



UNIVERSITÀ "SAPIENZA" DI ROMA
FACOLTÀ DI INFORMATICA

BloodlineSeeker

Wikipedia web scraper per gli imperatori romani

Authors

Alessio Bandiera
Matteo Benvenuti
Simone Bianco
Andrea Ladogana

15 giugno 2022



Indice

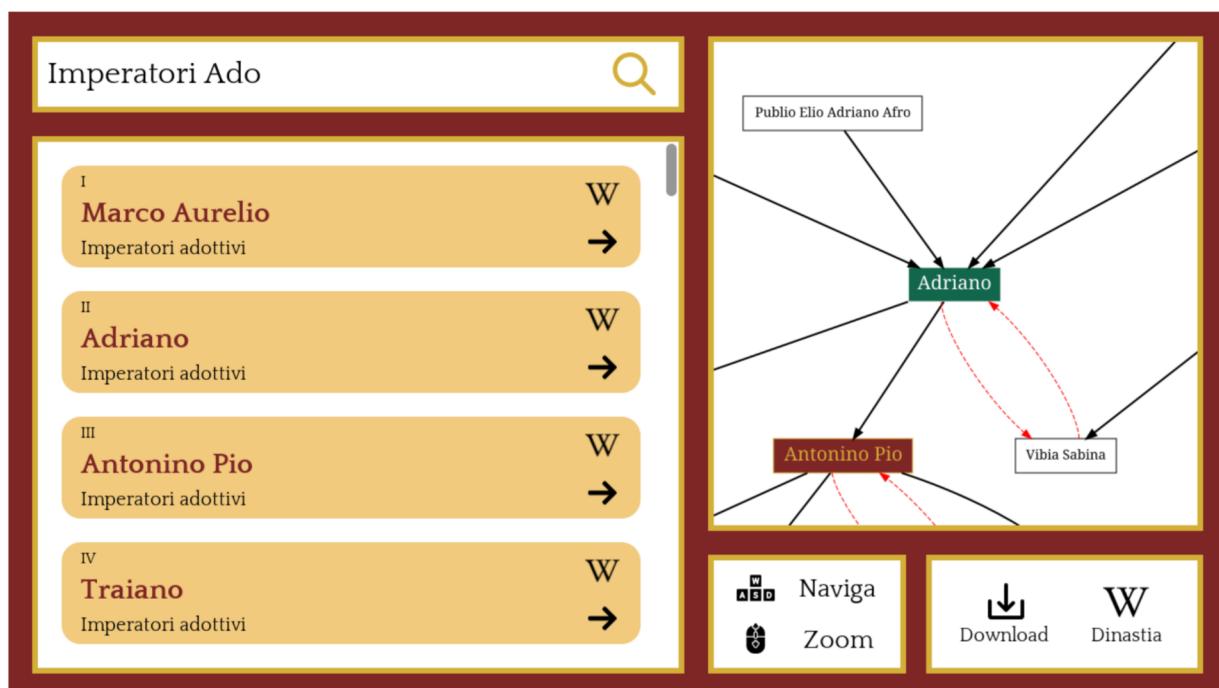
1	Introduzione	1
2	Funzionalità del programma	2
2.1	Download dei dati	2
2.1.1	Scraping delle dinastie	2
2.1.2	Serializzazione delle dinastie	3
2.2	Interfaccia grafica	5
2.2.1	Aspetto dell'albero genealogico	5
2.2.2	Navigazione del grafico	5
2.2.3	Salvataggio delle immagini delle dinastie complete	6
2.2.4	Antialiasing	6
2.3	Ricerca di persone e dinastie	7
2.3.1	Algoritmo di ricerca	7
2.3.2	Suggerimenti di ricerca	7
2.3.3	Visualizzazione dei risultati	7
2.4	Funzioni generali	7
2.4.1	Interpretazione degli argomenti	7
2.4.2	Multithreading	8
2.4.3	File JAR	8
3	Struttura del programma e librerie utilizzate	9
3.1	Progettazione delle classi	9
3.1.1	Descrizione e gerarchia delle classi	9
3.2	Librerie utilizzate	18
4	Manuale della grafica	19
4.1	Finestra di avvio	19
4.1.1	Dinastie non trovate	19
4.1.2	Download delle dinastie	20
4.1.3	Dinastie trovate	21
4.2	Finestra di ricerca	22
4.2.1	Struttura generale	22
4.2.2	Suggerimento di ricerca	23
4.2.3	Risultati di ricerca e visualizzazione del grafico	24
5	Ripartizione dei compiti	25
5.1	Legenda	25
5.2	Ripartizione	25

Capitolo 1

Introduzione

La seguente relazione tratta delle specifiche di progettazione e implementazione del **programma "BloodlineSeeker"**, il quale ha l'obiettivo di facilitare la visualizzazione dei legami di parentela che intercorrono tra gli imperatori romani e i membri delle loro dinastie.

A tale scopo, l'applicazione utilizza un **web scraper** per ottenere da [Wikipedia](#) i dati necessari a generare gli alberi genealogici dei membri, fornendo inoltre una **interfaccia grafica** per un utilizzo facile e intuitivo, permettendo all'utente di navigare il grafico direttamente nella finestra.



Da un punto di vista tecnico, il progetto ha richiesto l'utilizzo di [Selenium](#), un framework open-source impiegato per l'**automazione delle operazioni sui browser**, e di [Graphviz](#), un programma open-source multi-piattaforma per **generare grafi**. Infine sono stati utilizzati strumenti per facilitare il **lavoro in gruppo** e aumentare la **produttività** generale del team, quali: [git](#), [GitHub](#) e la funzione "Live Share" di [Visual Studio Code](#), che ha permesso la collaborazione per la realizzazione delle funzionalità più complesse.

Tutti i file relativi all'implementazione del progetto possono essere trovati nella [seguinte repository GitHub](#).

Capitolo 2

Funzionalità del programma

In una fase antecedente alla realizzazione del programma, è stato necessario effettuare delle **precise scelte implementative**, al fine di delineare gli **obiettivi** del progetto e le **funzionalità** dell'applicazione. Tale processo preliminare ha permesso di distribuire equamente la mole di lavoro in fase di programmazione, e di evitare errori in fase di implementazione.

2.1 Download dei dati

Per funzionare, il programma necessita delle informazioni delle dinastie, ottenibili tramite download utilizzando **Selenium** o **richieste HTTP** o, alternativamente, possono essere **caricati da locale**; questo è possibile in quanto i dati, dopo essere stati scaricati, vengono salvati in locale tramite un processo di **serializzazione**, supportando dunque l'utilizzo offline del programma.

2.1.1 Scraping delle dinastie

Per l'acquisizione dei dati da Wikipedia, lo scraper si occupa di prelevare il **codice HTML delle pagine richieste**. Tale operazione può essere svolta in due modalità:

- tramite il **framework Selenium**, il quale avvierà una sessione in uno dei tre **browser** supportati (Mozilla Firefox, Google Chrome, Microsoft Edge) attraverso l'uso di **driver** già forniti all'interno del progetto
- tramite **richieste HTTP**, dalle quali verrà prelevato il body restituito nelle risposte ricevute

Poiché Selenium è progettato per svolgere compiti più complessi, come il testing estensivo di siti web e automatizzazione di task su di essi, il suo utilizzo come strumento di prelevamento del codice HTML risulta **estremamente inefficiente e laborioso**; per tale motivo, l'impiego delle richieste HTTP risulta essere **notevolmente più efficiente**.

Una volta ottenuto il codice della pagina richiesta tramite una delle due modalità, esso verrà **interpretato** dal programma, ricostruendo l'**individuo** associato alla pagina stessa, per poi andare ad **effettuare ricorsivamente lo scraping** sugli individui appartenenti al suo stesso nucleo familiare.

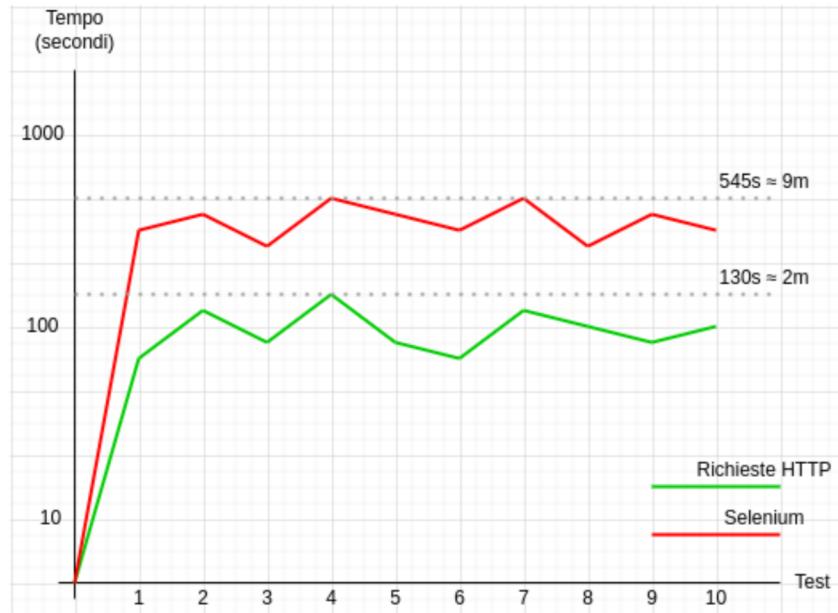


Figura 2.1: Performance di Selenium vs Richieste HTTP su 10 test

2.1.1.1 Console di download

Per permettere all’utente di **verificare lo stato del download**, è stata fornita un’interfaccia grafica che mostra in tempo reale le operazioni eseguite dallo scraper.

Per realizzare tale funzione è stato necessario eseguire l’aggiornamento della console su un **thread diverso** rispetto a quello di Swing, per mantenere **responsiva** la finestra durante il download, e avere il controllo della comunicazione tra questo thread e quello dedicato allo scraping, richiedendo l’uso di un oggetto ScaperStatusHolder contenente metodi di tipo **synchronized** al fine di poter sincronizzare i due thread.

2.1.2 Serializzazione delle dinastie

Oltre a poter scaricare le dinastie in tempo reale tramite scraping, il programma fornisce anche la possibilità di recuperare tutti i **dati scaricati nell’ultima sessione attiva**, i quali vengono salvati direttamente sulla memoria fisica dell’utente, in modo da poter essere letti all’avvio successivo.

2.1.2.1 Algoritmo di serializzazione

Per poter salvare i dati dell’ultima sessione di scraping effettuata, è stato necessario implementare una serializzazione su file degli oggetti **Dynasty** e **Member** caricati in memoria.

Le librerie di default di Java forniscono già una funzionalità di serializzazione degli oggetti, trasformandoli in una stringa univoca che può successivamente essere interpretata in modo da **riottenere l’oggetto** da cui essa è stata generata. Nel caso in cui sia presente un sotto-oggetto all’interno di un oggetto, l’algoritmo serializzerà anche esso. Tuttavia, il funzionamento della classe **Member** implica riferimenti vicendevoli tra più oggetti, impedendo il completamento della serializzazione.

Per risolvere tale problema, è stato scelto di implementare un **nuovo algoritmo di serializzazione**: ogni oggetto **Dynasty** viene trasformato in un JSON contenente al suo interno dei sotto-oggetti JSON, ognuno rappresentante un oggetto **Member** appartenente a quella dinastia. Poiché ogni oggetto **Member** possiede un **attributo ID** che lo identifica univocamente, l'algoritmo sostituisce tutti i puntatori ad un altri oggetti **Member** con gli ID corrispondenti, ovviando al problema dei puntatori vicendevoli.

Di seguito viene fornito un esempio di **oggetto Member serializzato in formato JSON**:

```
{  
  "265": {  
    "ID": "265",  
    "Name": "Gaio Giulio Cesare",  
    "Wiki-link": "https://it.wikipedia.org/wiki/Gaio_Giulio_Cesare",  
    "isEmperor": false,  
    "Parents": ["270", "271"],  
    "Spouses": ["266", "267", "268", "269"],  
    "Children": ["40"],  
  }  
}
```

Data la necessità di dover ricondurre ogni ID ad un oggetto **Member** già esistente, l'**algoritmo di de-serializzazione** è stato strutturato in due fasi:

1. si scorre una prima volta la dinastia serializzata, **creando tutti gli oggetti Member** tramite l'ID, il nome, il link e il campo **isEmperor**.
2. successivamente, si ripercorre la dinastia serializzata controllando i campi **Parents**, **Spouses** e **Children**, **ricostruendo le relazioni** di parentela tra gli oggetti **Member**.

2.1.2.2 Salvataggio e caricamento delle dinastie

Una volta serializzate tutte le dinastie caricate in memoria, ogni stringa ottenuta verrà **salvata all'interno di un file <nomeDinastia>.json**.

All'avvio del programma verrà controllata, nella stessa directory dell'applicazione, la presenza della **cartella data/jsons**; nel caso in cui venga trovata, il programma proverà a **caricare i file JSON** contenuti in essa, se presenti. Nel caso in cui il caricamento sia avvenuto con successo, la finestra permetterà all'utente di avviare il programma utilizzando **le dinastie salvate in locale**. Nell'eventualità in cui la cartella sia assente, o contenga file corrotti, la finestra di avvio del programma verrà modificata per permettere all'utente di far partire l'applicazione solo dopo aver effettuato il **download delle dinastie in locale**.

ATTENZIONE: *effettuare una nuova sessione di download delle dinastie tramite scraping sovrascriverà i file locali.*

2.2 Interfaccia grafica

Per la GUI del programma è stata usata la libreria Swing, servendosi della libreria [Flatlaf](#) per il Look and Feel dell'interfaccia.

2.2.1 Aspetto dell'albero genealogico

La libreria [Graphviz](#), unita ai dati provenienti dallo scraping e ad un algoritmo ricorsivo, genera l'**albero genealogico**, avente come **nodo centrale** l'individuo selezionato; data l'importanza e la centralità del grafico nel progetto, è stata dedicata particolare attenzione alla fruibilità da parte dell'utente di tale funzione.

Dunque, per migliorarne la consultazione, è stato **limitato il grado di parentela massimo mostrato**, e sono stati usati nei **nodi di colori verde e rosso** per indicare, rispettivamente, il **nodo centrale** del grafo e gli **imperatori** all'interno dell'albero genealogico. Anche i collegamenti tra membri, indicati con delle frecce, presentano una colorazione specifica; **nera per i legami parentali e rossa per i legami coniugali**.

2.2.2 Navigazione del grafico

Volendo permettere all'utente di visualizzare interamente l'albero genealogico della persona cercata all'interno dell'interfaccia grafica del programma, è stato necessario implementare un **sistema di navigazione delle immagini**, poiché altrimenti la visualizzazione sarebbe risultata difficoltosa; inoltre, dal momento che Swing non dispone di un'implementazione per lo spostamento dinamico delle immagini, è stata creata una classe preposta a tale funzione.

La navigazione è simulata facendo seguire ad ogni input dell'utente un corrispettivo **aggiornamento** della visualizzazione. Il grafico collegato ad un membro viene generato sotto forma di `BufferedImage` e salvato nella memoria del programma; prima della visualizzazione, attraverso un **algoritmo per il centramento**, vengono calcolate le coordinate del nodo dell'individuo selezionato; successivamente, a partire da queste, viene generata e mostrata una sotto-immagine all'interno della finestra. L'utente può effettuare uno **spostamento** sull'asse verticale o orizzontale, oppure modificare l'**area visibile** del grafico; tali operazioni vengono simulate aggiornando l'immagine:

- lo **spostamento** viene emulato prendendo una porzione del grafico, delle stesse dimensioni dell'immagine visualizzata, partendo però da coordinate differenti
- lo **zoom del grafico** viene emulato prendendo una sotto-immagine con lo stesso centro di quella attuale, ma avente dimensioni diverse; infine, viene **ridimensionata** per poter essere visualizzata correttamente

2.2.2.1 Centramento del grafico

Non disponendo di un metodo nativo di [Graphviz](#) che permetta di restituire le coordinate di un nodo all'interno di un grafico, per compiere tale operazione è stato utilizzato un algoritmo che applica dei **controlli sui pixel**, sfruttando la **differenza di colore** tra il nodo centrale e il resto dell'immagine:

- inizialmente, vengono controllati alcuni pixel dell'immagine, posti tra loro alla distanza massima tale da garantire di trovarne **almeno uno all'interno del rettangolo colorato**, contenente il nodo centrale
- partendo dal pixel trovato, scorrendo l'immagine, si ricavano le coordinate del **vertice in alto a sinistra** del rettangolo, e le sue **dimensioni**
- tali informazioni sono sufficienti a ricavare le coordinate del **punto medio del nodo**, usato successivamente per centrare l'immagine

2.2.3 Salvataggio delle immagini delle dinastie complete

Per fornire una visualizzazione completa degli alberi genealogici, il programma permette di **scaricare in locale** il grafico contenente **tutti i membri della dinastia scelta**; tale funzionalità presenta però alcune problematiche dal punto di vista delle performance, per via delle **grandi dimensioni** di alcune dinastie, rendendo inutilizzabile l'interfaccia grafica per tutta la durata del processo di salvataggio dell'immagine.

Per ovviare a questo problema, si è reso necessario eseguire l'operazione su **thread differenti** rispetto a quello utilizzato da Swing, permettendo inoltre di **salvare più immagini contemporaneamente**. Inoltre, effettuando l'operazione utilizzando la libreria [JDeli](#), il salvataggio verrà effettuato a **blocchi di byte**, alleggerendo il carico sull'heap del programma, il quale altrimenti, utilizzando la libreria standard di Java, potrebbe andare in **overflow**.

2.2.4 Antialiasing

L'**antialiasing** è una tecnica che permette di ridurre l'aliasing, ovvero l'**effetto "sgarnatura"** che si ottiene quando un segnale a bassa risoluzione viene mostrato ad una **risoluzione maggiore**. Per ottenere tale risultato su ogni piattaforma, sono state applicate diverse procedure:

- sono state modificate delle **proprietà di sistema**
- sono stati **creati dei componenti** che ereditano da quelli predefiniti di Swing, effettuando l'override il metodo `paintComponent`
- è stata sfruttata l'implementazione dell'**antialiasing di default** di [Flatlaf](#)

2.3 Ricerca di persone e dinastie

Il programma fornisce la possibilità di **ricercare** qualsiasi persona in ogni dinastia presente nei file JSON salvati in locale, consentendo inoltre di inserire come **chiave di ricerca** il nome di una dinastia, mostrandone ogni membro in ordine di importanza.

2.3.1 Algoritmo di ricerca

La chiave di ricerca inserita viene **filtrata** dal programma per rimuovere eventuali caratteri non validi, quali quelli non alfabetici o spazi superflui. Successivamente, il programma utilizzerà la chiave filtrata per trovare dei risultati, mostrandoli **ordinati secondo un criterio di rilevanza** basato sul **grado di importanza** dei vari membri, dando precedenza agli **imperatori** e al **numero di legami relazionali**.

2.3.2 Suggerimenti di ricerca

Inserendo una chiave di ricerca che non porta ad alcun risultato, il programma proverà a produrre un **suggerimento**, controllando ogni sotto-stringa all'interno della query, cercando, per ognuna di esse, una corrispondenza; infine, il suggerimento mostrato sarà il **nome più corto tra le persone trovate**, per garantire la maggior quantità di riscontri e massimizzare la possibilità di fornire il risultato voluto.

2.3.3 Visualizzazione dei risultati

L'ordine di apparizione dei risultati viene scelto valutando il **numero di relazioni parentali e coniugali** che intercorrono nell'albero genealogico di ogni persona, assegnando ad ognuna di queste un punteggio dato dal proprio numero di genitori, figli e coniugi.

La scelta di utilizzare questo tipo di ordinamento è stata presa per poter mostrare in modo prioritario i membri delle dinastie con **più informazioni personali documentate** nella sua pagina di Wikipedia.

2.4 Funzioni generali

2.4.1 Interpretazione degli argomenti

Conclusa la fase preliminare, vengono interpretati gli **argomenti passati al programma**, attraverso i quali è possibile:

- visualizzare un **messaggio di help** sul terminale che elenca i possibili argomenti e le loro funzioni, e successivamente terminare l'esecuzione
- abilitare la **modalità di debug**, che permette di mostrare sul terminale dei messaggi relativi all'esecuzione del programma, utili per la risoluzione dei problemi
- avviare esclusivamente l'**esecuzione dei test**
- mostrare la **versione del programma**, e terminarne l'esecuzione
- abilitare la **modalità no-headless**, che modifica le impostazioni di Selenium per mostrare il browser usato dalla libreria

2.4.2 Multithreading

Per l'interfaccia grafica, il programma utilizza la libreria [Swing](#), la quale presenta alcune **limitazioni** per quanto riguarda la **gestione dei thread**; in particolare, Swing non possiede alcuna implementazione nativa per il multithreading. Ciò implica che le operazioni più onerose dal punto di vista computazionale, che verranno necessariamente eseguite nello stesso thread utilizzato da Swing, interromperanno la possibilità dell'utente di **interagire con la finestra**.

Per ovviare a questa problematica, il programma esegue determinate operazioni su **più thread**.

2.4.3 File JAR

Il codice è stato compilato anche sotto forma di JAR per favorire la **portabilità del programma**, quindi è stato necessario che questo fosse in grado di funzionare autonomamente:

- le **risorse dell'applicazione** sono state poste all'interno della cartella del sorgente in modo che il programma possa averne accesso, anche una volta all'interno del JAR
- sono stati implementati dei metodi capaci di **estrarre** dal file compresso le **dipendenze necessarie** al suo funzionamento

Ad esempio, all'avvio dell'applicazione, viene effettuato un controllo preliminare all'interno della cartella di sistema dedicata ai **file temporanei**, per verificare l'eventuale presenza di risorse del programma precedentemente importate che, in tal caso, verranno rimosse per **evitare conflitti**. Successivamente, in essa vengono **decompressi** i file necessari per il funzionamento di [Graphviz](#).

Capitolo 3

Struttura del programma e librerie utilizzate

3.1 Progettazione delle classi

La struttura del codice è stata progettata secondo un **sistema gerarchico** che permette ad ogni classe di svolgere esclusivamente un compito, rispettando il principio di singola responsabilità enunciato all'interno dei **paradigmi SOLID**. Di conseguenza, sono presenti delle **classi Manager** che mettono in comunicazione i singoli elementi del programma.

Una **lista completa e dettagliata** delle **classi** implementate e dei loro **metodi** può essere trovata al [seguente link](#).

3.1.1 Descrizione e gerarchia delle classi

Nel seguente **diagramma UML** viene rappresentata la gerarchia delle classi implementate all'interno del programma, descrivendo le varie **tipologie di relazione**.

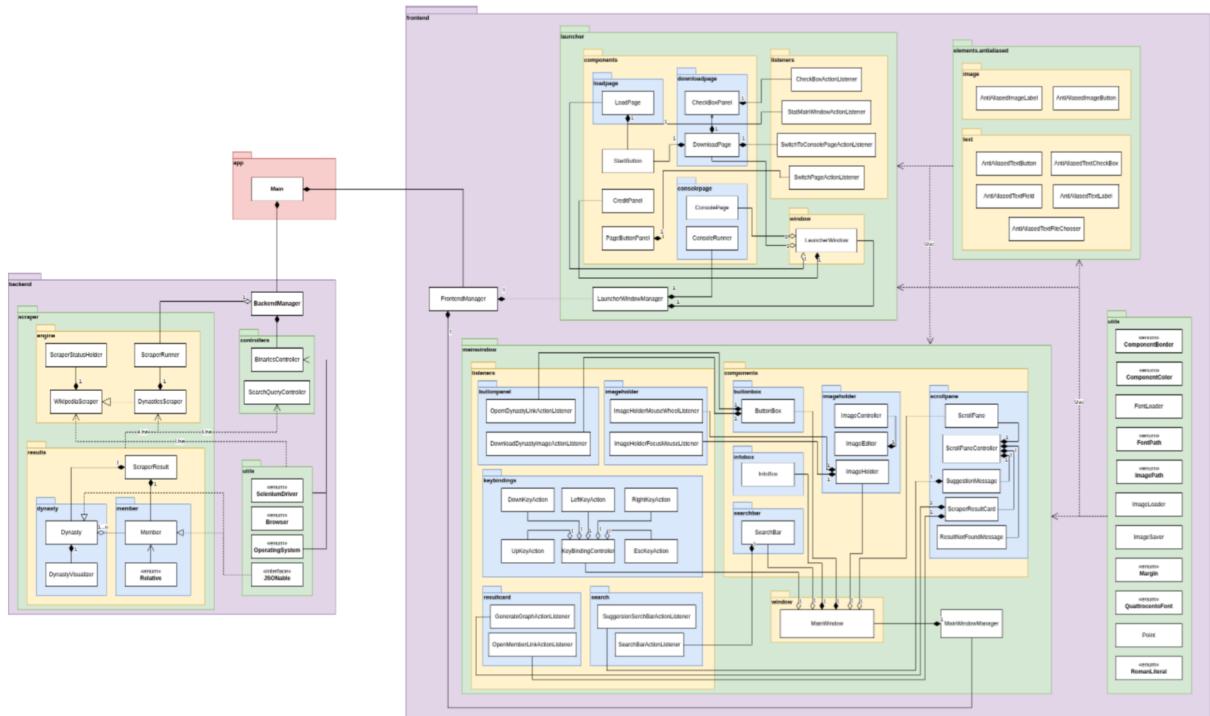


Figura 3.1: Diagramma UML raffigurante la gerarchia delle classi

Il programma risulta quindi essere diviso in due macro-pacchetti: il pacchetto **frontend**, contenente tutte le classi necessarie all'applicazione per poter gestire l'interfaccia grafica (GUI), e il pacchetto **backend**, contenente il vero e proprio codice relativo allo scraper.

Il pacchetto frontend potrà accedere alle funzionalità del programma **interfacciandosi** con il pacchetto backend, il quale assume il comportamento di un **servizio API**.

Per facilitare la lettura del diagramma, di seguito verranno analizzati i due macro-pacchetti in modo indipendente:

3.1.1.1 Pacchetto Backend

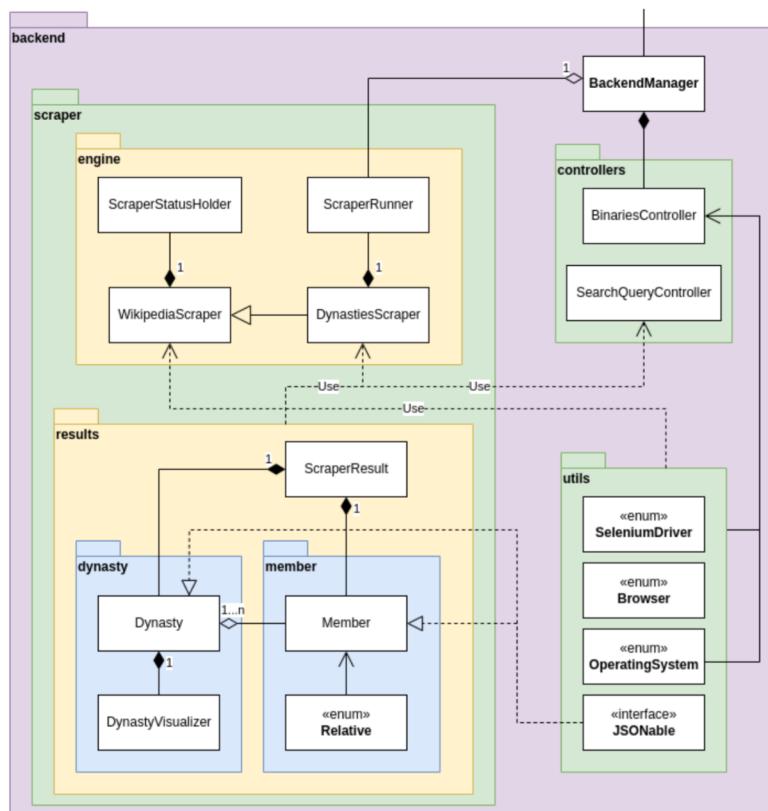


Figura 3.2: Il pacchetto backend, contenente le vere funzionalità del programma

- **BackendManager**: la classe principale del pacchetto backend. Tramite essa il frontend è in grado di accedere a tutte le funzionalità non grafiche del programma.
- **utils**:
 - **SeleniumDriver**: contiene le tipologie di driver utilizzabili tramite il framework Selenium e i metodi utilizzabili su di essi
 - **Browser**: contiene le tipologie di browser utilizzabili tramite il framework Selenium
 - **OperatingSystem**: contiene le tipologie di sistema operativo e i metodi utilizzabili su di essi

- **JSONable**: interfaccia in grado di rendere una classe serializzabile all'interno di un file JSON
- *controllers*:
 - **BinariesController**: gestisce l'estrazione e la lettura di librerie e file binari necessari al backend
 - **SearchQueryController**: gestisce la ricerca di una dinastia o di un suo membro specifico tramite una query
- *scraper*:
 - *engine*:
 - * **WikipediaScraper**: un generico scraper in grado di interpretare le pagine wikipedia, raccogliendone le informazioni
 - * **ScraperStatusHolder**: contiene lo status attuale del WikipediaScraper
 - * **DynastyScraper**: eredita da WikipediaScraper assumendo funzionalità più specifiche, in particolare la capacità di ricostruire le dinastie richieste dal programma
 - * **ScraperRunner**: esegue e controlla un oggetto DynastyScraper all'interno di un processo multi-thread secondario
 - *results*:
 - * **ScraperResult**: un oggetto rappresentante una tupla (Dinastia, Membro). Implementa l'interfaccia Comparable al fine di poter confrontare più risultati tra di loro
 - * *dynasty*:
 - **Dynasty**: descrive tutti gli attributi e metodi relativi ad una dinastia. Implementa l'interfaccia JSONable, rendendo un oggetto di questo tipo in grado di essere serializzato all'interno di un JSON
 - **DynastyVisualizer**: genera un grafico a partire da una dinastia o da un suo membro specifico
 - * *member*:
 - **Dynasty!!!**: descrive tutti gli attributi e metodi relativi ad un membro di una dinastia. Implementa l'interfaccia JSONable, rendendo un oggetto di questo tipo in grado di essere serializzato all'interno di un JSON
 - **Relative**: un enum descrivente i vari gradi di parentela possibili tra uno o più membri

3.1.1.2 Pacchetto Frontend

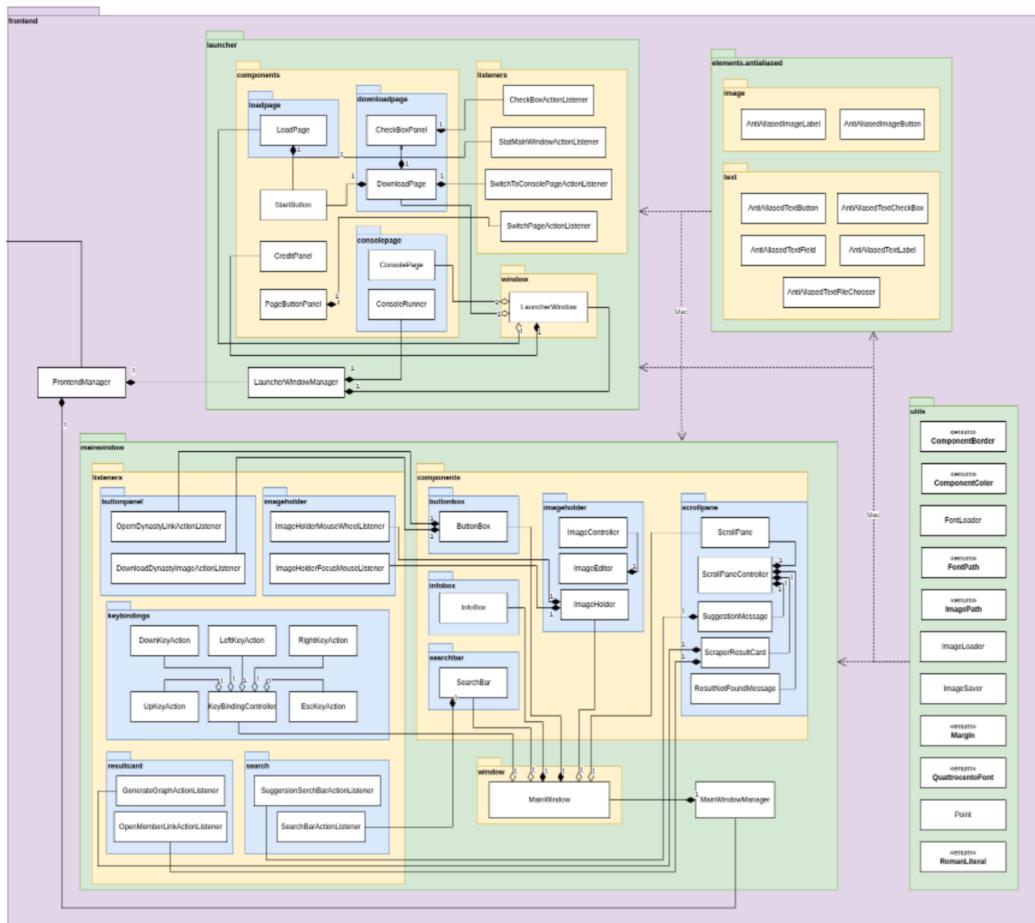


Figura 3.3: Il pacchetto frontend, contenente la GUI del programma

- **FrontendManager**: la classe principale del pacchetto frontend, mettendo in comunicazione le altre classi Manager presenti nei sotto-pacchetti
- *elements.antialiased*:
 - *image*:
 - * **AntiAliasedImageLabel**: eredita da `JLabel` sovrascrivendo il suo metodo `paintComponent()`, adattato per le immagini
 - * **AntiAliasedImageButton**: eredita da `JButton` sovrascrivendo il suo metodo `paintComponent()`, adattato per le immagini
 - *text*
 - * **AntiAliasedTextLabel**: eredita da `JLabel` sovrascrivendo il suo metodo `paintComponent()`
 - * **AntiAliasedTextButton**: eredita da `JButton` sovrascrivendo il suo metodo `paintComponent()`
 - * **AntiAliasedTextCheckBox**: eredita da `JTextCheckBox` sovrascrivendo il suo metodo `paintComponent()`

- * **AntiAliasedTextField**: eredita da JTextField sovrascrivendo il suo metodo paintComponent()
- * **AntiAliasedTextFileChooser**: eredita da JFileChooser sovrascrivendo il suo metodo paintComponent()

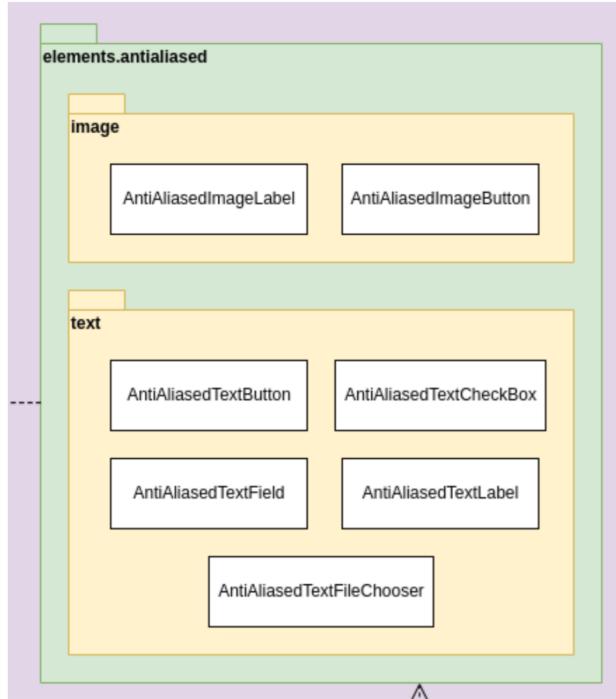
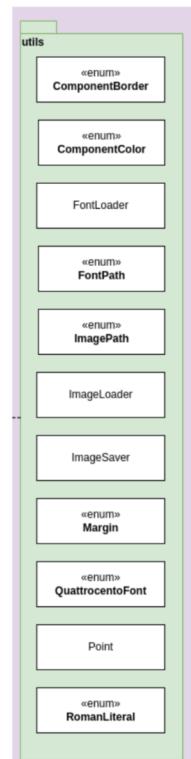


Figura 3.4: Il pacchetto elements.antialiased, contenente i JComponent modificati

- *utils*

- **ComponentBorder**: contiene i bordi utilizzabili all'interno della grafica
- **ComponentColor**: contiene i colori utilizzabili all'interno della grafica
- **Margin**: contiene i margini utilizzabili all'interno della grafica
- **QuattrocentoFont**: contiene i vari stili del font utilizzabili all'interno della grafica
- **FontLoader**: gestisce il caricamento dei font all'interno della grafica
- **ImageLoader**: gestisce il caricamento delle immagini all'interno della grafica
- **ImageSaver**: gestisce il salvataggio delle immagini in un processo multi-thread secondario
- **FontPath**: contiene i path dei font caricabili
- **ImagePath**: contiene i path delle caricabili
- **RomanLiteral**: contiene le conversioni da numero decimale a letterale romano
- **Point**: rappresenta una tupla (X, Y)



- *launcher*:

- **Launcher WindowManager**: la classe principale del pacchetto launcher. Tramite essa tutti gli elementi della pagina Launcher possono comunicare tra di loro
- *window*:
 - * **LauncherWindow**: il frame principale contenente tutti gli elementi della pagina Launcher
- *components*:
 - * **StartButton**: il bottone di avvio del programma
 - * **CreditPanel**: il pannello contenente i nomi degli autori del progetto
 - * **PageButtonPanel**: il pannello contenente i bottoni con cui poter cambiare pagina
 - * *loadpage*:
 - **LoadPage**: la pagina di caricamento delle dinastie pre-salvate
 - * *downloadpage*:
 - **CheckBoxPanel**: il pannello contenente i checkbox con cui poter selezionare la modalità di scraping
 - **DownloadPage**: la pagina di scaricamento delle dinastie
 - * *consolepage*:
 - **ConsolePage**: la console in cui viene mostrato lo scraping in tempo reale
 - **ConsoleRunner**: esegue e controlla un oggetto ConsolePage all'interno di un processo multi-thread secondario
- *listeners*:
 - * **CheckBoxActionListener**: action listener utilizzato da CheckBoxPanel
 - * **StartMainWindowActionListener**: action listener utilizzato da LoadPage
 - * **SwitchToConsolePageActionListener**: action listener utilizzato da DownloadPage
 - * **SwitchPageActionListener**: action listener utilizzato da PageButtonPanel

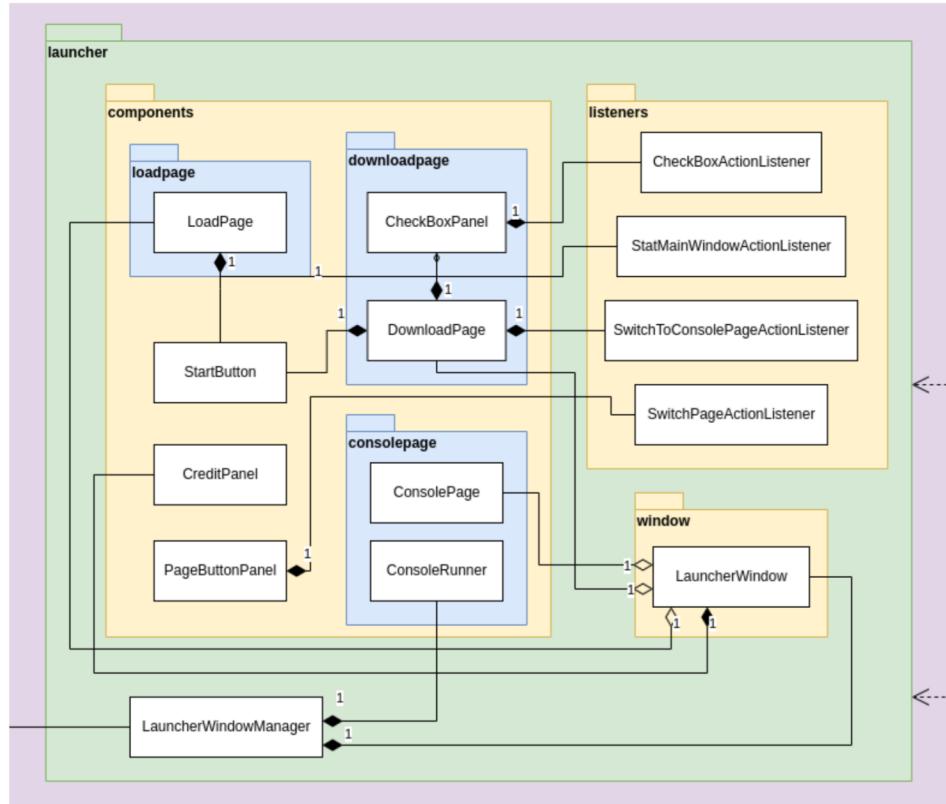


Figura 3.5: Il pacchetto launcher, contenente gli elementi della pagina Launcher

- *mainwindow*:

- **MainWindowManager**: a classe principale del pacchetto `mainwindow`. Tramite essa tutti gli elementi della pagina Principale possono comunicare tra di loro

- *window*:

- * **MainWindow**: il frame principale contenente tutti gli elementi della pagina Principale

- *components*:

- * *buttonbox*:

- **ButtonBox**: il box-panel contenente i bottoni relativi alle dinastie

- * *infobox*:

- **InfoBox**: il box-panel contenente delle info relative all'ImageHolder

- * *searchbar*:

- **SearchBar**: la barra di ricerca con cui poter filtrare i membri da mostrare nello ScrollPane

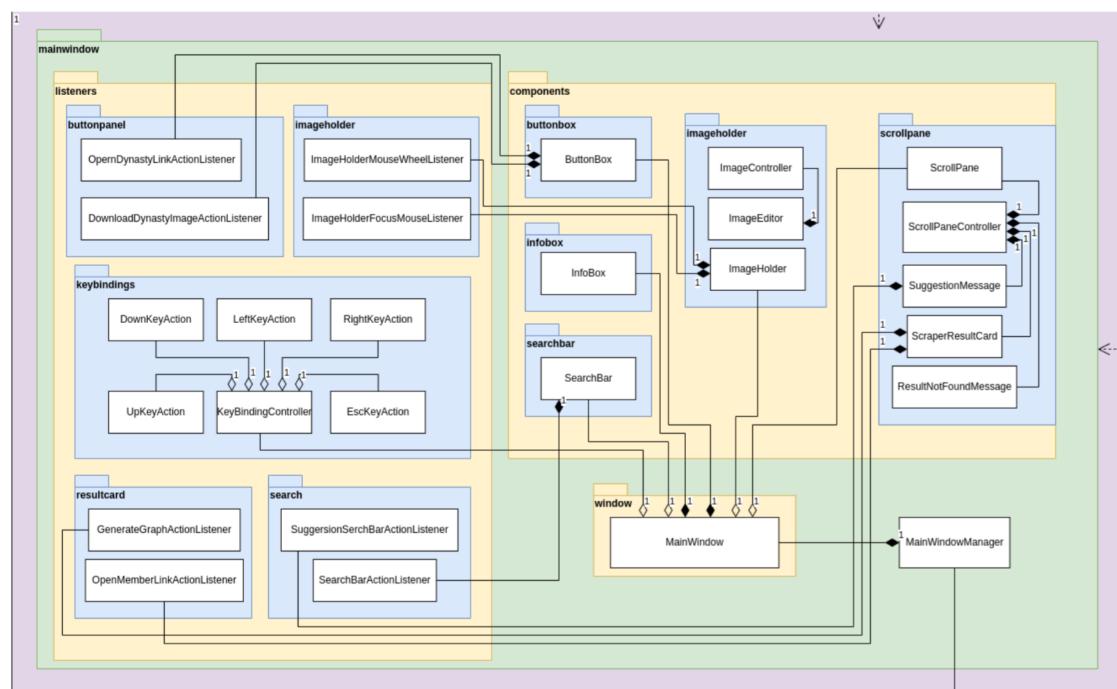
- * *imageholder*:

- **ImageHolder**: il pannello contenente il grafico relativo al membro selezionato

- **ImageController**: gestisce le operazioni da effettuare sull'ImageHolder
 - **ImageEditor**: modifica le immagini caricate nell'ImageHolder
- * *scrollpane*:
- **ScrollPane**: il pannello scorribile contenente i vari ScraperResultCard relativi alla ricerca effettuata
 - **ScrollPaneController**: gestisce le operazioni da effettuare sullo Scroll-Pane
 - **ScraperResultCard**: un card-panel contenente le informazioni relative ad un membro di una dinastia
 - **SuggestionMessage**: la label contenente il suggerimento visualizzato in caso di assenza di risultati trovati nella ricerca
 - **ResultNotFoundMessage**: la label contenente il messaggio visualizzato in caso di assenza di risultati trovati nella ricerca
- *listeners*:
- * *buttonpanel*:
 - **OpenDynastyLinkActionListener**: action listener utilizzato da ButtonBox
 - **DownloadDynastyLinkActionListener**: action listener utilizzato da ButtonBox
- * *imageholder*:
- **ImageHolderMouseWheelListener**: action listener utilizzato da ImageHolder
 - **ImageHolderFocusMouseListener**: action listener utilizzato da ImageHolder
- * *keybindings*:
- **DownKeyAction**: definisce l'azione da svolgere alla pressione della freccia in giù della tastiera
 - **UpKeyAction**: definisce l'azione da svolgere alla pressione della freccia in su della tastiera
 - **LeftKeyAction**: definisce l'azione da svolgere alla pressione della freccia a sinistra della tastiera
 - **RightKeyAction**: definisce l'azione da svolgere alla pressione della freccia a destra della tastiera
 - **EscKeyAction**: definisce l'azione da svolgere alla pressione della pulsante Esc della tastiera
 - **KeyBindingController**: imposta e gestisce tutti gli eventi eseguibili tramite tastiera

* *resultcard*:

- **GenerateGraphActionListener**: action listener utilizzato da ScraperResultCard
 - **OpenMemberLinkActionListener**: action listener utilizzato da ScraperResultCard
- * *search*:
- **SuggestionSearchBarActionListener**: action listener utilizzato da SuggestionMessage
 - **SearchBarActionListener**: action listener utilizzato da SearchBar

Figura 3.6: Il pacchetto `mainwindow`, contenente gli elementi della pagina Principale

3.2 Librerie utilizzate

- **Selenium 4.1.4**

La libreria di **Selenium** per Java è stata utilizzata per effettuare lo scraping delle pagine da Wikipedia, in particolare per eseguire le richieste del codice HTML delle pagine.

Inoltre, il programma dispone di alcuni driver per Selenium, sia per **Windows** che per **UNIX**:

- Microsoft Edge
- Google Chrome
- Mozilla Firefox

- **Log4j 1.2.17** e **SLF4J-Log4j 1.6.4**

Le librerie **Log4j** e **SLF4J** vengono utilizzate per monitorare gli output di Graphviz.

- **Graphviz 0.18.1**

La libreria **Graphviz** per Java viene utilizzata per la generazione degli alberi genealogici a partire dai dati ottenuti dallo scraping.

- **Flatlaf 2.2**

Flatlaf è una libreria multi-piattaforma per modificare il Look and Feel di Swing, e viene utilizzata per aumentare la personalizzazione dell'interfaccia grafica. Alcuni dei suoi utilizzi sono: la possibilità di impostare bordi curvi ai componenti, migliorare l'aspetto grafico del JFileChooser, modificare la posizione e le dimensioni della barra di scorrimento del JScrollPane.

- **Jsoup 1.14.3**

La libreria **Jsoup** viene utilizzata dal programma per effettuare il parsing del codice HTML delle pagine di Wikipedia.

- **JSON-Java 20220320**

La libreria **JSON-Java** viene utilizzata dal programma sia per effettuare il salvataggio in locale delle informazioni delle dinastie in file JSON, sia per interpretare questi ultimi e caricare le dinastie in memoria.

- **Thumbnailator 0.4.8**

La libreria **Thumbnailator** fornisce dei metodi efficienti per il ridimensionamento dell'immagine, indispensabili per la navigazione del grafico in tempo reale.

- **JDeli 2022.05**

JDeli è una libreria che permette il salvataggio efficiente di molte tipologie di file. Inoltre, effettua il salvataggio in blocchi, alleggerendo l'utilizzo delle risorse del programma.

Capitolo 4

Manuale della grafica

4.1 Finestra di avvio

All'avvio del programma, la prima schermata che viene mostrata permette all'utente di scegliere se **caricare** le dinastie **salvate in locale**, qualora ve ne fossero, oppure **scaricarle dal web** tramite **scraping**.

4.1.1 Dinastie non trovate

Nel caso in cui i dati delle dinastie **non siano** salvati in locale, ad esempio al primo avvio, verrà mostrata la seguente schermata:

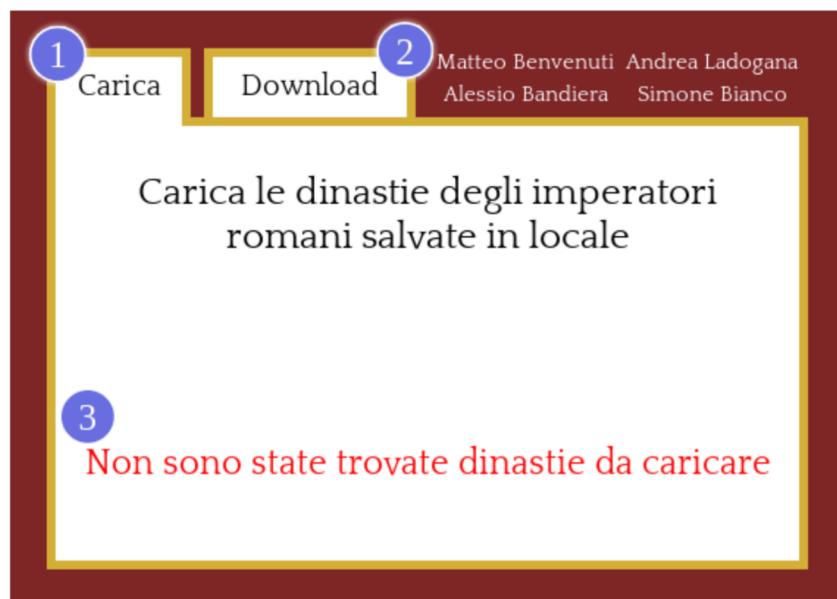


Figura 4.1: Schermata "Carica" con file delle dinastie non trovati.

1. Il **pulsante "Carica"** mostra la schermata di caricamento.
2. Il **pulsante "Download"** mostra la schermata di download.
3. Questo messaggio indica che non sono state trovate dinastie nella directory del programma.

4.1.2 Download delle dinastie

Nella pagina di download, invece, si potranno **scaricare** i dati delle dinastie e **salvarli in locale**:



Figura 4.2: Schermata di download delle dinastie.

1. Questa casella, se spuntata, indica che il download verrà effettuato utilizzando Selenium.
2. Questa casella, se spuntata, indica che il download verrà effettuato utilizzando le richieste HTTP.
3. Il **pulsante "Avvia"** farà partire la finestra principale del programma, mostrando il download ed il salvataggio delle dinastie in locale tramite una **console**:



Figura 4.3: Console per la visualizzazione del download delle dinastie.

4.1.3 Dinastie trovate

Nel caso in cui i dati delle dinastie **vengano trovati** nella directory del programma, la schermata "Carica" apparirà in questo modo:

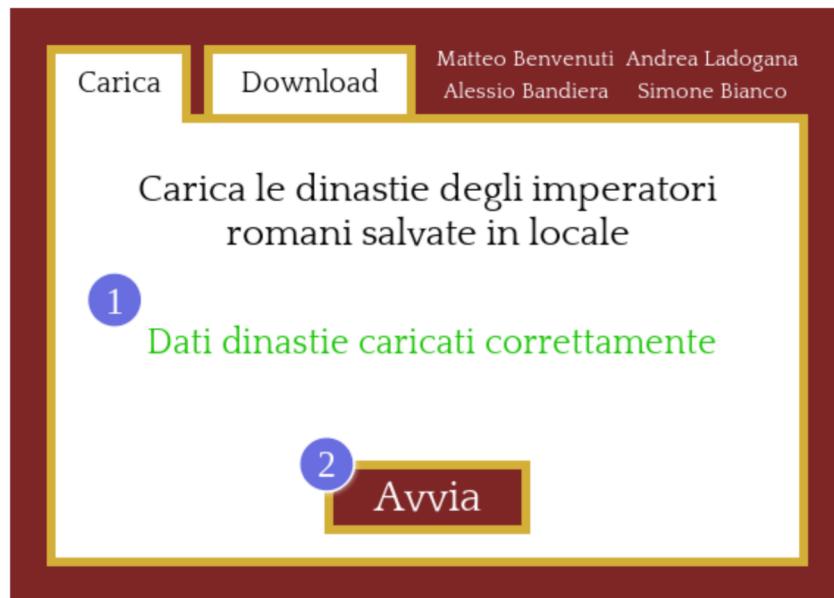


Figura 4.4: Schermata "Carica" con file delle dinastie caricati.

1. Questo messaggio indica che le dinastie sono state caricate correttamente.
2. Il pulsante "**Avvia**" farà partire la finestra principale del programma utilizzando i dati trovati in locale, quindi senza eseguire nessun download.

4.2 Finestra di ricerca

Una volta caricate le dinastie all'interno del programma, dopo aver premuto il pulsante "Avvia", apparirà la **finestra di ricerca**, tramite la quale sarà possibile **cercare le persone** all'interno delle dinastie caricate e visualizzarne l'**albero genealogico**.

4.2.1 Struttura generale

Una volta avviata, la finestra principale si presenta divisa in diverse sezioni:

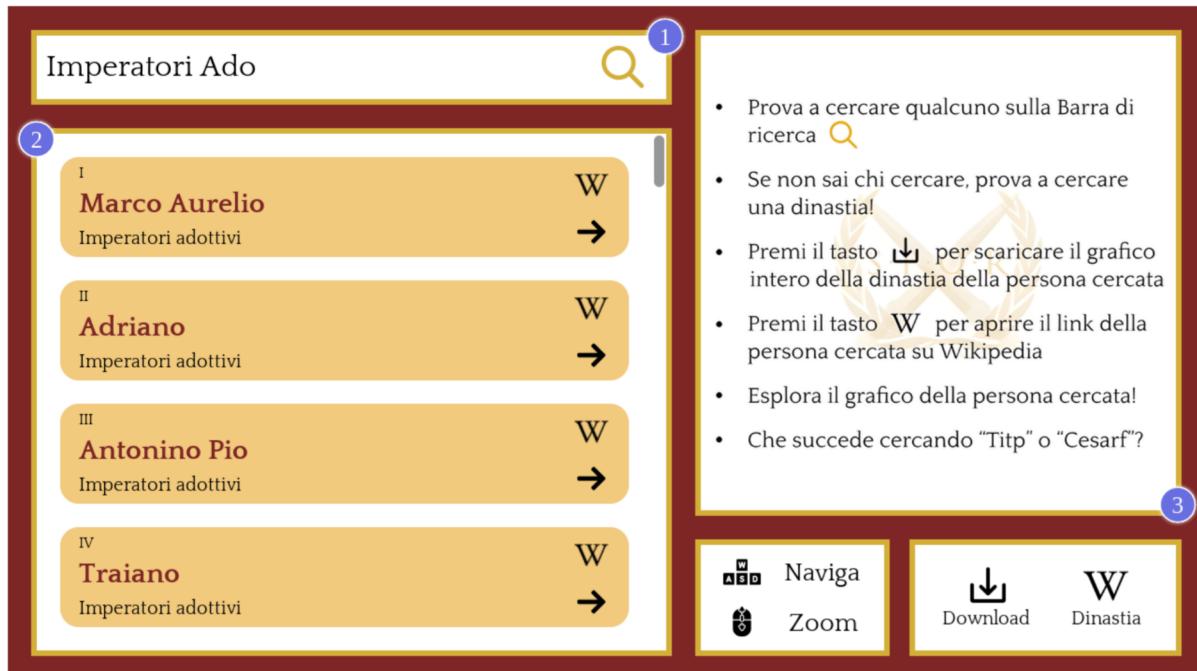


Figura 4.5: Finestra di ricerca con grafico non caricato (all'avvio).

1. La barra di ricerca, nella quale si potrà inserire il nome o la dinastia che si vuole cercare; per effettuare la ricerca premere il tasto invio, oppure cliccare la lente di ingrandimento.
2. Dopo aver effettuato la ricerca, il suo esito verrà mostrato nel pannello dei risultati; se il numero dei risultati è troppo elevato, sulla destra apparirà una barra di scorrimento che, se trascinata verso il basso, permetterà di visualizzare ulteriori risultati; il pannello può essere navigato anche usando la rotella del mouse.
3. Se non è stato caricato nessun grafico, il pannello di visualizzazione mostrerà una schermata contenente suggerimenti riguardanti l'utilizzo del programma.

4.2.2 Suggerimento di ricerca

In caso di ricerca senza alcun risultato, il programma mostrerà un suggerimento di ricerca:

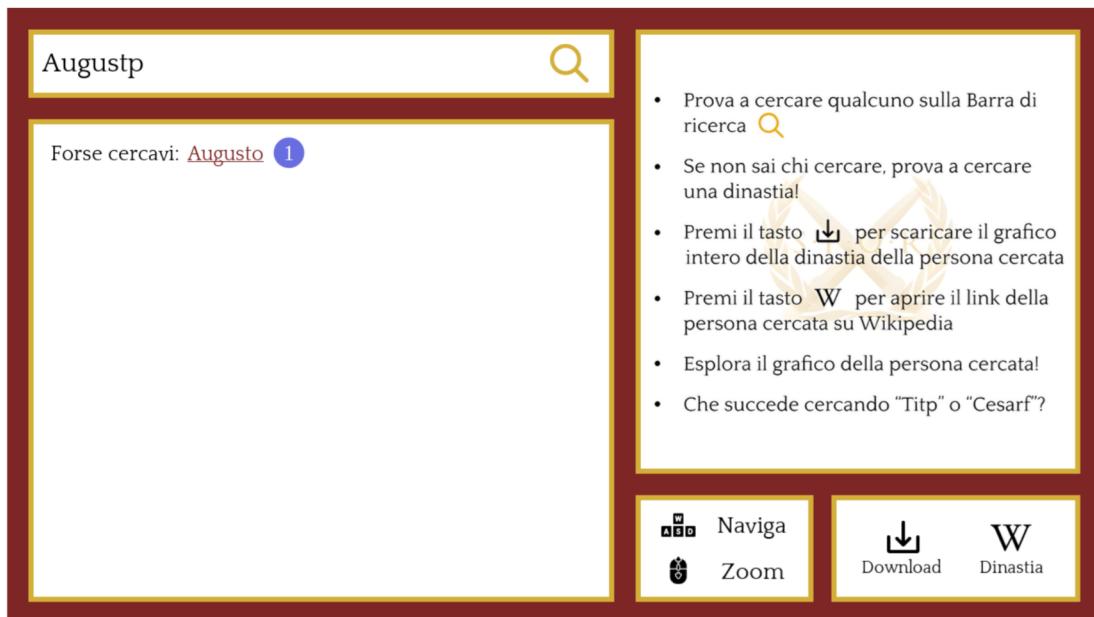


Figura 4.6: Pannello dei risultati con suggerimento.

1. Una volta cliccato il suggerimento (in rosso), il programma lo ricercherà e ne mostrerà i risultati.

In caso non sia possibile generare suggerimenti, verrà mostrata la seguente schermata:

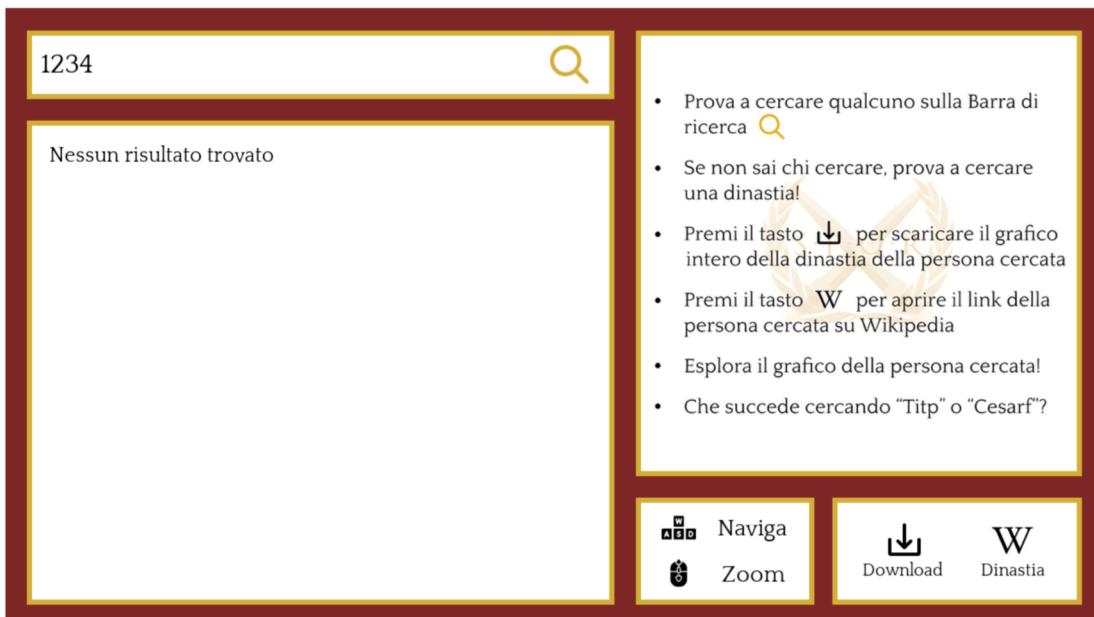


Figura 4.7: Finestra di ricerca con grafico caricato

4.2.3 Risultati di ricerca e visualizzazione del grafico

La finestra di ricerca del programma si presenta in questo modo:

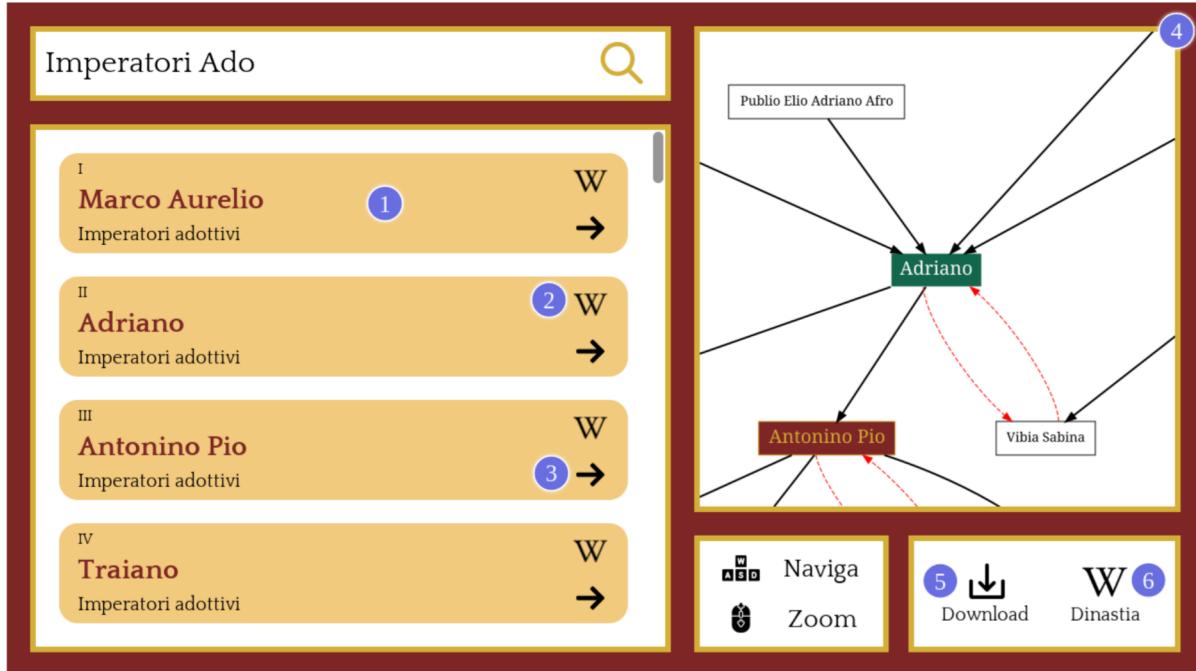


Figura 4.8: Finestra di ricerca con grafico caricato.

1. I risultati appaiono in delle card che mostrano i nomi delle persone e le rispettive dinastie di appartenenza; il nome degli imperatori è colorato in rosso, inoltre i risultati sono ordinati per importanza.
2. Il pulsante raffigurante una "W", se cliccato, porterà alla pagina di Wikipedia della persona corrispondente.
3. Il pulsante raffigurante una freccia caricherà nel pannello di visualizzazione l'albero genealogico della persona cercata.
4. Il pannello di visualizzazione mostra il grafico della persona cercata, e gli imperatori appartenenti all'albero genealogico sono mostrati in rosso; i legami di parentela tra genitori e figli è indicato con frecce nere (anche per genitori adottivi), mentre i legami coniugali sono rappresentati con frecce rosse tratteggiate; infine, il grafico viene mostrato centrato sulla persona cercata, colorata in verde.
5. Questo pulsante permette di salvare l'immagine della dinastia *completa* a cui appartiene la persona cercata.
6. Questo pulsante, raffigurante una "W", se cliccato, porterà alla pagina di Wikipedia della dinastia a cui appartiene la persona cercata.

Capitolo 5

Ripartizione dei compiti

5.1 Legenda

- Matteo Benvenuti
- Andrea Ladogana
- Simone Bianco
- Alessio Bandiera

5.2 Ripartizione

• Relazione

Funzionalità ● ●

UML, gerarchia e descrizioni delle classi ● ●

Manuale della grafica ● ●

• Interfaccia Grafica

Pannello di scorrimento dei risultati ●

Pianificazione delle interfacce grafiche ● ●

Interfaccia della finestra di avvio ● ●

Interfaccia della console ● ●

Informazioni di aiuto iniziali ● ●

Grafica del suggerimento ● ●

Centramento del grafico ● ●

Grafica del risultato assente ● ●

Gestione del caricamento delle immagini e dei font ● ●

Interfaccia della finestra principale ● ●

Componenti Antialiased ● ●

Navigazione del grafico ● ●

Apertura dei link con browser predefinito ●

Apertura del gestore dei file per salvataggio del grafico ●

- **Backend**

Controllo e caricamento dei file JSON • •

Scraping di Wikipedia • •

Interpretazione dati Scraper • •

Gestione dei membri delle dinastie • •

Generazione delle dinastie • •

Algoritmo di ricerca • •

Ordinamento dei risultati • •

Estrazione degli eseguibili dal JAR • •

Gestione del framework Selenium •

Gestione dei driver per Selenium •

Generazione del grafico delle dinastie •

Serializzazione delle dinastie •

- **Multithreading**

Gestione della console • •

Scraping delle dinastie •

Salvataggio del grafico •

- **Altro**

Refactoring del progetto e del codice • • • •

Gestione della [repository GitHub](#) • • • •

Script di compilazione del programma •

Classi di testing •

Aggiornamento README e TODO list •

Redazione della Javadoc •