataset raramente modificati". In ogni caso, prima della realizzazione di un server principale o dell'interfaccia grafica, è stato necessario configurare ed implementare i server che si interfacciano con le basi di dati. I task tecnici sono riportati in ordine di completamento.

2 Task

2.1 Server Java Springboot

2.1.1 Soluzione

Il server Java Springboot è stato implementato per primo in tempo relativamente breve, seguendo gli esempi visti durante le lezioni e mantenendo la corretta struttura JPA entità, service e repository. Inizialmente si era pensato ad un unico RestController per gestire tutte le rotte, le quali sono però state divise successivamente in un controller per entità, al fine di rendere il codice più ordinato.

2.1.2 Problemi

Per alcune query, abbiamo preferito restituire solo parte dei singoli record, a volte un solo campo. In questi casi abbiamo dovuto creare delle interfacce apposite che riflettessero queste strutture[1]. Per quanto non fosse stato possibile configurare un database con chiavi esterne, vista la mancata consistenza dei dati, alcuni dataset presentavano chiavi primarie composte. Per identificare univocamente i record di questi dataset, è stato necessario creare delle "Classi Id"[2].

2.1.3 Requisiti

L'implementazione del server Springboot rispetta la richiesta di interfacciarsi ad un database relazionale con DBMS PostreSQL.

2.1.4 Limitazioni

Le rotte del servizio sono state realizzate pensando alle informazioni che l'interfaccia grafica dovrà mostrare e supponendo che solo il backend dell'interfaccia stessa effettuerà le richieste. Per questo motivo, nonostante i parametri permettano una relativa flessibilità nelle richieste, non sono stati effettuati rigidi controlli.

2.2 Server MongoDB

2.2.1 Soluzione

Il server Express che si interfaccia con il database MongoDB e' stato realizzato seguendo lo schema MVC visto a lezione: mongoose schemas, routes e controller. Quest'ultimi si interfacciano con il database tramite Mongoose, con il quale eseguiranno le query. A volte è stato necessario aggiungere endpoint delle route durante lo sviluppo del frontend e dei jupyter notebooks. Vista la ripetitività del lavoro, è stata utilizzata l'IA generativa ChatGPT nella scrittura degli schemi di mongoose e swagger, fornendo come prompt un esempio di document preso da MongoDB per ogni collection.

2.2.2 Problemi

L'unico problema riscontrato è stato risolto facilmente ed riguardava la nomenclatura dei modelli di Mongoose. Abbiamo dovuto utilizzare nomi al singolare poiché Mongoose accede al database convertendo al plurale i nomi forniti al modello.

2.2.3 Requisiti

Il server Express Mongoose rispetta i requisiti di interfacciarsi a un database MongoDB che contiene i dati che cambiano spesso nel tempo.

2.2.4 Limitazioni

Le rotte del servizio sono state realizzate con l'interfaccia grafica in mente: per questo motivo non sono da ritenersi esaustive ma sono facilmente estensibili in vista di future esigenze.

2.3 React Single Page Application

2.3.1 Soluzione

La scelta nell'utilizzo del framework React è dovuta ad un utilizzo pregresso di quest'ultimo da parte dei membri del gruppo. Grazie ad una progettazione iniziale è stata lineare l'implementazione di componenti "pagina" e "contenuto", riutilizzando inoltre il layout e lo stile dei prototipi precedentemente realizzati. Una volta terminato lo sviluppo, l'applicazione react e' stata compilata tramite il comando build offerto dal modulo nodejs react-scripts, ottenendo file servibili staticamente dal main express server .

2.3.2 Problemi

I problemi riscontrati sono dovuti principalmente all'inconsistenza, incompletezza e separazione delle fonti di dati. Con l'idea di affidare l'elaborazione dei dati al lato client, avere un server centrale con il solo compito di inoltrare

le richieste e l'impossibilità di utilizzare operazioni di join, è stato in alcuni casi necessario concatenare e attendere la risoluzione di più promises per ottenere un'informazione completa e presentabile, complicando la logica di alcuni componenti. Durante l'implementazione, al fine di superare problemi dovuti alle CORS, è stato necessario definire un "proxy-middleware" per interrogare direttamente i server Springboot ed MongoDB. E' stato utilizzato ChatGPT a scopo didattico per visualizzare esempi e studiare la logica dietro le promise concatenate e array di promise.

2.3.3 Requisiti

Come da consegna, il framework è stato utilizzato solamente per la sue funzioni di single page application: le richieste ai server sono state tutte effettuate mediante axios requests.

2.3.4 Limitazioni

Con il presupposto che solo l'interfaccia grafica avrebbe effettuato le richieste ai servizi mediante la pressione o ricerca di contenuti da parte dell'utente, il numero di limitazioni è ridotto. E' però comune trovare sezioni con contenuti mancanti a causa dei dati. Mediante la "navigazione tramite url", è possibile tentare ad accedere a pagine con identificatori non esistenti.

2.4 Express Main Server

2.4.1 Soluzione

Il server express centrale è stato realizzato dopo aver reputata pressoché terminata l'implementazione del front-end. Con l'idea di avere semplici metodi di inoltro richieste, senza elaborazione di dati, è stato rapido creare due route principali al fine di dividere le query dirette ai due database servers e tante route interne quante sono quelle implementate nei vari server database. Con un server centrale funzionante, sfruttando il codice scritto durante l'esercitazione in aula, è stata realizzata la chat basata su SocketIO.

2.4.2 Problemi

I problemi riscontrati durante lo sviluppo riguardano principalmente la chat. Nella fase di configurazione iniziale, dopo l'installazione ed importazione dei moduli lato server, importare SocketIO lato client tramite CDN si è rivelato errato. Anche nel progetto React è stato necessario installare il modulo tramite npm.

2.4.3 Requisiti

Come richiesto, il server centrale svolge la funzione di fornire la single-page application e si comporta da interfaccia tra front-end e database servers. La

chat fornisce le istruzioni elementari di connessione ad una stanza e scambio di messaggi.

2.4.4 Limitazioni

Le rotte api disponibili sono copie di quelle presenti nei due database servers, non sono stati aggiunti filtri aggiuntivi, controlli o elaborazioni lato server. Non sono state implementate misure di sicurezza o complesse tecniche di autenticazione nella chat. Un utente può scegliere una stanza e collegarsi con qualsiasi username non in utilizzo sul server al momento della connessione, inoltre i messaggi inviati in precedenza non sono salvati dal server.

2.5 Analisi dei dati tramite Jupyter Notebooks

2.5.1 Soluzione

Dopo lo studio dei dati forniti, abbiamo deciso di creare tre notebooks: uno dedicato all'analisi di una competizione, uno dedicato all'analisi dei giocatori di un paese e uno dedicato ad un'analisi globale. Per ottenere i dati su jupyter notebooks ci interfacciamo con il Main Server Express tramite la libreria requests, con cui effetuiamo le query. I pacchetti utilizzati sono inclusi nel file enviroment di miniconda. Per velocizzare la creazione dei grafici abbiamo utilizzato l'IA generativa ChatGPT come supporto nella rielaborazione e preparazione dei dati.

2.5.2 Problemi

Abbiamo riscontrato molteplici problemi nella compatibilita' dei pacchetti di Python; abbiamo dovuto più volte eseguire downgrade o upgrade delle versioni per ottenere compatibilità tra i moduli. Altri problemi sono emersi nella rielaborazione dei dati, data la nostra inesperienza con il linguaggio e con pandas.

2.5.3 Requisiti

Abbiamo creato uno o piu' jupyter notebooks che ci permettono di richiedere e analizzare i dati come da specifica.

2.5.4 Limitazioni

I nomi dei paesi non corrispondono del tutto con la mappa a nostra disposizione, perdendo alcuni dettagli (ad esempio England, Scotland e Wales non sono presenti sulla mappa e sono stati normalizzati con il valore United Kingdom).

2.6 Conclusioni

Il progetto risulta sviluppato nella sua completezza rispettando la consegna e coprendo tutti i requisiti. Vista la scala del codice implementato e la diversità

dei servizi, nonostante i server siano stati realizzati "ad-hoc", abbiamo compreso l'utilità e la correttezza della realizzazione di sistemi modulari. Abbiamo inoltre avuto l'occasione di utilizzare Github per un progetto di scala più larga rispetto a quelli realizzati precedentemente, migliorando le nostre abilità cooperative. Dalla creazione dei Jupyter Notebook e dall'analisi dei dati abbiamo compreso quanto sia complesso lo studio e la presentazione dei dati, soprattutto la difficoltà che risiedono nella creazione di una narrativa interessante e non banale. Infine, con l'utilizzo di ChatGPT, abbiamo realizzato quanto le IA generative siano uno strumento potente e quanto possano aiutare e velocizzare il processo di sviluppo software quando viengono utilizzate correttamente, con prompt precisi e descrittivi.

2.7 Divisione del lavoro

Essendo il gruppo composto da soli due membri, è stato semplice dividere l'implementazione iniziale dei due server che si interfacciano ai database, i quali sono stati successivamente modificati da entrambi i membri in base alle necessità durante il proseguimento dello sviluppo. Il resto del lavoro è stato equamente diviso eseguendo commit di codice scritto in solitaria o, più frequentemente, mediante la funzione "code with me" dei programmi JetBrains.

2.8 Informazioni aggiuntive

Per poter eseguire il progetto, i server Nodejs devono essere aperti come progetti mediante WebStorm e il server Springboot deve essere aperto come progetto su Intellij. Per eseguire i Jupyter Notebooks bisogna importare l'enviroment miniconda con il file env.yml tramite il comando "conda env create -f percorso_file". Una volta aperto il progetto in Pycharm bisogna impostare l'enviroment appena importato chiamato "ium" come interprete Python nelle impostazioni del progetto.
Il database mongodb deve essere importato con il tool mongorestore reperibile sul sito di mongodb tramite il comando "mongorestore path_cartella_mongodump" eseguito in modalita' amministratore sul cmd. Il database postgresql deve essere importato creando un database chiamato "FootballXData" ed in seguito copiando ed eseguendo mediante il query tool le istruzioni presenti nel file .sql fornito.
Per eseguire il server Springboot tramite IntelliJ bisogna usare JDK 21 e impostare la JVM usata da Gradle a JDK 21 dalle impostazioni dell'IDE sotto la sezione build tools.

2.9 Bibiliografia

[1] *https://www.baeldung.com/mongodb-return-specific-fields*
[2] *https://www.baeldung.com/jpa-composite-primary-keys*