Report For IUM-TWeb

1. Soluzione:

> **Risposta**
>
> **Suddivisione del Contenuto nei Database:**
>
> Abbiamo deciso di **pulire i dati** prima di inserirli nei rispettivi database. Ciò ci ha reso più semplice, leggera e veloce la loro gestione.
> Inoltre, abbiamo optato per non usare il solo criterio di staticità / dinamicità dei dati per l'inserimento in MongoDB e PostgreSQL. Infatti, abbiamo considerato delle eccezioni che riguardano il basso grado di accoppiamento di determinate tabelle, quali ad esempio `competitions` e `flags`.
>
> **Responsività**
>
> Il nostro progetto è **responsive**, pertanto prevede un'interfaccia che si adatti all'uso sia *desktop* sia *mobile*. Tale implementazione ci ha richiesto di progettare una struttura degli elementi della pagina tale, da poter essere adattabile. *(Una delle sfide iniziali più impegnative)*
>
> **Immagini & Icone**
>
> Ove possibile, abbiamo usato collegamenti a repositories di GitHub che permettessero l'uso di immagini, quali le bandiere degli Stati. [1] Le altre immagini (come il bottone della chat, lo sfondo d'erba del *footer* e il logo del sito) sono state create da noi, mediante l'uso del software **Figma**.
>
> **User Flow**
>
> **Premessa**: il nostro gruppo non è esperto dell'argomento calcistico. Pertanto, ci siamo informati su quale fosse il modo più sensato di mostrare i dati.
>
> Siamo arrivati alla conclusione che informazioni come le **partite** hanno poco senso, se estrapolate dal contesto di un **campionato**. Di conseguenza, il flusso d'uso principale parte da un certo campionato di una certa **stagione**, da cui si possono reperire le partite e le **squadre** associate.
> Dalle partite si possono reperire i **giocatori**, le squadre e gli **eventi** della partita.
> Dalle **squadre** si possono trovare informazioni sia sui giocatori **attuali** sia su quelli **passati**, più tutte le partite della squadra.
> Il sito prevede una certa libertà d'uso, non vincolata allo user flow descritto.

2. Difficoltà:

Risposta

Chat & Socket

Durante la creazione della chat, abbiamo riscontrato qualche difficoltà con la registrazione dei nomi delle *'room'* e con la funzione di *'disconnect'*. Inoltre, la chat stessa, inizialmente implementata tramite `<iframe>`, ha subito molte variazioni.
Ora il file `chat.html` è manipolato tramite *JavaScript* per poter essere usato ovunque, anche nell'`offcanvas`.

Caroselli

I caroselli sono stati costruiti da noi in maniera tale da permettere una più dinamica responsività. Il numero di elementi mostrati varia in base alla larghezza del dispositivo utilizzato.
Abbiamo riscontrato non poche difficoltà per raggiungere questo obiettivo, quasi ogni difficoltà prevedeva conoscenze troppo avanzate rispetto alle nozioni accennate nel corso.
Dapprima abbiamo avuto problemi con l'uso degli `<iframe>`, poi con l'effettivo funzionamento delle funzioni di traslazione dei singoli elementi.

Solo dopo molto tempo, siamo venuti a conoscenza dell'esistenza di librerie apposite [2], che ad una prima ricerca non avevamo scovato. A tal punto, modificare il codice (strutturato seguendo un approccio di sviluppo *"a cascata"* e non *"iterativo"*) avrebbe richiesto un sforzo maggiore del completamento del progetto stesso.

Tuttora, il carosello presenta un difetto: l'uso delle frecce laterali prima e dopo aver ridimensionato una pagina genera un comportamento non voluto, costringendo a ricaricare la pagina per una corretta visualizzazione.

Responsività

Il progetto è stato sviluppato con l'idea di fornire una GUI responsiva che copra diverse dimensioni. Ciò ci ha costretti a studiare un'interfaccia che ci permettesse facilmente di nascondere, mostrare o ridimensionare determinati elementi.
In generale, abbiamo cercato di curare ogni dettaglio sia nella versione mobile che nella versione desktop.

Inserimento dati JPA

Abbiamo riscontrato alcune difficoltà con l'uso dell'API usate per il server SpringBoot nella gestione dei **vincoli** di **persistenza** tra le *tables* del database PostgreSQL. A causa di tali vincoli, l'ordine di inserimento dei dati è prestabilito a priori.

3. Requisiti:

Risposta

A noi sembra che il progetto soddisfi ogni requisito richiesto dall'assignment, fatta eccezione per l'**opzionale** collegamento tramite *Flask*.

4. Limitazioni:

Risposta

- In generale, per ogni single page si sarebbero potute mostrare più informazioni e più strutturate. Ad esempio, la `single_page/club` poteva essere organizzata per **stagioni** e si potevano mostrare più informazioni, come statistiche derivate dalle partite o i vari campionati vinti dalla squadra.
- Per le funzioni di ricerca si sarebbero potute fare ricerche più avanzate.
- L'uso della chat è leggermente limitato dal fatto che ricaricare la pagina fa perdere lo storico dei messaggi.
- Vedasi il paragrafo "*caroselli*" della sezione ***Difficoltà***.

5. Conclusioni:

Risposta

1. Aver effettuato l'analisi dei dati e una preliminare ricerca sul loro possibile utilizzo sono stati passi fondamentali per la riuscita del progetto.
2. L'utilizzo di un approccio di sviluppo **iterativo** (e non *"a cascata"*).
3. Evitare l'uso di `<iframe>` per il riuso del codice, appoggiarsi a framework aggiornati (es. librerie quali *slider.js*, uso di *vue.js*).
4. Impiegare più tempo nella ricerca di tool adeguati al conseguimento degli obiettivi prefissati.
5. Costruzione dei database relazionali affidata agli stessi strumenti che gestiscono i server che vi si interfacciano (es. uso di **Jakarta** per la creazione degli schemas PostgreSQL).

6. Siamo particolarmente soddisfatti e fieri del risultato ottenuto, pur tenendo conto dei suoi punti deboli.

6. Suddivisione del Lavoro:

Risposta

Abbiamo cercato di dividere equamente il lavoro tra di noi. Generalmente, ci siamo suddivisi i compiti come segue:

- *Lusso Ivan* → Si è focalizzato sull'implementazione di **chat** e **socket.io**, lavorando inoltre al set up del database **MongoDB**, alle routes del **Second Express Server** e alla **Data Analisi**.
- *Schiavone Marco* → Ha pensato all'**interfaccia** del client, focalizzandosi sull'implementazione di **GUI**, elementi HTML/CSS e relativi problemi (specialmente per i caroselli), alla **Data Analisi** e stesura delle routes per **MongoDB**.
- *Stefanetti Matteo* → Si è occupato della parte relativa al database **PostgreSQL** , il suo server e le sue *tables*, oltre che alle route del **SpringBoot Server**. Infine, ha anche lavorato all'implementazione della **GUI**.

Detto ciò, ogni membro del gruppo ha toccato, almeno in piccola parte, ogni aspetto del progetto.
Laddove una parte di codice sia stata implementata da un solo componente *(es. chat system, custom carousels, JPA set up)*, la comunicazione dei progressi tra i membri del gruppo non è mai mancata.

7. Informazioni Extra

Risposta

Prima dell'avvio
La versione funzionante del progetto si trova all'interno della cartella *'/solution'*. Le cartelle dei server all'interno devono essere estratte in quella posizione.

Popolamento dei databases
Innanzitutto, i dati *puliti* sono disponibili nella cartella mediafire (altrimenti creati tramite `jsonifier.py`), vanno scaricati e spostati come segue:

- Per **MongoDB** in `/solution/SecondExpressServerTWeb23/json` → vanno inseriti i file generati: `cleaned_appearances.json` , `cleaned_competitions.json` ,

`cleaned_flags.json`, `cleaned_player_valuations.json`, `cleaned_game_lineups.json`

- Per **PostgreSQL** in `/solution/JavaSpringBootServerTWeb23/json` → vanno inseriti i file generati: `cleaned_clubs.json`, `cleaned_games.json`, `cleaned_club_games.json`, `cleaned_players.json`, `cleaned_game_events.json`

Fatto ciò, si può procedere al popolamento.

- *MongoDB* → Una volta posizionati i file `.json`, basterà aprire una scheda nel browser e digitare il link `http://localhost:3002/insert_mongo`.
- *PostgreSQL* → ci sono 2 modi:
  1. Usare i file di dump nella cartella '*/databaseschemas*' (per cui non servono i file `.json`, ma potrebbe non funzionare ovunque)
  2. avviare il server *SpringBoot* (vedi sezione *Scripts*), aprire un terminale posizionato nella directory `/solution/JavaSpringBootServerTWeb23/json`, eseguire **in ordine** le seguenti CURL dal terminale:

```
curl -X POST -H "Content-Type: application/json" -d @cleaned_clubs.json
http://localhost:8081/clubs/add_clubs
```

```
curl -X POST -H "Content-Type: application/json" -d @cleaned_games.json
http://localhost:8081/games/add_games
```

```
curl -X POST -H "Content-Type: application/json" -d @cleaned_players.json
http://localhost:8081/players/add_players
```

```
curl -X POST -H "Content-Type: application/json" -d @cleaned_club_games.json
http://localhost:8081/club_games/add_club_games
```

```
curl -X POST -H "Content-Type: application/json" -d
@cleaned_game_events.json http://localhost:8081/game_events/add_game_events
```

Scripts

Abbiamo creato 3 file di tipo `.bat` e 3 di tipo `.sh` per avviare i server *senza* dover usare i software JetBrains:

- `C:\ownPathToTheProject>WinMainServer.bat`
- `~/ownPathToTheProject$ ./LinuxMainServer.sh`

Responsività

Il codice è pensato per essere responsivo, perciò il sito funziona sia in finestra che in

fullscreen. Tuttavia, non è pensato per funzionare durante il ridimensionamento della finestra del browser.

8. Bibliografia

Risposta

Abbiamo usato diverse fonti per aiutarci a comprendere come funzionino le librerie e le API usate nel progetto.
Più che una bibliografia, ci accingiamo ad elencare una **sitografia** del materiale di supporto che abbiamo usato.
Abbiamo usato la **documentazione ufficiale** di:

- '*socket.io*' per l'implementazione della **chat** (docs e get-started; vedi `public/javascripts/chat.js` e `socket.io/socket.io.js` su *MainExpressServer*)
- '*Bootstrap v5.3*' per l'interfaccia del client (getbootstrap.com; usata per *popover*, *tooltip*, *offcanvas*, *accordions* etc.)
- '*MDN Mozilla JavaScript*' per le funzioni usate dal client (MDN Mozilla)
- JPA, Express, e Mongoose per implementare le funzionalità dei server secondari.

Inoltre, abbiamo fatto uso di strumenti AI (come ad esempio *ChatGPT 4.0*, *Google Bard*, e *Gemini AI*), che ci sono stati particolarmente utili nello studio della sintassi di **Python** e nell'uso delle sue librerie per la **Data Analisi**.

Infine menzioniamo i siti w3Schools e StackOverflow, che ci hanno fornito supporto significativamente durante le fasi iniziali, pur non essendo sempre esaustivi ai fini del nostro progetto.

---

1. **Bandiere**: reperibili qui ↩
2. e.s. `slider.js` ↩