Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica per il Management

C.I.N.E.: APPLICAZIONE WEB PER ESTRARRE E GENERARE INFOGRAFICHE SU DATI CINEMATOGRAFICI

Relatore:
Chiar.mo Prof.
ANGELO DI IORIO

Presentata da:
SIMONE TOSI

Sessione III
2023/2024

Indice

Introduzione

Questa tesi tratta l'implementazione di un'applicazione Web, denominata C.I.N.E.(*Cinema Information and Notable Exploration*), che estrae i dati di vari film forniti in input, li elabora e genera delle visualizzazioni significative. Tra queste, un grafo che evidenzia le connessioni tra film, attori e registi, e sette grafici che confrontano le varie informazioni raccolte, come la lunghezza(in caratteri) delle trame, oppure l'ambientazione dei film.
Lo sviluppo dell'applicazione è nato dall'idea di raccogliere e salvare dati in ambito cinematografico, avendo a disposizione un sito che faceva da database per i film che avevano come argomento principale la Virtual Reality.[2] L'idea di base era, prendere i titoli dei film presenti nel sito, cercare un modo per estrarre le informazioni su di essi e utilizzarle per la creazione delle visualizzazioni citate nel paragrafo precedente.
I dati dei film vengono estratti da Wikipedia e Wikidata, eseguendo una richiesta API per recuperare le informazioni da Wikipedia e una richiesta SPARQL per accedere ai dati strutturati di Wikidata.
Inoltre, è stata effettuata un'analisi di alcuni sistemi esistenti che presentano funzionalità affini a C.I.N.E.. I sistemi trovati sono stati IMDB, Moviegalaxies e Tableau Public.
Dopo la parte di sviluppo, sono stati eseguiti dei test per osservare le varie visualizzazioni create. Per fare ciò sono stati utilizzati due dataset, il primo composto da 44 film e il secondo composto, invece, da 12 film.
La struttura della tesi è la seguente:

- Il primo capitolo analizza lo stato dell'arte relativo ad applicazioni simili.
- Il secondo capitolo descrive le funzionalità principali dell'applicazione e approfondisce le informazioni che vengono estrapolate per ogni film.
- Il terzo capitolo si concentra sull'implementazione del progetto, descrivendo le varie funzioni utilizzate e mostrando la struttura interna dell'applicazione. Approfondisce inoltre come un'applicazione esterna possa integrare le funzionalità di C.I.N.E.

- Il quarto capitolo descrive i vari test che sono stati effettuati sull'applicazione e presenta i risultati finali.
- Il quinto, ed ultimo, capitolo racconta i punti di forza e di debolezza di C.I.N.E. e i possibili sviluppi futuri.

Capitolo 1

Sistemi di visualizzazione

Durante lo sviluppo dell'applicazione, è stata effettuata una ricerca per trovare dei sistemi che avessero delle funzionalità simili a C.I.N.E.. I risultati della ricerca hanno portato a tre applicazioni principali: IMDB, Moviegalaxies e Tableau Public. Di seguito verrano descritti e analizzati i tre sistemi trovati.

1.1 IMDb

IMDb(Internet Movie Database)[1] è un sito Web proprietà di Amazon che cataloga, archivia film, attori, registi, personale di produzione, programmi televisivi, e anche videogiochi. È considerato il più grande database cinematografico online e viene utilizzato da appassionati di cinema, critici e professionisti del settore per ottenere informazioni dettagliate sulle opere audiovisive. IMDb organizza i suoi contenuti in diverse sezioni, che includono:

- **Film e Serie TV**: raccolta di tutti i dati relativi, tra cui titolo, anno di uscita, durata, genere, registi, sceneggiatori, produttori, cast principale, sinossi, budget, incassi al botteghino e altro.
- **Persone**: Attori, registi, produttori, montatori, sceneggiatori e altri membri dello staff tecnico con biografie dettagliate, filmografia e premi ricevuti.
- **Videogiochi**: Elenco di videogiochi con informazioni sugli sviluppatori, doppiatori e trama.
- **Recensioni e valutazioni**: Gli utenti possono assegnare voti da 1 a 10, che contribuiscono a formare il punteggio IMDb di un film.
- **Curiosità, errori, citazioni**: Sezione dedicata a dettagli dietro le quinte, errori di continuità e citazioni famose dei film.

Tuttavia, non tutte le informazioni sono accessibili gratuitamente: mentre le schede principali dei film e degli attori sono disponibili per tutti gli utenti, i dati più dettagliati, come le informazioni tecniche sulla produzione, i contatti dell'industria cinematografica e alcune statistiche avanzate, richiedono l'iscrizione a **IMDbPro**.

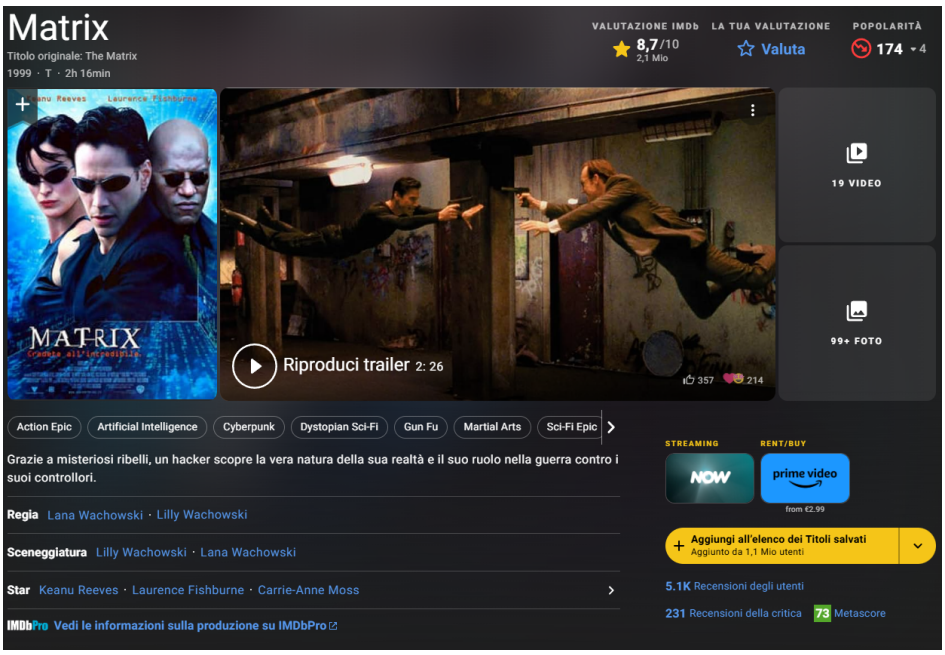

Figura 1.1: Esempio di schermata di IMDb del film The Matrix(1999)

Oltre all'archiviazione delle informazioni, IMDb offre anche statistiche dettagliate e classifiche basate sulle valutazioni degli utenti. Tra le più note vi sono:

- **Top 250 Movies**: classifica dei 250 film con il punteggio più alto assegnato dagli utenti.
- **Bottom 100**: elenco dei 100 film con le peggiori valutazioni.
- **Trending Movies & TV Shows**: lista dei film e delle serie TV più cercate e discusse sul sito.

Tuttavia, IMDb non fornisce degli strumenti per creare delle visualizzazioni significative utilizzando questi dati. Non presenta la possibilità di creare dei grafici che mettano in relazione tra di loro determinate informazioni come l'ambientazione di vari film per ottenere quella più diffusa. Inoltre, non offre strumenti per rappresentare i collegamenti tra film, attori e registi sotto forma di grafi. Questa limitazione rende difficile effettuare delle analisi avanzate delle informazioni, rendendo obbligatorio l'utilizzo di strumenti esterni.

1.2 Moviegalaxies

Moviegalaxies[3] è una piattaforma che genera grafi e altre visualizzazioni minori basate sulle relazioni tra personaggi di un film, mostrando come interagiscono tra loro nel corso della trama. I creatori hanno utilizzato un approccio basato sull'apparizione nella stessa scena per costruire queste reti sociali. Viene analizzata la sceneggiatura e ogni volta che due personaggi appaiono nella stessa scena viene creato un collegamento. Per ogni nodo, Moviegalaxies raccoglie:

- **id**: Identificatore univoco del personaggio.
- **name**: Il nome del personaggio.
- **color**: Il colore del nodo all'interno del grafo. Utilizzato per distinguere personaggi e gruppi.
- **group**: Un numero che indica il gruppo di appartenenza del personaggio.
- **degree**: Il numero totale di connessioni che ogni personaggio ha con altri personaggi. Più è alto questo valore più il personaggio è centrale nella rete.
- **pagerank**: Una misura dell'importanza del nodo nel grafo. Più alto è il valore, maggiore è l'importanza del personaggio rispetto agli altri.
- **triangles**: Il numero di triangoli in cui il personaggio è coinvolto. Un triangolo è formato da tre personaggi che si conoscono reciprocamente.
- **eccentricity**: La distanza massima di un personaggio dagli altri nella rete. Più il valore è basso e più il personaggio è centrale.
- **closeness centrality**: Una misura di quanto sia vicino un nodo agli altri nodi della rete. Valori più alti indicano che il personaggio può interagire rapidamente con altri personaggi.
- **betweenness centrality**: Indica il numero di volte in cui un personaggio funge da ponte tra altri due personaggi. Un valore elevato suggerisce che il personaggio ha un ruolo chiave nella comunicazione tra altri.
- **eigenvector centrality**: Una misura dell'importanza di un nodo basata non solo sul numero di connessioni, ma anche sulla qualità di quelle connessioni. Un personaggio ben connesso a personaggi influenti avrà un'eigenvector centrality più alta.

Di seguito è riportato un esempio di grafo sociale sul film The Matrix(1999).

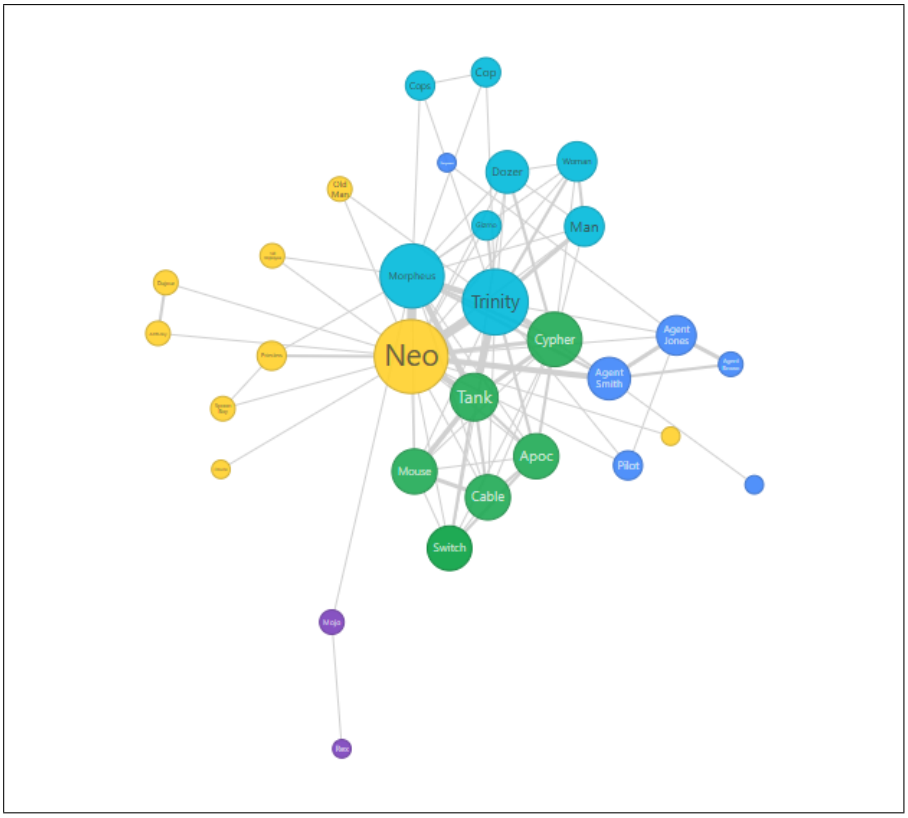

Figura 1.2: Esempio di un grafo sociale di Moviegalaxies del film The Matrix(1999)

Di seguito, invece, sono riportate le informazioni relative al nodo del personaggio di Neo del grafo precedente.[4]

```
{
    "id": "NEO",
    "name": "NEO",
    "color": "#FFD12B",
    "group": 2,
    "degree": 23,
    "pagerank": 0.149237101764019,
    "triangles": 54,
    "eccentricity": 3,
    "closeness centrality": 1.26666666666667,
    "betweenness centrality": 197.978571428571,
    "eigenvector centrality": 0
}
```

Listing 1.1: Informazioni sul nodo del personaggio di Neo

Tuttavia, a differenza di C.I.N.E. , Moviegalaxies si concentra esclusivamente sulle connessioni tra personaggi, senza includere altri tipi di analisi, rendendo questo sistema limitato.

1.3 Tableau Public

Tableau Public è uno strumento di data visualization che permette di creare dashboard interattive basate su dataset. Questo è l'applicativo, tra i tre che sono stati trovati, che più si avvicina a C.I.N.E, perché permette di creare un vasto numero di visualizzazioni utilizzando dei dataset che contengono informazioni sui film che, in questa applicazione, vengono recuperati tramite IMDb. Dopo un'attenta ricerca su Tableau Public, non sono state trovate visualizzazioni che sono invece presenti su C.I.N.E.. Infatti, non è stata trovata una visualizzazione che raggruppi i film per luogo di ambientazione, che confronti la differente lunghezza delle trame, o che analizzi l'ambientazione temporale del film utilizzando una linea del tempo.
Tra le visualizzazioni su Tableau Public che analizzano gli stessi dati che sono raccolti da C.I.N.E, ne è stata trovata una che mostra i film migliori raggruppati per genere e mostra anche altre informazioni su registi e attori. In questa visualizzazione è possibile filtrare per data di pubblicazione, lingua originale del film, genere e voto. I dati vengono visualizzati nei tre grafici sottostanti. Il primo mostra i film migliori tenendo conto dei filtri indicati. Il secondo, in basso a sinistra, mostra, selezionando un film dal primo grafico, una tabella contenente il/i regista/i e gli attori che hanno preso parte alla pellicola. Il terzo, in basso a destra, mostra le persone che durante l'anno hanno avuto più ruoli come attori e/o registi.[5]

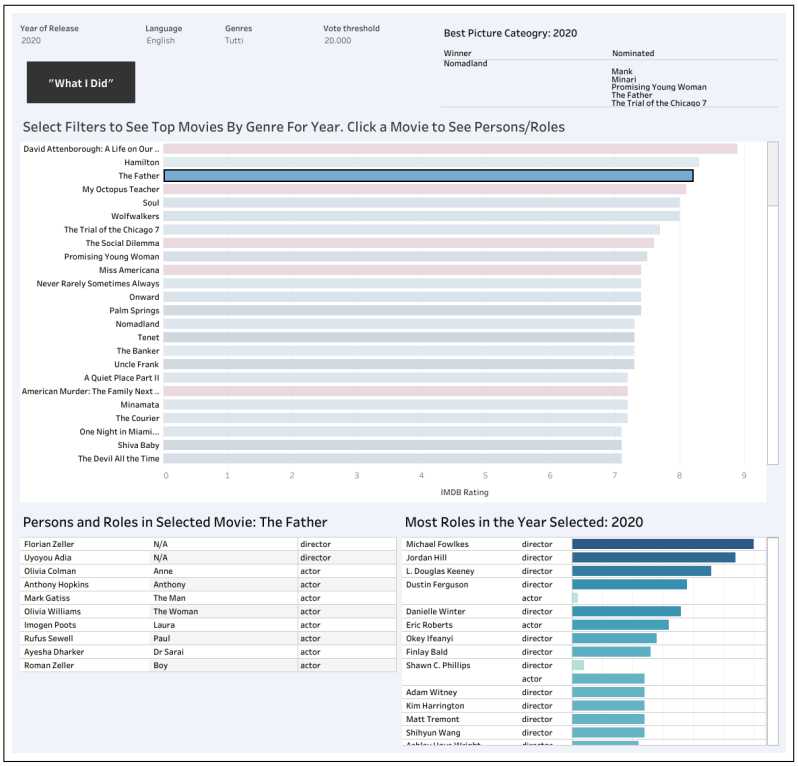

Figura 1.3: Prima visualizzazione relativa ai generi

Passando alle visualizzazioni che includono una linea del tempo, molte di queste si concentrano sulla data di pubblicazione del film anziché sulla sua ambientazione temporale interna. Ad esempio, una di queste rappresenta i dieci migliori film di Ridley Scott, secondo il rating di IMDb, disposti su una linea del tempo in base all'anno di uscita.[7]

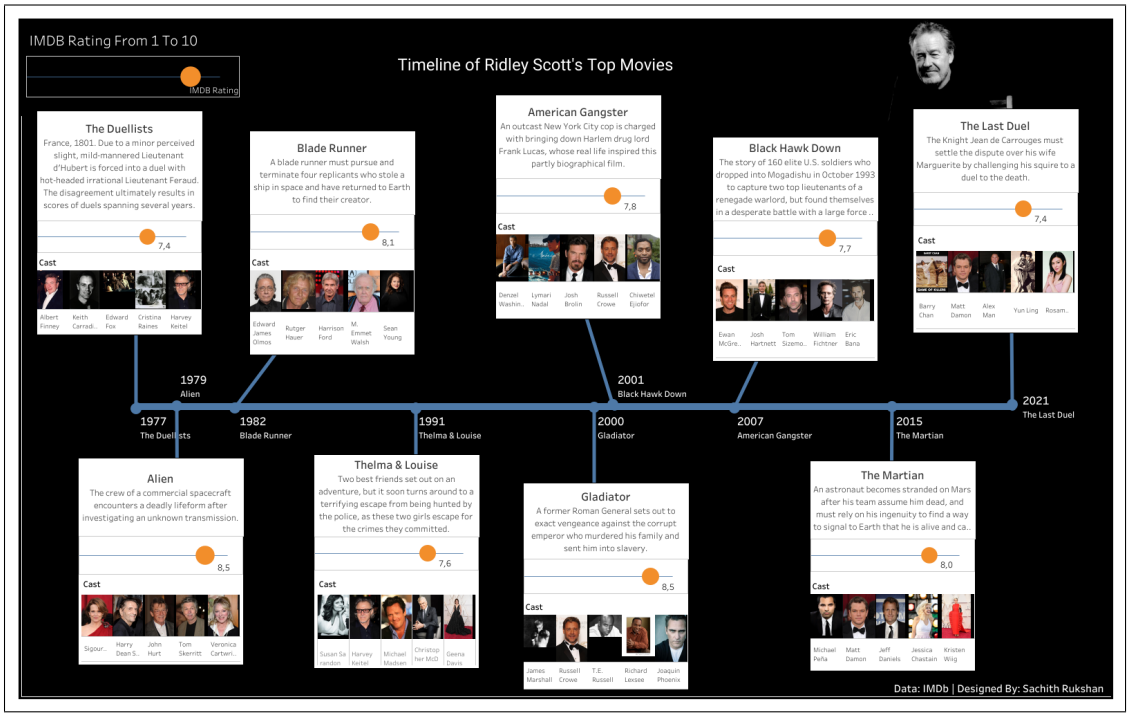

Figura 1.4: Seconda visualizzazione che mostra la timeline dei film di Ridley Scott

L'analisi di Tableau Public evidenzia la potenza dello strumento nella creazione di visualizzazioni, offrendo un'ampia gamma di rappresentazioni grafiche. Tuttavia, non include le visualizzazioni specifiche implementate su C.I.N.E, rendendo necessaria, se si desidera, la loro creazione su Tableau Public. Inoltre, le visualizzazioni presenti non analizzano la trama del film cercando di raccogliere determinate informazioni, ma si limitano a lavorare su dati facilmente reperibili come generi, data e paese di pubblicazione.

1.4 Conclusioni

L'analisi dei tre sistemi ha evidenziato come ognuno di essi abbia delle funzionalità ben precise, che però non si avvicinano in modo significativo a quelle di C.I.N.E.. IMDb presenta un ingente numero di informazioni sui film, ma non strumenti per poterle analizzare e creare delle visualizzazioni. Moviegalaxies presenta la possibilità di creare un grafo ma limitato alle sole interazioni dei personaggi all'interno del film. Infine, Tableau Public, permette di creare un numero significativo di visualizzazioni differenti, ma in questo momento, non presenta quelle che sono invece disponibili su C.I.N.E..
In conclusione, questa ricerca ha confermato l'utilità di sviluppare un'applicazione in grado di raccogliere informazioni sui film e generare visualizzazioni mirate, analizzando in particolare i dati ricavati dall'analisi della trama e altri dati di rilievo.

Capitolo 2

Descrizione del sistema

C.I.N.E. è un'applicazione Web che permette all'utente di navigare tra diverse visualizzazioni che utilizzano i dati estratti da due fonti differenti: Wikidata e Wikipedia.
Le visualizzazioni presenti nell'applicazione includono un grafo e sette grafici. Il grafo consente di visualizzare le relazioni tra il film, i registi e gli attori. Ad esempio, è possibile individuare quali registi hanno diretto più film o quanti attori hanno partecipato a più pellicole. I sette grafici, invece, permettono di visualizzare:

- Il numero di film ambientati in un determinato pianeta(Terra, Marte, Fantasy World ecc.)
- Il numero di film ambientati in un determinato Continente(Europa, Asia, America ecc.)
- Il numero di film ambientati in una specifica nazione(Italia, Giappone ecc.)
- L'anno o gli anni in cui i film sono ambientati.
- L'anno o gli anni in cui i film sono ambientati in relazione al loro anno di uscita.
- La lunghezza(in caratteri) delle trame dei film.
- Il numero di luoghi e date che sono state identificate per ogni film.

L'applicazione presenta quattro schermate principali: **Home**, **Film**, **Graph** e **Charts**. Queste, verranno analizzate nel dettaglio nelle prossime sezioni.

2.1 Sorgenti

Come descritto nella sezione precedente, i dati dei film vengono estratti da due fonti differenti: Wikipedia e Wikidata.

- Wikidata fornisce dati strutturati, tra cui il titolo del film, l'anno di uscita, i generi, il/i regista/i, gli attori e i relativi personaggi interpretati.

- Wikipedia viene utilizzata per recuperare informazioni non disponibili su Wikidata, tra cui la trama e il poster del film.

Inoltre, è presente uno script Python che analizza la trama estratta da Wikipedia per ottenere le date e i luoghi.

2.2 Homepage dell'applicazione

La homepage è la prima schermata mostrata all'utente e fornisce una panoramica delle operazioni disponibili all'interno dell'applicazione. Inoltre, vengono visualizzati i dati relativi al numero di film attualmente caricati nel sistema. Tramite il pulsante `Show loaded movies`, l'utente può visualizzare l'elenco dei film caricati. Selezionando il titolo di un film, vengono mostrate le relative informazioni dettagliate.

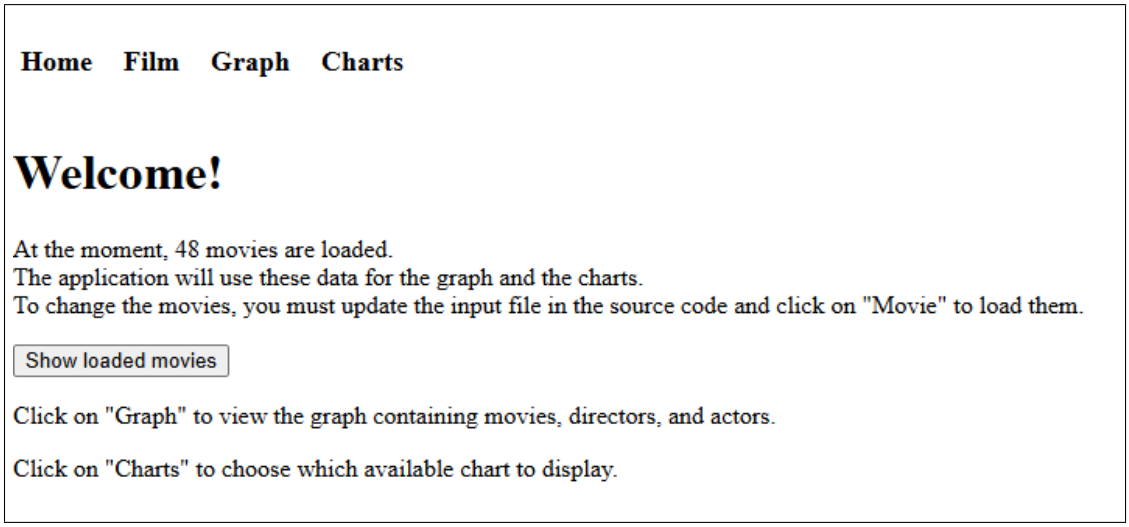

Figura 2.1: Homepage dell'applicazione con film caricati

Se l'applicazione viene avviata per la prima volta e non sono ancora stati caricati film, viene mostrato un messaggio che avvisa l'utente dell'assenza di dati e lo invita a navigare nella sezione `Film` per procedere con il caricamento.

2.3 Caricamento dei film

L'utente, semplicemente cliccando la voce `Film` nella barra di navigazione, avvia l'estrazione e il successivo salvataggio dei dati dei film da parte dell'applicazione. Se i dati sono stati salvati correttamente, viene mostrato l'elenco dei titoli dei film appena ottenuti. Cliccando su un titolo, l'utente viene reindirizzato a una schermata dedicata, in cui sono visualizzate tutte le informazioni relative al film selezionato. Se l'utente accede nuovamente a questa schermata dopo aver già eseguito il caricamento dei dati, questo non verrà eseguito di nuovo ma verranno mostrati i dati salvati in precedenza.

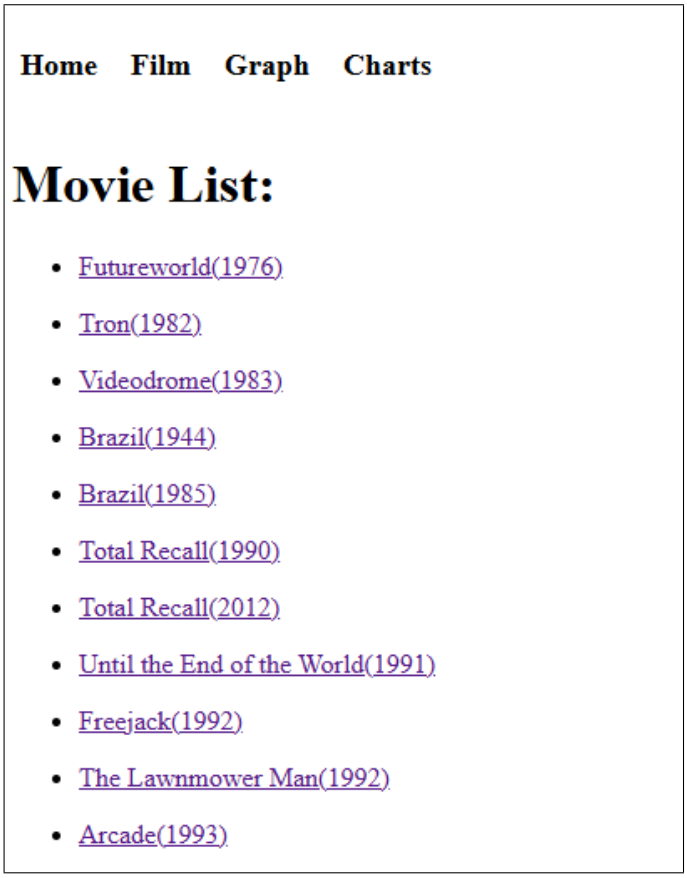

Figura 2.2: Schermata Film

2.4 Rete di relazioni tra film, registi e attori

L'utente, in questa schermata, può creare il grafo per visualizzare le relazioni tra film, registi e attori. Per farlo, è necessario cliccare il pulsante `Create Graph`. Questa azione genererà un grafo contenente solamente i nodi relativi ai titoli dei film. L'utente può aggiungere e togliere i nodi dei registi e degli attori a piacimento semplicemente spuntando le checkbox `Director` e/o `Actor`. Ogni nodo è contraddistinto da un colore diverso: i film sono colorati di azzurro, i registi sono colorati di verde mentre gli attori sono colorati di rosso se hanno recitato solo in un film e di giallo se hanno preso parte a più produzioni.

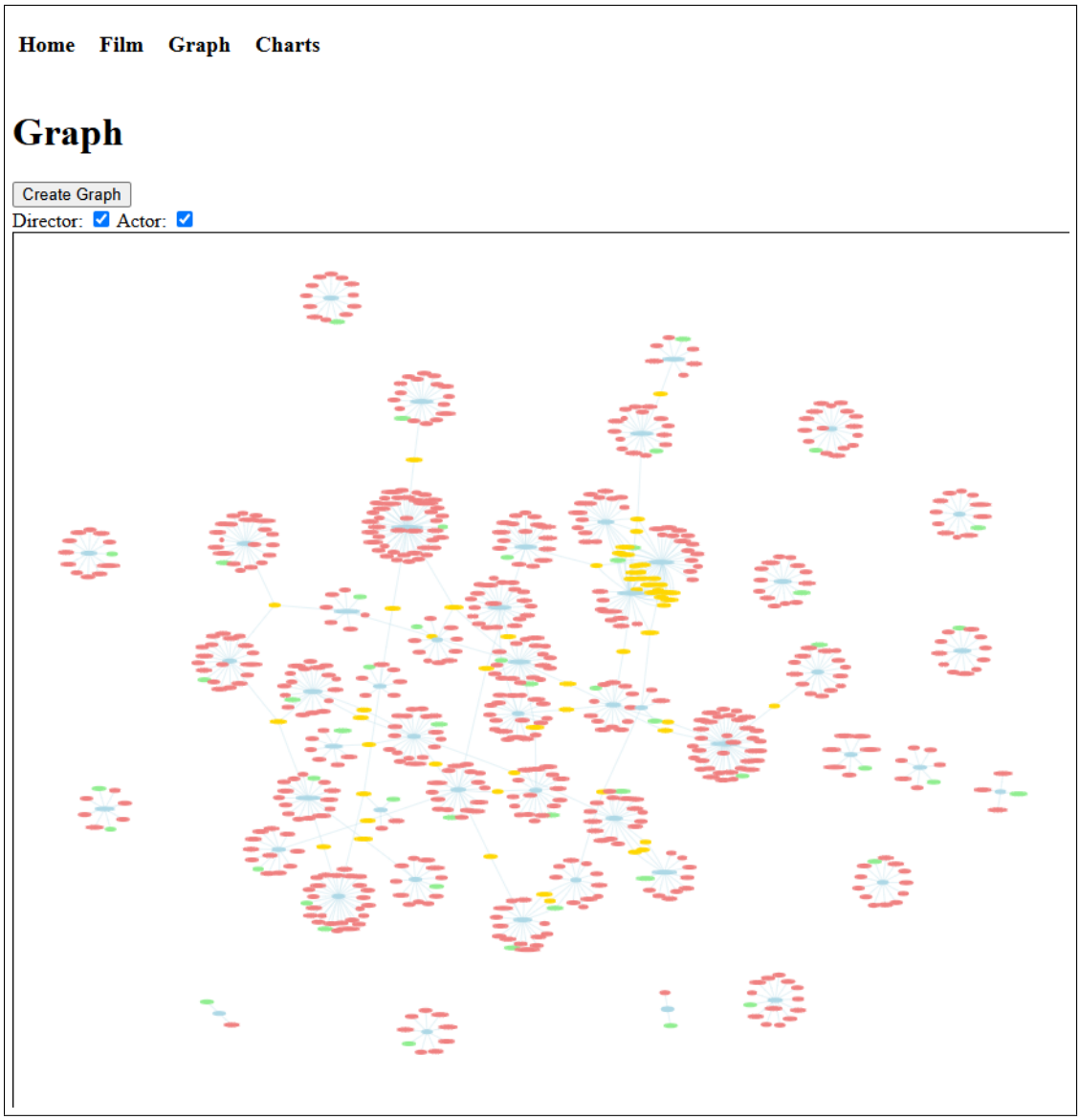

Figura 2.3: Schermata Graph

2.5 Introduzione sui grafici

In questa schermata, l'utente può selezionare il grafico di interesse semplicemente cliccando sul suo nome. Ogni grafico è accompagnato da una breve descrizione che ne illustra il contenuto.

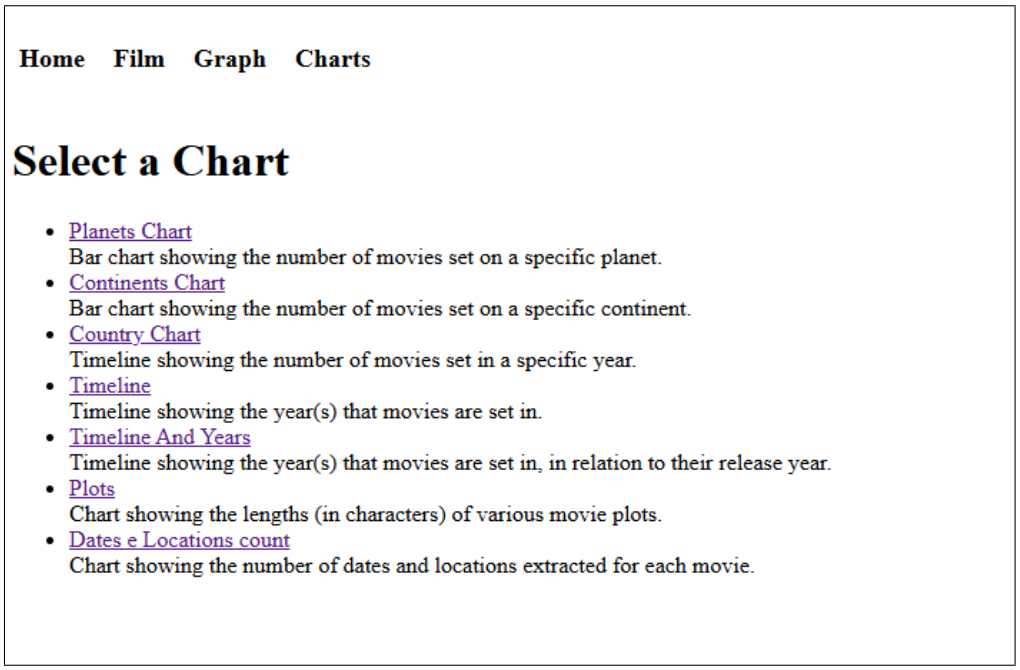

Figura 2.4: Schermata Charts

Di seguito vengono descritti i contenuti dei grafici presenti in C.I.N.E.. Per brevità verrà mostrato il primo grafico che offre l'applicazione, i restanti verranno presentati in un capitolo successivo.

2.5.1 Grafici sull'ambientazione dei film

Questi grafici a barre mostrano i film ambientati in specifiche località. Il primo mostra il numero di film ambientati in un determinato pianeta(Terra, Marte, Fantasy World ecc.), il secondo mostra il numero di film ambientati in un determinato continente(Europa, Asia, America ecc.), infine il terzo mostra il numero di film che sono ambientati in una determinata nazione(Italia, Giappone ecc.). L'utente, passando il mouse sopra una barra, può visualizzare l'elenco dei film ambientati in quel luogo specifico. Inoltre, ogni grafico presenta una legenda in cui è indicato il numero totale di film analizzati, il numero di film che sono presenti nel grafico, il numero di film che compaiono solamente in un luogo e il numero di quelli che sono ambientati in più località.

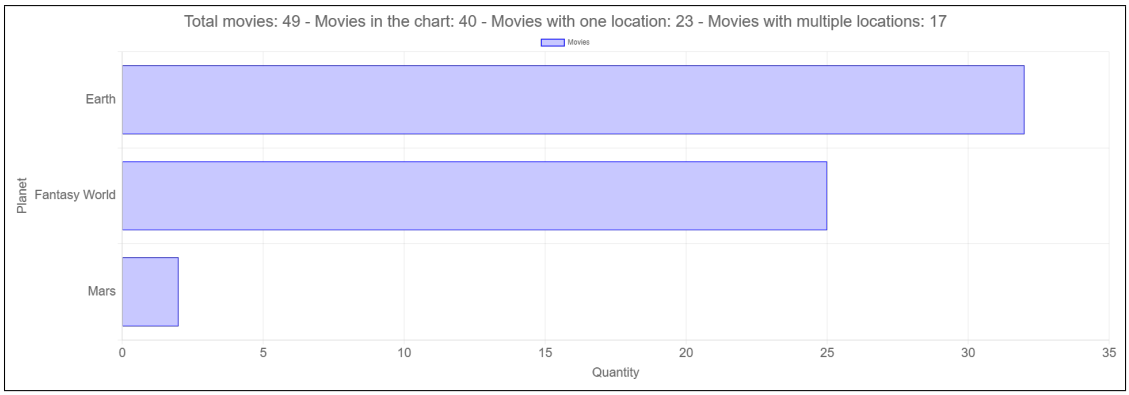

Figura 2.5: Grafico che raggruppa i film per pianeta

2.5.2 Grafici sulla collocazione temporale dei film

Questi grafici a barre analizzano la distribuzione temporale del film. Il primo rappresenta l'intervallo di tempo in cui è ambientato ciascun film, mentre il secondo mette in relazione questo intervallo con l'anno di uscita del film, permettendo di osservare quanto l'ambientazione temporale del film si discosti dalla data di rilascio.

2.5.3 Grafico sulla lunghezza delle trame dei film

Questo grafico a barre confronta la lunghezza (in caratteri) delle trame dei film presenti. Inoltre, include un'indicazione della lunghezza media delle trame. Il grafico e la media sono utili per valutare la qualità dei dati su cui opera lo script Python per l'analisi delle date e dei luoghi.

2.5.4 Grafico sul numero di date e luoghi identificati

Questo grafico a barre mostra il numero di date e di luoghi che sono stati estratti dallo script Python.

2.6 Integrazione con un sistema esterno

L'applicazione, oltre alle funzionalità precedentemente descritte, consente di integrare la generazione dei vari grafici in un'applicazione esterna. Questo permette di utilizzare i vari grafici supportati da C.I.N.E. , a condizione che i dati impiegati per la loro creazione siano relativi a film e formattati un modo specifico. I dettagli implementativi verranno forniti nel capitolo successivo.

Capitolo 3

Dettagli implementativi

L'applicazione C.I.N.E. è stata sviluppata utilizzando Node.js per la gestione del backend, mentre il frontend è stato realizzato con HTML, CSS e JavaScript. Il backend, basato su Express.js, espone una serie di API REST, che consentono al frontend di recuperare, tramite richieste eseguite dagli script JavaScript, le informazioni necessarie sui film per la creazione del grafo e dei grafici.
Di seguito viene mostrato il diagramma di sequenza di C.I.N.E. che illustra il percorso che porta dall'estrazione al salvataggio delle informazioni sui film.

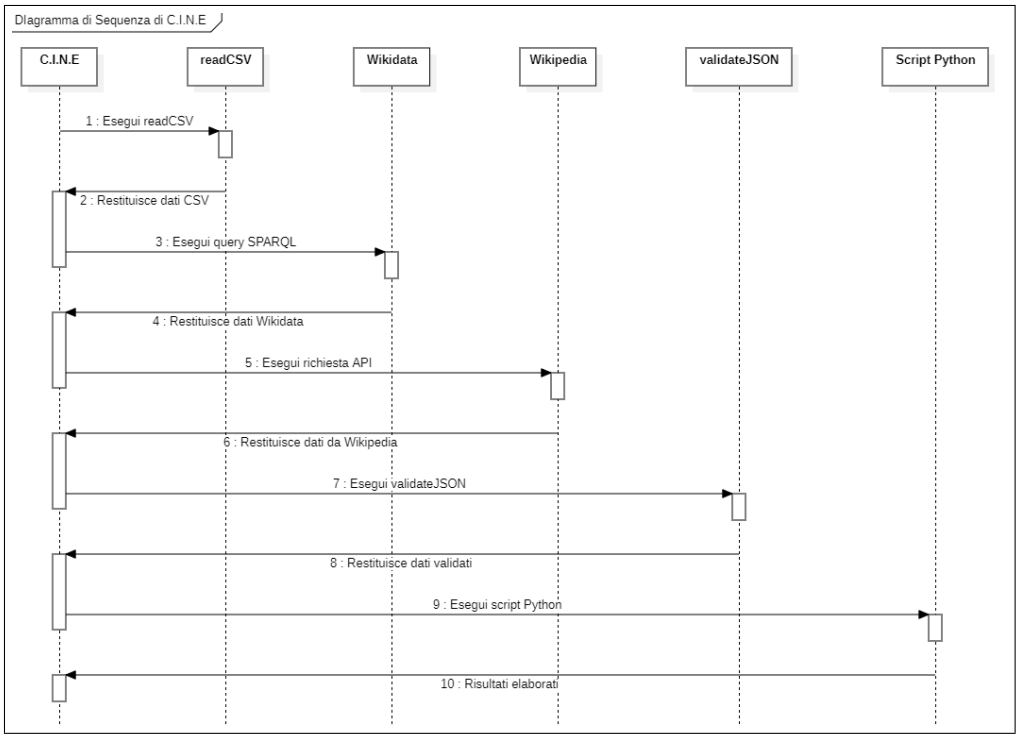

Figura 3.1: Diagramma di sequenza di C.I.N.E.

3.1 Struttura del progetto

L'applicazione è divisa in due parti principali: backend e frontend. Il codice backend è contenuto nella cartella `src`, mentre il codice frontend è contenuto nella cartella `public`. Per mostrare l'integrazione della funzionalità di creazione dei grafici di C.I.N.E. in un progetto esterno, è stato aggiunto un progetto di esempio nella cartella `examples/external-projects`. Questo aspetto verrà approfondito in una sezione successiva.

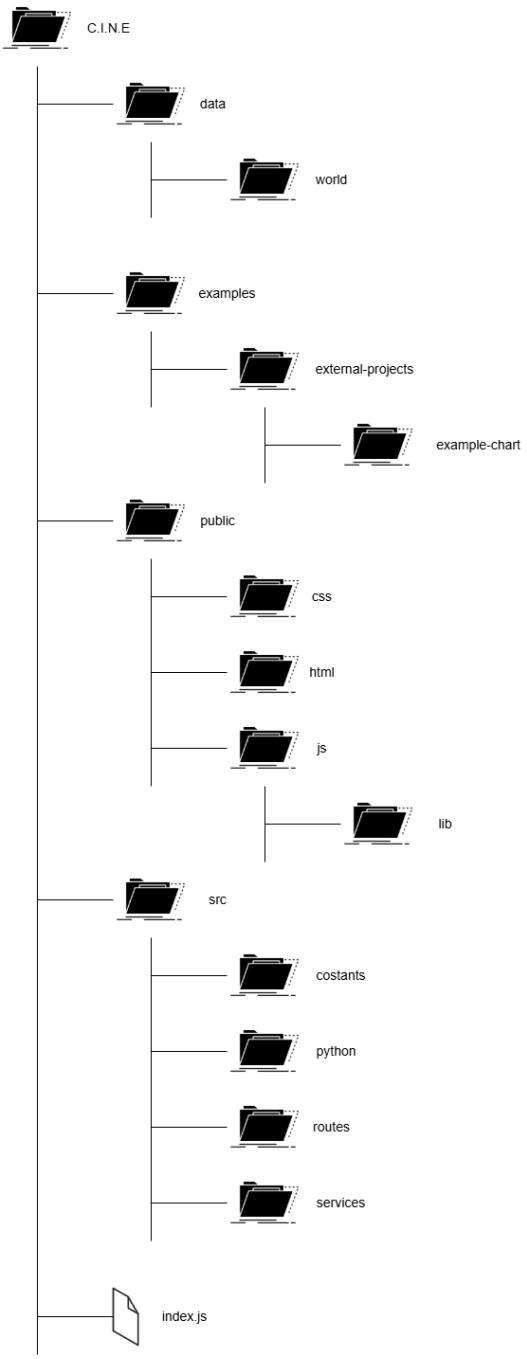

Figura 3.2: Schema delle directory (parte 1)

3.2 Flusso dei dati: operazioni interne e interazione con API esterne

Le informazioni sui film vengono estratte da due fonti differenti: Wikipedia tramite richiesta API e Wikidata tramite richiesta SPARQL. Inoltre viene eseguito uno script Python, interno al progetto, per l'estrazione di date e luoghi dalla trama e vengono eseguite due funzioni principali: la prima legge i titoli dei film da analizzare da un file CSV, mentre la seconda valida i dati ricevuti da Wikipedia e Wikidata controllando che non ci siano campi vuoti.
Le informazioni sui film sono archiviate in un file JSON, che funge da database locale, strutturato nel modo seguente:


```
[
 {
    "id": "Q83495",
    "title": "The Matrix",
    "director": [
      "Lana Wachowski",
      "Lilly Wachowski"
    ],
    "year": "1999",
    "genres": [
      "dystopian",
      "action thriller"
    ],
    "cast": {
      "actors": [
        "Keanu Reeves",
        "Laurence Fishburne",
        "Carrie-Anne Moss",
      ],
      "characters": [
        "Neo",
        "Morpheus",
        "Trinity",
      ]
    },
    "plot": "In 1999, in an unnamed city, Thomas Anderson, a
       computer programmer known as Neo in hacking circles,
       delves into the mystery of the Matrix.",
    "poster": "https://en.wikipedia.org/wiki/Special:FilePath/
       The_Matrix_Poster.jpg",
    "isValid": true,
    "temporalInfo": {
      "id": "Q83495",
```

```
      "locations": [
        "Earth",
        "Zion"
      ],
      "dates": [
        "1999"
      ]
    }
  }
]
```

Listing 3.1: Struttura di esempio del file JSON

Inoltre, per centralizzare alcuni valori chiave dell'applicazione, è stato definito un file di costanti `constants.js`. Questo file contiene:

- schema, host e porta del server Node.js
- i percorsi dei file di input e output
- i valori che non devono essere vuoti nel JSON affinché sia considerato valido
- l'endpoind di Wikidata

```
export default Object.freeze({
    SCHEMA: 'http',
    HOST: 'localhost',
    PORT: 3000,
    INPUT_FILE_PATH: path.resolve('data', 'input.csv'),
    OUTPUT_FILE_PATH: path.resolve('data', 'output.json'),
    REQUIRED_FIELDS_FOR_VALID_JSON: ['id', 'title', 'year', '
       director', 'genres', 'cast', 'plot'],
    WIKIDATA_ENDPOINT: "https://query.wikidata.org/sparql"
})
```

Listing 3.2: Struttura del file constants.js

L'endpoint che esegue le richieste a Wikidata, Wikipedia e allo script Python è `/process`, definito nel file `api.js`, e viene richiamato ogni volta che l'utente accede alla pagina `Film`. Per evitare di effettuare una nuova richiesta a ogni visita, è stato implementato un sistema di caching. A tale scopo, è stato definito un oggetto `const cache = {}` nel quale i dati vengono salvati al primo accesso. Quando l'utente torna sulla schermata `Film`, i dati vengono recuperati dalla cache anziché effettuare nuovamente la richiesta. Questo meccanismo è gestito tramite il seguente codice:

```
const cache = {}

router.get('/process', async (req, res) => {
    try{
        const cacheKey = 'filmsData'

        if(cache[cacheKey]){
            console.log('Dati ottenuti dalla cache')
            return res.json({ message: 'Dati ottenuti dalla cache
                ', data: cache[cacheKey] })
        }

        //Codice per l'elaborazione dei dati...

        cache[cacheKey] = finalData.filter(film => film.isValid)

        //Codice per il salvataggio su file...

        res.json({ message: 'Processo completato', data: cache[
            cacheKey] })
    } catch (error) {
        console.log(error)
        res.status(500).json({ error: 'Errore durante il processo
            ', details: error.message })
    }
})
```

Listing 3.3: Cache

L'endpoint, come prima operazione, richiama la funzione `export function readCSV(filePath)` per leggere il file `csv` che contiene i titoli dei film di cui si vogliono recuperare le informazioni.

```
export function readCSV(filePath){
    return new Promise((resolve, reject) => {
        const results = []
        fs.createReadStream(filePath)
        .pipe(csv())
        .on('data', (row) => {
            results.push(row)
        })
        .on('end', () => {
            resolve(results)
        })
        .on('error', (error) => reject(error))
    })
}
```

Listing 3.4: Codice della funzione readCSV()

Successivamente, viene effettuata una richiesta a Wikidata tramite la funzione `export async function fetchWikidata(title)` che prende in input il titolo del film. Questa funzione invia una richiesta all'endpoint `https://query.wikidata.org/sparql`, passando una query SPARQL come parametro.

```
SELECT DISTINCT
?film
?filmLabel
(GROUP_CONCAT(DISTINCT ?directorLabel; separator=", ") AS ?
   directors)
(GROUP_CONCAT(DISTINCT ?genreLabel; separator=", ") AS ?
   genres)
(GROUP_CONCAT(DISTINCT ?actorLabel; separator=", ") AS ?
   actors)
(GROUP_CONCAT(DISTINCT ?characterLabel; separator=", ") AS ?
   characters)
(MIN(?releaseDate) AS ?firstReleaseDate)
WHERE {
    ?film wdt:P31 wd:Q11424;          # Film
            rdfs:label "${title}"@en;
            wdt:P57 ?director;          # Director
            wdt:P136 ?genre;            # Genre
            wdt:P577 ?releaseDate;      # Release date
            wdt:P161 ?actor.            # Actor

    OPTIONAL {
        ?film p:P161 [
        ps:P161 ?actor;
        pq:P453 ?character
        ].
    }

    SERVICE wikibase:label {
        bd:serviceParam wikibase:language "en".
        ?director rdfs:label ?directorLabel.
        ?film rdfs:label ?filmLabel.
        ?genre rdfs:label ?genreLabel.
        ?actor rdfs:label ?actorLabel.
        ?character rdfs:label ?characterLabel.
    }
}
GROUP BY ?film ?filmLabel
```

Listing 3.5: Query SPARQL

Se la richiesta va a buon fine, viene effettuata una seconda richiesta a Wikipedia, richiamando la funzione `export async function fetchWikipedia(wikimediaId)` che prende in input l'ID della risorsa di Wikidata.
La funzione `fetchWikipedia()` esegue inizialmente una richiesta a Wikidata per ottenere il link della risorsa su Wikipedia in lingua inglese. Se disponibile, procede con una richiesta alle API di Wikipedia, interrogando l'endpoint `https://en.wikipedia.org/w/api.php` per ottenere la revisione della pagina della risorsa desiderata. Da questa, vengono estratti trama e poster del film. Se il film non esiste su Wikipedia, questo non viene salvato.
Di seguito viene mostrato il codice principale delle due funzioni appena descritte:

```
export async function fetchWikidata(title){
    try{
        const response = await axios.get(constants.
            WIKIDATA_ENDPOINT, {
            params: {
                query,
                format: "json"
            },
            headers: {
                "Accept": "application/json"
            }
        })

        //Codice per la costruzione degli oggetti con i dati
            presi da Wikidata...

    } catch(error) {
        console.error("Errore durante la richiesta SPARQL:",
            error)
        return null
    }
}
```

Listing 3.6: Codice della funzione fetchWikidata()

```
export async function fetchWikipedia(wikimediaId) {
    try {
        const response = await axios.get('https://www.wikidata.
            org/w/api.php', {
            params: {
                action: "wbgetentities",
                format: "json",
```

```
                props: "sitelinks",
                ids: wikimediaId,
                sitefilter: "enwiki"
            }
        })

        const data = response.data;

        if (data.entities && data.entities[wikimediaId] && data.
            entities[wikimediaId].sitelinks && data.entities[
            wikimediaId].sitelinks.enwiki) {
            const title = data.entities[wikimediaId].sitelinks.
                enwiki.title;
            try {
                const response = await axios.get('https://en.
                    wikipedia.org/w/api.php', {
                    params: {
                        action: "query",
                        prop: "revisions",
                        rvprop: "content",
                        rvslots: "main",
                        format: "json",
                        titles: title
                    }
                })

                //Codice per la costruzione dell'oggetto con
                    trama e poster...

            } catch (error) {
                console.error("Errore durante il recupero delle
                    informazioni da Wikipedia:", error)
            }
        } else {
            console.error("Il titolo non e' disponibile per l'
                entit  richiesta.", wikimediaId);
            return null
        }
    } catch (error) {
        console.log("Errore durante il recupero della trama da
            Wikipedia:", error)
    }
}
```

Listing 3.7: Codice della funzione fetchWikipedia()

Una volta ottenute tutte le informazioni, viene eseguita la funzione `export async function validateJSON(inputData)`, che verifica la presenza di campi vuoti e aggiunge una variabile booleana `isValid` a ciascun film. Questa viene impostata su `true` se il film ha tutti i campi necessari, altrimenti su `false`.

```
export async function validateJSON(inputData) {
    try{
        const json = Array.isArray(inputData) ? inputData : [
           inputData]

        const updatedJSON = json.map((item) => {
            const isValid = constants.
               REQUIRED_FIELDS_FOR_VALID_JSON.every((field) =>
                item[field] && (Array.isArray(item[field]) ? item
                   [field].length > 0 : item[field] !== "")
            )

            return {
                ...item,
                isValid: isValid
            }
        })

        return updatedJSON.filter(film => film.isValid)
    } catch(error) {
        console.error("Errore durante la validazione:", error)
    }
}
```

Listing 3.8: Codice della funzione validateJSON()

Dopo la validazione, viene richiamato tramite l'utilizzo di `spawn`, lo script Python che analizza la trama ed estrae i luoghi e le date. Alla prima categoria appartengono tutti i luoghi geografici, come pianeti, stati, continenti e città, mentre nella seconda categoria rientrano gli anni, ad esempio 2025, e anche espressioni temporali come "Next Day" o "50 years ago". Per questa operazione viene utilizzata la libreria `spaCy`, con il modello `en_core_web_lg`. Lo script implementa tre funzioni principali:

- **def normalize_location(location)**: prende in input un luogo ed esegue operazioni di normalizzazione. Queste comprendono l'eliminazione degli articoli "the" e "a" all'inizio del luogo e l'eliminazione di "'s" alla fine. Inoltre, viene effettuata la sostituzione di "u.s." e "america" con "united states" e "new york city" con "new york".

```
def noramlize_location(location):
    location = location.lower()
```

```
    if(location.startswith("the ")):
        location = location.replace("the ", "", 1)
    if location.startswith("a "):
        location = location.replace("a ", "", 1)
    location = location.replace("'s", "")
    location = location.replace("u.s.", "united states").
       replace("america", "united states")
    location = location.replace("new york city", "new york")
    location.strip()
    return location.title()
```

Listing 3.9: Codice dello script Python

- **def noramlize_date(date)**: prende in input una data ed esegue operazioni di normalizzazione. Queste comprendono l'eliminazione di "the", "year", "late" e "early" all'inizio della data e l'eliminazione di "s" alla fine.

```
def noramlize_date(date):
    date = date.lower()
    if(date.startswith("the ")):
        date = date.replace("the ", "", 1)
    if(date.startswith("year ")):
        date = date.replace("year ", "", 1)
    if(date.startswith("late ")):
        date = date.replace("late ", "", 1)
    if(date.startswith("early ")):
        date = date.replace("early ", "", 1)
    date.strip()
    if(date.endswith("s") and date[:-1].isdigit()):
        date = date[:-1]
    date.strip()
    return date.title()
```

Listing 3.10: Codice dello script Python

- **def extract_entities(data)**: analizza la trama e restituisce un oggetto JSON contenente l'ID del film, i luoghi e le date estratte.

```
nlp = spacy.load("en_core_web_lg")

def extract_entities(data):
    ruler = nlp.add_pipe("entity_ruler", before="ner")
    patterns = [
        {"label": "LOC", "pattern": [{"LOWER": {"REGEX": r"\b
            \w+(?:world)$"}}]}
    ]
```

```
    ruler.add_patterns(patterns)
    results = []

    for item in data:
        film_id = item.get("id")
        plot = item.get("plot")
        if(plot and plot != ''):
            doc = nlp(plot)

            locations = set(ent.text for ent in doc.ents if
                ent.label_ in ["GPE", "LOC"])
            dates = set(ent.text for ent in doc.ents if ent.
                label_ == "DATE")

            noramlized_locations = set(noramlize_location(
                location) for location in locations)
            noramlized_dates = set(noramlize_date(date) for
                date in dates)

            extracted = {
                "id": film_id,
                "locations": list(noramlized_locations),
                "dates": list(noramlized_dates)
            }

            results.append(extracted)

    return results

//Blocco main...
```

Listing 3.11: Codice dello script Python

Di seguito viene mostrato il codice JavaScript che richiama lo script Python tramite spawn.

```
const runPythonScript = (data) => {
    return new Promise((resolve, reject) => {
        const pythonProcess = spawn('python', ['src/python/
            temporal_info.py'])
        let output = ''
        let errorOutput = ''

        pythonProcess.stdin.write(JSON.stringify(data))
        pythonProcess.stdin.end()

        pythonProcess.stdout.on('data', (data) => {
```

```
            output += data.toString()
        })

        pythonProcess.stderr.on('data', (data) => {
            errorOutput += data.toString()
        })

        pythonProcess.on('close', (code) => {
            if(code !== 0) {
                console.error('Errore nello script Python:',
                   errorOutput)
                reject(new Error('Errore nello script Python: ' +
                    errorOutput))
            }else {
                try {
                    const extractedData = JSON.parse(output);
                    resolve(extractedData);
                } catch (error) {
                    console.error('Errore durante il parsing dell
                       \'output Python:', error);
                    reject(new Error('Errore durante il parsing
                       dell\'output Python: ' + error.message));
                }
            }
        })
    })
}
```

Listing 3.12: Codice JavaScript che richiama lo script Python

3.3 Implementazione delle funzioni JavaScript lato client

Oltre alle funzioni per recuperare i dati dei film dalle varie fonti, sono presenti anche funzioni lato client che gestiscono la visualizzazione delle informazioni sui film, la creazione del grafo e dei vari grafici. Per ottenere i dati necessari, queste funzioni invocano la funzione `async function fetchFilmData()`, che richiama l'endpoint `/films`. Questo endpoint legge il file JSON in cui sono archiviate le informazioni sui film e le restituisce al client. Di seguito viene riportato il codice della funzione e dell'endpoint.

```
async function fetchFilmData() {
    const response = await fetch('/api/films')
    if (!response.ok) {
        throw new Error('Errore nel recupero dei dati')
    }
    return response.json()
}
```

Listing 3.13: Codice della funzione fetchFilmData()

```
router.get('/films', async (req, res) =>{
    try{
        const data = fs.readFileSync(constants.OUTPUT_FILE_PATH,
            'utf-8');
        const films = JSON.parse(data)
        res.json(films)
    } catch(error){
        console.error(error)
        res.status(500).json({ error: 'Errore durante il recupero
            dei dati dei film', details: error.message })
    }
})
```

Listing 3.14: Codice dell'endpoint /films

3.3.1 Funzioni per la visualizzazione delle informazioni sui film

Le funzioni utilizzate per la visualizzazione delle informazioni sui film sono contenute nei file `film.js` e `films.js`. Questi file gestiscono rispettivamente la visualizzazione delle informazioni di un singolo film e il caricamento dell'elenco completo dei film, richiamando l'endpoint `/api/process`.
Nel primo file, `film.js`, per ottenere le informazioni di un singolo film, viene eseguita una richiesta all'endpoint `/api/film/:id`. Utilizzando l'ID del film passato come parametro nell'URL, il server esegue una ricerca nel file JSON e restituisce al frontend i dati della risorsa richiesta.
Il codice che esegue questa operazione è il seguente:

```
app.get('/api/film/:id', async (req, res) => {
    const id = req.params.id
    try{
        const data = await fs.readFile(constants.OUTPUT_FILE_PATH
            , 'utf-8')
```

```
        const films = JSON.parse(data)
        const film = films.find(item => item.id === id)

        if(film){
            res.json(film)
        } else {
            res.status(404).json({ error: 'Film non trovato' })
        }
    } catch (error) {
        console.error('Errore durante la lettura del file JSON:',
            error)
        res.status(500).json({ error: 'Errore interno del server'
            })
    }
})
```

Listing 3.15: Codice del file film.js

Di seguito viene riportato il codice dei file `film.js` e `films.js`.

```
document.addEventListener('DOMContentLoaded', async () => {
    //Inizializzazione elementi HTML

    const id = decodeURIComponent(window.location.pathname.split(
       '/').pop())

    try {
        const response = await fetch('/api/film/' + id)
        if (!response.ok) {
            throw new Error('Errore nel caricamento dei dettagli
               del film')
        }

        const film = await response.json();

        titleElement.textContent = film.title
        plotElement.textContent = film.plot
        posterElement.src = film.poster
        posterElement.alt = 'Poster: ' + film.title

        detailsElement.innerHTML =
        '<li> <b>Year: </b>' + film.year + '</li>' +
        '<li> <b>Director: </b>' + film.director + '</li>' +
        '<li> <b>Genres: </b>' + film.genres.join(', ') + '</li>'
            +
        '<li> <b>Actors: </b>' + film.cast.actors.join(', ') + '
           </li>' +
```

```
        '<li> <b>Characters: </b>' + film.cast.characters.join(',
            ') + '</li>'

        film.temporalInfo.locations.forEach(location => {
            placesElement.innerHTML +=
            '<li>' + location + '</li>'
        });

        film.temporalInfo.dates.forEach(date => {
            datesElement.innerHTML +=
            '<li>' + date + '</li>'
        });


    } catch (error) {
        console.error(error);
        document.body.innerHTML = '<p>Error loading data</p>'
    }
})
```

Listing 3.16: Codice del file film.js

```
document.addEventListener('DOMContentLoaded', async () => {
    const resultsContainer = document.getElementById('results')
    const loadingContainer = document.getElementById('loading')

    try{
        loadingContainer.style.display = 'block'

        const response = await fetch('/api/process')

        if (!response.ok) {
            throw new Error('Errore nella richiesta al server')
        }

        const result = await response.json()
        const data = result.data

        console.log(data.length)

        resultsContainer.innerHTML = ''

        data.forEach(film => {
            if(film.isValid){
                const div = document.createElement('div')
                div.classList.add('film')
                div.innerHTML =
```

```
                '<ul>' +
                '<li><a href="/film/' + encodeURIComponent(film.
                    id) + '">' + film.title + '(' + film.year + ')
                    </a></li>' +
                '</ul>'
                resultsContainer.appendChild(div)
            }
        })

        loadingContainer.style.display = 'none'
    } catch (error){
        console.error(error);
        results.innerHTML = '<p>Error loading data</p>'
        loadingContainer.style.display = 'none'
    }
})
```

Listing 3.17: Codice del file films.js

3.3.2 Funzioni per la creazione del grafo

Per la creazione del grafo è stata utilizzata la libreria `vis-network`. Il grafo viene creato richiamando la funzione `function updateGraph()`, la quale viene chiamata ogni volta che l'utente preme il pulsante `Create Graph` e spunta le checkbox per registi e/o attori. Quando viene premuto il pulsante, viene inizialmente creato un grafo contenente solo i nodi relativi ai film tramite il seguente codice:

```
document.addEventListener('DOMContentLoaded', async () => {
    try {
        //Codice iniziale...

        graphButton.addEventListener('click', async () => {

            let network = null
            const nodes = new vis.DataSet()
            const edges = new vis.DataSet()
            const addedNodes = new Set()
            const addedEdges = new Set()

            films.forEach(film => {
                const filmId = film.title + '(' + film.year + ')'
                if (!addedNodes.has(filmId)) {
                    nodes.add({ id: filmId, label: filmId, color:
                        'lightblue' })
                    addedNodes.add(filmId)
                }
```

```
            })

            //Mostra schermata caricamento...

            const container = document.getElementById('mynetwork'
                )
            const options = {
                //Opzioni
            }
            network = new vis.Network(container, { nodes, edges
                }, options)

            //Nasconde schermata di caricamento...

            function updateGraph() {
                //Codice che aggiorna il grafo...
            }

            directorCheckbox.addEventListener('change',
                updateGraph)
            actorCheckbox.addEventListener('change', updateGraph)
        })
    } catch(error) {
        console.error(error)
    }
})
```

Listing 3.18: Codice del file graph.js

Se l'utente seleziona le checkbox, i nodi corrispondenti a registi e attori vengono aggiunti dinamicamente, insieme alle relative connessioni. Tuttavia, se l'utente seleziona le checkbox senza prima aver prima generato il grafo base, non accadrà nulla poiché la funzione si basa sui nodi già esistenti per effettuare le modifiche. Di seguito viene mostrato il codice che aggiorna il grafo con i nodi dei registi e/o degli attori. Per brevità viene mostrato solo il codice relativo all'aggiunta dei nodi dei registi, in quanto quello per gli attori è analogo.

```
document.addEventListener('DOMContentLoaded', async () => {
    try {
            //Codice iniziale...

            function updateGraph() {
                //Calcolo tempo di caricamento...

                const nodes = new vis.DataSet()
                const edges = new vis.DataSet()
```

```
                const addedNodes = new Set()
                const addedEdges = new Set()

                //Mostra schermata di caricamento...


                //Aggiunta nodi dei film...

                    if(directorCheckbox.checked) {
                        const directors = Array.isArray(film.
                            director) ? film.director : [film.
                            director]
                        directors.forEach(director => {
                            if (!addedNodes.has(director)) {
                                nodes.add({ id: director, label:
                                    director, color: 'lightgreen'
                                    })
                                addedNodes.add(director)
                            }
                            const directorEdge = `${filmId}-${
                                director}`
                            if (!addedEdges.has(directorEdge)) {
                                edges.add({ from: filmId, to:
                                    director })
                                addedEdges.add(directorEdge)
                            }
                        })
                    } else {
                        //Rimozione nodi e archi dei registi...
                    }

                //Aggiorna con nodi degli attori...

                    network.setData({ nodes, edges })
                    //Nasconde schermata di caricamento...
            }

            directorCheckbox.addEventListener('change',
                updateGraph)
            actorCheckbox.addEventListener('change', updateGraph)
        })
    } catch(error) {
        console.error(error)
    }
})
```

Listing 3.19: Codice del file graph.js

3.3.3 Funzioni per la creazione dei grafici

Il codice per la creazione dei grafici è contenuto nel file `charts-lib.js` nella cartella `public/js/lib`. Questo file contiene una singola funzione `export async function createGraph(films, dataset, canvas, downloadButton)` che viene richiamata al caricamento della pagina. Questa funzione accetta quattro parametri in ingresso:

- **films**: i dati su cui generare i grafici. Devono essere in formato JSON e seguire la struttura precedentemente descritta.
- **dataset**: il tipo di grafico da generare.
- **canvas**: l'elemento HTML `<canvas>` in cui verrà disegnato il grafico. Questo elemento deve avere un ID.
- **downloadButton**: l'ID dell'elemento HTML `<button>` utilizzato per scaricare il grafico. Questo parametro è facoltativo.

Il file che richiama questa funzione è `charts.js` contenuto nella cartella `public/js` che contiene, oltre alla chiamata alla funzione, un'altra funzione, `function getQueryParameter(name)`, che estrae il valore del parametro di query corrispondente al tipo di grafico selezionato dall'utente. Questo parametro viene passato nell'URL quando l'utente sceglie quale grafico visualizzare nella schermata dedicata.
La funzione adibita alla creazione dei grafici è stata spostata in un file separato per consentire la sua integrazione in un'applicazione esterna. Questo passaggio verrà approfondito in una sezione successiva.
Per la creazione dei grafici che mostrano quanti film sono ambientati in un determinato luogo, è stato effettuato un mapping dei luoghi ottenuti dallo script Python. Questa operazione viene eseguita utilizzando quattro dataset presenti nella cartella `data/world`:

- **planets**: contiene i nomi dei pianeti principali.
- **continents**: contiene i nomi dei continenti.
- **countries**: contiene i nomi delle nazioni del rispettivo contiente e subcontinente.
- **cities**: contiene i nomi delle città e delle rispettive nazioni e dei rispettivi stati.

L'oggetto risultante dal mapping è una struttura dati in cui ogni elemento ha come chiave il nome del luogo e presenta i seguenti campi:

- **type**: il tipo di luogo(pianeta, continente, nazione o città).
- **continent**: continente di appartenenza(se applicabile).
- **country**: nazione di appartenenza(se applicabile).
- **city**: città di appartenenza(se applicabile).
- **titles**: un array contenente i titoli dei film ambientati in quel luogo.

Di seguito viene riportato il codice dei file `charts.js` e `charts-lib.js`. A titolo esemplificativo, viene mostrato solamente il codice per la creazione del primo grafico sui luoghi.

```
function getQueryParameter(name) {
    const urlParams = new URLSearchParams(window.location.search)
    return urlParams.get(name)
}

document.addEventListener('DOMContentLoaded', async () => {
    const films = await fetchFilmData()
    const dataset = getQueryParameter('dataset')
    const canvas = document.getElementById('chart')
    const downloadButton = document.getElementById('downloadChart
        ')

    await createGraph(films, dataset, canvas, downloadButton)
})
```

Listing 3.20: Codice del file charts.js

```
export async function createGraph(films, dataset, canvas,
   downloadButton){

    //Funzioni per la preparazione dei dati per la creazione dei
       grafici...

    if(dataset === 'planet'){
        const mappingData = await mappingLocations()
        const locationData = await prepareLocationData(
           mappingData)
        const planetData = locationData['planet']

        const location = await locationsLabel(planetData)

        const labels = Object.keys(planetData).sort()
        const data = labels.map(location => planetData[location].
           titles.length)
```

```
        const titles = labels.map(location => planetData[location
            ].titles)
        canvas.height = 100
        const ctx = canvas.getContext('2d')

        const planetChart = new Chart(ctx, {
            type: 'bar',
            data: {
                labels: labels,
                datasets: [
                    {
                        label: 'Movies',
                        data: data,
                        backgroundColor: 'rgb(200, 200, 255)',
                        borderColor: 'rgb(0, 0, 255)',
                        borderWidth: 1.5
                    }
                ]
            },
            options: {
                //Opzioni
                plugins: {
                    //Impostazioni per legenda e plugin
                },
                scales: {
                    //Impostazioni per gli assi X e Y
                }
            },
        })

        downloadChart(planetChart, dataset)
    } else if(dataset === 'continent'){
        //Codice per la creazione del grafico
    } else if(dataset === 'country'){
        //Codice per la creazione del grafico
    } else if(dataset === 'timeline'){
        //Codice per la creazione del grafico
    } else if(dataset === 'timeline_years'){
        //Codice per la creazione del grafico
    } else if(dataset === 'plots'){
        //Codice per la creazione del grafico
    } else if(dataset === 'dates_locations'){
        //Codice per la creazione del grafico
    }

    function downloadChart(chart, dataset){
        //Codice per il downlod del grafico
```

```
    }
}
```

Listing 3.21: Codice del file charts-lib.js

3.4 Implementazione dell'integrazione con un sistema esterno

Per permettere ad un sistema esterno di integrare la creazione dei grafici nella propria applicazione, è stato necessario rendere visibile esternamente il file JavaScript `charts-lib.js`. Per fare ciò, l'applicazione utilizza Express per servire le risorse statiche dalla directory `public`, permettendo ai sistemi esterni di includere direttamente il file tramite un semplice tag `<script>`. Per testare questa funzionalità è stato creato un progetto Node.js di prova che implementasse la libreria. Di seguito verrà descritto il processo per la corretta implementazione. Prima di tutto, è necessario implementare un file HTML e un file JavaScript. Nel file HTML sarà necessario importare tre moduli dalla sorgente:

- **https://cdn.jsdelivr.net/npm/chart.js**: libreria base per la creazione dei grafici.
- **https://cdn.jsdelivr.net/npm/chartjs-plugin-annotation**: plugin per `chart.js` per includere annotazioni all'interno del grafico.
- **/js/charts.js**: libreria di C.I.N.E. che gestisce la creazione dei grafici.

Per caricare la libreria `charts.js` di C.I.N.E. all'interno del progetto esterno è necessario implementare una funzione `async function loadChartLibrary()` che che importa dinamicamente la libreria dall'endpoint in cui è ospitata. Il tutto è stato reso modulabile tramite costanti per rendere il cambio di endpoint semplice e immediato. In questo caso le costanti vengono gestite tramite un `Object.freeze` e vengono recuperate tramite il metodo `async function fetchConstants()`. Tuttavia, possono essere gestite nel modo preferito dallo sviluppatore.
Di seguito viene riportato un esempio di due file, uno HTML e uno JavaScript, che implementano quanto descritto sopra:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
        scale=1.0">
```

```
    <script type="module" src="https://cdn.jsdelivr.net/npm/chart
        .js"></script>
    <script type="module" src="https://cdn.jsdelivr.net/npm/
        chartjs-plugin-annotation"></script>
    <script type="module" src="/js/charts.js"></script>
    <title>Grafico</title>
</head>
<body>
    <h1>Grafico</h1>
    <button id="downloadChart">Download Chart</button>
    <canvas id="chart"></canvas>
</body>
</html>
```

Listing 3.22: Esempio di implementazione nel file HTML

```
async function fetchConstants() {
  const response = await fetch('/api/constants')
  if(!response.ok){
      throw new Error('Errore nel recupero delle costanti')
  }
  return response.json()
}

async function loadChartLibrary(){
  const constants = await fetchConstants()

  const module = await import(constants.SCHEMA + '://' +
     constants.HOST + ':' + constants.PORT + constants.ENDPOINT)

  return module
}

document.addEventListener('DOMContentLoaded', async () => {
    const films = [..]
    const dataset = 'plots'
    const canvas = document.getElementById('chart')

    const { createGraph } = await loadChartLibrary()

    await createGraph(films, dataset, canvas, 'downloadChart')
})
```

Listing 3.23: Esempio di implementazione nel file JavaScript

Capitolo 4

Casi di studio

Per eseguire dei test su C.I.N.E. , sono stati utilizzati due dataset diversi. Il primo contiene film che hanno come argomento comune la Virtual Reality, che sono stati presi dal sito individuato all'inizio dello sviluppo. Mentre il secondo contiene tutti i film del regista Cristopher Nolan. È stato scelto questo come secondo dataset perché è composto da pochi film rispetto al primo e questo è utile per vedere come si comporta l'applicazione con dataset limitati.
Durante i test, è stato osservato che l'applicazione estrae le informazioni di tutti i film che hanno lo stesso titolo di quelli presenti nel file di input. Questo accade perché il file di input contiene solo i titoli, senza altre informazioni, e di conseguenza Wikidata restituisce tutti i film con lo stesso nome. Pertanto, i test risultano leggermente alterati.

4.1 Dataset sui film a tema Virtual Reality

In questo dataset sono presenti 44 film:

- The World of Robots
- Futureworld
- Tron
- Videodrome
- Brazil
- Total Recall
- Until the End of the World
- Freejack
- The Lawnmower Man
- Arcade
- Brainscan
- Evolver
- Johnny Mnemonic
- Hackers
- Strange Days
- Ghost in the Shell
- Contact
- Dark City
- The Matrix
- The Thirteenth Floor

- eXistenZ
- The Cell
- Avalon
- The Matrix Reloaded
- The Matrix Revolutions
- Gamer
- The World of Replicants
- Avatar
- Inception
- Tron: Legacy
- The Congress
- Pacific Rim
- The Zero Theorem
- Vice
- Humandroid
- Assassin's Creed
- Sword Art Online: The Movie - Ordinal Scale
- Ready Player One
- Pacific Rim: Uprising
- Alita: Battle Angel
- Spider-Man: Far From Home
- Coma
- Archive
- Bliss

4.1.1 Rete di relazioni tra film, registi e attori

Il grafo generato quando l'utente clicca sul pulsante `Create Graph` contiene solamente i nodi dei film, senza nessun collegamento tra di loro.

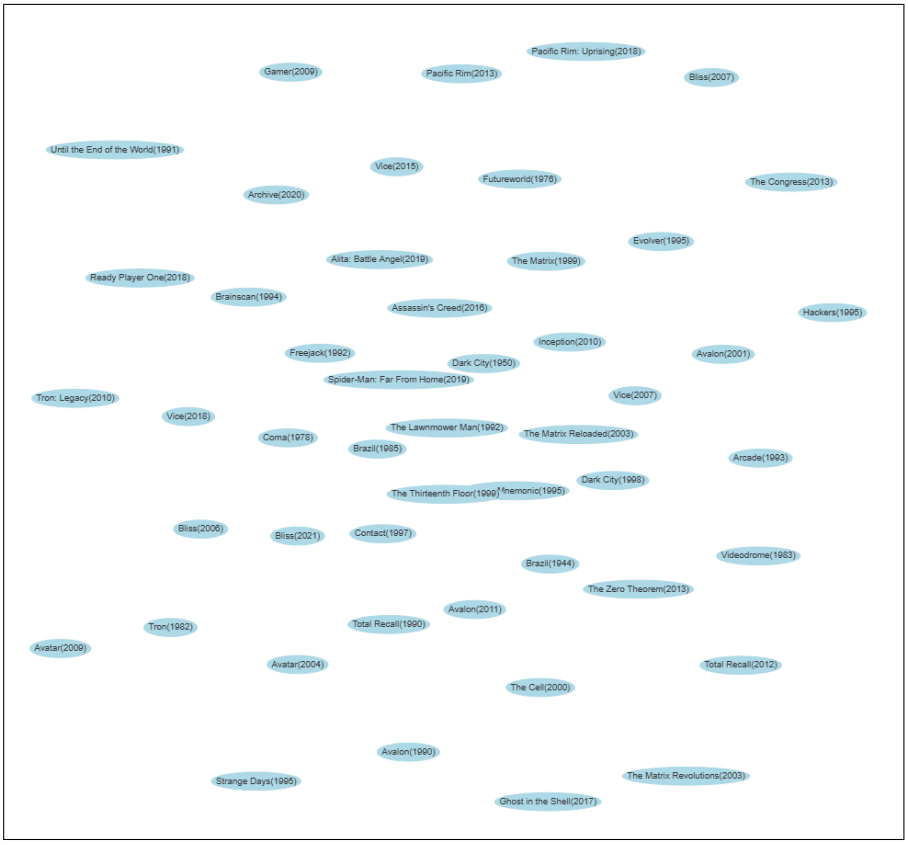

Figura 4.1: Grafo contenente i nodi dei film

Aggiungendo i nodi dei registi, è possibile osservare che pochi di loro hanno diretto più di un film. Questo accade solo per la trilogia di The Matrix, curata da Lana e Lilly Wachowski, mentre tutti gli altri registi presenti hanno diretto un unico film. Questo può essere dato dall'eterogeneità dei film che, anche se hanno un argomento comune, sono comunque molto diversi tra di loro.

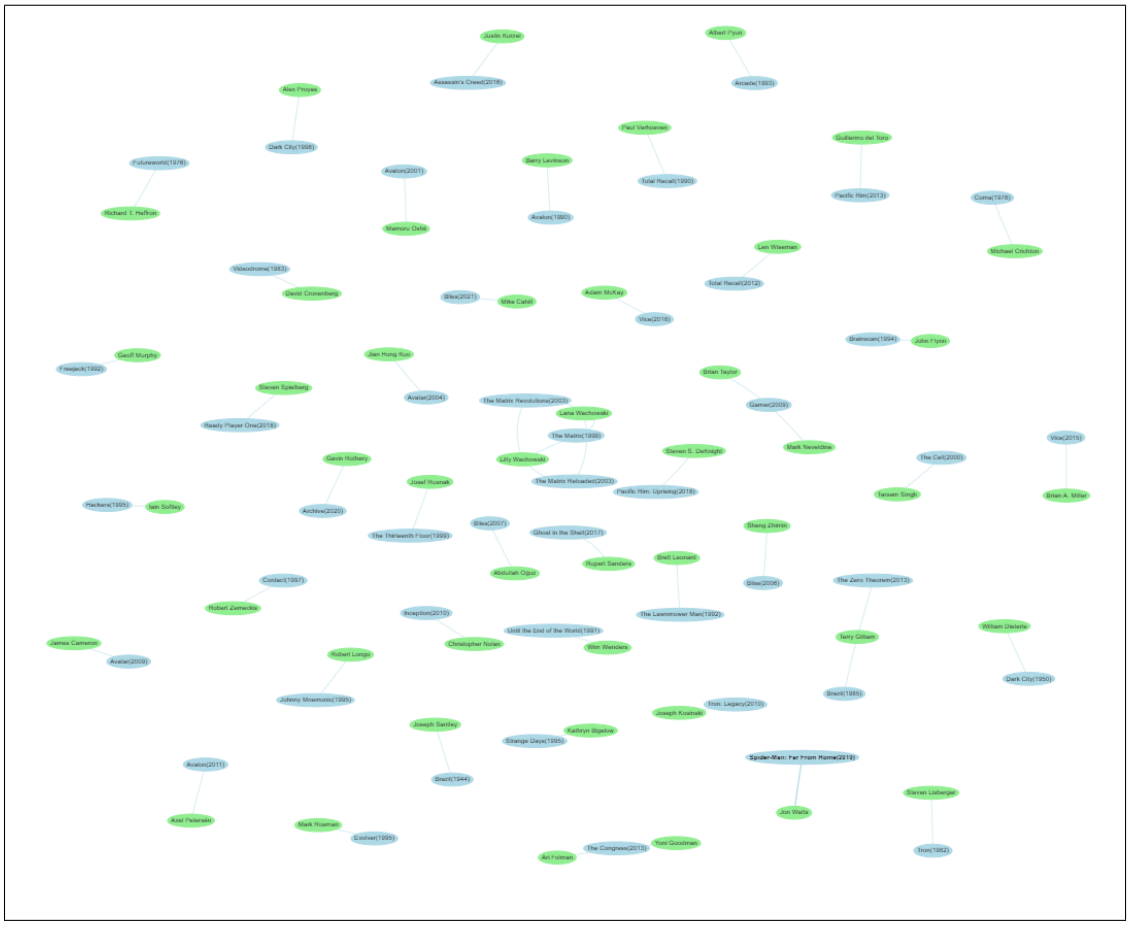

Figura 4.2: Grafo contenente i nodi dei film e dei registi

Aggiungendo anche i nodi degli attori, emergono alcune informazioni interessanti. In particolare, si nota la presenza di 15 film isolati, su un totale di 49, senza alcun collegamento con il resto del grafo. Tuttavia, 6 di questi non appartengono al dataset iniziale per via del problema descritto in precedenza. Di conseguenza, i film effettivamente isolati dal resto del dataset sono 9.

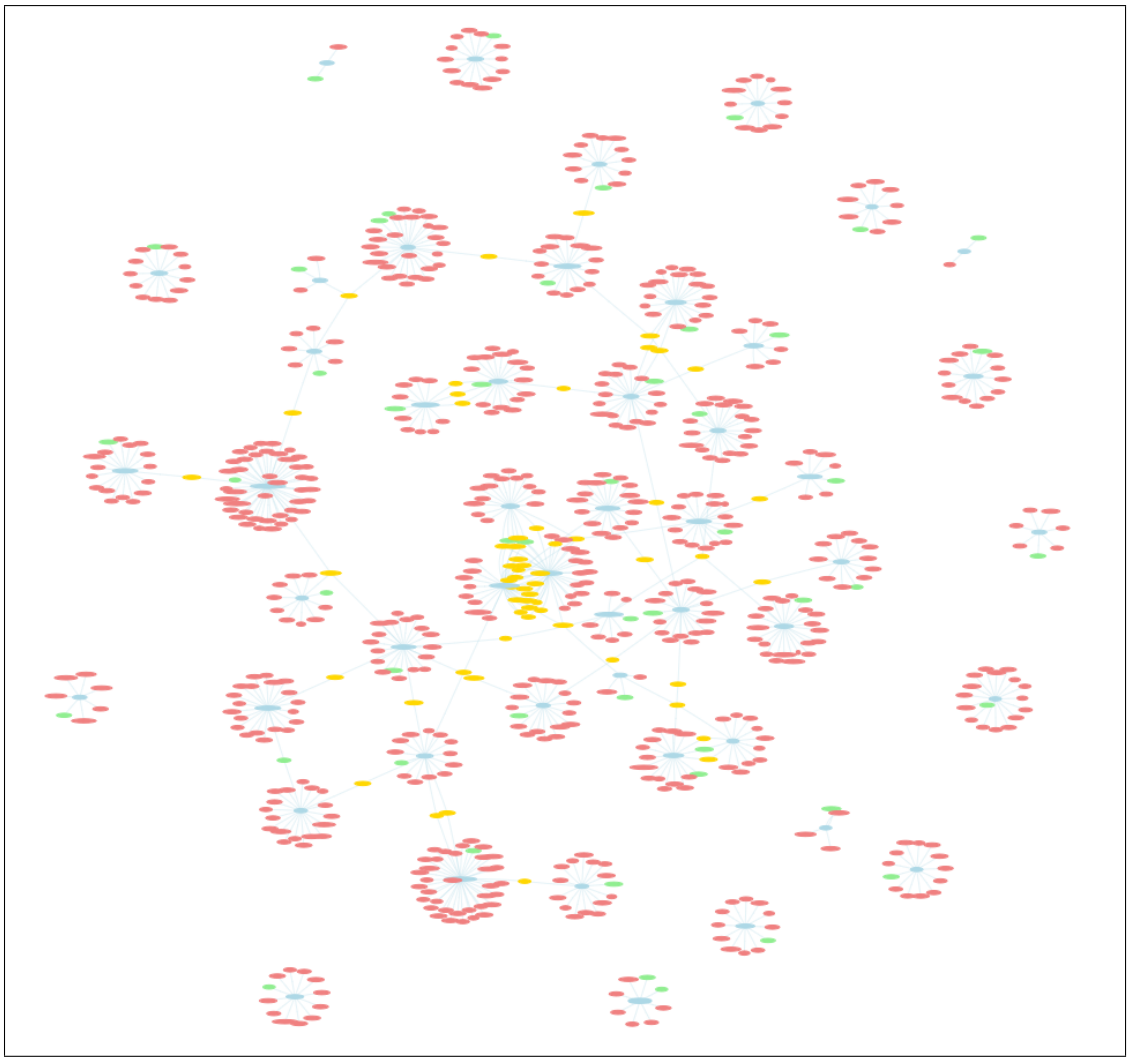

Figura 4.3: Grafo contenente i nodi dei film, dei registi e degli attori

Inoltre, grazie alla colorazione differente dei nodi, si può osservare che ci sono vari attori che hanno partecipato a più film. La concentrazione maggiore si trova al centro del grafo, in corrispondenza della trilogia di The Matrix: essendo tre film appartenenti allo stesso universo narrativo, molti attori hanno ripreso i propri ruoli nei film successivi al primo. Un'altra saga che viene rappresentata è Pacific Rim. Tuttavia, il numero di attori presenti nel grafo non è elevato e quelli che hanno ripreso il loro ruolo nel secondo film risultano solamente quattro.
Di seguito viene mostrato uno zoom della parte del grafo che mostra la trilogia di The Matrix:

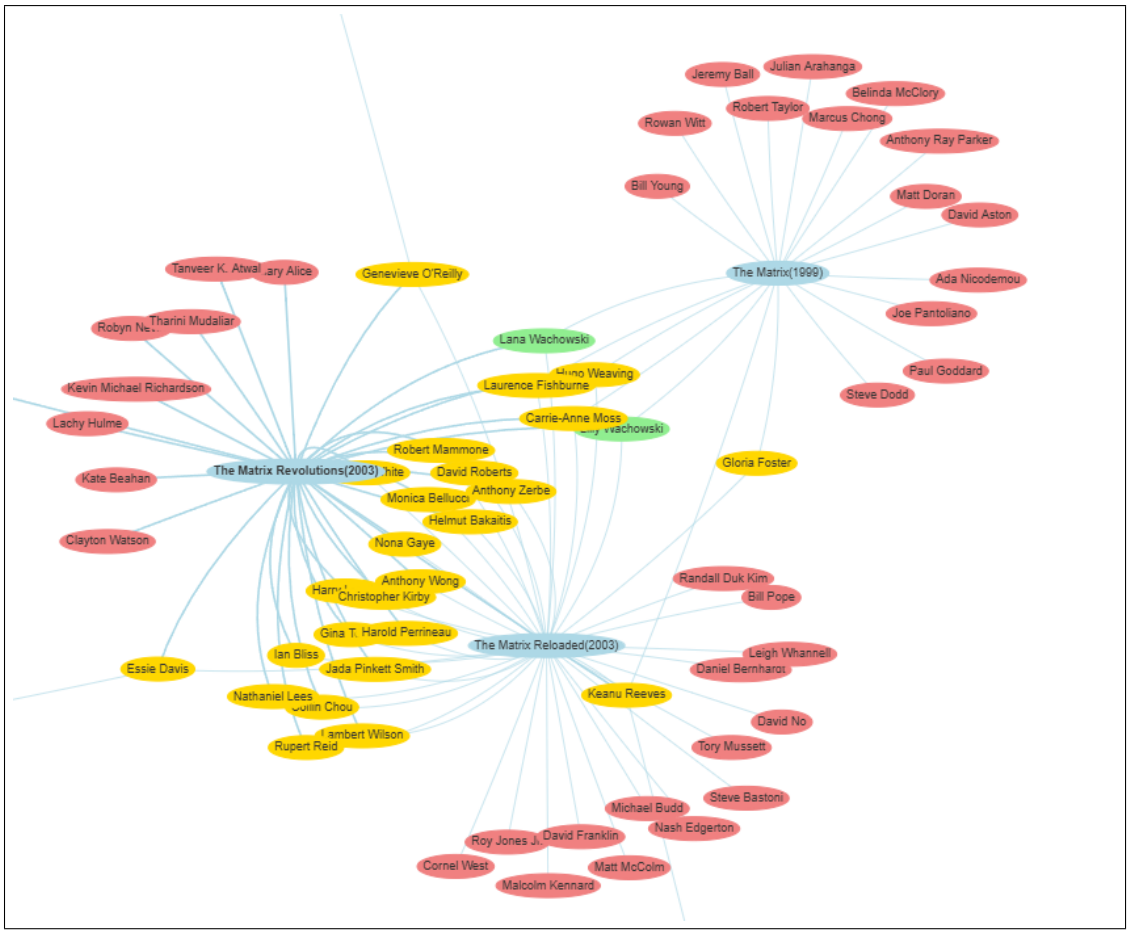

Figura 4.4: Grafo contenente i nodi dei film, dei registi e degli attori della trilogia di The Matrix

4.1.2 Grafici

I primi tre grafici generati mostrano i film raggruppati in base al loro luogo di ambientazione.
Il primo grafico raggruppa i film per pianeta. Dalla legenda si evince che le informazioni sono state estrapolate da 49 film, ma solo 39 sono rappresentati nel grafico. Tra questi, 22 sono ambientati in un solo luogo, mentre 17 si svolgono in più ambientazioni. Osservando il grafico, si nota che ben 32 film sono ambientati sulla Terra, 2 sono ambientati su Marte e 25 sono ambientati in un Fantasy World. In questa categoria rientrano tutti quei luoghi che non hanno trovato una corrispondenza all'interno dei dataset utilizzati per il mapping.

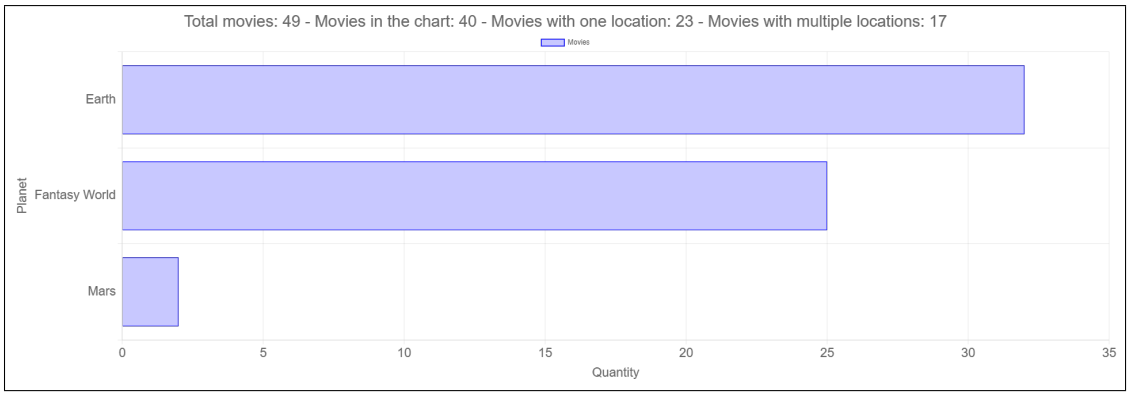

Figura 4.5: Grafico che raggruppa i film per pianeta

Il secondo grafico raggruppa i film per continente. Anche in questo caso, i film analizzati sono 49, ma quelli presenti nel grafico sono 28: 15 ambientati in un solo continente e 12 in più continenti. Si osserva che l'America è il continente più rappresentato, con 22 film, mentre Europa e Oceania sono i meno presenti, con 6 film ciascuno. Sono inoltre rappresentati 13 film ambientati in Asia, mentre l'Africa e l'Antartide non compaiono nel dataset.

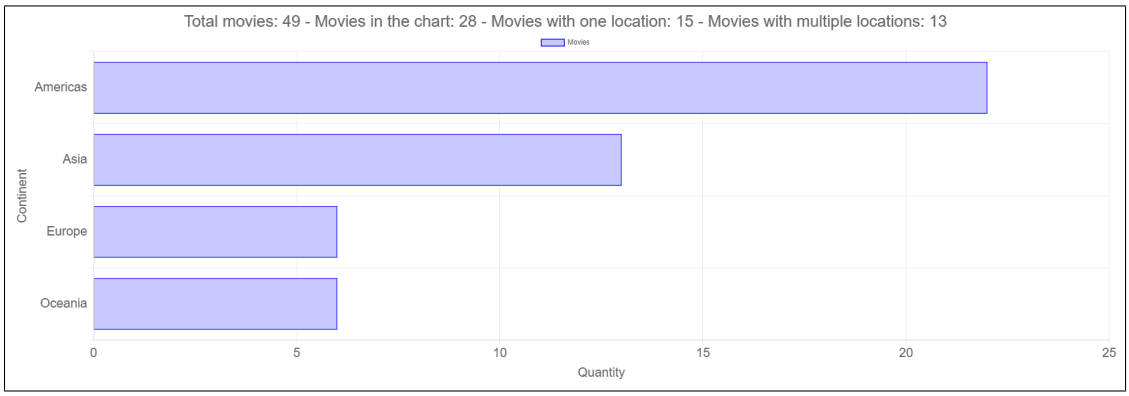

Figura 4.6: Grafico che raggruppa i film per continente

Il terzo grafico raggruppa i film per stato. Anche in questo caso, i film analizzati sono 49, di cui 28 sono presenti nel grafico. Di questi, 14 sono ambientati in un solo stato, mentre gli altri 14 si svolgono in più stati. Nel grafico sono presenti un totale di 29 stati, quello più rappresentato sono gli Stati Uniti con ben 18 film, seguito dal Messico con 7 film e l'Australia con 6 film.

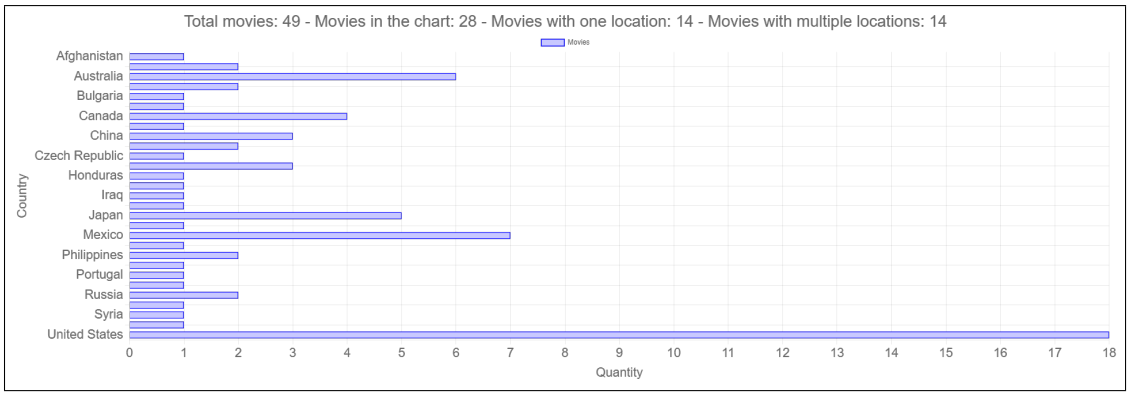

Figura 4.7: Grafico che raggruppa i film per stato

Segue un grafico che rappresenta la timeline delle ambientazioni temporali dei film. Osservando il grafico si può notare che la maggior parte dei film viene ambientato in un intervallo di tempo che va dal 1900 al 2100. Fanno eccezione alcuni film come Assassin's Creed(2016) che ha un'ambientazione temporale che va dal 1492 al 2016, The Matrix(1999) che va dal 1999 al 2199 e Alita: Battle Angel(2019) che è ambientato nel 2563.

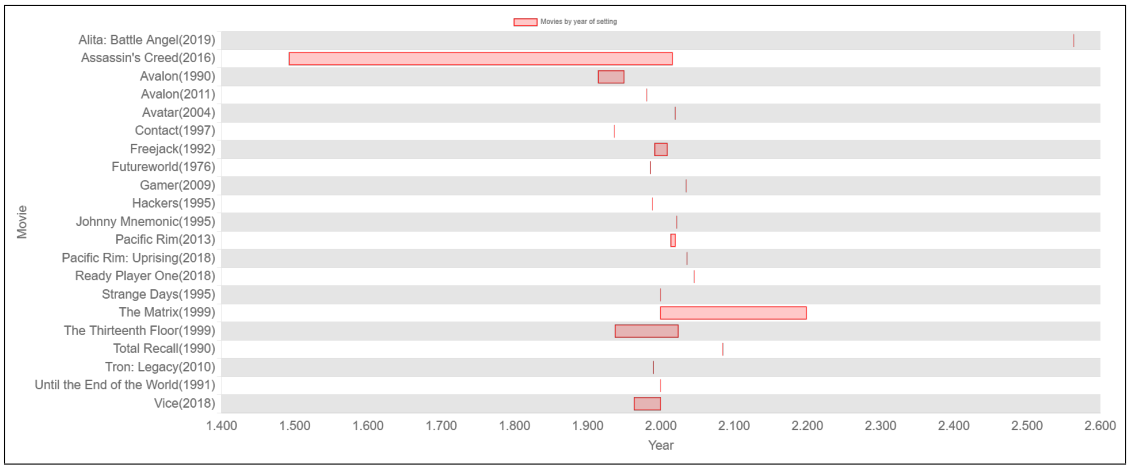

Figura 4.8: Timeline che mostra l'ambientazione dei film

Questo grafico, invece, mostra la stessa timeline presente nel grafico precedente affiancata da una timeline delle date di uscita. Questo è stato fatto per vedere quanto discostasse l'ambientazione temporale dalla data di uscita del film. Osservando il grafico, si può notare che l'ambientazione temporale non differisce di troppo rispetto alla data di uscita del film, l'unica eccezione è sempre Alita: Battle Angel(2019).

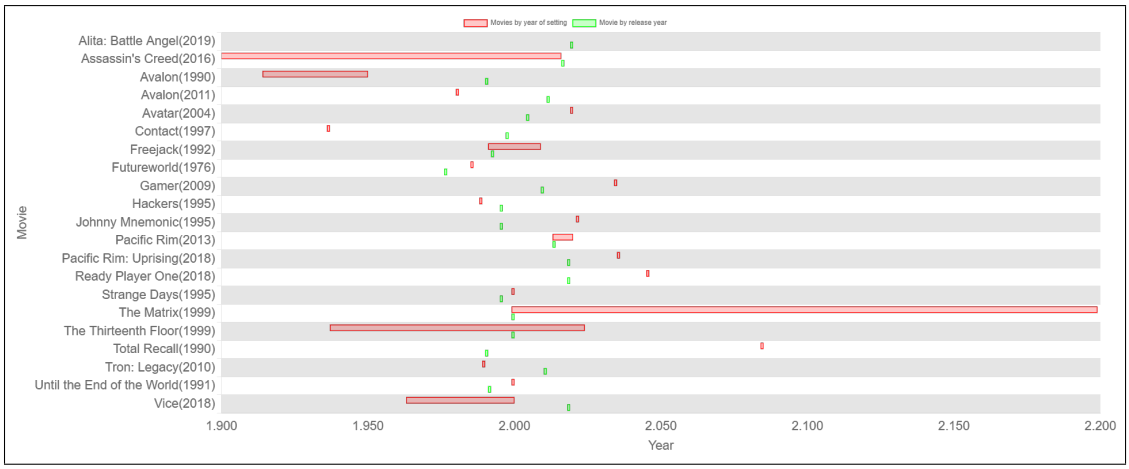

Figura 4.9: Timeline che mostra l'ambientazione dei film in relazione alla data di pubblicazione

Un altro grafico analizza le lunghezze(in caratteri) delle trame dei film. Questo grafico è stato creato per valutare la quantità di testo che sarebbe stato disponibile allo script Python per l'analisi. Osservandolo, si può notare che la media è 2943 caratteri, con un valore massimo di 5415 caratteri e un valore minimo di 78 caratteri. Quindi le trame sono abbastanza lunghe e i valori dei caratteri abbastanza omogenei, con solamente qualche eccezione.

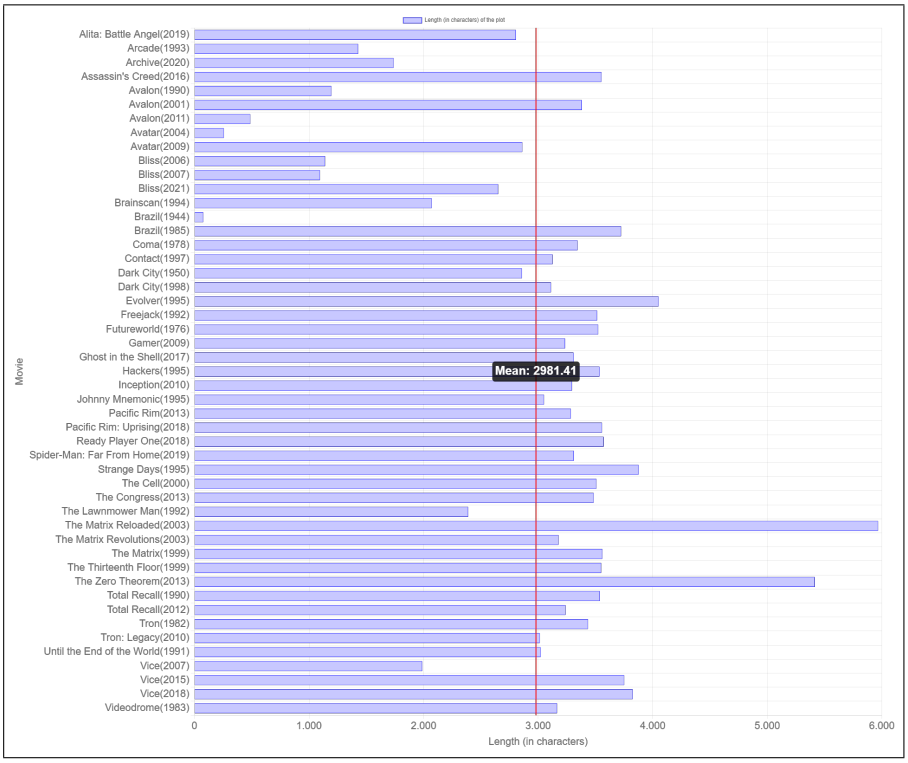

Figura 4.10: Grafico che mostra le varie lunghezze(in caratteri) delle trame

Infine, viene mostrato un grafico che mostra quante date e quanti luoghi sono stati estrapolati dallo script Python. Il valore maggiore di date raccolte è stato 8 per il film Vice(2008), mentre il numero maggiore di luoghi raccolti è stato 15 per il film Until the End of the World(1991). Su 49 film, 3 film non hanno fornito alcuna informazione, 5 film hanno restituito solo date e 3 film hanno restituito solamente luoghi.

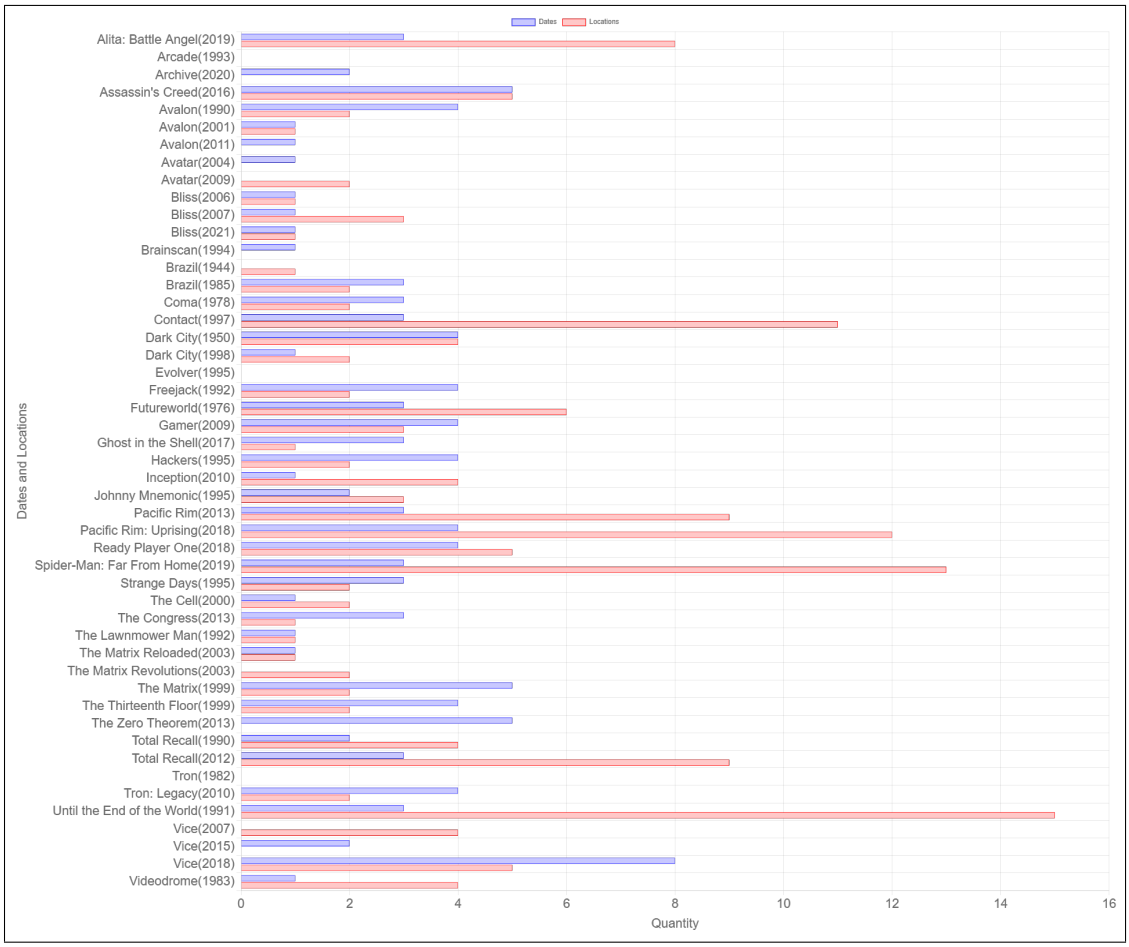

Figura 4.11: Grafico che mostra quanti luoghi e quante date sono state estrapolate

4.2 Dataset sui film di Cristopher Nolan

Come secondo dataset, invece, sono stati utilizzati i film di Cristopher Nolan:

- Following
- Memento
- Insomnia
- Batman Begins
- The Prestige
- The Dark Knight
- Inception
- The Dark Knight Rises
- Interstellar
- Dunkirk
- Tenet
- Oppenheimer

4.2.1 Rete di relazioni tra film, registi e attori

Anche in questo caso, il grafo generato quando l'utente clicca sul pulsante `Create Graph` contiene solamente i nodi dei film, senza nessun collegamento tra di loro.

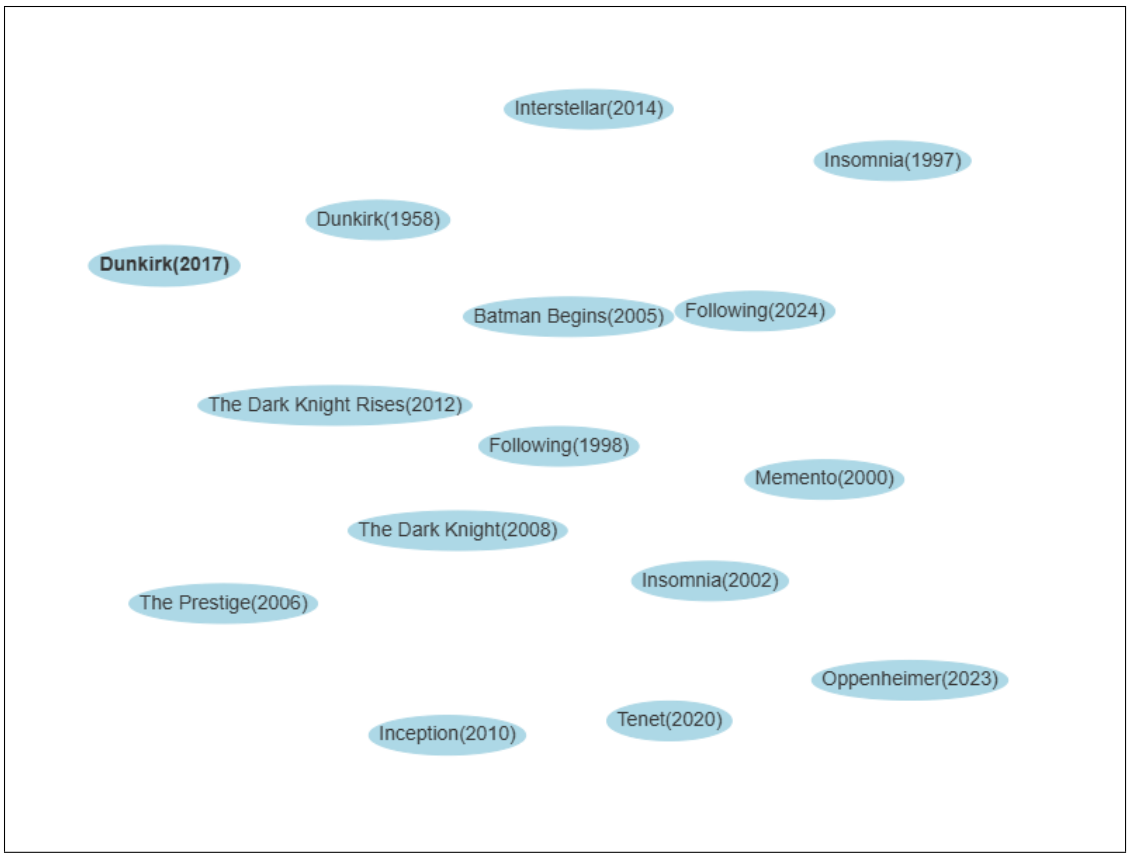

Figura 4.12: Grafo contenente i nodi dei film

Aggiungendo i nodi dei registi, l'unico elemento evidente è che tutti i film appartengono a Christopher Nolan. Tuttavia, dato il numero ridotto di film e l'unicità del regista, risulta più semplice individuare eventuali errori nel dataset. In particolare, si notano film che non avrebbero dovuto essere inclusi, ma che sono stati salvati erroneamente perché condividevano il titolo con quelli presenti nel file di input.

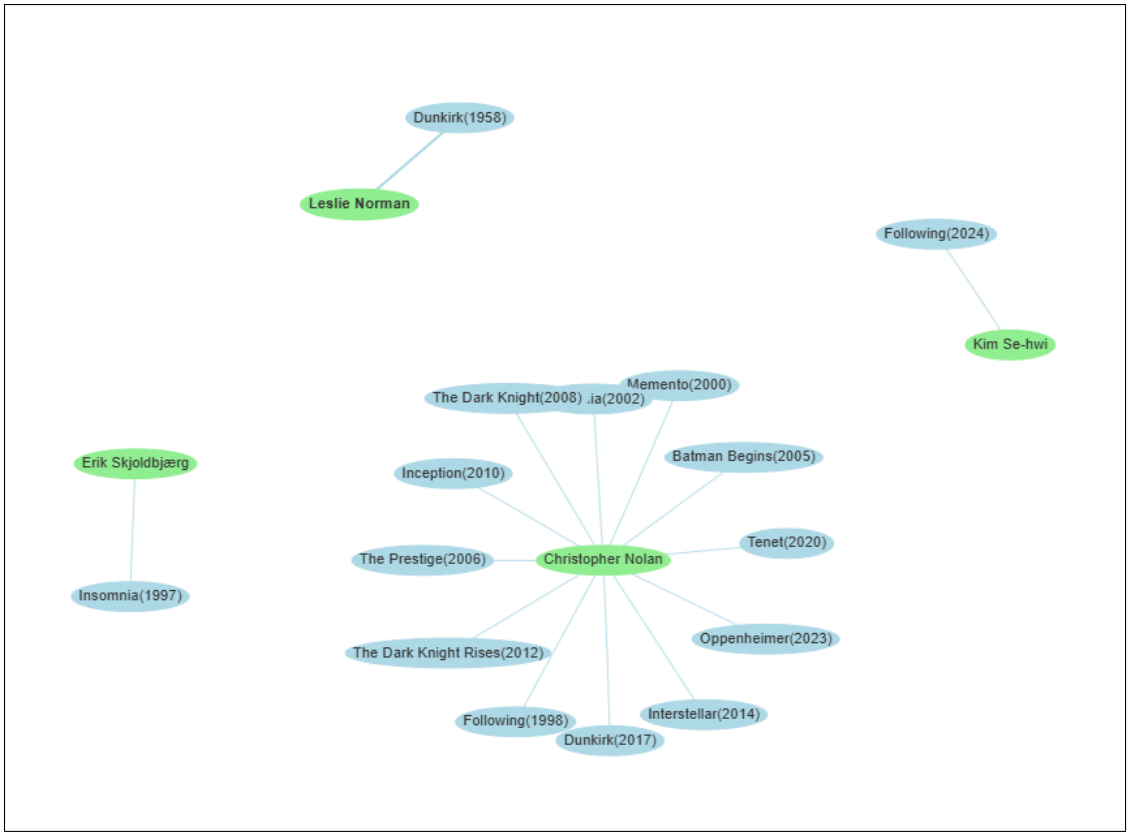

Figura 4.13: Grafo contenente i nodi dei film e dei registi

Aggiungendo i nodi degli attori, si nota che alcuni di loro hanno partecipato a più film di Christopher Nolan. Un aspetto interessante è che l'unica trilogia diretta dal regista è quella di Batman, nella quale alcuni attori hanno ripreso il proprio ruolo nei film successivi al primo. Tuttavia, ci sono anche altri attori ricorrenti nella sua filmografia, come Cillian Murphy, presente in cinque film, e Gary Oldman, che ha preso parte a quattro pellicole dirette dal regista.

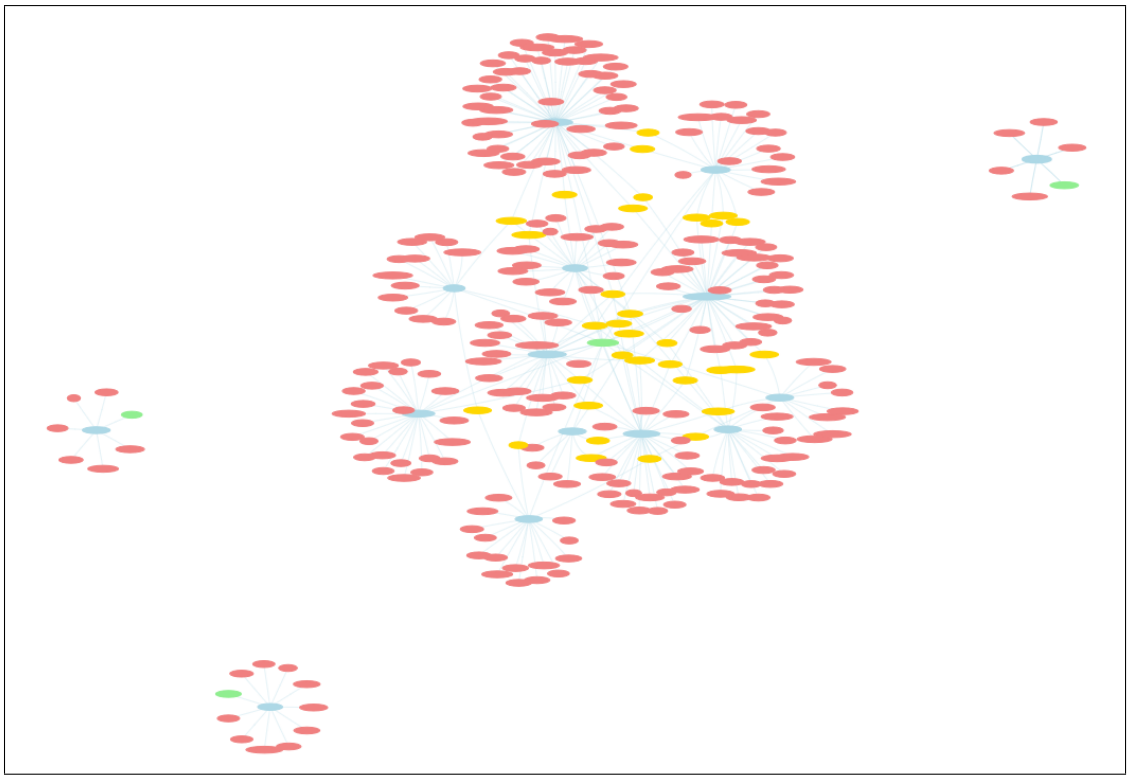

Figura 4.14: Grafo contenente i nodi dei film, dei registi e degli attori

4.2.2 Grafici

Dalla legenda del primo grafico, che raggruppa i film per pianeta, si evince che il totale dei film di cui sono state estrapolate le informazioni sono 15, invece, nel grafico sono presenti 13 film, dei quali, 6 ambientati in un solo luogo mentre 7 ambientati in entrambi. Dal grafico si evince che tutti i film sono ambientati o sulla Terra oppure in un luogo fantastico.

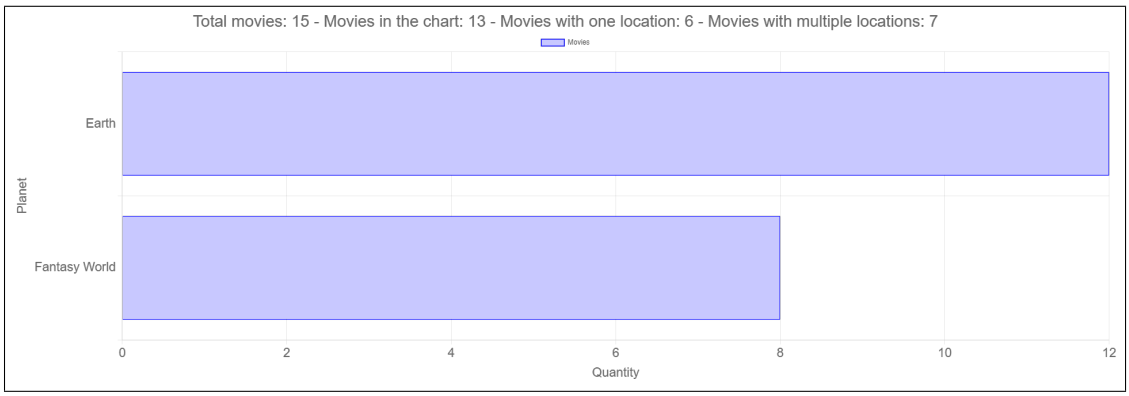

Figura 4.15: Grafico che raggruppa i film per pianeta

Dalla legenda del secondo grafico, che raggruppa i film per continente, si evince che il totale dei film analizzati sono sempre 15, quelli presi in considerazione dal grafico sono 11, dei quali, 3 ambientati in un solo luogo mentre 8 ambientati in più continenti. Dal grafico si può notare come siano presenti tutti i continenti tranne l'Antartide.

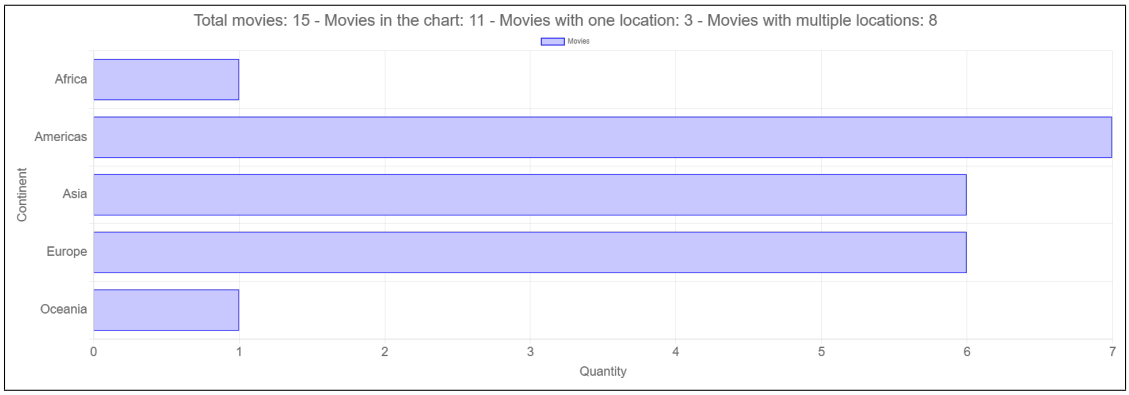

Figura 4.16: Grafico che raggruppa i film per continente

Dalla legenda del secondo grafico, che raggruppa i film per stato, si evince che il totale dei film analizzati sono sempre 15, quelli presi in considerazione dal grafico sono 11, dei quali, un solo film è ambientato in un singolo luogo mentre 10 sono ambientati in più stati. In totale sono presenti 21 stati e anche in questo caso quello rappresentato di più sono gli Stati Uniti con 6 film, seguiti dal Canada con 3 film e Italia e Norvegia con 2 film.

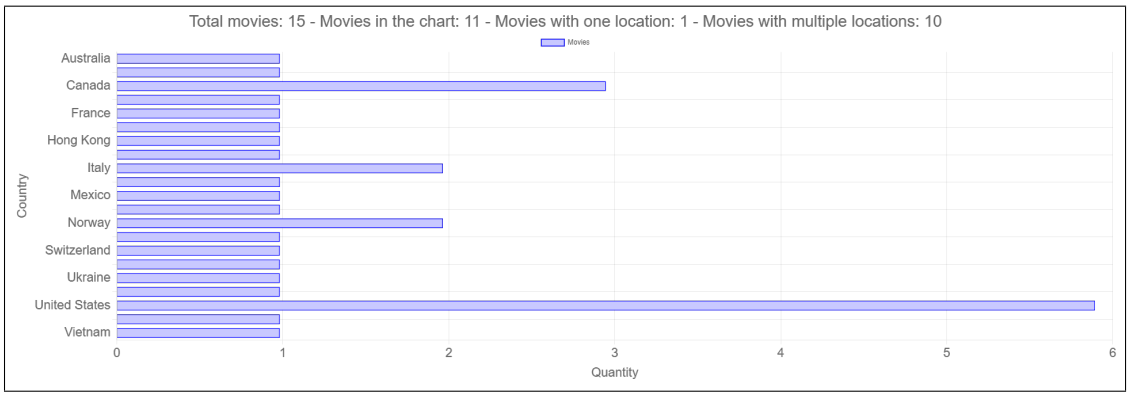

Figura 4.17: Grafico che raggruppa i film per stato

Segue un grafico che rappresenta la timeline delle ambientazioni temporali dei film. Osservando il grafico si può notare che i film presenti sono solamente 4. Questo è dovuto dal numero limitato di film nel dataset e il fatto che non sono presenti date numeriche all'interno della trama. Però, dai pochi dati presenti, e senza

considerare il primo film che non dovrebbe essere stato salvato, si può evincere che i 3 film restanti sono ambientati nel '900.

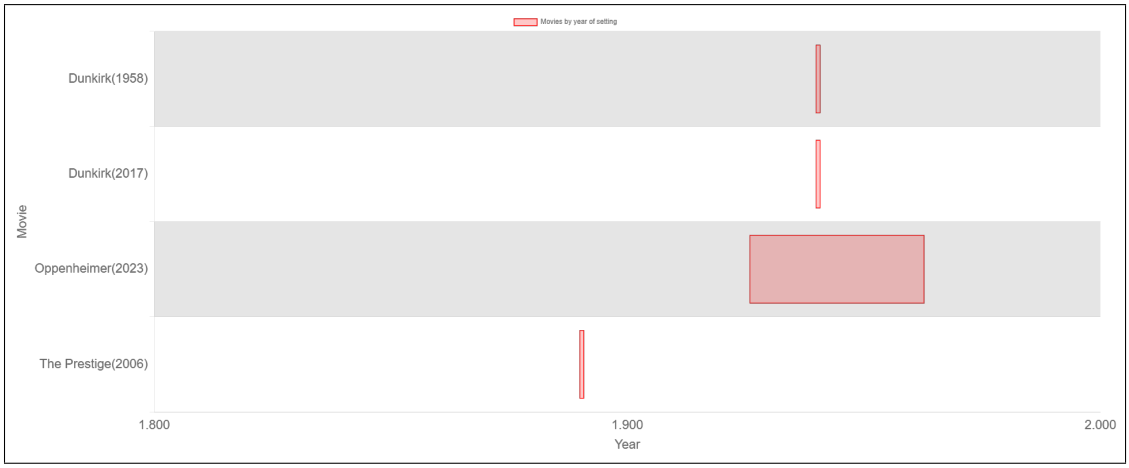

Figura 4.18: Timeline che mostra l'ambientazione dei film

Questo grafico, invece, mostra la stessa timeline presente nel grafico precedente affiancata da una timeline delle date di uscita. Questo è stato fatto per vedere quanto discostasse l'ambientazione temporale dalla data di uscita del film. Osservando il grafico, si può notare, anche qui, che l'ambientazione temporale non differisce significativamente dall'epoca di ambientazione, dato che, tranne per il primo film, che però non dovrebbe far parte del grafico, sono stati pubblicati tra il 2006 e il 2023.

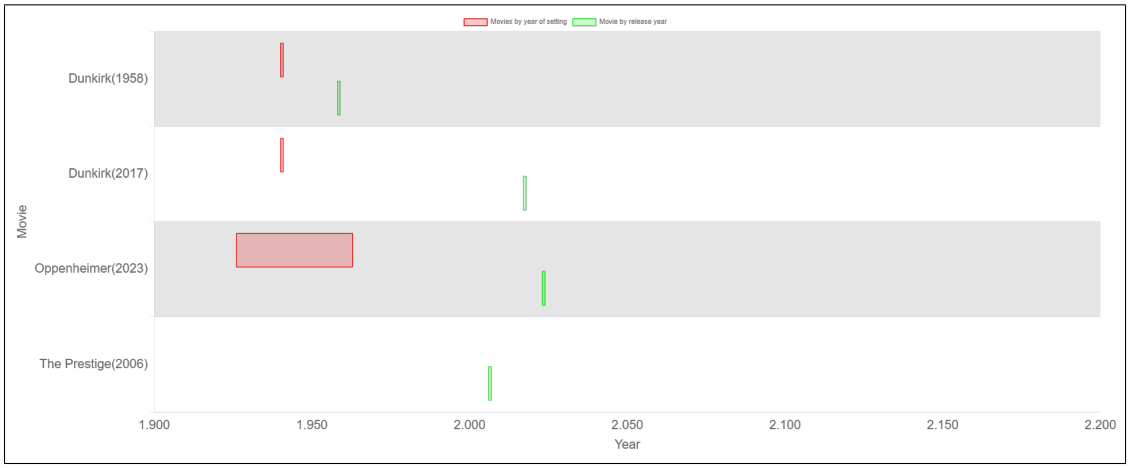

Figura 4.19: Timeline che mostra l'ambientazione dei film in relazione alla data di pubblicazione

Un altro grafico analizza le lunghezze(in caratteri) delle trame dei film. Osservandolo, si può notare che la media è 3057.60 caratteri, con un valore

massimo di 3645 caratteri e un valore minimo di 587 caratteri. Quindi, anche in questo caso, i valori sono abbastanza omogenei, con solamente un'unica eccezione che è il film con il numero minimo di caratteri, per il resto hanno tutti valori sopra i 2500 caratteri.

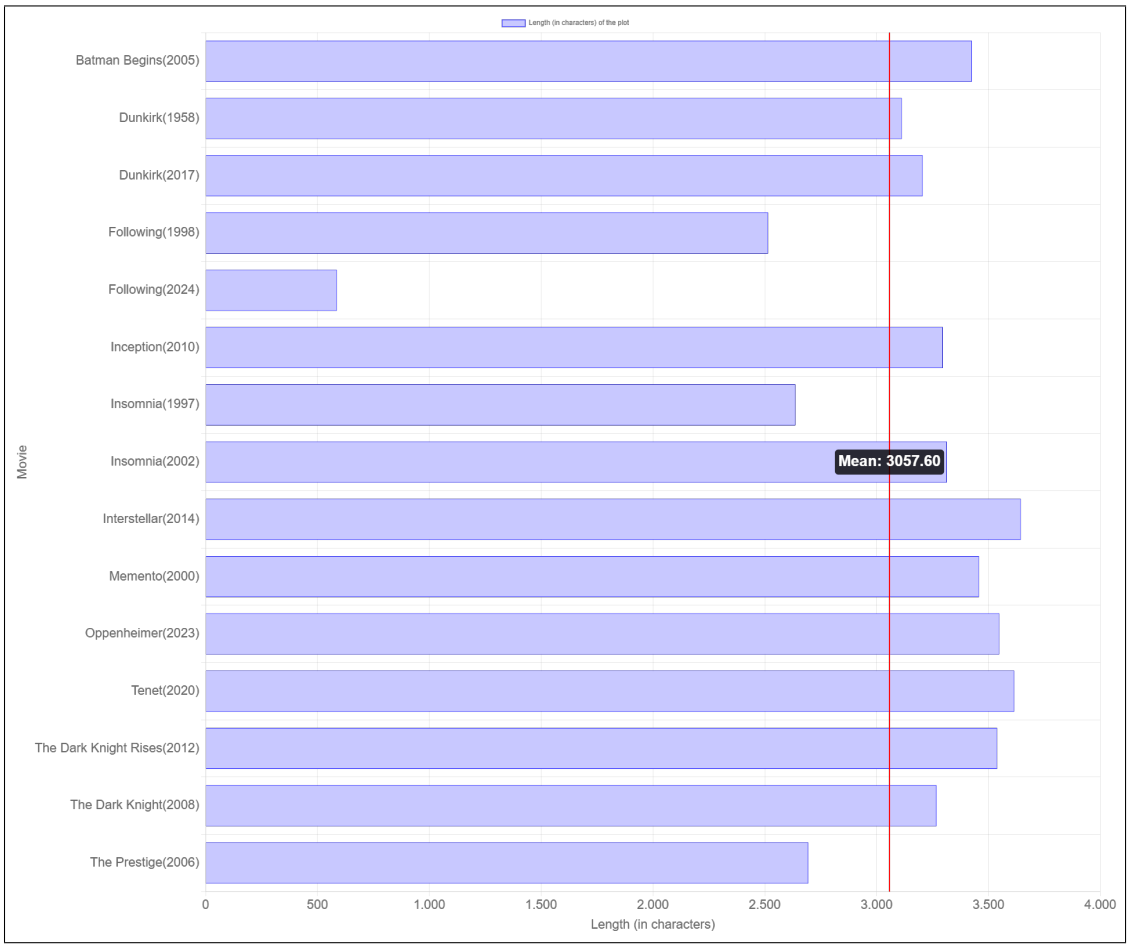

Figura 4.20: Grafico che mostra le varie lunghezze(in caratteri) delle trame

Infine, viene mostrato un grafico che mostra quante date e quanti luoghi sono stati estrapolati dallo script Python. Il valore maggiore di date raccolte è stato 9 per il film Oppenheimer(2023), mentre il numero maggiore di luoghi raccolti è stato sempre 8 per due film: Oppenheimer(2023) e Tenet(2020). Su 15 film, tutti hanno fornito informazioni, con 2 film hanno restituito solo date e 2 film hanno restituito solamente luoghi.

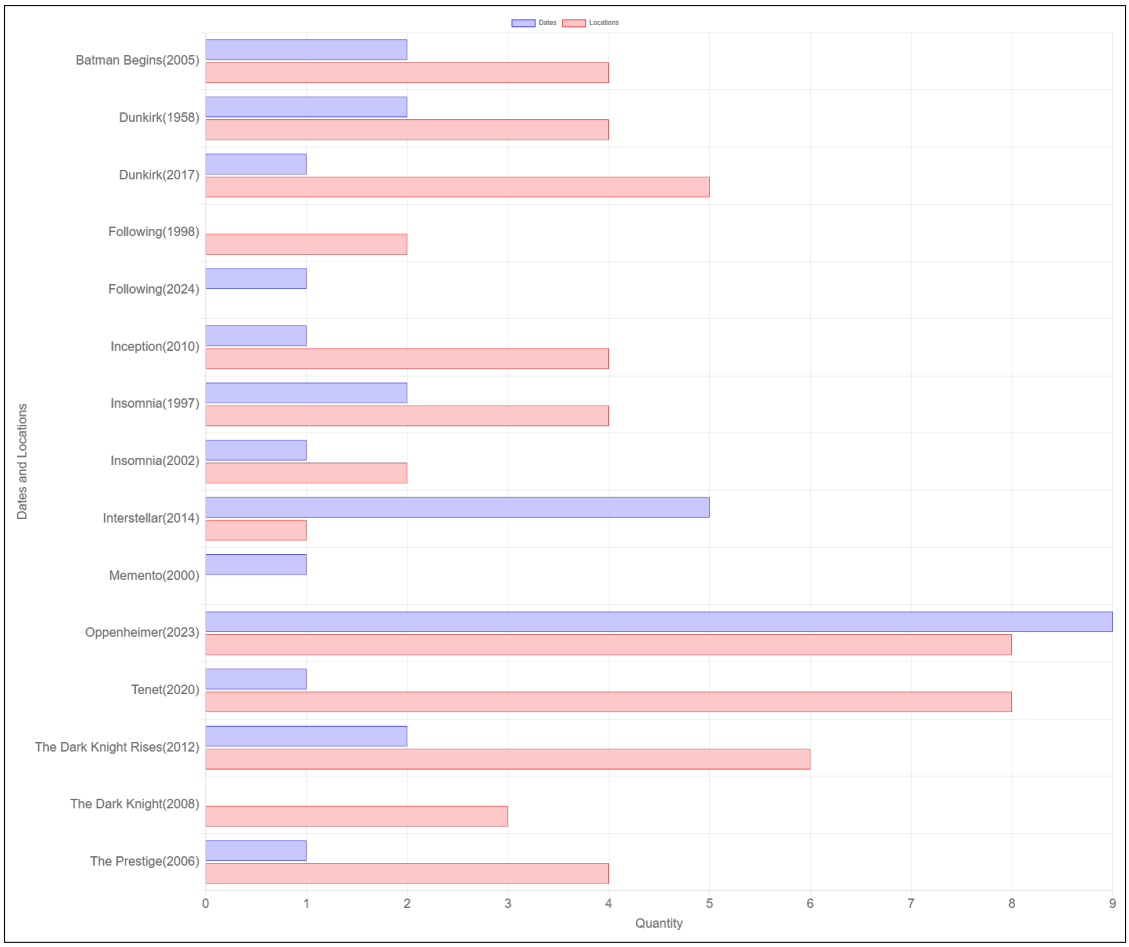

Figura 4.21: Grafico che mostra quanti luoghi e quante date sono state estrapolate

4.3 Considerazioni finali

I grafi hanno evidenziato come, i film del primo dataset, avessero tutti registi diversi, tranne per la trilogia di The Matrix. Per quanto riguarda gli attori, entrambi i dataset hanno mostrato la presenza di attori che hanno partecipato a più pellicole. La concentrazione maggiore di attori, per entrambi i dataset, è stata riscontrata per film dello stesso universo narrativo, dove gli interpreti hanno ripreso i propri ruoli. In particolare, per il secondo dataset, sono emersi diversi attori ricorrenti nella filmografia di Cristopher Nolan.
Passando all'analisi dei grafici, per entrambi i dataset il pianeta più rappresentato è risultato essere la Terra, il continente più frequente è stata l'America e, a livello di Stati sono prevalsi gli Stati Uniti. Quindi si ha una predominanza dell'America, cosa abbastanza prevedibile essendo la maggior parte dei film di produzione statunitense. Inoltre, si è osservato che le trame dei film tendono a essere piuttosto lunghe e con una lunghezza relativamente omogenea, salvo

alcune eccezioni. Questo conferma che lo script Python ha lavorato con una buona quantità di dati ed è riuscito ad estrarre un buon numero di informazioni tra date e luoghi. Infine, si è osservato che le ambientazioni temporali dei film si concentrano prevalentemente tra il 1900 e il 2100, con qualche eccezione. Si è anche notato che le date di pubblicazione, salvo rari casi, non si discostano molto dall'epoca in cui sono ambientati i film.

Capitolo 5

Conclusioni

Lo sviluppo di C.I.N.E. è stato molto interessante sia per l'argomento trattato, che per i risultati che sono stati trovati. Ha permesso di esplorare tramite visualizzazioni interattive le informazioni dei film che sono state raccolte. Anche se con qualche limite, offre degli strumenti validi per l'analisi di elementi chiave dei film in modo approfondito.
Di seguito verranno discussi i punti di forza e di debolezza di C.I.N.E. e verranno analizzati possibili sviluppi futuri per l'applicazione.
I punti di forza dell'applicazione sono:

- La presenza di grafici mirati all'analisi di aspetti chiave dei film, come l'ambientazione e la collocazione temporale.
- L'integrazione di un grafo che permette di analizzare le relazioni tra film, attori e registi.
- La possibilità di analizzare la trama per raccogliere informazioni utili con cui creare le visualizzazioni.

I punti di debolezza dell'applicazione sono:

- L'impossibilità di decidere esattamente quali film estrarre, il che porta a ottenere tutti i film con lo stesso titolo di quelli presenti nel file di input.
- La disponibilità dei dati solo in lingua inglese.
- Il mapping delle ambientazioni non è preciso, alcuni luoghi che non sono presenti nei dataset, non vengono mappati correttamente.

Possibili miglioramenti futuri dell'applicazione includono:

- Implementare un metodo per estrarre informazioni solo sui film presenti nel file di input, evitando risultati non pertinenti.

- Aggiungere il supporto per più lingue.
- Aggiungere ulteriori grafici per analizzare aspetti che non sono ancora stati considerati.
- Migliorare l'efficienza e la precisione dello script Python che analizza le date i luoghi.
- Creare uno script in grado di determinare l'ambientazione del film, ad esempio se è ambientato in prigione, su una spiaggia o in un contesto urbano.
- Rendere integrabile anche la creazione del grafo all'interno di sistemi esterni.
- Migliorare la precisione del mapping delle ambientazioni dei film.

Bibliografia

[1] Amazon. Imdb. https://www.imdb.com/it/?ref_=nv_home.

[2] Jorge Cardoso. Vr movies database. https://jorgecardoso.notion.site/VR-Movies-Database-5226dc5aeb354021b4b7d030a6f4fc11.

[3] Jermain Kaminski, Michael Schober, Oleksandr Zastupailo, and Cesar Hidalgo. Moviegalaxies: Social networks in movies, December 2012. http://moviegalaxies.com/.

[4] Volodymyr Miz. Moviegalaxies: Data analysis and genre classification using social network topologies of movies, 2018. https://shorturl.at/ZDhfg.

[5] Benjamin Nemceff. Data+movies. https://public.tableau.com/app/profile/benjamin.nemceff/viz/DataMovies_17010360628300/MovieAnalysis.

[6] Tableau Public. Tableau public powered by imdb. https://www.tableau.com/gallery/data-plus-movies.

[7] DKS. Rukshan. Timeline of ridley scott's top movies. https://public.tableau.com/app/profile/rukshan/viz/CompleteTimelineofRidleyScottMovies/TimelineofRidleyScottsTopMovies.

[8] Simone Tosi. simone_tosi_tesi, 2025. https://github.com/simotos/tesi_simone_tosi.

[9] Wikipedia. Imdb. https://it.wikipedia.org/wiki/IMDb.

Ringraziamenti

Ringrazio i miei genitori, Daniele e Albertina, e mia sorella Gaia, per avermi sempre supportato e aiutato durante il mio percorso di studi.
Ringrazio i miei nonni, Franco e Giusy, Adriana e Giordano che con il loro affetto hanno reso questi anni più piacevoli e un po' più spensierati. Un ringraziamento speciale va a mio nonno Giordano che, anche se non potrà vedermi laureato, è sempre stato entusiasta del mio percorso. Questa tesi è anche per lui.
Infine, ringrazio il Professore Di Iorio per la sua disponibilità e per il prezioso aiuto nella stesura di questa tesi.