Movie Platform

Applicazioni e Servizi Web

Pagnini Lorenzo - 0000942265 {lorenzo.pagnini2@studio.unibo.it}

7 Aprile 2021

Indice

Introduzione

Il progetto consiste nella realizzazione di una piattaforma legata al mondo cinematografico, televisivo.

L'applicazione è stata pensata principalmente per proporre, agli utenti, un catalogo di film divisi per categorie: in base alla popolarità, ai più votati e alle nuove uscite. Successivamente, in un secondo momento, si è pensato di arricchire la libreria con contenuti legati alla TV e agli attori. Ogni contenuto è corredato di un'immagine e alcune informazioni che lo descrivono.

I contenuti sono resi disponibili da **TMDB** (The Movie DataBase). Una comunità, nata nel 2008, che ogni anno arricchisce il proprio database rendendolo fruibile ai propri iscritti. Per maggiori informazioni è possibile consultare il seguente *link*.

Gli utenti, dopo essersi registrati, possono consultare i contenuti in base alle tipologie precedentemente descritte o ricercare direttamente il titolo del film / programma TV o il nome dell'attore. Altre funzionalità offerte all'utente sono quelle di votare, gestire un proprio elenco preferito e caricare nuovi contenuti.

Capitolo 1

Requisiti

In questo capitolo vengono elencate le funzionalità messe a disposizione all'utente elencando, di fatto, le caratteristiche dell'applicativo. In questo elaborato, è presente un'unica tipologia di utilizzatore ovvero l'utente: fruitore del servizio.

1.1 Requisiti utente

L'applicazione offre all'utente le seguenti funzionalità:

- **Registrazione**. In questa sezione l'utente crea un nuovo account nella piattaforma. I dati richiesti sono: un nome, un username (univoco nel sistema), l'email (univoca nel sistema) e una password senza vincoli prestabiliti (è comunque consigliato scegliere una password alfanumerica di almeno 8 caratteri). Per aumentare la sicurezza dei dati nei confronti degli utilizzatori, la password viene prima cifrata e poi salvata nel database. A questo punto per potersi autenticare è necessario prima verificare l'account, cliccando sul link ricevuto nell'email all'indirizzo specificato.

  Ogni qualvolta che viene creato un nuovo account, viene generato un token (relativo alla verifica dell'account) con validità di 12 ore. Scaduto il tempo sarà necessario farsi re-inviare dal sistema una nuova email con un nuovo token. Questo avviene quando l'utente clicca nel link dell'email con il token scaduto. Solo in questo caso viene visualizzato un pulsante che permette l'invio di una nuova email. Dopo aver verificato l'account è possibile autenticarsi correttamente.

- **Autenticazione**. L'autenticazione avviene fornendo l'email o l'username e la password forniti in fase di registrazione. Se le credenziali non sono corrette, verrà mostrato un *alert* con la motivazione dell'errore. L'autenticazione verrà mantenuta anche alla chiusura e riapertura del browser fino allo scadere del token generato per questa fase.

- **Dashboard**. Dopo l'autenticazione, l'utente avrà modo di consultare i vari contenuti all'interno di un menù, selezionando la categoria di suo piacimento.

- **Ricerca di uno specifico contenuto**. Oltre a visualizzare i contenuti mostrati, l'utente può cercare uno specifico film, programma TV e/o attore. Questo è possibile farlo utilizzando la barra di ricerca. Scrivendo una qualsiasi parola o frase, il sistema troverà tutti i contenuti che "matchano" in base alla categoria in cui si è impostati.

- **Votare un contenuto**. Ogni contenuto ha una griglia di valutazione sotto forma di stelle. Tale valutazione parte da un minimo di una stella a un massimo di 5 stelle. L'utente non ha limiti di voto sia sullo stesso contenuto che sul numero totale di contenuti del sistema indipendentemente dalla categoria di cui appartiene. E' sempre possibile modificare la valutazione di un contenuto già votato.

- **Visualizzazione dei contenuti votati**. Nell'apposita sezione del profilo, è possibile consultare i contenuti votati selezionando la categoria desiderata. Anche nella visualizzazione dei contenuti votati è comunque possibile cambiare la valutazione.

- **Inserimento di nuovi contenuti**. L'utente può inserire dei propri contenuti che, per esempio, non sono presenti nel sistema. In questo caso sarà necessario inserire le informazioni richieste, a seconda della categoria selezionata. Inoltre è anche possibile inserire una immagine del contenuto appena caricato.

- **Modifica delle informazioni personali**. L'utente può anche modificare le proprie informazioni personali inserite in fase di registrazione. Le modifiche possono essere apportate al nome, all'username e all'email. Ad ogni nuova modifica, il sistema controllerà che non vi siano già degli altri utenti con un username o email uguali a quella inserite dall'utente.

- **Visualizzazione delle notifiche ricevute**. Grazie alle notifiche in tempo reale, l'utente verrà notificato ogni volta che verrà inserito un nuovo contenuto da parte di altri utenti. La notifica apparirà nell'apposita icona e indica il numero di notifiche ricevute ma ancora non lette. Cliccando sulla specifica icona si potrà consultare le informazioni per ognuna di essa: il nome del contenuto, la categoria a cui appartiene e l'username dell'utente che ha caricato il contenuto. E' presente anche un pulsante per eliminare tutte le notifiche ricevute fino a quel momento.

L'applicazione realizzata deve essere fruibile sia da dispositivi mobile che da desktop.

Si rimanda al prossimo capitolo per la visualizzazione dell'interfaccia utente creata per le singole funzionalità.

Capitolo 2

Design

Nella progettazione del design del sistema e quindi dell'interfaccia utente, si sono seguite le seguenti fasi:

- **Creazione di modelli base**: attraverso la realizzazione di *mockup* utilizzando il tool Balsamiq (come descritto nel capitolo 4). Questa fase si è rivelata utile soprattutto per individuare i principali componenti grafici e identificare le linee guida di design dell'applicazione;
- **Confronto con persone esterne** al fine di migliorare l'interfaccia utente;
- **Scelta del framework** con cui sviluppare l'interfaccia utente. Si è deciso di utilizzare Material UI (per maggiori informazioni consultare il capitolo 4) poichè si è appreso essere una delle librerie più popolari per sviluppare le interfacce utente utilizzando il framework React.

Di seguito vengono riportate le varie interfacce dell'applicazione, in versione mobile. Per rendere gli screenshot maggiormente visibili e nitidi, nella relazione, sono stati ritagliati il più possibile. Questo significa che le dimensioni reali dell'interfaccia potrebbero variare leggermente. Per questo motivo si consiglia di prendere come riferimento per le dimensioni, l'interfaccia sviluppata con balsamiq (a sinistra delle due foto).

2.1 Registrazione

In fase di registrazione, vengono richieste all'utente le proprie informazioni personali. Per una migliore esperienza utente si è deciso di aggiungere, nei campi delle password, i pulsanti per nascondere e rendere visibile la password che si sta inserendo.

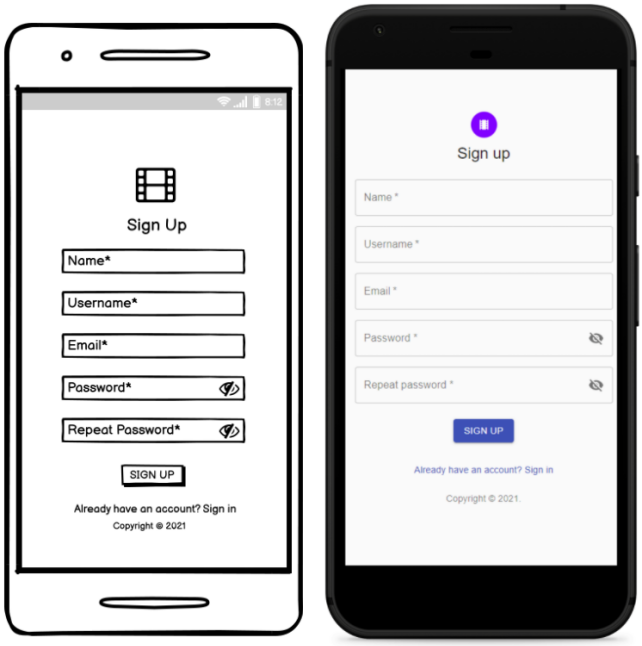

Figura 2.1: Schermata di registrazione

2.2 Autenticazione

In fase di autenticazione l'utente deve inserire le proprie credenziali. Se sono corrette, l'autenticazione avviene con successo altrimenti viene mostrato un *alert* di errore.

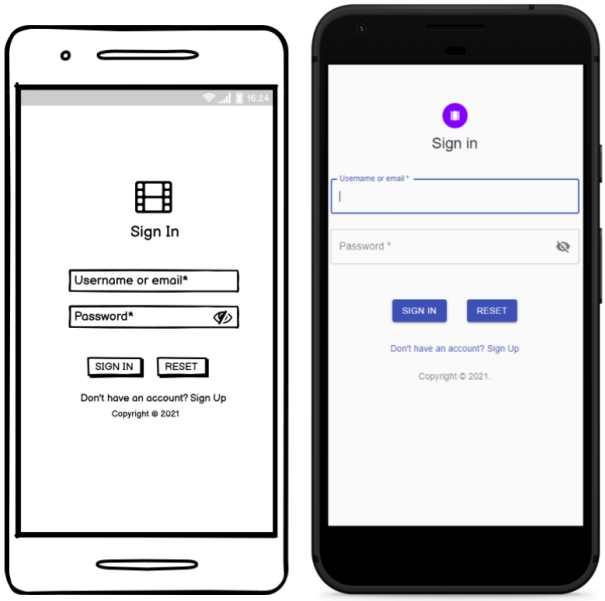

Figura 2.2: Schermata di autenticazione

2.3 Dashboard

La dashboard è composta da un *drawer* (o menù) che elenca le categorie disponibili. Al centro sono presenti i contenuti. Ognuno di essi è una *card* contenente l'immagine, alcune informazioni quali il titolo, il rank e la valutazione dell'utente (se presente).

Figura 2.3: Schermata della dashboard

2.4 Funzionalità legate al profilo utente

In questo paragrafo vengono mostrate le funzionalità offerte accedendo al profilo utente. In ogni schermata è sempre presente una *toolbar* che permette di ritornare alla dashboard (icona home), consultare le notifiche e uscire dall'account.

2.4.1 Visualizzazione dei contenuti votati

Per visualizzare i contenuti votati è possibile spuntare gli *switch* in base alla categoria scelta. Nel caso in cui ci sono contenuti da visualizzare, questi vengono mostrati e appaiono come nella dashboard, altrimenti appare un *alert* informando che non sono presenti elementi da visualizzare.

Figura 2.4: Schermata per visualizzare i contenuti votati

2.4.2 Caricamento di un nuovo contenuto

In questo caso è presente un *menù* a tendina che permette di selezionare la categoria desiderata per caricare un contenuto. Dopodichè, in base alla categoria, vengono richieste le informazioni che l'utente deve inserire. E' anche possibile caricare un immagine cliccando sull'apposito pulsante *upload*. Non è possibile invece caricare solo l'immagine e lasciare i campi testuali vuoti. In caso di corretto inserimento apparirà un *alert* di successo altrimenti uno di fallimento che riporta l'errore.

Figura 2.5: Schermata per il caricamento di un nuovo contenuto

2.4.3 Modifica delle informazioni personali

Nell'ultima sezione, l'utente può vedere le proprie informazioni personali ed eventualmente modificarle. Questo è possibile spuntando l'apposita matita, inserire i nuovi dati e salvarli. Questa sezione permette anche di eliminare il proprio account. Premendo il pulsante, verrà mostrata una *dialog* per confermare l'operazione.

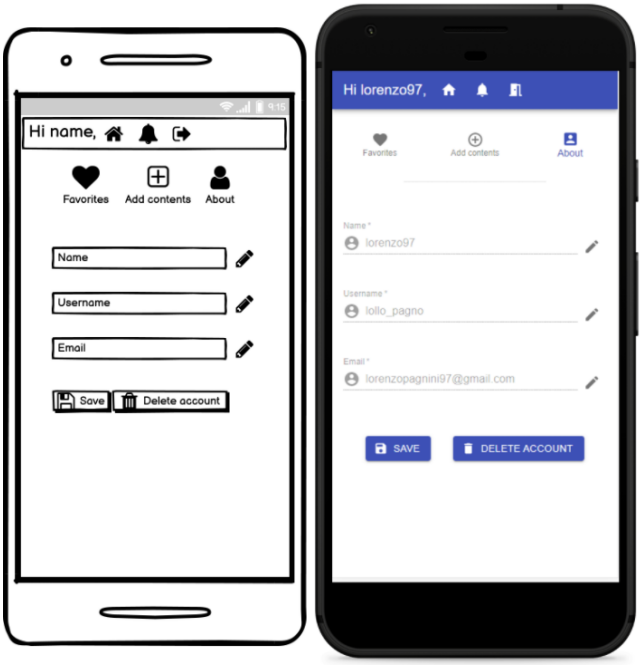

Figura 2.6: Schermata per la modifica delle informazioni personali

2.5 Notifiche

Dalla dashboard o dal profilo utente, cliccando l'apposita icona è possibile visualizzare le notifiche ricevute. Ogni notifica indica il titolo o nome del contenuto aggiunto, la categoria a cui appartiene e l'username dell'utente. Per cancellare la liste delle notifiche è stato messo a disposizione l'apposito pulsante.

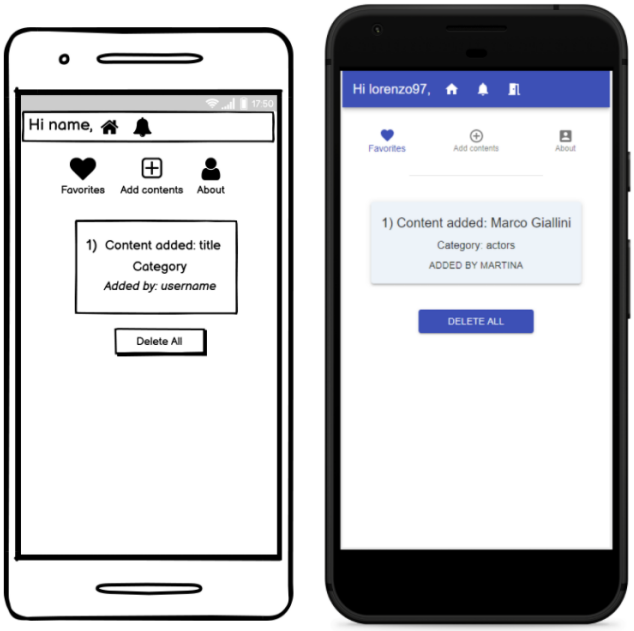

Figura 2.7: Schermata di visualizzazione delle notifiche

Capitolo 3

Tecnologie

In questo capitolo verranno illustrate le principali tecnologie utilizzate nel progetto, chiarendone le motivazioni del loro utilizzo.

3.1 Stack MERN

Come scelta progettuale, si è deciso di adottare lo stack MERN, adottando il framework React per lo sviluppo della parte frontend, Express.js e Node.js per lo sviluppo della parte backend e il DBMS MongoDb per la gestione della persistenza.

3.2 Server side

3.2.1 Node.js

Node.js è un framework JavaScript open source, multipiattaforma, orientato agli eventi per l'esecuzione di codice JavaScript. Questa tecnologia è stata usata perchè risulta essere una valida soluzione sia per lo sviluppo e la creazione di servizi web, sia per l'integrazione con altri framework come database documentali e librerie Javascript utilizzate per la parte di frontend. Inoltre utilizzando Node.js si ha una migliore efficienza e produttività agevolando i tempi di sviluppo e ottenendo un software veloce e prestante.

3.2.2 Express.js

Express.js è lo standard de facto fra i framework per applicazioni server costruite su Node.js. All'interno del progetto è stato utilizzato principalmente per la parte di routing.

3.2.3 MongoDB

MongoDb è un sistema di gestione di database (DBMS) open source che, rispetto ai database SQL, è orientato ai documenti, gestendo i dati in particolari strutture chiamate collezioni. Per questo motivo viene classificato come un database di tipo NoSQL.
Poiché si presta bene allo sviluppo di applicazioni web, rivelandosi molto semplice ed intuitivo, l'utilizzo di un database di questo tipo mi ha permesso di modellare e gestire i dati in maniera più agevole e più efficiente rispetto ai database relazionali.

3.2.4 Node mailjet

Node mailjet è un wrapper per Node.js messo a disposizione da Mailjet (maggiori informazioni a questo *link*) che permette l'interfacciamento con il client mail per l'invio di posta elettronica. Nel progetto è stato utilizzato per verificare l'account creato da un utente durante la registrazione all'applicativo.

3.2.5 Deep email validator

Deep email validator è un modulo per JavaScript che permette di verificare se uno specifico indirizzo di posta elettronica ha un formato valido ovvero che contiene una "@" e un "." a destra di esso. Inoltre permette di individuare errori comuni di battitura e verificare l'esistenza di tale casella postale sul server SMTP. In caso di errore fornisce una breve descrizione del problema.

3.2.6 Bcrypt

La Bcrypt è una funzione di hashing delle password che, tra le tante funzionalità, permette anche di impostare un valore di sale aumentando maggiormente le protezione da possibili attacchi. All'interno del progetto è stato utilizzato un modulo per Node.js che permette di implementare la funzione di cifratura e decifratura. Il modulo è stato utilizzato per cifrare le password degli utenti, prima di memorizzarle all'interno del database.

3.2.7 Socket.io

Socket.io è una libreria JavaScript che implementa dei canali bidirezionali tra server e client permettendo l'invio di notifiche in tempo reale. All'interno del progetto è stata utilizzata per inviare delle notifiche a tutti gli utenti iscritti al sistema, ogni qualvolta veniva creato un nuovo contenuto da parte di uno specifico utente. La notifica mostra il titolo o nome del contenuto (a seconda se si tratti di un film, programma TV o attore), a quale categoria appartiene e l'username dell'utente che ha creato il contenuto.

3.2.8 Json Web Token

Il Json Web Token (JWT) è uno standard che definisce uno schema in formato JSON per lo scambio di informazioni. Permette di generare un token firmato attraverso una chiave segreta, composto da 3 parti elencate qui di seguito:

- Header: indica la tipologia del token e l'algoritmo di cifratura utilizzato;
- Payload: contiene informazioni di interscambio come i parametri passati secondo una precisa struttura che è stata definita;
- Signature: viene creata a partire dall'header codificato, il payload, la chiave segreta e l'algoritmo di cifratura specificato da cui poi generare il token.

La tecnologia JWT è stata utilizzata per controllare l'accesso degli utenti nel sistema. In particolare ogni utente esegue i seguenti passi, così descritti:

1. **Registrazione nel sistema**: gli utenti tramite l'interfaccia dedicata possono registrarsi inserendo i propri dati (nome, username, email e password).
2. **Autenticazione**: a seguito della registrazione e della verifica dell'account tramite email, l'utente potrà autenticarsi al sistema. In caso di credenziali corrette, il server restituisce il JWT (creato come descritto in precedenza) per lo specifico utente.
3. **Esecuzione di una richiesta**: Ad ogni richiesta eseguita dall'utente, in un primo momento viene controllato il token. Se scaduto si richiede all'utente di autenticarsi nuovamente, altrimenti viene visualizzata la pagina richiesta. Questo perchè il token generato ha una durata massima di 1 ora. Questo significa che se l'utente lascia la postazione di lavoro per più di un'ora, la sessione scade e previene l'uso del proprio account da parte di malintenzionati. Inoltre ad ogni richiesta eseguita correttamente dall'utente, il token viene ricreato, impostando ad 1 ora il tempo massimo di scadenza.

Il JWT risulta essere completamente invisibile all'utente che utilizza l'applicazione. La memorizzazione e la gestione del token vengono effettuate in maniera integrata attraverso il frontend, che memorizza il JWT sotto forma di cookie.

3.3 Client side

3.3.1 React

React è una libreria JavaScript utilizzata per la creazione di interfacce utente. Il motivo per cui ho deciso di utilizzare questa libreria è dato dalla curiosità e voglia di approfondire questo framework rispetto agli altri proposti e descritti durante il corso. Alcune peculiarità di React sono:

- Fornisce una struttura basata sui componenti. E' possibile creare un grande componente attraverso un wrapper composto da componenti più piccoli;
- Rendering veloce con DOM (Document Object Model) virtuale;
- Possibilità di integrare in maniera efficiente librerie JavaScript come Redux (si veda il prossimo paragrafo) e React Router per la gestione del routing.

3.3.2 Redux

Redux è una libreria JavaScript open source per la gestione dello stato dell'applicazione, maggiormente utilizzata con librerie React. E' stata progettata per gestire un contenitore che rappresenti lo stato dell'applicazione. Si basa sul concetto di avere un unico stato, rappresentato da un oggetto JSON e conservato in uno store. Questo stato può mutare solo in seguito ad azioni, ma non direttamente. La modifica avviene tramite l'invocazione di una funzione pura denominata reducer. Alcuni principi fondamentali di questa libreria sono:

- Lo stato è in sola lettura e può mutare solo con un intento ben definito, a partire da una azione attraverso un reducer;
- I cambiamenti allo stato avvengono solo attraverso funzioni pure, cioè una funzione che dato un certo input restituisce sempre lo stesso output.

Nell'elaborato Redux è stato molto utile per la gestione del JWT durante la fase di registrazione e di autenticazione. Di grande aiuto è stato nel gestire i dati utente tutte le volte che era necessario eseguire delle richieste che prevedevano tali informazioni o qualora l'utente voleva modificarli.

3.3.3 Axios

Axios è una libreria che permette di gestire la comunicazione client - server. Si è rivelata essere un'ottima libreria poiché semplifica l'invio di richieste HTTP asincrone e permette l'esecuzione di operazioni CRUD.

3.3.4 Material UI

Material UI è una libreria molto utilizzata con React, che permette di realizzare l'interfaccia utente. Il suo utilizzo mi ha permesso di produrre agevolmente una *user interface* che si adatta molto facilmente a schermi di varie dimensioni di smartphone e tablet. Per maggiori informazioni è possibile consultare il seguente *link*.

3.4 Tecnologie di supporto

3.4.1 GitHub

GitHub è un servizio di hosting per progetti software che permette di fare *version control system* (VCS) del proprio progetto. La piattaforma è stata scelta perchè ampiamente conosciuta. Per l'intero sviluppo del progetto si è seguito il seguente workflow: sono stati creati due branch, `master` e `devel`. Ogni nuova funzionalità è stata sviluppata su `devel`. Al completamento di ognuna, veniva creata una pull request verso il branch `master` e "mergiata" al suo interno.

3.4.2 Postman

Postman è una piattaforma per lo sviluppo di API. Le funzionalità offerte semplificano ogni passaggio della creazione di un'API in modo tale da poter creare API migliori e più velocemente. L'utilizzo di questa piattaforma mi ha permesso di testare l'API create in maniera rapida sia per quanto riguarda lo stato che il payload di risposta delle chiamate.

3.4.3 Heroku

Heroku è una *platform as a service* (PaaS) basata su un sistema di contenitori gestiti, con servizi dati integrati e un potente ecosistema, per la distribuzione e l'esecuzione di app moderne. Heroku è stata selezionata in ambito di progetto come tecnologia per il deployment del sistema. Per maggiori informazioni sul deploy si veda il paragrafo 6.2 del capitolo 6.

3.4.4 Balsamiq

Balsamiq è un tool grafico che permette la creazione di schermate (wireframe) e interfacce utente di siti web. Tale software è stato utilizzato per realizzare le interfacce grafiche in fase di design.

3.4.5 Trello

Trello è un software gestionale che permette di tenere traccia delle attività correlate alla fase di sviluppo. Per una migliore organizzazione si sono realizzate delle etichette personalizzate da associare alle diverse features da implementare. Inoltre si sono create delle colonne in cui raggruppare le features a seconda del suo stato di sviluppo.Alcune di queste sono:

- **To Do**: contiene le card associate alle features da sviluppare;
- **Done**: contiene le card associate alle features sviluppate;
- **Bug**: contiene le card associate alle features che presentano bug da 'fixare'.

La bacheca è possibile consultarla al seguente link: *trello*.

Capitolo 4

Codice

In questo capitolo si andrà a descrivere il codice sviluppato, trattandone gli aspetti rilevanti fornendo alcuni esempi.

Al livello root del repository sono contenute le due cartelle server e client. Nei successivi paragrafi verranno descritte le rispettive architetture. Si invita comunque a consultare il codice per maggiori informazioni sul suo funzionamento, in quanto è correlato da una ricca documentazione.

4.1 Struttura del server

Il codice del server è stato realizzato interamente in linguaggio JavaScript, e si appoggia sul gestore di pacchetti npm. L'entry point del server è il file `server.js`. Eseguendo tale file in locale, la libreria Express avvia un server sulla porta 3000.

Il codice è stato organizzato nelle seguenti sotto directory:

- **controllers**: la directory contiene i controller sia quelli per la gestione utente e sia quelli per la gestione dei contenuti di Tmdb. Contiene quindi, tutte le funzioni invocate da ogni route dell'API. Gestisce inoltre tutte le operazioni che prevedono l'interazione con il database mongoDb;
- **db**: contiene il file per la connessione al database locale in caso di connessione con localhost, remoto in caso di connessione con l'host di heroku;
- **model**: contiene gli schemi utilizzati dai documenti di mongoDb. In particolare permettono di memorizzare le informazioni dell'utente e quelle del token, salvare nuovi contenuti e gestirne i voti;
- **route**: contiene il file responsabile della generazione delle varie route, collegate alle funzioni presenti nella directory controllers;

- **utils**: contiene funzionalità condivise e moduli di utility che ho voluto raggruppare in un unica directory. Sono presenti anche le variabili d'ambiente e i vari codici di stato di una richiesta.

4.1.1 Esempi

In questo paragrafo vengono mostrati degli esempi sull'utilizzo della libreria mongoose per interagire con MongoDb.

Nella figura 4.1 è possibile vedere la struttura dello schema *user* usato per memorizzare le informazioni inserite dall'utente durante la registrazione del proprio account.

```
const userSchema = mongoose.Schema(
    {
        name: {type: String, required: true},
        email: {type: String, required: true, unique: true},
        username: {type: String, required: true, unique: true},
        password: {type: String, required: true},
        isVerified: {type: Boolean, default: false},
        createAt: {type: Date, required: true, default: Date.now},
        updateAt: {type: Date, required: true, default: Date.now}
    }, {
        versionKey: false
    });

module.exports = mongoose.model( name: 'User', userSchema,  collection: 'user');
```

Figura 4.1: Struttura dello schema utente di MongoDb

Su ogni schema dell'elaborato, è presente un campo *id* generato automaticamente da mongoDb, che permette di identificare ogni singola entità di una collezione.

Nell'immagine successiva (figura 4.2) ho voluto mostrare una query di esempio effettuata sulla collezione *rating* del database. La query viene utilizzata per ricercare i contenuti votati dall'utente in base alle categorie selezionate (film, programmi TV e/o attori). Dopo aver "matchato" la ricerca in base all'id dell'utente, successivamente vengono trovati tutti i contenuti in base alla categorie indicate. Per fare quest'ultimo passaggio, si usa la keyword *$filter* che applicata al precedente output si ricava il risultato finale che verrà mostrato lato frontend.

```
let result = await RatingSchema.aggregate( pipeline: [

    {$match: {_userId: ObjectId(userId)}},
    {
        $project: {
            content: {
                $filter: {
                    input: '$content',
                    as: 'item',
                    cond: {
                        $or: [
                            eqMovies,
                            eqTvs,
                            eqActors
                        ]
                    }
                }
            }
        }
    }
])
```

Figura 4.2: Query che ricerca i contenuti preferiti di uno specifico utente

4.2 Struttura del client

Il codice del client è stato realizzato utilizzando il framework React e la libreria Redux per la gestione dello stato. L'entry point del client è il file `index.js`. Eseguendo tale file in locale, il client viene messo in ascolto sulla porta 8000.

Il codice è stato organizzato nelle seguenti sotto directory a partire dalla cartella *src*:

- **components**: contiene tutti i principali componenti che caratterizzano l'applicativo. Tutto ciò che riguarda la parte di view, i componenti material, gli elementi di navigazione, fogli di stile `.css`;
- **redux**: contiene tutta la logica di gestione dello stato che redux prevede: lo store e i vari reducers;
- **requests**: contiene tutte le richieste che il client effettua verso il server;
- **route**: contiene le route del client implementate mediante la libreria React Router.

4.2.1 Esempi

In questo paragrafo ho approfondito maggiormente l'utilizzo della libreria Redux. Come si può notare dalla figura 4.3, lo stato di Redux è combinato da un insieme di sotto stati chiamati reducers.

```
const combineReduces = combineReducers( reducers: {
    user: userReducer,
    token: tokenReducer,
    signIn: signInReducer,
    socket: socketReducer
})
```

Figura 4.3: Stato di Redux ottenuto combinando i vari reducers (sotto stati)

I reducers, che compongono lo stato globale di Redux, sono:

- **userReducer**: parte di stato che gestisce le informazioni dell'utente;
- **tokenReducer**: parte di stato che gestisce il token generato dal JWT;
- **signInReducer**: parte di stato che gestisce gli alert di autenticazione;
- **socketReducer**: parte di stato che permette di gestire le notifiche in tempo reale.

Per meglio comprendere la struttura di un reducer, osserviamo la figura 4.4. In questo caso è stato preso in considerazione lo *userReducer*. In primo luogo è necessario realizzare la struttura del reducer definendo tutti i campi di cui è composto (in questo caso *email*, *username*, *name* e *_id*) e assegnarli dei valori iniziali. Successivamente vengono definite tutte le azioni che quel reducer può avere. Per brevità nella figura sono presenti solo due azioni (per approfondire tutte le azioni del reducer si invita a consultare il progetto del client). Ogni azione restituisce sempre un nuovo stato, questo è dato dall'immutabilità che Redux impone per lavorare con essa. A questo stato vengono memorizzate nuove informazioni, a seconda del tipo di azione eseguita, passandogli in input il payload.

Nell'immagine si può notare l'azione *signUpSuccess* che viene eseguita quando un utente crea correttamente un nuovo account e l'azione *signInSuccess* che viene eseguita quando un utente esegue correttamente l'autenticazione al sistema. Da entrambi le azioni si può notare il nuovo stato creato, in cui se l'azione corrente non modifica un particolare valore di un campo dello stato, tale valore sarà uguale al valore dello stato precedente all'esecuzione dell'azione.

```
export const user = createSlice( options: {
    name: 'user',
    initialState: {
        email: undefined,
        username: undefined,
        name: undefined,
        _id: undefined
    },
    reducers: {
        /**
         * Set user data
         * @param state current state
         * @param action payload
         */
        signUpSuccess: (state : Draft<State> , action : PayloadAction<any> ) => {
            const {email, name, username} = action.payload
            return {
                ...state,
                name: name,
                email: email,
                username: username
            }
        },
        /**
         * Set user data
         * @param state current state
         * @param action payload
         */
        signInSuccess: (state : Draft<State> , action : PayloadAction<any> ) => {
            const {email, name, username, _id} = action.payload
            return {
                ...state,
                _id: _id,
                name: name,
                username: username,
                email: email
            }
        },
        /**
```

Figura 4.4: Struttura del reducer userReducer

Capitolo 5

Test

In questo capitolo verranno illustrate le metodologie di testing che si è deciso di adottare.

5.1 Test con utenti

Durante il progetto si è cercato di eseguire i test al sistema man mano che venivano completate delle nuove funzionalità. Queste sono state effettuate grazie al supporto amici e familiari. I test sono stati fatti in rete locale con diversi tipi di device quali smartphone, tablet e computer cercando di simulare l'utilizzo reale dell'applicazione.

Agli utenti sono stati richiesti di eseguire alcuni tra questi compiti:

- Registrazione di un account, autenticazione e logout;
- Sfogliare i contenuti delle diverse categorie presenti, votando alcuni di essi;
- Sfogliare i contenuti votati e caricare un nuovo contenuto.

Di seguito vengono riportate le considerazioni emerse durante i test. Si è deciso di elencare due categorie di utenti che accomunano quelli testati realmente.

- **Utente A, esperto nella navigazione web**. Questo utente non ha avuto grandi difficoltà ed è riuscito a svolgere tutti i compiti elencati precedentemente. Alcune criticità emerse sono, ad esempio, la mancanza di una *dialog* sia per l'eliminazione di un utente dal sistema e sia per il logout. Questo poteva portare ad un eliminazione del profilo utente se veniva cliccato accidentalmente il pulsante specifico. Tale osservazione è stata accolta provvedendo ad inserire per entrambe le azioni una *dialog* di conferma.

- **Utente B, poca esperienza nella navigazione web**. Questo utente ha avuto alcune difficoltà durante la registrazione e consultazione dei contenuti. In particolare ha trovato la *user experience* un pò complicata e confusionaria portandolo, alcune volte, a non capirne il suo funzionamento. Questo è accaduto quando, inserendo nella barra di ricerca un titolo di un programma TV, non appariva nei risultati poichè non si era nell'apposita sezione ma si era nella sezione dei films. Un'altra osservazione è che l'icona utente, sopratutto se si è in versione mobile, per accedere alle varie funzionalità era nascosta e piccola. In generale, una volta testato l'intero sistema e capito il suo funzionamento, guidato anche dallo sviluppatore, ha ritenuto abbastanza soddisfacente l'utilizzo di questa piattaforma.

Capitolo 6

Deployment

6.1 Installazione e messa in funzione

Il codice sorgente dell'applicazione è disponibile al seguente repository di GitHub: `https://github.com/pagno4/MoviePlatform`.
Clonando il progetto è possibile mettere in esecuzione sia la parte client che la parte server, entrando nelle rispettive directory, previo download dei pacchetti necessari al funzionamento degli stessi, mediante il package manager npm.

Per avviare il server è necessario avviare una shell all'interno della directory server, posizionandosi quindi alla radice del progetto, e digitare il comando `node index.js`. Per avviare il client, la procedura sarà la stessa del server, entrando però nella directory client, e avviare il seguente comando `yarn run start` dalla shell. Per evitare errori durante l'esecuzione, poichè il server rimane in ascolto sulla porta 3000 e il client sulla porta 8000, è necessario verificare che non ci siano altri processi già in ascolto sulle rispettive porte.

Sarà quindi possibile accedere all'applicazione al seguente indirizzo:
`http://localhost:8000`

Per questioni di sicurezza, si è deciso di non caricare sul repository alcuni file di configurazione contenenti le chiavi dei servizi utilizzati (ad esempio la chiave per utilizzare il servizio di invio delle email, la chiave per accedere ai contenuti delle API di Tmdb, la chiave utilizzata da JWT). E' comunque possibile però, provare una demo grazie al rilascio del software sulla piattaforma Heroku.

6.2 Rilascio su Heroku

Si è deciso di utilizzare la piattaforma Heroku per il rilascio del software. Per fare ciò, si è pensato di creare un nuovo repository su GitHub, modificando nel progetto, le informazioni di configurazione affinchè le richieste vengono effet-

tuate sul nuovo host. Così facendo ad ogni richiesta pull fatta sul repository, Heroku effettua una nuova build, mantenendo aggiornata la demo rispetto al repository. Su Heroku è stato caricata sia la parte server che client dell'applicativo, mentre per la parte di persistenza è stato creato un cluster su Atlas MongoDb che comunica direttamente con Heroku.

Il repository per il deploy è visitabile al seguente link
`https://github.com/pagno4/MoviePlatformDeploy`.

Al link `https://movieplatform.herokuapp.com/` è possibile provare una versione demo dell'applicativo.

Per accedere e testare il sistema si possono usare le seguenti credenziali:

- **Username**: admin oppure lorenzo.pagnini2@studio.unibo.it
- **Password**: moviePlatform2021

Conclusioni

Al termine dell'elaborato, posso sostenere di essere soddisfatto del lavoro ottenuto. Realizzare questo progetto mi ha permesso di approfondire tutto quello che concerne la parte di sviluppo web. Inoltre ho avuto la possibilità di approfondire l'uso di tecnologie poco conosciute, fino a qualche tempo fa, e aumentare di conseguenza la mia conoscenza. La curiosità, mi ha portato a cercare di utilizzare il maggior numero di librerie e framework apprezzandone gli svantaggi e vantaggi di ognuna di esse. Nonostante qualche imperfezione di design e/o di *user experience*, portando a termine questo progetto da solo, viste anche le numerose difficoltà incontrate durante lo sviluppo, sono contento del risultato prodotto.