

# CAPITOLO 3

---

## Analisi dei requisiti

---

*Il capitolo dedicato all’analisi dei requisiti si pone l’obiettivo di definire e strutturare le specifiche necessarie per lo sviluppo dello scraper progettato per Threads. La suddivisione del capitolo in sezioni consente di analizzare in dettaglio gli aspetti legati agli obiettivi, ai requisiti funzionali e non funzionali, e alle specifiche tecniche.*

*Nella sezione iniziale si descrivono gli obiettivi generali e il contesto operativo dello scraper, con un focus sui casi d’uso che rappresentano le principali interazioni tra il sistema e gli utenti. Successivamente, si passa ad un’analisi funzionale dei requisiti, distinguendo tra requisiti funzionali, che definiscono ciò che il sistema deve fare, e requisiti non funzionali, che stabiliscono vincoli e caratteristiche aggiuntive del sistema. Viene, inoltre, introdotta una panoramica sugli ulteriori sviluppi che potrebbero essere implementati in futuro per migliorare o ampliare le funzionalità dello scraper.*

*La parte tecnica del capitolo approfondisce le tecnologie utilizzate, tra cui il linguaggio di programmazione Python, il framework Selenium per la simulazione delle interazioni web e la parte del salvataggio dei dati basata su Docker e MySQL. Si includono, anche, dettagli sulle dipendenze necessarie per garantire il corretto funzionamento del sistema, fornendo una base solida per la sua implementazione.*

*Per la parte di Ingegneria del Software si farà riferimento ad Arlow e Neustadt [2009].*

### 3.1 Obiettivi e Contesto dello scraper

In questo capitolo sull’analisi dei requisiti si seguiranno le direttive esposte in Figura 3.1, come riportato nel libro *UML 2 and the Unified Process*, Arlow e Neustadt [2009].

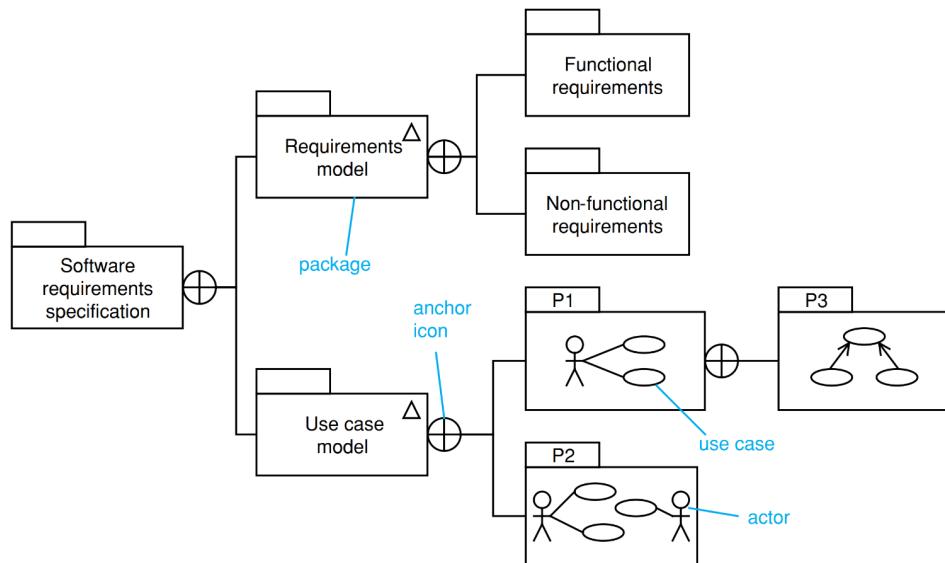
Il modello dei requisiti comprende requisiti funzionali (che specificano ciò che il sistema deve fare) e requisiti non funzionali (che esprimono vincoli non funzionali sul sistema).

Il modello dei casi d’uso include: i casi d’uso stessi, i quali specificano le funzionalità del sistema; gli attori, ovvero i ruoli esterni che interagiscono direttamente con il sistema; le relazioni tra casi d’uso ed attori.

I requisiti funzionali e non funzionali vengono presentati nel prossimo paragrafo, mentre in questo si analizzano gli obiettivi in termini generali, il contesto di sviluppo dello scraper e gli stakeholder coinvolti.

Tra le competenze dell’Ingegneria del Software rientra la definizione dei requisiti, ovvero descrivere le attività legate all’individuazione, alla documentazione e alla gestione di un insieme di requisiti per un sistema software. L’obiettivo è comprendere ciò di cui gli stakeholder hanno bisogno che il sistema faccia.

I requisiti incompleti e la mancanza di coinvolgimento degli stakeholder sono le principali cause di fallimento dei progetti. Entrambi questi problemi rappresentano carenze nell’implementazione dei requisiti.



**Figura 3.1:** Approccio all'analisi dei requisiti

Poiché il sistema software finale si basa su un insieme di requisiti, un'efficace ingegneria dei requisiti costituisce un fattore critico per il successo nei progetti di sviluppo software. Innanzitutto si deve descrivere l'obiettivo del sistema da implementare.

In termini generali, quello che si vuole ottenere è un sistema di scraping in grado di estrarre informazioni dagli utenti della piattaforma social Threads. L'obiettivo finale è quello di fornire, in modo strutturato, dati e informazioni sulle pubblicazioni degli utenti.

Essendo il progetto sviluppato interamente in autonomia per il tirocinio, gli stakeholder coincidono con:

- *autore del progetto*, responsabile dello sviluppo dello scraper;
- *tutor accademico*, figura di supporto che supervisiona il progetto, responsabile della definizione dei requisiti, con l'obiettivo di valutarne il risultato finale.

Non essendoci committenti esterni, il progetto si concentra principalmente sull'autovalutazione dell'efficacia degli strumenti sviluppati e sulla loro utilità per eventuali futuri utenti o collaboratori. I risultati del lavoro sono pensati per essere scalabili e riutilizzabili, offrendo un potenziale valore per:

- *ricercatori o studenti*, che necessitano di raccogliere dati web in modo strutturato per analisi;
- *enti accademici*, che possono sfruttare gli strumenti sviluppati per progetti di ricerca o didattici;
- *professionisti del settore*, che potrebbero utilizzare gli scraper come base per applicazioni più complesse.

Dunque, lo sviluppo degli scraper si inserisce nel contesto di un progetto di tirocinio volto a esplorare le tecniche di web scraping e di analisi dei dati per finalità accademiche.

Il progetto si colloca in un ambito multidisciplinare che combina aspetti di programmazione, Ingegneria del Software, Big Data e Data Analysis, con particolare attenzione alla progettazione e all'implementazione di strumenti software per il web scraping. L'adozione

di tecnologie moderne e le *best practice* per lo sviluppo di scraper robusti contribuiscono a migliorare la qualità del prodotto realizzato.

L'utilizzo dei dati raccolti per l'archiviazione costituisce l'aspetto realizzativo finale, utile per la successiva fase di analisi, rivolta ai destinatari del progetto, i quali si occuperanno di individuare pattern, relazioni, tendenze e ideologie dietro i numerosi dati raccolti. In particolare, si vogliono:

- identificare tendenze;
- confrontare offerte o strategie competitive;
- fornire informazioni aggiornate e dettagliate su argomenti specifici, come prezzi o recensioni di prodotti;
- aiutare aziende, ricercatori o individui a prendere decisioni informate basate sui dati.

Il contesto è principalmente accademico, ma le informazioni acquisite possono trovare applicazione pratica in ambiti professionali, come l'e-commerce, la ricerca di mercato o l'analisi competitiva. I risultati ottenuti possono fornire un punto di partenza interessante per analisi socio-economiche sul paradigma concettuale attualmente in vigore. L'applicazione e i risvolti di questi dati, a seguito di un'importante operazione di pulizia, hanno un grande potenziale informativo.

### 3.1.1 Casi d'uso

In questo paragrafo si iniziano a mostrare in termini generali i requisiti; successivamente, nell'analisi funzionale si entrerà più nel dettaglio per un'introspezione più approfondita dei requisiti, suddividendoli in funzionali e non funzionali. In questa parte, invece, si parte dai casi d'uso, con l'obiettivo di avere una prima esposizione dei requisiti, al fine di comprendere cosa deve fare il sistema in termini generali.

Rispetto all'analisi funzionale, i casi d'uso rappresentano un modo complementare di documentare i requisiti del sistema. I due sistemi formano insieme un connubio che riesce a cogliere tutte le diverse sfumature implementative che lo scraper deve soddisfare.

Nella modellazione dei casi d'uso si procede gradualmente fissando alcuni componenti fondamentali, ovvero

- *il box che contiene il modello dei casi d'uso*, che definisce i limiti entro cui il sistema opera;
- *gli attori*, cioè gli attuatori che interagiscono direttamente con il sistema;
- *i casi d'uso stessi*, cioè ciò che gli attori possono fare con il sistema;
- *le relazioni tra gli attori e i casi d'uso*, che sono rappresentate dalle linee continue che collegano casi d'uso ad attori.

Gli attori sono esterni al sistema e sono solitamente rappresentati con un omino stilizzato. Al fine di identificare gli attori è necessario domandarsi che ruolo occupano nell'ecosistema e nell'interazione con il sistema stesso.

Un caso d'uso rappresenta un qualcosa che l'attore vuole il sistema faccia. Ci sono due importanti condizioni che i casi d'uso devono avere: sono sempre inizializzati da un attore e sono sempre scritti dal punto di vista dell'attore stesso. La sintassi UML prevede che i casi d'uso siano scritti all'interno di ovali.

Un'ultima considerazione prima di passare all'esposizione dei casi d'uso concerne la realtà implementativa dello scraper su Threads. Si precisa che gli scraper su Threads in

realità sono tre, ma si preferisce riferirsi ad essi come un insieme, appellandosi al singolare, anche se in realtà si articolano in tre componenti. Nella fattispecie gli scraper sono:

- *Home Scraper*;
- *Profile Scraper*;
- *Post Scraper*.

La decisione implementativa di suddividere il codice in più crawler è stata presa anche in vista dei requisiti funzionali esposti successivamente. A questi tre scraper si accompagna uno script che si occupa di interagire con i dati raccolti da questi e scaricarne il contenuto. Si entrerà nel dettaglio più avanti nel capitolo.

Per potersi appropciare alla realizzazione dei casi d'uso è importante analizzare cosa effettivamente facciano questi scraper. La spiegazione dei risultati di questi tre scraper va ad anticipare, in maniera sommaria, il risultato applicativo dell'implementazione fattuale dei requisiti funzionali e non funzionali.

Si precisa che tutti e tre gli scraper vanno lanciati in modo sincrono e che il coordinamento tra di essi è gestito a livello di codice. Entrando nei particolari di ogni crawler si ha una panoramica più chiara dei comportamenti e delle aspettative sui risultati finali.

Quindi, si può partire dallo scraper principale, quello cardine da cui tutti gli altri dipendono, ovvero lo *Home Scraper*. Questo si occupa di scorrere fino alla fine la home page di Threads e di raccogliere le informazioni su ogni post. Tali informazioni sono:

- link del post;
- username autore del post;
- link al profilo dello user.

Successivamente lo scraper salva questi dati all'interno di un database (MySQL) nella tabella `posts`. Gli altri due scraper, la cui iniziativa dipende dalle righe presenti in tale tabella, svolgono due compiti fondamentali, ovvero l'analisi del profilo utente e l'analisi di ogni singolo post.

Nel primo caso entra in azione il *Profile Scraper* il quale ha il compito di recuperare ogni singola riga all'interno della tabella `posts` e di continuare con lo scraping. In particolare, per ogni username presente nella tabella, lo scraper recupera informazioni sul profilo utente quali:

- username;
- immagine del profilo;
- biografia;
- numero di seguaci.

Queste informazioni, assieme al link associato al profilo in esame, vengono salvate in un'ulteriore tabella denominata `post_users`.

Successivamente, lo scraper scorre tutti i post del profilo utente con le stesse modalità dello *Home Scraper*, cioè salvando, per ogni post:

- il link del post;
- lo username dell'autore del post;

- il link al profilo dello user.

L'ultimo, e probabilmente il più importante, crawler è il *Post Scraper*. Questo si occupa di raccogliere i dati inerenti ogni post in modo tale da costruire il substrato su cui poi effettuare analisi per l'estrapolazione di informazioni dai dati.

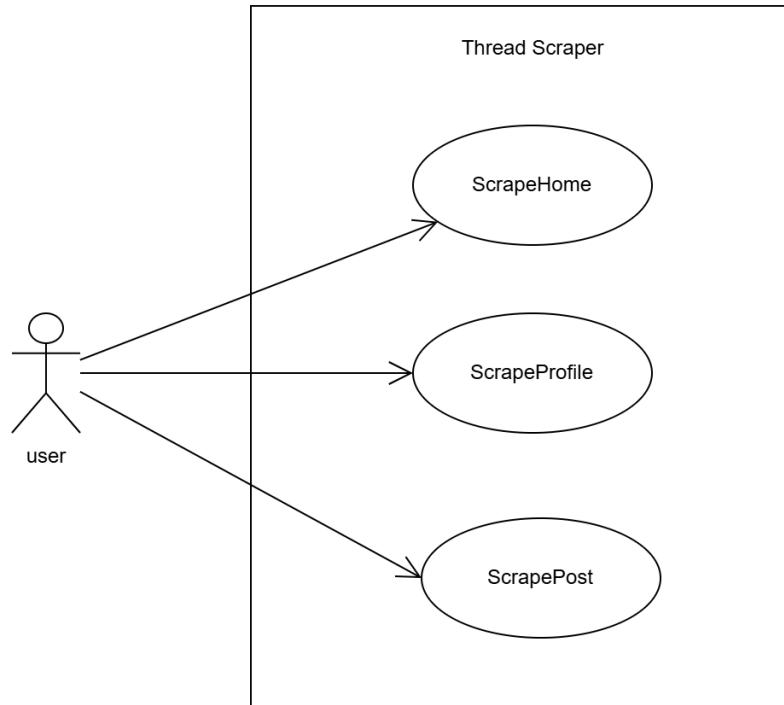
Il crawler in questione recupera il link associato ad ogni post dalla tabella `posts` e ne visualizza il contenuto. Per ogni post vengono recuperate informazioni fondamentali come:

- post id;
- la descrizione del post;
- eventuali immagini nel post;
- eventuali video nel post;
- tutti i commenti testuali sotto il post.

Queste informazioni, che nel caso dei contenuti multimediali sono sotto formato di link, vengono salvate in modo permanente in un'ulteriore tabella chiamata `posts_info`.

In ultima analisi, come anticipato, interviene uno script che si occupa di scaricare effettivamente le immagini ed i video presenti nella tabella `posts_info`. Questa attività deve essere svolta contemporaneamente all'esecuzione degli scraper, perché Threads, dopo poche ore o giorni, cambia gli indirizzi delle immagini e dei video rendendo inutilizzabili i link salvati in `posts_info`.

A questo punto, in Figura 3.2, si mostrano i casi d'uso. Il glossario dei termini di base viene omesso per semplicità.



**Figura 3.2:** Diagramma dei casi d'uso

Una volta definiti i casi d'uso, è necessario analizzare nel dettaglio ogni caso d'uso presentato, cioè si deve formalizzare quanto esposto a parole per ogni scraper.

In realtà, non vi è uno standard UML universalmente riconosciuto per specificare i casi d'uso; per questa ragione si definiscono delle proprietà che deve presentare ogni caso d'uso analizzato. Tra queste si elencano:

- ID;
- nome del caso d'uso;
- descrizione del caso d'uso;
- attori coinvolti;
- precondizioni, cioè lo stato del sistema prima che il caso d'uso si attivi;
- *main flow*, ovvero i passaggi che caratterizzano il caso d'uso, considerando, per semplicità, solamente il caso ideale;
- postcondizioni, cioè lo stato del sistema dopo che il caso d'uso si è attivato.

I casi d'uso specifici sono illustrati per tutti e tre gli scraper, rispettivamente, nelle Figure 3.3, 3.4 e 3.5.

Caso d'uso: ScrapeHome
ID: 1
<p>Breve descrizione. Lo scraper scorre la Home di Threads e recupera, per ogni post, il link, lo user associato e il link al profilo dello user.</p>
Attore: user
Precondizione: container MySQL attivo
<p>Main flow:</p> <pre> 1 While True   2 inizializza web driver   3 accetta i cookie   4 if not "posts":     4.1 crea la tabella "posts"   5 while scroll:     5.1 recupera post link, username e user profile link per ogni post     5.2 salva i dati nella tabella "posts" del db </pre>
Postcondizione: creazione della tabella "posts" e riempimento della stessa

**Figura 3.3:** Caso d'uso relativo a *Home scraper*

## 3.2 Analisi funzionale

Un requisito può essere definito essenzialmente come una specifica di ciò che deve essere implementato. In particolare, si distinguono due tipi di requisiti, ovvero i requisiti funzionali e i requisiti non funzionali.

Caso d'uso: ScrapeProfile
ID: 2
<p>Breve descrizione.            Lo scraper, per ogni user, recupera descrizione, link al profilo, username, followers e immagine del profilo.            Salva questi dati nella tabella "post_users".            Per ogni profilo recupera tutti i post con le stesse modalità del caso d'uso Scrape Home,            ovvero recuperando, per ogni post, il link associato, lo username e il link al profilo.</p>
Attore: user
Precondizione: la tabella "posts" esiste ed ha valori al proprio interno.
<p>Main flow:</p> <ol style="list-style-type: none"> <li>1 if "posts":</li> <li>2 if not "post_users":</li> <li>    2.1 crea la tabella "post_users"</li> <li>3 inizializza web driver</li> <li>4 while True:</li> <li>    5 recupera il link di un utente dalla tabella "posts"</li> <li>    6 apre il profilo</li> <li>    7 if post id = 1:</li> <li>        7.1 accetta i cookie</li> <li>    8 recupera bio, followers e img del profilo</li> <li>    9 salva questi dati + username e link user in "post_users"</li> <li>10 while scroll:</li> <li>        10.1 recupera il link di ogni post</li> <li>        10.2 salva link del post, username e link user in "posts"</li> </ol>
Postcondizione: i dati dell'utente vengono salvati in "post_users", mentre i post dell'utente vengono salvati in "posts"

**Figura 3.4:** Caso d'uso relativo a *Profile scraper*

Caso d'uso: ScrapePost
ID: 3
<p>Breve descrizione.            Lo scraper recupera il link di un post dalla tabella "posts" e apre tale link.            Dal post recupera descrizione, eventuali immagini e/o video ed i commenti.            Salva questi dati nella tabella "posts_info".</p>
Attore: user
Precondizione: la tabella "posts" esiste ed ha valori al proprio interno.
<p>Main flow:</p> <ol style="list-style-type: none"> <li>1 if "posts"</li> <li>2 if not "posts_info":</li> <li>    2.1 crea la tabella "posts_info"</li> <li>3 inizializza web driver</li> <li>4 while True:</li> <li>    5 recupera il link di un post dalla tabella "posts"</li> <li>    6 apre il post</li> <li>    7 if post id = 1:</li> <li>        7.1 accetta i cookie</li> <li>    8 recupera descrizione, immagini, video e commenti</li> <li>    9 salva questi dati con il post id in "posts_info"</li> </ol>
Postcondizione: i dati relativi ad ogni singolo post sono salvati nella tabella "posts_info"

**Figura 3.5:** Caso d'uso relativo a *Post scraper*

Questi due tipi di requisiti si distinguono per il diverso approccio che hanno nella cate-

gorizzazione dei requisiti stessi. In particolare:

- i requisiti funzionali descrivono i comportamenti che il sistema dovrebbe offrire;
- i requisiti non funzionali indicano una proprietà specifica o un vincolo del sistema.

I requisiti rappresentano, o dovrebbero rappresentare, la base di tutti i sistemi. Essi costituiscono essenzialmente una dichiarazione di ciò che il sistema dovrebbe fare. In linea di principio, i requisiti dovrebbero limitarsi a specificare "cosa" il sistema deve fare, senza entrare nel dettaglio di "come" questa funzionalità debba essere realizzata. Questa distinzione è importante: è possibile descrivere cosa un sistema deve fare e quale comportamento deve esibire, senza necessariamente specificare come tali funzionalità debbano essere implementate.

Sebbene, in teoria, l'ideale sarebbe di separare il "cosa" dal "come", nella pratica un insieme di requisiti tende a mescolare questi due aspetti. Ciò avviene, in parte, perché spesso è più semplice scrivere e comprendere una descrizione dell'implementazione piuttosto che una dichiarazione astratta del problema, e in parte perché possono esistere vincoli di implementazione che predeterminano il "come" del sistema.

I casi d'uso visti precedentemente non sono sufficienti per formalizzare i requisiti; di conseguenza, si preferisce aggiungere un ulteriore formato per elencarli al meglio. Seguendo l'Arlow e Neustadt [2009] si fa riferimento ad alcune affermazioni strutturate in una maniera ben precisa: ogni requisito ha un identificativo univoco, una keyword (*dove*) e la funzione da compiere. In questo caso si userà la struttura proposta: *<id> il <sistema> deve <funzionalità>*; dove il *<sistema>* coincide con lo scraper stesso.

Si propongono due esempi, uno per requisito, in modo tale da rendere più chiari i concetti:

- un requisito funzionale potrebbe essere: "lo scraper deve estrarre i titoli e i prezzi dei prodotti da un sito di e-commerce";
- un requisito non funzionale potrebbe essere: "lo scraper deve completare l'estrazione dei dati entro 30 secondi per una pagina contenente 50 elementi".

Per definire i requisiti possono anche essere utilizzati dei tool, come *Rational RequisitePro* di IBM. Solitamente questo approccio è consigliato per progetti di grandi dimensioni, dove i requisiti sono categorizzati per tipi, in modo da facilitarne l'organizzazione, e si espandono in profondità. Tuttavia, per semplificare la trattazione, si farà a meno di questi strumenti; infatti, si farà unicamente riferimento alla sintassi di descrizione dei requisiti sopra citata.

Ogni requisito può essere arricchito con attributi, che descrivono ulteriori caratteristiche specifiche di quel requisito. Primo tra tutti è l'attributo inerente la priorità di un requisito. Un modo molto utilizzato per assegnare le priorità è tramite lo schema *MoSCow*, in cui ogni requisito può avere le priorità "M", "S", "C" e "W". Queste lettere definiscono le priorità dalla più alta più bassa. Anche in questo caso, si preferisce non appesantire i requisiti e rimanere in una definizione più leggera e semplice.

### 3.2.1 Requisiti funzionali

In questa sezione si elencano i requisiti funzionali che deve avere lo scraper su Threads:

- *RF1: lo scraper deve essere in grado di accettare i cookie;*
- *RF2: lo scraper deve scorrere tutti i post presenti nella home page;*

- RF3: lo *scraper* deve estrarre l’indirizzo di tutti i post presenti nella home page;
- RF4: lo *scraper* deve recuperare i link ai profili degli utenti presenti nella home page;
- RF5: lo *scraper* deve salvare i dati raccolti dalla home page in una tabella posts nel database;
- RF6: lo *scraper* deve interrogare la tabella posts per recuperare i link ai profili degli utenti;
- RF7: lo *scraper* deve estrarre i dati del profilo, inclusi username, biografia, seguaci e immagine profilo da ogni singolo utente;
- RF8: lo *scraper* deve salvare i dati estratti dai profili degli utenti in una tabella post\_users del database;
- RF9: lo *scraper* deve scorrere il profilo utente e recuperare tutti i post presenti con le stesse modalità viste per la home page;
- RF10: lo *scraper* deve salvare i post raccolti dal profilo utente nella tabella posts nel database;
- RF11: lo *scraper* deve interrogare la tabella posts per recuperare i link dei singoli post;
- RF12: lo *scraper* deve accedere alle pagine dei singoli post e raccogliere dati specifici come descrizione del post, comprese immagini e video, ed estrapolare tutti i commenti sotto il post;
- RF13: lo *scraper* deve salvare i dati raccolti da ogni singolo post nella tabella posts\_info del database;
- RF14: lo *scraper* deve predisporre immagini e video con link all’interno della tabella nominata posts\_info, in modo tale da poter esseri scaricati da uno script di appoggio.

### 3.2.2 Requisiti non funzionali

In questa sezione si elencano i requisiti non funzionali che deve avere lo *scraper*:

- RNF1: lo *scraper* deve essere implementato in Python;
- RNF2: lo *scraper* deve essere in grado di gestire l’esecuzione di codice JavaScript;
- RNF3: lo *scraper* deve utilizzare la libreria Selenium per poter recuperare i dati sulla piattaforma Threads;
- RNF4: lo *scraper* deve essere in grado di sfuggire a meccanismi anti-bot;
- RNF5: lo *scraper* deve supportare una memoria stabile, un database su cui salvare i dati raccolti;
- RNF6: lo *scraper* deve supportare l’utilizzo di MySQL in un ambiente dockerizzato per la portabilità;
- RNF7: lo *scraper* deve essere in grado di non interrompersi in caso di errore nella raccolta di informazioni;
- RNF8: lo *scraper* della home page deve riavviarsi automaticamente, con una nuova sessione sulla home page, una volta raggiunta la fine della pagina principale;
- RNF9: lo *scraper* deve permettere una manutenzione ordinaria del codice, in particolare per gli XPATH che variano continuamente;

- RNF10: lo *scraper* deve poter gestire situazioni in cui la banda a disposizione è ridotta senza bloccarsi permanentemente;
- RNF11: lo *scraper* deve completare l'estrazione dei dati da un post entro pochi secondi;
- RNF12: lo *scraper* deve supportare aggiornamenti incrementali per evitare di riscrivere dati già raccolti;
- RNF13: lo *scraper* deve fornire log dettagliati delle operazioni per facilitare il debugging e il monitoraggio delle attività;
- RNF14: lo *scraper* deve operare senza interrompere il normale funzionamento del sito da cui raccoglie i dati;
- RNF15: lo *scraper* deve essere conforme alle normative sulla privacy e non raccogliere dati personali protetti.

### 3.2.3 Ulteriori sviluppi

Lo *scraper* ha ampi margini di miglioramento, quindi dà adito a nuove idee implementative per migliorare l'efficienza, la scalabilità e la qualità del software stesso. Per queste ragioni si propongono alcune possibili evoluzioni e miglioramenti per lo scraping su Threads.

I suggerimenti sugli ulteriori sviluppi del software mantengono la stessa sintassi dei requisiti funzionali e non funzionali:

- US1: lo *scraper* potrebbe essere eseguito in modalità distribuita utilizzando Selenium Grid per abilitare la parallelizzazione delle operazioni di scraping;
- US2: lo *scraper* potrebbe tradurre i file Jupyter Notebook in script Python eseguibili, con l'aggiunta della modalità headless per l'uso in produzione;
- US3: lo *scraper* potrebbe utilizzare variabili d'ambiente per gestire dinamicamente i percorsi degli XPATH e del webdriver;
- US4: lo *scraper* potrebbe essere integrato con un framework come Scrapy per migliorare la modularità e la gestione dei dati estratti;
- US5: lo *scraper* potrebbe implementare la rotazione dei proxy o l'utilizzo di una VPN per aumentare l'anonimato e ridurre il rischio di potenziali blocchi;
- US6: lo *scraper* potrebbe essere esteso per sfruttare le API non-ufficiali di Threads, qualora disponibili, per un accesso più diretto e regolamentato ai dati;
- US7: lo *scraper* potrebbe essere completamente dockerizzato, includendo tutte le dipendenze e configurazioni necessarie per semplificare il deployment su diversi ambienti;
- US8: lo *scraper* potrebbe adottare strumenti di Big Data, come Apache Spark, per gestire ed elaborare grandi volumi di dati estratti;
- US9: lo *scraper* potrebbe implementare la persistenza dei volumi Docker per garantire la conservazione dei dati nel database anche dopo la ricostruzione dei container;
- US10: lo *scraper* potrebbe utilizzare un ORM (Object Relational Mapping) per semplificare la connessione e l'interazione con il database, migliorando la manutenibilità del codice;

- *US11: lo scraper potrebbe implementare un controllo temporale che impedisca di scorrere tutto il profilo di un utente se il tempo trascorso dall'ultima volta che ciò è stato fatto è inferiore a una certa soglia, in modo da evitare di analizzare lo stesso profilo ripetutamente.*

### 3.3 Requisiti tecnici

In questa sezione si analizzeranno i requisiti tecnici necessari per il corretto funzionamento dello scraper. In particolare, si renderanno noti il linguaggio di programmazione, l’ambiente di sviluppo, le librerie, il DBMS e ulteriori software utilizzati.

Il linguaggio di riferimento durante tutto lo sviluppo del progetto sarà Python; per semplicità si eviterà l’uso della tipizzazione.

Il sistema operativo di riferimento non sarà univoco; infatti, si utilizzeranno, in modo ambivalente, sia Windows che Linux, di cui il secondo per la dockerizzazione di alcuni strumenti utili alle attività di scraping.

Il materiale di riferimento è interamente disponibile in un repository GitHub pubblico appartenente al sottoscritto.

#### 3.3.1 Linguaggio di riferimento

Python è un linguaggio versatile e indipendente dalla piattaforma, ideale per applicazioni che spaziano dal machine learning allo sviluppo web. La sua sintassi leggibile e la vasta gamma di librerie lo rendono la scelta principale per il web scraping e l’analisi dei dati.

Per il web scraping, Python offre strumenti come `requests`, `urllib` e `BeautifulSoup` per la gestione di HTTP e il parsing HTML, e librerie come `pandas` e `numpy` per l’analisi dei dati. Inoltre, framework come `Flask` e `Django` permettono di integrare facilmente i dati raccolti in applicazioni web.

Grazie al supporto costante della comunità, Python si aggiorna regolarmente, mantenendo la sua rilevanza nei progetti moderni.

Prima di procedere con l’implementazione, è necessario assicurarsi che Python3 e pip3 siano correttamente installati (Listato 3.1). In caso contrario, è necessario:

1. scaricare il programma di installazione di Python dal sito ufficiale: <https://www.python.org/>;
2. durante l’installazione su Windows, assicurarsi di selezionare l’opzione *Add Python to PATH*;
3. al termine dell’installazione, verificare che Python e pip siano funzionanti eseguendo i comandi nel terminale espressi dal Listato 3.1.

---

```
C:\Users\luca>python --version
Python 3.12.4
```

```
C:\Users\luca>pip --version
pip 24.2
```

---

**Listing 3.1:** Verifica delle dipendenze Python e pip

### 3.3.2 Ambiente di sviluppo

Il codice verrà sviluppato tramite *Jupyter Notebook*. Quest'ultimo è un'applicazione web open source che consente di creare e condividere documenti che contengono codice live, equazioni, visualizzazioni e testo narrativo. Si deve, quindi, dapprima installare *Notebook*, comprensivo del kernel in modo tale da lavorare su un ambiente virtuale Python dedicato, al fine di evitare conflitti e di avere una gestione più pulita dei pacchetti installati. Questa parte di sviluppo verrà realizzata su sistema Windows e si consiglia di utilizzare *cmd* per i seguenti comandi, invece di *PowerShell*, che potrebbe applicare delle restrizioni.

Si crea l'ambiente virtuale, come mostrato nel Listato 3.2.

---

```
python -m venv jupynote
```

---

#### **Listing 3.2:** Creazione dell'ambiente virtuale

Si attiva l'ambiente virtuale (Listato 3.3).

---

```
jupynote\Scripts\activate
```

---

#### **Listing 3.3:** Attivazione dell'ambiente virtuale

Con l'ambiente virtuale attivo, si installa *Jupyter Notebook* ed il *kernel* (Listato 3.4).

---

```
pip install notebook ipykernel
```

---

#### **Listing 3.4:** Installazione di *Jupyter Notebook* e del *kernel*

Si aggiunge manualmente l'ambiente virtuale come *kernel* alternativo per *Jupyter Notebook* (Listato 3.5).

---

```
python -m ipykernel install --user --name=jupynote
```

---

#### **Listing 3.5:** Aggiunta dell'ambiente virtuale come *kernel* al *Notebook*

Infine, il *Notebook* può essere lanciato tramite il comando illustrato nel Listato 3.6.

---

```
jupyter notebook
```

---

#### **Listing 3.6:** Avvio di *Jupyter Notebook*

Si aprirà una pagina web; il *Notebook* viene eseguito, di default, sulla porta 8888. Ora si può creare un nuovo *notebook* e relativo *kernel*, selezionando: *kernel>change kernel*, come mostrato in Figura 3.6.

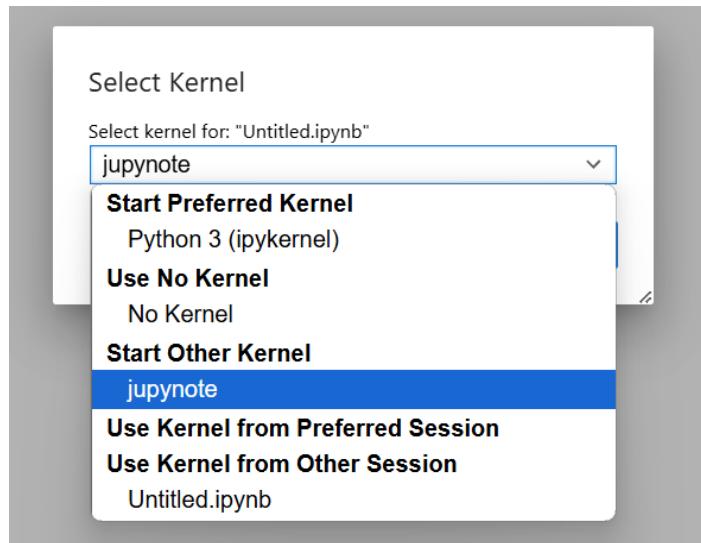
Per disattivare l'ambiente virtuale, come in Linux, si esegue da terminale il comando mostrato nel Listato 3.7.

---

```
deactivate
```

---

#### **Listing 3.7:** Codice per disattivare l'ambiente virtuale



**Figura 3.6:** Aggiunta del *kernel* a *Jupyter Notebook*

La scelta di utilizzare *Jupyter Notebook* su Windows è stata fatta per osservare il comportamento dello scraper su una finestra di Google Chrome. L'obiettivo è quello di osservare graficamente movimenti ed eventuali errori dello scraper durante lo sviluppo.

### 3.3.3 Selenium

Nello scraping, come anticipato nel capitolo precedente, si può scegliere se utilizzare un metodo tradizionale, quale l'utilizzo di librerie di scraping che interagiscono direttamente con l'interfaccia web del sito interessato, oppure utilizzare le API. Threads, purtroppo, almeno fino a qualche tempo fa, non supportava API ufficiali, e quelle in circolazione non erano manutenute regolarmente. Nei tempi recenti Threads ha una API ufficiale; tuttavia gli autori di Meta hanno imposto delle limitazioni sul proprio utilizzo, in modo tale da rendere impossibili le operazioni di crawling. L'API necessita di accesso autenticato ed è limitata nelle operazioni.

Selenium è stato scelto come strumento per lo scraping su Threads per la sua capacità di gestire pagine web dinamiche che utilizzano JavaScript. A differenza di altre librerie di scraping, come *BeautifulSoup* o *requests*, Selenium emula un vero browser, consentendo di interagire con elementi dinamici e caricati in modo asincrono, come post, profili utente e contenuti multimediali. Inoltre, l'utilizzo di Selenium permette di aggirare alcune misure anti-bot implementate dalla piattaforma, come la gestione dei cookie. Questa scelta garantisce un'acquisizione dei dati più completa e affidabile rispetto ad altri strumenti.

Selenium, come molti altri *tool* di scraping, è stato creato per scopi di testing nella programmazione web, ma viene spesso utilizzato per effettuare scraping su pagine dinamiche. La principale caratteristica di questa suite di *tool* è che non contiene un proprio web browser, ma ha bisogno di un'integrazione per poter funzionare correttamente. In sostanza, una volta collegato al browser di riferimento in modalità non *headless*, Selenium aprirà autonomamente una scheda del browser con la scritta: "Chrome è controllato da software di test automatizzato". L'elemento che permette a Selenium di controllare il browser è il *webdriver*.

#### Installazione di Selenium

Dunque, un browser è necessario per il corretto funzionamento di Selenium; quindi, Chrome installato sulla propria macchina è un prerequisito per il corretto funzionamen-

to di Selenium. Successivamente, all'interno dell'ambiente virtuale creato in precedenza, si può installare Selenium (Listato 3.8).

---

```
pip install selenium
```

---

#### **Listing 3.8:** Installazione di Selenium

Le versioni precedenti di Selenium richiedevano il download manuale del *webdriver*. Quest'ultimo funziona come un driver software specifico per il browser; analogamente ai driver utilizzati per i dispositivi hardware, esso consente al pacchetto Python di Selenium di comunicare e controllare il browser. Tuttavia, con l'aggiornamento continuo dei browser, reso ancora più frequente dagli aggiornamenti automatici, l'aggiornamento del driver risultava un'operazione tediosa: ogni volta che veniva rilasciata una nuova versione del browser, era necessario scaricare nuovamente il driver. Negli ultimi anni, questo processo è stato semplificato grazie all'introduzione del pacchetto Python *webdriver manager*. Questo strumento automatizza la gestione del driver, eliminando la necessità di interventi manuali. L'utilizzo del *webdriver manager* non solo riduce gli errori legati a incompatibilità di versioni, ma migliora l'esperienza di sviluppo.

Per installare il *webdriver* è sufficiente utilizzare il comando `pip` (Listato 3.9).

---

```
pip install webdriver-manager
```

---

#### **Listing 3.9:** Installazione del *webdriver*

Il *webdriver manager* permette di scaricare automaticamente l'ultimo driver a disposizione.

### Individuazione degli elementi

Come per lo scraping statico, l'obiettivo fondamentale è quello di individuare, tramite dei selettori, la posizione esatta delle porzioni di HTML che vogliono essere recuperate dalla pagina web. Selenium offre due metodi principali per localizzare elementi sulla pagina web, che ricordano due metodi visti nel capitolo precedente:

- `find_element`;
- `find_elements`.

I due metodi sono sostanzialmente simili: entrambi, in base al tipo di *selettore*, cercano l'elemento nella pagina. Il primo metodo ritorna soltanto il primo elemento che trova nella ricerca; il secondo metodo ritorna tutti gli elementi individuati, sotto forma di lista. Per localizzare gli elementi ci si appoggia alla classe `By`, seguita dalla sintassi propria del selettore da cercare. Si possono cercare gli elementi secondo i seguenti pattern:

- `id`;
- attributo `name`;
- XPath;
- stringhe all'interno del tag `<a>`;
- `tag`;

- classe;
- selettore CSS.

La Tabella 3.1 chiarisce ulteriormente il tipo di selettore e la relativa applicazione.

Tipo	DOM	Esempio
By.ID	<div id="myID">	find_element(By.ID, "myID")
By.NAME	<input name="myNAME">	find_element(By.NAME, "myNAME")
By.XPATH	<span>My <a>Link</a></span>	find_element(By.XPATH, "//span/a")
ByLINK_TEXT	<a>My Link</a>	... (ByLINK_TEXT, "My Link")
By.TAG_NAME	<h1>	... (By.TAG_NAME, "h1")
By.CLASS_NAME	<div class="myCLASS">	... (By.CLASS_NAME, "myCLASS")
By.CSS_SELECTOR	<span>My <a>Link</a></span>	... (By.CSS_SELECTOR, "span > a")

**Tabella 3.1:** Selettori Selenium

## Operazioni sugli elementi

Selenium, oltre ad avere la capacità di manovrare un browser su pagine dinamiche, permette di effettuare operazioni sugli elementi recuperati; in particolare:

- `element.text` garantisce di recuperare la parte visibile di un elemento HTML;
- `element.click()` permette di cliccare su un elemento, come nel caso di un pulsante;
- `element.get_attribute('attribute_name')` permette di recuperare il valore di un attributo;
- `element.send_keys('your_text')` è in grado di completare delle form.

## Eseguire script JS

Selenium prevede la possibilità di eseguire codice JavaScript nativo all'interno del browser stesso. Il metodo `execute_script` permette di eseguire azioni dinamiche che non sono supportate direttamente da Selenium.

Selenium è in grado di stilare gli elementi del DOM. Si può alterare lo stile di elementi applicando direttamente il CSS come argomento del metodo (Listato 3.10).

---

```
javaScript = """document.querySelectorAll('a').
  forEach(e => e.style.border='red')"""

driver.execute_script(javaScript)
```

---

**Listing 3.10:** Alterazione dello stile del CSS

Selenium ha la capacità di ritornare valori derivanti dall'esecuzione dello script al fine di elaborarli in Python (Listato 3.11).

---

```
title = driver.execute_script('return document.title')
```

---

**Listing 3.11:** Ritorno dei valori dall'esecuzione di script

---

```
javaScript = "window.scrollBy(0, 1000);"  
driver.execute_script(javaScript)
```

---

**Listing 3.12:** Scorrimento di una pagina

In ultima analisi, Selenium permette di scorrere la pagina alla ricerca di elementi non ancora caricati (Listato 3.12).

Quest'ultimo esempio troverà applicazione pratica nel codice dello scraper principale. In particolare, si va ad eseguire uno scroll infinito finché non si raggiunge la fine della home page di Threads; in tal caso si riavvia lo script per ricominciare con il crawling su una nuova home page.

### 3.3.4 MySQL Docker container

#### Windows Subsystem for Linux

Windows Subsystem for Linux (WSL) consente agli sviluppatori di eseguire un ambiente GNU/Linux direttamente su Windows. Questo evita l'utilizzo di una macchina virtuale (VM) tradizionale o di una configurazione dual-boot.

Per installare WSL con Ubuntu integrato, si può lanciare il comando nel Listato 3.13.

---

```
wsl --install
```

---

**Listing 3.13:** Installazione di WSL con Ubuntu

Per installare una distribuzione particolare si può utilizzare lo stesso comando di prima seguito dal nome della distribuzione: `wsl -install <Distribution Name>`. La lista delle distribuzioni disponibili può essere ottenuta lanciando il comando: `wsl -list -online`. Se si ha una sola distribuzione Linux installata, è sufficiente eseguire il comando del Listato 3.14 per avviarla.

---

```
wsl
```

---

**Listing 3.14:** Avvio di una distribuzione Linux predefinita

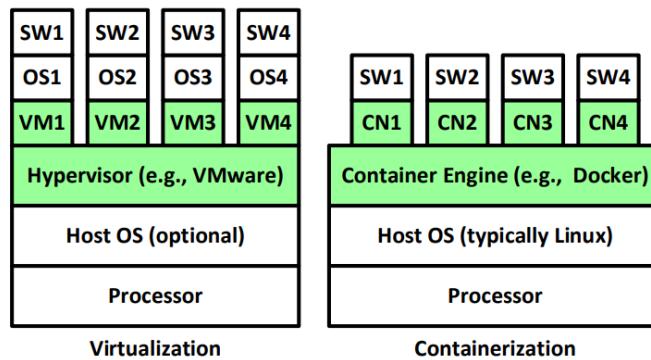
Se si hanno più distribuzioni installate e si desidera specificare quale avviare, si utilizza il comando `wsl -d <Distribution Name>`.

Si elencano alcuni comandi utili per gestire WSL:

- per controllare le distribuzioni installate: `wsl -l -v`;
- per verificare lo stato di WSL: `wsl -status`;
- per aggiornare WSL: `wsl -update`;
- per controllare la versione di WSL: `wsl -version`;
- per arrestare la macchina Linux su WSL: `wsl -shutdown`.

### Virtualizzazione e Containerizzazione

Docker è un potentissimo strumento che, negli ultimi anni, contrapponendosi al concetto tradizionale di *Virtual Machine* (VM), ha preso il sopravvento. Docker è un'espressione particolare del concetto più generale di containerizzazione, mentre le virtual machine si particolarizzano nelle realizzazioni pratiche dei prodotti di VMware o di VirtualBox e si generalizzano nel concetto più ampio di virtualizzazione (Figura 3.7).



**Figura 3.7:** Schema di base di Virtualizzazione e Containerizzazione

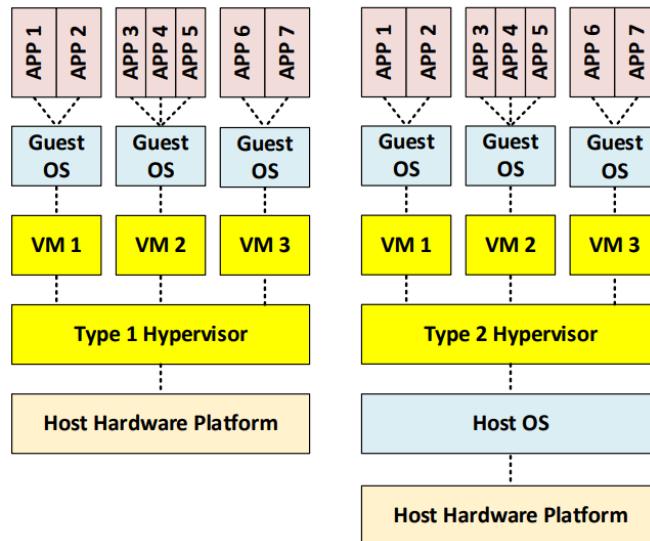
Per spiegare le caratteristiche di Docker è bene partire dal concetto di VM. Una Virtual Machine, anche detta *guest machine*, è una simulazione software di hardware in grado di fornire un sistema operativo virtuale all'utente. La caratteristica fondamentale è che le VM vengono eseguite su un *hypervisor*. Un *hypervisor* è un programma software eseguito su dell'hardware reale della macchina su cui è installato e gestisce i sistemi operativi relativi ad ogni singola VM presente. In generale, si possono avere due configurazioni di base per un *hypervisor*:

- *Tipo 2*: è la più semplice ed è quella presente sui personal computer quando si utilizza un servizio come VirtualBox. Prevede l'*hypervisor* installato sul sistema operativo della macchina host. Ad un livello superiore dell'*hypervisor* è possibile installare una moltitudine di VM con i relativi sistemi operativi, sui quali eseguire le varie applicazioni.
- *Tipo 1*: questa tipologia di architettura è presente nelle applicazioni industriali e, per avere una migliore efficienza, prevede che l'*hypervisor* interagisca direttamente con l'hardware fisico della macchina host, senza intermediari software, come il sistema operativo della macchina host.

Le due tipologie sono graficamente rappresentate in Figura 3.8. Le VM hanno avuto grande successo perché garantiscono alcuni aspetti positivi, tra i quali spiccano: un maggior isolamento tra le varie VM e la macchina host, un minor costo dal punto di vista hardware e una maggiore sicurezza in generale, dovuta all'isolamento delle componenti.

Un *container*, diversamente, è un ambiente isolato che esegue un'applicazione e le sue dipendenze su un singolo kernel di sistema operativo condiviso, senza la necessità di emulare l'hardware sottostante.

In particolare, Docker si basa su due concetti fondamentali: *container* ed *immagini*. Le immagini contengono le informazioni necessarie per costruire ed eseguire un container; i container rappresentano l'istanziazione di queste immagini, cioè la loro esecuzione. Per fare un'analogia con il concetto di programmazione ad oggetti, l'istanza di una immagine è un container quanto l'istanza di una classe è un oggetto.



**Figura 3.8:** Hypervisor di Tipo 1 e di Tipo 2

Il motivo principale del successo dei container, e di conseguenza di Docker, è dovuto al fatto che prevedono gli stessi vantaggi di isolamento delle VM, ma eliminando tutto l'*overhead* proprio di queste ultime. Tra i vantaggi di questo nuovo approccio si elencano: scalabilità, disponibilità, affidabilità, load balancing, supporto per CI/CD, etc.

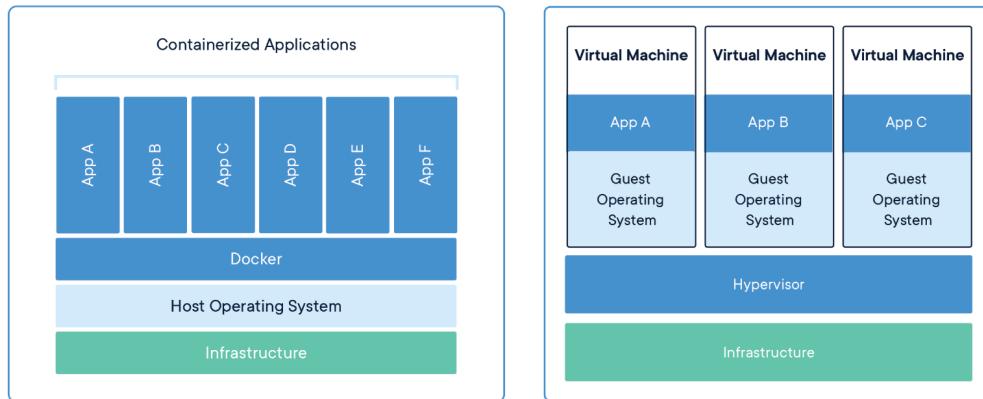
Docker e VM presentano alcune somiglianze, ma anche delle notevoli differenze, come anticipato. Le somiglianze possono essere riassunte nel seguente elenco:

- i container Docker e le macchine virtuali sono entrambi creati a partire da immagini;
- un'immagine specifica tutte le risorse di sistema necessarie per eseguire le applicazioni;
- è presente un controllo delle versioni;
- le immagini Docker e le immagini VM sono entrambe portabili tra architetture.

Per elencare le differenze si deve tenere, innanzitutto, in considerazione lo scopo dei due diversi approcci: le VM sono state originariamente progettate per consentire l'esecuzione di più sistemi operativi su un'unica macchina fisica, mentre Docker è stato progettato per fornire un modo leggero e portatile per impacchettare ed eseguire applicazioni in un ambiente isolato e riproducibile. In altri termini, Docker si rivolge di più allo sviluppo di applicazioni, volendo risolvere il problema di implementare app in ambienti diversi tra loro; da questi presupposti nasce la containerizzazione. Un container Docker, diversamente da una VM, contiene solo l'applicazione e le sue dipendenze. Una rappresentazione grafica delle due architetture è visibile in Figura 3.9.

I Docker container funzionano su architettura Linux; per questo motivo si utilizza WSL. L'esecuzione del container Docker prevede i seguenti passaggi:

- scaricare ed avviare Docker Desktop;
- abilitare Docker su WSL (Settings > General > Use the WSL 2 based engine);
- selezionare, tra le distribuzioni di WSL installate, quella in cui si vuole abilitare l'integrazione di Docker (Settings > Resources > WSL Integration).



**Figura 3.9:** Differenze tra Docker Container e Virtual Machine

Docker Desktop contiene i seguenti componenti: Docker Engine, Docker CLI client, Docker Scout, Docker Build, Docker Extensions, Docker Compose, Docker Content Trust, Kubernetes, Credential Helper. Di questi quelli di interesse sono Docker Engine, che permette la containerizzazione, e Docker Compose, che verrà utilizzato per settare il container di MySQL.

Ora, per verificare che Docker sia correttamente installato, è necessario aprire una distribuzione WSL e visualizzare la versione e il numero di build immettendo il comando del Listato 3.15.

---

```
docker --version
```

---

**Listing 3.15:** Controllo della versione di Docker installata

Per verificare che l'installazione funzioni correttamente si può eseguire una semplice immagine Docker incorporata; a tal fine si lancia il comando del Listato 3.16.

---

```
docker run hello-world
```

---

**Listing 3.16:** Test sul corretto funzionamento Docker

La risposta dovrebbe essere quella del Listato 3.17.

---

```
Hello from Docker! This message shows that your installation  
appears to be working correctly.
```

---

**Listing 3.17:** Esito del test sul corretto funzionamento Docker

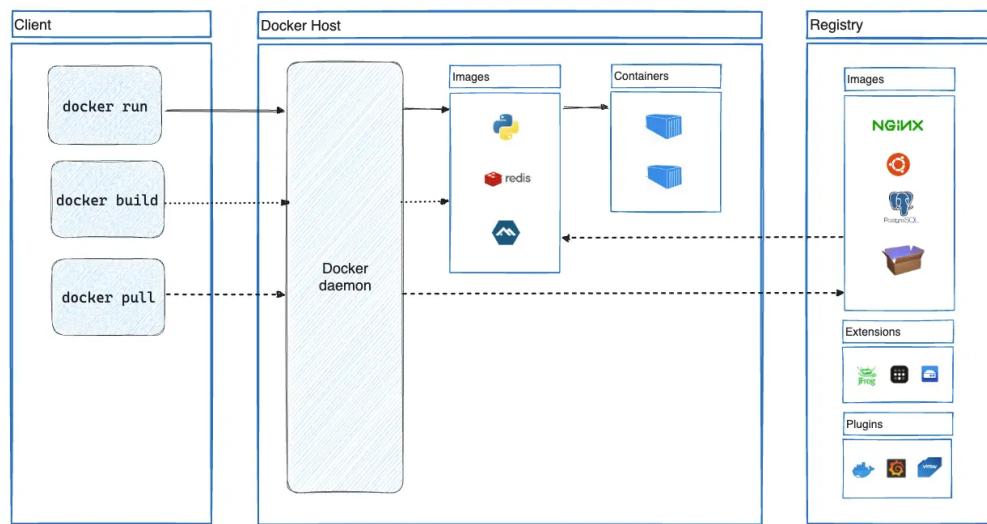
## Docker nel dettaglio

Docker, avvalendosi anche della sua caratteristica di essere open source, fornisce una libreria di immagini, Docker Hub, dalla quale recuperare qualsiasi tipo di software necessario. Di conseguenza, è proprio dal Docker Hub che si può scaricare l'immagine di MySQL.

I comandi fondamentali di Docker, che verranno ripresi nella sezione successiva, possono essere suddivisi in tre macro-categorie ad indicare tre operazioni fondamentali:

- `docker pull` per il recupero dell'immagine;
- `docker build` per costruire l'immagine;
- `docker run` per avviare il container.

La Figura 3.10 illustra graficamente le implicazioni pratiche dei tre comandi appena elencati.



**Figura 3.10:** Le tre operazioni principali di Docker

Docker, inoltre, implementa un sistema di rete flessibile che consente ai container di comunicare tra di loro, con l'host e con reti esterne.

Relativamente alle immagini ed ai container, si possono lanciare diversi comandi. Per eseguire un container a partire da un'immagine, si utilizza il comando `docker run`. Ad esempio, si osservi il Listing 3.18.

---

```
docker run -d --name CONTAINER_NAME IMAGE_NAME
```

---

**Listing 3.18:** Avvio di un container da un'immagine

Il flag `-d` esegue il container in modalità *detached*, ovvero in background, mentre `-name` assegna un nome al container per facilitarne la gestione. Si può aggiungere anche il flag `-rm` il quale elimina il container al suo spegnimento.

Per accedere a un container in esecuzione, si utilizza il comando `docker exec` (Listing 3.19).

---

```
docker exec -it CONTAINER_NAME /bin/bash
```

---

**Listing 3.19:** Avvio di un container con accesso interattivo

Il flag `-it` apre una sessione interattiva nella shell del container specificato; questo flag può essere anche applicato direttamente al `docker run`.

Se, invece, si vuole fermare un container in esecuzione si usa il comando illustrato nel Listing 3.20.

Per visualizzare il containerID si lancia il comando del Listing 3.21.

---

```
docker stop CONTAINER_ID
```

---

**Listing 3.20:** Stop di un container in esecuzione

---

```
docker ps [-a]
```

---

**Listing 3.21:** Visualizzazioni dei container attivi [tutti i container]

Le statistiche dei container si ottengono con il comando del Listato 3.22.

---

```
docker stats
```

---

**Listing 3.22:** Statistiche live dei container sull'utilizzo delle risorse

Per visualizzare le immagini, invece, si può lanciare il comando nel Listato 3.23.

---

```
docker images
```

---

**Listing 3.23:** Visualizzazione delle immagini Docker presenti sul sistema

Si possono visualizzare le info specifiche di un container, come indirizzo IP e MAC, tramite il comando nel Listato 3.24.

---

```
docker container inspect CONTAINER_NAME
```

---

**Listing 3.24:** Visualizzazione delle informazioni sul container

Per creare un'immagine Docker personalizzata, è necessario partire da un file chiamato `Dockerfile`, che descrive i passaggi per configurare l'immagine. Per costruire l'immagine, si utilizza il comando `docker build` (Listato 3.25).

---

```
docker build -t IMAGE_NAME PATH_TO_DOCKERFILE
```

---

**Listing 3.25:** Costruzione di un'immagine Docker

Il flag `-t` permette di assegnare un nome all'immagine, mentre `PATH_TO_DOCKERFILE` indica la directory in cui si trova il `Dockerfile`. Ad esempio, se il `Dockerfile` si trova nella directory corrente, si può far riferimento al Listato 3.26.

---

```
docker build -t IMAGE_NAME .
```

---

**Listing 3.26:** Esempio di costruzione di un'immagine

Inoltre, si può cancellare un container o una immagine con i comandi illustrati nei Listati 3.27 e 3.28.

Infine, per eliminare tutti i container e le immagini non più utilizzati si può fare riferimento al Listato 3.29.

---

```
docker rm CONTAINER_ID
```

---

**Listing 3.27:** Eliminazione di un container

---

```
docker image rm IMAGE_ID
```

---

**Listing 3.28:** Eliminazione di un'immagine

---

```
docker system prune
```

---

**Listing 3.29:** Rimozione di container e immagini inutilizzati

Si nota che, per lanciare i comandi Docker, sono necessari i privilegi da superUser (sudo), si consiglia, quindi, di controllare se l'utente corrente appartenga al gruppo docker, tramite il comando del Listato 3.30.

---

```
groups
```

---

**Listing 3.30:** Visualizzazione del gruppo di appartenenza User corrente

Se i container sono ancora attivi vanno prima spenti o "killati"; a volte può essere necessario forzare le operazioni tramite il classico flag `-f`.

## MySQL

Nel primo capitolo si sono esplorate le finalità del web scraping; prima tra tutti spicca la necessità di svolgere delle analisi sui dati raccolti. Per questa ragione, è indispensabile salvare in modo permanente i dati raccolti durante lo scraping. Da qui l'idea di creare una memoria persistente, organizzata, disponibile ed interrogabile; in una parola: un database relazionale.

MySQL è uno dei sistemi di gestione di database relazionali (RDBMS) più popolari e diffusi al mondo. Open source e sviluppato originariamente da MySQL AB, è ora gestito da Oracle Corporation. Grazie alla sua efficienza, scalabilità e semplicità d'uso, MySQL è una scelta ideale sia per piccoli progetti personali che per grandi applicazioni enterprise. Per questi motivi, esso è stato scelto come database per il nostro progetto.

Progettato per essere rapido e affidabile, MySQL utilizza il linguaggio SQL (Structured Query Language) per la gestione e la manipolazione dei dati. Tra le sue caratteristiche principali figurano:

- *Compatibilità multi-piattaforma*: funziona su sistemi operativi come Windows, macOS e Linux.
- *Scalabilità*: è in grado di gestire database di grandi dimensioni e supportare un numero elevato di query simultanee.
- *Sicurezza avanzata*: è in grado di garantirla grazie a funzionalità come autenticazione basata sugli account e cifratura delle connessioni.

MySQL permette di interagire con le tabelle utilizzando le classiche query del linguaggio SQL. Inoltre, il DBMS fornisce una propria sintassi per effettuare alcune operazioni particolari, come, ad esempio, selezionare un determinato database (Listato 3.31).

---

```
USE DATABASE_NAME;
```

---

**Listing 3.31:** Selezione di un database

D’altro canto, per visualizzare tutti i database disponibili, si deve lanciare il comando del Listato 3.32.

---

```
SHOW DATABASES;
```

---

**Listing 3.32:** Visualizzazione dei database disponibili

Inoltre, si può visualizzare la struttura di una tabella con il comando mostrato nel Listato 3.33.

---

```
DESCRIBE TABLE_NAME;
```

---

**Listing 3.33:** Visualizzazione delle caratteristiche di una tabella

Il database MySQL verrà utilizzato in combinazione con Docker Compose, invece di essere installato direttamente sulla macchina, in modo tale da favorire la portabilità e una versatilità maggiore nello sviluppo.

### DockerCompose

*Docker Compose* è uno strumento aggiuntivo che semplifica la gestione e l’orchestrazione di applicazioni multi-container. Invece di avviare ogni container manualmente con comandi separati, come mostrato nelle sezioni precedenti, Docker Compose utilizza un file di configurazione in formato YAML (solitamente `docker-compose.yml`), dove è possibile definire:

- i container richiesti;
- le immagini da utilizzare;
- la configurazione delle reti e dei volumi;
- eventuali variabili d’ambiente e dipendenze.

Utilizzando il comando nel Listato 3.34 è possibile avviare tutti i servizi definiti nel file YAML, semplificando notevolmente il processo di avvio e gestione dei container.

---

```
docker compose up
```

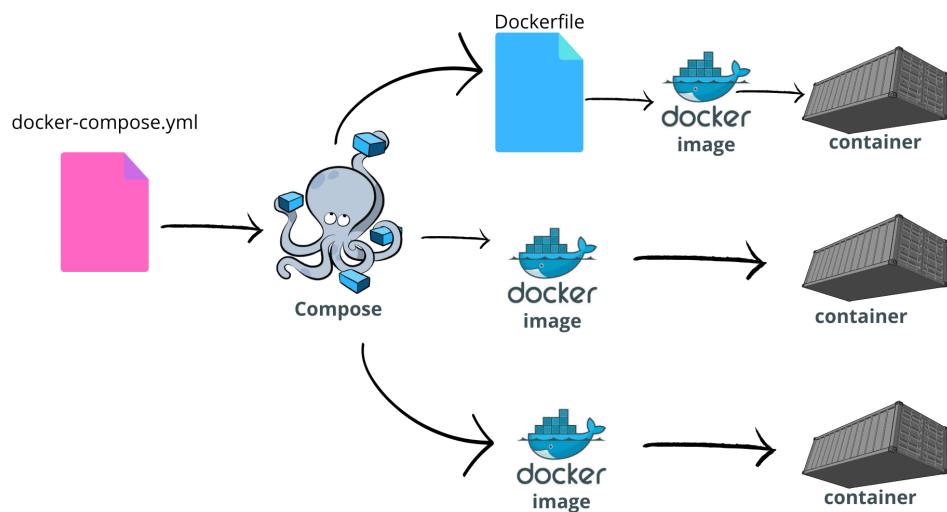
---

**Listing 3.34:** Avvio di Docker Compose

Infatti, Compose scarica le immagini (se necessario), esegue le operazioni di building e avvia i container, tutto in un unico comando. Compose è particolarmente utile in progetti complessi che coinvolgono più container, poiché centralizza la configurazione e assicura una riproducibilità dell’ambiente.

Una rappresentazione di come funziona Compose è mostrata in Figura 3.11.

Per Docker Compose si elencano i seguenti comandi principali:

**Figura 3.11:** Funzionamento Docker Compose

- Il Listato 3.35 avvia tutti i servizi definiti nel file `docker-compose.yml`. Se le immagini non sono già presenti localmente, il comando provvederà a scaricarle dal Docker Hub o a costruirle, se specificato.

---

```
docker-compose up
```

---

**Listing 3.35:** Avvio dei container con Docker Compose

- Il Listato 3.36 avvia i container in modalità `detached`, ovvero in `background`, permettendo di continuare ad utilizzare il terminale per altre operazioni.

---

```
docker-compose up -d
```

---

**Listing 3.36:** Avvio dei container in background

- Il Listato 3.37 costruisce le immagini definite nel file `docker-compose.yml` prima di avviare i container. Questo comando è utile se sono state effettuate modifiche al file di configurazione (Dockerfile).

---

```
docker-compose up --build
```

---

**Listing 3.37:** Ricostruzione delle immagini e avvio dei container

- Il Listato 3.38 avvia i container già creati ma attualmente non in esecuzione, senza ricreare o ricostruire alcun servizio.

---

```
docker-compose start
```

---

**Listing 3.38:** Riavvio di container esistenti

- Il Listato 3.39 ferma i container in esecuzione definiti nel file docker-compose.yml, lasciandoli, però, disponibili per un successivo riavvio.

---

```
docker-compose stop
```

---

**Listing 3.39:** Stop dei container in esecuzione

- Il Listato 3.40 arresta ed elimina tutti i container, le reti e i volumi creati. Questo comando è particolarmente utile per effettuare un "reset" completo dell'ambiente.

---

```
docker-compose down
```

---

**Listing 3.40:** Eliminazione di container, reti e volumi

Nel Listato 3.41 è riportata la configurazione del docker-compose.yml per il database MySQL.

---

```
services:  
  
  db:  
    image: mysql  
    restart: always  
    ports:  
      - 3306:3306  
    environment:  
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}  
      MYSQL_DATABASE: ${MYSQL_DATABASE}  
      MYSQL_USER: ${MYSQL_USER}  
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}  
  
  adminer:  
    image: adminer  
    restart: always  
    ports:  
      - 8080:8080
```

---

**Listing 3.41:** docker-compose.yml

Il file di configurazione definisce due servizi:

- MySQL:
  - utilizza l'immagine mysql;
  - la configurazione di riavvio è impostata su always, garantendo il riavvio automatico del container in caso di arresto;
  - la porta 3306 del container viene "mappata" sulla stessa porta dell'host per consentire l'accesso al database;
  - sono definite alcune variabili di ambiente:
    - \* MYSQL\_ROOT\_PASSWORD;
    - \* MYSQL\_DATABASE;

```
* MYSQL_USER;  
* MYSQL_PASSWORD;
```

per configurare le credenziali e il database iniziale. Questi valori sono letti da un file esterno (`.env`) tramite la sintassi `$ {}`.

- Adminer:

- rappresenta un’istanza di Adminer, un’interfaccia grafica web per la gestione dei database;
- utilizza l’immagine `adminer`;
- la configurazione di riavvio è impostata su `always`;
- la porta 8080 del container viene “mappata” in modo tale da permettere l’accesso all’interfaccia Adminer sulla porta 8080 della macchina host.

In questa configurazione del `docker-compose.yml` è stato omesso, a causa della grande mole di dati, l’utilizzo dei volumi, i quali permettono la persistenza dei dati all’eliminazione del container. Nel caso si voglia garantire la persistenza, è sufficiente aggiungere un volume alla fine delle impostazioni del db.

La connessione al database può essere effettuata sia tramite interfaccia web dalla macchina host sia tramite il terminale. Partendo da quest’ultimo, il comando per connettersi, essendo dockerizzato, è nel Listato 3.42.

---

```
docker exec -it CONTAINER_ID mysql -u USERNAME -pPASSWORD
```

---

**Listing 3.42:** Connessione a MySQL tramite terminale

Di seguito, nel Listato 3.43, si riporta una breve interazione con il DBMS; in particolare, si seleziona il database e la tabella di interesse e di quest’ultima si visualizza la struttura.

A questo punto, per visualizzarne i dati, si può procedere con le classi query SQL.

Questo approccio, essendo poco pratico, ha spinto per un’alternativa grafica da utilizzare come comunicazione con il database. Infatti, si è fatto uso di un *Database management*, ovvero `Adminer`, per gestire MySQL, senza ricorrere alle query tramite terminale. Questo permette una maggior fluidità nello sviluppo e nella visualizzazione dei dati all’interno del database.

Per connettersi ad Adminer si accede all’URL `http://localhost:8080`, da qui vengono immesse le credenziali. Sulla sinistra, in blu, si avranno a disposizione tutte le tabelle. Per fare un esempio pratico, si seleziona la stessa tabella del terminale, `posts`, e si clicca su *visualizza struttura*. Il risultato è analogo al precedente, ma decisamente più immediato.

Tra le caratteristiche di Adminer si elencano: la capacità di connettersi a un server di database utilizzando credenziali di accesso (username e password), con la possibilità di selezionare un database esistente o crearne uno nuovo. È possibile visualizzare e gestire dettagli delle tabelle, come campi, indici e chiavi esterne. Adminer permette di modificare il nome ed il valore `auto_increment` delle tabelle, oltre a cambiare proprietà delle colonne, come nome, tipo e valore predefinito. Supporta l’aggiunta e l’eliminazione di tabelle, colonne e indici, nonché la gestione delle chiavi esterne. È possibile creare, modificare e cancellare viste, procedure memorizzate e funzioni, oltre a effettuare ricerche basate su indici. Adminer consente di visualizzare e manipolare i dati delle tabelle, con funzioni di ricerca, ordinamento, aggregazione e limitazione dei risultati. Si possono inserire nuovi record, aggiornarli o eliminarli, con supporto per tutti i tipi di dati. Il sistema permette di eseguire comandi SQL direttamente da un campo di testo o da un file, esportare strutture e dati in formato SQL o CSV e stampare schemi del database collegati da chiavi esterne.

---

```
mysql> show databases
      -> ;
+-----+
| Database          |
+-----+
| information_schema |
| mysql              |
| performance_schema |
+-----+
3 rows in set (0.02 sec)

mysql> use mysql;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;

mysql> show tables;
+-----+
| Tables_in_mysql   |
+-----+
| ...               |
| posts             |
| posts_info        |
| ...               |
+-----+
40 rows in set (0.00 sec)

mysql> describe posts;
+-----+-----+-----+-----+-----+-----+
| Field       | Type      | Null | Key | Default | Extra      |
+-----+-----+-----+-----+-----+-----+
| id          | bigint    | NO   | PRI | NULL    | auto_increment |
| username    | varchar(255) | NO   |     | NULL    |             |
| username_link | text     | NO   |     | NULL    |             |
| post_link   | varchar(255) | NO   | UNI | NULL    |             |
| created     | timestamp | YES  |     | TIMESTAMP | DEFAULT_GENERATED |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

---

**Listing 3.43:** Connessione a MySQL tramite terminale

### 3.3.5 Dipendenze

In Python è necessario installare il pacchetto PyMySQL per potersi connettere a MySQL direttamente dall'interno dell'ambiente di sviluppo. Per l'installazione si lancia il comando del Listato 3.44.

---

```
pip install PyMySQL
```

---

**Listing 3.44:** Installazione del pacchetto PyMySQL

PyMySQL permette di creare una connessione con il database e di scrivere delle query SQL da lanciare per interrogare il database, direttamente in Python. Unica nota da sottolineare per questa libreria riguarda la connessione al db; infatti, entrano in gioco due soggetti fondamentali: il connection object (conn) ed il cursor object (cur). Il primo è responsabile di creare la connessione con il database ed è unico; il secondo è associato alla connessione, può essere una più di uno e si occupa principalmente di eseguire le query. Un aspetto importante è quello di chiudere sia la connessione che il cursore una volta che non devono essere più utilizzati. Un esempio è riportato nel Listato 3.45.

---

```
# Connection to the database
conn = pymysql.connect(
    host='localhost',
    user='mysql',           # Database username
    password='mysql',       # Database password
    database='mysql',       # Target database
    port=3306               # Port for MySQL server
)

# Creating a cursor object to interact with the database.
# The cursor allows execution of SQL commands and retrieval of results.
cur = conn.cursor()

# Executing an SQL query
cur.execute("SELECT * FROM TABLE")

# Fetching and printing a single row from the result set
print(cur.fetchone())

# Closing the cursor
cur.close()

# Closing the connection
conn.close()
```

---

**Listing 3.45:** Esempio di utilizzo del pacchetto PyMySQL

Per il resto, nella documentazione ufficiale, sono descritti i vari metodi con cui effettuare le query.

MySQL richiede un altro pacchetto da installare inerente la crittografia (Listato 3.46).

---

```
pip install cryptography
```

---

**Listing 3.46:** Installazione del pacchetto cryptography

Una volta installati tutti i pacchetti necessari, i requisiti possono essere salvati nel file convenzionalmente utilizzato per Python (`requirements.txt`), tramite il comando illustrato nel Listato 3.47.

---

```
pip freeze > requirements.txt
```

---

**Listing 3.47:** Salvataggio delle dipendenze

Viceversa, per installare i requisiti, si lancia il comando del Listato 3.48.

---

```
pip install -r requirements.txt
```

---

**Listing 3.48:** Installazione delle dipendenze

# CAPITOLO 4

---

## Progettazione delle classi e del database

---

In questo capitolo viene descritta la progettazione del sistema, con particolare attenzione all’architettura software, alla strutturazione delle classi, alla definizione delle funzioni e alla progettazione del database. In particolare, la Sezione 4.1 presenta l’architettura generale del sistema tramite un diagramma delle componenti, evidenziando le interazioni tra gli scraper, il database e gli altri elementi fondamentali del sistema. Successivamente vengono illustrati i diagrammi di sequenza, che mostrano in dettaglio il flusso delle operazioni per ciascuno scraper. La Sezione 4.2 si concentra sulla definizione delle classi utilizzate nel progetto, descrivendo le loro responsabilità e relazioni. Si analizzano, quindi, le strutture implementate per gestire gli utenti, i post e i dati associati. Nella Sezione 4.3 vengono elencate e descritte le principali funzioni sviluppate per il progetto. Ogni funzione è associata ad uno specifico compito, come la gestione dei driver, l’estrazione dei dati o la gestione del database. Infine, la Sezione 4.4 è dedicata alla progettazione del database, con una descrizione dettagliata delle tabelle posts, post\_users e posts\_info; per tali tabelle vengono illustrati i campi, i vincoli e le relazioni.

### 4.1 Architettura generale

Questa sezione ha il compito di introdurre, in maniera formale e particolarizzata, il comportamento del sistema. Si mostreranno due tipologie di diagrammi UML per esplorare ulteriormente quanto detto nel capitolo precedente; in particolare, si utilizzeranno un diagramma dei componenti e tre diversi diagrammi di sequenza.

Il diagramma dei componenti riassume in termini generali la relazione che intercorre tra lo scraper, articolato nei suoi componenti, lo script di appoggio (*Media downloader*) e il database. Si vuole mettere in atto quanto premesso nell’analisi dei requisiti, in modo tale da soddisfare questi ultimi a pieno.

La seconda rappresentazione UML è costituita da diversi diagrammi di sequenza che entrano maggiormente nel dettaglio nella struttura degli script. Si avrà un diagramma di sequenza per ognuno dei tre script di scraping (*Home Scraper*, *Profile Scraper* e *Post Scraper*).

#### 4.1.1 Diagramma dei componenti

Un diagramma dei componenti è utile per visualizzare, in termini generali, come le diverse parti del sistema interagiscano tra loro e che tipo di relazione vi intercorre. La caratteristica di questo tipo di diagrammi è quella di fornire una panoramica ad alto livello e statica del sistema, senza scendere nei dettagli implementativi.

In un diagramma dei componenti l'elemento fondamentale è costituito dai componenti. Un componente è definito come una unità indipendente all'interno del sistema; ogni componente è munito di una interfaccia per interagire con altri componenti. Questi ultimi sono solitamente rappresentati come delle *black box*.

Quindi, all'interno di un diagramma dei componenti, gli elementi fondamentali sono i componenti e le relazioni che intercorrono tra di essi per mezzo delle interfacce.

Le interfacce possono essere di due tipologie: *required interface* o *provided interface*.

Una *provided interface* rappresenta un insieme di funzionalità che un componente mette a disposizione di altri componenti. In altre parole, definisce i servizi che il componente è in grado di offrire. Queste interfacce sono visibili verso l'esterno e sono utilizzabili da altri componenti che desiderano interagire con il componente.

Una *required interface*, invece, rappresenta le funzionalità di cui un componente necessita per operare correttamente. Indica, quindi, le dipendenze del componente verso altre parti del sistema o servizi esterni.

La differenza principale tra le due interfacce risiede nella direzione del servizio: la *provided interface* offre un servizio al resto del sistema, mentre la *required interface* richiede un servizio da altri componenti. In un diagramma dei componenti UML, le *provided interface* vengono rappresentate come un cerchio collegato al componente, mentre le *required interface* vengono indicate con un mezzo cerchio aperto, rivolto verso l'esterno del componente.

Nel diagramma dei componenti si considerano, in questo caso applicativo, cinque componenti principali:

- *Home Scraper*;
- *Profile Scraper*;
- *Post Scraper*;
- *Media downloader*;
- *MySQL*.

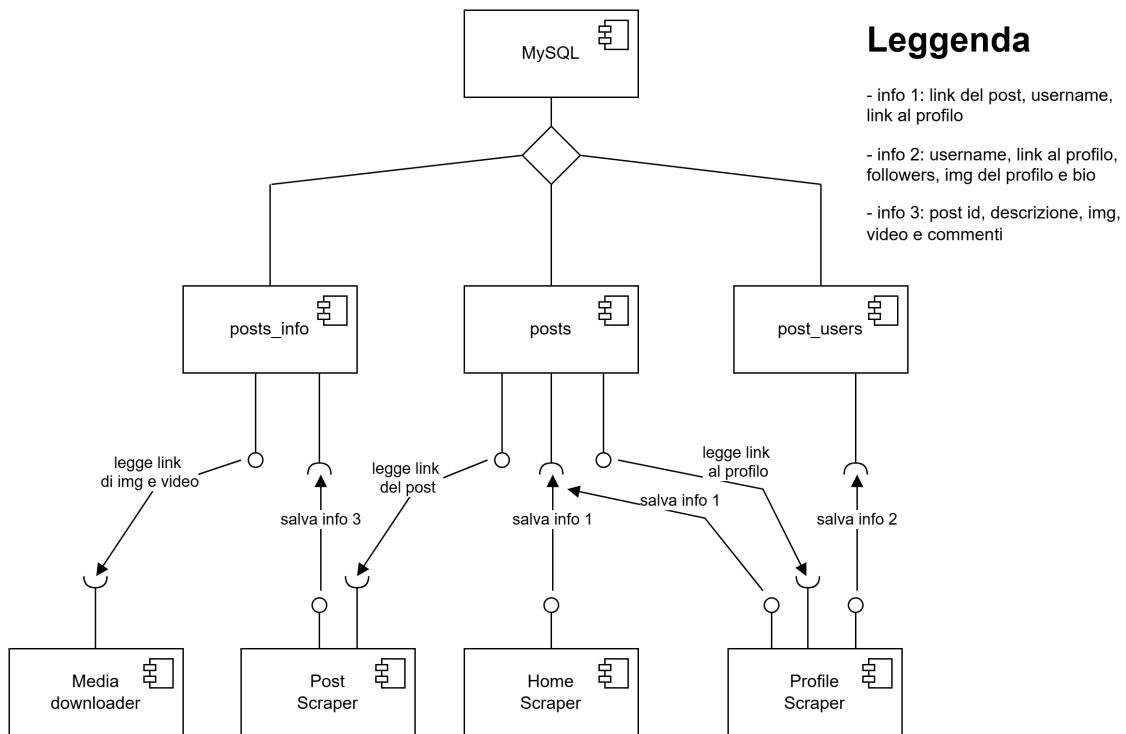
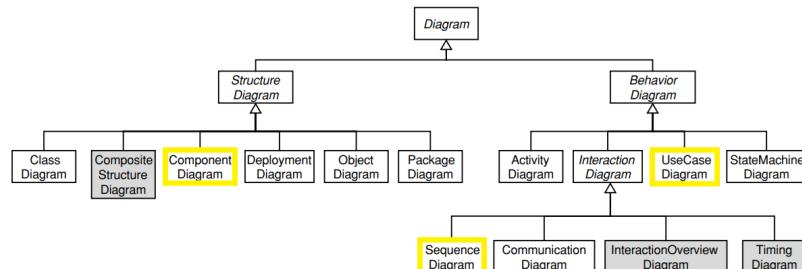
In particolare, l'ultimo componente, che rappresenta il database, a sua volta si sviluppa in ulteriori componenti, uno per ogni tabella di interesse: *posts*, *post\_users* e *posts\_info*. In altre parole, il database è rappresentato da sotto-componenti che raffigurano le singole tabelle.

Le interfacce sono descritte nel diagramma dei componenti in modo tale da mettere in risalto le relazioni che intercorrono tra i componenti degli scraper e il database. In Figura 4.1 è mostrato il diagramma UML in questione.

### 4.1.2 Diagrammi di sequenza

In questo paragrafo vengono mostrati i diagrammi di sequenza. Ad ogni script di scraping viene associato un diagramma di sequenza; si hanno, quindi, tre diagrammi di sequenza. I casi d'uso specifici illustrati nel capitolo precedente vengono, in questo caso, ulteriormente approfonditi tramite i diagrammi di sequenza, in cui vengono illustrate le chiamate di funzioni e l'interazione tra di esse.

Nella Figura 4.2, presa da Arlow e Neustadt [2009], sono raffigurati tutti i diagrammi UML raggruppati per tipologia. Si sono evidenziati (in giallo) i diagrammi utilizzati in questa tesi. In particolare, si nota come i diagrammi di sequenza facciano parte della stessa categoria dei diagrammi dei casi d'uso visti nel capitolo precedente; tuttavia appartengono ad una sotto categoria ulteriore, quella dei diagrammi di interazione. Per questa tipologia di

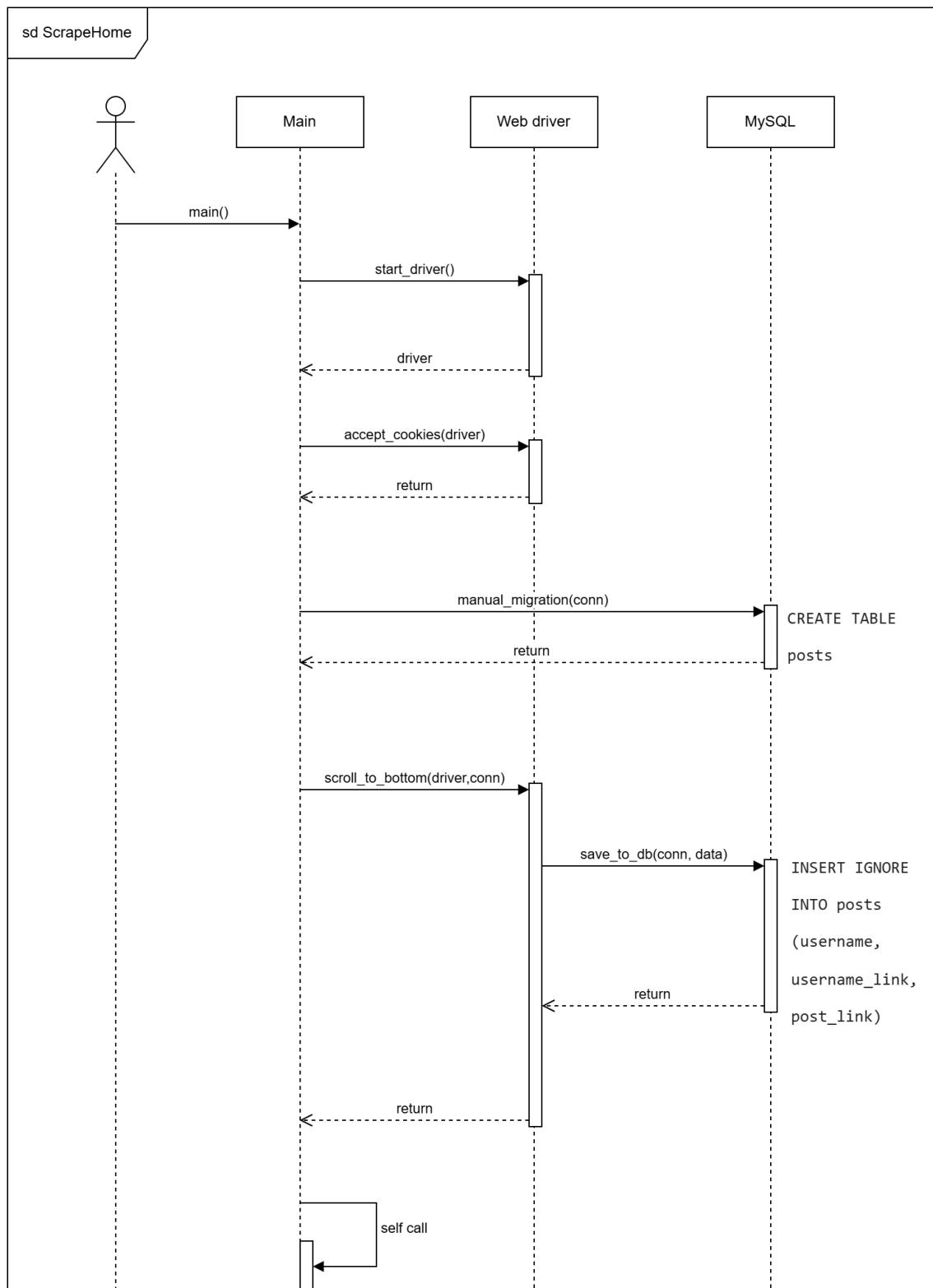
**Figura 4.1:** Diagramma dei componenti relativo al nostro sistema**Figura 4.2:** Classificazione dei diagrammi UML

diagrammi, secondo le specifiche di UML 2, il nome dei diagrammi deve essere preceduto da un prefisso `sd`.

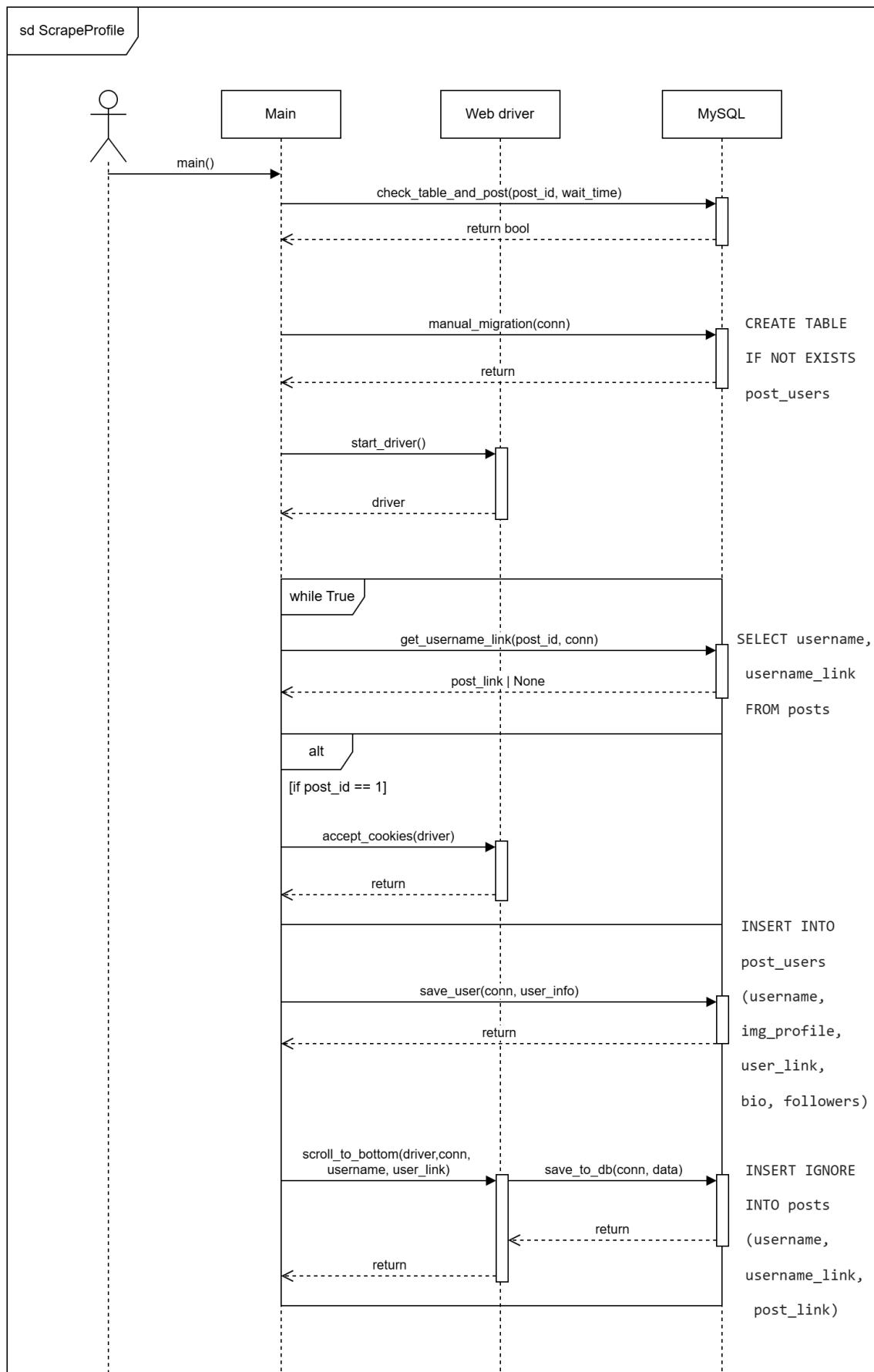
Nella rappresentazione dei diagrammi di sequenza si raffigurano quattro soggetti:

- lo user;
- la funzione `Main` dello script;
- il `driver` con cui Selenium opera;
- il database.

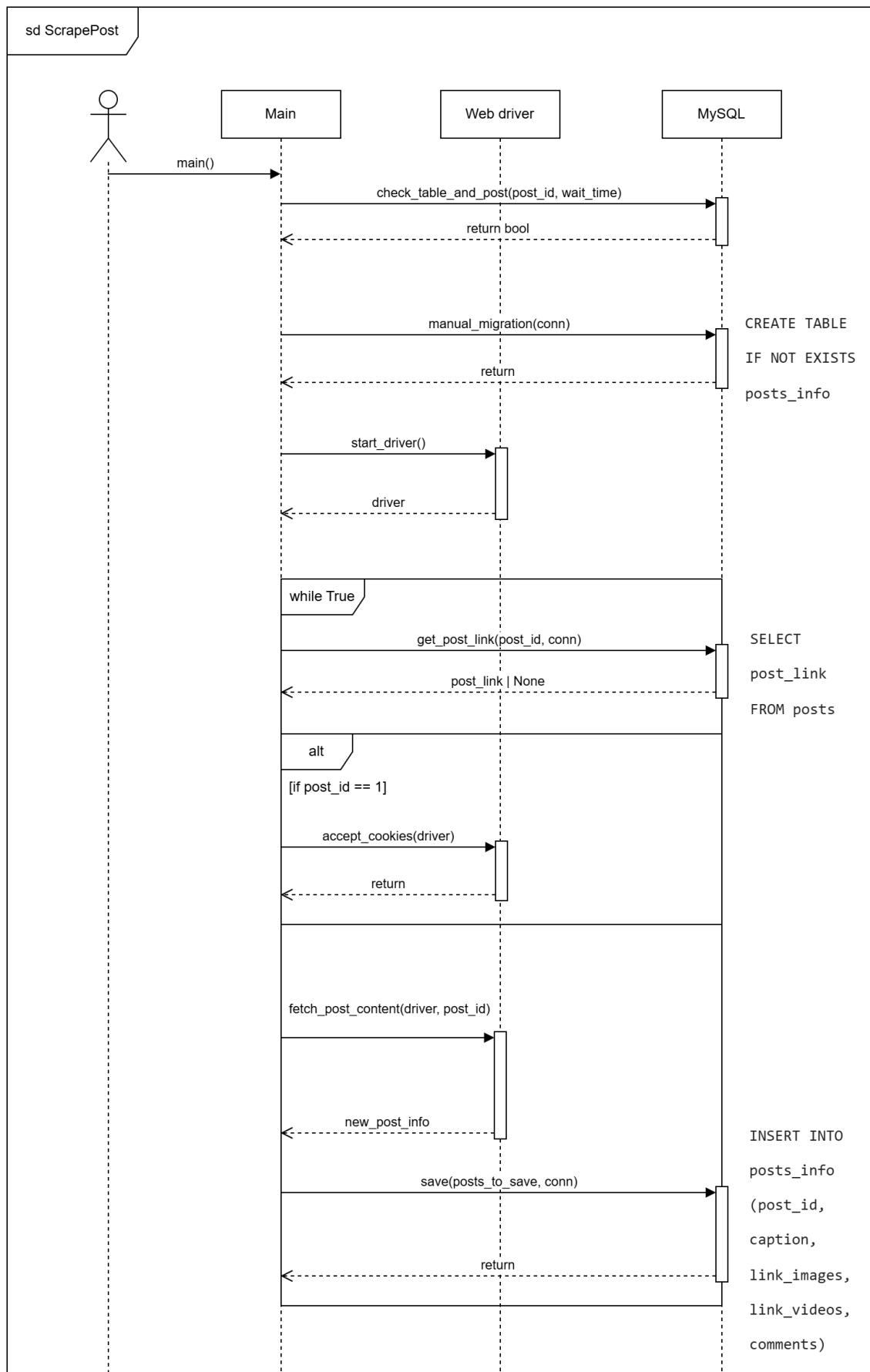
Nelle Figure 4.3, 4.4 e 4.5 si possono osservare i diagrammi di sequenza dei casi d'uso relativi a *Home Scraper*, *Profile Scraper* e *Post Scraper*, rispettivamente.



**Figura 4.3:** Diagramma di sequenza del caso d'uso *Home Scraper*



**Figura 4.4:** Diagramma di sequenza del caso d’uso *Profile Scraper*

Figura 4.5: Diagramma di sequenza del caso d'uso *Post Scraper*

## 4.2 Progettazione delle classi

La progettazione delle classi per il nostro scraper si basa sull'organizzazione dei dati e sulle operazioni richieste da ciascun componente del sistema. Per ciascun tipo di scraper (*Home Scraper*, *Profile Scraper* e *Post Scraper*) sono state definite classi che semplificano la gestione dei dati raccolti e garantiscono la modularità del codice.

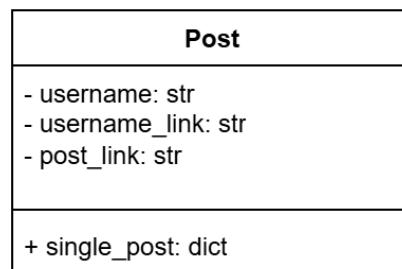
Per lo *Home Scraper*, la classe principale è `Post`, che rappresenta un singolo post raccolto dalla home page di Threads. La classe include i seguenti attributi:

- `username`, cioè il nome dell'utente che ha pubblicato il post;
- `username_link`, ovvero il link al profilo dell'utente;
- `post_link`, ovvero il link specifico al post.

Questi attributi vengono completati tramite il costruttore durante la generazione dell'oggetto istanza della classe.

Il metodo (*getter*) `single_post` restituisce un dizionario contenente le informazioni raccolte, facilitando il salvataggio dei dati nel database. L'idea è quella di costruire una lista di dizionari per le operazioni di scrittura sul database; ogni dizionario è composto da una coppia chiave-valore, in cui la chiave rappresenta un campo della tabella `posts`, mentre il valore è il dato associato alla relativa colonna. Questo approccio è valido per tutte le classi relative al salvataggio dei dati nel database, anche per i restanti scraper. Gli attributi della classe, in questo caso così come negli altri, sono resi privati, in modo tale che la manipolazione degli stessi è protetta.

La classe `Post` viene riutilizzata anche dal *Profile Scraper* per recuperare i post dalla pagina dell'utente. Questa classe è rappresentata in Figura 4.6 secondo la notazione UML. Nella prima sezione orizzontale vi è il nome della classe, nelle restanti due vi sono gli attributi ed i metodi, rispettivamente.

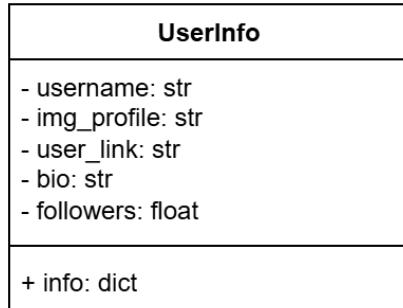


**Figura 4.6:** Classe `Post` secondo la notazione UML

Per il *Profile Scraper* la classe principale è `UserInfo`, la quale rappresenta un utente di Threads e ne organizza i dettagli. Gli attributi principali della classe includono:

- lo `username` dell'utente;
- `img_profile`, ovvero il link all'immagine del profilo dell'utente;
- `user_link`, cioè il link al profilo dell'utente;
- la `bio` dell'utente;
- il numero di `followers` dell'utente.

La classe include il metodo `info` che restituisce un dizionario con tutte le informazioni raccolte, come nel caso precedente, per il salvataggio nel database. La classe è rappresentata in Figura 4.7.



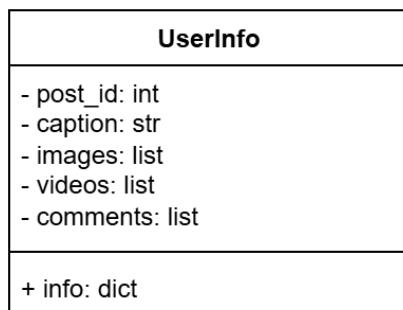
**Figura 4.7:** Classe `UserInfo` secondo la notazione UML

Per il *Post Scraper* è stata progettata la classe `PostInfo`, che gestisce i contenuti specifici di ciascun post, come didascalie, immagini, video e commenti. Gli attributi principali della classe sono:

- `post_id`, il quale rappresenta l'identificativo univoco del post associato nella tabella `posts`;
- `caption`, cioè la didascalia del post;
- `images`, costituita da una lista di link alle immagini del post;
- `videos`, ovvero una lista di link ai video del post;
- `comments`, che memorizza una lista di commenti presenti sotto il post.

Anche questa classe include un *getter*, `info`, che restituisce le informazioni raccolte in un formato facilmente gestibile, cioè un dizionario.

La classe è presentata in Figura 4.8.



**Figura 4.8:** Classe `PostInfo` secondo la notazione UML

La progettazione delle classi è stata pensata per garantire chiarezza, modularità e riutilisabilità del codice. Ogni classe è focalizzata su una specifica entità o operazione dello scraper, permettendo di gestire facilmente i dati raccolti e di integrarli con il database progettato. Questo approccio rende il sistema scalabile e facile da mantenere, facilitando eventuali modifiche future o l'integrazione di nuove funzionalità.

### 4.3 Progettazione delle funzioni

In questa sezione si elencano tutte le funzioni utilizzate, accompagnate da una breve descrizione. Si inizia dalle funzioni utilizzate nel modulo *Home Scraper*. Le principali funzioni utilizzate in tale modulo sono le seguenti:

- `start_driver()`. Questa funzione inizializza il driver di Chrome utilizzando il pacchetto selenium. Si occupa di navigare alla pagina principale di Threads all’indirizzo `https://www.threads.net/?hl=en` e imposta un’attesa esplicita per garantire il caricamento della pagina.
- `accept_cookies(driver)`. La funzione accetta i cookie visualizzati nella pagina iniziale. Utilizza un selettore CSS o un percorso XPath per individuare e cliccare il pulsante corrispondente. Gestisce eventuali eccezioni nel caso in cui il pulsante non sia presente o non sia cliccabile.
- `grab_post(driver, post_number)`. Questa funzione recupera le informazioni relative a un post specifico della pagina, identificato dal parametro `post_number`. Ritorna un dizionario contenente i seguenti campi:
  - `username`: lo username dell’utente del post;
  - `username_link`: il link al profilo dell’utente;
  - `post_link`: il link al contenuto del post.
- `scroll(driver, px)`. La funzione esegue lo scroll verticale della pagina di un numero di pixel specificato dal parametro `px`, utilizzando un comando JavaScript.
- `manual_migration(conn)`. Questa funzione crea la tabella `posts` nel database MySQL, se essa non esiste già. La tabella è strutturata per memorizzare i dati dei post con gestione di duplicati e supporto Unicode.
- `save_to_db(conn, data)`. La funzione salva un insieme di post nella tabella `posts` del database MySQL. Utilizza l’istruzione `INSERT IGNORE` per evitare l’inserimento di duplicati.
- `scroll_to_bottom(driver, conn)`. Questa funzione scorre automaticamente fino alla fine della pagina, raccogliendo i dati relativi ai post tramite la funzione `grab_post`. I dati raccolti vengono salvati nel database a blocchi di dimensione predefinita, utilizzando la funzione `save_to_db`.
- `close_session(driver)`. Questa funzione chiude la scheda del browser attiva e termina la sessione del driver di Chrome.

Le funzioni vengono richiamate dalla funzione principale, che è `main`.  
Brevemente, nel caso del *Profile Scraper* si elencano le seguenti funzioni:

- `get_username_link(post_id, conn)`. Questa funzione restituisce il link del profilo utente, l’immagine del profilo e lo username da un database, una volta dato un `post_id`.
- `start_driver()`. Questa funzione inizializza il driver di Chrome per l’automazione tramite Selenium.
- `accept_cookies(driver)`. Questa funzione accetta automaticamente i cookie utilizzando CSS Selector o XPath.

- `grab_post(driver, post_number, username, username_link)`. Questa funzione estraе le informazioni di un post (username, link utente e link del post) dalla pagina attiva.
- `scroll(driver, px)`. Questa funzione effettua uno scroll verso il basso della pagina di un determinato numero di pixel.
- `manual_migration(conn)`. Questa funzione crea una tabella nel database chiamata `post_users`, se essa non esiste già.
- `save_to_db(conn, data)`. Questa funzione salva una lista di post nel database, ignorando i duplicati.
- `scroll_to_bottom(driver, conn, username, user_link)`. Questa funzione effettua uno scroll automatico fino al fondo della pagina, estraе i post e li salva nel database.
- `check_table_and_post(post_id, wait_time)`. Questa funzione verifica l'esistenza della tabella `posts` e controlla la presenza di un post specifico (`post_id`) con tentativi ripetuti.
- `get_bio(driver)`. Questa funzione restituisce la biografia dell'utente attuale.
- `get_followers(driver)`. Questa funzione restituisce il numero di follower dell'utente attuale.
- `get_profile_pic(driver)`. Questa funzione restituisce il link all'immagine del profilo dell'utente attuale.
- `fetch_user_info(driver, username, user_link)`. Questa funzione estraе le informazioni principali del profilo utente (bio, follower, immagine profilo).
- `save_user(conn, user_info)`. Questa funzione salva le informazioni del profilo utente nella tabella `post_users`.
- `close_session(driver)`. Questa funzione chiude il driver di Chrome aperto per evitare sessioni inattive.
- `main()`. Questa è la funzione principale che gestisce l'intero processo di scraping.

Il *Post Scraper* prevede le seguenti funzioni:

- `get_post_link(post_id, conn)`. Questa funzione recupera il link di un post dal database in base al `post_id`.
- `start_driver()`. Questa funzione inizializza il driver di Chrome e apre la pagina del post.
- `accept_cookies(driver)`. Questa funzione accetta automaticamente i cookie tramite CSS Selector o XPath.
- `get_caption(post)`. Questa funzione estraе e restituisce il testo della didascalia associata al post.
- `get_images(post)`. Questa funzione estraе e restituisce i link alle immagini associate al post.

- `get_videos(post)`. Questa funzione estrae e restituisce i link ai video associati al post.
- `get_comments(post)`. Questa funzione estrae e restituisce i commenti associati al post.
- `fetch_post_content(driver, post_id)`. Questa funzione estrae le informazioni principali di un post, tra cui didascalia, immagini, video e commenti, e le restituisce sotto forma di dizionario.
- `manual_migration(conn)`. Questa funzione crea la tabella `posts_info` nel database, se essa non esiste già.
- `save(posts_to_save, conn)`. Questa funzione salva i dati dei post nella tabella `posts_info` del database, convertendo liste di dizionari in formato JSON.
- `check_table_and_post(post_id, wait_time)`. Questa funzione verifica che la tabella `posts` esista e che contenga un post specifico con `post_id`.
- `close_session(driver)`. Questa funzione chiude la scheda attiva del browser per liberare risorse.
- `main()`. Questa è la funzione principale che gestisce l'intero processo di scraping per i post, comprese le operazioni di connessione al database, di estrazione delle informazioni e di salvataggio dei dati.

## 4.4 Progettazione del database

Il database rappresenta una componente fondamentale per l'architettura dello scraper, consentendo di archiviare, organizzare e gestire i dati raccolti in modo strutturato ed efficiente. La progettazione del database si basa sull'analisi dei requisiti funzionali e tecnici dello scraper, garantendo la coerenza e l'integrità dei dati.

Il sistema è stato progettato per supportare grandi volumi di dati, includendo informazioni relative ai post, agli utenti e ai dettagli specifici dei contenuti. Ogni tabella è stata definita per soddisfare un obiettivo specifico; in particolare:

- la tabella `posts` si occupa di gestire i dati principali dei post raccolti dalla home page;
- la tabella `post_users` memorizza i dettagli degli utenti associati ai post;
- la tabella `posts_info` raccoglie informazioni dettagliate sui contenuti dei singoli post.

Entrando nel dettaglio, la tabella `posts` rappresenta i post raccolti nella home page dello scraper. Ogni record include informazioni principali sul post, come il nome utente, il link al profilo dell'utente, il link al post e la data di creazione del record.

L'`id` del post è una chiave primaria che si autoincrementa ad ogni nuovo record aggiunto. Trattandosi di una grande mole di dati si è scelto il tipo `BIGINT` al posto del classico `INT`. Inoltre, per gestire i duplicati, si è adottato un approccio multiplo, sia a livello di codice che a livello di database, imponendo come `UNIQUE` l'attributo `post_link`; in questa maniera ci si assicura quasi sempre di avere link relativi a post differenti.

Durante le fasi di test è capitato qualche raro caso in cui allo stesso post siano stati associati due indirizzi URL differenti. Questa situazione, che comunque rimane rara, può essere gestita a posteriori nelle successive fasi di ETL o di Business Intelligence.

Nella creazione della tabella da Python si è reso esplicito l’uso di UNICODE in modo da catturare tutti i caratteri presenti, dalle emoticon ai caratteri cinesi, arabi, etc.

Nella Tabella 4.1 è mostrata la struttura della tabella del database.

Campo	Tipo di dato	Descrizione
id	BIGINT AUTO_INCREMENT	Identificativo univoco del post.
username	VARCHAR (255)	Nome utente associato al post.
username_link	TEXT	Link al profilo dell’utente.
post_link	VARCHAR (255)	Link specifico al post (univoco).
created	TIMESTAMP	Data e ora di creazione del record.

**Tabella 4.1:** Struttura della tabella posts

La tabella post\_users memorizza i dettagli degli utenti, tra cui: username, link dell’immagine del profilo, link al profilo, biografia e follower. Anche in questo caso l’id è chiave primaria, si autoincrementa ed è di tipo BIGINT. I follower sono salvati nell’omonimo campo secondo il tipo DOUBLE per gestire la numerazione con la virgola. Infine, l’attributo username è univoco; infatti, viene salvato un utente con lo stesso username una sola volta, mentre i dati utente vengono, poi, aggiornati durante il crawling (da qui l’aggiunta della colonna updated). In Tabella 4.2 viene mostrata la struttura della tabella post\_users.

Campo	Tipo di dato	Descrizione
id	BIGINT AUTO_INCREMENT	Identificativo univoco dell’utente.
username	VARCHAR (255)	Nome utente (univoco).
img_profile	TEXT	URL dell’immagine del profilo.
user_link	TEXT	Link al profilo dell’utente.
bio	TEXT	Biografia dell’utente.
followers	DOUBLE	Numero di follower dell’utente.
created	TIMESTAMP	Data e ora di creazione del record.
updated	TIMESTAMP	Ultima modifica del record.

**Tabella 4.2:** Struttura della tabella post\_users

La tabella posts\_info raccoglie dettagli più approfonditi sui contenuti dei post, come didascalie, link a immagini, video e commenti. Ogni record è associato ad un post tramite un identificativo. Come nei casi precedenti, anche qui l’id è chiave primaria, si autoincrementa ed è di tipo BIGINT. In aggiunta, vi è l’attributo post\_id, che è una chiave esterna alla chiave primaria id della tabella post, in una relazione 1:1. Le immagini ed i video, così come i commenti, sono salvati nel formato JSON perché in Python sono rappresentate come liste di stringhe. Nella Tabella 4.3 viene riportata la struttura della tabella posts\_info.

Campo	Tipo di dato	Descrizione
id	BIGINT AUTO_INCREMENT	Identificativo univoco del record.
post_id	BIGINT	Identificativo del post associato.
caption	TEXT	Didascalia del post.
link_images	JSON	Elenco dei link alle immagini.
link_videos	JSON	Elenco dei link ai video.
comments	JSON	Elenco dei commenti.
created	TIMESTAMP	Data e ora di creazione del record.

**Tabella 4.3:** Struttura della tabella posts\_info

# CAPITOLO 5

---

## Implementazione dello scraper

---

*In questo capitolo si descrive l'implementazione dei tre scraper sviluppati sulla piattaforma Threads. Gli scraper in questione sono l'Home Scraper, il Profile Scraper e il Post Scraper. Ogni scraper è progettato per svolgere un ruolo specifico e complementare all'interno del sistema, consentendo di raccogliere dati organizzati in modo efficiente e accurato. L'Home Scraper è responsabile di navigare nella home page della piattaforma Threads per raccogliere informazioni di base sui post visibili. Il Profile Scraper esplora i profili degli utenti recuperati dall'Home Scraper, estraendo dettagli, come la biografia, il numero di follower, il link all'immagine del profilo e l'elenco dei post pubblicati dall'utente. Infine, il Post Scraper approfondisce l'analisi dei singoli post, raccogliendo contenuti multimediali (immagini e video), didascalie e commenti, e salvando queste informazioni nel database.*

### 5.1 Home Scraper

In questa sezione viene trattato lo scraper principale. In particolare, lo scraper si occuperà di recuperare informazioni fondamentali dalla home page di Threads.

Lo scopo di questo primo scraper è quello di recuperare i dati che poi verranno utilizzati dagli altri scraper per continuare il lavoro di crawling sulla piattaforma social. Come anticipato, per ogni post nella home page vengono recuperati il link, lo username dell'autore del post ed il link alla pagina dello username; questi dati vengono salvati nella tabella posts.

Per questo scraper si andrà nel dettaglio delle funzioni; mentre per i restanti due, al fine di evitare di essere ripetitivi, verranno segnalate unicamente le funzioni che denotano una differenza significativa, tralasciando quelle sostanzialmente simili.

#### 5.1.1 Installare ed avviare il driver

Innanzitutto, come spiegato nel capitolo precedente, Selenium ha bisogno del driver per poter interfacciare e controllare in modo autonomo il browser. In questo caso si utilizza Chrome come browser di riferimento. Il driver va, quindi, scaricato e inizializzato.

Nel Listato 5.1 è presentato il codice per installare il driver. Questa operazione può essere effettuata una singola volta, evitando la necessità di dover scaricare il driver ad ogni esecuzione dello script. Inoltre, è possibile anche definire un path specifico in cui scaricare il webdriver. Nell'esempio viene stampato sul video il percorso sulla macchina in cui viene scaricato il driver.

---

```
# Chrome driver installation
CHROMEDRIVER_PATH = ChromeDriverManager().install()

print(CHROMEDRIVER_PATH) # Chrome driver path
```

---

**Listing 5.1:** Installazione del driver necessario per il controllo del browser da parte di Selenium

A questo punto, si avvia il driver tramite la funzione `start_driver()` definita nel Listato 5.2. La funzione avvia il driver e apre Chrome alla pagina principale di Threads. Infine, restituisce il driver stesso, il quale è necessario per le ulteriori operazioni da svolgere.

---

```
def start_driver():
    """ Initialize Chrome driver and open main page """

    # Threads URL
    URL = "https://www.threads.net/?hl=en"
    # Chrome driver PATH
    CHROMEDRIVER_PATH = r"REDACTED"

    # Initialize the Chrome driver
    driver = webdriver.Chrome(service=Service(CHROMEDRIVER_PATH))

    # Navigate to the URL + explicit wait
    driver.get(URL)
    time.sleep(2)

    return driver
```

---

**Listing 5.2:** Avvio del driver

A volte si farà uso della keyword `REDACTED` all'interno del codice per evitare di mostrare stringhe eccessivamente lunghe, come può essere il caso dei selettori.

### 5.1.2 Accettare i cookie

Un'operazione richiesta sulla pagina di Threads, almeno per la prima volta, è quella di accettare o meno i cookie. Il bot, simile ad un operatore umano, seleziona il pulsante relativo all'accettazione dei cookie. Questa funzione, rappresentata nel Listato 5.3, verrà poi riutilizzata negli altri due script di crawling.

Nel codice è stata utilizzata la tecnica dell'*implicit wait* al fine di caricare correttamente il pulsante da cliccare per accettare i cookie. Nello specifico, sono stati utilizzati due selettori per aumentare la ridondanza, ovvero un selettore basato su CSS e uno su XPath. Nel caso fallisse il primo, si ritenta con il secondo; se anche questo fallisce, il driver viene chiuso. I metodi per recuperare, rispettivamente, il percorso del selettore CSS o del selettore XPath sono stati presentati nei capitoli precedenti.

### 5.1.3 Recuperare un post

In questa parte, prima di passare al codice, si mostra la metodologia utilizzata per recuperare le informazioni dalla pagina. L'approccio viene mostrato in questa particolare applicazione; tuttavia può essere generalizzato e riapplicato anche in contesti differenti.

La parte centrale dello scraper è quella che si occupa di recuperare le informazioni presentate nella descrizione dello scraper stesso. Per fare questo, lo scraper si basa unicamente sugli XPath. In particolare, viene recuperato prima l'XPath relativo ad un singolo post, ovvero una sorta di *"contenitore"* del post; successivamente, da quest'ultimo, si estrapolano i

---

```

def accept_cookies(driver):
    """ accept cookies """

    # CSS SELECTOR
    ACCEPT_COOKIE_SELECTOR = 'REDACTED'
    # XPATH
    ACCEPT_COOKIE_XPATH = "//div[contains(text(), 'Allow all cookies')]""

    # TIMEOUT (15 seconds)
    TIMEOUT = 15

    try:
        # implicit wait
        button = WebDriverWait(driver, TIMEOUT).until(
            EC.presence_of_element_located((By.CSS_SELECTOR, ACCEPT_COOKIE_SELECTOR))
        )
        # click to accept cookies with CSS_SELECTOR
        button.click()
        print("CSS cookies button click\n")

    except TimeoutException:
        try:
            button = WebDriverWait(driver, TIMEOUT).until(
                EC.presence_of_element_located((By.XPATH, ACCEPT_COOKIE_XPATH))
            )
            button.click() # with XPATH
            print("XPATH cookies button click\n")

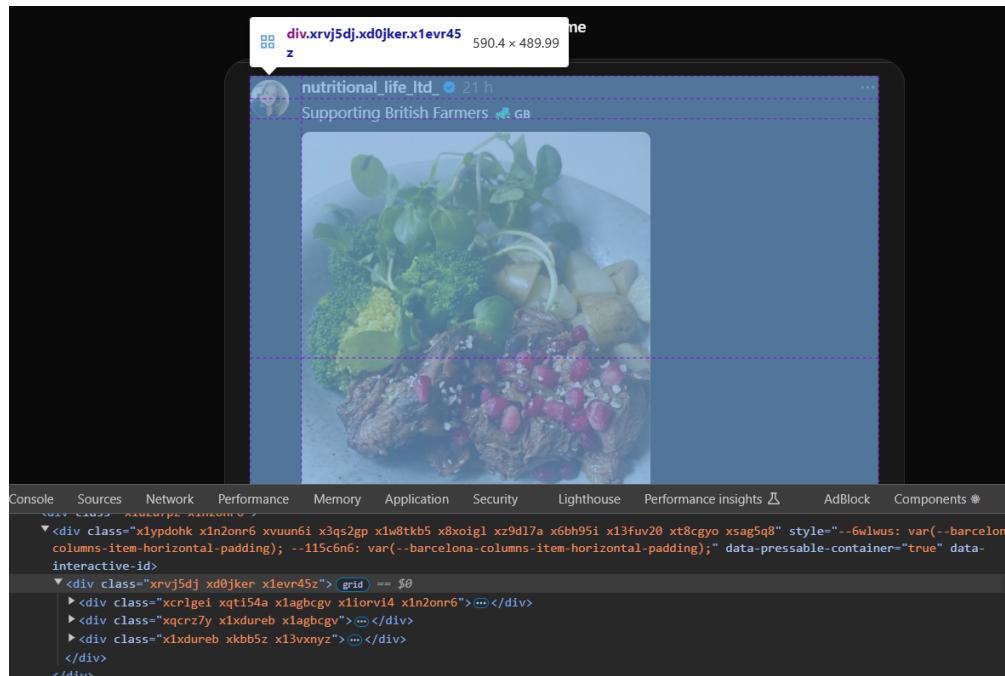
        except TimeoutException:
            print("ERROR: Cookies button missing")
            driver.quit()

```

---

**Listing 5.3:** Accettazione dei cookie

dati di interesse. Il "contenitore" del post a cui si fa riferimento è visibile in Figura 5.1, dove la riga HTML evidenziata rispecchia l'elemento del DOM selezionato, ovvero l'intero post.

**Figura 5.1:** "Container" del post

Questa soluzione è dovuta ad una scelta implementativa fatta a priori. Nel recuperare i

vari post si sarebbe potuto procedere in due modi alternativi, cioè scegliere di utilizzare il metodo `find_elements`, e quindi recuperare tutti i post attualmente visibili, oppure utilizzare `find_element` specializzandosi su un unico post. La prima strada sembra quella più ovvia da percorrere, ma presenta alcune importanti problematiche che hanno portato a scartarla. Infatti, utilizzare il metodo `find_elements` recupera tutti i post attualmente visibili sulla pagina web, ma, allo stesso tempo, dopo che si effettua uno scorrimento della home page di Threads, vengono recuperati sia i nuovi post, sia quelli vecchi già salvati. Questo perché tale metodo recupera tutti i post visibili in un dato istante sulla pagina. Dunque, se da un lato questa tecnica è evidentemente più onerosa computazionalmente, perché recupera tutti i post, dall'altro lato impone anche il problema di gestire i duplicati.

Un'alternativa a queste problematiche è quella di utilizzare il metodo `find_element`. Infatti, da un'attenta analisi del DOM di Threads, è possibile osservare come ogni post abbia, rispetto ad un elemento `<div>`, un numero associato secondo un ordine crescente. Quindi, si procede recuperando l'XPath di ogni singolo post, come evidenziato di Figura 5.1; da questo, poi, si estraggono i dati di interesse.

A supporto della nostra analisi, si osserva effettivamente come, partendo dal primo post della home page e scorrendo un singolo post alla volta, questi effettivamente siano numerati sequenzialmente, come evidenziato dagli XPath associati al "contenitore" di ogni post nel Listato 5.4.

---

```
XPath 1ST POST: //*[@id="barcelona-page-layout"]/div/div/div[2]/div[1]/div[1]/div/div/div[1]/div[1]/div/div/div/div
XPath 2ND POST: //*[@id="barcelona-page-layout"]/div/div/div[2]/div[1]/div[1]/div/div/div[1]/div[2]/div/div/div/div
XPath 3RD POST: //*[@id="barcelona-page-layout"]/div/div/div[2]/div[1]/div[1]/div/div/div[1]/div[3]/div/div/div/div
XPath NTH POST: //*[@id="barcelona-page-layout"]/div/div/div[2]/div[1]/div[1]/div/div/div[1]/div[N]/div/div/div/div
```

---

**Listing 5.4:** XPath dei post presenti nella home page a partire dal primo

Si osserva, quindi, che gli XPath dei post sono quasi uguali tra di loro, ad eccezione di un elemento `div[N]`, che differisce a seconda della posizione del post nella home page di Threads. Nell'esempio di Figura 5.1 l'XPath è il numero undici; infatti, in questo particolare caso, il post è in undicesima posizione (Listato 5.5).

---

```
//*[@id="barcelona-page-layout"]/div/div/div[2]/div[1]/div[1]/div/div/div[1]/div[11]/div/div/div/div
```

---

**Listing 5.5:** XPath del post di Figura 5.1

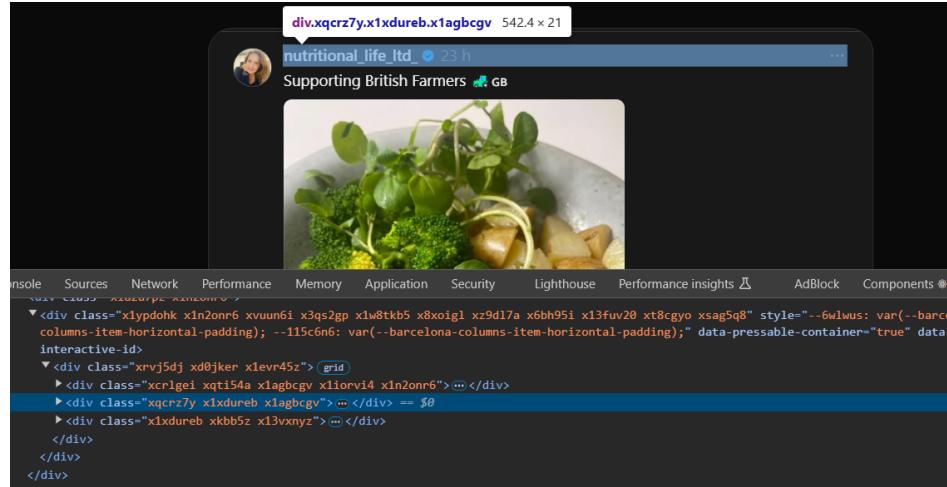
Ora, sempre facendo riferimento alla Figura 5.1, si può osservare come all'interno del cosiddetto "contenitore" di un singolo post ci sono tre elementi `div`. Proprio da questi, in particolare dal secondo, è possibile recuperare informazioni associate al singolo post, quali:

- lo username dell'utente;
- il link al profilo dell'utente;
- il link del post.

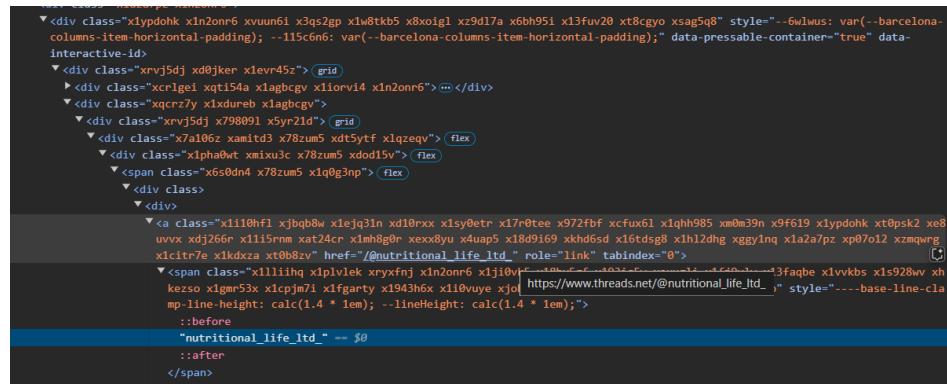
È molto facile perdersi nel DOM da analizzare; tuttavia, prendendo i giusti punti di riferimento, si riesce a mantenere un certo orientamento.

Si nota, quindi, che lo username fa parte del secondo elemento `div` dei tre elencati prima, come si può osservare in Figura 5.2.

L'elemento `div` in esame può essere ulteriormente espanso per trovare sia lo username testuale che il link associato alla pagina del profilo relativa allo username. Infatti, si può

**Figura 5.2:** XPath generale dello username

osservare in Figura 5.3, che l'elemento evidenziato in blu indica lo username in formato testuale, mentre, poco più sopra, l'elemento evidenziato in grigio indica il link relativo al profilo dell'utente.

**Figura 5.3:** XPath dello username e del link al profilo associato

A questo punto si possono recuperare gli XPath associati. In particolare, nei Listati 5.6 e 5.7, si elencano gli XPath in questione, ma relativi ai primi due post della home page.

---

```

1ST POST :
//*[@id="barcelona-page-
layout"]//div/div/div[2]/div[1]/div[1]/div/div/div[1]/div[1]/div/div/div/div[2]/div/div/div[1]/div/span[1]/div/div/a/span/
```

```

2ND POST :
//*[@id="barcelona-page-
layout"]//div/div/div[2]/div[1]/div[1]/div/div/div[1]/div[2]/div/div/div/div[2]/div/div/div[1]/div/span[1]/div/div/a/span/
```

---

**Listing 5.6:** XPath associati alla stringa dello username

Nel Listato 5.6 si nota come l'XPath dello username, in realtà, espanda (in verde) quanto già visto nel Listato 5.4 per il singolo "contenitore" del post. A questo punto si può pensare di procedere per XPath relativi, in modo tale da semplificare la scrittura degli stessi. Una volta fissato l'XPath associato al contenitore del post, l'XPath relativo allo username è pressoché identico per tutti i post, perché la differenza risiede unicamente nella posizione del post stesso. Lo stesso ragionamento è valido per l'XPath associato al link del profilo utente (Listato 5.7).

---

```
1ST POST :
//*[@id="barcelona-page-
layout"]//div/div/div[2]/div[1]/div/div/div[1]/div[1]/div/div/div/div[1]/div[2]/div/div/div[1]/div/span[1]/div/div/a
```

---

```
2ND POST :
//*[@id="barcelona-page-
layout"]//div/div/div[2]/div[1]/div/div/div[1]/div[2]/div/div/div/div[2]/div/div/div[1]/div/span[1]/div/div/a
```

---

**Listing 5.7:** XPath associati al link alla pagina utente

Per essere più chiari, si faccia riferimento al Listato 5.8; è chiaro come scrivere gli XPath relativi risulti vantaggioso da un punto di vista implementativo, dato che la gestione del cambiamento dell'XPath è rimandata al contenitore del post.

Questo approccio suggerisce il seguente modo di procedere: si recupera ogni post singolarmente facendo riferimento all'XPath generale del "contenitore"; successivamente, tramite gli XPath relativi che sono statici, si recuperano i dati di interesse.

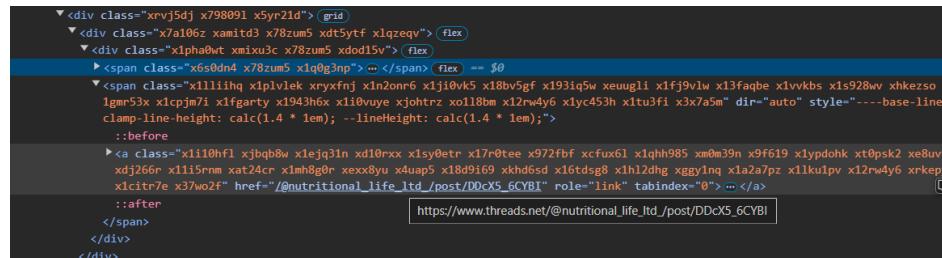
---

```
USERNAME_REL_XPATH      = './/div[2]/div/div[1]/div/span[1]/div/div/a/span'
USERNAME_LINK_REL_XPATH = './/div[2]/div/div[1]/div/span[1]/div/div/a'
```

---

**Listing 5.8:** XPath relativi per lo username e il link al profilo utente

L'ultimo elemento chiave da recuperare è il link relativo al post che si sta analizzando. Come nel caso dello username, si può far riferimento al secondo elemento div di Figura 5.2. Come illustrato in Figura 5.4, la differenza con gli XPath relativi visti in precedenza sta nel sezionare correttamente l'ultimo elemento span, ovvero prendendo il secondo dei due span evidenziati nella Figura.

**Figura 5.4:** XPath associato al link del post

Gli XPath di interesse, per il primo e l'undicesimo post nella home page, sono mostrati nel Listato 5.9. Quindi, l'XPath relativo può essere implementato in una variabile in Python, come mostrato nel Listato 5.10.

---

```
1ST POST :
//*[@id="barcelona-page-
layout"]//div/div/div[2]/div[1]/div/div/div[1]/div[1]/div/div/div/div[2]/div/div/div[1]/div/span[2]/a
```

---

```
11TH POST :
//*[@id="barcelona-page-
layout"]//div/div/div[2]/div[1]/div/div/div[1]/div[11]/div/div/div/div[2]/div/div/div[1]/div/span[2]/a
```

---

**Listing 5.9:** XPath associati al link alla pagina utente

Stabiliti gli XPath, si può passare al recupero delle informazioni. Prima di questo, però, come anticipato nel capitolo sull'analisi dei requisiti, viene definita una classe per la costruzione degli oggetti da salvare nel database. Nel Listato 5.11 si riporta la classe Post utilizzata a tale scopo. La classe viene utilizzata per costruire una lista di dizionari necessari per riempire la tabella posts di MySQL.

---

```
POST_LINK_REL_XPATH      = './div[2]/div/div[1]/div/span[2]/a'
```

---

**Listing 5.10:** XPath relativo per il link del post

---

```
class Post:
    def __init__(self, username, username_link, post_link):
        self._username = username
        self._username_link = username_link
        self._post_link = post_link

    @property
    def single_post(self):
        return {
            "username": self._username,
            "username_link": self._username_link,
            "post_link": self._post_link
        }
```

---

**Listing 5.11:** Implementazione della classe Post

La funzione `grab_post` si occupa di recuperare effettivamente, tramite gli XPath sopra definiti, le informazioni necessarie (Listato 5.12). Ci sono due considerazioni importanti da fare per questa parte. La prima riguarda l'uso della `f-string` per definire l'XPath del contenitore del post. Infatti, in ingresso alla funzione viene dato il `post_number`, che servirà a modificare la stringa al fine di recuperare il post specifico che si vuole cercare. La seconda considerazione riguarda la dimensione del post. Se il post viene trovato nella home page, si recupera la sua dimensione fisica (l'altezza in pixel) e la si ritorna, insieme al dizionario di elementi da salvare, alla funzione chiamante sotto forma di tupla.

---

```
def grab_post(driver, post_number):
    """ Grab post and return: username, username link and post link """

    POST_XPATH = f'''//[@id="barcelona-page-layout"]/div/div/div[2]/div[1]/div[1]
    /div/div/div[1]/div[{post_number}]/div/div/div'''

    # retrieves a specific post on the page
    post = driver.find_element(By.XPATH, POST_XPATH)

    if post:
        post_height = post.size['height'] # post height
        try:
            # username visible text
            username = post.find_element(By.XPATH, USERNAME_REL_XPATH).text
            # username link profile
            username_link = post.find_element(By.XPATH,
                → USERNAME_LINK_REL_XPATH).get_attribute('href')
            # post content
            post_link = post.find_element(By.XPATH,
                → POST_LINK_REL_XPATH).get_attribute('href')
            # new instance of the class Post
            new_post_obj = Post(username, username_link, post_link)
            # create dictionary
            new_post = new_post_obj.single_post

        except Exception as e:
            print("Exception in grab_post function:", e)

    return (new_post, post_height)
```

---

**Listing 5.12:** Funzione per recuperare i dati da ogni post della home page

### 5.1.4 Migrazione e Salvataggio in MySQL

In questa parte si analizzano due funzioni che si occupano, rispettivamente, di creare la tabella posts (Listato 5.13) e di salvare i dati raccolti dalla funzione `grab_post` in questa tabella (Listato 5.14).

---

```
def manual_migration(conn):
    """ Create the 'posts' table only if it doesn't exist """

    try:
        with conn.cursor() as cursor:

            # SQL query to check if the table exists
            check_table_query = """
                SELECT COUNT(*)
                FROM information_schema.tables
                WHERE table_schema = DATABASE() AND table_name = 'posts';
            """

            cursor.execute(check_table_query)
            table_exists = cursor.fetchone()[0]

            if table_exists:
                print("'posts' table already exists. No changes made.")

            else:
                # SQL query to create the posts table with Unicode support
                create_table_query = """
                    CREATE TABLE posts (
                        id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
                        username VARCHAR(255) CHARACTER SET utf8mb4 NOT NULL,
                        username_link TEXT CHARACTER SET utf8mb4 NOT NULL,
                        post_link VARCHAR(255) CHARACTER SET utf8mb4 NOT NULL,
                        UNIQUE (post_link),
                        created TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                    ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
                """

                # execute query
                cursor.execute(create_table_query)
                print('"posts" table created')

            # Commit the transaction
            conn.commit()

    except MySQLError as e:
        print('Mysql Error:', e)
```

---

**Listing 5.13:** Migrazione della tabella posts

La funzione `save_to_db` (Listato 5.14) si occupa di salvare la lista di dizionari nel database, prima trasformandola in una lista di tuple (*list comprehension*), e poi salvando i dati tramite il metodo `executemany`. Quest'ultimo restituisce, a sua volta, il numero di righe correttamente inserite nel database, le quali vengono "loggate" dallo script, in modo tale da tenere traccia di quanti dati siano stati effettivamente salvati nel database.

### 5.1.5 Scorrimento della pagina

Il fulcro dello scraper è definito dalla funzione che scorre, fino alla fine, la home page di Threads e recupera le informazioni. Innanzitutto, si definisce una funzione di appoggio, capace di scorrere la pagina di una quantità definita ogni volta che essa viene chiamata (Listato 5.15).

---

```

def save_to_db(conn, data):
    """Save data into MySQL with duplicate handling using INSERT IGNORE"""

    try:
        with conn.cursor() as cursor:
            # Create an SQL insert statement with INSERT IGNORE to skip duplicates
            sql = """
                INSERT IGNORE INTO posts (username, username_link, post_link)
                VALUES (%s, %s, %s)
            """

            # Prepare the data as a list of tuples
            values = [(post['username'], post['username_link'], post['post_link']) for post
                      in data]

        try:
            # Execute the insert statement for all data at once
            rows_inserted = cursor.executemany(sql, values)

            # Commit the transaction
            conn.commit()
            print(f'{rows_inserted} rows saved in the database correctly')

        except MySQLError as e:
            print('Mysql insert query Error:', e)

    except MySQLError as e:
        print('Mysql Error:', e)

```

---

**Listing 5.14:** Salvataggio dei dati nella tabella posts

---

```

def scroll(driver, px):
    """Scroll down by the height of a post"""

    js = f"window.scrollBy(0, {px});"
    driver.execute_script(js)

```

---

**Listing 5.15:** Scorrimento della pagina di una quantità definita

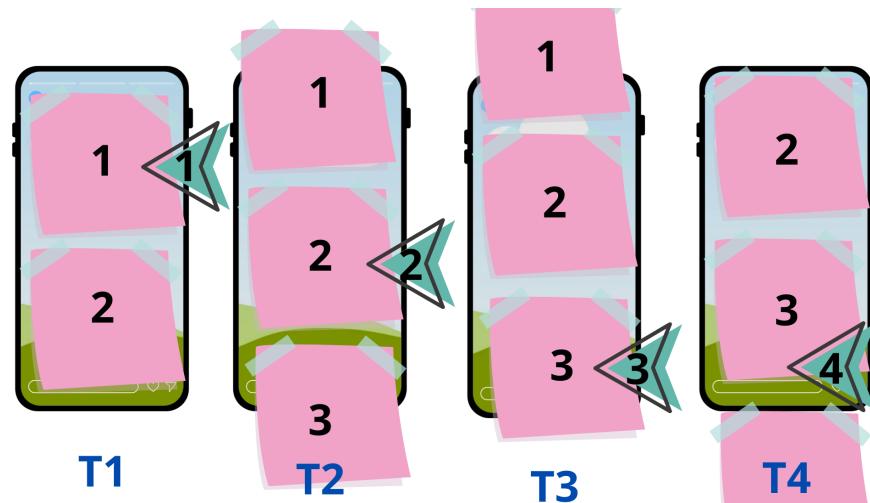
La quantità di pixel da scorrere, come si vedrà nella funzione `scroll_to_bottom`, è definita come il massimo tra venti pixel e l'altezza del post diminuita di venti pixel. Questi dati empirici sono frutto di test e di una scelta implementativa fondamentale. Invece di scorrere di una quantità fissa, si è scelto di scorrere di una quantità di poco inferiore allo spazio verticale occupato dal post. Allo stesso tempo, per non ricadere in situazioni in cui si scorra la pagina di una quantità infinitesimale, ovvero quando il post è di dimensioni molto vicine a venti pixel, si è scelta una soglia sotto la quale non si può scendere, cioè venti pixel. In altre parole, la quantità di pixel da scorrere è pari al massimo tra l'altezza del post diminuita di venti pixel e venti pixel.

Questa scelta implementativa può portare ad un'unica situazione relativa alla mancanza del post cercato, ovvero il caso in cui il post non è ancora stato caricato. In altri termini, essendo lo scorrimento più lento della capacità dello scraper di recuperare i post, lo scraper si troverà, a volte, al limite del caricamento. Nel caso in cui un post non è reperibile è perché non si è scesi giù abbastanza nella pagina; quindi, si continua a scorrere finché il post non viene correttamente caricato e recuperato. Questa strategia permette di recuperare tutti i post della home page, senza replica e senza escluderne nessuno.

Questo concetto può essere meglio compreso con un esempio. Si consideri la situazione ipotetica in cui tutti i post abbiano la stessa dimensione e che lo scorrimento sia impostato pari ad un terzo del post recuperato. Ora, si faccia riferimento alla Figura 5.5; si immagini

che lo schermo dello smartphone sia l'effettiva disponibilità grafica della pagina, e che i post numerati (in rosa) siano effettivamente dei post che scorrono nel tempo. La freccia verde acqua, invece, rappresenta lo scraper in cerca del post specificato nella didascalia.

Al tempo T1 lo scraper recupera il post numero 1 e scorre di una quantità piccola, pari circa ad un terzo del post. Questo viene ripetuto per gli istanti T2 e T3. All'istante T4 però, avendo scrollato di una quantità insufficiente a caricare il post numero 4, lo scraper non lo trova, a meno che non scorra nuovamente, cioè finché non viene renderizzato. Per questo motivo, l'unico caso in cui un post non viene recuperato è perché non è stato ancora visualizzato nel browser. Questa implementazione permette di ricadere in questa unica casistica di incertezza nella gestione del recupero dei post, evitando di scorrere troppo, perdendo alcuni post, oppure troppo poco, cioè rallentando notevolmente il processo di crawling.



**Figura 5.5:** Esempio di scorrimento della home page e di recupero dei post

La funzione `scroll_to_bottom` rappresenta il fulcro dello script. Le sue operazioni possono essere riassunte in un ciclo infinito che si occupa di:

- recuperare il post con le informazioni associate, quali username, link al profilo dello username e link del post;
- recuperare l'altezza in pixel del post;
- costruire un oggetto in base alle informazioni ricavate dal post;
- aggiungere questo oggetto ad una lista;
- se la lista ha una dimensione maggiore di una soglia predeterminata, salvare i dati raccolti nel database e svuotare la lista, in modo tale da liberare la memoria;
- scorrere la schermata di una quantità definita secondo le metodologie descritte precedentemente;
- incrementare la variabile contatore associata al post (successivo) da ricercare;
- se lo scorrimento è nullo, e quindi si è raggiunta la fine della pagina, uscire dal loop e salvare gli eventuali dati rimanenti.

Nel Listato 5.16 viene riportata la funzione `scroll_to_bottom`.

---

```

def scroll_to_bottom(driver, conn):
    """ auto-scroll to the bottom of the page and save posts to db """

    # post number to retrieve
    post_number = 1
    # number of posts to be saved together
    post_no_to_save = 50

    # list of posts
    list_to_push_to_db = []

    old_position = driver.execute_script("return window.pageYOffset;")
    px_to_scroll = 0

    while True:

        # Wait for the page to load
        #time.sleep(3)

        try:
            # retrieve post
            post, post_height = grab_post(driver, post_number)
            # Add post to list only if valid
            if post:
                list_to_push_to_db.append(post)
                print("post added number:", post_number)
                print("post added to list:", post)

            # save to db in batches
            if len(list_to_push_to_db) >= post_no_to_save:
                save_to_db(conn, list_to_push_to_db)
                list_to_push_to_db = [] # free memory

            # Scroll a little less than the height of the post
            px_to_scroll = max(20, abs(post_height - 20))

            # update post position
            post_number += 1

        # post not found
        except NoSuchElementException:
            print(f"Post number {post_number} not found")
            continue

        except Exception as e:
            print(f"Post number {post_number} Error: {e}")
            # skip post
            post_number += 1
            continue

        finally:
            # scroll down
            print("scroll down")
            scroll(driver, px_to_scroll)
            time.sleep(3)

            # update position
            new_position = driver.execute_script("return window.pageYOffset;")

            # Exit the loop if the page hasn't scrolled, meaning end of page
            if new_position == old_position:
                print('End of Page. Restarting...')
                break

            # update new position
            old_position = new_position

    # Save remaining posts to the database
    if list_to_push_to_db:
        print(f"Saving remaining {len(list_to_push_to_db)} posts to the database...")
        save_to_db(conn, list_to_push_to_db)

```

---

**Listing 5.16:** Funzione che scorre la home page e recupera i dati necessari dai post

### 5.1.6 Main function

La funzione principale che coordina tutto lo *Home Scraper* è rappresentata nel Listato 5.17.

---

```
def main():
    driver = None

    try:
        # connection to db
        conn = pymysql.connect(
            host='localhost',
            user='mysql',
            password='mysql',
            database='mysql',
            port=3306
        )

        # start web driver
        driver = start_driver()
        # accept initial cookies
        accept_cookies(driver)

        print("Website:", driver.title) # fetches the title of the current webpage
        print("Current URL:", driver.current_url)

        # create posts table if not exists
        manual_migration(conn)

        # infinite scroll page
        scroll_to_bottom(driver, conn)

    except Exception as e:
        print(f"Error during scraping: {e}")

    finally:
        # close sessions and driver
        if driver:
            close_session(driver)
            print("Web driver closed")

        if conn:
            conn.close()
            print("Database connection closed")
```

---

**Listing 5.17:** Funzione principale dello *Home Scraper*

Tale funzione, che viene richiamata in un loop infinito, esegue le seguenti azioni:

- crea il connettore per il database;
- avvia il driver;
- accetta i cookie;
- crea, se necessario, la tabella posts;
- richiama la funzione per scorrere la home page di Threads.

Nel caso in cui si arrivi alla fine della pagina, lo scraper si riavvia automaticamente creando una nuova sessione.

## 5.2 Profile Scraper

Il *Profile Scraper* si occupa sia di recuperare informazioni sul singolo utente e salvarle nella tabella `post_users`, sia di inserire i post pubblicati dall’utente nella tabella `posts`. La funzione principale dello scraper è illustrata nel Listato 5.18.

---

```

def main():
    """ main function """

    post_id = 1
    try:
        # connection to db
        conn = pymysql.connect(
            host='localhost',
            user='mysql',
            password='mysql',
            database='mysql',
            port=3306
        )

        # Ensure the table "posts" exists and contains the post_id == 1
        if not check_table_and_post(post_id, 60):
            print("Required table or post not found. Exiting.")
            return

        # posts_info table creation
        manual_migration(conn)

        # start driver
        driver = start_driver()

        while True:
            try:
                # Attempt to get the user link
                (username, user_link) = get_username_link(post_id, conn)

                if user_link:
                    print("\nUser Profile:", user_link)
                    # open user's profile page
                    driver.get(user_link)

                    # first user
                    if post_id == 1:
                        accept_cookies(driver)

                    # fetch user info
                    user_info = fetch_user_info(driver, username, user_link)
                    # save user info
                    save_user(conn, user_info)
                    # scroll page, fetch posts and add to the list of post to save in the db
                    scroll_to_bottom(driver, conn, username, user_link)
                    time.sleep(5)

            except Exception:
                print(f'Post n.{post_id} not found in "posts" db:')

            finally:
                post_id += 1 # Increment post_id to fetch the next post

    except Exception as e:
        print('Main Error:', e)

    finally:
        close_session(driver)
        conn.close()

```

---

**Listing 5.18:** Funzione principale del *Profile Scraper*

In questa sezione e nella successiva si avrà un approccio differente rispetto alla precedente; invece di elencare le funzioni una per una, si parte direttamente dal `main` per descrivere il funzionamento generale; successivamente si entra nel dettaglio di qualche funzione propria dello script in esame.

Il *Profile Scraper* attende che sia stata creata la tabella `posts` e che vi sia il primo elemento con `id` uguale ad uno. Questo è permesso dalla funzione `check_table_and_post`, che effettua tentativi multipli distanziati temporalmente.

Una volta che si è verificata la condizione necessaria, si effettua una migrazione manuale, tramite la funzione `manual_migration`, per la creazione della tabella appropriata (`post_users`). Quindi, si inizializza il driver.

A questo punto si entra in un loop, il cui primo passo è recuperare lo username ed il link associato alla pagina utente, tramite la funzione `get_username_link`, la quale riceve in ingresso l'`id` del post ricercato e il connettore (`conn`) al database. Questa funzione è illustrata nel Listato 5.19.

---

```
def get_username_link(post_id, conn):
    """ get a single user link profile, ima profile and username from the db """
    try:
        with conn.cursor() as cursor:
            # SQL query to fetch the post link by ID
            sql = "SELECT username, username_link FROM posts WHERE id = %s"
            cursor.execute(sql, (post_id,))
            result = cursor.fetchone()

        if result:
            return result
        else:
            return None

    except MySQLError as e:
        print('Mysql Connection Error:', e)
```

---

**Listing 5.19:** Funzione per recuperare il link al profilo di un utente

Successivamente si apre il driver al link specificato e, unicamente per il primo utente, si accettano i cookie. Selenium provvederà automaticamente a salvare questi ultimi nel driver per le successive operazioni.

A questo punto la funzione `fetch_user_info` si occupa di recuperare le informazioni relative all'utente (Listato 5.20).

---

```
def fetch_user_info(driver, username, user_link):
    """ catch user profile important info and return a dict """
    time.sleep(2)

    # get user bio
    bio = get_bio(driver)

    # get user followers
    followers = float(get_followers(driver))

    # get user profile pic
    img_profile = get_profile_pic(driver)

    # build class
    user = UserInfo(username, img_profile, user_link, bio, followers)

    return user.info
```

---

**Listing 5.20:** Recupero delle informazioni su un utente

Tale funzione si appoggia a varie funzioni per recuperare le singoli informazioni tramite gli XPath. Ad esempio, il recupero della biografia di un utente è illustrato nel Listato 5.21. Per brevità si ometteranno le altre funzioni di appoggio.

---

```
def get_bio(driver):
    """ return user bio """

    try:
        return driver.find_element(By.XPATH, BIO_XPATH).text

    except NoSuchElementException:
        print("Bio not found!")
        return None
```

---

**Listing 5.21:** Recupero della biografia di un utente

A questo punto, una volta recuperate le informazioni, si fa uso del costruttore della classe `UserInfo` con le stesse modalità viste per lo *Home Scraper*, ma ovviamente con attributi differenti, quali: `username`, `img_profile`, `user_link`, `bio` e `followers`. Dopo di ciò si restituisce il dizionario costruito dal `getter`.

Questo dizionario va in input alla funzione `save_user` che provvede a salvare i dati nella tabella `post_users` del database tramite il metodo `execute`.

Infine, sulla pagina del profilo utente, si richiama la funzione `scroll_to_bottom`, per recuperare tutti i post e salvarli, tramite la funzione `save_to_db`, nella tabella `posts`.

Il programma ricomincia il ciclo prelevando un secondo utente dalla tabella `posts` e così via.

## 5.3 Post Scraper

Questo scraper si occupa di recuperare informazioni sui singoli post, quali descrizione del post, immagini, video e commenti testuali. Il codice è simile al *Profile Scraper*; per questo motivo ci si soffermerà unicamente sulle funzioni che implementano delle differenze.

La funzione `main` è illustrata nel Listato 5.22. Lo scraper effettua i seguenti passi:

- crea la connessione al database;
- verifica se la tabella `posts` esiste e se contiene un post con id uguale ad uno;
- crea la tabella `posts_info`;
- inizializza il driver;
- entra un loop che prevede le seguenti azioni:
  - tramite la funzione `get_post_link` recupera il link di un post;
  - il driver apre graficamente il post;
  - accetta i cookie solo per il primo post;
  - recupera le informazioni dal post tramite la funzione `fetch_post_content`;
  - salva la lista di informazioni nel database per mezzo della funzione `save`;
  - incrementa il numero del post da ricercare di uno;

---

```

def main():
    """ main function """

    # first post to scrape
    post_id = 1

    # post to push into db
    posts_to_save = []
    # number of post to save
    post_no_to_save = 10

    try:
        # connection to db
        conn = pymysql.connect(
            host='localhost',
            user='mysql',
            password='mysql',
            database='mysql',
            port=3306
        )

        # Ensure the table "posts" exists and contains the post_id == 1
        if not check_table_and_post(post_id, 60):
            print("Required table or post not found. Exiting.")
            return

        # posts_info table creation
        manual_migration(conn)

        # start driver
        driver = start_driver()

        while True:
            # Attempt to get the post link
            post_link = get_post_link(post_id, conn)

            if post_link:
                print("\nPost:", post_link)
                # open post link
                driver.get(post_link)

                # first post
                if post_id == 1:
                    # accept cookies only once
                    accept_cookies(driver)

                # fetch post info and add to the list of post to save in the db
                new_post_info = fetch_post_content(driver, post_id)
                if new_post_info is not None:
                    posts_to_save.append(new_post_info)
                    print("post added to the list:", new_post_info)

                # save post to db
                if len(posts_to_save) >= post_no_to_save:
                    save(posts_to_save, conn)
                    posts_to_save = []

                time.sleep(5)

            else:
                print(f"\nPost id {post_id} not found in db!")

            # Increment post_id to fetch the next post
            post_id += 1

    except Exception as e:
        print('Main Error:', e)

    finally:
        close_session(driver)
        if conn:
            conn.close()

```

---

**Listing 5.22:** Funzione principale del *Post Scraper*

Nel Listato 5.23 è rappresentata la funzione necessaria per il recupero delle informazioni contenute nei post.

---

```
def fetch_post_content(driver, post_id):
    """ fetch info within each post: caption, images, video and comments """
    POST_TIMEOUT = 10

    try:
        # wait and fetch the post
        post = WebDriverWait(driver, POST_TIMEOUT).until(
            EC.presence_of_element_located((By.XPATH, POST_CONTAINER_XPATH))
        )

        # fetch the post caption
        caption = get_caption(post)

        # fetch images
        link_images = get_images(post)

        # fetch videos
        link_videos = get_videos(post)

        # fetch comments
        comments = get_comments(post)

        # create object
        post_obj = PostInfo(post_id, caption, link_images, link_videos, comments)
        # return post info
        return post_obj.info

    except TimeoutException:
        print("Unable to fetch the post..next post")
        return None
```

---

**Listing 5.23:** Funzione per recuperare le informazioni dai singoli post

Anche in questo caso si è creata una funzione per recuperare ogni singolo aspetto del post e si è fatto uso di una classe nella costruzione dell'oggetto che verrà, poi, salvato nel database.

Per recuperare i commenti si è utilizzato il classico XPath, mentre per le immagini ed i video, da un'analisi del DOM, si è preferito costruire un selettore più robusto basato sul tipo di implementazione adottata da Threads. Infatti, si è notato come il link dei video è sempre recuperabile all'interno di un tag video; mentre, nel caso delle immagini, è emerso che ogni immagine è contenuta dentro un tag img, a sua volta contenuto in un tag picture. Nel Listato 5.24 vi è l'esempio della funzione get\_images.

---

```
def get_images(post):
    """ return post images """

    link_images = []
    pictures = post.find_elements(By.TAG_NAME, "picture")

    if pictures:
        for picture in pictures:
            image_tag = picture.find_element(By.TAG_NAME, "img")
            link = image_tag.get_attribute('src')

            link_images.append(link)
    return link_images
return None
```

---

**Listing 5.24:** Funzione per recuperare le immagini presenti nella descrizione del post

Le immagini, così come i video ed i commenti, sono salvati in una variabile avente una struttura di lista di stringhe. Questo impone di utilizzare il metodo `json.dumps` nel salvataggio di tali informazioni nel database (Listato 5.25).

Le immagini ed i video vengono successivamente scaricati nella memoria di massa da uno script appositamente creato. Quest'ultimo recupera i link dalla tabella `posts_info` ed esegue il download. Per brevità si omette il codice in questione.

---

```
def save(posts_to_save, conn):
    """ save post info to db """
    try:
        with conn.cursor() as cursor:
            # Create an SQL insert statement
            sql = """
                INSERT INTO posts_info (post_id,caption,link_images,link_videos,comments)
                VALUES (%s, %s, %s, %s, %s)
            """

            # Prepare the data as a list of tuples
            values = [(post['post_id'], post['caption'], json.dumps(post['images']),
                       json.dumps(post['videos']), json.dumps(post['comments'])) for post in
                      posts_to_save]

        try:
            # Execute the insert statement for all data at once
            rows_inserted = cursor.executemany(sql, values)

            # Commit the transaction
            conn.commit()
            print(f'\n{rows_inserted} rows saved in the database correctly')

        except MySQLError as e:
            print('Mysql insert query Error:', e)

    except MySQLError as e:
        print('Mysql Error:', e)
```

---

**Listing 5.25:** Funzione per recuperare le immagini presenti nella descrizione del post

# CAPITOLO 6

---

## Messa in esercizio dello scraper

---

*Questo capitolo illustra il processo di messa in esercizio dello scraper progettato per l'estrazione automatica di dati dalla piattaforma Threads. L'obiettivo è fornire una descrizione dettagliata delle attività necessarie per avviare i servizi e garantire il corretto funzionamento del sistema, oltre a definire i risultati attesi dall'operatività dello scraper. Nella Sezione 6.1 si descrive l'avvio coordinato dei servizi principali, inclusi il database e i tre scraper. In particolare, viene presentata l'esecuzione pratica delle operazioni, con particolare attenzione alla gestione dei log. La Sezione 6.2 si concentra, invece, sui risultati attesi dall'esecuzione dello scraper, evidenziando le metriche principali e gli obiettivi di raccolta dei dati. In particolare, viene discusso come il sistema dovrebbe comportarsi in scenari operativi, con particolare enfasi sulla qualità e sull'integrità dei dati raccolti.*

### 6.1 Avvio dei servizi

L'avvio dei servizi per lo scraping di Threads è organizzato in tre principali passaggi, ovvero: l'esecuzione di Docker Compose, l'inizializzazione dei tre scraper e la gestione dei log. Questo approccio consente un'operatività coordinata ed efficiente, riducendo al minimo i problemi durante l'esecuzione.

#### 6.1.1 Avvio con Docker Compose

Il primo passo consiste nell'avviare il servizio di database utilizzando Docker Compose. Questo strumento facilita l'esecuzione dei container configurati nel `docker-compose.yml`, garantendo un ambiente stabile e replicabile. Il comando per avviare i servizi è riportato nel Listato 6.1:

---

```
docker compose up -d
```

---

**Listing 6.1:** Avvio dei servizi tramite Docker Compose

Dopo l'esecuzione del comando, il database MySQL è accessibile e pronto a ricevere le richieste provenienti dagli scraper.

È possibile consultare il database all'URL `http://localhost:8080/` della macchina host, per mezzo del servizio Adminer, configurato nel file di configurazione del Docker Compose.

### 6.1.2 Esecuzione dei tre scraper

Ogni scraper viene avviato in modo indipendente tramite i rispettivi script Python. Gli scraper lavorano in parallelo, ognuno focalizzato su un aspetto specifico; in particolare:

- lo *Home Scraper* raccoglie i post dalla home page di Threads;
- il *Profile Scraper* estrae le informazioni degli utenti ed i post associati ai profili;
- il *Post Scraper* analizza i singoli post per estrarre contenuti, immagini, video e commenti.

I tre scraper possono essere avviati manualmente, a seguito di una conversione in script, utilizzando il comando riportato nel Listato 6.2; in alternativa, essi possono essere lanciati direttamente all'interno dell'ambiente Jupyter Notebook.

---

```
python home_scraper.py &
python profile_scraper.py &
python post_scraper.py &
```

---

**Listing 6.2:** Avvio manuale dei tre scraper in parallelo

Nel seguito si considera quest'ultima casistica. Dunque, una volta lanciati contemporaneamente i tre scraper, vengono aperte, di conseguenza, tre finestre di Chrome che eseguiranno le operazioni di crawling.

La finestra relativa allo *Home Scraper* scorrerà visivamente tutta la home page di Threads. Una volta che si è giunti al termine della pagina, si riavvierà autonomamente per riprendere tale operazione di *scrolling* con raccolta dati.

Il *Profile Scraper* prevede una finestra di Google Chrome che si apre su un profilo utente e lo scorre fino alla fine. Successivamente, ripete l'operazione per un altro utente, raccogliendo post e informazioni su di lui.

La finestra associata al *Post Scraper* si apre ripetutamente su un post differente, con lo scopo di recuperare le informazioni contenute.

### 6.1.3 Gestione dei log

Durante l'esecuzione, ciascuno scraper genera log dettagliati che includono informazioni sulle operazioni eseguite, eventuali errori e post non trovati nel database. I log sono utili per il debugging e il monitoraggio delle attività. Ad esempio vi sono:

- *log per righe duplicate*, che notificano i tentativi di salvare dati già presenti nel database, grazie alla configurazione `INSERT IGNORE`;
- *log per post non trovati*, che indicano quando un post richiesto non è presente nel database;
- *log per operazioni di salvataggio correttamente effettuate*.

Grazie alla gestione dei log, è possibile identificare rapidamente eventuali problemi e ottimizzare il flusso di scraping. Inoltre, l'approccio modulare adottato per l'avvio dei servizi facilita l'integrazione di nuovi scraper o la modifica delle configurazioni esistenti.

In particolare, si vuole mostrare un'analisi dei log passo passo dall'avvio dei servizi. Lo *Home Scraper* inizia per primo le operazioni di crawling, come testimoniano i suoi log (Listato 6.3).

---

```
CSS cookies button click
```

```
Website: Threads
Current URL: https://www.threads.net/?hl=en
"posts" table created
post added number: 1
post added to list: {'username': 'kristophershinn', 'username_link':
    ↳ 'https://www.threads.net/@kristophershinn', 'post_link':
    ↳ 'https://www.threads.net/@kristophershinn/post/DDlIWFUSIA'}
scroll down
post added number: 2
post added to list: {'username': '80s.deennice', 'username_link':
    ↳ 'https://www.threads.net/@80s.deennice', 'post_link':
    ↳ 'https://www.threads.net/@80s.deennice/post/DDlQm6rMSTV'}
scroll down
post added number: 3
post added to list: {'username': 'rgiii', 'username_link':
    ↳ 'https://www.threads.net/@rgiii', 'post_link':
    ↳ 'https://www.threads.net/@rgiii/post/DDkUtilPTbr'}
scroll down
post added number: 4
post added to list: {'username': 'astronomiaum', 'username_link':
    ↳ 'https://www.threads.net/@astronomiaum', 'post_link':
    ↳ 'https://www.threads.net/@astronomiaum/post/DDlEEanpTUF'}
scroll down
post added number: 5
post added to list: {'username': 'hunterwalk', 'username_link':
    ↳ 'https://www.threads.net/@hunterwalk', 'post_link':
    ↳ 'https://www.threads.net/@hunterwalk/post/DDlGb0Ky4qM'}
scroll down
post added number: 6
post added to list: {'username': '101espn', 'username_link':
    ↳ 'https://www.threads.net/@101espn', 'post_link':
    ↳ 'https://www.threads.net/@101espn/post/DDkMrR_vSaR'}
scroll down
post added number: 7
post added to list: {'username': 'abc7newsbayarea', 'username_link':
    ↳ 'https://www.threads.net/@abc7newsbayarea', 'post_link':
    ↳ 'https://www.threads.net/@abc7newsbayarea/post/DDkDwtcxIl'}
scroll down
post added number: 8
post added to list: {'username': 'vciiuu', 'username_link':
    ↳ 'https://www.threads.net/@vciiuu', 'post_link':
    ↳ 'https://www.threads.net/@vciiuu/post/DDmDS4ruNP4'}
scroll down
post added number: 9
post added to list: {'username': 'darksideofthemat', 'username_link':
    ↳ 'https://www.threads.net/@darksideofthemat', 'post_link':
    ↳ 'https://www.threads.net/@darksideofthemat/post/DDlU-ejJ0dr'}
scroll down
post added number: 10
post added to list: {'username': 'dan_regan_comedy', 'username_link':
    ↳ 'https://www.threads.net/@dan_regan_comedy', 'post_link':
    ↳ 'https://www.threads.net/@dan_regan_comedy/post/DDkqW_ez2Eb'}
scroll down
```

---

**Listing 6.3:** Log iniziali dello *Home Scraper*

Dai log del Listato 6.3 è possibile osservare:

- il selettori utilizzati per accettare i cookie;
- Il titolo della pagina web che si sta visualizzando;
- l'URL della pagina;
- la conferma della creazione della tabella posts;
- i dettagli associati ad ogni singolo post;
- l'operazione di scorrimento della pagina.

Il *Profile Scraper* ed il *Post Scraper*, come ormai risulta chiaro, dipendono dallo *Home Scraper*. Infatti, all'avvio attendono che la tabella posts sia riempita con i dati per avviare le operazioni. Nel Listato 6.4 sono riportati i log a testimonianza di quanto appena detto.

---

```
Post with id=1 not found in table 'posts'. Retrying in 60 seconds...
Post with id=1 not found in table 'posts'. Retrying in 60 seconds...
Post with id=1 not found in table 'posts'. Retrying in 60 seconds...
Table 'posts' exists and contains post with id=1.
```

---

**Listing 6.4:** Log del *Profile Scraper* in attesa dei dati dalla tabella posts

Tornando allo *Home Scraper*, una volta riempita la lista di dati da caricare nella tabella posts, si possono caricare le informazioni raccolte nel database, come mostrato dai log del Listato 6.5.

---

```
post added number: 48
post added to list: {'username': 'that80sand90spage', 'username_link':
    ↳ 'https://www.threads.net/@that80sand90spage', 'post_link':
    ↳ 'https://www.threads.net/@that80sand90spage/post/DDk6uQUvGt3'}
scroll down
post added number: 49
post added to list: {'username': 'la12tuittera', 'username_link':
    ↳ 'https://www.threads.net/@la12tuittera', 'post_link':
    ↳ 'https://www.threads.net/@la12tuittera/post/DDk3RKLvkHw'}
scroll down
post added number: 50
post added to list: {'username': 'jpz_media', 'username_link':
    ↳ 'https://www.threads.net/@jpz_media', 'post_link':
    ↳ 'https://www.threads.net/@jpz_media/post/DDmf8yIRcZF'}
50 rows saved in the database correctly
scroll down
post added number: 51
post added to list: {'username': 'sullyoon__nmixx', 'username_link':
    ↳ 'https://www.threads.net/@sullyoon__nmixx', 'post_link':
    ↳ 'https://www.threads.net/@sullyoon__nmixx/post/DDk3zmEzoIL'}
scroll down
```

---

**Listing 6.5:** Salvataggio dei dati raccolti dallo *Home Scraper* nella tabella posts

Ogni cinquanta elementi all'interno della lista si salvano i dati nel database. Prima di passare ad analizzare i log degli altri due scraper, si vogliono mostrare due casistiche interessanti appartenenti allo *Home Scraper*, ovvero la gestione dei duplicati e lo scorrimento della pagina.

Nel primo caso, quando si ha nella lista da caricare un link che è già presente nel database, questo, ad eccezione di tutti gli altri elementi della lista, viene automaticamente scaricato. Un esempio di questo particolare caso si può osservare nel Listato 6.6, in cui sono stati caricati nel database 49 su 50 elementi, a causa proprio di un duplicato.

---

```
post added number: 198
post added to list: {'username': 'axCreatesOfficial', 'username_link':
    ↳ 'https://www.threads.net/@axCreatesOfficial', 'post_link':
    ↳ 'https://www.threads.net/@axCreatesOfficial/post/DDkOTXpPyrX'}
scroll down
post added number: 199
post added to list: {'username': 'dicascriativaspvc', 'username_link':
    ↳ 'https://www.threads.net/@dicascriativaspvc', 'post_link':
    ↳ 'https://www.threads.net/@dicascriativaspvc/post/DDkz9fTOKVn'}
scroll down
post added number: 200
post added to list: {'username': 'foodObservations', 'username_link':
    ↳ 'https://www.threads.net/@foodObservations', 'post_link':
    ↳ 'https://www.threads.net/@foodObservations/post/DDktAqFTkbK'}
49 rows saved in the database correctly
scroll down
```

---

**Listing 6.6:** Gestione dei post duplicati da parte dello *Home Scraper* nella tabella posts

Un'ultima considerazione, prima di passare ai restanti due scraper, riguarda lo scorrimento e il recupero dei post. Come anticipato nel capitolo precedente, quando un post non viene correttamente recuperato è perché non è ancora stato visualizzato nella pagina. Quindi, per recuperarlo, lo scraper deve continuare a scorrere la pagina finché tale post non viene renderizzato. Un esempio di questa casistica è nel Listato 6.7.

---

```
post added number: 84
post added to list: {'username': 'rae.in_wonderland', 'username_link':
    ↳ 'https://www.threads.net/@rae.in_wonderland', 'post_link':
    ↳ 'https://www.threads.net/@rae.in_wonderland/post/DDlmtD9Alvi'}
scroll down
Post number 85 not found
scroll down
Post number 85 not found
scroll down
Post number 85 not found
scroll down
post added number: 85
post added to list: {'username': 'carnage4life', 'username_link':
    ↳ 'https://www.threads.net/@carnage4life', 'post_link':
    ↳ 'https://www.threads.net/@carnage4life/post/DDkOyJMv9lj'}
scroll down
```

---

**Listing 6.7:** Recupero di un post ancora non renderizzato da parte dello *Home Scraper*

A questo punto, a seguito del primo caricamento dei valori da parte dello *Home Scraper* nella tabella posts, i restanti due scraper entrano in azione.

Si analizzano per primi i log del *Profile Scraper*. Lo scraper apre, quindi, la pagina del profilo associata all’utente e recupera informazioni sul profilo utente e sui post pubblicati (Listato 6.8).

```
User Profile: https://www.threads.net/@kristophershinn
CSS cookies button click
User 'kristophershinn' saved successfully.
post number 1 added: {'username': 'kristophershinn', 'username_link':
    ↵ 'https://www.threads.net/@kristophershinn', 'post_link':
    ↵ 'https://www.threads.net/@kristophershinn/post/DDIWFUSIAN'}
scroll down
post number 2 added: {'username': 'kristophershinn', 'username_link':
    ↵ 'https://www.threads.net/@kristophershinn', 'post_link':
    ↵ 'https://www.threads.net/@kristophershinn/post/DDkI7dIv4OU'}
scroll down
post number 3 added: {'username': 'kristophershinn', 'username_link':
    ↵ 'https://www.threads.net/@kristophershinn', 'post_link':
    ↵ 'https://www.threads.net/@kristophershinn/post/DDiFotrzE7Q'}
scroll down
post number 4 added: {'username': 'kristophershinn', 'username_link':
    ↵ 'https://www.threads.net/@kristophershinn', 'post_link':
    ↵ 'https://www.threads.net/@kristophershinn/post/DDfPRPFvb0_'}
scroll down
post number 5 added: {'username': 'kristophershinn', 'username_link':
    ↵ 'https://www.threads.net/@kristophershinn', 'post_link':
    ↵ 'https://www.threads.net/@kristophershinn/post/DDadmupyrrz'}
scroll down
post number 6 added: {'username': 'kristophershinn', 'username_link':
    ↵ 'https://www.threads.net/@kristophershinn', 'post_link':
    ↵ 'https://www.threads.net/@kristophershinn/post/DDaJBKsyWiU'}
scroll down
post number 7 added: {'username': 'kristophershinn', 'username_link':
    ↵ 'https://www.threads.net/@kristophershinn', 'post_link':
    ↵ 'https://www.threads.net/@kristophershinn/post/DDYSB_JyF4_'}
scroll down
post number 8 added: {'username': 'kristophershinn', 'username_link':
    ↵ 'https://www.threads.net/@kristophershinn', 'post_link':
    ↵ 'https://www.threads.net/@kristophershinn/post/DDXgXhczy8o'}
scroll down
post number 9 added: {'username': 'kristophershinn', 'username_link':
    ↵ 'https://www.threads.net/@kristophershinn', 'post_link':
    ↵ 'https://www.threads.net/@kristophershinn/post/DDXOzdiRN28'}
scroll down
post number 10 added: {'username': 'kristophershinn', 'username_link':
    ↵ 'https://www.threads.net/@kristophershinn', 'post_link':
    ↵ 'https://www.threads.net/@kristophershinn/post/DDSHou5vmxr'}
9 rows saved in "posts" database correctly
scroll down
post number 11 added: {'username': 'kristophershinn', 'username_link':
    ↵ 'https://www.threads.net/@kristophershinn', 'post_link':
    ↵ 'https://www.threads.net/@kristophershinn/post/DDP5tKdy6aR'}
```

**Listing 6.8:** Salvataggio del profilo utente e dei primi dieci post da parte del *Profile Scraper*

Dai log è possibile ricavare il comportamento dello scraper; in particolare:

- si effettua il loggin dell'URL associato al profilo utente in esame;
- si accettano i cookie la prima volta;
- si salvano le informazioni relative al profilo nella tabella `post_users`;
- si scorre il profilo e, ogni dieci post, si effettua un salvataggio di questi ultimi nella tabella `posts`.

Anche in questo caso i post duplicati, ovvero quelli già presenti nella tabella `posts`, vengono automaticamente scartati, come avveniva per lo *Home Scraper*.

Purtroppo, per gli utenti non autenticati si ha un massimo di venticinque post visualizzabili su ogni profilo utente, come testimoniato dal Listato 6.9. Salvati tutti i dati, si passa all'utente successivo per effettuare le medesime operazioni.

---

```

post number 25 added: {'username': 'kristophershinn', 'username_link':
    ↳ 'https://www.threads.net/@kristophershinn', 'post_link':
    ↳ 'https://www.threads.net/@kristophershinn/post/DCHyTv3TcpV'}
scroll down
Post number 26 not found
scroll down
End of Page. Taking a new user...
Saving remaining 5 posts to the database...
5 rows saved in "posts" database correctly

User Profile: https://www.threads.net/@80s.deennice
User '80s.deennice' saved successfully.
post number 1 added: {'username': '80s.deennice', 'username_link':
    ↳ 'https://www.threads.net/@80s.deennice', 'post_link':
    ↳ 'https://www.threads.net/@80s.deennice/post/DDlQm6rMSTV'}
scroll down

```

---

**Listing 6.9:** Salvataggio dei post associati ad un utente da parte del *Profile Scraper*

In ultima analisi, si mostrano i log del *Post Scraper*. In questo caso si preferisce utilizzare la Figura 6.1 per mostrare i risultati dei log a causa delle emoticon. I dati salvati vanno ovviamente "ripuliti" con alcuni strumenti appropriati, al fine di essere utilizzati propriamente per svolgere analisi.

Lo scraper, quindi, mostra l'URL associato al post, accetta i cookie solo per il primo post e salva le informazioni in una lista di dizionari. Ogni dieci elementi nella lista, i dati vengono salvati nella tabella `posts_info` (Figura 6.2).

Post: <https://www.threads.net/@kristophershinn/post/DD1WIWFUSIA>  
CSS cookies button click  
post added to the list: {'post\_id': 1, 'caption': 'Bainbridge Island 🌏\nSeattle Threads', 'images': ['[https://www.threads.net/@80s.deennice/post/DD1Qm6rMSTV](https://instagram.fmxp12-1.fna.fbcdn.net/v/t51.29350-15/47000622_926640356088594_1926570998338909776_n.jpg?stp=dst-jpg_e35_tt6&efg=eyJ2ZW5jb2R1X3RhzYi5ImlyWld1X3ywgbd1x4NDQwDE30TyCz2RyLmYyOTM1MC5KZWldNx0x21tyWd1In0&nc_ht=instagram.fmxp12-1.fna.fbcdn.net&nc_cat=106&nc_ohc=j9kT_r_kWWEQ7kNvgE8hCdo&nc_gid=63a6e1871bd4e9696e0c141145609737&edm=APs17CUBAAA&ccb=7-5&ig_cache_key=MzUyMzI1ODk4NTM5Ngj2MzAxNQ93D%3D.3-ccb7-5&oh=00_AYBgmCYTyCaViqb0PlcJh4loZqgWdD5fsJeuw5S_Abw&oe=6764F691&nc_sid=10d13b'], 'videos': None, 'comments': ['Beautiful! ❤️']}</p><p>Post: <a href=)  
post added to the list: {'post\_id': 2, 'caption': "'Santa Claus Is Comin' to Town' first aired on ABC 54 years ago today on December 14, 1970. This holiday classic has continued to air annually every holiday season since.", 'images': None, 'videos': ['<a href="https://instagram.fmxp12-1.fna.fbcdn.net/o1/v/t16/f2/m84/AQNmVQUFWS86xZbg\_FvTn9Pfjzr8tNzCNixRt\_7b9IAR26nCDOSW9dM9McZaJ4t9nTIdTh\_TG2Zn3R2TnK9n-K6p9xZB\_4zsf20.mp4?stp=dst-mp4&amp;efg=eyJxV9nmcn91cHMi0ijbXcJpZ1932WfZGvsaXZlcnlfndRz2X90ZlwixSiZ1zlbn1vZGVfdGFnIjoiJdnRzX3VZFS91cmxnZw4UzmV1ZC5jMj43MjaUyMfzZwxbpUmUiQ&amp;nc\_cat=111&amp;vs=3644585962498635\_4237476591&amp;nc\_vs=BHksFQIYTGlNx23H2ytmaWxsX3RpblwVsaw51X3zV2C8yNj0xRjq0D0IDXNEVD0Mjg20M0GVTQ1MuYQ01Mn2MR92Awr1b19kYXNa05wdp5CtcdQVAALIAQAVAhgcGfZc3Rcm91Z2hfXZLcnB03j1l0dMX2NCUnhnyJNUWFiyY0hBRU4yak5V0hhdylia11MQUBRKhuCasGAGABAABUACbeobB2Fws40NQUBQbMwXQGNM2MzMzMWBYErhc2HfyFZwXpbmfvMfW92MREadeoHAA%3D%3D&amp;nc\_rid=685b6d4160&amp;ccb=9-4&amp;h=00\_AYDjdms5NrrUhXk1YeYNP-51Dmgbu\_m60lctYj-WDObZqg&amp;e=6760DB82&amp;nc\_sid=10d13b'], 'comments': ['Watch til the end for a bumper and Cabbage Patch Kids commercial from this 1985 recording 😂', 'I watched this last night 😊', '👉👉👉👉', '😊😊😊😊😊😊😊😊😊😊😊😊😊', 'Yes yes yes', '❤️❤️❤️', 'I'm old enough to remember when it first aired! Still love it!', '❤️❤️❤️', 'Love it ❤️', 'Just watched it tonight.', '❤️❤️❤️', 'Notice the VCR flashing 12:00? 🎵🎵', 'Oh Wow 54 Years Ago Amazing!! ❤️❤️❤️!! Yay Happy Birthday Santa Claus Is Coming To Town You Never Get Old I Watched It With My Family Last Night Best Holiday Show Ever ❤️❤️🎅🎅❤️', '❤️ And Thank You For Sharing Yay!! ❤️❤️!! And I Hope That You Are Having A Happy And Safe Early Sunday', 'My first Christmas!!', 'stand claus', 'I've already watched it 3 times. 😊❤️🎄', '😊😊😊', 'The full film is on youtube\nyoutu.be/EMUQ0...']}</p>

Post: <https://www.threads.net/@rgiiii/post/DDKUtilPTbr>  
post added to the list: {'post\_id': 3, 'caption': 'How you treat people is more important than how many likes you get on social media.', 'images': None, 'videos': None, 'comments': ['Well said RG3!', 'Yea but pickleball is life', 'Say it louder for the people in the back!!!!', 'Agreed.', 'Facts', 'Remember when you platformed a dude that slapped his wife in public?', 'Facts.', 'Agreed', 'Not really!', 'I agree. Get respect give respect and vice versa', 'Agreed!!', "This is why I've never been about the clout. Unfortunately, I know some businesses that only care about the clout, and they're also some of the most toxic work environments as a result. Be kind to one another out there.", 'Are you sure about that\n@rgiiii\n\n😊 🎉 😊', 'Where did you get this gem from? A fortune cookie?']}]

**Figura 6.1:** Log del *Post Scraper*

Post: <https://www.threads.net/@darksideofthemat/post/DD1U-ejJ0dr>  
post added to the list: {'post\_id': 9, 'caption': 'My dad just took a piece of bread, buttered it.. then salted it 😊', 'image\_s': None, 'Videos': None, 'comments': ['Unsalted butter??', 'Salted butter 🥰', 'totally normal.', 'If it was truly normal this post would have no engagement ijs', 'It was delicious.', 'Legend', 'Sounds good actually', "I'm happy for him 😊", '】,【Yum!', 'Yes? You have an issue with seasoning?', 'Yum!', "I mean...buttered bread is a thing, and it's best with salted butter, but even better with salt right in the surface.", 'Salt on the surface > salt in the stuff', 'To be fair I salted the butter on the bread.', 'This is so wrong 😂', 'My dad did this with saltine crackers', "Benny has been putting butter, salt and pepper on toast with eggs recently and it's a restaurant level trick. So delicious", 'Okay but on eggs and toast is different', 'But also the toast that we have with soup! You should try it 😊', 'This is how I eat bread - what's the problem?']}]

Post: [https://www.threads.net/@dan\\_regan\\_comedy/post/DDKqW\\_ez2Eb](https://www.threads.net/@dan_regan_comedy/post/DDKqW_ez2Eb)  
post added to the list: {'post\_id': 10, 'caption': 'The people who would "Bust A Move" and "Jump Around" in the 90s now have house Ibuprofen and car Ibuprofen. \nTraduci', 'images': None, 'videos': None, 'comments': ['And work ibuprofen \nTraduci', 'I don\'t have a car, so I have house (in two places) and backpack instead. \nTraduci', 'I identify \nTraduci', 'Not me! 😱 😳 😱 \nTraduci', 'Work ibuprofen, purse ibuprofen, bathroom ibuprofen \nTraduci', 'Goody or BC in the bag and in the car. Lol \nTraduci', 'Word', 'Yes...and I still jump around pretty hard! \nTraduci', 'Damn! It\'s true! \nTraduci', 'Home ibuprofen and office ibuprofen. \nTraduci', 'And work ibuprofen as well. \nTraduci', 'And work ibuprofen \nTraduci', 'ok that was laff outloud time \nTraduci', 'Same as house reading glasses and car reading glasses \nTraduci', 'Just in the house, but I haven\'t had to take any in a while. \nTraduci', 'And lunchbag ibuprofen and work ibuprofen.. \nTraduci', 'Workbag ibuprofen and frankly a few other drugs just in case. Allergy meds, headache meds, cough drops. \nTraduci', 'I got desk ibuprofen... \nTraduci', 'I feel attacked \nTraduci', '"When I move you move...AAGH MY BACK!" \nTraduci', 'Home, car and purse \nTraduci']}}

10 rows saved in the database correctly

**Figura 6.2:** Salvataggio dei dati nella tabella posts\_info da parte dello *Post Scraper*

## 6.2 Risultati attesi

In questa sezione vengono descritti i risultati attesi dall'esecuzione dello scraper, con particolare attenzione ai dati raccolti e alla loro organizzazione all'interno del database.

L'obiettivo principale è garantire che i tre scraper funzionino correttamente, popolando le tabelle `posts`, `post_users` e `posts_info` con informazioni coerenti e prive di duplicati. Ogni tabella rappresenta un aspetto specifico dei dati raccolti; in particolare: la tabella `posts` memorizza i dettagli dei post trovati nella home page, la tabella `post_users` raccoglie i dettagli relativi agli utenti, mentre la tabella `posts_info` contiene informazioni più dettagliate sui contenuti dei singoli post.

I dati memorizzati sono pronti per essere analizzati o utilizzati in fasi successive, con la possibilità di estrarre informazioni utili per applicazioni di Data Analysis o Business Intelligence. Infine, vengono presentati esempi concreti di dati organizzati all'interno delle tabelle, mostrando come lo scraper consenta di trasformare le informazioni non strutturate del web in un formato strutturato e facilmente interrogabile.

I risultati ottenuti possono essere visualizzati direttamente dall'interfaccia di Adminer e sono riportati nelle Figure: 6.3 per la tabella `posts`; 6.4 e 6.5 per la tabella `post_users`; 6.6 e 6.7 per la tabella `post_info`.

I dati sono il risultato di un'attività di crawling di soli quindici minuti. È facile osservare come lo *Home Scraper* sia il più veloce tra tutti, in quanto è lo script principale che prepara i dati nella tabella `posts` da cui, poi, i restanti due scraper recuperano informazioni sui singoli utenti e post.

Ovviamente si può "giocare" con i ritardi impostati nel codice per rendere più o meno veloce l'attività di crawling, sempre tenendo presente che un'eccessiva velocità è un chiaro campanello d'allarme per eventuali meccanismi anti-bot.

SELECT * FROM `posts` LIMIT 50 (0.000 s) Modifica						
	Modify	<b>id</b>	<b>username</b>	<b>username_link</b>	<b>post_link</b>	<b>created</b>
<input type="checkbox"/>	modifica	1	kristophershinn	<a href="https://www.threads.net/@kristophershinn">https://www.threads.net/@kristophershinn</a>	<a href="https://www.threads.net/@kristophershinn/post/DDIIWFUSIA">https://www.threads.net/@kristophershinn/post/DDIIWFUSIA</a>	2024-12-15 17:14:14
<input type="checkbox"/>	modifica	2	80s.deennice	<a href="https://www.threads.net/@80s.deennice">https://www.threads.net/@80s.deennice</a>	<a href="https://www.threads.net/@80s.deennice/post/DDIQm6rMSTV">https://www.threads.net/@80s.deennice/post/DDIQm6rMSTV</a>	2024-12-15 17:14:14
<input type="checkbox"/>	modifica	3	rgiii	<a href="https://www.threads.net/@rgiii">https://www.threads.net/@rgiii</a>	<a href="https://www.threads.net/@rgiii/post/DDKUtl1Ptbr">https://www.threads.net/@rgiii/post/DDKUtl1Ptbr</a>	2024-12-15 17:14:14
<input type="checkbox"/>	modifica	4	astronomiaum	<a href="https://www.threads.net/@astronomiaum">https://www.threads.net/@astronomiaum</a>	<a href="https://www.threads.net/@astronomiaum/post/DDIEEanpTUF">https://www.threads.net/@astronomiaum/post/DDIEEanpTUF</a>	2024-12-15 17:14:14
<input type="checkbox"/>	modifica	5	hunterwalk	<a href="https://www.threads.net/@hunterwalk">https://www.threads.net/@hunterwalk</a>	<a href="https://www.threads.net/@hunterwalk/post/DDIGb0Ky4qM">https://www.threads.net/@hunterwalk/post/DDIGb0Ky4qM</a>	2024-12-15 17:14:14
<input type="checkbox"/>	modifica	6	101espn	<a href="https://www.threads.net/@101espn">https://www.threads.net/@101espn</a>	<a href="https://www.threads.net/@101espn/post/DDKMrr_vSaR">https://www.threads.net/@101espn/post/DDKMrr_vSaR</a>	2024-12-15 17:14:14
<input type="checkbox"/>	modifica	7	abc7newbayarea	<a href="https://www.threads.net/@abc7newbayarea">https://www.threads.net/@abc7newbayarea</a>	<a href="https://www.threads.net/@abc7newbayarea/post/DDKdWtCx1I">https://www.threads.net/@abc7newbayarea/post/DDKdWtCx1I</a>	2024-12-15 17:14:14
<input type="checkbox"/>	modifica	8	vciuu	<a href="https://www.threads.net/@vciuu">https://www.threads.net/@vciuu</a>	<a href="https://www.threads.net/@vciuu/post/DDmDS4ruNp4">https://www.threads.net/@vciuu/post/DDmDS4ruNp4</a>	2024-12-15 17:14:14
<input type="checkbox"/>	modifica	9	darksideofthemat	<a href="https://www.threads.net/@darksideofthemat">https://www.threads.net/@darksideofthemat</a>	<a href="https://www.threads.net/@darksideofthemat/post/DDIU-ej0dr">https://www.threads.net/@darksideofthemat/post/DDIU-ej0dr</a>	2024-12-15 17:14:14
<input type="checkbox"/>	modifica	10	dan_regan_comedy	<a href="https://www.threads.net/@dan_regan_comedy">https://www.threads.net/@dan_regan_comedy</a>	<a href="https://www.threads.net/@dan_regan_comedy/post/DDkqW_ez2Eb">https://www.threads.net/@dan_regan_comedy/post/DDkqW_ez2Eb</a>	2024-12-15 17:14:14
<input type="checkbox"/>	modifica	11	carnage4life	<a href="https://www.threads.net/@carnage4life">https://www.threads.net/@carnage4life</a>	<a href="https://www.threads.net/@carnage4life/post/DDkQsovhUT">https://www.threads.net/@carnage4life/post/DDkQsovhUT</a>	2024-12-15 17:14:14
<input type="checkbox"/>	modifica	12	akmulford	<a href="https://www.threads.net/@akmulford">https://www.threads.net/@akmulford</a>	<a href="https://www.threads.net/@akmulford/post/DDk01UeT0_Z">https://www.threads.net/@akmulford/post/DDk01UeT0_Z</a>	2024-12-15 17:14:14
<input type="checkbox"/>	modifica	13	michaelwarburton1	<a href="https://www.threads.net/@michaelwarburton1">https://www.threads.net/@michaelwarburton1</a>	<a href="https://www.threads.net/@michaelwarburton1/post/DDlfIWOT1q">https://www.threads.net/@michaelwarburton1/post/DDlfIWOT1q</a>	2024-12-15 17:14:14
<input type="checkbox"/>	modifica	14	shamsnba	<a href="https://www.threads.net/@shamsnba">https://www.threads.net/@shamsnba</a>	<a href="https://www.threads.net/@shamsnba/post/DDkuxvpSa5G">https://www.threads.net/@shamsnba/post/DDkuxvpSa5G</a>	2024-12-15 17:14:14
<input type="checkbox"/>	modifica	15	curiosidadesoautomobilismo	<a href="https://www.threads.net/@curiosidadesoautomobilismo">https://www.threads.net/@curiosidadesoautomobilismo</a>	<a href="https://www.threads.net/@curiosidadesoautomobilismo/post/DDkfakSJwp2">https://www.threads.net/@curiosidadesoautomobilismo/post/DDkfakSJwp2</a>	2024-12-15 17:14:14
<input type="checkbox"/>	modifica	16	winter_aespas	<a href="https://www.threads.net/@winter_aespas">https://www.threads.net/@winter_aespas</a>	<a href="https://www.threads.net/@winter_aespas/post/DDDgkFz_yu">https://www.threads.net/@winter_aespas/post/DDDgkFz_yu</a>	2024-12-15 17:14:14
<input type="checkbox"/>	modifica	17	kace_mcdonald	<a href="https://www.threads.net/@kace_mcdonald">https://www.threads.net/@kace_mcdonald</a>	<a href="https://www.threads.net/@kace_mcdonald/post/DDKts08Rd3O">https://www.threads.net/@kace_mcdonald/post/DDKts08Rd3O</a>	2024-12-15 17:14:14
<input type="checkbox"/>	modifica	18	u_tamanvlog	<a href="https://www.threads.net/@u_tamanvlog">https://www.threads.net/@u_tamanvlog</a>	<a href="https://www.threads.net/@u_tamanvlog/post/DDmJLGczOV2">https://www.threads.net/@u_tamanvlog/post/DDmJLGczOV2</a>	2024-12-15 17:14:14
<input type="checkbox"/>	modifica	19	sportskeeda1	<a href="https://www.threads.net/@sportskeeda1">https://www.threads.net/@sportskeeda1</a>	<a href="https://www.threads.net/@sportskeeda1/post/DDle96GP8Td">https://www.threads.net/@sportskeeda1/post/DDle96GP8Td</a>	2024-12-15 17:14:14
<input type="checkbox"/>	modifica	20	paddockpassion	<a href="https://www.threads.net/@paddockpassion">https://www.threads.net/@paddockpassion</a>	<a href="https://www.threads.net/@paddockpassion/post/DDkswAzN1Nu">https://www.threads.net/@paddockpassion/post/DDkswAzN1Nu</a>	2024-12-15 17:14:14

Figura 6.3: Dati contenuti nella tabella `posts`

SELECT * FROM `post_users` LIMIT 50 (0.001 s) Modifica					
	Modif	<b>id</b>	<b>username</b>	<b>img_profile</b>	<b>user_link</b>
<input type="checkbox"/>	modifica	1	kristophershinn	<a href="https://instagram.fmxp12-1.fna.firebaseio.net/v/t51.2885-19/351045258_653848513230152_6190189031999161390...">https://instagram.fmxp12-1.fna.firebaseio.net/v/t51.2885-19/351045258_653848513230152_6190189031999161390...</a>	<a href="https://www.threads.net/@kristophershinn">https://www.threads.net/@kristophershinn</a>
<input type="checkbox"/>	modifica	2	80s.deennice	<a href="https://instagram.fmxp12-1.fna.firebaseio.net/v/t51.2885-19/358511244_20102196261075_789848910307222371...">https://instagram.fmxp12-1.fna.firebaseio.net/v/t51.2885-19/358511244_20102196261075_789848910307222371...</a>	<a href="https://www.threads.net/@80s.deennice">https://www.threads.net/@80s.deennice</a>
<input type="checkbox"/>	modifica	3	rgiii	<a href="https://instagram.fmxp12-1.fna.firebaseio.net/v/t51.2885-19/357761108_289938913404122_1161578299969560795...">https://instagram.fmxp12-1.fna.firebaseio.net/v/t51.2885-19/357761108_289938913404122_1161578299969560795...</a>	<a href="https://www.threads.net/@rgiii">https://www.threads.net/@rgiii</a>
<input type="checkbox"/>	modifica	4	astronomiaum	<a href="https://instagram.fmxp12-1.fna.firebaseio.net/v/t51.2885-19/358002232_285416914007755_9130725633918787732...">https://instagram.fmxp12-1.fna.firebaseio.net/v/t51.2885-19/358002232_285416914007755_9130725633918787732...</a>	<a href="https://www.threads.net/@astronomiaum">https://www.threads.net/@astronomiaum</a>
<input type="checkbox"/>	modifica	5	hunterwalk	<a href="https://instagram.fmxp12-1.fna.firebaseio.net/v/t51.2885-19/357916634_822705579422959_3172389966570531096...">https://instagram.fmxp12-1.fna.firebaseio.net/v/t51.2885-19/357916634_822705579422959_3172389966570531096...</a>	<a href="https://www.threads.net/@hunterwalk">https://www.threads.net/@hunterwalk</a>
<input type="checkbox"/>	modifica	6	101espn	<a href="https://instagram.fmxp12-1.fna.firebaseio.net/v/t51.2885-19/358040065_226567873667353_1812718187382787177...">https://instagram.fmxp12-1.fna.firebaseio.net/v/t51.2885-19/358040065_226567873667353_1812718187382787177...</a>	<a href="https://www.threads.net/@101espn">https://www.threads.net/@101espn</a>
<input type="checkbox"/>	modifica	7	abc7newbayarea	<a href="https://instagram.fmxp12-1.fna.firebaseio.net/v/t51.2885-19/375395112_1382132682646496_752075938833366075...">https://instagram.fmxp12-1.fna.firebaseio.net/v/t51.2885-19/375395112_1382132682646496_752075938833366075...</a>	<a href="https://www.threads.net/@abc7newbayarea">https://www.threads.net/@abc7newbayarea</a>
<input type="checkbox"/>	modifica	8	vciuu	<a href="https://instagram.fmxp12-1.fna.firebaseio.net/v/t51.2885-19/469312632_584138554321679_8899602388970488877...">https://instagram.fmxp12-1.fna.firebaseio.net/v/t51.2885-19/469312632_584138554321679_8899602388970488877...</a>	<a href="https://www.threads.net/@vciuu">https://www.threads.net/@vciuu</a>
<input type="checkbox"/>	modifica	9	darksideofthemat	<a href="https://instagram.fmxp12-1.fna.firebaseio.net/v/t51.2885-19/425330734_919381422888460_5764060184733658174...">https://instagram.fmxp12-1.fna.firebaseio.net/v/t51.2885-19/425330734_919381422888460_5764060184733658174...</a>	<a href="https://www.threads.net/@darksideofthemat">https://www.threads.net/@darksideofthemat</a>
<input type="checkbox"/>	modifica	10	dan_regan_comedy	<a href="https://instagram.fmxp12-1.fna.firebaseio.net/v/t51.2885-19/449698377_1024623285742722_871475856593550515...">https://instagram.fmxp12-1.fna.firebaseio.net/v/t51.2885-19/449698377_1024623285742722_871475856593550515...</a>	<a href="https://www.threads.net/@dan_regan_comedy">https://www.threads.net/@dan_regan_comedy</a>

Figura 6.4: Dati contenuti nella tabella `post_users` 1/2

bio	followers	created	updated
Seattle, WA Prints and calendars available below!	30.511	2024-12-15 17:15:06	2024-12-15 17:15:06
"Life moves pretty fast. If you don't stop and look around once in a while, you could miss it." -Fe...	205.345	2024-12-15 17:16:45	2024-12-15 17:16:45
Trust God-Thrower of Footballs-Speaker of Sports-Racer of Flying Hawks -Husband gretegii ❤️ -Father ...	163.76	2024-12-15 17:18:30	2024-12-15 17:18:30
⚡   A maior e melhor página de astronomia do Threads brasileiro	195.477	2024-12-15 17:20:51	2024-12-15 17:20:51
📍	5406	2024-12-15 17:22:35	2024-12-15 17:22:35
Sports Talk For St. Louis 101.1 FM	3067	2024-12-15 17:24:32	2024-12-15 17:24:32
#1 source for breaking news, weather, sports and more around the San Francisco Bay Area. ⬇️ Downloa...	73.798	2024-12-15 17:26:15	2024-12-15 17:26:15
Simple life	7810	2024-12-15 17:27:49	2024-12-15 17:27:49
Bay Area   Data @ IG #153 on threads. Opinions are my own.	958	2024-12-15 17:29:30	2024-12-15 17:29:30
Gen X 🌏 Marriage 🌏 Misc For Business & Collaboration 💬 danregancomedy@gmail.com	15.275	2024-12-15 17:32:25	2024-12-15 17:32:25

Figura 6.5: Dati contenuti nella tabella post\_users 2/2

SELECT * FROM "posts_info" LIMIT 50 (0.001 s) Modifica			
Modify	id	post_id	caption
modifica	1	1	Bainbridge Island 🛰 Seattle Threads
modifica	2	2	'Santa Claus Is Comin' to Town' first aired on ABC 54 years ago today on December 14, 1970. This hol...
modifica	3	3	How you treat people is more important than how many likes you get on social media.
modifica	4	4	Estamos sozinhos no universo?
modifica	5	5	If I worked at a large venture fund and left, I'd let them pay me a bonus to attribute all the bad d...
modifica	6	6	BLUES GAME DAY ON 101 ESPN ↳ stiblues vs. Stars 🕒 Pregame 6 PM   Puck Drop 7 PM 📍 American Airlines C...
modifica	7	7	A rare Tornado Warning was issued for all of San Francisco Saturday morning as a strong Level 3 stor...
modifica	8	8	♥️
modifica	9	9	My dad just took a piece of bread, buttered it... then salted it ☺️
modifica	10	10	The people who would "Bust A Move" and "Jump Around" in the 90s now have house Ibuprofen and car Ibu...
modifica	11	11	While edtech startups have mostly fizzled after the pandemic, Duolingo is doing better than ever. Du...
modifica	12	12	2024 Author Wrapped! 🎉
modifica	13	13	A beautiful photo for a wonderful Album cover. Nancy Wilson 'Hollywood My Way' (1963)
modifica	14	14	▲ ▲ ▲
modifica	15	15	Lotus Elise GT1 🚗 Lotus V8 TwinTurbo 1997
modifica	16	16	NULL
modifica	17	17	Does anyone else rewatch Game of Thrones? I know how it ends, but I still go back to season one to r...
modifica	18	18	Subaru

Pagina — Intero risultato — Modificare — Selected (0) — Esporta (140) —  
1 2 3 | □ 140 righe | Salva | Modifica | Clona | Elimina |

Figura 6.6: Dati contenuti nella tabella posts\_info 1/2

link_videos	comments	created
null	[{"Beautiful! 😍"}]	2024-12-15 17:16:09
{"https://instagram.fxp12-1.fna.firebaseio.net/v/t16/f2/m69/AQmU_Nz97GRhccuV-B2HMy10AJ1ASqWbIT7caL...	["Watch till the end for a bumper and Cabbage Patch Kids commercial from this 1985 recording 🎵", "I w...	2024-12-15 17:16:09
null	["👉👉👉", "Well said RD!", "Yea but pickleball is life", "Say it louder for the people in the back!!!"]	2024-12-15 17:16:09
null	["M&M's", "Impossible", "S'more", "Hunca", "H&O, vc n tá sozinha", "Não existe vida fora da Terra, então ..."]	2024-12-15 17:16:09
null	["Two envelopes"]	2024-12-15 17:16:09
{"https://instagram.fxp12-1.fna.firebaseio.net/v/t16/f2/m69/AQmU_Nz97GRhccuV-B2HMy10AJ1ASqWbIT7caL...	null	2024-12-15 17:16:09
{"https://instagram.fxp12-1.fna.firebaseio.net/v/t16/f2/m69/AQmU_Nz97GRhccuV-B2HMy10AJ1ASqWbIT7caL...	["Winds over 110 + on the coast with- level 4 hurricane and tornado warning sirens are in effect"]	2024-12-15 17:16:09
null	null	2024-12-15 17:16:09
null	["Unsalted butter?!", "Salted butter 🥀", "totally normal.", "If it was truly normal this post would ..."]	2024-12-15 17:16:09
null	["And work ibuprofen 🌟", "I don't have a car, so I have house (in two places) and backpack ..."]	2024-12-15 17:16:09
null	["The risk for Duolingo is that AI-powered universal translators will make learning Foreign language..."]	2024-12-15 17:17:25
{"https://instagram.fxp12-1.fna.firebaseio.net/v/t16/f2/m69/AQmU_Nz97GRhccuV-B2HMy10AJ1ASqWbIT7caL...	["Holy guacamole! That's some serious work there! I don't think most people know how much blood swea...	2024-12-15 17:17:25
null	null	2024-12-15 17:17:25
null	["How can u trade a person thots out for the year. I never heard of that", "That's crazy. Schroeder ..."]	2024-12-15 17:17:25
null	["Perfect. Did nt see this can before. Beast, rrrrr 🐻", "Обеих этих едак неопредельно, но такое 🐾"]	2024-12-15 17:17:25
null	null	2024-12-15 17:17:25
null	["I go to select episodes or follow particular relationships. I revisit Arya and the Hound's Relatio...	2024-12-15 17:17:25
{"https://instagram.fxp12-1.fna.firebaseio.net/v/t16/f2/m69/AQmU_Nz97GRhccuV-B2HMy10AJ1ASqWbIT7caL...	null	2024-12-15 17:17:25

Figura 6.7: Dati contenuti nella tabella posts\_info 2/2

# CAPITOLO 7

---

## Conclusioni

---

Nel corso di questo lavoro di tesi, è stato progettato e implementato un sistema di scraping automatizzato per la piattaforma Threads. Tale sistema si compone di tre scraper distinti, ciascuno dedicato a specifiche aree della piattaforma: la home page, i profili degli utenti e i post individuali. Ogni scraper è stato realizzato con un'attenzione particolare all'organizzazione modulare del codice e all'efficienza delle operazioni, garantendo la raccolta strutturata dei dati in un database MySQL. La progettazione delle classi e delle funzioni è stata orientata alla manutenibilità del software, mentre la gestione del database è stata curata per rappresentare i dati in modo chiaro e coerente, attraverso tre tabelle principali, che sono `posts`, `post_users` e `posts_info`. Durante lo sviluppo, è stata utilizzata la libreria Selenium per consentire l'interazione dinamica con i contenuti web.

Gli sviluppi futuri di questo lavoro potrebbero includere diverse direzioni. In primo luogo, l'integrazione con un framework come Scrapy potrebbe migliorare ulteriormente l'efficienza del sistema e ridurre la complessità del codice legata alla gestione manuale dei dettagli tecnici dello scraping. In aggiunta, l'adozione di tecniche avanzate di anonimizzazione, come la rotazione dei proxy o l'utilizzo di reti VPN, potrebbe migliorare la robustezza del sistema rispetto ai meccanismi anti-bot delle piattaforme web. Un'altra direzione interessante potrebbe consistere nell'analisi dei dati raccolti tramite strumenti di Data Science e Machine Learning, al fine di estrarre informazioni utili o identificare pattern nascosti. Questi sviluppi, combinati con un'ulteriore ottimizzazione delle prestazioni e della scalabilità, potrebbero rendere il sistema uno strumento ancora più efficace per lo studio e l'analisi delle piattaforme social.