GoalNet360 Project - Davide La Rosa & Sofia Trucco - IUM_TWEB (12 crediti)

Questo report ha lo scopo di esplicitare/fornire, in dettaglio, per ognuno degli elementi cardine del nostro progetto, le scelte implementative adottate, i problemi riscontrati nella loro attuazione ed eventuali limitazioni.
In particolare, quindi, verrà fornita una panoramica abbastanza dettagliata dei Servers utilizzati e della gestione dei database.
I servers che abbiamo creato e utilizzato sono: Main Server (Express), Java Server (Spring Boot) e MongoDB Server (Express). Inoltre, è stata fornita una dettagliata documentazione Swagger per i servers Express.
Nel nostro percorso di sviluppo, abbiamo integrato ChatGPT-4 come risorsa di supporto, particolarmente utile in situazioni specifiche. Il suo impiego si è focalizzato principalmente sulla creazione di funzioni particolarmente sofisticate e sulla risoluzione di errori (ove non bastasse il tradizionale debugging). È importante sottolineare che, nonostante l'efficacia di ChatGPT-4, ogni soluzione proposta è stata rigorosamente valutata e modificata, per garantire che si integrasse in modo coerente nel nostro sistema, rispettando le esigenze specifiche del progetto.

MAIN SERVER
Il Main Server (localhost:3000) si interfaccia con due servers secondari, un Server Express (localhost:3001), il quale comunica con un database MongoDB per i dati dinamici del dataset fornito, e un Server Spring Boot (localhost:8080), il quale comunica con un database Postgre per i dati statici, in modo da soddisfare le richieste degli utenti.

1) SOLUZIONE: La struttura del nostro Main Server è strettamente legata alla logica adottata per il nostro sito GoalNet360, basata sull'utilizzo di filtri progressivi. Questa metodologia implica l'impiego di un'architettura di back-end dinamica, in grado di gestire richieste complesse e di filtrarle in modo efficiente. Abbiamo implementato algoritmi di ricerca e di categorizzazione dei dati che permettono agli utenti di navigare attraverso il contenuto in modo intuitivo, affinando progressivamente la ricerca attraverso parametri specifici. Questa struttura non solo migliora l'esperienza dell'utente, ma ottimizza anche le performance del server riducendo i tempi di caricamento e incrementando la velocità di risposta alle query complesse. Inoltre, abbiamo optato per un'architettura integrata, consolidando il server e il client. All'interno di una cartella 'public', abbiamo organizzato le risorse statiche: sottocartelle per HTML, JavaScript, Images e Stylesheet, gestendo le richieste del client direttamente. Questo approccio semplifica la gestione delle risorse, riduce la latenza migliorando i tempi di risposta, e facilita la manutenzione e il debugging, offrendo un ambiente di sviluppo più coeso e performante.

[1] : https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js
https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css

2) PROBLEMI RISCONTRATI: Durante lo sviluppo, abbiamo affrontato sfide legate principalmente alla gestione di un dataset esteso, dove il principale ostacolo era discernere se gli errori emergessero da problemi nel codice o dalla mancanza di dati. Questi sono stati superati attraverso tecniche di debugging approfondite. Nel lato client la complessità risiedeva nel garantire dinamicità nell'interfaccia utente e, nonostante l'uso di HTML puro anziché di framework più avanzati come EJS o React, abbiamo risolto assicurando il passaggio efficiente dei parametri attraverso URL, necessari per popolare dinamicamente gli elementi di selezione dei filtri nelle diverse pagine HTML.

3) RICHIESTE DA CONSEGNA: La nostra implementazione aderisce a tutte le richieste specificate.

4) LIMITAZIONI: E' opportuno evidenziare che, benché il sistema sia generalmente efficiente, potrebbe manifestarsi una certa latenza nelle situazioni in cui il server deve gestire query complesse, specialmente quando queste richiedono l'intersezione con altre query sull'ampio dataset fornito (ad esempio, elaborazioni che coinvolgono tutti i Games nel dataset, come per la creazione della Classifica per una certa Competition). Inoltre, l'uso di HTML puro, senza tecnologie front-end più dinamiche, potrebbe contribuire a un incremento dei tempi di risposta, specialmente in scenari di elevata interattività utente.

5) CONCLUSIONI: Nonostante il risultato ottenuto sia soddisfacente, l'adozione di HTML puro, sebbene funzionale, potrebbe non essere l'opzione ottimale per la scalabilità e l'estetica del client. L'utilizzo di framework più dinamici e moderni, come React, potrebbe offrire vantaggi significativi, tra cui un'interfaccia utente più reattiva e accattivante, e una gestione più efficiente del carico di lavoro, specialmente in contesti di dati complessi e interazioni utente intense.

6) DIVISIONE DEL LAVORO: Per quanto riguarda la divisione del lavoro, Davide si è concentrato maggiormente sul lato server, gestendo le route e il JavaScript lato client, e inoltre si è occupato della documentazione Swagger e del codice;
Sofia ha dimostrato ottime competenze grafiche occupandosi della creazione dei file HTML e CSS (utilizzando inoltre Bootstrap [1]) e contribuendo significativamente anche alla parte di scripting, con particolare impegno nello sviluppo della chat tramite Socket.io e nella gestione del Login.

7) Non sono richiesti dettagli extra per eseguire il codice.

MONGODB SERVER
Il MongoDB Server, configurato su 'localhost:3001', funge da server secondario, interagendo direttamente con il database MongoDB per recuperare i dati dinamici necessari al Main Server. Questo include dettagli sui Games, ClubGames,

[1] : https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js
https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css

Appearances, GameLineups, e GameEvents, garantendo così una gestione efficace e centralizzata dei dati.

1) SOLUZIONE: Il server MongoDB è strutturato con un'architettura Express modularizzata, dove ogni entità del database, come 'Games', 'ClubGames', ecc., ha un suo modulo dedicato, un controller specifico, e un insieme di routes dedicate. Questa organizzazione modulare non solo semplifica la gestione e il mantenimento del codice ma assicura anche che ogni componente sia facilmente scalabile e indipendente. Inoltre, tutte le routes sono prefissate con '/mongo', una convenzione adottata per semplificare l'identificazione delle richieste al MongoDB Server dal Main Server, rendendo il processo di smistamento delle richieste più efficiente e meno soggetto a errori.

2) PROBLEMI RISCONTRATI: La principale difficoltà incontrata è stata la complessità nello sviluppo di funzioni avanzate, dovuta principalmente alla maggiore familiarità con le query in Spring Boot rispetto a Express. Questo ha richiesto un adattamento e un apprendimento nella loro gestione.

3) RICHIESTE DA CONSEGNA: La nostra implementazione aderisce a tutte le richieste specificate.

4) LIMITAZIONI: Funzioni complesse come 'getFinalPositionsOfAllClubs', che in alcuni casi potrebbero dover agire su una consistente quantità di dati, potrebbero portare, come già detto precedentemente per il Main Server, a rallentamenti.

5) CONCLUSIONI: Il MongoDB Server si distingue per la sua struttura ben organizzata e per la chiarezza delle sue routes e funzioni. Nonostante la complessità di alcune operazioni che possono gravare sulle prestazioni, il sistema dimostra solidità e un'architettura riflessiva, offrendo una base robusta per gestire e ottimizzare le future esigenze del progetto.

6) DIVISIONE DEL LAVORO: Davide ha gestito il perfezionamento delle route e delle funzioni del controller, assicurando la logica operativa del server, e si è occupato della documentazione Swagger e del codice, mentre Sofia ha contribuito sviluppando i modelli e le routes iniziali, garantendo così una solida base di dati.

7) Non sono richiesti dettagli extra per eseguire il codice.

JAVA SERVER

Il Java Server, configuarto su 'localhost:8080', funge da server secondario, interagendo direttamente con il database Postgre per recuperare i dati statici necessari al Main Server. Questo include dettagli sulle Competitions, Clubs, Players e PlayerValuations.

[1] : https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js
https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css

1) SOLUZIONE: Il Java Server, strutturato con Spring Boot, si distingue per una chiara organizzazione in classi Java, controller, services e repository per ogni entità gestita. In aggiunta, abbiamo definito strutture DTO per ogni entità, eccetto PlayerValuations, ottimizzando così la trasmissione dati con il Main Server. Le routes, iniziate con '/postgres', assicurano una gestione chiara delle richieste, mentre i DTO, fornendo strutture dati essenziali, come nome e identificatore per le Competitions, migliorano ulteriormente l'efficienza e la sicurezza nella gestione e nel trasferimento dei dati. Inoltre, il file 'WebConfig' è stato configurato per gestire le politiche CORS, consentendo solo le richieste provenienti da "http://localhost:3000" ad accedere alle risorse sotto il percorso "/api/**", assicurando un'interazione sicura e regolata tra i diversi domini durante lo sviluppo.

2) PROBLEMI RISCONTRATI: Non si sono riscontrate difficoltà significative nello sviluppo grazie all'approccio intuitivo e alla strutturazione chiara che Spring Boot offre.

3) RICHIESTE DA CONSEGNA: La nostra implementazione aderisce a tutte le richieste specificate.

4) LIMITAZIONI: Non ci sono particolari limitazioni dovute alla struttura del server.

5) CONCLUSIONI: Esprimiamo grande soddisfazione per l'ordine e la compattezza raggiunti nel Java Server. L'adozione di strutture DTO mirate, la chiara organizzazione delle classi e la convenzione di routing efficiente hanno confluìto in un sistema coeso, che si distingue per la sua funzionalità e manutenibilità.

6) DIVISIONE DEL LAVORO: Davide ha gestito autonomamente lo sviluppo del Java Server, curando ogni aspetto, dalla creazione delle classi Java, alla definizione dei controller, dei services e delle repository per ciascuna entità trattata.

7) Non sono richiesti dettagli extra per eseguire il codice.

GESTIONE DEI DATABASE E INSERIMENTO DEI DATI

Nel progetto GoalNet360, abbiamo impostato in modo efficiente i database MongoDB e PostgreSQL attraverso i rispettivi server, MongoDB Server e Java Server, assicurando una strutturazione adeguata e performante per i dati dinamici e statici.

Per l'inserimento dei dati, abbiamo adottato un approccio interattivo e automatizzato utilizzando un Jupyter Notebook. Questo metodo ha permesso una manipolazione precisa dei dati e un inserimento coerente e affidabile nei database.

[1] : https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js
https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css

L'approccio integrato nella gestione dei database e nell'inserimento dei dati ha garantito una solida base dati per il progetto, contribuendo significativamente alla sua efficienza e successo.

[1] : https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js
https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css