UNIVERSITÀ DEGLI STUDI DI UDINE

Dipartimento di Scienze Matematiche, Informatiche e Fisiche

Corso di Laurea Triennale in Internet of Things, Big Data & Web

Tesi di Laurea

PROGETTAZIONE E SVILUPPO DI UN CRAWLER PER IL REPERIMENTO AUTOMATICO DI IMMAGINI

Relatore:
Prof. IVAN SCAGNETTO

Laureando:
MASSIMILIANO BALDO

ANNO ACCADEMICO 2020-2021

Indice

Capitolo 1

Introduzione

1.1 Ambito di ricerca

Il presente elaborato ha come scopo l'implementazione di uno strumento di supporto per l'addestramento di reti neurali. In particolare, la necessità di tale strumento è nata in seguito all'esigenza di sviluppare un sistema in grado di acquisire immagini dall'ambiente circostante al fine di determinare la presenza eventuale di ordigni inesplosi. Si tratta di un problema che affligge una vasta porzione del territorio di molti stati, tra cui l'Italia, ed è dovuto non solo al rinvenimento di residui bellici di guerre passate, ma anche alla bonifica di aree prima dedicate all'attività di esercitazioni militari. Per quanto riguarda la parte di riconoscimento visivo di un ordigno, uno dei requisiti principali era di essere in grado di effettuare il rilevamento in ogni possibile scenario, con particolare riguardo ad ambienti subacquei. Tecnicamente si era deciso di puntare su algoritmi di deep learning e, in particolare, su una rete neurale di tipo convoluzionale. Uno degli ostacoli principali che si è riscontrato era nella parte di "apprendimento", ovvero nel momento in cui la rete neurale impara a riconoscere la presenza o meno di uno specifico oggetto all'interno dell'immagine (o frame). In particolare, il problema consisteva nella mancanza di materiale sul quale la rete neurale cerca di apprendere. Questo fatto portava ad avere delle performance molto basse. Pertanto si doveva ampliare il campo possibile di immagini inerenti a bombe inesplose, in modo tale da averne una mole consistente affinché la rete neurale potesse diventare più affidabile. Per reperire una certa quantità consistente di immagini specifiche, le possibili opzioni attuali sono:

- Trovare una raccolta che sia conforme a quello che si desidera avere, o almeno in parte,
- Selezionare e scaricare manualmente i dati necessari, cercandoli in rete.

L'obiettivo di questo scritto è automatizzare la ricerca e il salvataggio, in un supporto di memorizzazione locale, di immagini presenti sul web e correlate a determinate parole chiave.

1.2 Risultati ottenuti

Quello che si voleva creare era un programma a linea di comando (CLI)[1], che permettesse all'utente di reperire un determinato certo numero di immagini inerenti a un argomento (specificato mediante delle parole chiave). Si è ottenuto un programma che è possibile eseguire tramite terminale[2] in modo tale che un qualsiasi dispositivo, anche privo di un'interfaccia grafica, sia in grado di eseguire il software.

1.3 Sinossi

Lo sviluppo della tesi è suddiviso come segue:

- Nel capitolo 2 si definiscono i concetti chiave del web scraping e i principali servizi di crawling.
- Nel capitolo 3 si illustrano i requisiti che il sistema deve avere.
- Nel capitolo 4 si mostrano le tecniche e le tecnologie usate per creare un crawler, in aggiunta ci sono dei consigli di implementazione.
- Nel capitolo 5 si descrive come è stato implementato il sistema e delle problematiche riscontrate.
- Nel capitolo 7 è presente una conclusione con dei possibili sviluppi futuri per l'applicazione.

[1] Command Line Interface

[2] Il terminale è lo strumento col quale l'utente comunica col sistema di calcolo, introducendo comandi, programmi e dati, controllandone il funzionamento e ricevendo su video risultati e altre informazioni. Traduzione citata dal sito https://www.treccani.it/vocabolario/terminale/

Capitolo 2

Stato dell'arte attuale

In questo capitolo si definiscono i concetti chiavi come "motore di ricerca" (Sezione 2.1) e "crawler" (Sezione 2.2).

Successivamente, si citano i diversi servizi di scraping dei siti web (Sezione 2.3) con le rispettive recensioni (Sezioni 2.3.1-2.3.3).

2.1 Cos'è un motore di ricerca

Un motore di ricerca è un software in grado di reperire informazioni dal WWW[1] sfruttando parole chiave oppure frasi. La caratteristica principale di questo software è la rapidità con la quale riesce a trovare informazioni, pur essendoci potenzialmente milioni di possibili risultati correlati. Questa peculiarità è possibile grazie all'uso di crawler, i quali scansionano l'interno WWW e indicizzano i risultati scoperti in maniera continua. Si consulti il seguente riferimento per una documentazione più completa [1].

2.2 Cos'è un crawler

È giusto premettere che non esiste un vero e proprio standard nella nomenclatura inerente al web scraping[2], in particolare non esiste una guida che espliciti chiaramente quando usare un termine rispetto a un altro. In molti casi, si usa come sinonimo di crawler il termine *"web scraper"* oppure, molto più generico, il termine *"bot"*.

[1]World Wide Web: sistema interconnesso di pagine web pubbliche, accessibili tramite internet. Definizione tradotta da https://developer.mozilla.org

[2]Insieme di azioni eseguite autonomamente da un programma specifico. Traduzione dal sito https://en.wikipedia.org/wiki/Web_scraping

Quello che segue è un raggruppamento delle principali caratteristiche che contraddistinguono i diversi nomi, secondo le due distinte fonti [2] [3]:

- per **crawler** si intende un programma che ha il compito di analizzare in modo autonomo una pagina web, seguendo tutti i possibili riferimenti esterni (ovvero link ad altre pagine web) ed indicizzando ognuna delle pagine analizzate;
- per **web scraper** si intende un software che analizza pagine autonomamente, ma estrapola specifiche informazioni e le salva (in locale o in remoto) affinché vengano analizzate o elaborate in un secondo momento;
- per **bot** si intende un qualsiasi programma automatizzato per risolvere un specifico compito.

Solitamente, il termine *bot* non viene usato comunemente poiché troppo generico rispetto a quello che effettivamente un crawler, come un web scraper, può fare.

Come citato a inizio paragrafo, non esiste una divisione netta tra quello che può essere definito *crawler* oppure *web scraper*. Pertanto è possibile usare i due termini alternativamente, come per esempio in questo stesso elaborato, senza incombere in errori di nomenclatura.

Infine, se un crawler è specializzato per una specifica tipologia di dati (come ad esempio video, immagini oppure anche dati tabulati) è possibile usare la terminologia *entità*-crawler. Un esempio è il caso trattato in questo componimento, il quale si specializza nella creazione di un *image-crawler*.

2.3 Principali servizi per l'image crawling

Attualmente i principali servizi per l'analisi e l'elaborazione di informazioni dai siti sono i seguenti: Octoparse , WebScrapingApi e WebScraper. Per ognuno di questi si riporta una breve descrizione e le principali funzionalità.

2.3.1 Octoparse

Octoparse è uno strumento di web crawling basato su client per ottenere dati web in fogli di calcolo. Con un'interfaccia point-and-click[3], il software è fondamentalmente costruito per i non programmatori in modo tale da rendere agevole l'interazione con il sistema.

[3]Punta e clicca: si intende usa interfaccia nella quale si una il click del mouse per selezione ciò che si desidera (può essere d'esempio un titolo, una descrizione oppure un prezzo e così via).

Le funzionalità più rilevanti sono le seguenti:

- scraper pre-costruiti: tool costruiti a priori per analizzare e reperire i dati da siti web popolari come Amazon, eBay, Twitter, ecc.;
- modalità auto-detection: dopo aver inserito l'URL di destinazione, il sistema rileverà automaticamente i dati strutturati pronti per essere scaricati;
- modalità avanzata: consente agli utenti di personalizzare lo scraper per estrarre i dati desiderati da siti complessi;
- formato dei dati: è possibile specificare il formato dei dati (come per esempio xlx, xml, html, csv) o spedirli direttamente a un database tramite API[4].

Per una analisi più dettagliata sul servizio e l'azienda software, che lo ha sviluppato, è possibile fare riferimento al loro sito [4].

2.3.2 WebScraper

WebScraper è una estensione per browser che propone una interfaccia point-and-click per la selezione degli elementi da reperire.

Le caratteristiche principali sono:

- estrazione dei dati da siti dinamici;
- formato dei dati: anche in questo caso è possibile specificare il formato dei dati;
- possibilità di eseguire in cloud il software attraverso uno scheduler, senza dover installare nessun programma.

È possibile consultare il sito ufficiale per una descrizione più approfondita alla seguente voce [5].

2.3.3 WebScrapingApi

WebScrapingApi è un insieme di API che permettono lo "scraping" delle pagine web.

I suoi punti di forza sono:

[4]Application Programming Interface: è l'insieme di definizioni e protocolli per la creazione ed integrazione di applicazioni software. Traduzione e maggiori dettagli sono consultabili dal sito https://www.redhat.com/.

- uso di tecnologie cloud Amazon Web Services per la scalabilità del software;
- implementazione di un sistema di "rotazione proxy" in modo tale da non poter essere bloccati dal sito web in quanto eseguendo molte richieste contemporaneamente.

Si riporta il sito ufficiale [6] per una lettura approfondita.

2.4 Conclusioni

Tutti i software descritti nella Sezione 2.3 offrono solo un breve periodo di prova dei servizi con molte limitazioni. Pertanto non ci si poteva affidare a una soluzione già esistente e si è deciso di creare un image crawler ad hoc per la situazione descritta nella Sezione 1.1.

Capitolo 3

Obiettivi

In questo capitolo si descrivono in maniera grezza i requisiti richiesti dal sistema (Sezione 3.1) e successivamente tali requisiti vengono estrapolati e strutturati (Sezione 3.2).

3.1 Obiettivi generali

Come descritto nella Sezione 2.4, le principali soluzioni inerenti al web scraping presenti sul mercato non sono esaustive in funzione dell'ambito di ricerca, stabilito nella Sezione 1.1. Intuitivamente, si vuole ottenere un programma che sia accessibile a chiunque, e soprattutto possa essere usufruito da qualsiasi utilizzatore, indipendentemente dalla tipologia o dalla potenza di calcolo del dispositivo in uso. Sicuramente, il software deve essere in grado di agire da solo, sapendo affrontare qualsiasi casistica in modo tale da poter ottenere le immagini desiderate. Successivamente, il programma deve poter essere invocato facilmente, si vuole cercare di ottenere nel minor tempo possibile una quantità discreta di immagini. In aggiunta, si desidera che il software possa essere configurato in base ad alcuni parametri, come per esempio il numero di immagini da reperire. Infine, il programma deve essere *auto contenuto*, ovvero non deve esserci necessità di installare ulteriori pacchetti o software affinché si possa utilizzare il programma principale.

3.2 Obiettivi dettagliati

Una volta definiti i requisiti sotto forma di testo naturale, è utile individuare ogni singola voce da rispettare. Pertanto, si riscrivono i requisiti sotto forma di lista.

Quello che si evince è che il programma deve:

- essere portabile[1],
- essere semi-autonomo,
- essere lightweight[2],
- prevedere una interazione tramite terminale,
- poter ricevere parametri con i quali interagire,
- essere auto contenuto.

Si riporta un diagramma DFD[3] di livello 0 nel quale si mostra il flusso dei dati inerenti al sistema.

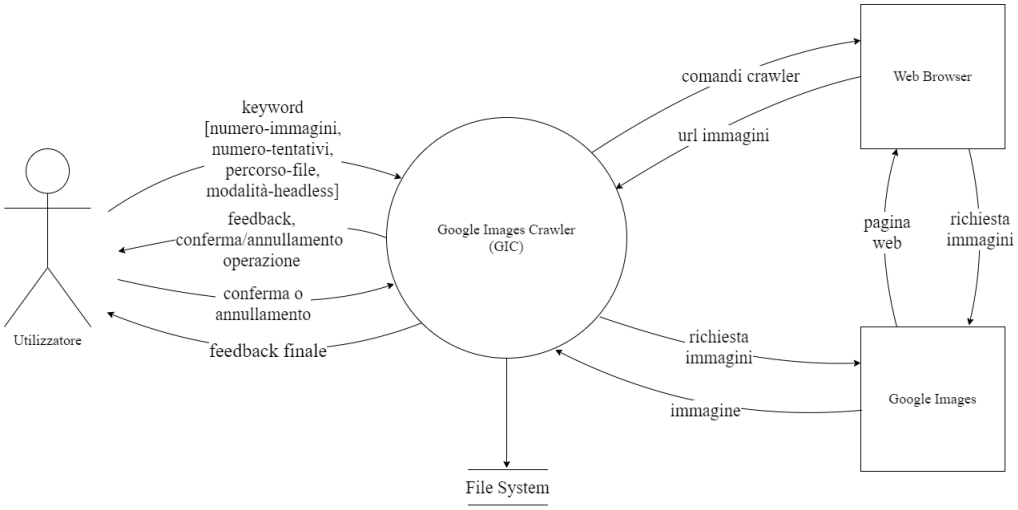


Figura 3.1: Diagramma DFD di livello 0

[1]L'aggettivo indica un programma che possa essere usato su qualsiasi dispositivo senza preoccuparsi di possibili incompatibilità con il sistema.

[2]Peso leggero: termine usato per definire un software che non occupi troppo spazio nel disco locale.

[3]Data Flow Diagram: diagramma usato per rappresentare il flusso dei dati di un sistema.

Capitolo 4

Tecniche e tecnologie di progettazione di un crawler

In questo capitolo si descrivono le principali tipologie e tecnologie per l'implementazione di un crawler (Sezione 4.1 e Sezione 4.3). Successivamente, si descrive un approccio generale nella creazione di un crawler (Sezione 4.2). Infine, si motivano le scelte di implementazione per la creazione dell'image-crawler (Sezione 4.4).

4.1 Tipologie di crawler

Come descritto nella Sezione 2.2, la mancanza di uno standard nell'ambito del web scraping non permette in maniera oggettiva di distinguere tipologie di crawler. Si è deciso pertanto di restringere il campo di ricerca alla tipologia *image-crawler*, cercando di determinare i possibili approcci per l'implementazione di tale software.

Quello che ne risulta è una distinzione in due macro categorie di *image-crawler*:

- Soluzione basata su sistemi distribuiti, nella quale si esegue la stessa ricerca su motori diversi e si aggrega il risultato.

- Soluzione basata sul sistema locale, la quale sfrutta un singolo motore di ricerca per reperire le immagini desiderate.

Una possibile implementazione della soluzione basata su sistemi distribuiti dai professori Shrinivasacharya [7] e Rajkumar [8] è descritta nei loro rispettivi articoli scientifici.

4.2 Descrizione del processo di creazione di un crawler

Nel momento in cui si decide di implementare un crawler, è fondamentale seguire alcuni accorgimenti in modo tale che il processo di creazione venga facilitato.

Come primo aspetto da sottolineare, è importante avere ben chiari i passaggi che il crawler deve eseguire per reperire determinate informazioni. Bisogna ricordare che il crawler non è in grado di prendere decisioni autonomamente, pertanto è fondamentale descrivere dettagliatamente ogni azione da eseguire. Solitamente, si eseguono i diversi passaggi manualmente una prima volta, annotandosi ogni azione (come per esempio una selezione di un elemento, l'attesa di un certo evento, il click del mouse e così via).

Dopo aver stipulato una lista contenente tutti i passi da compiere, bisogna capire come implementare le diverse azioni all'interno del crawler. Un esempio comune è la selezione di un insieme di elementi presenti nella pagina web. La soluzione più intuitiva e semplice è sfruttare un selettore CSS[1], che sia comune a tutti gli elementi desiderati.

Solitamente, questa funzionalità, insieme a molte altre, viene implementata all'interno di librerie o di framework già sviluppati, pertanto non c'è la necessità di implementare il crawler senza avere delle tecnologie di supporto.

4.3 Tecnologie per creare un crawler

Quello che segue è un elenco di tecnologie dedite alla creazione di crawler. Le principali sono le seguenti:

- Selenium WebDriver
- Scrapy
- Puppeteer
- Playwright

Per ognuna di queste soluzioni, segue una panoramica generale e una breve descrizione delle caratteristiche peculiari.

[1]Cascading Style Sheets: linguaggio usato per definire uno stile nelle pagine web.

4.3.1 Selenium WebDriver

Selenium WebDriver è un insieme di librerie, che permettono principalmente l'automazione di applicazioni web con lo scopo di eseguire test. È possibile estendere l'utilizzo a una vastità di campi, come per esempio l'esecuzione di specifici compiti. Offre un'ampia scelta di possibili linguaggi, tra cui C#, Java, JavaScrpit, Ruby e Python. Nel giugno del 2018, il progetto è divenuto una *"W3C* [2] *Reccomandation"*[10], ovvero è stato riconosciuto come linea guida nell'ambito di interazione con il browser. Grazie a questo titolo, i principali sviluppatori, i quali Mozilla, Google, Apple e Microsoft, hanno potuto supportare ed integrare al meglio il progetto con i loro rispettivi prodotti.

Si riporta di seguito lo schema rappresentante l'uso classico di Selenium WebDriver.

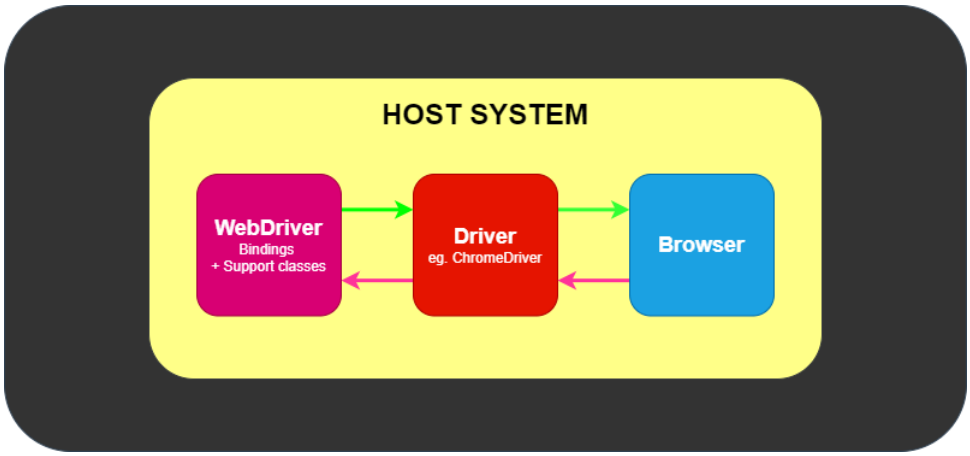


Figura 4.1: Architettura basata su Selenium WebDriver

Come viene mostrato in figura 4.1, le principali componenti sono:

- WebDriver: è il programma che sfrutta la libreria per automatizzare i compiti da eseguire nel browser.
- Driver: è un modulo specifico per il tipo di browser su cui si vuole eseguire l'automazione. Tale modulo deve essere scaricato manualmente e bisogna controllarne le versioni periodicamente: è importante avere la versione corretta prima di usare il programma stesso.
- Browser: è il browser che si vuole usare.

Si consulti il sito ufficiale [9] per maggiori dettagli sulla libreria.

[2] W3C: è una comunità internazionale con il compito di creare degli standard all'interno del web. Sito ufficiale: https://www.w3.org/

4.3.2 Scrapy

Citando dalla documentazione ufficiale, "Scrapy è un framework di web crawling e web scraping di alto livello, utilizzato per scansionare siti web ed estrarre dati strutturati dalle loro pagine. Può essere utilizzato per una vasta gamma di scopi, dal data mining all'esecuzione di test automatizzati."[3]

È un progetto open source scritto esclusivamente in Python, il quale offre una CLI molto robusta e ricca di funzionalità, come per esempio la creazione di progetti, l'esecuzione di un file singolarmente oppure di un benchmark[4].

Si riporta in seguito un semplice esempio di web scraper basato sul framework Scrapy.

```
1  import scrapy
2
3  class BlogSpider(scrapy.Spider):
4      name = 'blogspider'
5      start_urls = ['https://www.zyte.com/blog/']
6
7      def parse(self, response):
8          for title in response.css('.oxy-post-title'):
9              yield {'title': title.css('::text').get()}
10
11         for next_page in response.css('a.next'):
12             yield response.follow(next_page, self.parse)
```

Listing 4.1: Esempio di scraper usando Scrapy

Come si può notare nella Sezione di Codice 4.1, la leggibilità è molto semplice anche per chi per esempio non conosce bene la sintassi del linguaggio Python. Nell' esempio, lo scopo è quello di reperire tutti i titoli degli articoli presenti nel blog, cercando pagina per pagina fino ad arrivare all'ultima. Tutto questo è racchiuso in una dozzina di righe di codice.

Tramite il comando `"scrapy runspider myspider.py"` è possibile eseguire il file singolarmente senza dover creare un progetto.

4.3.3 Puppeeter

Puppeteer è una libreria basata su Node.js, che permette di interagire con il browser Chrome o Chromium attraverso i DevTools Protocols[5]. Il progetto è sviluppato e mantenuto dagli stessi ideatori dei DevTools, pertanto Puppeteer

[3]Citazione dal sito https://scrapy.org/doc.

[4]Misurare la qualità di un oggetto in funzione di un altro oggetto considerato standard. Traduzione dal sito https://dictionary.cambridge.org/.

[5]Sono dei programmi integrati all'interno del browser che facilitano lo sviluppo di pagine e di applicazioni web.

riceve aggiornamenti tanti quanti ne riceve Chrome. Questo favorisce ad avere una libreria sempre aggiornata e in continua crescita. Una delle peculiarità di questa libreria è la modalità definita *headless* impostata come di default, la quale permette di eseguire un'istanza del browser senza che l'utente possa effettivamente vederla sullo schermo.

La libreria espone delle API in modo che l'utilizzatore possa interagire al meglio con il browser e le sue diverse componenti. Si riporta uno schema raffigurante l'architettura interna di Puppeteer.

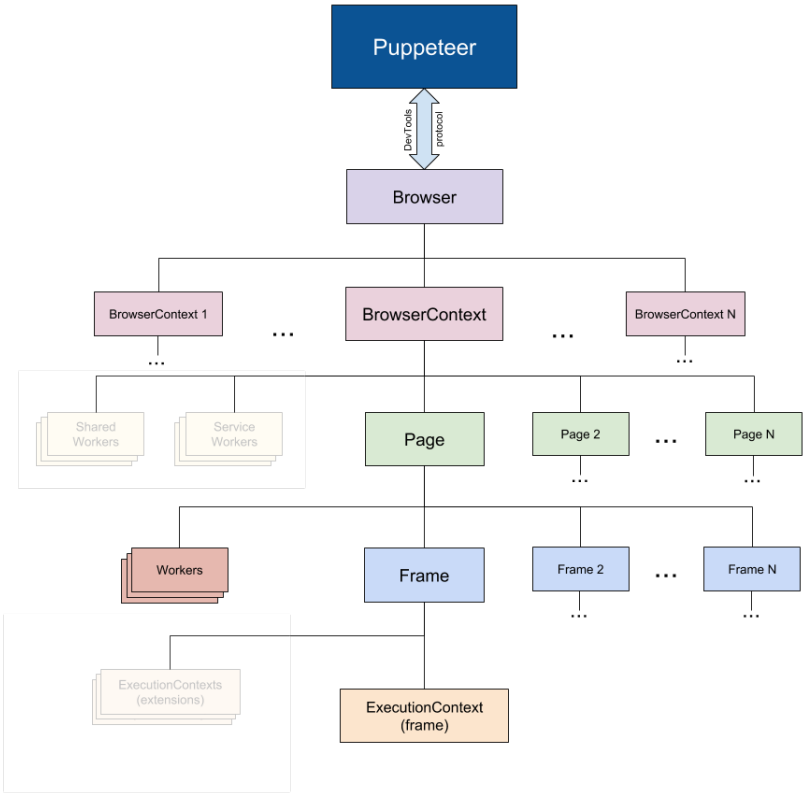


Figura 4.2: Architettura di Puppeteer

Come raffigurato nella Figura 4.2, la struttura interna di Puppeteer è rappresenta mediante un albero. Ogni livello rispecchia un componente che permette di interagire con il browser e i suoi contenuti.

In particolare:

- **BrowserContext**: rappresenta il contesto del browser, ovvero quale caratteristiche deve avere quell'istanza di browser.

- **Page**: rappresenta la pagina aperta all'interno dell'istanza di browser.
- **Frame**: rappresenta il contenuto che viene renderizzato all'interno della pagina, solitamente sono i **worker** che si occupano di questo compito.
- **ExecutionContext**: rappresenta il contesto nel quale vengono renderizzati componenti dinamici.

Avendo chiara l'architettura sulla quale si basa Pupeeteer, è più intuitivo comprendere la sintassi usata per interagire con la libreria.

Se ne riporta un esempio:

```
1  const puppeteer = require('puppeteer');
2
3  (async () => {
4    const browser = await puppeteer.launch();
5    const page = await browser.newPage();
6    await page.goto('https://example.com');
7    await page.screenshot({ path: 'example.png' });
8
9    await browser.close();
10 })();
```

Listing 4.2: Esempio introduttivo di Puppeteer

Come mostrato nella Sezione di Codice 4.2, fondamentalmente bisogna creare una istanza di browser e una pagina (righe 5, 6) e successivamente esplicitare la pagina che si vuole analizzare (riga 7). In questo esempio, si esegue uno screenshot della pagina e si chiude l'istanza del browser (righe 8-10).

È possibile consultare la documentazione della libreria facendo riferimento al sito ufficiale [11].

4.3.4 Playwright

Playwright è una libreria open source sviluppata da Microsoft per eseguire test end-to-end[6] nelle applicazioni web. La principale caratteristica di questa libreria è quella di sfruttare le API dei browser quali Chromium, Firefox e WebKit per eseguire azioni automaticamente. Molte funzionalità presenti in Playwright sono presenti in Puppeteer, infatti la libreria di Microsoft venne rilasciata nel 1 Febbraio del 2020 [12].

Si riporta un esempio di uso di Playwright, simile a quello introdotto nella Sezione 4.3.3.

[6]Test di isolamento: sono dei test che verificano il comportamento di un aspetto specifico.

```
const { webkit } = require('playwright');

(async () => {
  const browser = await webkit.launch();
  const page = await browser.newPage();
  await page.goto('http://whatsmyuseragent.org/');
  await page.screenshot({ path: `example.png` });
  await browser.close();
})();
```

Listing 4.3: Esempio introduttivo di Playwright

Quello che differisce dalla Sezione di Codice 4.3 alla Sezione di Codice 4.2 è nella prima riga, nella quale è possibile selezionare quale tipologia di API usare, e pertanto quale istanza di browser creare.

Per maggiori esempi sull'uso delle API offerte da Playwright, si riporta il sito ufficiale [13] contenente la documentazione.

4.4 Scelte di implementazione

Una volta determinate tutte le possibili tecniche e tecnologie per la creazione personalizzata di un crawler, si è dovuto decidere quale fosse la scelta migliore in funzione di quello che non prevedeva lo stato dell'arte attuale, si veda la Sezione 2.4, e gli obiettivi descritti, si veda la Sezione 3.2. Si ricorda pertanto che il programma deve poter essere eseguito da un qualsiasi calcolatore, anche quelli senza dispositivi di input/output connessi, e che debba essere auto contenuto.

Si è giunti alla conclusione di implementare una CLI che esegua in locale lo scraping delle immagini, tramite un image-crawler, basandosi su un solo motore di ricerca, sfruttando la modalità *headless* di Puppeteer.

Si è scelto come motore di ricerca Google Images poiché si è provato anche a usare altri motori come Microsoft Bing e Yandex per il reperimento di immagini, ma i test non sono andati a buon fine.

Infine, si è deciso di dare il nome *"gic"*, che è l'acronico di *Google Images Crawler*, al sistema.

Capitolo 5

Sviluppo del sistema

In questo capitolo si descrive l'architettura implementata all'interno dell'applicazione (Sezione 5.1), in aggiunta si descrivono tutte le problematiche riscontrate nella creazione del sistema (Sezione 5.2). Infine, si mostrano degli esempi di codice che rappresentano la logica della CLI (Sezione 5.3).

5.1 Architettura del sistema

Si riporta il diagramma rappresentate la struttura interna del sistema nella Figura 5.1.

La scelta più opportuna è stata suddividere il sistema in microcomponenti, suddivise in base alle tipologie di azioni da compiere. L'organizzazione è stata permessa grazie all'uso dei moduli[1] in JavaScript, creando anche un certo ordine all'interno della struttura dell'applicazione. Si è deciso pertanto che il sistema dovesse essere formato da 5 moduli principali:

- **Parser**: analizza la presenza di opzioni aggiuntive quando viene invocato il sistema.
- **Prompt**: interagisce con l'utilizzatore per avere conferme di poter eseguire certe azioni.
- **Crawler**: esegue l'istanza del browser ed estrapola le immagini desiderate.
- **Worker**: scarica le immagini reperite dal crawler.
- **Cli**: raggruppa tutte le funzionalità dei moduli all'interno di esso.

[1]Strumenti che permettono la suddivisone del codice. Per maggiori dettagli consultare https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules

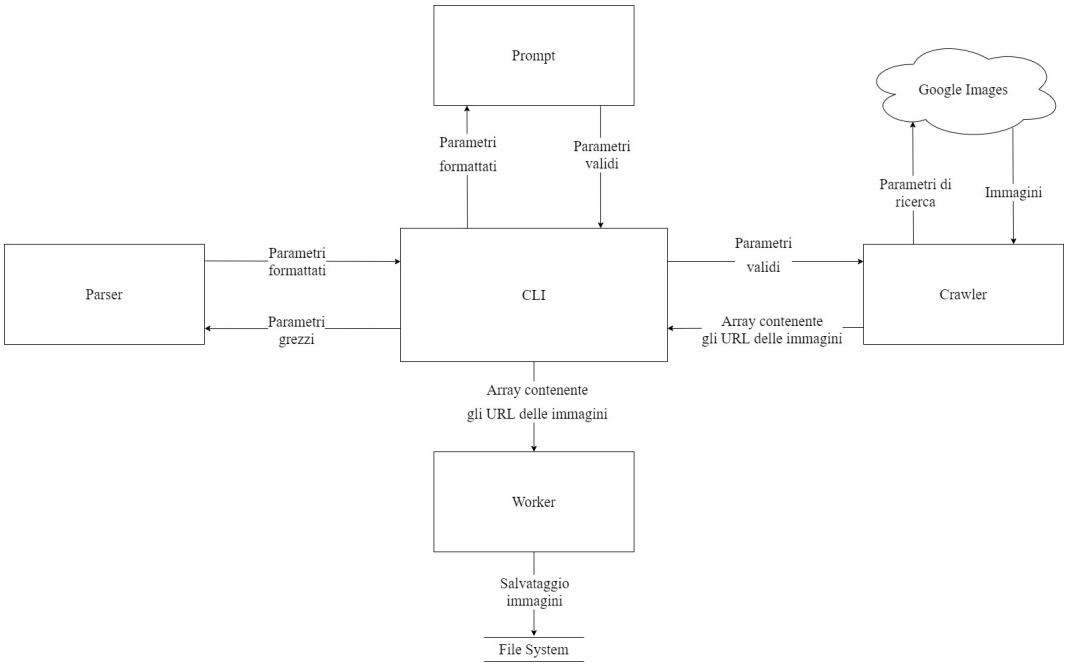


Figura 5.1: Diagramma dei componenti

5.2 Problematiche riscontrate nell'implementazione del sistema

Durante la creazione del sistema si sono riscontrate alcune problematiche che sono state affrontate secondo specifici criteri.

La prima è stata la presenza di doppioni: era fondamentale il fatto che non ci fossero elementi duplicati, altrimenti la rete neurale non sarebbe stata performante. Ovviamente, questa condizione non poteva essere rispettata a prescindere, pertanto c'era il possibile rischio che venissero reperite una o più immagini uguali. La soluzione è stata quella di analizzare ogni immagine sotto forma di byte, usare una funzione di hash per ottenere un identificativo univoco e assegnarlo come nome a tale immagine. In questa maniera, nel momento in cui veniva scaricata una immagine uguale a una di quelle già presenti su memoria locale, tale immagine sovrascriveva quella già presente.

Un'ulteriore complicanza era il calcolo di immagini presenti nella pagina web: quello che accade è che Google Images decide di renderizzare solo una parte non definita di immagini, lasciando tutti gli altri possibili risultati da caricare successivamente. Anche se si forza il browser a scorrere la pagina web in modo da caricare tutti gli elementi, Google Images esegue solo in parte la loro renderizzazione all'interno del browser. Questo comporta che il crawler

non è in grado di determinare se ha raggiunto il numero massimo di immagini presenti nella pagina web, oppure se Google Images non ha ancora caricato altre possibili immagini. In questo caso, si è deciso di implementare un numero di tentativi minimi prima di scaricare le immagini recuperate. In questo modo, si cerca di superare questo ostacolo solo in parte, poiché non è detto che nel numero di tentativi minimi Google Images renderizzi delle nuove immagini.

5.3 Esempi di codice

Si riportano i seguenti esempi di codice, che rappresentano le funzionalità principali del sistema.

5.3.1 Crawler

Il modulo crawler è il corpo centrale del programma. All'interno di esso è presente tutta la logica inerente all'interazione con il browser ed i suoi contenuti. Si riporta la funzione che esegue il reperimento degli URL delle immagini.

```
1  /**
2   * Main core of the module, count all the images and retrive
       their urls
3   * @param {String} keyword is the keyword to search
4   * @param {Number} number is the desire number of images
5   * @param {Number} attempt is the number of attempt before
       retrive urls
6   * @param {Boolean} isHeadless needs to set the headless mode
       or not
7   * @returns {Promise<Array[String]>} the array of all the urls
       refered to the images
8   */
9  async function retriveImages(keyword, number, attempt,
       isHeadless) {
10     const { page, browser } = await instance(isHeadless);
11     try {
12         await goToImagePage(page, keyword);
13         const arrayOfTumbnailImages = await
       retriveArrayOfTumbnailHandler(page, number, attempt);
14         const arrayOfUrlImages = await retriveArrayOfUrlImages(
       page, arrayOfTumbnailImages);
15         return arrayOfUrlImages;
16     } catch (error) {
17         throw new Error(error);
18     } finally {
19         await page.close();
```

```
20          await browser.close();
21      }
22  }
```

Listing 5.1: Funzione di reperimento degli URL delle immagini

Quello che accade nella Sezione di Codice 5.1 è che si crea l'istanza di browser con la pagina al suo interno (riga 10). Successivamente, si indirizza la pagina all'URL desiderato (ovvero quello di Google Images, riga 12) nel quale si esegue la ricerca delle immagini correlate alla parola chiave. Una volta raggiunto il sito, si esegue una prima analisi nella quale si cerca di reperire il maggior numero di immagini (riga 13) e successivamente per ognuna delle immagini si cerca di estrapolare il rispettivo URL (riga 14).

Si mostra ora nel dettaglio queste due fasi all'interno delle Sezioni di Codice 5.2 e 5.3.

```
1  /**
2   * Retrive all the possible images loaded from Google Images
3   * @param {Page} page is the istance of the page where to find
      images
4   * @param {Number} requestNumber is the desire number of images
5   * @param {Number} attempt is the number of attempt before
      retrive urls
6   * @returns {Promise{Array[ElementHandle]}} is the array
      containing the handler of the thmbnail
7   */
8  async function retriveArrayOfTumbnailHandler(page,
     requestNumber, attempt) {
9      let imageSelector = 'img[class="rg_i Q4LuWd"]';
10     let actualAttempt = 0;
11     let precedentNumberOfImages = 0;
12     let actualNumberOfImages = await page.$$eval(imageSelector,
       a => a.length);
13
14     while ((actualNumberOfImages < requestNumber) && (
     actualAttempt < attempt)) {
15         // Find if there is the button "show other results"
16         await isButtonVisible(page);
17         await page.keyboard.press('Space', { delay: 1000 });
18         precedentNumberOfImages = actualNumberOfImages;
19         actualNumberOfImages = await page.$$eval(imageSelector,
       a => a.length);
20         if (precedentNumberOfImages != actualNumberOfImages)
21             actualAttempt = 0;
22         else
23             actualAttempt++;
24     }
25
26     let arrayOfHandlerImages = await page.$$(imageSelector);
```

```
27      arrayOfHandlerImages = arrayOfHandlerImages.slice(0,
    requestNumber);
28      await page.keyboard.press('Home');
29
30      if (actualAttempt >= attempt) {
31          isEnoughAttempt = false;
32      }
33
34      return arrayOfHandlerImages;
35  }
```

Listing 5.2: Analisi numero di immagini

La logica che sta dietro al codice presente nella Sezione di Codice 5.2 è di cercare tutte le immagini che hanno in comune lo stesso attributo "classe". Per fare questo si utilizza un selettore CSS.

Quello che accade nel ciclo while presente dalla riga 14 a quella 24 è lo scorrimento della pagina web, in modo da forzare Google Images a renderizzare le immagini. Purtroppo, non sempre il numero di immagini renderizzate è uniforme: può accadere che vengano caricate 200 immagini in un secondo anche senza per forza aver fatto scorrere la pagina fino ad arrivare ad esse.

Questo aspetto non ha potuto permettere un approccio generale nel determinare il numero di immagini e, come descritto nella Sezione 5.2, si è dovuti implementare un numero massimo di tentativi da eseguire. Questo numero di tentativi rispecchia l'azione di continuo scorrimento della pagina, pur rimanendo invariato il numero di immagini. La ricerca di immagini si conclude quando il numero di immagini reperite è uguale a quello richiesto oppure si raggiunge il numero massimo di tentativi (righe 14-24).

La funzione ritorna un array contenete gli elementi rappresentati le immagini da scaricare.

```
1  /**
2   * Retrive all the url of the images from the given array
3   * @param {Page} page is the istance of the page where to find
      images
4   * @param {Array[ElementHandle]} arrayOfTumbnailHandler is the
      array of thumbnail where to strat extract urls
5   * @returns {Promise{Array[String]}} the array of url of the
      images
6   */
7  async function retriveArrayOfUrlImages(page,
      arrayOfTumbnailHandler) {
8       let arrayOfUrlImages = [];
9
10      for (const thubmnailHandler of arrayOfTumbnailHandler) {
11          await thubmnailHandler.click();
12          await page.waitForSelector('img[class=n3VNCb]')
```

```
13          let imageHandled = await page.$('img[class=n3VNCb]');
14          let srcHandler = await imageHandled.getProperty('src');
15          let actualSrcImage = await srcHandler.jsonValue();
16          arrayOfUrlImages.push(actualSrcImage);
17      }
18
19      return arrayOfUrlImages;
20  }
```

Listing 5.3: Funzione di reperimento degli URL delle immagini

La funzione descritta nella Sezione di Codice 5.3 permette di estrapolare gli URL derivanti dalle immagini analizzate. Si noti come prima di accedere all'URL effettivo, bisogna cliccare sull'immagine (riga 11) e aspettare che si carichi l'immagine originale (righe 12-13). Una volta determinata la vera immagine da scaricare, si estrapola il corrispondente URL tramite l'attributo "src" (righe 13-15). Tutte queste azioni si ripetono per ogni elemento presente nell'array risultante dalla funzione descritta nella Sezione di Codice 5.2.

Si restituisce un array contenente tutti gli URL delle immagini reperite.

5.3.2 Worker

Il modulo worker permette di scaricare le immagine partendo dal rispettivo URL ed evita che ci siano doppioni a fine esecuzione.

Si riporta la funzione principale all'interno della Sezione di Codice 5.4.

```
1  /**
2   * Download the image given in the url with the specific path
3   * @param {String} url is the url of the image ti be downloaded
4   * @param {String} path is the path to save the file (it is
      created if dosen't exist
5   */
6  async function fetchImage(url, path) {
7      fs.mkdir(path, { recursive: true }, (err) => {
8          if (err)
9              throw new Error(err);
10     });
11
12     const response = await fetch(url);
13     const buffer = await response.buffer();
14     const sha = crypto.createHash('sha1').update(buffer).digest
       ('hex');
15     const filename = path.concat('/', sha, '.png');
16     // Open the file and start write inside it
17     async.waterfall([
18         (callback) => {
19             fs.open(filename, 'a', callback);
20         },
```

```
21          (fd, callback) => {
22              fs.write(fd, buffer, 0, buffer.length, null,
    callback);
23          }
24      ], (err) => {
25          if (err)
26              throw new Error(err);
27      });
28  }
```

Listing 5.4: Scaricamento di immagini tramite l'URL

Quello che avviene all'interno della Sezione di Codice 5.4 è la creazione della cartella dove salvare le immagini (righe 1-3). Tramite il parametro *"recursive": true*, è possibile generare anche delle sottocartelle e pertanto un utilizzatore ha piena libertà di scelta su dove memorizzare le immagini. Successivamente, si esegue una richiesta HTTP all'URL e si memorizza la risposta in un *buffer*[2] (righe 12-13). Essendo che il buffer rappresenta i dati in conversione binaria, è possibile usare una funzione di hash (come per esempio sha1) per creare un identificativo che fa riferimento al contenuto esplicito dell'immagine (riga 14). Assegnando tale identificativo come nome del file (riga 15), si ha la certezza che se una immagine ha un URL diverso, ma contenuto uguale a una immagine già scaricata, si sovrascriverà quella precedente in modo da avere una singola copia. Infine, si esegue in cascata con il metodo *waterfall*[3] l'apertura del file e la scritta dei dati al suo interno (righe 17-27).

5.3.3 Cli

Il modulo cli rappresenta tutta la logica che permette l'esecuzione del sistema. Infatti, in essa sono racchiusi tutti gli altri moduli in modo che fosse il modulo stesso a gestire le diverse componenti dell'architettura.

Si riporta il codice relativo al modulo cli nella Sezione di Codice 5.5.

```
1 const spinner = ora();
2
3 // Handle 'CTRL + C' keybind
4 process.on('SIGINT', () => {
5     spinner.warn('Caught interrupt signal. Closing the CLI');
6     process.exit(0);
7 });
8
```

[2]Strutta dati presente in Node.js per manipolare dati binari. Maggiori dettagli sono presenti nel sito https://nodejs.org/api/buffer.html

[3]Costrutto per eseguire funzione callback consecutive. Argomento ampiamente discusso nel corso "Tecnologie Web per il Cloud".

```
9  export async function cli(args) {
10     // Parsing the flags
11     let options = parseArguments(args);
12     if (options.help) {
13         printHelp();
14         process.exit(0);
15     } else if (options.version) {
16         printVersion();
17         process.exit(0);
18     }
19
20     // Control if there is the required flag
21     if (!options.keyword)
22         // Asking the use to input the keyword
23         options.keyword = await promptRequiredOption();
24
25     spinner.start('Retriving images (This can take a while...)'
       );
26     // Retriving the number of images
27     let arrayOfUrlImages = [];
28     try {
29         arrayOfUrlImages = await retriveImages(options.keyword,
       options.number, options.attempt, options.debug);
30     } catch (error) {
31         spinner.fail(error.message);
32         process.exit(1);
33     }
34
35     if (!isEnoughAttempt) {
36         spinner.warn('Reached max number of attempts!');
37     } else
38         spinner.succeed('Done!');
39
40     // Ask the user if the number is good enough
41     if (options.number === Infinity || arrayOfUrlImages.length
       < options.number)
42         if (!await promptForNumberOfImages(arrayOfUrlImages.
       length)) {
43             spinner.warn('Closing the CLI');
44             process.exit(0);
45         }
46
47     spinner.start('Starting the download');
48     // Start downloading the images
49     const path = options.path;
50     for (const url of arrayOfUrlImages) {
51         try {
52             await fetchImage(url, path);
53         } catch (error) {
```

```
54                 spinner.fail(error.message);
55                 process.exit(1);
56             }
57         }
58         spinner.succeed('Download Complete!')
59 }
```

Listing 5.5: Esecuzione completa della Cli

Come si può evincere, il codice rappresentato nella Sezione di Codice 5.5 è autoesplicativo poiché si riesce a determinare chiaramente i diversi passaggi che avvengono.

Come primo passaggio che avviene all'interno della funzione *cli* è l'analisi dei parametri che vengono inseriti quando si richiama il programma riga (11). Si controlla se l'utente abbia richiesto di visionare la sezione di aiuto (tramite il flag *-h*) oppure la sezione di verifica versione (tramite il flag *-v*). In questi casi, bisogna stampare in output la risposta corrispondente e chiudere l'esecuzione del sistema (righe 12-18).

Successivamente, si controlla che l'utilizzatore abbia immesso la parola chiave necessaria per la ricerca di immagini (riga 22). Se ciò non avviene, il sistema entra in modalità interattiva e chiede all'utente l'inserimento di una parola chiave (riga 23).

Eseguito il controllo sulla parola chiave, si invoca il comando per l'esecuzione dell'image-crawler, il quale eseguirà il reperimento degli URL delle immagine analizzate (per maggiori dettagli si consulti la Sezione 5.3.1). Il blocco *try...catch* serve per gestire un qualsiasi errore derivante dall'uso del browser, in modo da avere un controllo sul comportamento del programma (righe 27-33).

Un'ulteriore fase fondamentale è la verifica del numero di immagini reperite (righe 41-45). Si richiede all'utente la conferma quando non viene esplicitato un numero di immagini oppure il numero di immagini recuperate è minore rispetto a quello richiesto. Nel momento in cui l'utente non acconsenta lo scaricamento, la cli viene chiusa.

Infine, se l'utente desidera che le immagini vengano scaricate, viene invocato il modulo worker affinché esegua lo scaricamento delle immagini (righe 49-57).

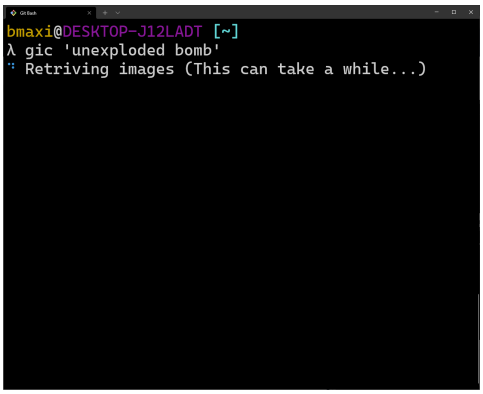
Si sottolinea come venga gestito il caso nel quale l'utilizzatore prema il comando *CTRL+C*, in modo da fermare l'esecuzione della applicazione in maniera controllata (righe 3-7).

Le restanti parti di codice servono per dare informazioni all'utente su cosa stia accadendo durante l'esecuzione del sistema, in particolare tutti i comandi *spinner.comando()* sono a scopo decorativo per rendere la cli esteticamente gradevole.

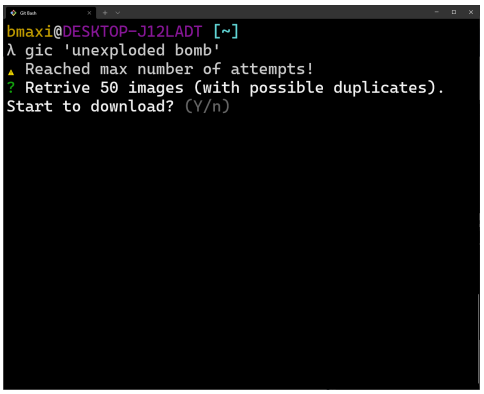
Capitolo 6

Esempio di Utilizzo

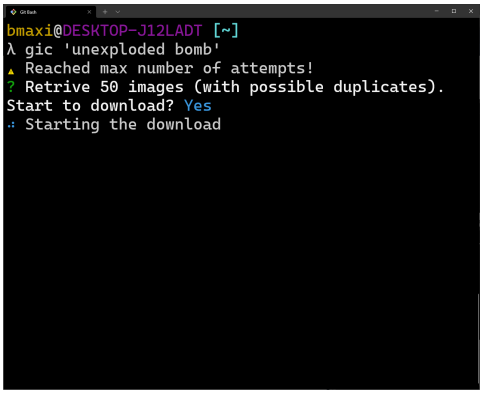
Si riportano diversi esempi di utilizzo della Cli implementata.



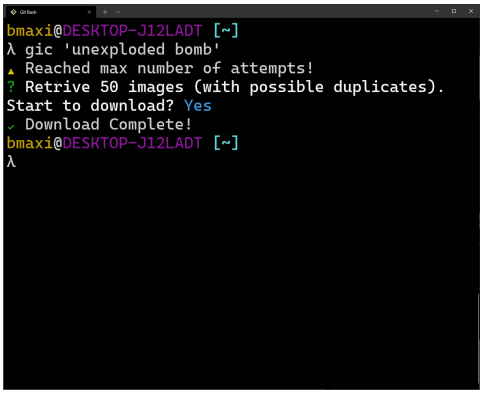
(a) Prima fase



(b) Seconda fase



(c) Terza fase



(d) Quarta fase

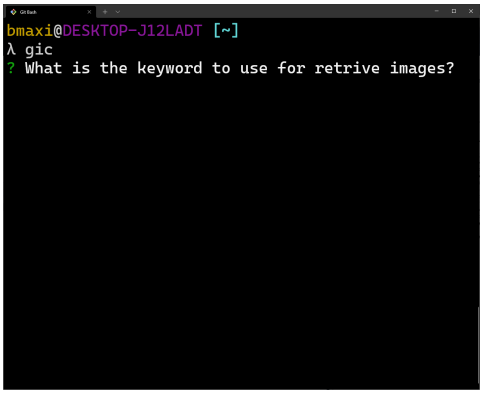
Figura 6.1: Caso d'uso semplice

Quello che viene mostrato in Figura 6.1 è l'utilizzo più comune dell'applicazione.

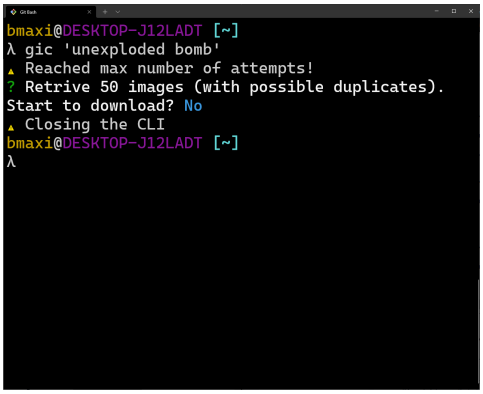
L'utilizzatore richiama il programma tramite il comando *gic* e immette la parola chiave. Nel momento in cui non viene esplicitato tale parametro, il programma chiede all'utente di digitarlo come mostrato in Figura 6.2a. Se non vengono aggiunti altri parametri, il software inizia a cercare immagini inerenti alla parola chiave.

Come spiegato nella Sezione 5.2, il crawler ha un numero di tentativi per trovare il maggior numero possibile di immagini (inizialmente tale valore è settato a 7). Come mostrato nella Figura 6.1b, il software ha raggiunto il numero massimo di tentativi e ha riportato le 50 immagini iniziali. Il programma attende poi la conferma di inizio download delle immagini (Figura 6.1c), questo accadde o perché il numero di tentativi è stato superato (come nell'esempio descritto nella Figura 6.2 o perché il numero di immagini richiesto é maggiore rispetto a quello trovato.

Se l'utente acconsente lo scaricamento delle immagini, il programma lo esegue automaticamente all'intero di una cartella denominata *gic* all'interno della cartella home dell'utente. Nel momento in cui l'utente non sia soddisfatto del numero reperito, può non autorizzare il download delle immagini come riportato in Figura 6.2b.



(a) Richiesta della parola chiave



(b) Fermare l'esecuzione

Figura 6.2: Casi speciali

Tramite il paramento *-h*, è possibile avere una lista di tutti i parametri che si possono usare richiamando la CLI. Si riporta un'immagine con l'uso e la descrizione di ogni parametro nella Figura 6.3.

Infine, nella Figura 6.4 si rappresenta l'output finale dell'esecuzione della cli.

Figura 6.3: Sezione di aiuto nell'uso della CLI

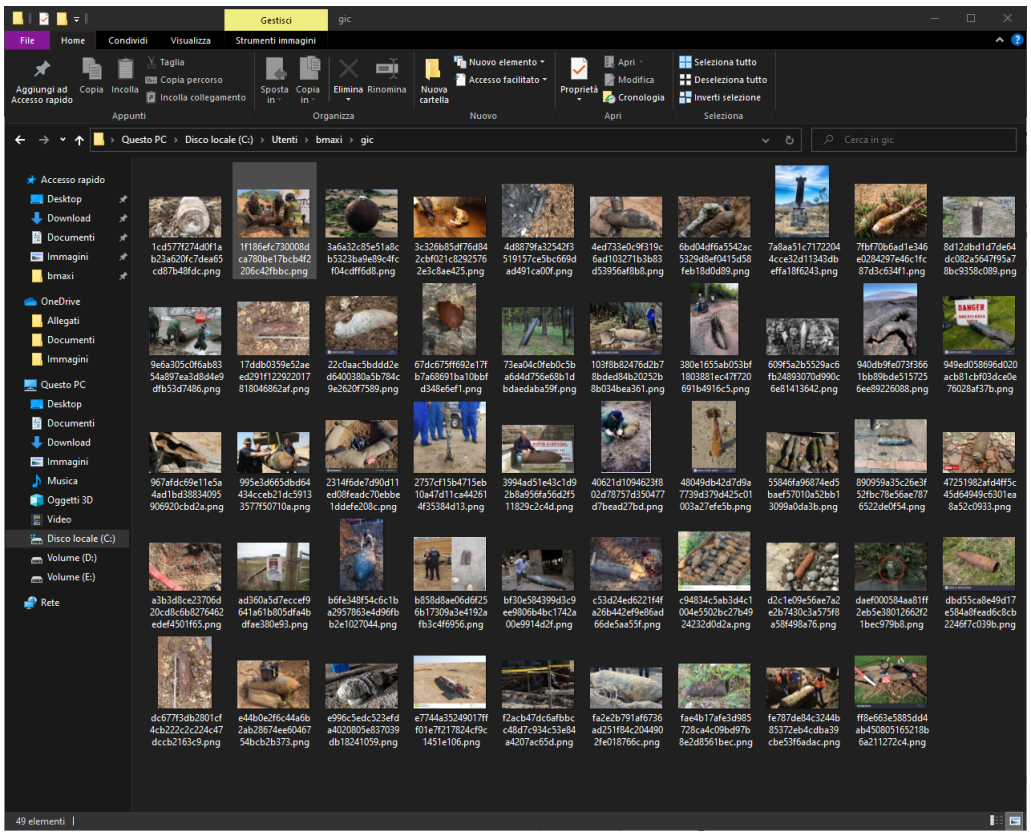


Figura 6.4: Cartella risultante dopo l'uso della cli

Capitolo 7

Conclusioni

In questo capitolo si riportano alcuni pensieri generali sull'intero progetto (Sezione 7.1) e dei possibili sviluppi futuri per l'applicazione (Sezione 7.2). In conclusione, si riportano i ringraziamenti (Sezione 7.3).

7.1 Considerazioni generali

Il risultato è abbastanza soddisfacente. Il sistema riesce ad interagire al meglio con il browser e l'interazione con esso è intuitiva e chiara. Rimangono ancora dei dettagli per i quali il prodotto non è performante al massimo. La colpa di questo risiede nel fatto che Google Images implementa dei metodi in modo che non si possa automatizzare la ricerca di immagini tramite un crawler. Per esempio, quello che accade è che alcune volte fa scomparire delle immagini dalla pagina senza un motivo valido: questo comporta il fatto che il crawler, avendo ancora un riferimento a tale immagine, non possa più interagire con esse e si genera un errore. Anche implementando delle azioni che un utente normale esegue su Google Images, come per esempio delle pause o dei rallentamenti nello scorrimento, si genera comunque quel tipo di errore. In conclusione, il progetto è stata una ottima occasione per ampliare le conoscenze sulle tecnologie di web-scraping e di approfondire molti argomenti studiati durante il corso di laurea. La speranza è che un giorno possa tornare utile per altri scopi e soprattutto si possa migliorare con strumenti e tecnologie future.

7.2 Sviluppi futuri

Come descritto nelle Sezioni 5.2 e 7.1, l'applicazione presenta delle possibili migliorie. Come primo aspetto, cercare di rendere controllato il possibile comportamento di Google Images sfruttando diverse metodologie.

In secondo luogo, provare a usare un diverso motore di ricerca come per esempio Pixaby [14] o Unsplash [15], i quali magari permettono l'uso di API già pronte o sono più aperti nella possibilità di reperire immagini ulteriormente.

Un'ultima voce che merita essere sottolineata è quella di implementare una architettura a sistema distribuito come quella descritta nella Sezione 4.1. Nel dettaglio, se si creasse un sistema distribuito che eseguisse degli image-crawler parallelamente sfruttando diversi motori di ricerca e successivamente aggregando i risultati, l'intero carico di lavoro sarebbe dato in gestione al sistema e non al dispositivo dell'utilizzatore.

7.3 Ringraziamenti

Credo sia opportuno e corretto dedicare questa sezione a chi mi ha permesso di aver raggiunto questo obiettivo.

Ci tengo a ringraziare il professore Ivan Scagnetto, per avermi fatto avvicinare al progetto e avermi supportato su ogni scelta implementativa. In aggiunta, ci tengo a ringraziarlo per il tempo speso a rispondere a tutte le mie incessanti email e per la gentilezza nell'essermi stato vicino in questi momenti snervanti e complicati.

Voglio ringraziare i miei, per avermi permesso di iscrivermi a questo corso di laurea e per aver dato il loro saggio consiglio nel momento del bisogno.

Voglio ringraziare Cristina, per tutte le volte in cui non pensavo di riuscire a farcela o pensavo di gettare la spugna, era sempre lì al mio fianco facendo il tifo per me.

Infine ci tengo a ringraziare i docenti che ho incontrato in questo percorso, ma soprattutto le amicizie che ho potuto stringere, per le risate, per le ore infinite a studiare assieme, per i riassunti scritti, per le serate e anche per le semplici chiamate.

Bibliografia

[1] Definizione di motore di ricerca: https://en.wikipedia.org/wiki/Search_engine

[2] Differenza tra crawler e scraper: https://smartproxy.com/what-is-web-scraping/crawling-vs-scraping

[3] Definizione completa di bot: https://www.kaspersky.com/resource-center/definitions/what-are-bots

[4] Sito ufficiale di Octoparse: https://www.octoparse.com/

[5] Sito ufficiale di WebScraper: https://webscraper.io/

[6] Sito ufficiale di WebScrapingApi: https://www.webscrapingapi.com/

[7] P. Shrinivasacharya, M. V. Sudhamani, *An image crawler for content based image retrieval system.* IJRET **02** (Nov-2013).

[8] R. Rajkumar, M. V. Sudhamani, *Crawler for image acquisition from World Wide Web.* IJRET (May-2017)

[9] Documentazione di Selenium WebDriver: https://www.selenium.dev/documentation/webdriver/

[10] Articolo ufficiale di nomina da parte del W3C: https://www.w3.org/TR/webdriver1/

[11] Documentazione di Puppeteer: https://pptr.dev/

[12] Prima versione rilasciata di Playwright: https://github.com/microsoft/playwright/releases/tag/v0.10.0

[13] Documentazione di Playwright: https://playwright.dev/

[14] Sito ufficiale di Pixaby: https://pixabay.com/

[15] Sito ufficiale di Unslaph: https://unsplash.com/