Cairo University
Faculty of Computers and Artificial intelligence
Department of Networks and Cybersecurity

Xtract

Supervised by

Dr. Nour Mahmoud

Implemented by

20205022 Maram Alaa Eldein Abd Elgayed
20205025 Souhila Sherif Mahmoud
20205026 Tasneem Yousry Abd Almageed

Graduation Project

Academic Year 2023-2024

Final Documentation

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Motivation

You can tell a lot about a person from their social media accounts. That would've been debatable if there weren't 2 billion daily active users on Facebook, or over 1 billion daily users on LinkedIn and 335 million users on Twitter. But the question remains, "How much is "a lot" in this context?", "Should I be concerned?", "Don't those platforms have sophisticated, multi-million-worth security measures?". The short answer is yes. The long answer we will explore in this project. Our main goal is to try and scrape data from the above-mentioned platforms using basic tools, minimum technical skills, and little to no paid services. We do that to prove that anyone can do it and hopefully raise awareness of the dangers of having the habit of revealing too much about yourself online. Not to mention that our project can be used for other purposes such as OSINT, threat intelligence, and data analysis.

For this project, we will use Python and Nodejs for data extraction and automation and then save that data in a MongoDB database for faster access in subsequent requests. Our website has numerous filters/search methods the user can choose from to get the most accurate results.

1.2 Background

While conducting searches for ideas related to the cyber security field, we stumbled upon the interesting concept of OSINT (Open-Source Intelligence) , with a simple search on LinkedIn more than 800 OSINT related jobs was found in the US with job titles ranging from intelligence consultant , OSINT investigator and even Interpol related positions, but with the existence of billions of social media users around the world who would also occasionally want to search for specific data with little to no technical background , we decided to provide a tool that would be of help to both technical and non-technical users.

It is A website designed with the aim of making open-source intelligence an easier and more accessible process, it focuses on social media public data collection. Users can search using a person's name to find all possible results of accounts holding the same name, and then can utilize this information to start collecting data from respective user's accounts, they can choose to collect all available data or start using filters to extract what they believe is valuable for their search.

The focus of this project is web scraping through different social media platforms using different languages and libraries, then representing the findings in an easy and user-friendly interface.

1.3 Problem definition

In today's digital landscape, social media platforms have become integral to our daily lives, with millions of users engaging in various activities that generate a vast amount of valuable data. This data, created unconsciously by users through their interactions, posts, and shared content, holds immense potential for insights across numerous fields such as marketing, sociology, psychology, and more. As the number of social

media users continues to rise, with projections indicating significant growth in the coming years, the need to harness this data becomes increasingly vital.
Upon analysing current statistics and reports, we recognized a critical opportunity to develop a solution that effectively gathers and utilises this wealth of information. However, our preliminary research into existing tools and services revealed several limitations. While there are numerous websites, libraries, and tools that partially address the need for social media data collection, they often fall short in key areas:

2. **Platform-Specific Limitations**: Many existing solutions focus on a single social media platform, thereby missing out on the broader picture that can be obtained by aggregating data from multiple sources.
3. **Lack of Aggregation**: Even tools that do collect data from multiple platforms often fail to aggregate this data into a cohesive and unified format, limiting the ability to perform comprehensive analyses.
4. **User Accessibility and Usability**: The existing tools are often designed for developers or data scientists, lacking user-friendly interfaces that allow a broader audience to access and utilise the data effectively.
5. **Data Filtering and Customization**: There is a noticeable gap in providing robust filtering and customization options, which are essential for users who need specific types of data for targeted analyses.

Driven by these insights and the evident gap in the market, we decided to embark on developing an all-in-one web application that addresses these shortcomings. Our project, Xtract, aims to provide a comprehensive solution for collecting, aggregating, and analysing public data from various social media platforms. Key features of our application include:

- **Multi-Platform Integration**: Seamless integration with major social media platforms to gather diverse datasets.
- **Data Aggregation**: Combining data from different platforms into a single, cohesive repository for more comprehensive analysis.

- **User-Friendly Interface**: Designing intuitive and accessible interfaces to ensure usability by a wide range of users, from researchers to marketers.
- **Advanced Filtering and Customization**: Offering powerful tools to filter, customize, and extract the precise data needed for specific purposes.

By addressing these critical gaps, Xtract aims to revolutionise the way social media data is collected and utilised, unlocking new possibilities for insights and applications. This project represents a significant step forward in making social media data more accessible, usable, and valuable for various stakeholders.

1.4 Our solution

The primary objective of our project, Xtract, is to develop a robust and comprehensive web application that seamlessly integrates with multiple social media platforms to collect, aggregate, and analyse publicly available data. By addressing the limitations of existing tools, Xtract aims to revolutionise the way users access and utilize social media data, transforming it into actionable insights. Our solution will be designed with a focus on user-friendliness, ensuring that a wide range of users, from researchers and data scientists to marketers and casual users, can easily navigate and harness the power of the application.

Xtract will feature multi-platform integration, enabling it to gather data from major social media sites like Facebook, Twitter, Instagram, LinkedIn, and more. This data will be aggregated into a single, cohesive repository, allowing for a more holistic view and comprehensive analysis. Our application will provide advanced filtering and customization options, empowering users to extract the precise information they need based on various criteria such as time frame, location, keywords, and user demographics.

In addition to these core functionalities, Xtract will incorporate sophisticated data analysis and visualization tools, presenting the collected data in intuitive formats such as graphs, charts, and interactive dashboards. This will facilitate deeper insights and enable users to identify trends, patterns, and correlations with ease. Our application

will also prioritize data privacy and ethical considerations, ensuring that all collected data is handled in compliance with relevant regulations and best practices.

By achieving these objectives, Xtract aims to fill the current market gap and provide a comprehensive solution that not only simplifies the process of social media data collection but also enhances its utility and accessibility. Ultimately, our goal is to empower users with the tools they need to unlock the full potential of social media data, driving informed decision-making and fostering innovative applications across various domains.

1.5 Gantt chart

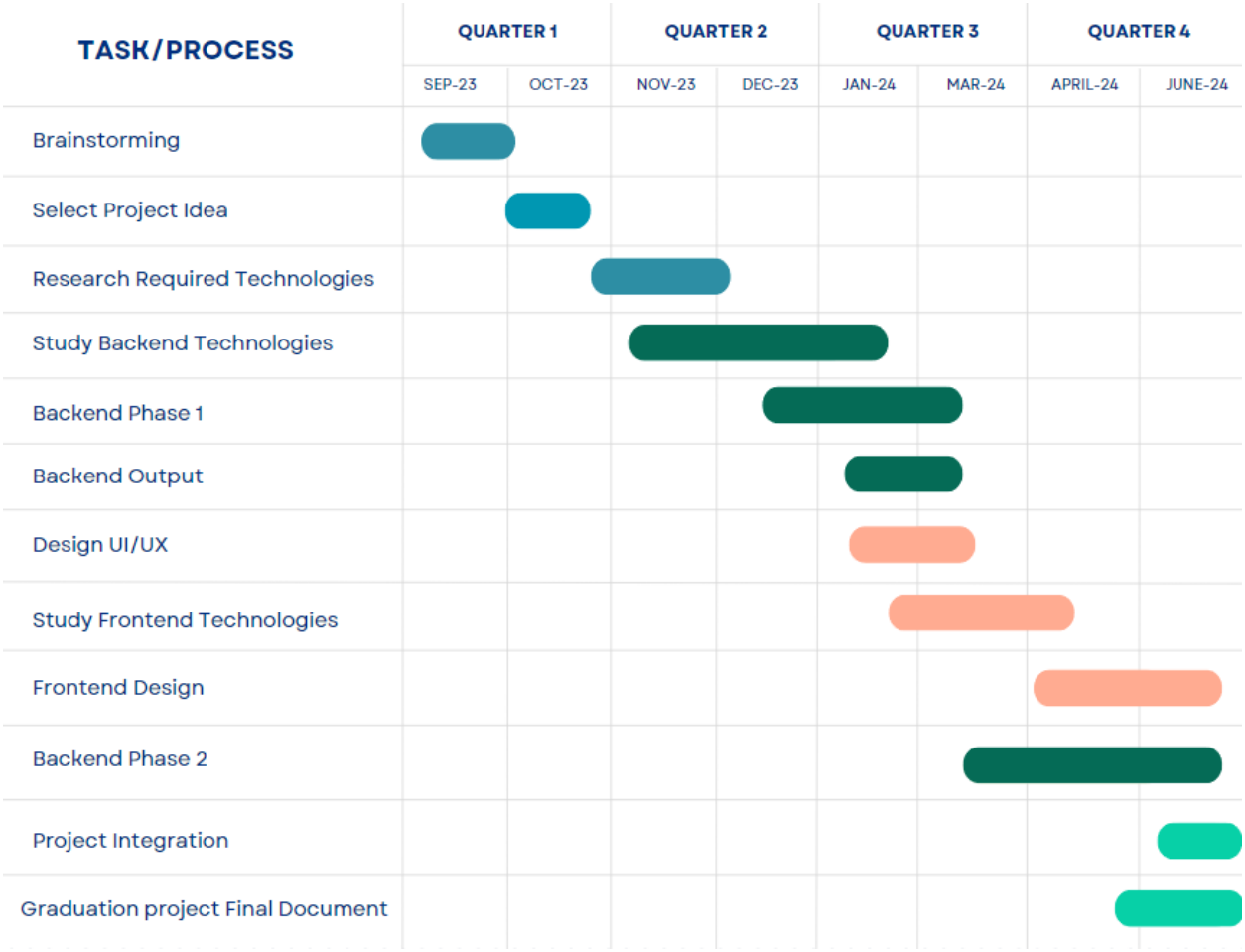


Figure 1 Gantt chart

1.6 Project development methodology

Agile framework

Our development process is grounded in the principles of the Agile framework, which allows us to be flexible, iterative, and responsive to change. Agile helps us maintain a steady pace of development while adapting to new requirements and feedback.

- **Sprints**: We operate in bi-weekly sprints. Each sprint begins with a planning session to define clear objectives and ends with a review meeting to evaluate progress and gather feedback.
- **Sprint Planning**: During sprint planning, we prioritize the backlog, allocate tasks based on team capacity, and set sprint goals. This ensures everyone is on the same page and working towards common objectives.
- **Sprint Reviews**: At the end of each sprint, we hold a review session to showcase completed work, discuss what was achieved, and gather feedback from stakeholders. This helps us ensure we're meeting user needs and project goals.
- **Reviews and Retrospectives**: At the end of each cycle, we review completed work, gather feedback, and reflect on our processes to identify areas for improvement.

Collaboration and Communication

Effective collaboration and communication are critical to our success. We use a variety of tools and practices to ensure our team stays connected, informed, and efficient.

- **Version Control with Git**: Git allows us to manage our codebase collaboratively. Branching, merging, and pull requests help us handle changes smoothly and maintain code quality.

- **Code Reviews**: Regular code reviews are integral to our process. They help us maintain high code quality, share knowledge, and identify potential improvements. Peer feedback ensures robust and maintainable code.

Technology Stack

Our diverse technology stack enables us to build a powerful and flexible platform. Each component is chosen to meet specific needs and optimize performance.

- **Node.js Backend**: Node.js is used for its non-blocking, event-driven architecture, which makes it ideal for handling asynchronous operations and scalable network applications.
- **Python Backend**: Python is used for data processing, automation, and certain backend tasks. Its simplicity and rich ecosystem of libraries make it an excellent choice for various functionalities.
- **Vue.js Front-End**: Vue.js is our chosen framework for building interactive and responsive user interfaces. Its component-based architecture allows for modular and maintainable code.
- **Automation and Scraping Tools**: Puppeteer and Selenium are employed for web scraping and automation. These tools help us collect and process data from various social media platforms efficiently.

Team Responsibilities

To ensure focused expertise and efficient development, each team member is responsible for a specific social media platform. This specialization allows us to leverage in-depth knowledge and tailor our approaches to the unique aspects of each platform.

- **Platform Assignments**: Team members are assigned to different social media platforms, ensuring comprehensive coverage and specialized knowledge. This structure enables us to tackle platform-specific challenges effectively and optimize data collection.

User-Centred Design

Our development process prioritizes the needs and experiences of our users. We employ a user-centred design approach to ensure our platform is intuitive, accessible, and meets user expectations.

- **User Research**: We conduct thorough user research to understand the behaviours, needs, and pain points of our target audience. This informs our design decisions and helps us create user-friendly features.
- **Iterative Development**: We continuously iterate on our features based on testing results. This ensures our platform evolves in line with user needs and provides the best possible experience.

Quality Assurance

Quality is paramount in our development process. We have robust QA practices in place to ensure our platform is reliable, performant, and bug-free.

- **Manual Testing**: In addition to automated tests, our QA team conducts manual testing to cover edge cases and focus on user experience. This comprehensive approach ensures high quality.
- **Performance Monitoring**: We use monitoring tools to track the performance and stability of our platform. This allows us to identify and address issues proactively, ensuring a smooth user experience.

Continuous Improvement

Continuous improvement is a core principle of our methodology. We regularly review our processes, tools, and techniques to identify areas for enhancement and stay updated with industry best practices.

- **Process Reviews**: Regular process reviews help us identify inefficiencies and implement improvements. This ensures our workflows are optimized and effective.

- **Feedback Loop**: We maintain a feedback loop with users and stakeholders to ensure our platform meets their needs and expectations. This ongoing dialogue helps us refine our product and deliver value.

1.7 Used tools in the project

A variety of programming languages, frameworks, and applications are used to build this
website which are listed as follows:

Figma: which is a collaborative web application for interface design that we used in the
UI/UX design process.

HTML&CSS: HTML is the standard markup language for documents designed to be displayed in a web browser. It defines the content and structure of web content accompanied by CSS that is used to specify the presentation and styling of a document written in a markup language which is HTML in our case.

Vue.js: is open-source frontend JavaScript framework for building user interfaces and single-page applications, the version used in this website implementation is Vue 3.

JavaScript: is a programming language and core technology of the World Wide Web, alongside HTML and CSS, it is used in our frontend in the context of using Vue.js and also used in parts of the backend by using the framework Node.js.

Python: Python is a high-level, general-purpose programming language that is used alongside JavaScript on our website to build the backend.

Node.js: is an open-source JavaScript runtime environment, it lets developers use JavaScript to write command line tools and server-side scripting, which means that it is used as the framework to write the JavaScript parts of our backend.

Express.js: it is the framework used on top of Node.js to write the necessary API calls that allows the frontend to communicate with our Node.js backend.

Axios: is the JavaScript library we use in our vue.js frontend to make http requests to communicate with the backend.

Puppeteer: is a Node.js library which provides a chrome browser to run and test your JavaScript codes and requests.

Selenium: is an open-source, automated testing tool used to test web applications across various browsers. It can be used with various languages but in the context of this website it is used with Python.

MongoDB: is a cross-platform, document-oriented database program. Classified as a NoSQL database product, it is used on this website to store the resulted JSON data.

1.8 Report organization

For the rest of the document we will be delving into more details about Xtract as a project , In Chapter 2 we discuss similar solutions that exist in the market and how Xtract is different, in Chapter 3 we get into the actual system analysis that draws the big picture of how the project looks like , while in Chapter 4 we describe the system design that will then be shown as actual implementation in Chapter 5 that also includes project testing steps and results.

Chapter 2: Related Work

The comparison table provides a comprehensive overview of five web scraping platforms: Xtract, Octoparse, ParseHub, Social Scraper, and Twint. Each row represents a key feature or aspect of these platforms, ranging from their ability to target social media platforms to their pricing structure. By evaluating these features side by side, we can easily identify the strengths and weaknesses of each platform, ultimately aiding in their decision-making process when choosing a web scraping solution

Feature	Xtract	Octoparse	ParseHub	Social Scraper	Twint
Support Different Social Media Scraping	✓	✓	✓	✕	✕
Targets Social Media Platforms	✓	✓	✓	✓	✓
User-Friendly GUI	✓	✓	✓	✕	✕
Data Exports Formats (JSON, CSV)	✓	✓	✓	✓	✓
Transparency in Data Handling	✓	✕	✓	✕	✓
Price	✓	✕	✕	✓	✓

Table 1 Related work

Feature Descriptions

- **Support Different Social Media Scraping**: Indicates the platform's ability to scrape data from various social media platforms, essential for comprehensive data gathering.
- **Targets Social Media Platforms**: Shows whether the platform is specifically designed to scrape social media data, which is crucial for targeted data collection.
- **User-Friendly GUI**: A user-friendly graphical user interface makes it easier for users of all skill levels to navigate and use the platform effectively.
- **Data Exports Formats (JSON, CSV)**: The ability to export data in common formats like JSON and CSV allows for easier data manipulation and analysis.
- **Transparency in Data Handling**: Transparency in how data is handled and processed ensures trust and compliance with data protection regulations.
- **Price**: Indicates the cost-effectiveness of the platform, an important factor for budget-conscious users.
- **Statistical Analysis of CSV Data (Planned)**: Xtract aims to provide statistical analysis features for data exported in CSV format. This capability will allow users to gain deeper insights and extract meaningful patterns from their scraped data, enhancing the platform's analytical capabilities.

These planned features position Xtract as a more comprehensive and forward-looking solution for users seeking a web scraping platform that not only meets their current needs but also offers advanced analytical capabilities and future enhancements.

Chapter 3: System Analysis

3.1 Project specifications

3.1.1 System Architecture

The system is a client-server layered architecture.

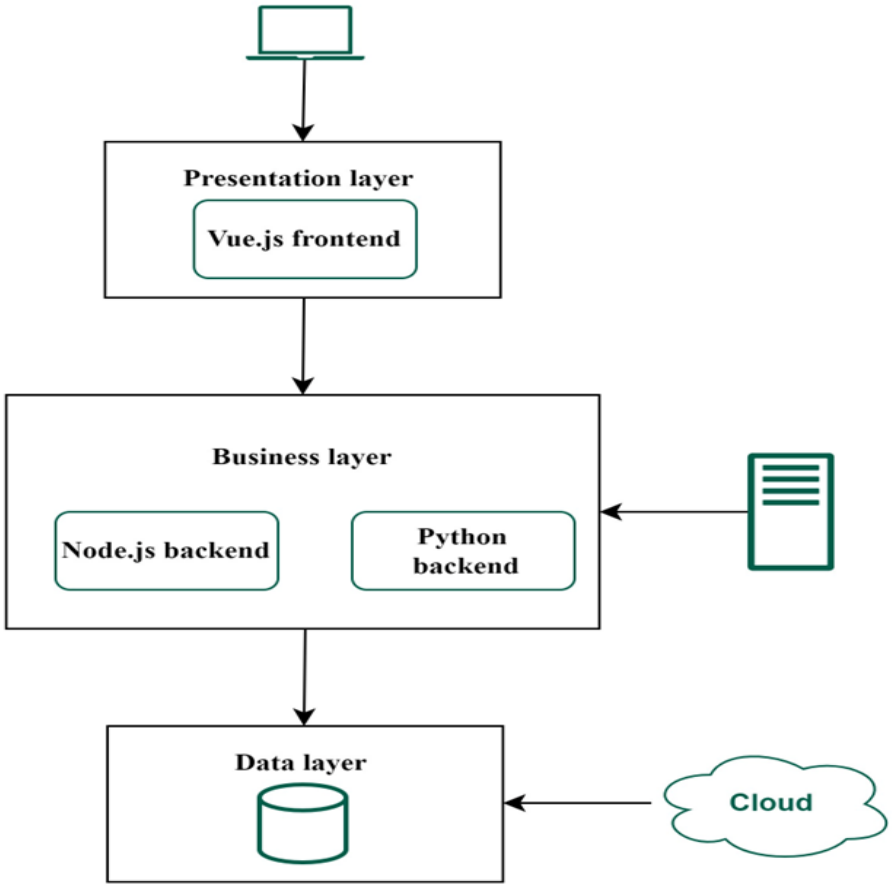


Figure 2 System Architecture

3.1.2 Stakeholders

- **Users:**
    - **Researchers and Data Scientists:** Seek comprehensive and reliable data for academic studies, market research, or trend analysis. They value robust data collection methods, advanced analytics capabilities, and customizable filtering options.
    - **Marketers:** Use social media insights to refine advertising strategies, understand audience behaviour, and measure campaign effectiveness. They prioritize user-friendly interfaces, real-time data updates, and integration with marketing tools.
    - **Casual Users:** Interested in exploring social media trends, discovering content, or monitoring personal or professional profiles. They appreciate intuitive navigation, simplified data visualization, and privacy controls.
- **Businesses and Organizations:**
    - **Market Research and Strategy Teams:** Depend on accurate and timely social media data to identify market trends, consumer preferences, and competitive insights. They value comprehensive data coverage across multiple platforms and actionable insights.
    - **Digital Marketing Agencies:** Utilize social media analytics to optimize client campaigns, enhance audience engagement, and measure return on investment (ROI). They seek integration capabilities with existing marketing platforms and customizable reporting features.
- **Developers:**
    - With a user-friendly UI and comprehensive API documentation, Xtract caters to developers with varying levels of experience in data extraction and analysis. The platform simplifies the integration of social media data into applications, enabling developers to build data-rich solutions without needing extensive expertise in web scraping or data aggregation. Xtract's robust API and flexible configuration options allow developers to focus on creating innovative applications and features with confidence.

- **Investors and Stakeholders:**
    - Investors and stakeholders are interested in the growth potential and market positioning of Xtract. They evaluate the platform's innovative features, market demand, and competitive advantage. Xtract's focus on user-friendliness, advanced data analysis, and compliance with data privacy regulations positions it as a valuable investment opportunity. Comprehensive reports and insights provided by Xtract give investors the confidence to support and fund the platform's development and expansion.

3.1.3 Functional requirements

1. Basic Requirements:

1.1 Login and Registration:

- The user should be able to register with a username and password.
- The user can login if he already has an account registered with us.
- The user doesn't have to login every time he visits the website, the user's session is stored and saved with an expiry timer.

1.2 User Main operations:

- The user can choose from the available services (LinkedIn, Twitter, Facebook, cross-platform search).

Twitter Search Service:

- The user can search in twitter by a keyword and get the tweets where that keyword is mentioned or by a username and retrieve the tweets of that user.
- The user can filter the results by date, time, keyword and number of tweets.

LinkedIn Search Service:

- The user can search in LinkedIn by a username and retrieve profile attributes such as account name, subtitle, about, education and experience section and posts.
- The user can control what information he gets and filter the posts by date, keyword and amount.
- The user can search by keyword and get posts where that keyword appeared.
- The user can filter the results by date and amount.

Facebook Search Service:

- The user can search in Facebook with a username and get information about the account such as education, work, living location, contact info, relationship, bio, events, and other basic info.
- The user can also get the account's photos and posts.
- The user can search by a keyword and get public posts where that keyword is mentioned.
- The user can filter posts by date and amount.

Cross-Platform Search Service:

- The user can look for a person in the three platforms at once by entering the person's name.

1.3 View and edit profile:

- The user can change his password.
- The user can view his search history.

2. I/O Requirement:

    - The user can download the results as CSV or JSON files.

3. Statistical and Graphical Requirements:

    - The user can view different statistics about his account and our website

3.1.4 Non-functional requirements

- User friendly: it is a website with easy-to-understand options and filters.

- Scalability: the layered nature of the architecture makes it easy to add different components and services in the future.

- Performance: the data gathering process takes relatively more time but should not exceed a couple of minutes, while the retrieval of data from MongoDB should take from 3 to 5 seconds, also MongoDB allows large volumes of data.

- Security: the users should be able to authenticate as legitimate users, and the website code should be immune to known vulnerabilities like XSS and CSRF.

3.2 Use case diagrams

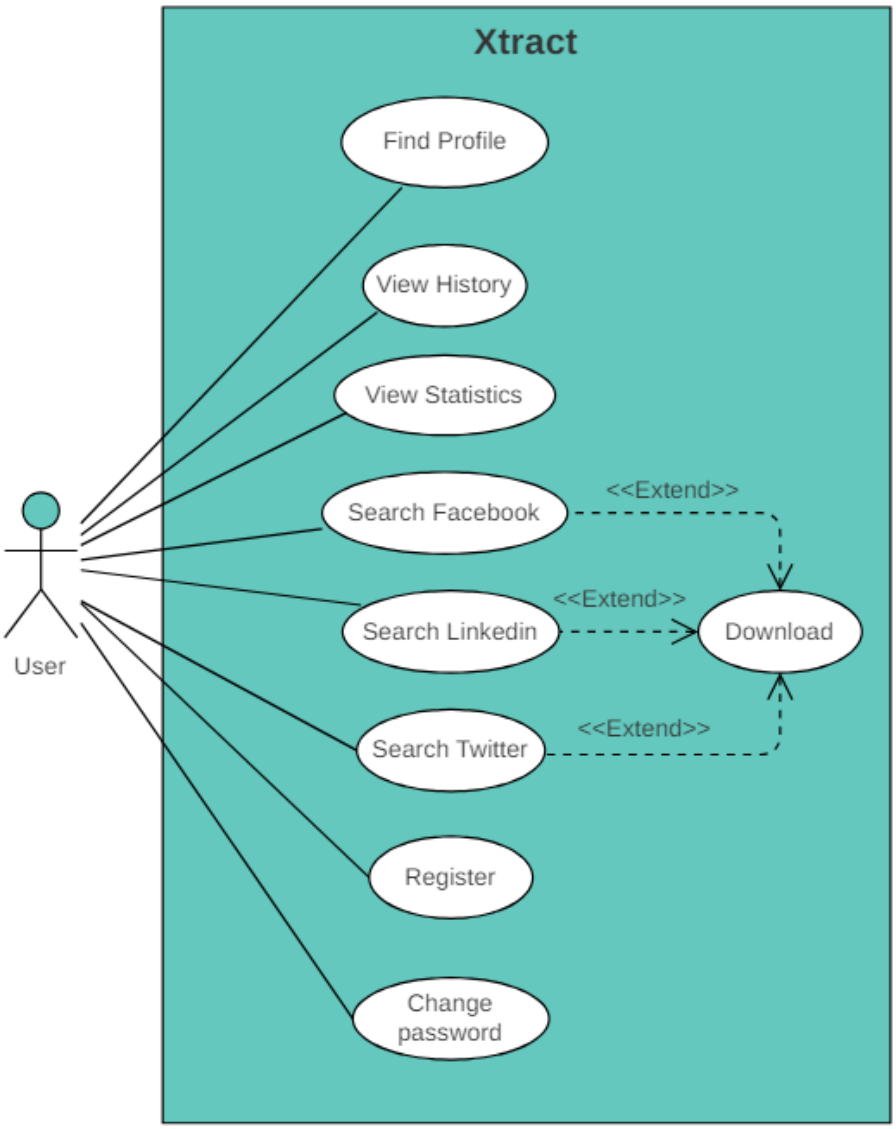


Figure 3 Use case diagram

Chapter 4: System Design

4.1 System component diagram

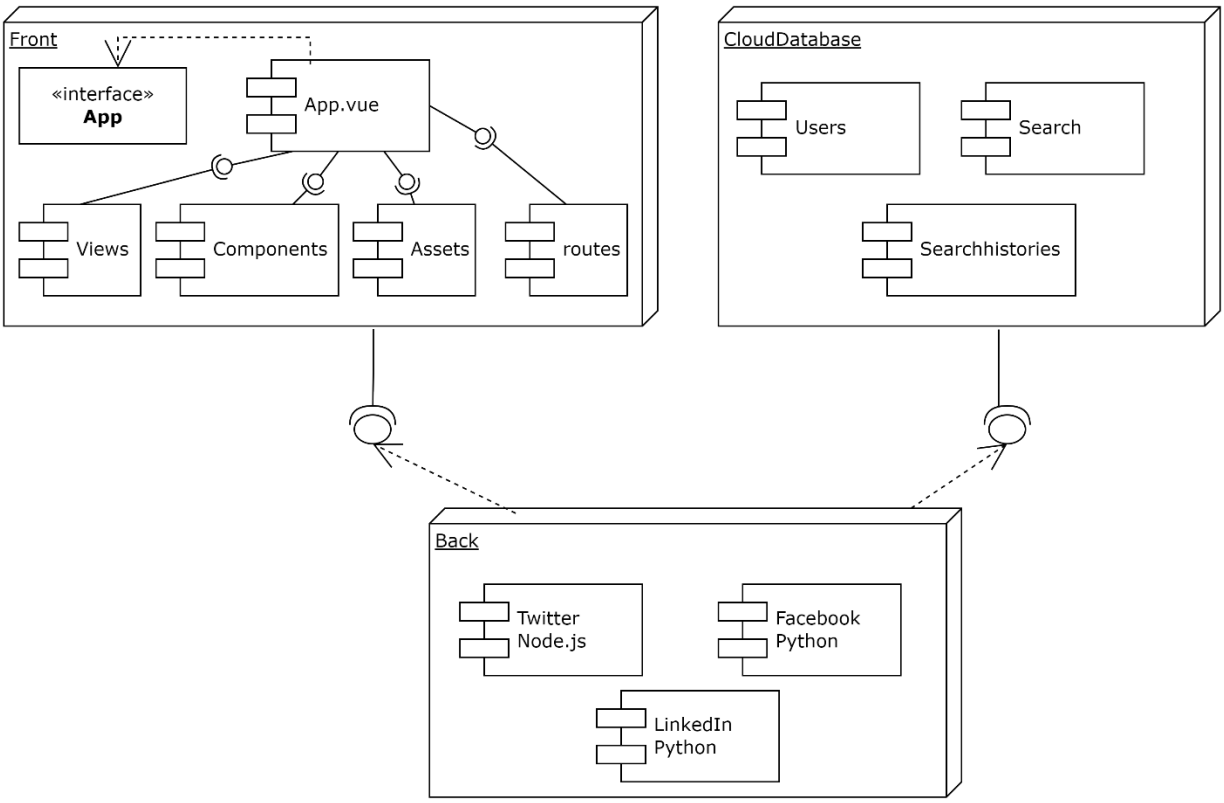


Figure 4 System component diagram

4.2 System class diagrams

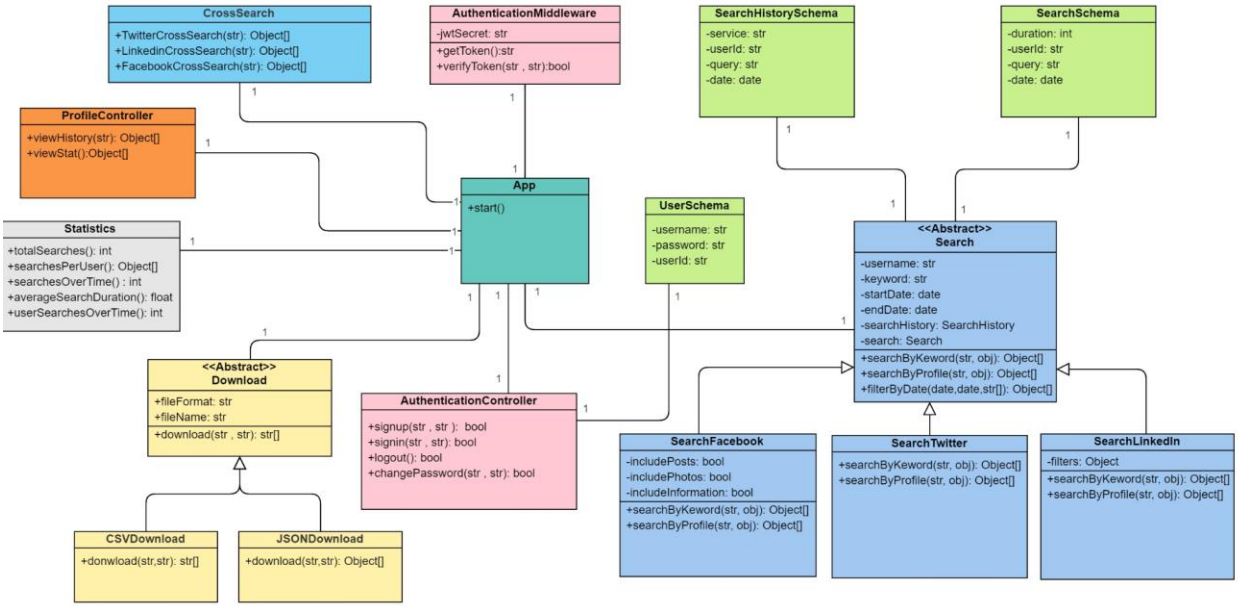


Figure 5 Class diagram

4.3 Sequence diagrams

4.3.1. Sign up

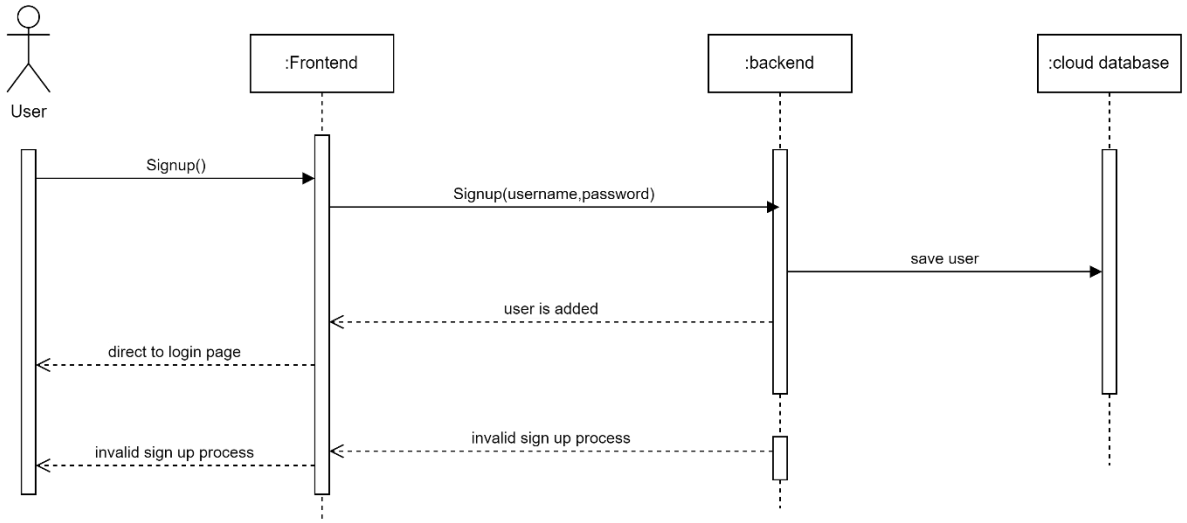


Figure 6 Sign up sequence diagram

4.3.2. Login

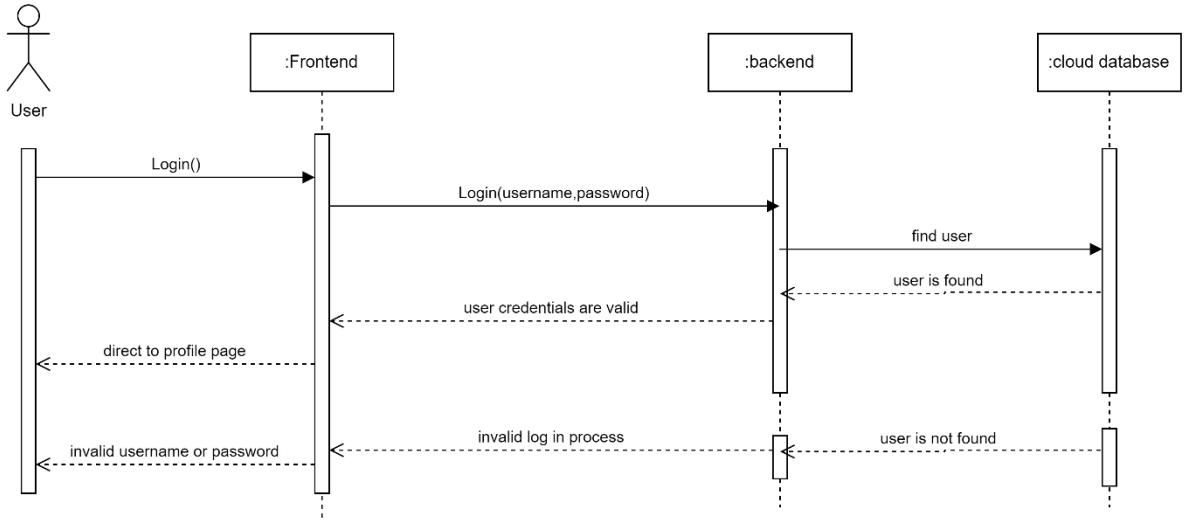


Figure 7 Log in sequence diagram

4.3.3. Cross-platform Profile search

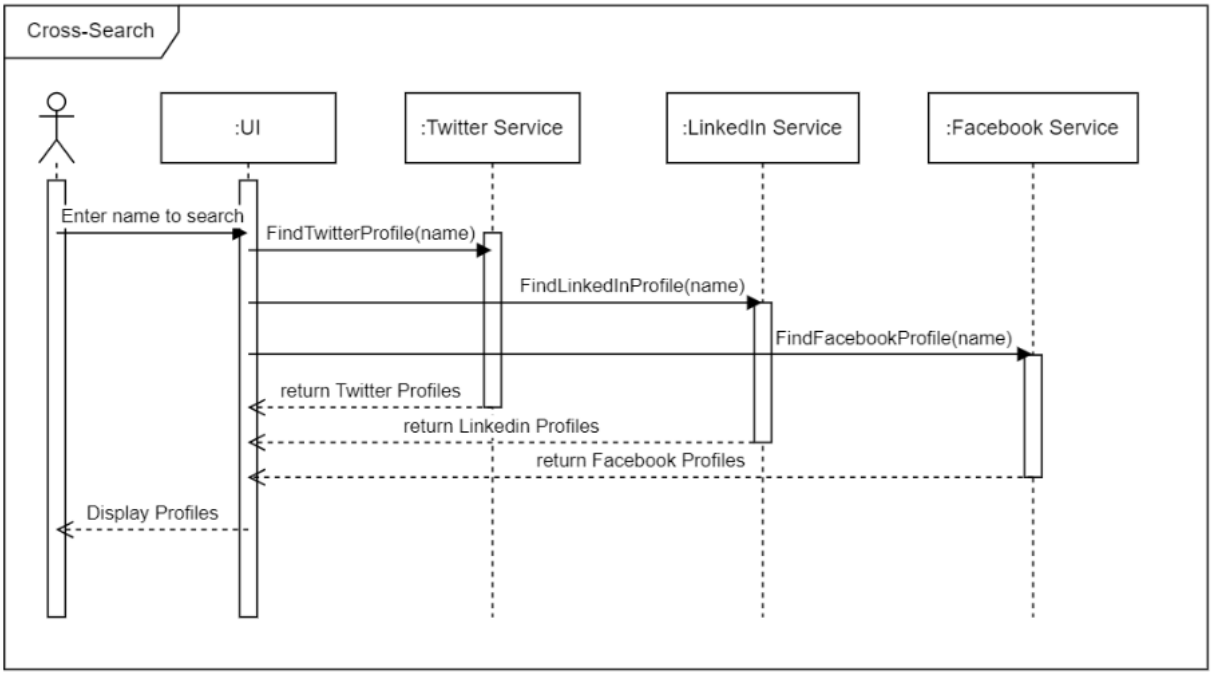


Figure 8 cross platform sequence diagram

4.3.4. Get user tweets

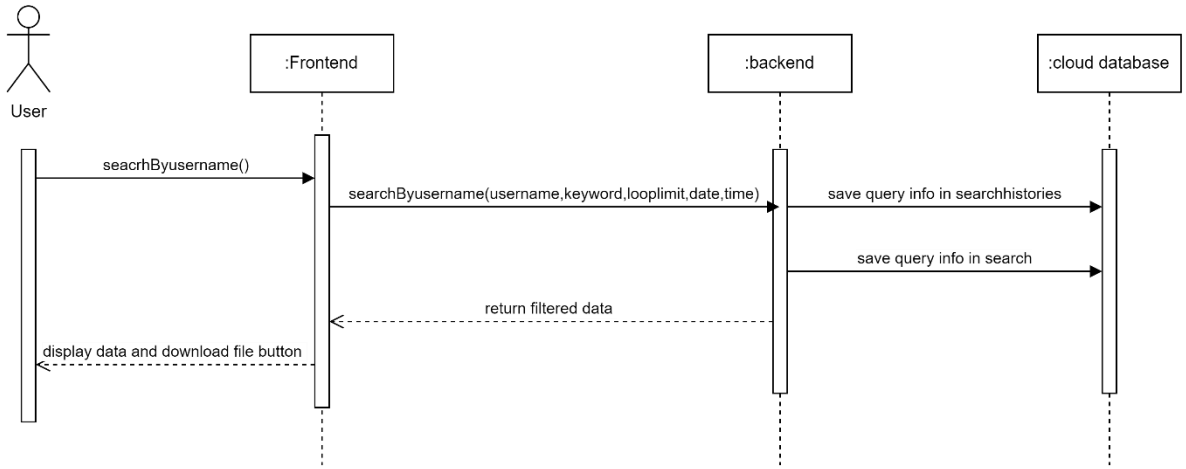


Figure 9 Get user tweets sequence diagram

4.3.5. Get keyword tweets

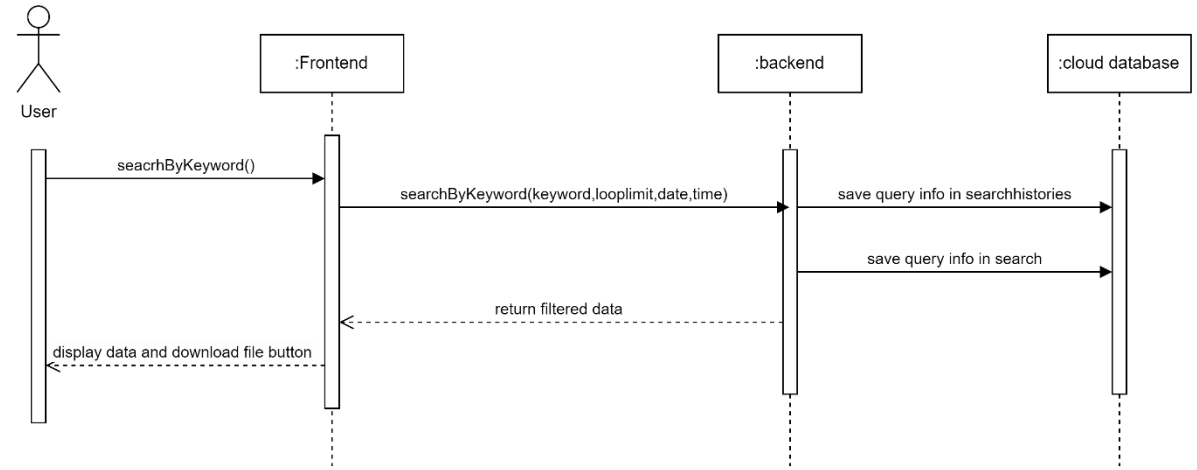


Figure 10 Get keyword tweets sequence diagram

4.3.6. Get User Facebook Data

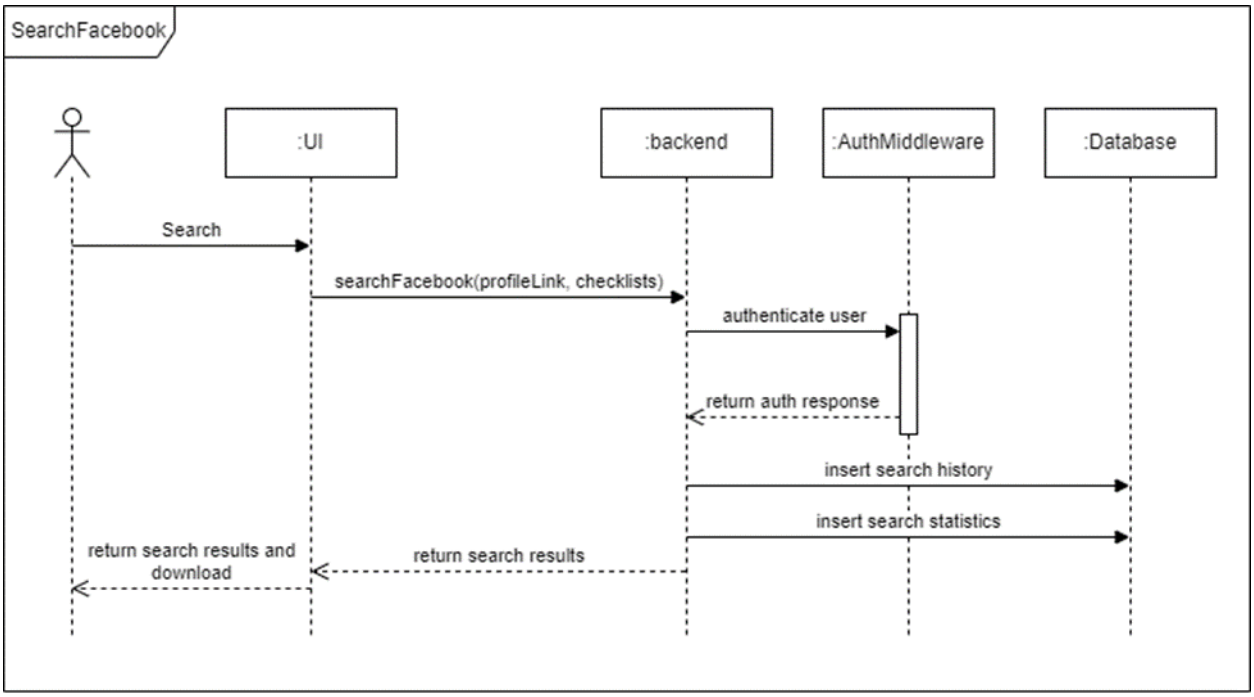


Figure 11 Get Facebook user data sequence diagram

4.3.7. Search Keyword in Facebook

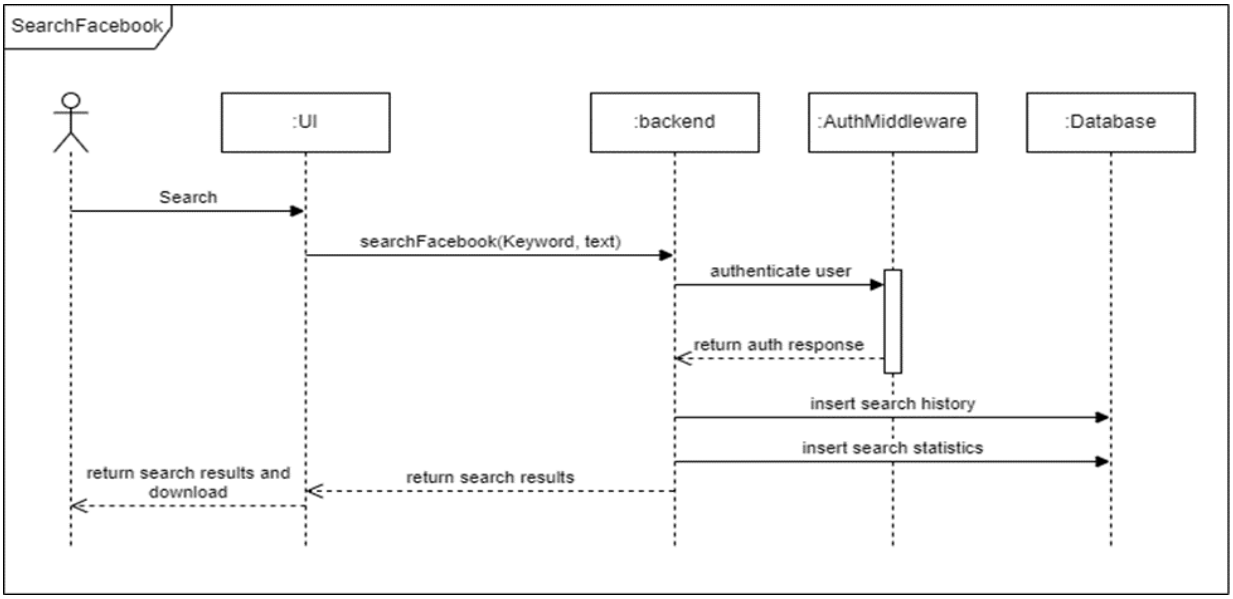


Figure 12 search keyword in Facebook sequence diagram

4.3.8. Search LinkedIn (By Username)

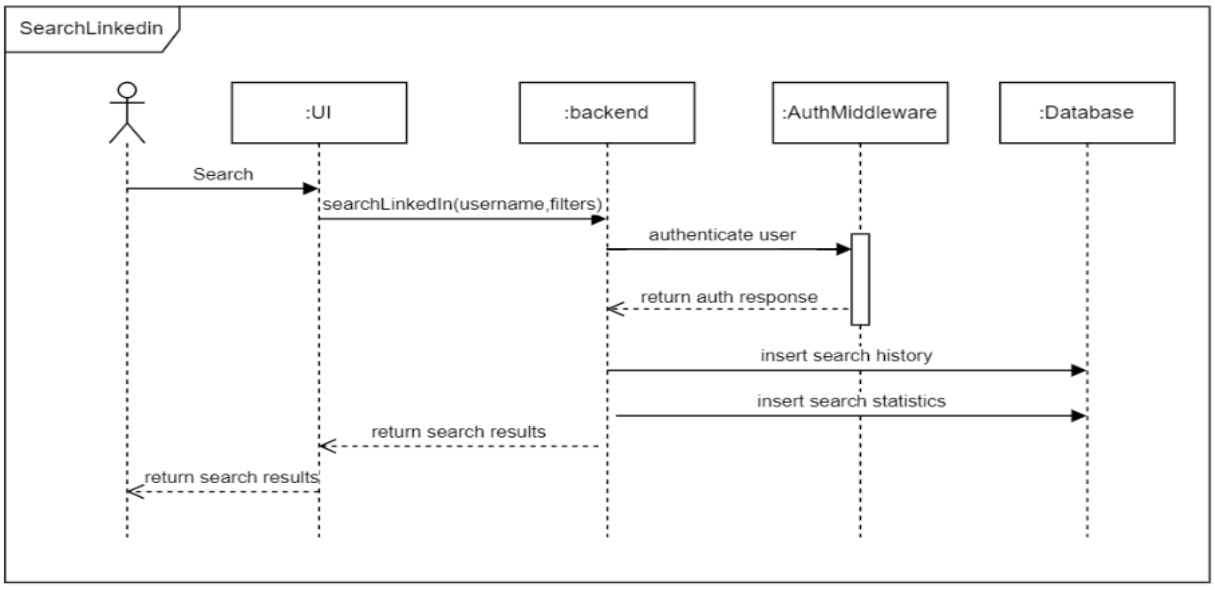


Figure 13 Search LinkedIn by username sequence diagram

4.3.9. View History

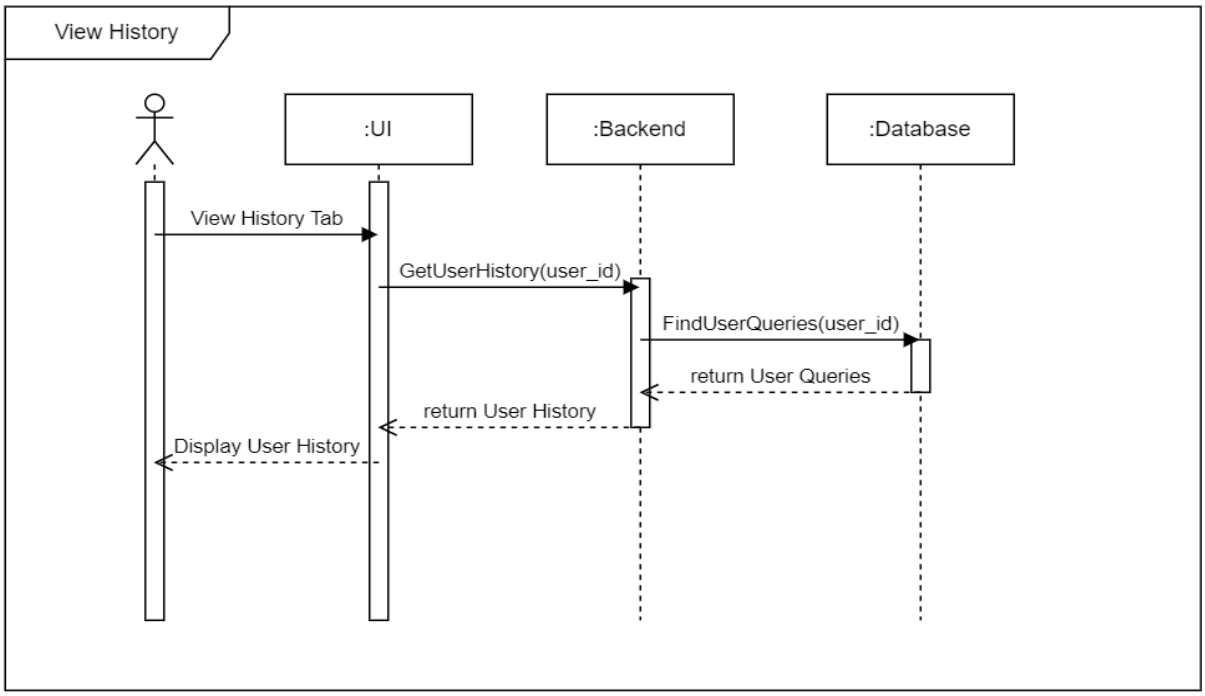


Figure 14 View history sequence diagram

4.4 Project ERD

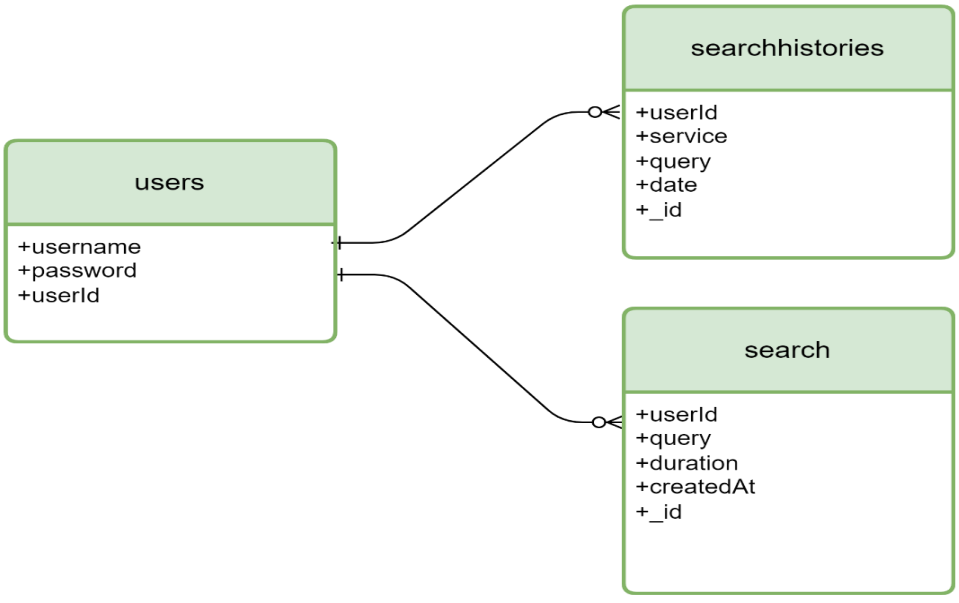


Figure 15 ERD

4.5 System GUI design

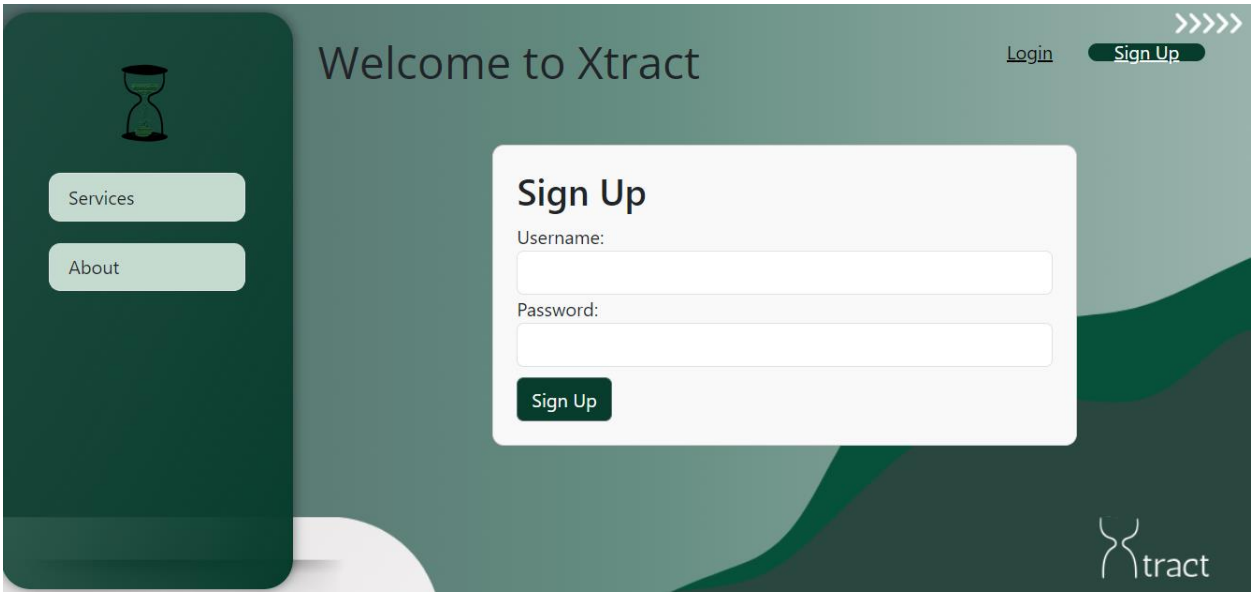


Figure 16 Sign up GUI

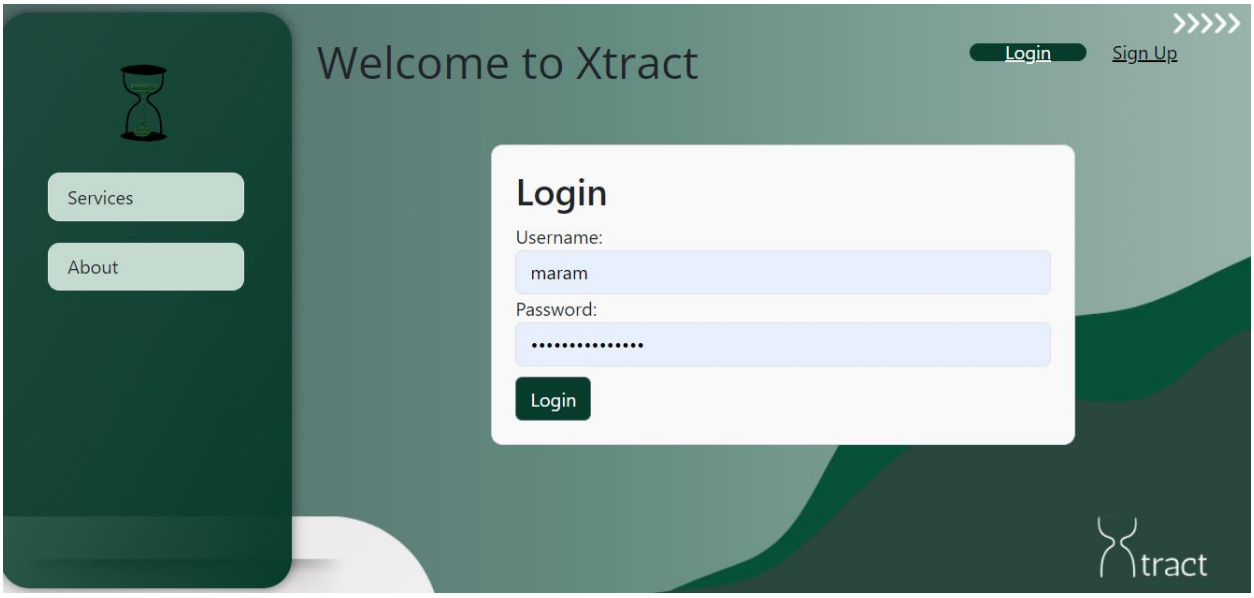


Figure 17 Log in GUI

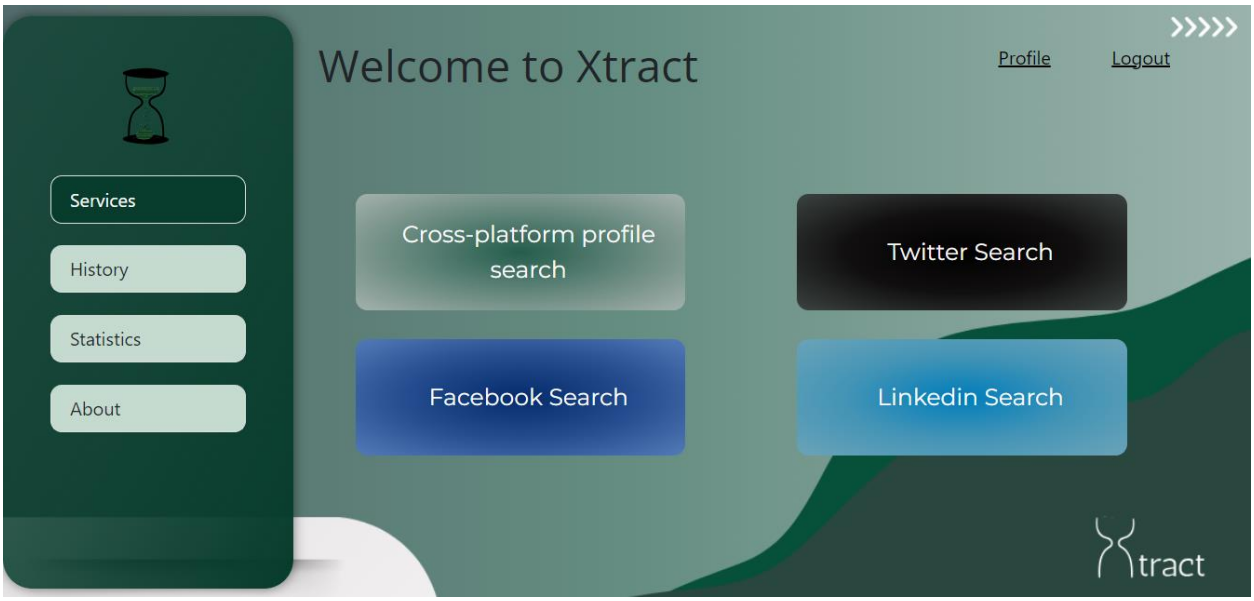


Figure 18 Services page GUI

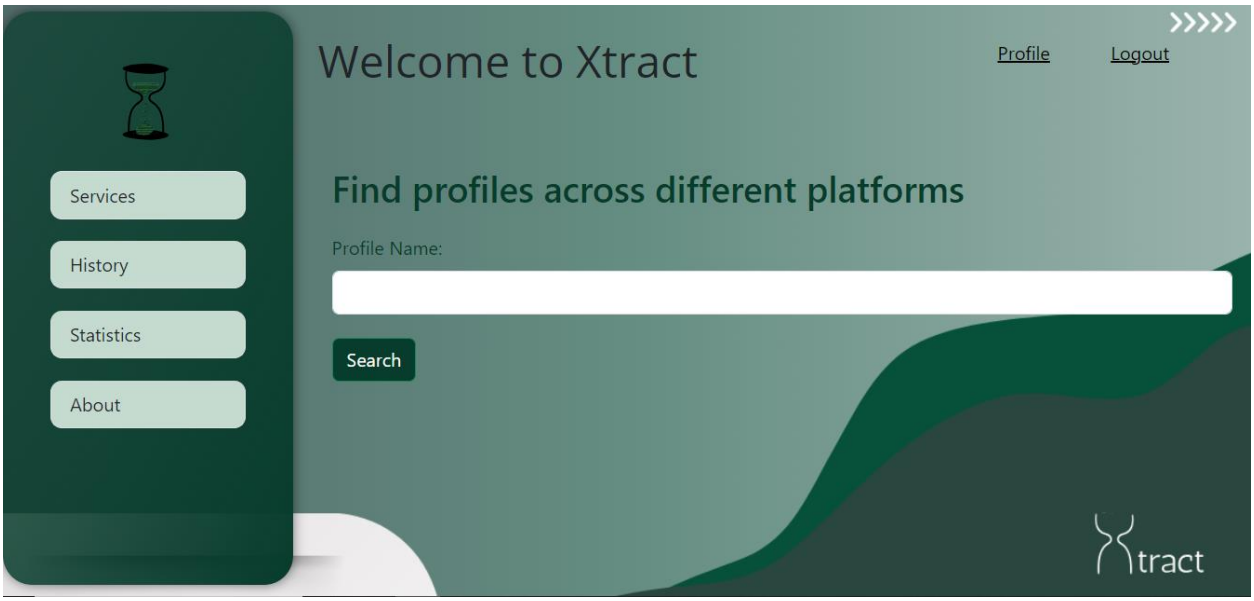


Figure 19 Cross platform GUI

Figure 20 About us page GUI

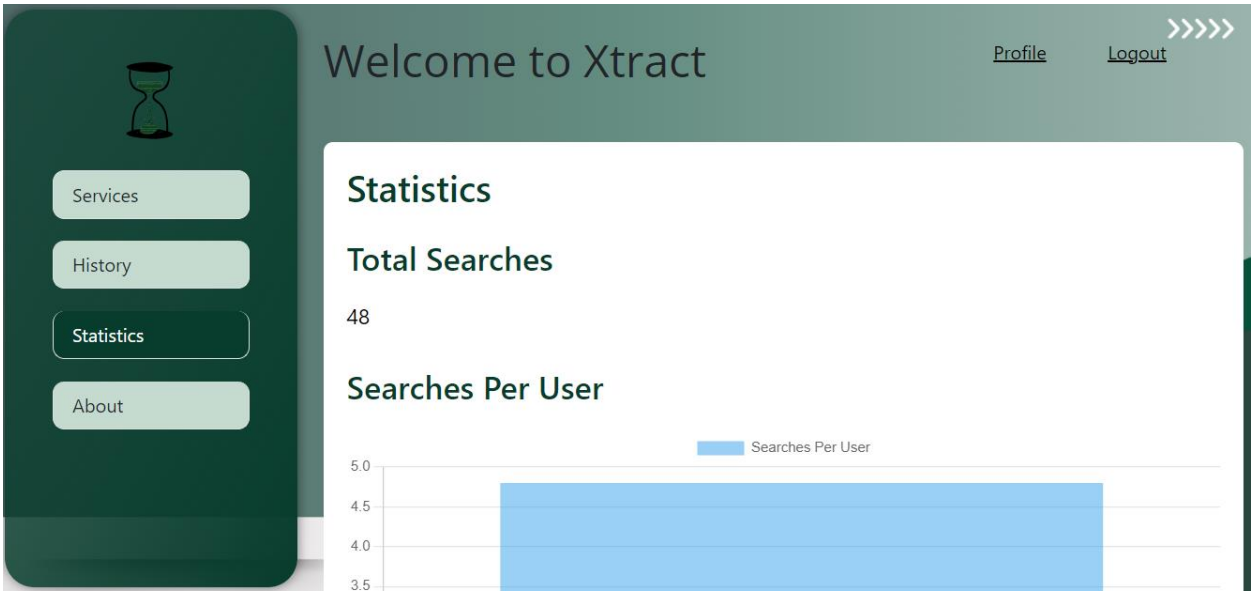


Figure 21 Statistics page GUI

Figure 22 History page GUI

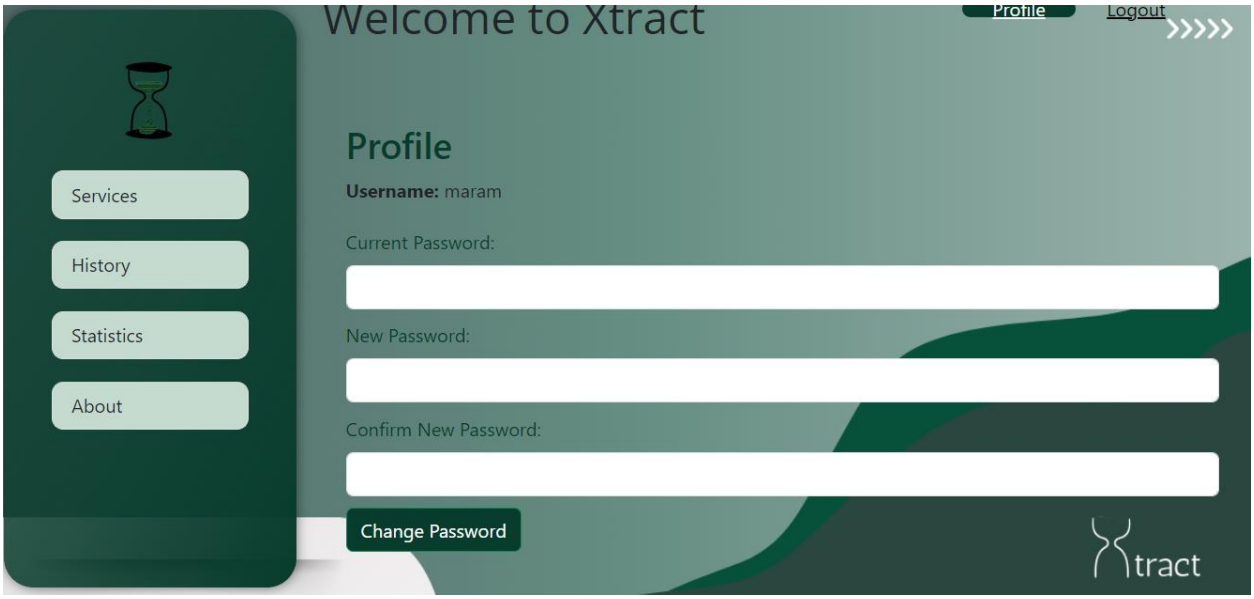


Figure 23 Profile page GUI

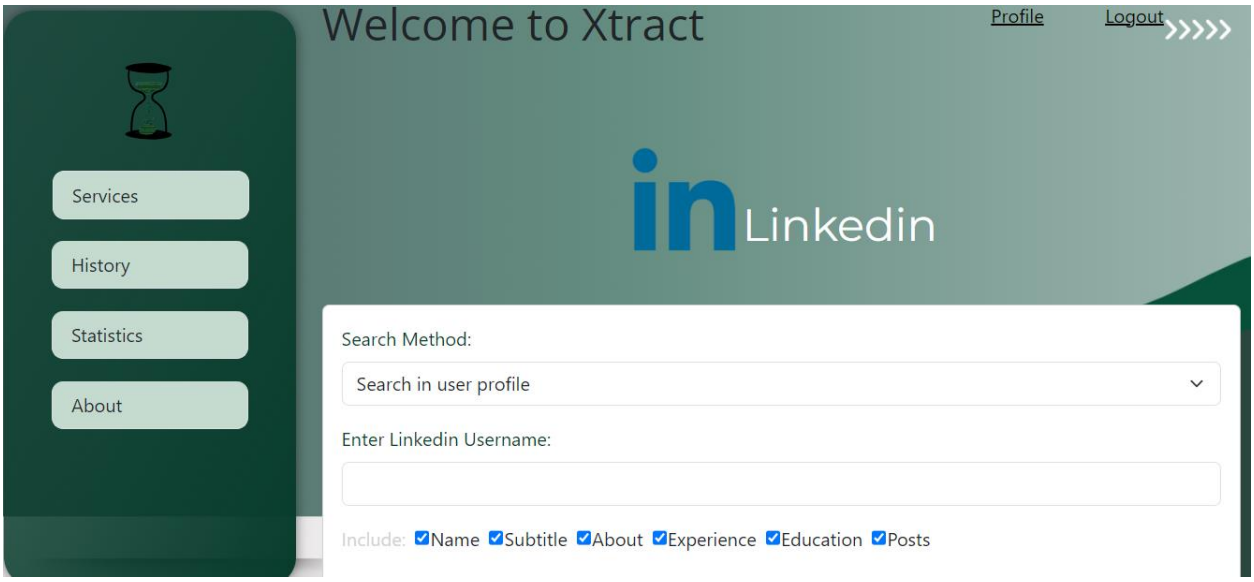


Figure 24 LinkedIn search GUI

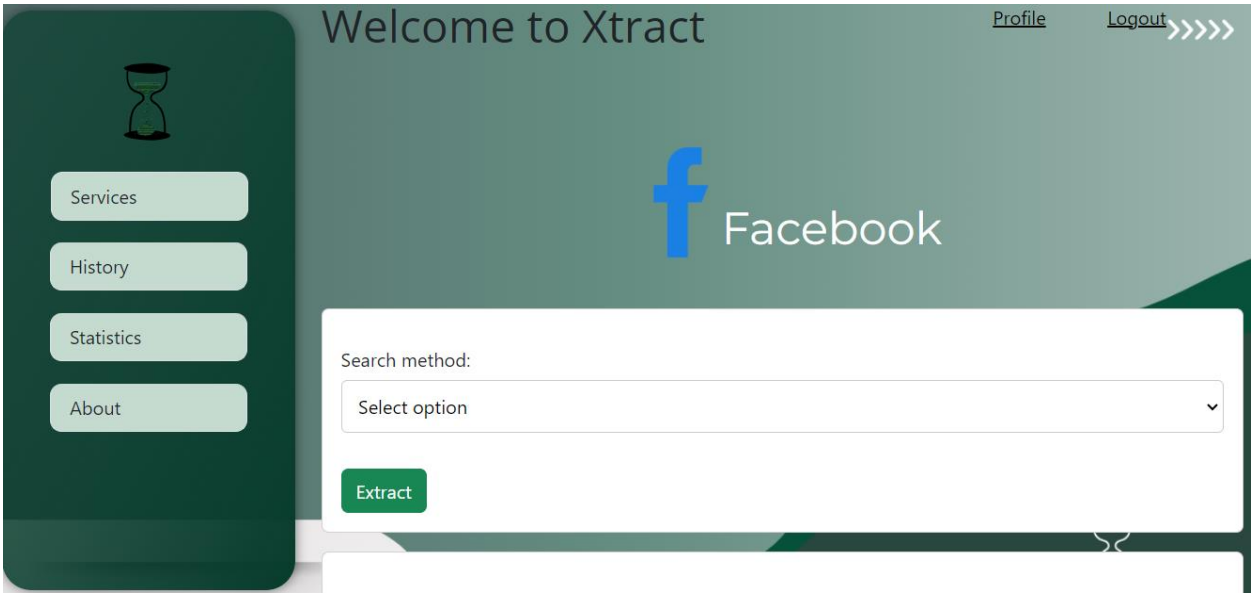


Figure 25 Facebook search GUI

Figure 26 Twitter search GUI

Chapter 5: Implementation and testing

5.1 Implementation

5.1.1 Frontend

We use Vue.js framework to build the frontend in our project, the main structure is the main App.vue file and the folders containing pages that build the project: Views, Components, Router, and assets as shown here in this file hierarchy:

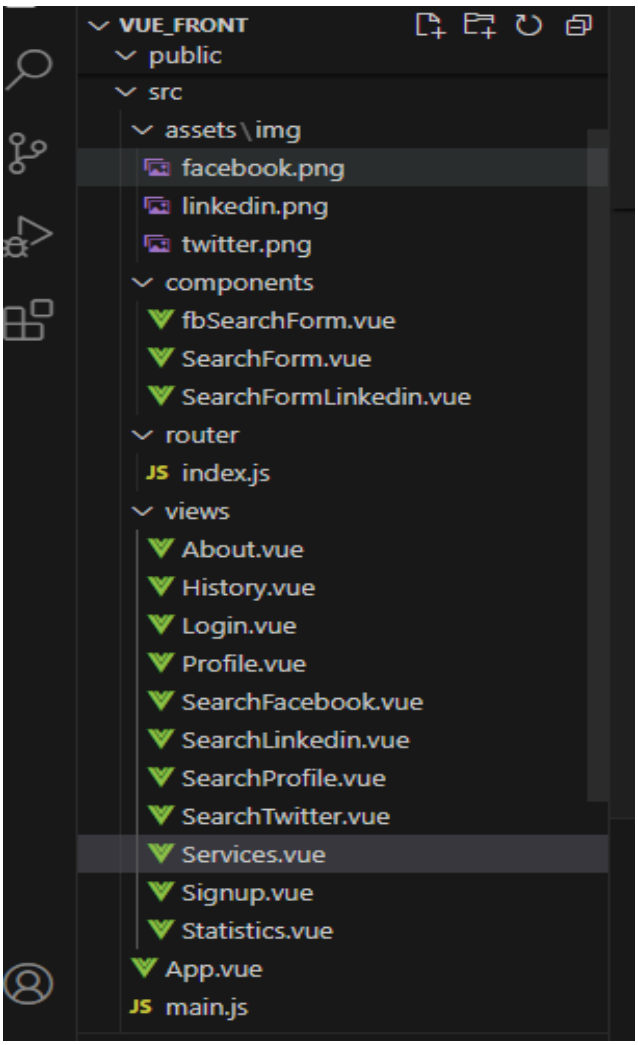


Figure 27 Frontend file hierarchy

5.1.2 Backend

Both Node.js and python are used to write the backend logic, we can say that there are total of three servers that contain the backend functionalities two of which are python servers running using the framework flask while the third is a Node.js server.
The node.js server basically hosts all the JavaScript code used for the service “Twitter search” as shown here in this code snippet:

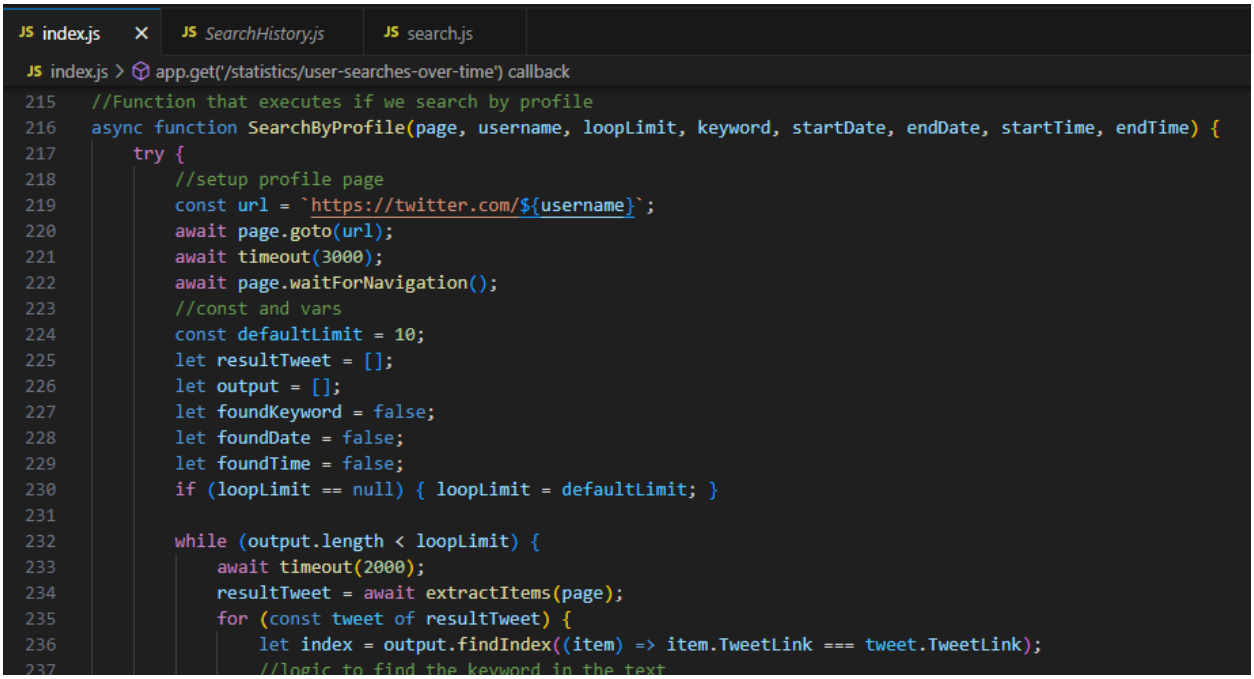


Figure 28 Node.js Backend ex.1

not only that but it also has the functions responsible for handling the Signup, Login, profile, History, and Statistics pages, as shown here:

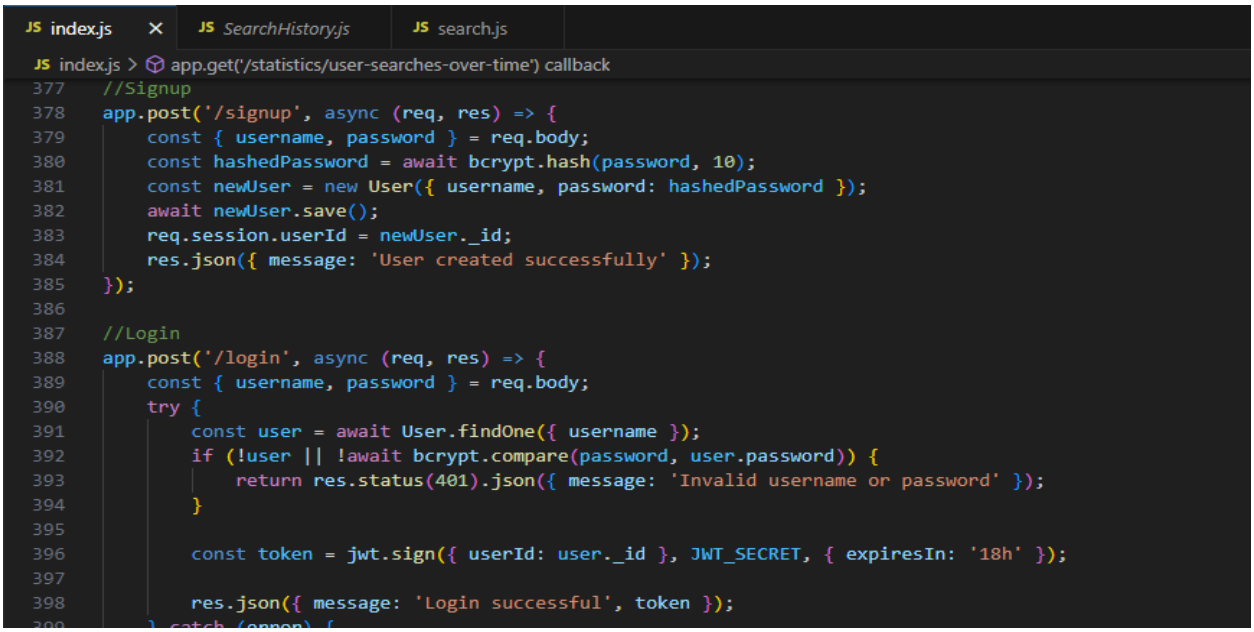


Figure 29 Node.js Backend ex.2

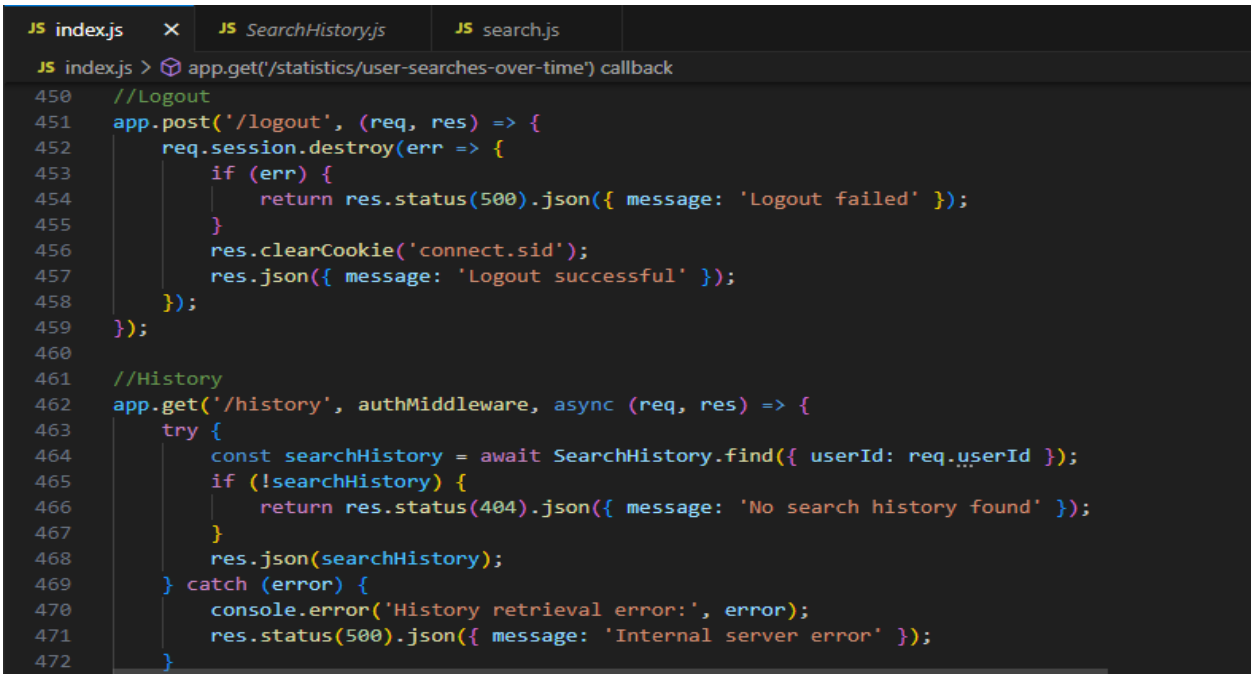


Figure 30 Node.js Backend ex.3

The first python server houses the logic for all the scraping needed in the “LinkedIn Service” part of the app and the code needed to accept requests from the front end and respond to them. The logic is divided across three files. One for scraping a user’s profile.

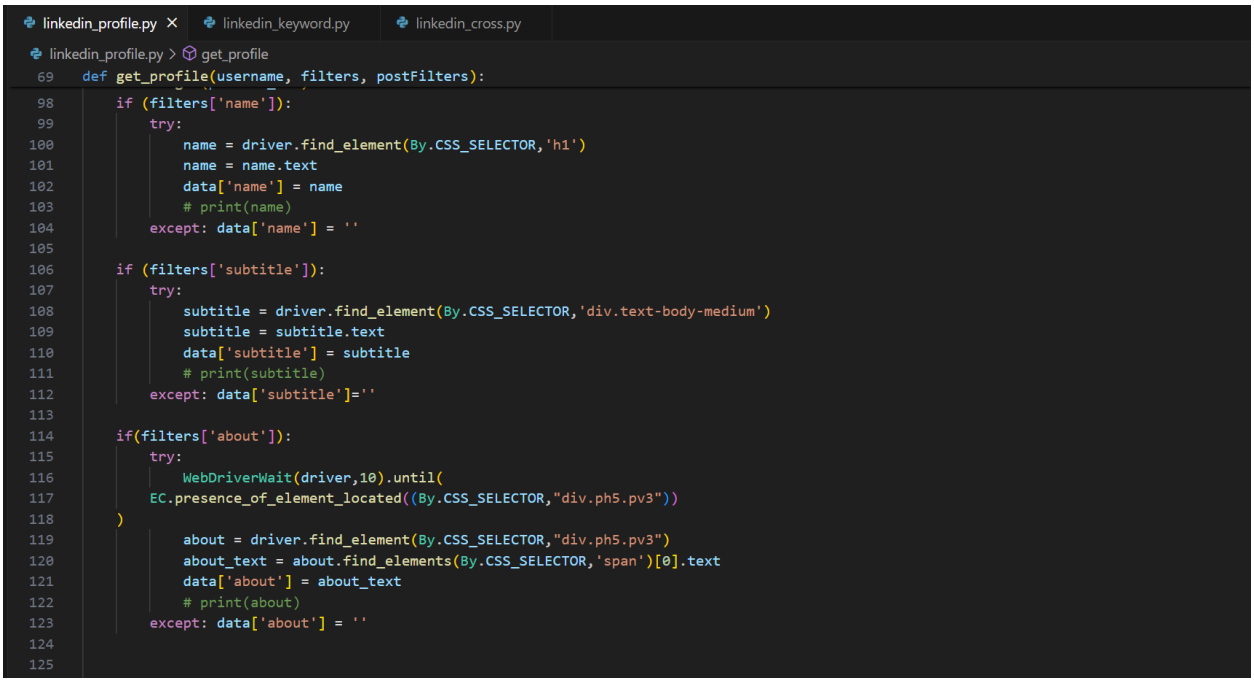
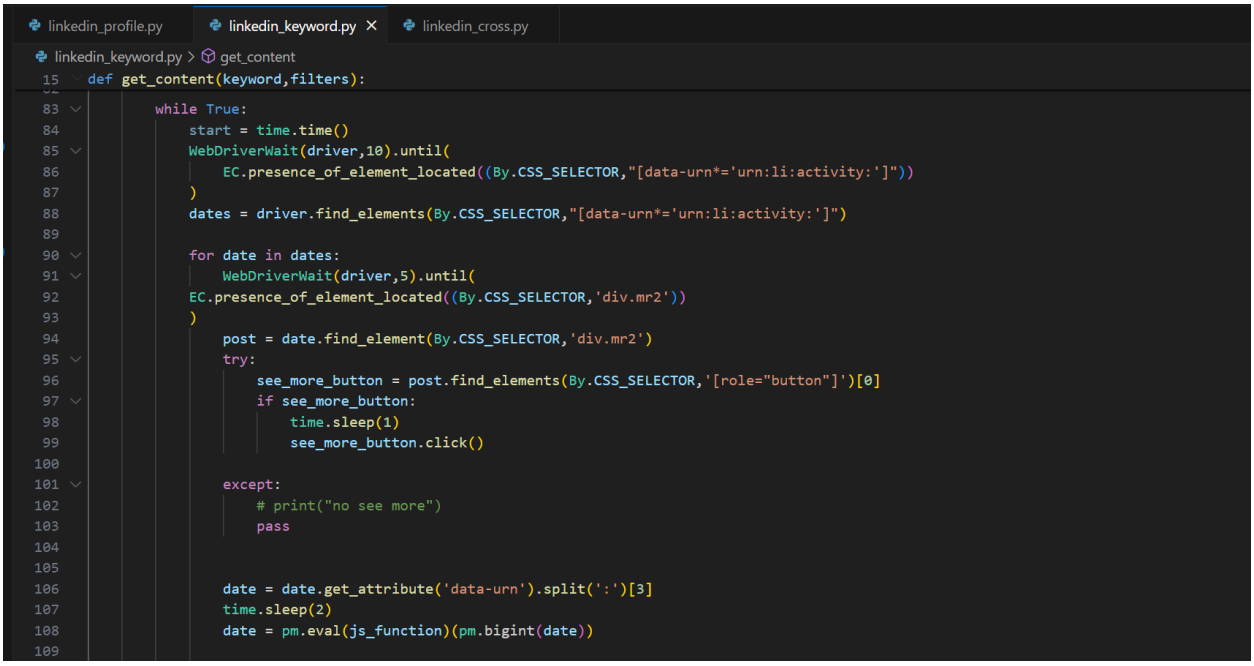


Figure 31 Python Backend ex.1

The second file contains the code for scraping posts using a specific keyword as well as filtering the results further by date and amount.



```
linkedin_profile.py   linkedin_keyword.py   linkedin_cross.py
linkedin_keyword.py > get_content
15  def get_content(keyword,filters):
83          while True:
84              start = time.time()
85              WebDriverWait(driver,10).until(
86                  EC.presence_of_element_located((By.CSS_SELECTOR,"[data-urn*='urn:li:activity:']"))
87              )
88              dates = driver.find_elements(By.CSS_SELECTOR,"[data-urn*='urn:li:activity:']")
89
90              for date in dates:
91                  WebDriverWait(driver,5).until(
92              EC.presence_of_element_located((By.CSS_SELECTOR,'div.mr2'))
93              )
94                  post = date.find_element(By.CSS_SELECTOR,'div.mr2')
95                  try:
96                      see_more_button = post.find_elements(By.CSS_SELECTOR,'[role="button"]')[0]
97                      if see_more_button:
98                          time.sleep(1)
99                          see_more_button.click()
100
101                 except:
102                     # print("no see more")
103                     pass
104
105
106                 date = date.get_attribute('data-urn').split(':')[3]
107                 time.sleep(2)
108                 date = pm.eval(js_function)(pm.bigint(date))
109
```

Figure 32 Python Backend ex.2

The third file is part of the “cross-platform search: functionality. It has a code that searches for profiles in LinkedIn.

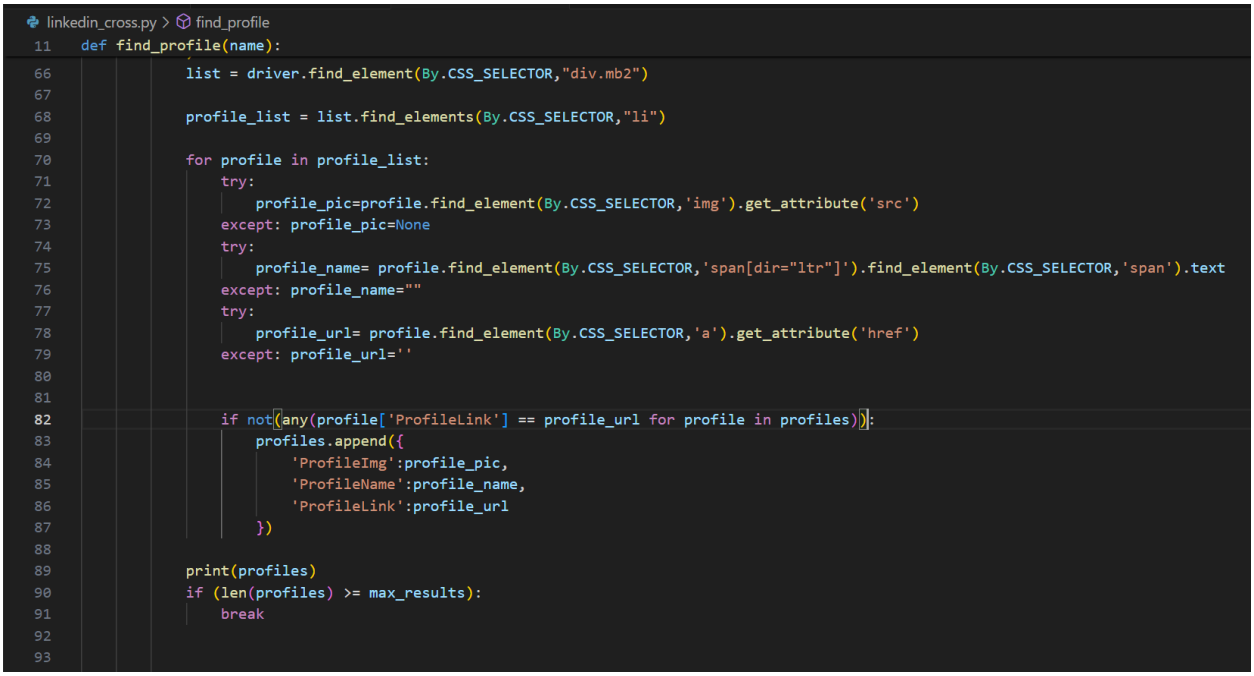


Figure 33 Python Backend ex.3

The “app.py” file has the routes that the front end sends requests to and the connection to the database.

app.py M

app.py > ...

```python
@app.route('/searchLinkedin',methods=['POST'])
@token_required
def search_linkedin(user_id):
    print(user_id)
    print(request.json)
    data = {}
    if request.method == 'POST':
        data['searchMethod'] = request.json['searchMethod']
        data['username'] = request.json['username']
        data['keyword'] = request.json['keyword']
        data['loopLimit'] = request.json['loopLimit']
        data['startDate'] = request.json['startDate']
        data['endDate'] = request.json['endDate']
        data['startTime'] = request.json['startTime']
        data['endTime'] = request.json['endTime']
        data['filters'] = request.json['filters']

        utc3 = pytz.timezone('Etc/GMT+3')
        now_utc3 = datetime.datetime.now(utc3)
        # Format the datetime
        formatted_date = now_utc3.strftime('%Y-%m-%d %H:%M')

        search_history = {
            "service": "Linkedin",
            "query": data["username"] or data['keyword'],
            "date": formatted_date,
            "userId": ObjectId(user_id)
        }
```

Figure 34 Python Backend ex.4

app.py M

app.py > ...

```python

        
@app.route('/download/<file_name>',methods=['GET'])
def download(file_name):
    format = request.args.get('fileFormat')
    if format =='csv':
        file_path = f"./download/{file_name}.csv"
        return send_file(file_path,download_name=file_name,as_attachment=True)
    elif format =='json':
        file_path = f"./download/{file_name}.json"
        return send_file(file_path,download_name=file_name,as_attachment=True)

    return jsonify('Not found 404')


@app.route('/findProfile',methods=['POST'])
@token_required
def search(user_id):
    name = request.json['profileName']

    profiles = find_profile(name)
    # print(profiles)
    return jsonify (profiles = profiles)


#
if __name__ == '__main__':
    app.run(port=5000,debug=True)
```

Figure 35 Python Backend ex.5

The server also has two files that retrieve and decode the user's token to verify his identity. The server doesn't accept any request with a token that fails the decoding.

middleware.py util.py

middleware.py > ...

```python
from functools import wraps
from flask import request, jsonify
from util import decode_auth_token

def token_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        auth_header = request.headers.get('Authorization')
        if not auth_header:
            return jsonify({'message': 'Token is missing!'}), 403
        try:
            token = auth_header.split(" ")[1]
            if token == "null":
                return jsonify({'message': 'Token is missing!'}), 403
            user_id = decode_auth_token(token)
        except Exception as e:
            return jsonify({'message': 'Token is invalid!'}), 403

        return f(user_id, *args, **kwargs)

    return decorated
```

Figure 36 Python Backend ex.6

middleware.py util.py

util.py > decode_auth_token

```python
import jwt
import datetime

SECRET_KEY = "your_jwt_secret"

def decode_auth_token(auth_token):
    try:
        payload = jwt.decode(auth_token, SECRET_KEY, algorithms=['HS256'])
        return payload['userId']
    except jwt.ExpiredSignatureError:
        return 'Signature expired. Please log in again.'
    except jwt.InvalidTokenError:
        return 'Invalid token. Please log in again.'

```

Figure 37Python Backend ex.7

The second python server houses the logic for all the scraping needed in the “Facebook Service” part of the app and the code needed to accept requests from the front end and respond to them. The logic is divided across six files. The first file for scraping user posts with filtering the result by number of posts and date.

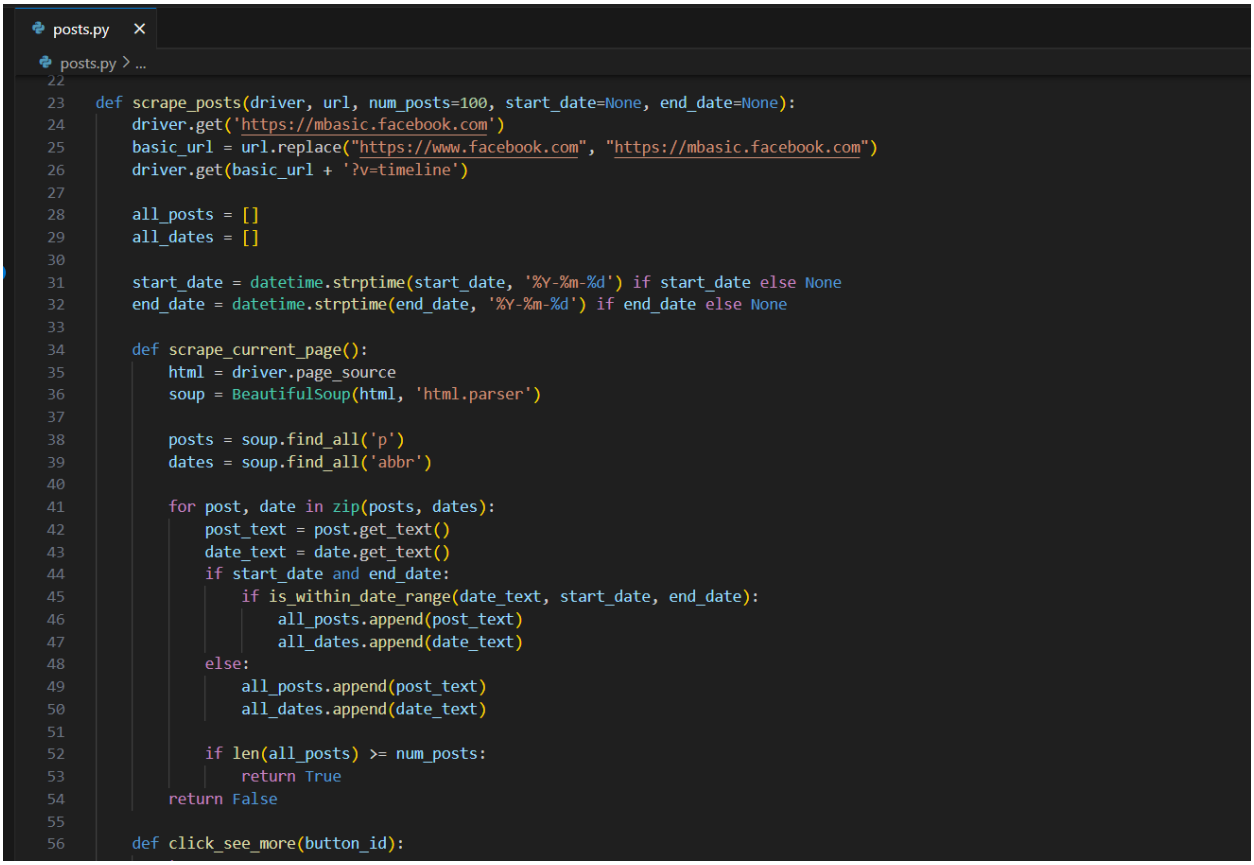
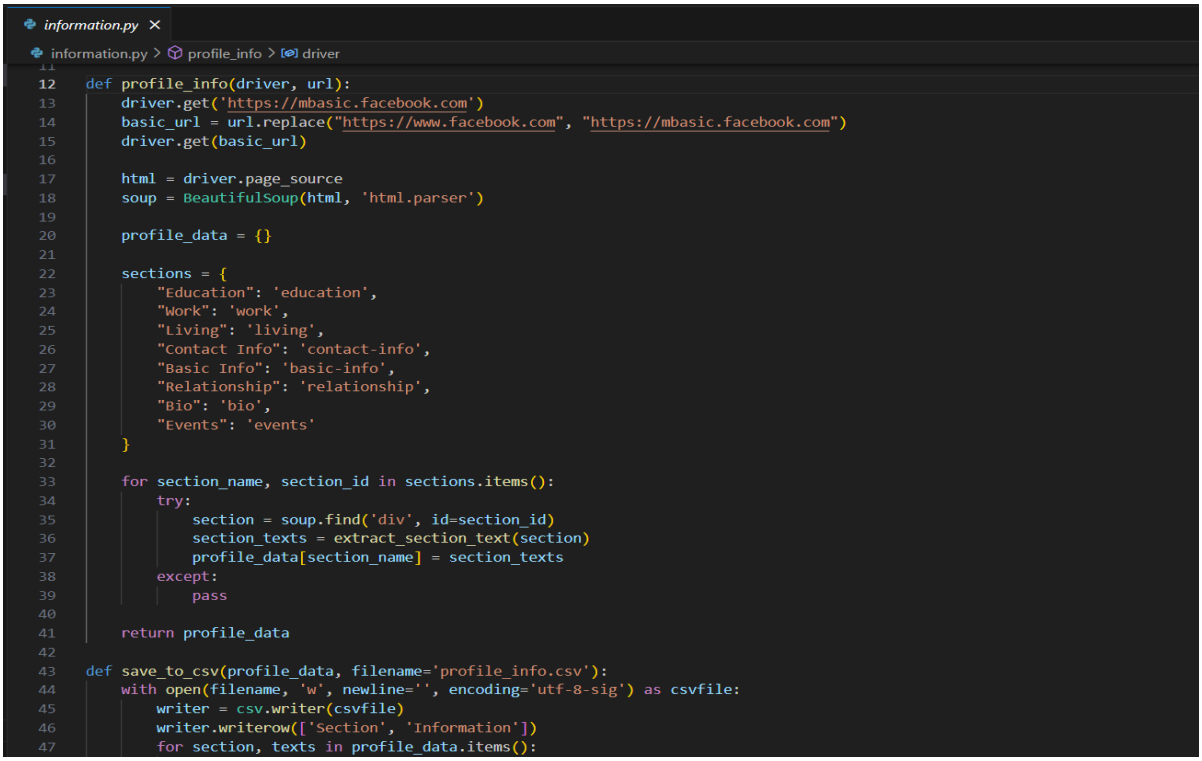


Figure 38 Python Backend ex.8

The second file contains the code for scraping user information such as education, work, living place, contact info, relationship ...etc



```python
def profile_info(driver, url):
    driver.get('https://mbasic.facebook.com')
    basic_url = url.replace("https://www.facebook.com", "https://mbasic.facebook.com")
    driver.get(basic_url)

    html = driver.page_source
    soup = BeautifulSoup(html, 'html.parser')

    profile_data = {}

    sections = {
        "Education": 'education',
        "Work": 'work',
        "Living": 'living',
        "Contact Info": 'contact-info',
        "Basic Info": 'basic-info',
        "Relationship": 'relationship',
        "Bio": 'bio',
        "Events": 'events'
    }

    for section_name, section_id in sections.items():
        try:
            section = soup.find('div', id=section_id)
            section_texts = extract_section_text(section)
            profile_data[section_name] = section_texts
        except:
            pass

    return profile_data

def save_to_csv(profile_data, filename='profile_info.csv'):
    with open(filename, 'w', newline='', encoding='utf-8-sig') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(['Section', 'Information'])
        for section, texts in profile_data.items():
```

Figure 39 Python Backend ex.9

The third file contains the code for scraping user photos.

```python
from selenium.webdriver.common.by import By
import time
import json
from login import login_facebook
import pandas as pd

# function to get user photo
def user_photos(driver, url):
    all_photos = []
    driver.get(url)
    for i in ["photos_of" , "photos_by"]:
        driver.get(url +"/"+ i)
        time.sleep(25)
        SCROLL_PAUSE_TIME = 5
        driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
        time.sleep(SCROLL_PAUSE_TIME)
        anchors = driver.find_elements(By.CSS_SELECTOR, "[class='x9f619 x1r8uery x1iyjqo2 x6ikm8r x10wlt62 x1n2onr6']")
        time.sleep(5)
        for img in anchors:
            try:
                user_images = img.find_element(By.TAG_NAME, "img")
                image_link = user_images.get_attribute('src')
            except:
                pass
            all_photos.append(image_link)

    for photo in all_photos:
        print(f"photo link: {photo}\n")

    return all_photos
```

Figure 40 Python Backend ex.10

The fourth file contains the code for scraping generic posts using a specific keyword.

searchKeyword.py

```
def scroll(driver):
        time.sleep(SCROLL_PAUSE_TIME)
        new_height = driver.execute_script("return document.body.scrollHeight")
        if new_height == page_height:
            break
        page_height = new_height
        counter = counter + 1


def searchText(driver, textToSearch):
    all_text = []
    all_link = []
    search_box = driver.find_element(By.XPATH, "//input[@type='search']")
    search_box.clear()
    search_box.send_keys(textToSearch)
    time.sleep(2)
    search_box.send_keys(Keys.ENTER)
    time.sleep(5)
    print(search_box)
    search_box.clear()

    driver.get(f"https://www.facebook.com/search/posts/?q={textToSearch}")
    scroll(driver)

    posts = driver.find_elements(By.CSS_SELECTOR, "[class='x1n2onr6 x1ja2u2z']")
    for post in posts:
        try:
            txt = post.find_element(By.CSS_SELECTOR, "[class='xdj266r x11i5rnm xat24cr x1mh8g0r x1vvkbs x126k92a']"
            txt_t = txt.text
            print(txt_t)
        except:
            txt_t = None
        all_text.append(txt_t)
    for link in posts:
        try:
            lnk = link.find_element(By.TAG_NAME, "a")
```

Figure 41 Python Backend ex.11

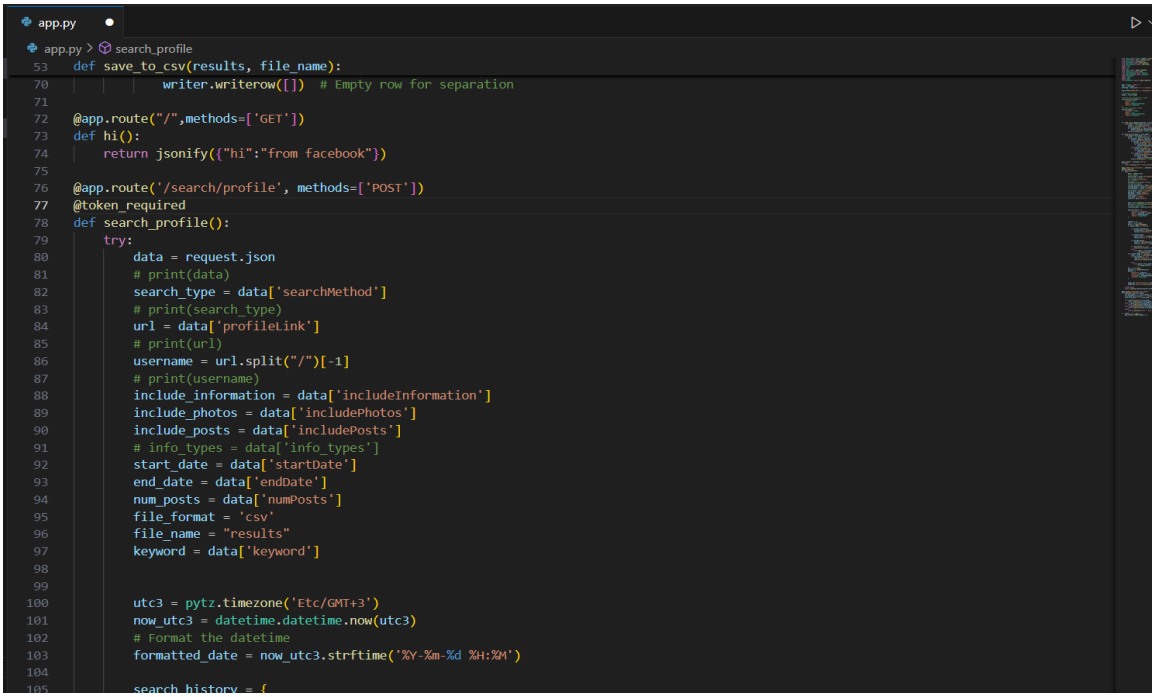
The fifth file is part of the “cross-platform search: functionality. It has a code that searches for profiles in Facebook.

search.py

```
def search_by_name(nameToSearch):
    #search by name
    search_box = driver.find_element(By.XPATH, "//input[@type='search']")
    search_box.clear()
    search_box.send_keys(nameToSearch)
    time.sleep(2)
    search_box.send_keys(Keys.ENTER)
    time.sleep(5)
    print(search_box)
    search_box.clear()


    #see all
    see_all = driver.find_element(By.CSS_SELECTOR, "[aria-label='See all']")
    if see_all:
        see_all.click()

    #scroll down with users name and password

    SCROLL_PAUSE_TIME = 2
    page_height = 0
    while True:
        driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
        time.sleep(SCROLL_PAUSE_TIME)
        new_height = driver.execute_script("return document.body.scrollHeight")
        if new_height == page_height:
            break
        page_height = new_height

    #user profile and image and description
    all_photos = []
    all_links = []

    searchResults = driver.find_elements(By.CSS_SELECTOR, "[class='x1yztbdb']")
    for img in searchResults:
        try:
            users_profilePhoto = img.find_element(By.TAG_NAME, "image")
```

Figure 42 Python Backend ex.12

The “app.py” file has the routes that the front end sends requests to and the connection to the database.



```python
app.py
app.py > search_profile
53  def save_to_csv(results, file_name):
70              writer.writerow([])  # Empty row for separation
71
72  @app.route("/",methods=['GET'])
73  def hi():
74      return jsonify({"hi":"from facebook"})
75
76  @app.route('/search/profile', methods=['POST'])
77  @token_required
78  def search_profile():
79      try:
80          data = request.json
81          # print(data)
82          search_type = data['searchMethod']
83          # print(search_type)
84          url = data['profileLink']
85          # print(url)
86          username = url.split("/")[-1]
87          # print(username)
88          include_information = data['includeInformation']
89          include_photos = data['includePhotos']
90          include_posts = data['includePosts']
91          # info_types = data['info_types']
92          start_date = data['startDate']
93          end_date = data['endDate']
94          num_posts = data['numPosts']
95          file_format = 'csv'
96          file_name = "results"
97          keyword = data['keyword']
98
99
100         utc3 = pytz.timezone('Etc/GMT+3')
101         now_utc3 = datetime.datetime.now(utc3)
102         # Format the datetime
103         formatted_date = now_utc3.strftime('%Y-%m-%d %H:%M')
104
105         search_history = {
```

Figure 43 Python Backend ex.13

The server also has two files that login to Facebook with dummy account to initiate the search.

```python
config.py
login.py
login.py > login_facebook
1   import json
2   import csv
3   from selenium import webdriver
4   from selenium.webdriver.common.keys import Keys
5   from selenium.webdriver.support import expected_conditions as EC
6   from selenium.webdriver.common.by import By
7   from selenium.webdriver.chrome.service import Service
8   from webdriver_manager.chrome import ChromeDriverManager
9   from selenium.webdriver.support.wait import WebDriverWait
10  from config import Config
11
12
13  def login_facebook():
14      chrome_options = webdriver.ChromeOptions()
15      prefs = {"profile.default_content_setting_values.notifications" : 2}
16      chrome_options.add_experimental_option("prefs", prefs)
17      chrome_options.add_argument("--disable-notifications")
18      driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()), options=chrome_options)
19
20      driver.get("http://www.facebook.com")
21      username = WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.CSS_SELECTOR, "input[name='email']")))
22      password = WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.CSS_SELECTOR, "input[name='pass']")))
23      username.clear()
24      username.send_keys(Config.FACEBOOK_EMAIL)
25      password.clear()
26      password.send_keys(Config.FACEBOOK_PASSWORD)
27      button = WebDriverWait(driver, 2).until(EC.element_to_be_clickable((By.CSS_SELECTOR, "button[type='submit']")))
28      button.click()
29
30      return driver
```

Figure 44 Python Backend ex.14

The server also has two files that retrieve and decode the user's token to verify his identity. The server doesn't accept any request with a token that fails the decoding.

middleware.py util.py

middleware.py > ...

```python
from functools import wraps
from flask import request, jsonify
from util import decode_auth_token

def token_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        auth_header = request.headers.get('Authorization')
        if not auth_header:
            return jsonify({'message': 'Token is missing!'}), 403
        try:
            token = auth_header.split(" ")[1]
            if token == "null":
                return jsonify({'message': 'Token is missing!'}), 403
            user_id = decode_auth_token(token)
        except Exception as e:
            return jsonify({'message': 'Token is invalid!'}), 403

        return f(user_id, *args, **kwargs)

    return decorated
```

Figure 45 Python Backend ex.15

middleware.py util.py

util.py > ...

```python
import jwt
import datetime

SECRET_KEY = "your_jwt_secret"

def decode_auth_token(auth_token):
    try:
        payload = jwt.decode(auth_token, SECRET_KEY, algorithms=['HS256'])
        return payload['userId']
    except jwt.ExpiredSignatureError:
        return 'Signature expired. Please log in again.'
    except jwt.InvalidTokenError:
        return 'Invalid token. Please log in again.'
```

Figure 46 Python Backend ex.16

5.1.3 Database

We use the cloud service that MongoDB provides to save our data in three main collections which are, users, search, and searchhistories as shown here:

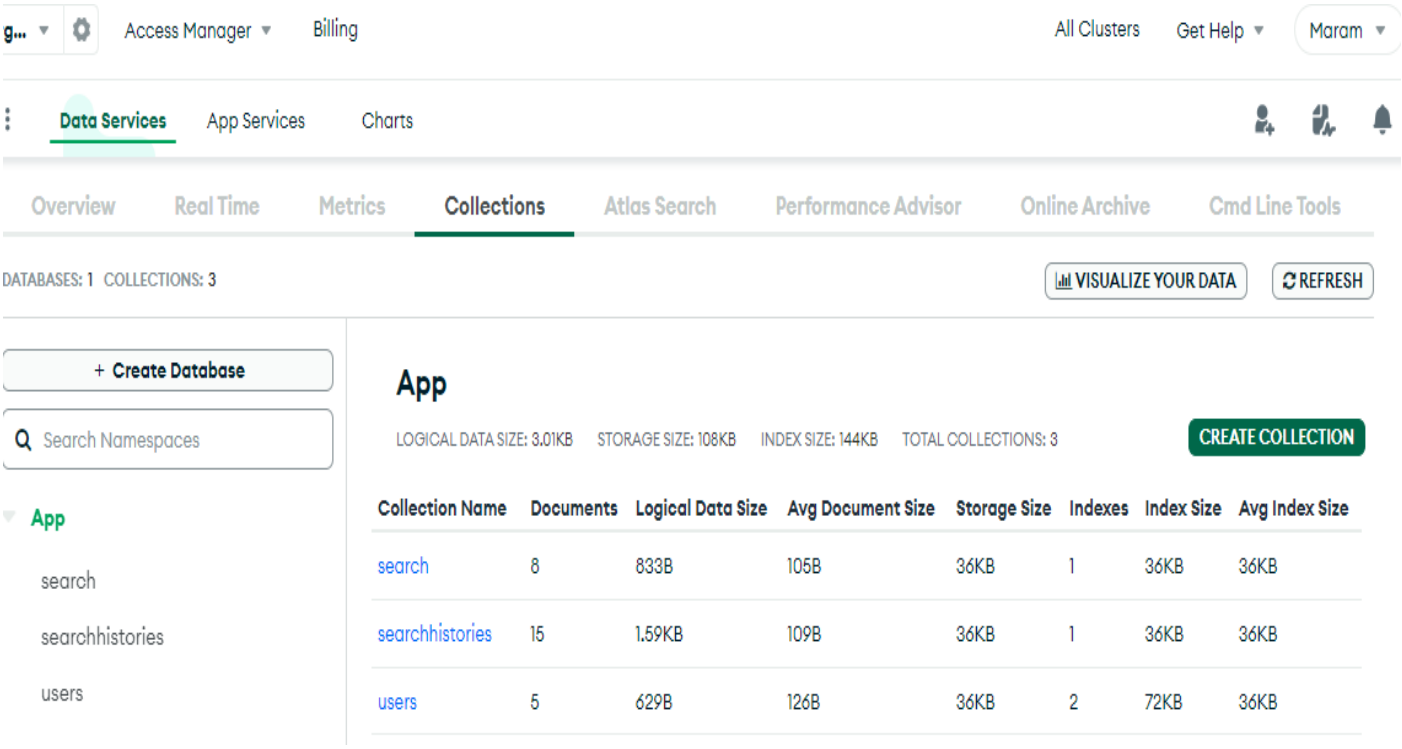


Figure 47 Cloud database collections

5.2 Testing

Functionality	Input	Expected Output
Sign up	Empty	A prompt tells the user to fill the username and password fields.
	Already used username with used or unused password.	A prompt tells user that sign up failed.
	Already used password with unused username.	Sign up is successful.
	New unused username and password.	Sign up successful and directed to Profile page.
Log in	Empty	A prompt tells the user to fill the username and password fields.
	Wrong username or password.	Log in failed alert.
	Valid username and password.	Log in successful and directed to profile page.
Twitter Search	No provided twitter username or keyword.	User cannot search unless both are provided.
	Enter loop limit of 0 or less or decimal.	A prompt tells the user to enter number that is equal to 1 or greater.
	Only Twitter username and keyword without providing data and time.	The search is done successfully.
Linkedin Search	A user tries to search without providing a username.	The app pop up “An error occurred during the search!”. The user needs to enter at least a username to carry on with the search.

	A user enters a username but doesn't select any data to be retrieved.	The app accepts the request and runs the code, but the results come back empty.
	The user asks the app to retrieve a section that is not there on the searched user account.	The app accepts request, and the search is done with an empty response.
	A user selects one or more sections for the search.	The search is done, and values of the attributes are returned if found.
	A user tries to search when his token is expired.	The app tells the user an error occurred and wait for the user to login again.
Facebook Search	A user tries to search without providing a URL.	The app pop up "An error occurred during the search!". The user needs to enter URL to continue
	A user enters a username but doesn't select any data to be retrieved.	The app accepts the request and runs the code, but the results come back empty.
	A user selects one or more sections for the search.	The search is done, and values of the attributes are returned if found.

Table 2 Testing

LinkedIn Test Cases:

1) A user tries to search without providing a username:

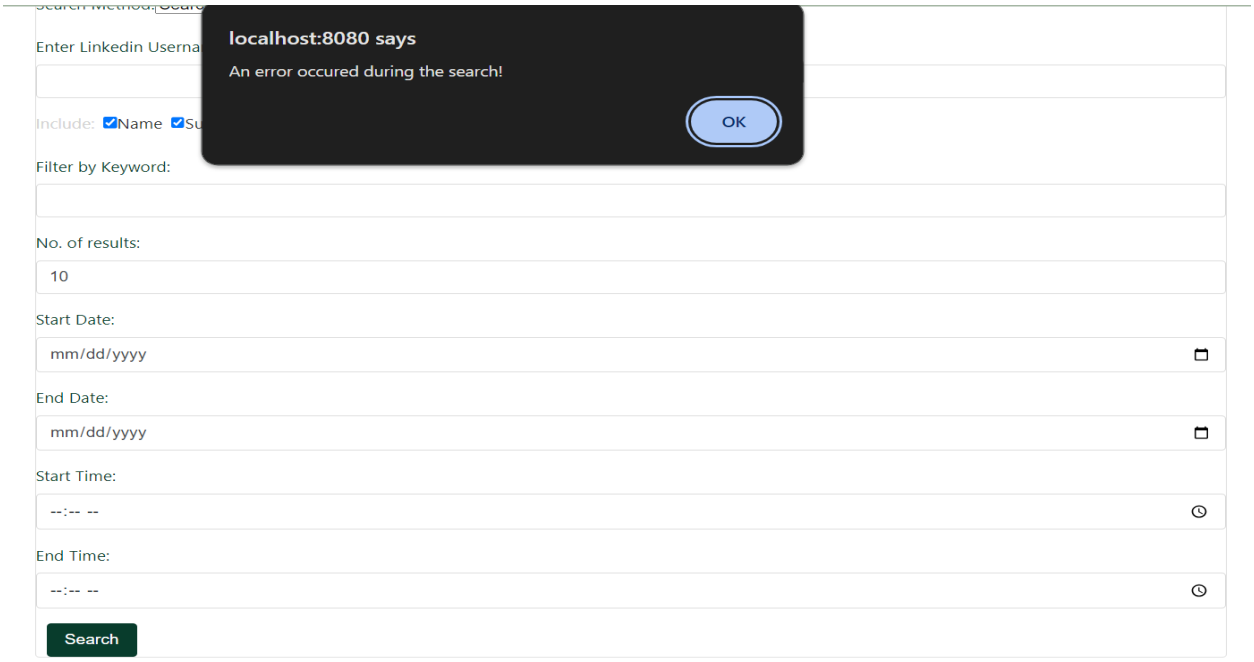


Figure 48 Test case 1

2) A user enters a username but doesn't select any data to be retrieved:

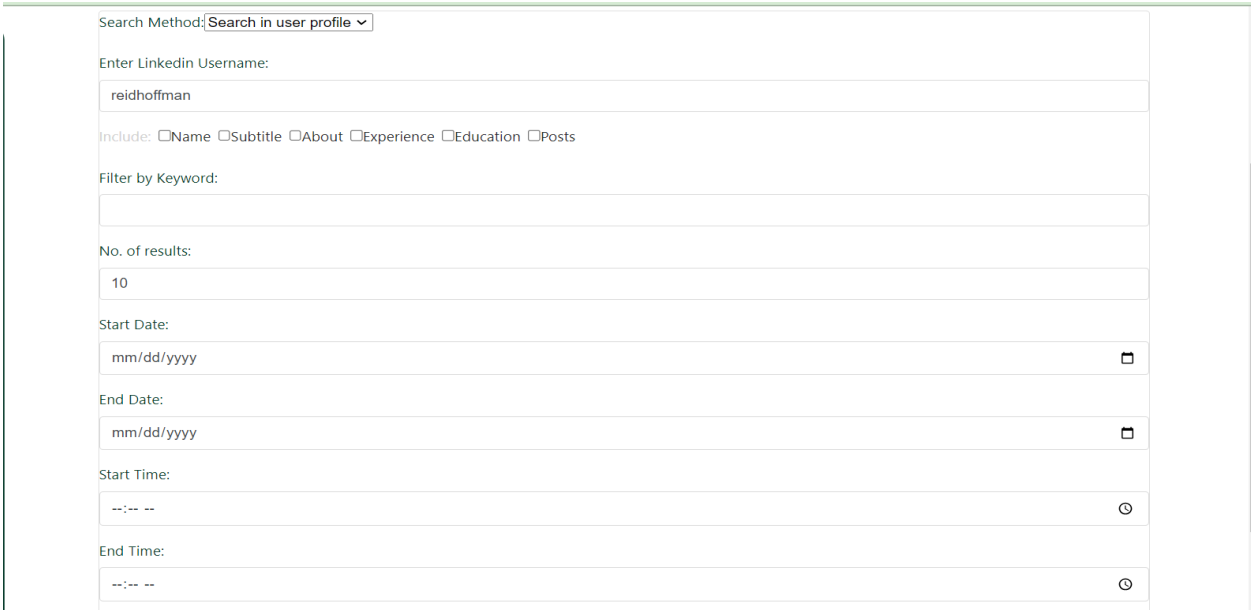


Figure 49 Test case 2.1

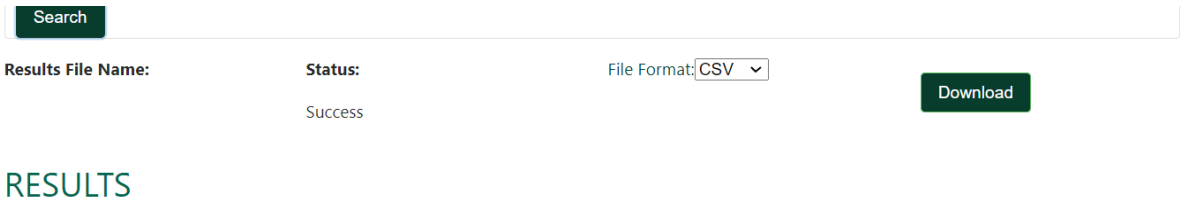


Figure 50 Test case 2.2

3) A user selects one or more sections for the search:

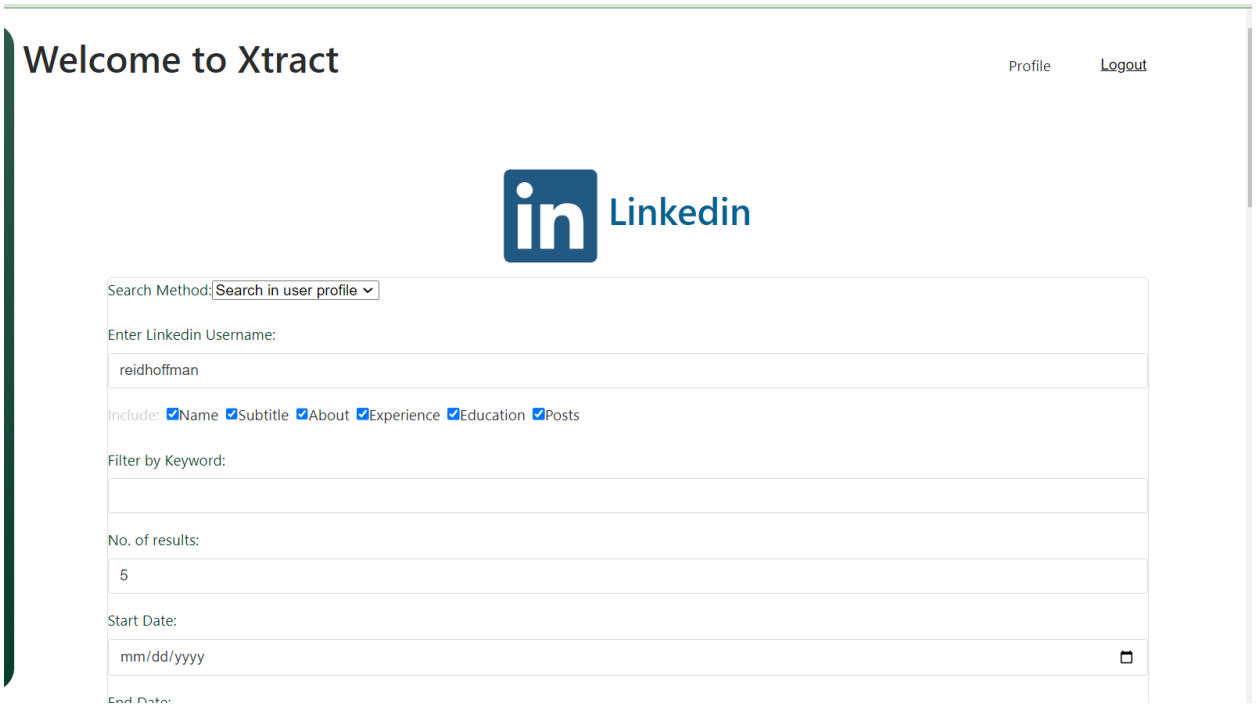


Figure 51 Test case 3.1

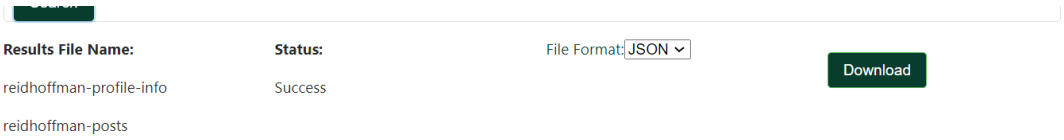


Figure 52 Test case 3.2

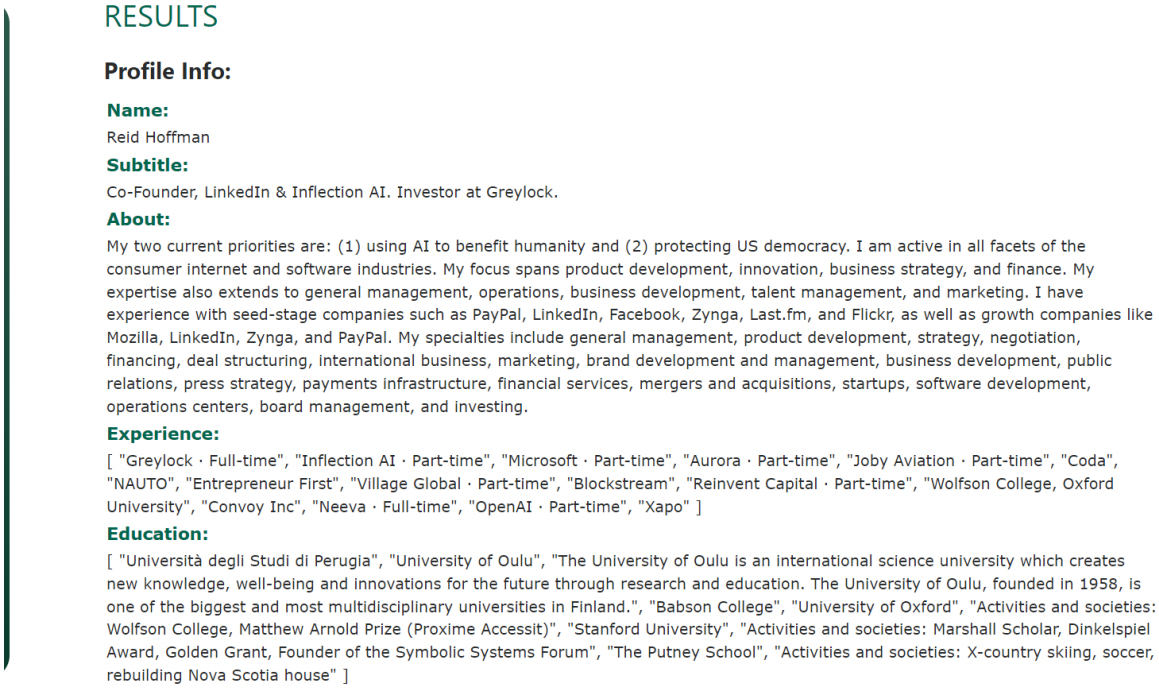
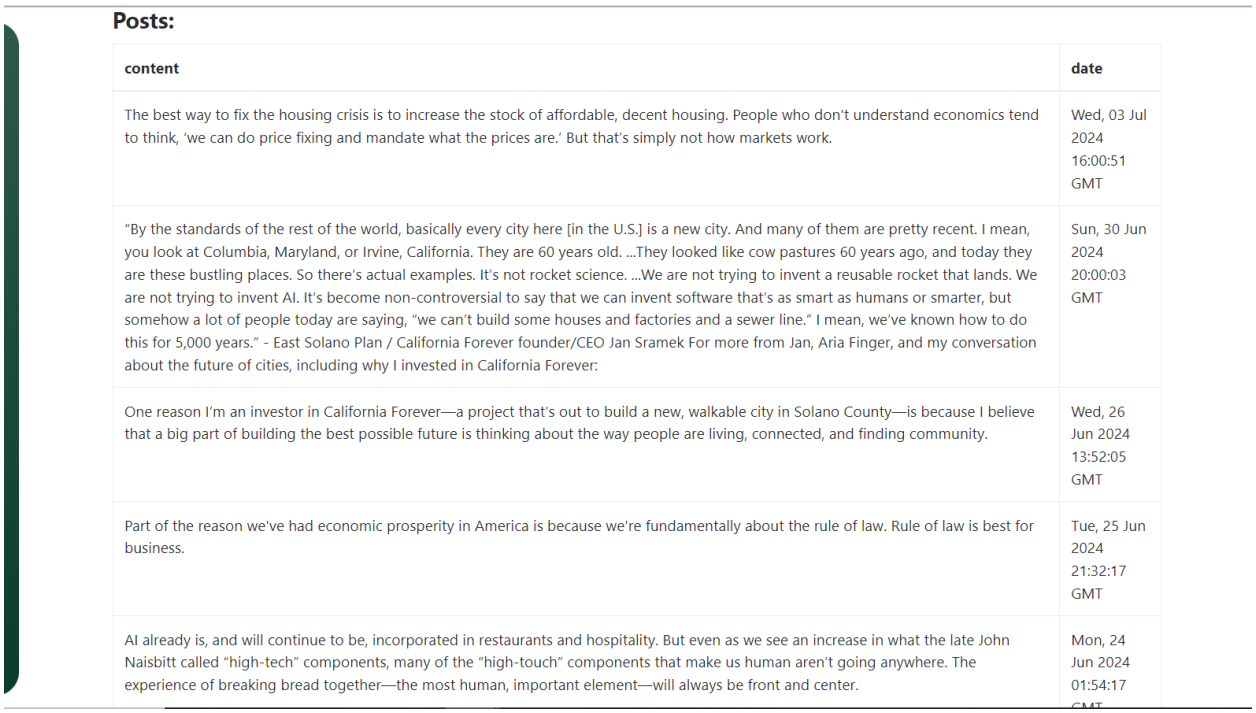


Figure 53 Test case 3.3



Posts:

content	date
The best way to fix the housing crisis is to increase the stock of affordable, decent housing. People who don't understand economics tend to think, 'we can do price fixing and mandate what the prices are.' But that's simply not how markets work.	Wed, 03 Jul 2024 16:00:51 GMT
"By the standards of the rest of the world, basically every city here [in the U.S.] is a new city. And many of them are pretty recent. I mean, you look at Columbia, Maryland, or Irvine, California. They are 60 years old. ...They looked like cow pastures 60 years ago, and today they are these bustling places. So there's actual examples. It's not rocket science. ...We are not trying to invent a reusable rocket that lands. We are not trying to invent AI. It's become non-controversial to say that we can invent software that's as smart as humans or smarter, but somehow a lot of people today are saying, "we can't build some houses and factories and a sewer line." I mean, we've known how to do this for 5,000 years." - East Solano Plan / California Forever founder/CEO Jan Sramek For more from Jan, Aria Finger, and my conversation about the future of cities, including why I invested in California Forever:	Sun, 30 Jun 2024 20:00:03 GMT
One reason I'm an investor in California Forever—a project that's out to build a new, walkable city in Solano County—is because I believe that a big part of building the best possible future is thinking about the way people are living, connected, and finding community.	Wed, 26 Jun 2024 13:52:05 GMT
Part of the reason we've had economic prosperity in America is because we're fundamentally about the rule of law. Rule of law is best for business.	Tue, 25 Jun 2024 21:32:17 GMT
AI already is, and will continue to be, incorporated in restaurants and hospitality. But even as we see an increase in what the late John Naisbitt called "high-tech" components, many of the "high-touch" components that make us human aren't going anywhere. The experience of breaking bread together—the most human, important element—will always be front and center.	Mon, 24 Jun 2024 01:54:17

Figure 54 Test case 3.4

4) A user tries to search when his token is expired:

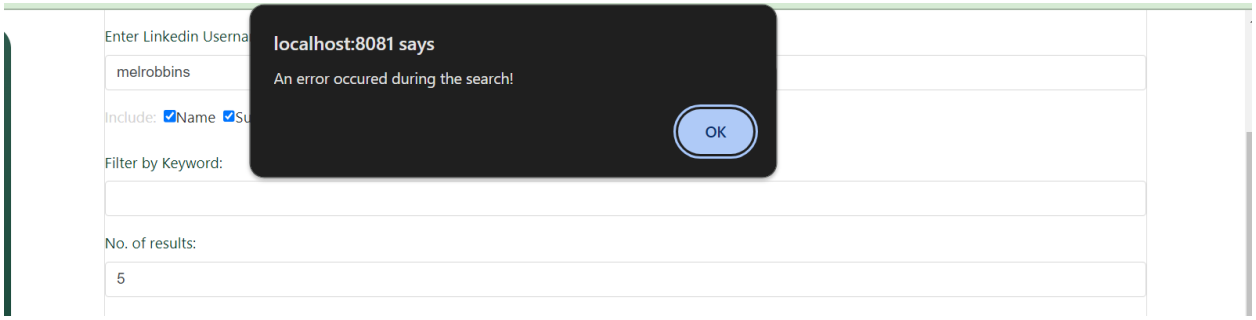


Figure 55 Test case 4.1

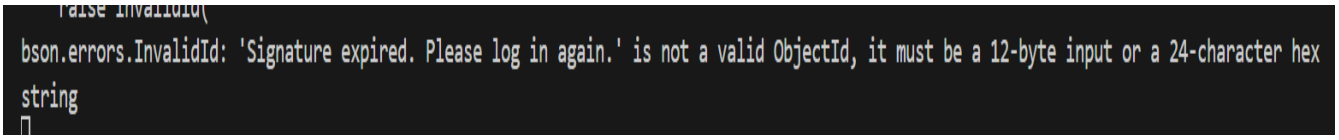


Figure 56 Test case 4.2

References

[1] “Social Media Data Service | Octoparse,” service.octoparse.com. https://service.octoparse.com/socialmedia

[2] J. T. Richelson, The U.S. Intelligence Community. Routledge, 2018.

[3] SuperSimpleDev, “HTML & CSS Full Course - Beginner to Pro,” www.youtube.com, Feb. 05, 2022. https://www.youtube.com/watch?v=G3e-cpL7ofc

[4] Stefan Hyltoft, “Web Scraping in Nodejs & JavaScript.” https://www.udemy.com/course/web-scraping-in-nodejs/

[5] Envato Tuts+, “Vue.js Tutorial: Beginner to Front-End Developer,” www.youtube.com, Jan. 25, 2023. https://www.youtube.com/watch?v=1GNsWa_EZdw

[6] “The Selenium Browser Automation Project,” *Selenium*. https://www.selenium.dev/documentation/

[7] Pandas, “pandas documentation — pandas 1.0.1 documentation,” pandas.pydata.org. https://pandas.pydata.org/docs/