uc3m | Universidad Carlos III de Madrid

University Degree in Computer Science

2022-2023

Bachelor Thesis

Design and Implementation of an Automated System for Human-Like Interaction

David Rico Menéndez

Angel Cuevas Rumin

Madrid, 1^{st} of January 2023

ABSTRACT

In recent years, automated systems have become an increasingly important tool in various fields, including online advertising, social media management, and customer support. However, many existing automated systems suffer from limitations such as inflexibility, lack of adaptability, and inability to mimic human behaviors effectively.

In this thesis, we present the design and implementation of a fully automated system for Facebook. The system is capable of mimicking human-like behavior, through granting our automated users an identity, access location and variable routine. The primary goal of this system is to create a high-quality automated system that can support other projects with their access tokens for advertising statistics from Facebook. The system has been thoroughly tested and shown to be effective in its ability to mimic human behavior on the Facebook platform. This thesis discusses the design and implementation of the automated system and provides insights into its capabilities and limitations.

Keywords: Automated system, Facebook, automated browsers, human behavior, social links, headless web interaction, headless web scraping, web automation system, Human-like automated system.

DEDICATORIA

A todas las personas que me han acompañado en el día a día de este proyecto, en especial a mis compañeros de trabajo y a mi tutor, Ángel Cuevas, por su paciencia y gran labor. Sin su apoyo y orientación, este trabajo no habría sido posible.

A mi padre y mi madre, quienes desde que era pequeño han hecho todo lo posible y lo imposible por darme la oportunidad de perseguir mis sueños y estudiar. Su sacrificio han sido mi mayor inspiración.

A mi hermana, por que pase lo que pase siempre sé que estará ahí. Tu me animas a tratar de ser cada día un mejor referente.

A Mercedes, por llegar a mi vida para quedarse y ser mi apoyo presente y futuro. Tu cariño ha sido mi refugio en los momentos de incertidumbre.

A Lucía, por ser amiga, consejera, compañera de alegrías y llantos. Tu fuerza me guía.

A Marina, por estar desde los inicios y redescubrirme cosas que ya creía aprendidas.

A Alejandro, por compartir incontables batallas, y terminar vencedores en todas.

A todos ellos, gracias por compartir este viaje.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Motivation and objective

In the field of research, the ability to access data in a scalable manner is of critical importance. Many systems and platforms impose restrictions and control measures [1] that limit the availability of information itself, making it challenging and sometimes almost impossible for researchers to gather comprehensive and extensive datasets. In some cases, the level of scalability offered by these systems falls short of the requirements for research purposes, and as the data gathered increases, this problem also does. To overcome this limitation, researchers often resort to using multiple user accounts to access the system, leading to increased complexity and the need for centralized orchestration systems.

Furthermore, to prevent account blocking, some systems employ mechanisms to verify whether the account's behavior is human-like. This presents an additional challenge for researchers, as they need to ensure that the accounts being used exhibit human-like behavior to avoid detection and potential account suspension. Therefore, in the realm of research, it is vital to develop methods or tools that can simulate human-like behavior to access data effectively and prevent unwanted restrictions.

The objective of this thesis is to create a scalable system for accessing public data available in the Facebook Ads Manager. The targeted data, known as aggregated data, does not pose any privacy risks to individual users as it does not contain personally identifiable information, being completely anonymous. However, these aggregated datasets have vital importance when addressing socio-economic issues, providing researchers with an unprecedented amount of data to analyze and derive insights from on their respective fields and researches.

By developing an automated system that can access the Facebook Ads Manager and retrieve public data in a scalable manner, this research aims to provide researchers with a powerful tool to dig into socio-economic phenomena and address research questions with access to sets of data previously unattainable. This system will enable researchers to harness the potential of large-scale data analysis, empowering them to discover patterns, trends, and correlations that can contribute to a deeper understanding of socio-economic dynamics.

Facebook is one of the most popular and influential social media platforms in the world, with more than 3.02 billion daily active users on average for March 2023[2]. Facebook also offers a powerful advertising platform that allows advertisers to target their audiences based on various criteria, such as demographics, interests, behaviors, and locations[3].

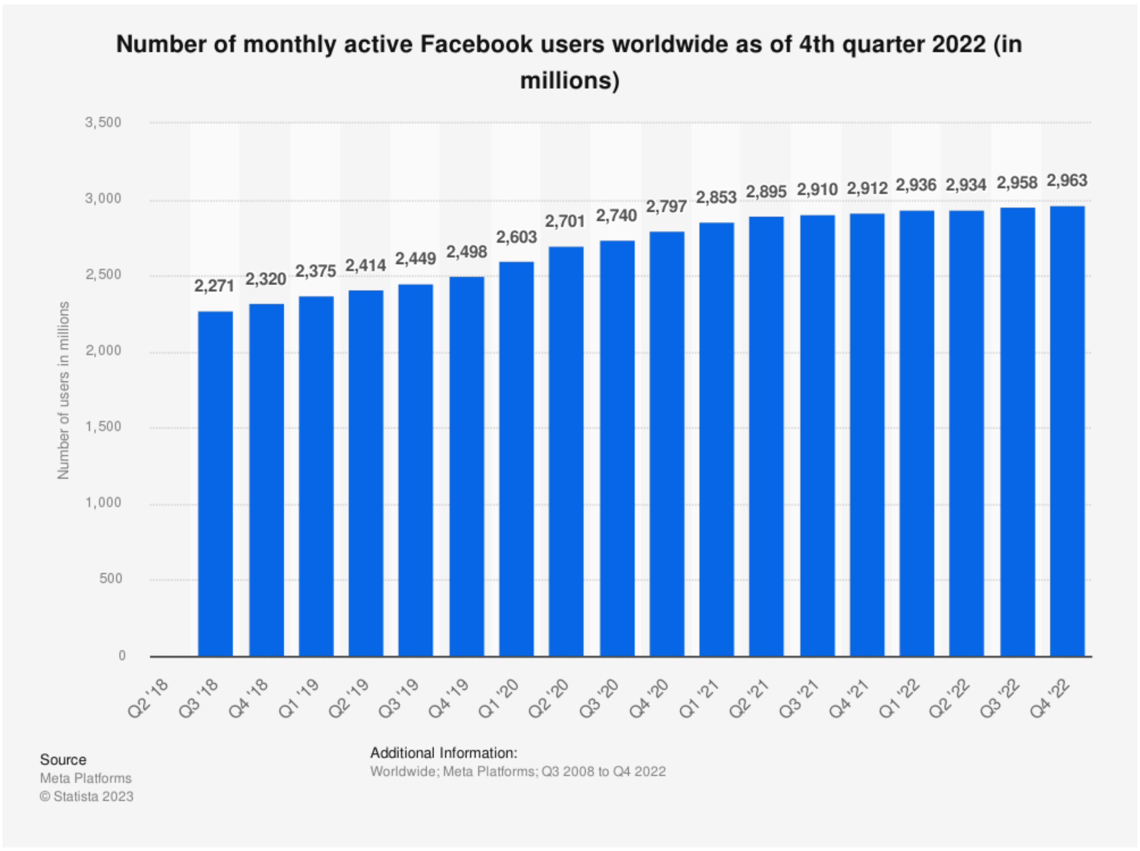


Fig. 1.1. Number of monthly active Facebook users worldwide as of 4th quarter 2022[4]

Facebook is a massive source of data that can provide insights into various aspects of human behavior and society. According to Meta's 2020 statistics, Facebook generates 4 petabytes of data per day[5], which is equivalent to 4 million gigabytes or 4 billion megabytes. Moreover, the average screen time on Facebook per user in the US was 55 minutes between 2015 and 2019, indicating a high level of engagement and activity. In addition to Facebook, Meta also owns other popular social media platforms, such as Snapchat and Instagram, which can also offer valuable data for research purposes.

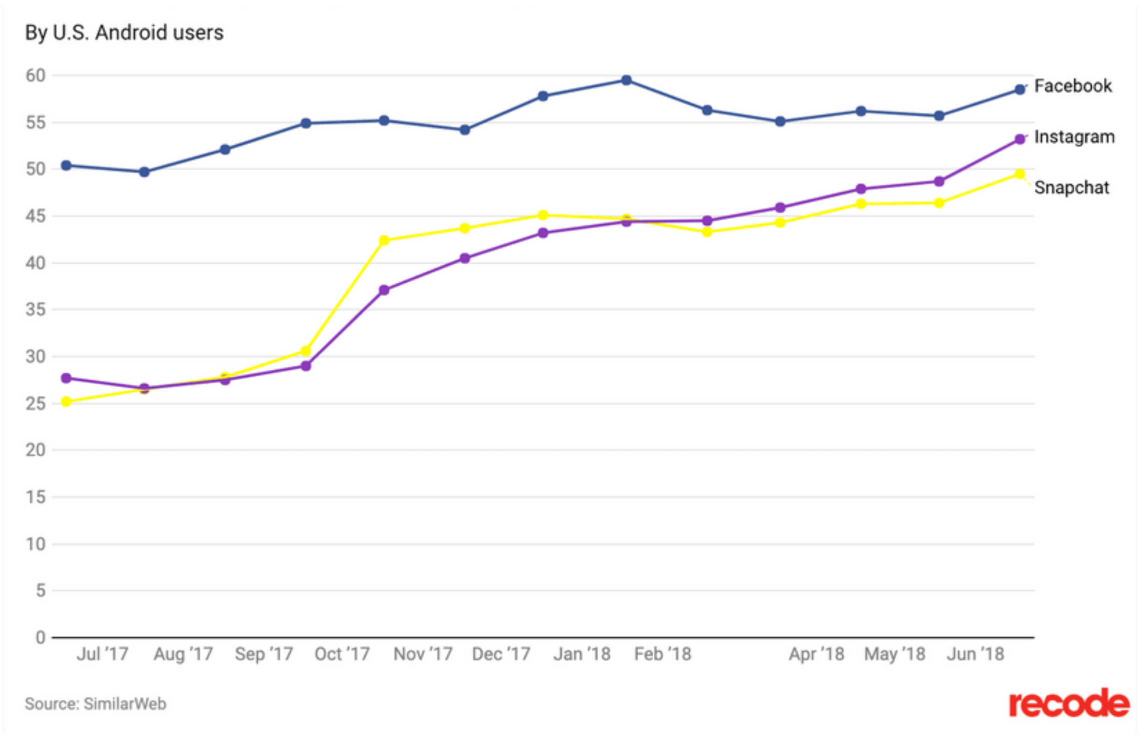


Fig. 1.2. Minutes spent per day on social apps

The system proposed will use Python as the programming language, Selenium as the web automation tool, and other libraries and frameworks to support the development and deployment of the automated system.

We chose Python as the programming language because it is simple, intuitive, and widely used for web automation tasks[6]. Python also supports various data structures, such as lists and dictionaries, that can store and manipulate data efficiently. Moreover, Python has a rich set of modules and packages that can facilitate web development, such as requests, BeautifulSoup, Flask, Django, etc.[7]

We chose Selenium as the web automation tool because it allows us to control a web browser programmatically and perform actions such as clicking, typing, scrolling, etc... Selenium also supports multiple browsers, such as Chrome or Firefox, and can run in headless mode, which means without a graphical user interface. Furthermore, Selenium can handle dynamic web elements that are generated by JavaScript or Ajax.[8]

The expected outcome of this thesis is to have a fully functional automated system that can create and operate multiple Facebook accounts that can access Facebook Ads Manager and obtain tokens for research purposes. We also aim to evaluate the performance and reliability of the system in terms of its success rate, speed, scalability, and robustness.

We hope to contribute to the field of web automation and social media research by providing a useful tool that can enable researchers to access the huge set of valuable data from Facebook advertising platform, independently of their economic capabilities.

1.2. Structure of the study

To provide a clear structure for this thesis, a summary of each chapter is presented below:

- Introduction: This chapter provides an overview of the project and aims to familiarize the reader with its objectives and significance.
- State of the Art: In this chapter, we examine the current state of web automated systems and web automation and discuss the various applications and significance of these technologies.
- Automated systems: This chapter serves as an introduction to web automated systems, providing an overview of the basic concepts and features needed to understand the topic.
- Development of the System and Functionalities: Here, we provide a detailed description of the system developed, including an in-depth analysis of the various functions and features, as well as the tools and technologies used to develop and feed the system.
  - Web driver: In this section, we will discuss the importance of a web driver in automated systems. We will explain the different types of web drivers available and why we chose a particular one for our system.
  - Main functionalities: This section will focus on the key features and functionalities of our automated system. We will describe in detail the different tasks that the automated system can perform, including logging in, navigating through the website, posting and liking other posts. We will also highlight the importance of each function and how they contribute to the overall performance of the system.
  - Human-like behavior of automated users: One of the critical aspects of our automated system is its ability to mimic human behavior. In this section, we will explain why it is essential for our automated users to act like humans, such as using random access times, accessing different locations through VPNs, and performing actions at different times of the day.
- Results and Tests: In this chapter, we present a discussion of the results and utility of the developed system, including a detailed analysis of the various tests conducted to evaluate the system's effectiveness.
- Future Work: This chapter briefly covers potential improvements to the present and future technologies that could be used to enhance the developed system's performance.
- Bibliography: This chapter contains a list of all sources used to conduct the study, including references to academic journals, books, and online sources.

2. STATE OF THE ART

2.1. Current state of automated systems

Automated systems have witnessed significant advancements in recent years, revolutionizing various industries and domains. In particular, they have played a critical role in transforming the field of investigations. With the rise of digital platforms and the vast amount of information available online, automated systems have become prevalent tools for efficient and streamlined interactions, especially in web interaction and scraping tasks.

In the realm of investigations, automated systems offer valuable capabilities for data collection, analysis, and monitoring. These systems provide automation techniques to navigate websites, extract relevant information, and gather data on a large scale. They enable investigators to efficiently process vast amounts of data, saving time and resources compared to manual methods.

One notable aspect is the emergence of automated web interaction systems is that they are designed to interact with web interfaces, perform tasks, and extract data without the need for complex AI models, and their higher complexity and computational costs. By leveraging automation frameworks and web scraping tools, investigators can navigate through web pages, submit queries, retrieve search results, and extract specific information from websites with a very fine grain filtering and personalization.

All this brings several benefits in the research field. Firstly, it enables investigators to access a vast amount of data from various online sources, allowing them to gather valuable evidence and insights at a very high rate, without requiring to designate huge teams to retrieve it. By automating repetitive tasks, investigators can focus on data analysis and interpretation, facilitating the discovery of patterns, connections, and anomalies.

Furthermore, these systems offer scalability and consistency in data collection. They can efficiently process numerous websites and extract data in a standardized manner, reducing the risk of human error and ensuring data integrity. Investigators can customize the automation workflows to target specific websites, define scraping rules, and extract relevant information with precision.

To conclude, automated systems, particularly in web interaction and scraping, have significantly transformed the field of investigations, and they are broadly used among numerous projects.

2.2. Overview of existing web automation tools and their limitations

Web automation tools have played a crucial role in creating online interactions and facilitating various tasks. Among the most widely used web automation tools, we can find Selenium, a powerful and versatile framework that allows for automated testing and interaction with web applications. Selenium provides a range of programming language bindings, making it accessible and adaptable for developers across different platforms.

In addition to Selenium, other popular web automation tools have emerged, each with its unique strengths and capabilities. Google's Puppeteer provides a high-level API for automating tasks in Chrome and Chromium browsers, including a headless mode for invisible browser automation. Microsoft's Playwright is another noteworthy tool that supports multiple browsers, such as Chromium, Firefox, and WebKit, offering features like screenshots, video recording, and handling file downloads. For web scraping, Beautiful Soup simplifies data extraction from HTML and XML documents, while Scrapy is a comprehensive framework for scalable web scraping.

For this project, we decided to use Selenium. In section 3.2 we discuss broadly the advantages and disadvantages of it, and how we planned to overcome them.

2.3. Related work in the field

In recent years, there has been a growing interest in the use of Facebook Ads Manager data for socio-economic research. Facebook Ads Manager data provides a wealth of information about the demographics, interests, and behaviors of Facebook users, which can be used to address a variety of research questions.

For example, Tiago Araújo used Facebook Ads Manager data to predict obesity and diabetes prevalence [9]. They found that Facebook users who were more likely to be obese or have diabetes were also more likely to be interested in certain topics, such as weight loss and health. This information could be used to develop targeted interventions to help people lose weight and manage their diabetes.

Mehdi Fatehkia used Facebook Ads Manager data to measure digital gender inequalities [10]. They found that men and women were not equally represented in all aspects of the digital world. For example, men were more likely to be interested in technology and gaming, while women were more likely to be interested in fashion and beauty. This information could be used to develop programs to help close the digital gender gap.

Other researchers have used Facebook Ads Manager data to map urban socioeconomic inequalities in developing countries [11], track mass migration [12], and measure the impact of migration on local economies [13]. João Vieira used in 2022 Facebook Ads Manager data to describe the socio-economic situation of Syrian refugees in Lebanon [14]. They found that Syrian refugees were more likely to be unemployed and living in poverty than the Lebanese population. This information could be used to develop programs to help Syrian refugees integrate into Lebanese society.

More recently, researchers have begun to use Facebook Ads Manager data to study the spread of misinformation and disinformation on social media. For example, a study by Allcott and Gentzkow found in 2017 that false news stories on Facebook were shared more often than true news stories [15]. This information could be used to develop strategies to combat the spread of misinformation and disinformation on social media.

David Garcia et al. provide a comprehensive overview of the use of Facebook Ads Manager data to study gender inequality [16]. They find that Facebook Ads Manager data can be used to identify gender inequalities in a variety of areas, including education, health, and economic opportunity. They also find that Facebook Ads Manager data can be used to track changes in gender inequality over time.

In a study conducted by Amir Mehrjoo et al., this same data was utilized to investigate real-life use cases focused on studying different population subsets in Spain [17]. Their objective was to identify potential cultural differences that could contribute to regional conflicts and subsequently lead to radical independence movements. The study achieved successful outcomes, proving the benefits of large-scale data collection capabilities to investigation.

In addition, studies have emphasized the significant advantages that arise from having access to a "sample of two billion humans" [18]. These studies highlight the immense potential of leveraging large-scale data, such as the one obtained through platforms like Facebook, to delve deeper into understanding human behavior and societal dynamics, such as drawing "distances" between countries in terms of similarities and differences of their population interests.

Overall, the research on the use of Facebook Ads Manager data for socio-economic research is still in its early stages. However, the studies that have been conducted so far suggest that this data can be a valuable resource for researchers who are interested in understanding the demographics, interests, and behaviors of Facebook users.

Furthermore, the development of an automated system for acquiring this data becomes crucial due to Facebook's increasing restrictions on access to such information. In recent years, there has been a growing limitation on accessing Facebook Ads Manager data, making the utilization of scalable techniques, like the ones proposed in this thesis, essential for ongoing research endeavors.

3. AUTOMATED SYSTEMS

3.1. A brief introduction to web spiders and automation

Web spiders and web automation are two related but distinct concepts that involve using software to interact with the internet. Web spiders, also known as web crawlers, web scrapers, or search engine automated systems, are programs that systematically visit and index web pages. They collect information about the content, structure, and metadata of the pages, and store it in a database or a file. Web spiders are essential for search engines, as they enable them to provide fast and relevant results to users' queries.[19]

Web automation, on the other hand, is the process of using software to perform tasks on the web that would otherwise require human intervention. Web automation can be used for various purposes, such as testing, data extraction, form filling, online booking, social media posting, etc. Web automation can be achieved by using tools such as Selenium, Scrapy, BeautifulSoup, etc., that allow users to write scripts or commands that instruct the software how to interact with web elements.[20][21]

Web spiders and web automation are both powerful and useful techniques for manipulating and accessing web data. However, they also pose some ethical and legal challenges, such as respecting the privacy and security of web users and owners, complying with the terms of service and robots.txt files of websites, and avoiding malicious or excessive crawling or automation that could harm the performance or functionality of the web1. Therefore, web spiders and web automation should be used responsibly and respectfully.[19]

3.2. Selenium and Web Interaction

Selenium is a popular and versatile open-source automated testing framework specifically designed for validating web applications across various browsers and platforms. It supports multiple programming languages, including Java, C, Python, and more, allowing developers to create Selenium Test Scripts with ease. With its extensive capabilities, Selenium empowers testers and developers to efficiently automate web application testing and ensure their functionality across different environments.[22]

To make an informed decision about selecting a web automation tool, it is crucial to evaluate the strengths and weaknesses of different options. In this regard, we have chosen Selenium over other competitors based on the following analysis:[23]

1. Advantages of Selenium:

    (a) Cross-platform compatibility: Selenium supports multiple operating systems

   (Windows, macOS, Linux) and web browsers (Chrome, Firefox, Safari, etc.), enabling broad compatibility for automation tasks. In our concrete case, it was crucial to operate with a tool that allowed our system to work wherever it is decided to be deployed.

   (b) Extensibility: Selenium offers a rich set of APIs and libraries, allowing developers to customize and extend its functionality according to specific requirements. It also has one of the biggest python libraries available, and once learned, allows huge flexibility.

   (c) Large community, support and Open Source: Selenium has a vast and active community of users and developers, providing a wealth of resources, tutorials, and forums for assistance. It is also an open source tool, matching our interests and values.

2. Limitations:

   (a) Complex setup and learning curve: Setting up Selenium and understanding its various components can be time-consuming, particularly for those new to automation or programming. We had almost no background on automation tools, so we had to invest a big amount of time to understand how to make our tool work.

   (b) Maintenance and updates: Web applications and browsers frequently suffer updates, which can lead to compatibility issues with Selenium scripts. This implies the need of constant updates and reviews, which was not a problem due to the fact that we wanted to work explicitly over the time to limit test its capabilities.

Overall, by leveraging the strengths of Selenium and overcoming its limitations through innovative approaches, our research project aims to contribute to the development of advanced automated systems that can revolutionize the way we interact with online platforms while maintaining a human-like presence.

4. DEVELOPMENT OF THE SYSTEM AND FUNCTIONALITIES

The automated system we developed consists of a set of scripts that are designed to run continuously while in idle mode, awaiting specific triggers to initiate a sequence of actions. One such trigger is the selection of a user profile from a list of pre-defined profiles. Once a profile has been selected, the appropriate VPN and Proxy configuration assigned to that profile is set up and loaded, and the automated system proceeds to log in to Facebook using the credentials associated with the profile.

At this point, a pre-defined action or set of actions is executed, and a corresponding log report is generated for record-keeping purposes. Once the action(s) have been completed, the automated system logs out of the account and returns to idle status, awaiting the next trigger.

An essential aspect of the system is ensuring that the automated system can successfully mimic human behavior and avoid being detected as an automated system by Facebook's security measures. To achieve this, we conducted experiments to determine the data that Facebook uses to validate that a user is human and the account owner.

Another critical design consideration was building the system to be network-like, allowing the automated system to establish social links with other users and expand those links to include additional users when appropriate. This network-like structure enables the system to simulate human behavior more effectively and enhances its ability to carry out various actions on Facebook.

In summary, the automated system designed in this project represents a complex solution that required the implementation of various techniques and technologies in order to achieve its goals. The system was designed to automate actions on Facebook and to simulate human behavior realistically, making it necessary to integrate tools such as VPNs, Proxies, and automated browsers. Additionally, it was necessary to perform extensive experimentation to ensure that the system was able to bypass Facebook's security measures and maintain a low profile. Despite the complexity of the system, it was possible to achieve the desired results, and the system represents a valuable contribution to the field of automated systems.

4.1. Headless Web Interaction vs Headless Web Scraping

A headless web interaction and a headless web scraper are related, but not the same thing.

A headless web interaction refers to a program that can interact with web pages without requiring a graphical user interface (GUI) or a web browser to be open. This means that it can automate web interactions, such as clicking buttons or filling out forms, without displaying the web page or any UI elements to the user.

On the other hand, a headless web scraper is a type of program that extracts data from websites automatically. It does this by simulating human interaction with the web page, such as clicking links, entering data into forms, and scrolling through pages, and then extracting the desired data from the page. Like a headless web interaction, a headless web scraper can operate without a GUI or web browser being open.

So, while a headless web interaction may involve web scraping as one of its functions, it is not exclusively focused on scraping data from websites.

The key advantage of using a headless web interactions is that it allows for more efficient and scalable web navigation, as it can be set up to run autonomously without requiring manual input or supervision. This can be particularly useful in cases where large or multiple sessions need to be executed or where the task would be too time-consuming or impractical to do manually.

However, there are also some potential disadvantages to using headless interactions. For example, it may not be able to accurately render certain website elements or data that are reliant on specific browser settings or plugins. This can lead to incomplete or bad executions. Additionally, the use of headless interactions may be prohibited or restricted by certain websites or online services, as it may be seen as a violation of their terms of service or user agreements.

Since our automated system will be an always-running script, working on the university servers where no GUI is used, it was critical to make is work headless.

4.2. Web driver: Chrome vs Mozilla

In this section, we face another design decision. Choosing a suitable selenium driver is an important design decision that affects the performance and functionality of a web automation system. The two most popular options are the Chrome-based driver and the Firefox driver.

The Chrome driver is based on the Chromium browser and provides fast and stable performance. It has a well-documented API and supports a wide range of browser features, including advanced user interactions, multiple tabs, and JavaScript execution. The Chrome driver is also widely used and supported by the Selenium community, which makes it easier to find help and resources.

On the other hand, the Firefox driver is based on the Firefox browser and provides good performance and compatibility with the latest web standards. It also supports advanced user interactions and JavaScript execution, and it has a well-documented API. However, it may not be as fast as the Chrome driver in some cases, and it may have compatibility issues with some websites.

On the early stages of the development, I used Firefox driver as the base of my system. This was due to my familiarity with it, since I worked with it previously when I developed a token extraction tool for large-scale data collection as part of my previous work [24].

However, soon I realized that some of the issues I was having, such as slow load times and crashes, were due to reported Firefox version 47 bugs.

As a result, a decision was made to switch to a Chrome-based driver. Chrome has gained popularity among developers due to its faster load times, better memory management, and stability compared to other drivers. Furthermore, it offers better support for web standards and has more advanced development tools for debugging and testing web applications. The transition to the Chrome driver proved to be a wise decision, as it significantly improved the performance and reliability of the automated system.

CODE 4.1. Base driver functions close and init

```
    def driver_close(driver):
        driver.quit()

    def driver_init():
        print("Initiating driver")
        c_options = webdriver.ChromeOptions()
        c_options.add_argument("--headless")
        prefs = {"profile.default_content_setting_values.
           notifications" : 2}
        c_options.add_experimental_option("prefs",prefs)
        driver = webdriver.Chrome('/home/davidryt/FBNetwork/assets/
           chromedriver', options=c_options)
        driver.get('http://google.com')
        return driver
```

In this code snippet, we are setting up the Chrome driver to run in headless mode. The headless option is set to True, which means that the browser will run in the background without a graphical interface, saving system resources and speeding up the execution of the program

Additionally, we are setting the profile.default_content_setting_values.notifications option to 2, which disables notifications on the web page. This option prevents any pop-ups or notifications from interrupting the automation process, which can cause errors and delays. This option was added in later stages of the development, after encountering execution errors caused by pop-ups.

Overall, these options improve the efficiency and reliability of the automation process by reducing unnecessary distractions and interruptions.

Now we have the driver created and configured, but as we mentioned before, proxies will be used in order to give our different users unique identities. Due to this, the driver must be modified and loaded differently depending on the user and proxy being used.

For further details on how the driver is modified, please refer to subsection 4.5.1.

4.3. The cookies' problem

Web cookies, also known as HTTP cookies or browser cookies, are small pieces of data that websites store on a user's computer while they are browsing. Cookies were initially designed to facilitate the exchange of information between a user's computer and a web server, allowing the server to remember certain information about the user's visit, such as login credentials, user preferences, or items in a shopping cart.

Facebook, like many other websites, uses cookies to personalize user experiences, maintain user sessions, and track user activities. When a user accesses Facebook, cookies are stored on their computer to manage authentication, remember preferences, and deliver targeted advertisements. In addition, Facebook employs third-party cookies from various advertising and analytics partners to track user activity across different websites, which helps in refining their advertising strategies and user targeting.

Since we are using Selenium to automate browser actions and interactions for our project, handling these cookies becomes essential.As we explained in subsection 3.2, Selenium automates browser actions by mimicking real user interactions, such as clicking buttons, filling out forms, and navigating between pages. However, when a cookies consent popup appears upon accessing Facebook, it can cause Selenium to crash if not handled appropriately.

CODE 4.2. Example of handling a cookies exception

```
    driver.get('https://www.facebook.com/')
    sleep(4)
    try:
        accept_cookies = driver.find_element_by_css_selector("[data-
            cookiebanner='accept_button']")
        accept_cookies.click()
        sleep(2)
    except:
        pass
```

In the code snippet shown, we see an example of the base method we are implementing in every access to a new page who might have cookies to accept by the user.

We use a try-except block to handle any exceptions that may occur when the cookie pop-up appears. The try block attempts to find the "Accept" button on the cookie pop-up using a CSS selector. We had to manually inspect the CSS used in order to be able to find where to click. It is also possible to find elements using XPATH, as we will see later on. If the button is found, the automated system clicks on it using the accept_cookies.click() function and waits for a few more seconds to ensure that the cookies are accepted.

If the "Accept" button is not found, we simply pass the exception and continue with the rest of the code. This ensures that the automated system does not crash or get stuck in an infinite loop when the cookie pop-up appears.

By implementing this method in every access to a new page that might have cookies to accept, we can ensure that our automated system can navigate Facebook and other websites smoothly without being interrupted by the cookie pop-up.

4.4. Creation of main functionalities

In the previous sections, we already settled the basics on driver selection and creation, as well as some keep in mind issues we might have like cookies handling. In this section, we will first show how we will structure our system, as well as an in-detail view of each of the main functions implemented.

In the preceding sections, we discussed the essential aspects of selecting and creating drivers, as well as certain considerations such as cookies handling that we must keep in mind when developing our system. In this section, we will expand on how we will structure our system and provide a detailed overview of each of the primary functions that we have implemented.

4.4.1. System Structure

The structure of our system has been designed to optimize its efficiency and ensure its scalability. The system is composed of several functional modules, each with its own set of specific functionalities, to provide an organized and modular approach to the automation process.

The main folder of our project is organized into several subdirectories, each serving a specific purpose in the functioning of our system. The first subdirectory is the "assets" directory, which stores all the files required by the Selenium library to operate, including the chromedriver executable and configuration files.

The "docs" directory contains the documentation files that provide detailed information on the usage of our system, including the system architecture, installation guide, and usage instructions. The documentation files serve as a comprehensive guide to understanding the system and its functionalities.

The "tests" directory is where we store the testing scripts used to debug and verify the functionality of our system. These scripts are essential for identifying and fixing errors that may occur during the execution of the system.

The "scripts" directory is subdivided into two subdirectories: "Final Release" and "Other". From those, it is important to highlight:

- "Scripts/Final Release": This subdirectory contains the core system files responsible for the execution of the system, including the main script and all its associated modules.
- "Script/Other/tools": This subdirectory contains various tools that we have developed to help us obtain resources, generate accounts, and retrieve data from the web. These tools are essential for the functioning of the system and will be reviewed in subsections 4.5.1 and 4.5.2.

4.4.2. System Workflow

In this section, we will take a look at the general system workflow, as well as go a little more in depth in some particular calls I consider crucial for the understatement of the overall functioning.

In Figure 4.1 depicts the main loop of the system, illustrating the functional design for a complete rotation of users. The workflow begins when the system is initiated or awakened, and it proceeds by retrieving all available user accounts. From this pool of users, the system randomly selects one and proceeds to assign proxies for that user. The selected user is then logged into the Facebook platform, initiating the interaction phase. The system then calls the action selector, which determines the specific actions to be performed by the user account. During this process, the system also updates relevant lists and waits for a random period of time before transitioning to the next randomly selected user. This workflow ensures a dynamic and varied pattern of user interactions, contributing to the appearance of natural and human-like behavior within the system.

In Figure 4.2, a more detailed workflow is presented, specifically focusing on the login process and action selection. This figure highlights the steps involved in logging into the Facebook platform, including the identification of the Facebook version. We can also observe in the action selection, the setup and utilization of the action limit control variable within the action selection process, so automated users are allowed to perform a random number of action. These details are crucial in ensuring that the system adheres to Facebook's guidelines and mimics human behavior while engaging with the platform. By incorporating these specific steps and control mechanisms, the system maintains a balance between efficient automation and the appearance of organic user activity.

Together, these system workflows provide a comprehensive visualization of the sequential processes and interactions within the automated system. These workflows not only illustrate the intricate steps involved in user rotation, login, and action selection but also emphasize the system's adherence to Facebook's guidelines, enabling the generation of human-like behavior and minimizing the risk of account suspensions or blocks.

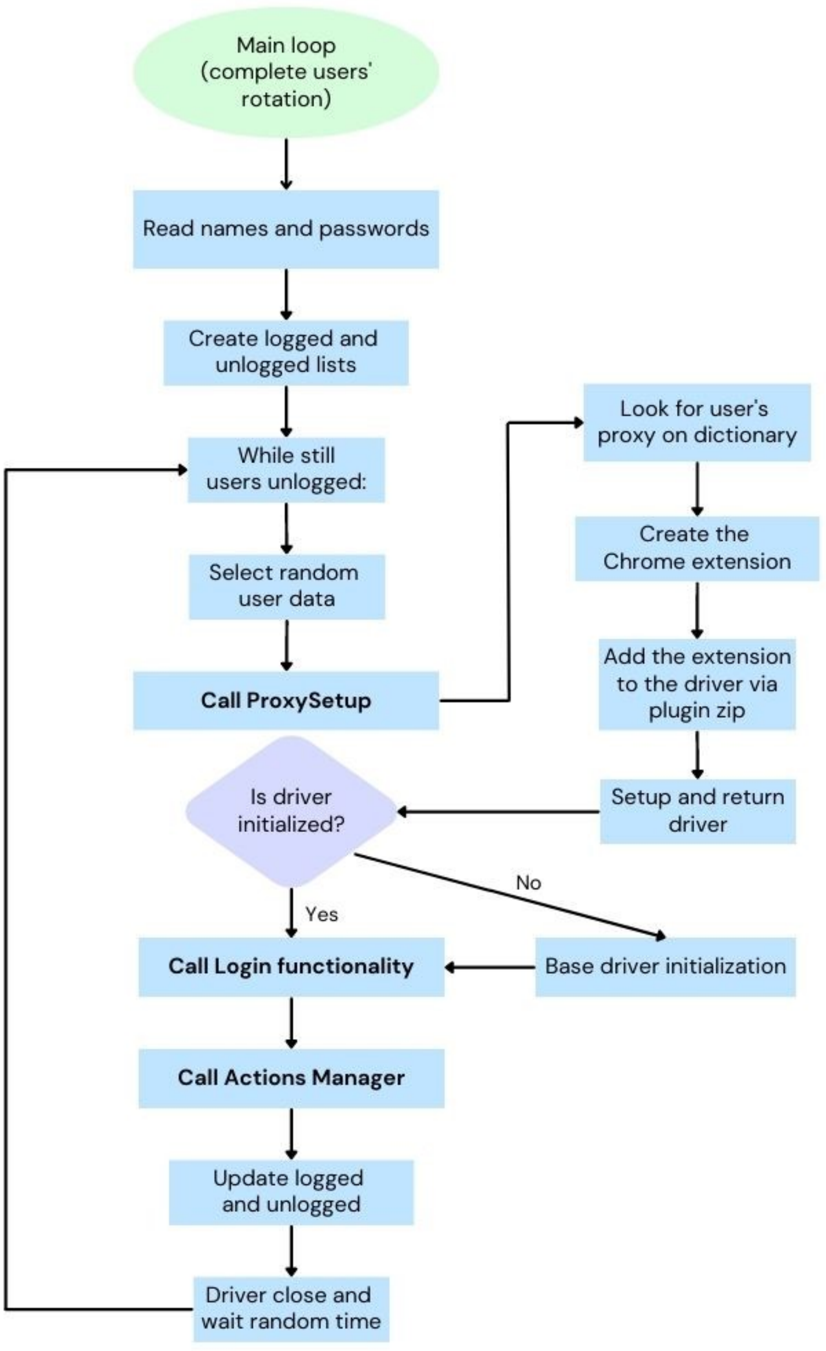

Main loop (complete users' rotation)
Read names and passwords
Create logged and unlogged lists
While still users unlogged:
Select random user data
Call ProxySetup
Look for user's proxy on dictionary
Create the Chrome extension
Add the extension to the driver via plugin zip
Setup and return driver
Is driver initialized?
No
Yes
Call Login functionality
Base driver initialization
Call Actions Manager
Update logged and unlogged
Driver close and wait random time

Fig. 4.1. Flowchart for main loop iteration

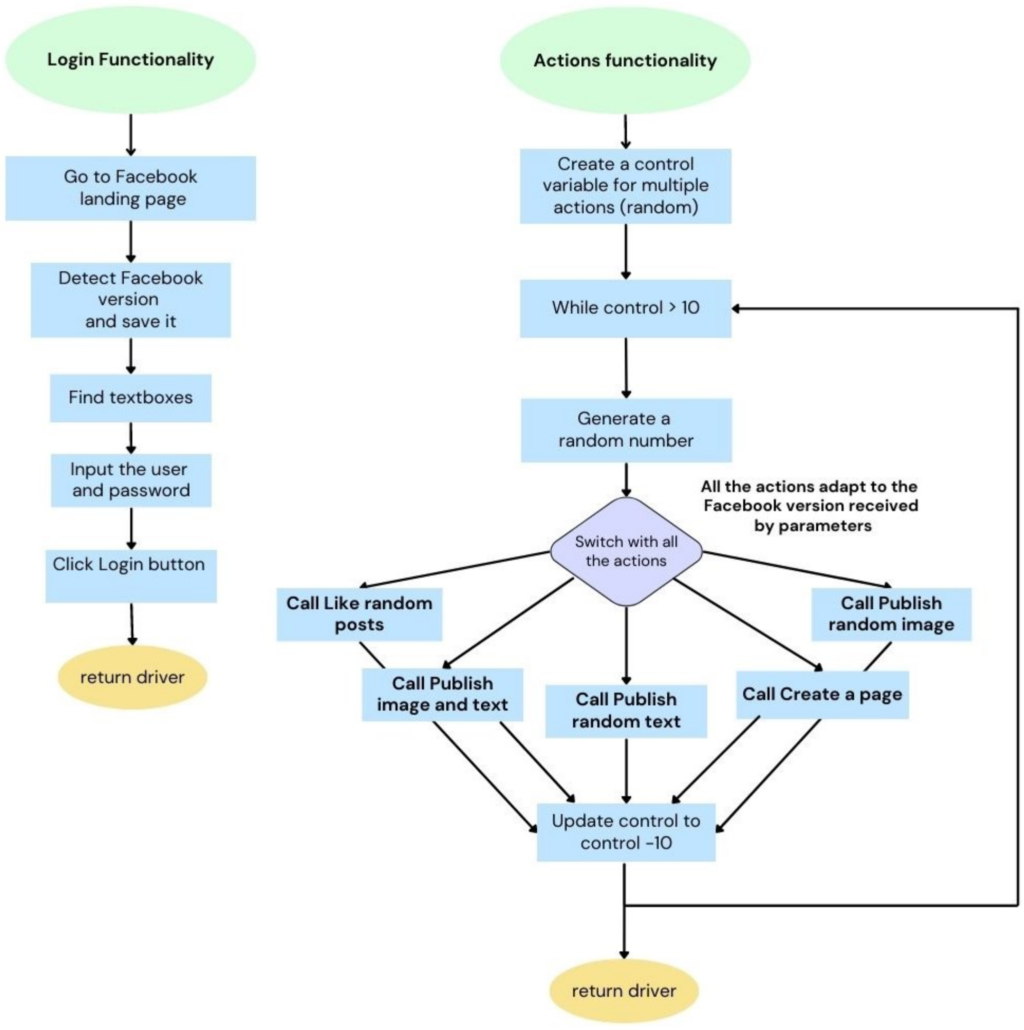

Login Functionality
Go to Facebook landing page
Detect Facebook version and save it
Find textboxes
Input the user and password
Click Login button
return driver
Actions functionality
Create a control variable for multiple actions (random)
While control > 10
Generate a random number
All the actions adapt to the Facebook version received by parameters
Switch with all the actions
Call Like random posts
Call Publish image and text
Call Publish random text
Call Create a page
Call Publish random image
Update control to control -10
return driver

Fig. 4.2. Flowchart for both calls, login and action selection

4.4.3. Users creation and logging in

When building an automated system for Facebook, we must consider a number of different components, including the ability to create and log in users. In this section, we will discuss the process of creating users, and logging in those users, which is a critical part of any automated system.

Creating users for an automated system involves a number of steps, from setting up user accounts to verifying email addresses and creating passwords. Once users have been created, they will need to be logged in to the system in order to use its various features. This process requires a secure and reliable login mechanism to ensure that only authorized users can access the system.

4.4.3.1 Creating the mail accounts

When we began building our Facebook automated system, we knew that we would need to create a set of users in order to test its functionality. Our initial plan was to automate the user creation process, but we quickly realized that this would be more difficult than we had anticipated. Our first attempt involved using Gmail to create a set of email addresses, but we ran into a roadblock when we discovered that new accounts required phone number verification.

After some research, we discovered ProtonMail, which is a secure email service that does not require phone number verification for account creation. ProtonMail is a popular choice for users who prioritize privacy and security, as it offers end-to-end encryption and other security features.

Then, I proceed to create the user creation script that automates the process of creating ProtonMail accounts for a list of usernames. The script takes a file with a list of usernames, and then generates a new ProtonMail account with the username and a suffix ("fbstats") added to it. I also use a preselected password for each account. Once the username and password are set, the script submits all the data and waits for confirmation before moving on to the next user in the list. By automating this process, I was able to quickly and easily create a large number of test users for my Facebook automated system.

CODE 4.3. Snippet from Proton Mail account creation tool

```
    with open(filedir) as f:
           usr = f.readline()
           while usr:
                usrnm= usr+"fbstats"
                passwd= "*******"
                driver = webdriver.Firefox()
                driver.get("https://mail.protonmail.com/create/new?language=en")
                driver.switch_to.frame(1)
                driver.find_element(By.ID, "username").click()
                driver.find_element(By.ID, "username").send_keys(usrnm)
                driver.switch_to.default_content()
                driver.execute_script("window.scrollTo(0,1080)")
                driver.find_element(By.ID, "password").click()
                driver.find_element(By.ID, "password").send_keys(passwd)
                driver.find_element(By.ID, "passwordc").click()
                driver.find_element(By.ID, "passwordc").send_keys(passwd)
                driver.switch_to.frame(0)
```

```
                                subbutton=driver.find_element(By.NAME, "
                                    submitBtn")
                                subbutton.click()
                                time.sleep(1)
                                driver.switch_to.default_content()
                                driver.find_element(By.ID, "confirmModalBtn"
                                    ).click()
                                counter=counter+1

                                usr = f.readline()
```

We were excited to begin creating a set of test users for our Facebook automated system . However, we quickly ran into a problem: when we attempted to use our ProtonMail-based email addresses to create Facebook accounts, we were told that ProtonMail email addresses were not allowed.

This was a surprise to us, as we had assumed that any email address could be used to create a Facebook account. After doing some research, we discovered that Facebook has strict rules about the types of email addresses that can be used to create accounts. According to their guidelines, email addresses from certain providers, including ProtonMail, are not allowed due to concern about spam and abuse. Those guidelines cannot be accessed anymore, since right now this restriction appears to be eliminated due to the fact now is compulsory to create accounts adding a phone number, but old guidelines can be checked using Wayback Machine (Internet archive)[25].

While this was a setback for our testing process, we realized that it was important to follow Facebook's guidelines in order to ensure that our automated system was functioning properly. We made the decision to switch back to Gmail, that is allowed by Facebook.

During our testing of the Gmail account creation process, we noticed that phone number verification was not always required. This may have been due to a variety of factors, such as the user's location, the type of device they were using, or even the time of day. We also encountered Captchas, which made the account creation process more difficult and time-consuming. However, we were determined to find a way to create accounts without having to provide a phone number.

After some research and experimentation, we discovered a method that allowed us to create accounts without phone verification. When accessing from a device that was trusted by Google, and using a chrome navigator, the phone number was less likely to get asked, and in case that happened, we will use a temporary phone number service, which allowed us to receive verification codes without having to provide our personal phone numbers. With this workaround, we were able to continue creating test accounts for our Facebook automated system.

4.4.3.2 Creating our set of Facebook users

When we began working on creating a set of Facebook users for our automated system, we initially attempted to automate the registration process. However, we quickly realized that this was not feasible due to the email activation step that required a confirmation code. As our main focus was building an interaction automated system for already created accounts, we decided to create the accounts manually.

To ensure that the account creation process was properly executed, we implemented a systematic approach. We created the table 4.1 that included each user's name, email, country, and VPN server. These are just some accounts created for the project. This enabled us to be careful with cookies and proper VPN usage, and ensured that the accounts were created in a proper and orderly manner.

Nombre de usuario	Email	Country	VPN server
Antiou Benito	antioubenito@gmail.com	PT	pt-lis.pvdata.host
Javier Rodriguez	javierrodriguezfb2020@gmail.com	ES	es-mad.pvdata.host
Savoir Marie	savoirmarie@gmail.com	FR	fr-par.pvdata.host
Malcom Sinneti	malcomsinneti@gmail.com	IT	it-mil.pvdata.host
Jonas Mozar	jonasmozar3@gmail.com	AT	at-wie.pvdata.host
Ella Mila	ellamilafb2020@gmail.com	BE	be-bru.pvdata.host
Alton Hester	adscomuc3m@gmail.com	DE	de-fra.pvdata.host
Adalia Heidi	adaliaheidi@gmail.com	DE	de-nur.pvdata.host
Roberto Perez	robertoperezfbstats@gmail.com	GR	gr-ath.pvdata.host
Otto Herrera	ottoherreradallas@gmail.com	US	us-dal.pvdata.host
David Rico	myphonedavid@gmail.com	ES	None
Ana rmy	an*******2@gmail.com	ES	None

TABLE 4.1. TABLE OF SOME FACEBOOK USERS CREATED

While creating accounts by hand was a more time-consuming process, it allowed us to have greater control over the accounts and ensured that they were set up correctly. This in turn allowed us to build a reliable automated system that could interact with the accounts without any issues.

4.4.3.3 The log-in functionality

The next step in our process was to create a script that would allow us to log in to Facebook using the accounts we had previously created. In order to do this, we created the FacebookLogger() class, which contained a loginaccount() function that would handle the log-in process.

CODE 4.4. Log in class

```python
    class FacebookLogger():
        def loginaccount(driver, fb_username, fb_password):
            baseUrl = "https://www.facebook.com/"
            driver.get(baseUrl)
            sleep(0.5)
            email = driver.find_element(By.XPATH,"//*[@id='email']")
            password = driver.find_element(By.XPATH,"//*[@id='pass']
                ")
            email.send_keys(fb_username)
            sleep(0.1)
            password.send_keys(fb_password)
            sleep(1)
            driver.find_element(By.XPATH,"//button[@data-
                testid='royal_login_button']").click()
            sleep(1)
```

The loginaccount() function takes three arguments: the driver, which is an instance of the WebDriver class provided by the Selenium library; fb_username, which is the username or email associated with the Facebook account we want to log in to; and fb_password, which is the password associated with the account.

The function first sets the base URL to "https://www.facebook.com/" and navigates to that page using the driver.get() method and waits for half a second using sleep().

Next, it locates the email and password input fields on the page using XPATH selectors, and enters the fb_username and fb_password values using the send_keys() method. It then waits for a short period of time before clicking the login button using another XPATH selector and the click() method.

This loginaccount() function was crucial in allowing us to automate interactions with Facebook, as it enabled us to log in to multiple accounts programmatically. However, creating this script was not without its challenges. One major challenge we faced was Facebook's use of Captchas and other security measures to prevent log-ins from unknown or unusual locations (for example, if you suddenly change location).

4.4.4. Publish functionality

When we started working on the publish functionality, we first analyzed what kind of content can be published on Facebook. We identified that text, images, videos, links, and other multimedia content can be posted on the platform. However, for our project, we decided to mainly focus on text and image posts as these types were the most common in the platform. We will divide this into three different scenarios: Only text, Image with text and only image (Images alone have been proven to be less engaging for the Facebook algorithm, but still implemented).

To implement this, we used a control variable named "media". When the value of this variable is set to 1, it indicates that the post will include both an image and text. On the other hand, when the value is set to any other integer, the post will only contain text. We randomly generate the value of this variable in Section 4.5.3.2.

Let's now break down the two cases in more detail:

4.4.4.1 Text publishing

When a text is selected to published, the AutoPublisher() method will simply generate a random line of text from a pre-defined file of clean books, and insert it into the text box of the Facebook post. The line of text is chosen randomly to add some variety to the posts, and to prevent duplicates from being posted. Once the text is inserted, the method clicks on the "Post" button to publish the post.

CODE 4.5. Text-only post

```
    else:
            post_box=driver.find_element(By.CSS_SELECTOR, ".jm1wdb64
                > .l9j0dhe7")
            post_box.click()
            sleep(3)
            text_box=driver.find_element(By.CSS_SELECTOR, ".rq0escxv
                > .\\_5rp7 .\\_1mf")
            text_box.click()
            text_box.send_keys(line)
            sleep(2)
            #var2=driver.find_element(By.XPATH,"//*[@class='n00je7tq
                arfg74bv qs9ysxi8 k77z8yql i09qtzwb n7fi1qx3
               b5wmifdl hzruof5a pmk7jnqg j9ispegn kr520xx4
               c5ndavph art1omkt ot9fgl3s rnr61an3']")  DEPRECATED
            var3=driver.find_element(By.CSS_SELECTOR, ".s1i5eluu")
            try:
                var3.click()
            except Exception as e:
               print(e)
            sleep(10)
```

This functionality was relatively straightforward to implement, since it only involves inserting text into a text box and clicking a button. However, it's important to note that the "quality" value Facebook gives to our posts will depend on the quality of the source material used to generate the random lines of text. Therefore, we took care to select a set of raw text from high-quality books, classic novels and other sources of content in different languages to ensure that the posts generated by the script would be interesting and engaging.

4.4.4.2 Image + Text publishing

When implementing the image publishing functionality, we encountered the challenge of handling two cases, Image only and the combination of text and image. In both cases, we will be uploading the image first, and then adding text if the “publish” flag was set to image+text (“publish” set to 1). For the text adding part, we decided to reuse the previous code to add text to a post, as reusability is a good programming practice (see figure 4.6).

CODE 4.6. Image and text post

```
if(media==1):

            FacebookImagePublisher.AutoImagePublisher(driver,0)
            ##Same as in only text from here ...
            text_box=driver.find_element(By.CSS_SELECTOR, ".rq0escxv
                > .\\_5rp7 .\\_1mf")
            text_box.click()
```

For the image uploading process, we created the ImagePublisher class, which contains the AutoImagePublisher function. This function accesses our pool of scraped images (as described in section 4.5.2) and randomly selects one to upload.

Once a suitable image is selected, its path is obtained, and after selecting the “upload media” option on Facebook, the path is inputted to initiate the upload process. We wait for the image to finish uploading before proceeding.

At this point, we check the publishing flag to determine whether we need to add text to the post or not. If the flag is set to image-only, we can click on the publish button right away.

However, if the flag is set to "image + text" (“publish” set to a value different from 1), we will need to add some text to the post. In this case, we skip the publishing step and return to the class that adds text to the post (as described in the previous section). By following this approach, we are able to automate the process of publishing images posts on Facebook.

CODE 4.7. Image and text post

```
def AutoImagePublisher(driver, publish):

        path= '../../Resources/downloaded_imgurs'
        img=random.choice(os.listdir(path))
        img_path= path + "/" + img
        print(img_path)
        image_box=driver.find_element_by_xpath('//input[@type="file
            "]')
        image_box.send_keys(img_path)
        sleep(3)
        if(publish==1):
            try:
                driver.find_element(By.CSS_SELECTOR, ".s1i5eluu").
                    click()
            except Exception as e:
               print(e)
            sleep(2)
```

4.4.5. Like functionality

When developing the "Like" functionality, we had to analyze how the like button was distributed over the main feed on a Facebook profile. This required us to inspect the HTML code of the Facebook page and identify the unique class name associated with the like button. We then had to come up with a way of obtaining an array containing the position of all the like buttons allowed on the page. This was accomplished by using the find_elements_by_xpath function, which allowed us to locate all the elements with the specified class name. Once we had the array, we could randomly pick some like buttons to click on.

Another challenge we faced was the need to scroll down the feed to refresh and increase the array and likes capability. We created a function called “RandomScroll” that can be found in Section 4.5.3.2, to perform this action, which scrolls down the feed and updates the position of the like buttons.

Finally, we had to handle the multiple execution errors that could happen during the operation of the script. Some examples of these errors are a failure to locate the like buttons, a click failure, or an error in the scrolling function. To tackle these issues, we implemented a counter variable called “tre” that keeps track of the number of errors and handles them accordingly. If the error count is below a certain threshold, the script waits for a few seconds before retrying. However, if the error count exceeds the threshold, the script refreshes the Facebook page and starts again from the beginning.

CODE 4.8. Text-only post

```
like_amount= int(like_desi)
        liked = 0
        tre = 0 #error counter
        i = 0
        while liked<like_amount:
            print ("Liked", liked,"/",like_amount)
            print("index="+str(i))
            sleep(5)
            newrand=RandomScroll(driver,rand)
            rand=newrand
            try:
                likes = driver.find_elements(By.XPATH,"//*[@class='
                    hu5pjgll m6k467ps sp_yMQbrHJc7-S sx_2d5f4e']")
                print(likes)
            except Exception as e:
                print (e)
                tre+=1
                pass
            if likes==[]:
                print("ERROR")
                return
            try:
                likes[i].click()
                print("liked?")
                liked +=1
                i+=1
                tre=0
            except Exception as e:
                print (e)
                tre+=1
                pass
            if tre < 20:
                print ("Refreshing in.. :", 20-tre)
                sleep(1)
            else:
                print ("Things are bad. refreshing")
                tre=0
                driver.get("http://www.facebook.com")
                sleep(5)
                i = 0
                rand = 600
```

Overall, the "Like functionality" required a thorough analysis of Facebook's interface and the development of a robust error handling mechanism to ensure the script's reliability.

4.4.6. Page Manager (discontinued)

In the middle stage of the project, we came across the interesting fact that Facebook pages associated with a profile were more likely to receive a higher query limit than normal ones. We thought about taking advantage of this and implementing a Page Manager, which would allow the user to create and manage Facebook pages in an automated way. By creating and managing a Facebook Page, the status of the account can increase into a Business account to Facebook's eyes, and this will increase the queries cap.

The idea was to use the Facebook API to create a new page and then associate it with the user's profile. This would allow us to have a higher query limit for the profile, which could then be used to carry out the desired actions more efficiently. We also planned to implement features such as automatically posting on the page, scheduling posts, and analyzing the account's performance vs. non-page ones.

For testing purposes, I created a couple of pages manually and then started working on the automation.

But, after further research and testing, we decided to evaluate the effort cost and results versus other solutions such as increasing our number of users active, and we found that we could effectively increase the query limit for the whole system without needing to create an associated page.

Given this, we decided to discontinue the development of the Page Manager feature and focus on improving the already created "account health" while increasing the number of active users. This allowed us to achieve higher query limits and carry out actions more efficiently without the need to create and manage Facebook pages.

However, I would like to show how the early skeleton of the page manager looked, such as the so-called "fan page creation":

CODE 4.9. Test draft for fan page creation

```
class FacebookPagger():
    def createfanpage(driver, fb_username):
        baseUrl = "https://www.facebook.com/pages/creation/?ref_type
            =comet_home"
        driver.get(baseUrl)
        sleep(5)
        try:
                title = driver.find_element(By.XPATH,"//*[@id='
                    jsc_c_bb']")
                categ = driver.find_element(By.XPATH,"//*[@id='
                    jsc_c_bf']")
                desc = driver.find_element(By.XPATH,"//*[@id='
                    jsc_c_bh']")
                button = driver.find_elements(By.XPATH,"//*[@class='
                    oajrlxb2 s1i5eluu gcieejh5 bn081pho humdl8nn
                    izx4hr6d rq0escxv nhd2j8a9 j83agx80 p7hjln8o
                    kvgmc6g5 cxmmr5t8 oygrvhab hcukyx3x jb3vyjys
                    d1544ag0 qt6c0cv9 tw6a2znq i1ao9s8h esuyzwwr
                    f1sip0of lzcic4wl l9j0dhe7 abiwlrkh p8dawk7l
                    beltcj47 p86d2i9g aot14ch1 kzx2olss cbu4d94t
                    taijpn5t ni8dbmo4 stjgntxs k4urcfbm tv7at329']")
        except Exception as e:
                print (e)
                return

        title.send_keys(fb_username+ "fan page")
        sleep(0.1)
        categ.send_keys("Fan Page")
        sleep(0.1)
        try:
                scroll = driver.find_elements(By.XPATH,"//*[@class='
                    d2edcug0 hpfvmrgz qv66sw1b c1et5uql gk29lw5a
                    a8c37x1j keod5gw0 nxhoafnm aigsh9s9 d9wwppkn
                    fe6kdd0r mau55g9w c8b282yb hrzyx87i jq4qci2q
                    a3bd9o3v lrazzd5p oo9gr5id']")
                scroll[scroll.size()-6]
        except Exception as e:
                print (e)
                return

        desc.send_keys("Welcome to my own fan page")
        sleep(2)
        button.click()
        sleep(1)
```

4.5. How do we make our different users unique

In the development of automated systems to interact with social media platforms, one of the challenges is to make the automated users appear human-like and not trigger any suspicion from the platform's algorithms. One way to achieve this is to make them look unique and independent, and to add a level of randomness to their behavior. By doing so, the automated users can blend in better with human users, and avoid being detected as automated.

To support this argument, research has shown the importance of this approach. For example, Zhu [26] found that social automation on Twitter that exhibited similar behavior were more likely to be detected and suspended. Similarly, Duggan [27] noted that the use of social media automated systems by foreign actors to interfere in political campaigns has raised concerns about the impact of automated systems on democratic processes.

In addition, studies have proposed different strategies to make automated users appear more human-like. One such approach is the use of social randomization, as proposed by Gomez and Pavon [28], which involves introducing randomness in automated system behavior to reduce the detection of autonomous bot-based accounts in online social networks.

Overall, these studies highlight the importance of making automated users appear unique and random in order to avoid detection by social media platforms, and provide guidance on how to achieve this.

Following the recommendations from previously mentioned sources as well as those from my project supervisors and my personal research, I have developed three different approaches to make our interactions with Facebook appear more human-like and unique. These strategies aim to add a level of randomness and diversity to the bot's behavior, while also avoiding any suspicious patterns that could be detected by Facebook's algorithms. In the following sections, I will explain each approach in detail and provide examples of how they can be implemented in our automated system.

4.5.1. Proxies and IP-Check

One way to make our automated systems appear more human-like is by using proxies. Proxies act as intermediaries between our automated users and the social media platform, making it more difficult for the platform to track the origin of that connection. In the context of automated systems interacting with social media platforms, proxies can be used to mask the IP address of the bot, making it more difficult for the platforms to detect and flag it as a bot. By using numerous proxies, we can also give each of our users a unique identity, further increasing their human-like appearance.

According to Barnes and Srinivasan [29], using proxies can decrease the risk of detection in bot-based online social systems. In their study, they found that the use of proxies helped to avoid their users avoid detection, and that the risk of detection decreased as the number of proxies used increased. Additionally, Wright [30] showed that using proxies and randomized accounts can help evade Twitter's automated user detection mechanisms.

4.5.1.1 Public vs private proxies

Now that we understand the basics of proxies, let's take a closer look at the two approaches we considered for our project. At first, we considered using public proxy servers like the ones found on `https://free-proxy-list.net/`, `https://www.proxy-list.download/api/v1/get?type=https`, and `https://api.proxyscrape.com/?request=displayproxies&proxytype=http&timeout=1000`. We thought that by scraping and rotating through these proxies over the requests made, we could effectively hide our bot's identity.

To do so, we developed the following scrapper:

CODE 4.10. Public proxies scrapper

```
def get_proxies_from(url):
    try:
        response = requests.get(url)
        response = response.text.split('\r\n')
        proxies = list(set(response))
    except:
        pass
    return proxies

def proxy_cleaner(tp, tw):
    url_to_check="https://httpbin.org/ip"
    try:
        r = requests.get(url_to_check, timeout=20)
        if r.status_code == 200:
            response = json.loads(r.text)
            prox_web = response['origin']
            if(prox_clean==prox_web):
                tw.append(original_prox)
            else:
                print("ERROR->  "+prox_web+"  vs  " + prox_clean)
    except Exception as e:
        print (e)

def setproxy(prox):
    proxy = prox.pop()
    print("setting up "+proxy)
    os.environ[’http_proxy’] = proxy
    os.environ[’HTTP_PROXY’] = proxy
    os.environ[’https_proxy’] = proxy
    os.environ[’HTTPS_PROXY’] = proxy
    return proxy

url1="https://www.proxy-list.download/api/v1/get?type=https"
url2="https://www.proxy-list.download/api/v1/get?type=http"
url3="https://api.proxyscrape.com/?request=displayproxies&proxytype=http&timeout=1000"

proxies_working=[]
total_proxies=get_proxies_from(url1)
total_proxies+=get_proxies_from(url2)
total_proxies+=get_proxies_from(url3)

for ele in total_proxies:
    original_prox=setproxy(total_proxies)
    prox_clean=(original_prox.split(’:’))[0]
    proxy_cleaner(total_proxies, proxies_working)

with open("proxy_ok.txt",’w’) as le:
    for ele in proxies_working:
        le.write(ele+’\n’)
```

In snippet 4.10, we can see how the scraper accesses those webpages and extracts the list of proxies. But, simply obtaining a list of proxies is not enough. We also need to verify whether those proxies are functional. To do this, we set up each proxy and send a request to https://httpbin.org/ip. If the proxy is working, we save it for future use. If not, we discard it and move on to the next one.

However, this approach came with some significant concerns. Some proxies on these public servers may be used for malicious purposes, such as spamming, hacking, or distributing malware. Additionally, constantly accessing the platform from a different location may not seem very human-like.

For these reasons, we ultimately decided to use a private proxy provider, specifically Private VPN in our case, since it was used previously for a scrapping task in a different project, and gave good results. By using the directions provided by the Private VPN, we were able to assign one or two private proxies to each of our bot/users, making them to access the platform only through those designated proxies. This approach not only provides a more secure and reliable way of hiding our bot's identity, but also ensures that their access patterns more closely resemble those of human users.

4.5.1.2 Private proxies final implementation

Following the proxy per user approach we mentioned, firstly we will need to map the different proxies to each user. To do so, we will use a dictionary as follows:

CODE 4.11. Username to proxy dictionary

```
    username_to_ppp = {
        "antioubenito@gmail.com": "pt-lis.pvdata.host",
        "javierrodriguezfb2020@gmail.com": "es-mad.pvdata.host",
        "savoirmarie@gmail.com": "fr-par.pvdata.host",
        "malcomsinneti@gmail.com": "it-mil.pvdata.host",
        "jonasmozar3@gmail.com": "at-wie.pvdata.host",
        #*
        #*       we continue adding all users
        #*
    }

    ppp = username_to_ppp.get(usrname)

    if ppp is None:
        return None
```

In order to use a selected proxy with the Chrome web browser, we need to create a Chrome extension that modifies the proxy values in the background. This extension will be added to the chromedriver. The configuration for this extension will be defined in the manifest.json file (code 4.11), which contains the name of the extension, permissions required to run, and the script it will be running.

CODE 4.12. manifest.json

```
manifest_json = """
        {
            "version": "1.0.0",
            "manifest_version": 2,
            "name": "Chrome Proxy",
            "permissions": [
                "proxy",
                "tabs",
                "unlimitedStorage",
                "storage",
                "<all_urls>",
                "webRequest",
                "webRequestBlocking"
            ],
            "background": {
                "scripts": ["background.js"]
            },
            "minimum_chrome_version":"22.0.0"
        }
        """
```

We will then create a JavaScript script (see code 4.13) that configures the actual proxy. The script creates a configuration object with a fixed mode, single proxy, and a bypass list of localhost. This configuration object is set as the value for the Chrome proxy settings using the chrome.proxy.settings.set() function.

CODE 4.13. background.js

```
background_js = """
        var config = {
                mode: "fixed_servers",
                rules: {
                singleProxy: {
                    scheme: "http",
                    host: "%s",
                    port: parseInt(%s)
                },
                bypassList: ["localhost"]
                }
            };
        chrome.proxy.settings.set({value: config, scope: "regular"},
             function() {});
        function callbackFn(details) {
            return {
                authCredentials: {
                    username: "%s",
                    password: "%s"
                }
            };
        }
        chrome.webRequest.onAuthRequired.addListener(
                    callbackFn,
                    {urls: ["<all_urls>"]},
                    ['blocking']
        );
        """ % (PROXY_HOST, PROXY_PORT, PROXY_USER, PROXY_PASS)
```

The script also defines a callback function that handles authentication when it is required for accessing a web resource. The function returns a username and password for the proxy server, which are supplied from the PROXY_USER and PROXY_PASS variables. Finally, the script adds this callback function as a listener to the chrome.webRequest.onAuthRequired event, which fires when authentication is required for a web request.

4.5.2. Getting the raw resources (image and text)

Obtaining raw resources such as images and text files is a crucial step in creating a automated system that looks human-like. This is because incorporating these resources into the bot's responses can make them more engaging and reliable to Facebook. Images are specially important since at that date, Facebook pushed user's to share images and rewards accounts who did so.

In order to generate more natural-looking posts, we recognized the importance of obtaining source material such as text files and images. To achieve this, we created pools of raw text files in various languages, drawing from sources such as novels, news articles, and other text-based content.

Obtaining images proved to be a more complex task. To automate the process, we utilized a script developed by Usman Mahmood [31] as our base. This script allowed us to retrieve a random image from Imgur, which we then modified to retrieve sets of random images from the website. However, we later discovered that some of the downloaded images may not comply with Facebook's restrictions. As a result, we had to filter them to ensure that the images we obtained were appropriate for use in our automated system.

We leveraged the existing functions save_image(download_dir, file_name, file_ext, img_data) and download_imgur(url) for downloading and saving images. With these functions as a foundation, we developed a new main function that takes two arguments: the number of images to be downloaded and the directory where they should be saved. We made necessary modifications to these functions to fit our requirements, allowing us to easily download multiple images from Imgur and save them to a designated directory.

CODE 4.14. main loop for image downloader

```
        count = 0
        while count < max_imgs:
                imgur_url, file_name = get_imgur_url()
                img_data, file_type = download_imgur(imgur_url)
                if not img_data or is_placeholder_image(img_data):
                        continue
                save_image(download_dir, file_name, file_type,
                   img_data)
                if verbose: print('Downloaded image',imgur_url)
                count += 1
```

4.5.3. Human-like interaction

To avoid being flagged as an automated system by Facebook, our system needs to mimic human behavior as closely as possible. This includes not only the content of the posts and comments, but also the timing and frequency of the interactions. In this section, we will discuss how we implemented two strategies to achieve this.

4.5.3.1 Scheduling the access

One important aspect of human-like behavior is that people don't interact with Facebook continuously throughout the day, but rather in discrete periods of time. To simulate this, we scheduled the access to Facebook using an always running script that every day, between 10 a.m. and 10 p.m. wakes up at a random given time our main.py script.

CODE 4.15. Wake-up Scheduler

```python
import random
import time
import subprocess

while True:
    current_time = time.localtime()
    time.sleep(1800 )
    if current_time.tm_hour >= 10 and current_time.tm_hour < 22:
        random_hour = random.randint(10, 21)
        time_until = (random_hour - current_time.tm_hour) * 3600 -
            current_time.tm_min * 60
        time.sleep(time_until)
        subprocess.call(['python3', 'main.py'])
```

4.5.3.2 Random timers and Action selection

Another aspect of human-like behavior is that people don't always interact with Facebook at regular intervals or in the same way every time. To simulate this, we implemented a system of random timers and action selection in our bot.

In particular, we used the Python random library [32] to introduce some randomness in the timing and type of actions that our automated system performs

Upon execution, the main.py script loads user data and randomly selects a user to log in to Facebook to then load the user's proxy profile. Once successfully logged in, the script randomly selects an action to perform, such as giving likes, posting text, posting images and text, or only posting images.

Randomized wait times are implemented between log in, action, and log out phases. When a user logs out, they are moved from the pool of unlogged users to the logged-in pool to prevent duplicating log-ins. The script shuffles the users and selects a new one until all users have been cycled through. The web browser is restarted and cleared for each user to ensure a completely fresh session without any lingering cookies or other traces.

CODE 4.16. Wake-up Scheduler

```python
def actions(driver):
    q=random.randint(10, 50)
    while 10<q:
        x=random.randint(1, 4)
        if x==2:
            driver.get("https://www.facebook.com/")
            sleep(random.randint(1, 3))
            if(newfacebook):
                FacebookPublisher.AutoPublisher(driver, 0)
            else:
                FacebookPublisher.AutoPublisher_old(driver, 0)
            q=q-10
        if x==1:
            driver.get("https://www.facebook.com/")
            sleep(random.randint(1, 3))
            if(newfacebook):
                FacebookLiker.AutoLiker(driver,2)
            else:
                FacebookLiker.AutoLiker_old(driver,2)
            q=q-10
        if x==3:
            driver.get("https://www.facebook.com/")
            sleep(random.randint(1, 3))
            if(newfacebook):
                FacebookImagePublisher.AutoImagePublisher(driver,1)
            else:
                FacebookImagePublisher.AutoImagePublisher_old(driver
                    ,1)
            q=q-10
        if x==4:
            driver.get("https://www.facebook.com/")
            sleep(random.randint(1, 3))
            if(newfacebook):
                FacebookPublisher.AutoPublisher(driver, 1)
            else:
                FacebookPublisher.AutoPublisher_old(driver, 1)
            q=q-10
print("*"*54)
print ("*"*11,"WELCOME  AT  FACEBOOK  MANAGER","*"*11)
print("*"*54)
logged=[]
filedir="/home/davidryt/FBNetwork/Resources/Text/names.txt"
with open(filedir) as f:
    usrnm = f.read().splitlines()
    passwd = "FBStats2020"
unlogged = usrnm
while unlogged:
    log=random.choice(unlogged) #Randomized
    if log not in logged:
        print("SELECTED USER= "+log)
        driver=ProxySetup.get_chromedriver(log,use_proxy=True)
        if driver == None:
            print("driver init extra")
            driver=driver_init()
        driver.get('https://httpbin.org/ip')
        driver_wait(driver) #Randomized
        login(driver, log, passwd)
        actions(driver)     #Randomized
        logged.append(log)
        unlogged.remove(log)
        driver_wait(driver) #Randomized
        driver_close(driver)
```

In conclusion, our system implements two strategies to simulate human-like behavior and avoid being flagged as an automated system by Facebook. Scheduling the access to Facebook enables our system to mimic the discrete periods of time that people interact with the platform, while the system of random timers and action selection introduces randomness in the timing and type of actions performed by our bot. By implementing these strategies, our automated system can interact with Facebook in a more natural and unpredictable way, reducing the risk of being detected as an automated system and increasing the chances of successfully achieving its intended goals. The use of the Python random library and the careful design of our main.py script allow us to effectively mimic human behavior and navigate the platform without raising any red flags.

5. RESULTS AND TESTS

This chapter provides an in-depth analysis of the outcomes and performance of the automated system. This chapter aims to present the findings obtained through various tests and measurements conducted during the evaluation phase. By examining the system's effectiveness, efficiency, and adherence to desired outcomes, we gain valuable insights into its overall performance and capabilities. We will categorize the analysis into three distinct sections, reflecting the different stages of results and development:

1. Initial Results: This section focuses on the initial outcomes of the automated system during its early stages. It encompasses the performance and effectiveness of the system when it was first implemented and provides insights into its initial capabilities, challenges encountered, and any preliminary successes achieved.
2. Results Before Break: In this section, we examine the outcomes and performance of the automated system leading up to the break period triggered due to the problem encountered on VPNs mentioned in section. It delves into the system's progress, improvements, and any significant findings obtained prior to this. This section sheds light on the system's evolution and performance trends over time.
3. Results After Break: Following the break, this section explores the post-break results and the system's performance during the subsequent development phase. It examines any changes, enhancements, or adjustments made to the system and how it regenerated over the time.

By dividing the analysis into these three sections, we can provide a comprehensive evaluation of the system's performance and development over time. This approach allows for a deeper understanding of the system's capabilities, challenges, and improvements at different stages, ultimately contributing to a more robust assessment of its effectiveness and potential for future enhancements

5.1. Measurement Criteria

In order to evaluate the success rate and satisfaction of the automated system, several key measurement criteria have been identified. These criteria serve as quantitative indicators to assess the performance of the system and its impact on user accounts. The following criteria will be represented as columns in the result tables, and they have been chosen based on their relevance to the evaluation process:

1. User: Records the unique identifier assigned to each user account. It helps in tracking individual accounts and their associated metrics throughout the evaluation.

2. Proxy: Documents the proxy used for each user account. Proxies play a crucial role in maintaining account privacy and avoiding detection. Certain areas could return better results than others, so it is important to take it into account.

3. Date Created: Indicates the date when the user account was initially created in the system. It helps in analyzing the account's longevity and identifying trends over time.

4. Date Canceled: If an account was canceled, this records the date of cancellation. It helps in tracking the account's lifecycle and identifying any patterns or reasons for account cancellations.

5. Reason for Cancellation: If an account was canceled, this provides a brief explanation or reason for the cancellation, when known. It helps in understanding the factors contributing to account terminations.

6. Number of Times Recovered: Again, if an account needed to be recovered, this column captures the number of times recovery was performed. Recovered account were probably flagged, and this could have an impact on their lifetime and performance. It helps in assessing the system's ability to handle and restore accounts effectively.

7. Lifetime: This column measures the duration for which an account remained active within the system. It provides insights into account usage and longevity.

8. Number of Actions Performed: Sectioned into likes or posts, recording the number of each action performed by each user account. It helps in evaluating the system's engagement levels and activity.

9. Number of Social Links Established: This counts the number of social links established by each user account. It indicates the extent to which the system was able to expand and connect with other users. Link establishment was done manually most of the time, but some accounts were use as test proves for expanding the system, and if an automated tool was used, would be highlighted in red.

5.2. Initial results

In the initial phase of our system's testing, we decided to start creating a diverse set of eight accounts. However, we encountered several challenges and observed a range of outcomes during this early stage. This comprehends between 1st of July until mid-October 2020.

Out of the eight accounts, three were immediately detected and deactivated by Facebook due to invalid email extensions, rendering them unsuitable for our purposes. We decided to keep going with the remaining accounts, so we were able to have at least a little set already being tested.

The five remaining accounts were strategically designed to include a combination of automated user accounts and one human-verified account. This approach aimed to provide a balance between automation and human-like behavior within the system. The inclusion of the human-verified account added an element of authenticity and credibility to the set, allowing us to simulate more realistic user interactions.

During the early stages of system implementation, we faced various issues that needed to be addressed. There were instances of unhandled exceptions, execution errors that led to left open sessions or unfinished ones, accounts accessing more frequently than expected, and insufficient time gaps between consecutive accesses. As a result, two out of the five remaining accounts were identified and deactivated. Since one account had human background and origin, and cannot be taken into account when examining results, we can say that this initial round yielded a success rate of 50% for the accounts that were active and functioning as intended. It is worth noting that the identification and deactivation of accounts that did not meet the expected behavior criteria highlight the Facebook system's capability to discern between human-operated and automated accounts, further emphasizing the importance of achieving a high level of account authenticity.

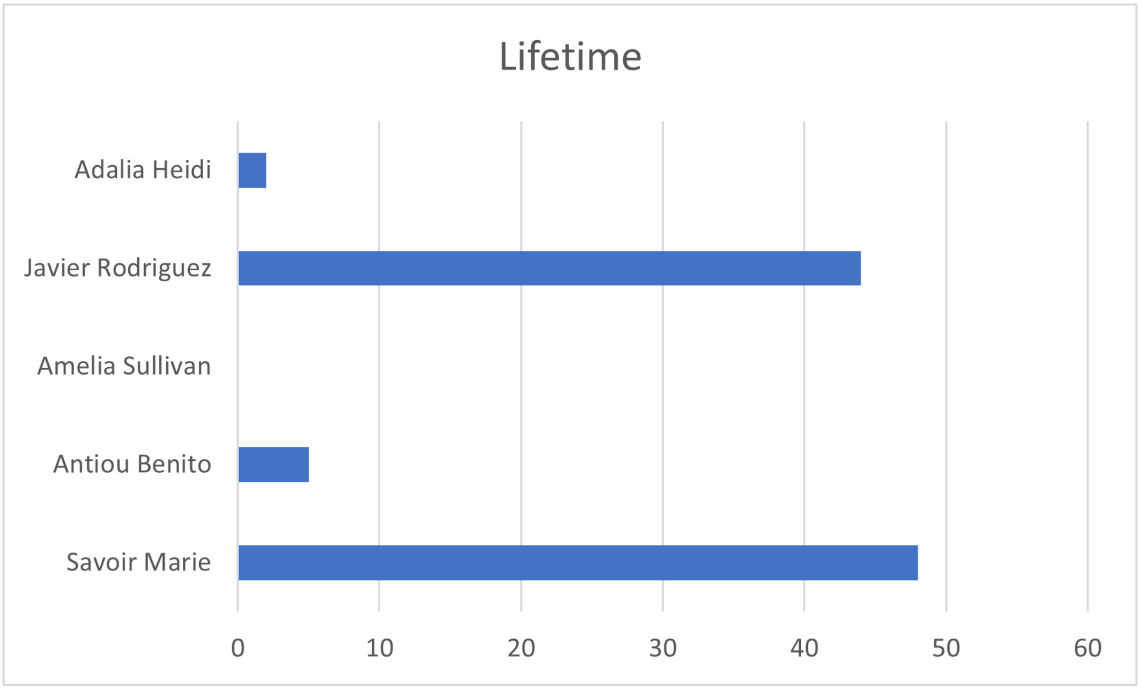


Fig. 5.1. First results lifetime

To provide a comprehensive overview and facilitate a detailed analysis of the initial results, we have included Graph 5.1, which illustrates the lifespan and activity of each account, and Table 5.1, which presents a breakdown of the essential metrics previously discussed.We must clarify that not all the system tools were implemented when we started the testing, and features such as multiple actions in one session, random wait times between clicks on buttons or scrolls or image+text posts were not available.

From this initial approach, we gained valuable insights into the need for meticulousness and system improvements. We recognized the importance of handling exceptions effectively and ensuring robust user access timer and session management. These lessons highlighted the significance of refining our system's resilience and stability. By addressing these areas, we aimed to enhance the overall performance and reliability of our system and minimizing disruptions.

Username	Country	VPN	Creation	Deactivation	Lifetime	Reason	Actions
Savoir Marie	FR	fr-par.pvdata.host	28/08/2020	NO	48		45
Antiou Benito	PT	pt-lis.pvdata.host	29/08/2020	03/09/2020	5	Disabled after fast switch	6
Amelia Sullivan		uk-lon.pvdata.host	30/08/2020	30/08/2020	0	Disabled on creation	0
Javier Rodriguez	ES	es-mad.pvdata.host	01/09/2020	NO	44		43
Adalia Heidi	DE	de-nur.pvdata.host	01/09/2020	03/09/2020	2	Disabled after session error	2

TABLE 5.1. FIRST RESULTS ANALYSIS

5.3. Results before break

In this section, we will delve into the outcomes achieved by our system during the period from early December 2020 to September 10, 2021, prior to the events explained on 6.1. This timeframe marked a significant milestone for us, as we were able to achieve peak performance and system volume, with over 12 accounts operating seamlessly.

A crucial aspect of evaluating our system's effectiveness is the detection and blocking of accounts by Facebook. During this period, only three accounts were identified and subsequently blocked. This low detection rate demonstrates the robustness of our system in maintaining account security and avoiding actions that trigger Facebook's security measures.

To provide a comprehensive overview, we have compiled data on the number of login instances and actions performed by these accounts in Table 5.2. This data enables us to assess the level of activity generated by each account and lifetime, as shown in Figure 5.2. From this data, it is evident that our system operated as intended, remaining unnoticed by Facebook algorithms and consistently performing actions on the platform.

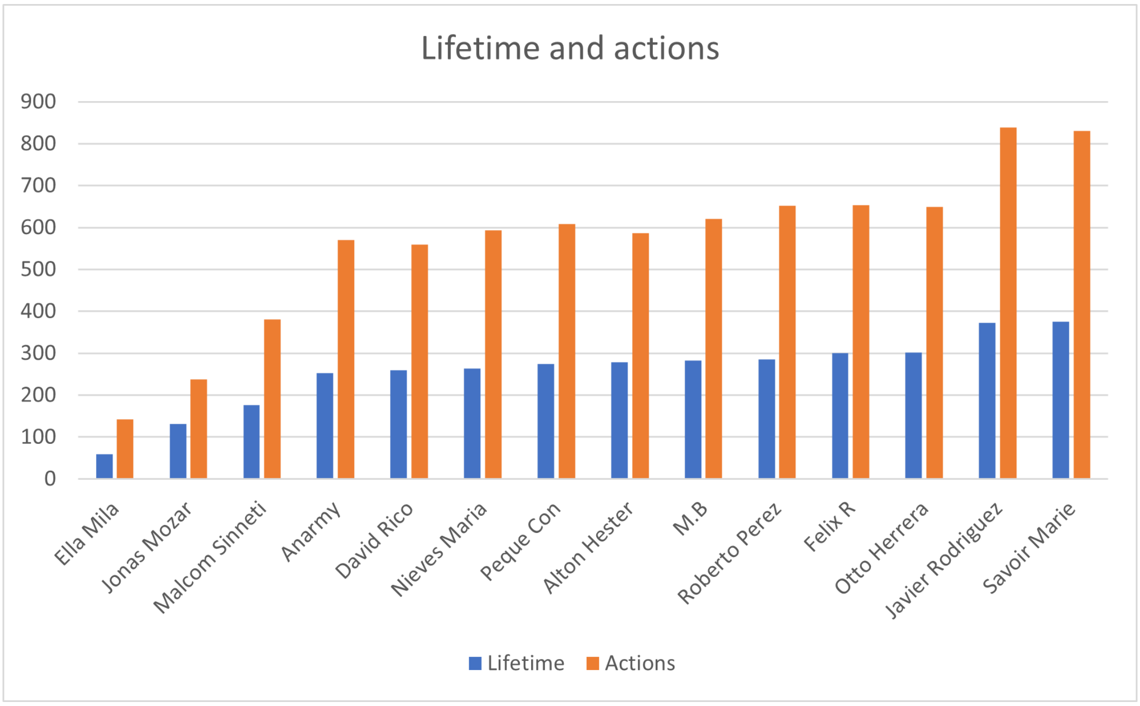


Fig. 5.2. Second results lifetime + actions

Throughout the duration of this period, the system required 4 maintenance shutdowns to update certain components of the main execution loop. As a result, the recorded number of actions is slightly lower than expected, as indicated in Table 5.4. The expected number of actions is calculated by multiplying the number of available days for each account by the average daily actions (ranging between 1 and 5). The deviation illustrated in Figure 5.3 shows that the margin of error is consistent across all accounts, which supports our previous explanation.

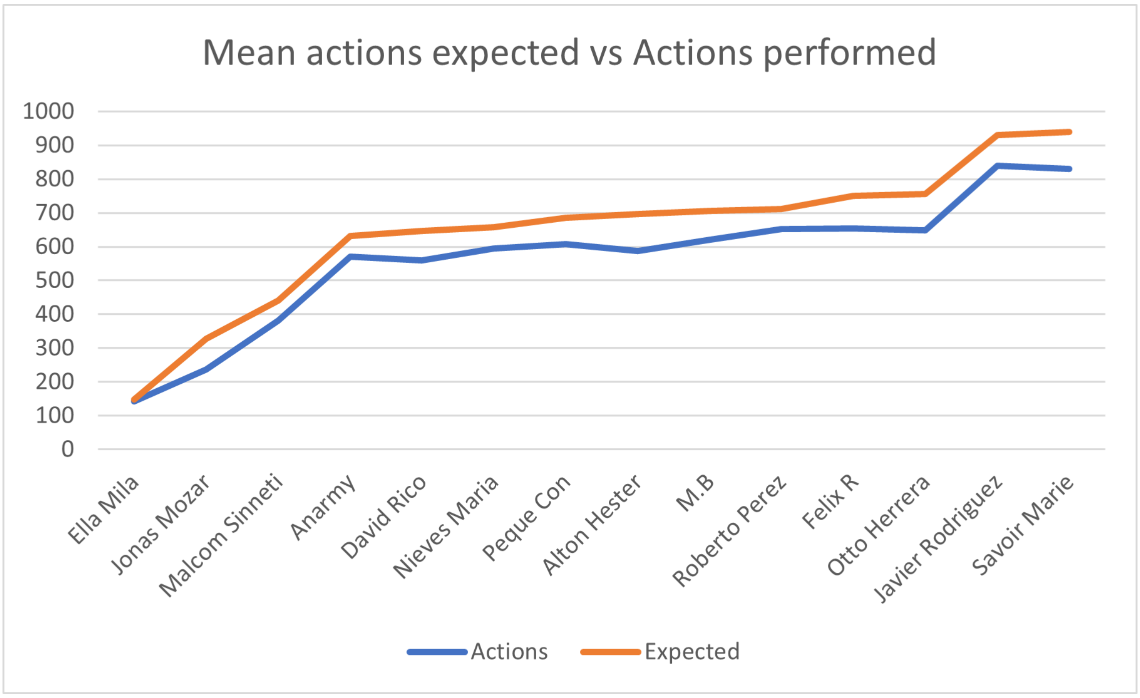


Fig. 5.3. Expected actions vs real actions performed

In addition to the login instances and actions performed, we have also examined the information on social links established by the accounts during this period. By categorizing the accounts into human-originated (In green), accounts with auto-friend enablement (In blue), and regular automated user accounts, we can observe the varying degrees of social interaction facilitated by each account type.

Human-originated accounts tend to have a higher number of social links, often exceeding 100. Accounts with auto-friend enablement typically have around 40 to 50 connections, representing a 100% increase over the regular automated user accounts, that due to the system's limited size, maintain between 10 and 20 connections.

Based on these findings, we can assert that we have successfully achieved our objectives. The results demonstrate the effectiveness and stability of our system, allowing us to proceed with scaling up by adding new accounts without compromising its autonomous nature, which was one of our primary goals.

In conclusion, the analysis of the data gathered during this period indicates that our system operated smoothly, achieving satisfactory results. The system's stability and performance provide confidence in expanding its size and capabilities, paving the way for further advancements in our research.

5.4. Results after the break

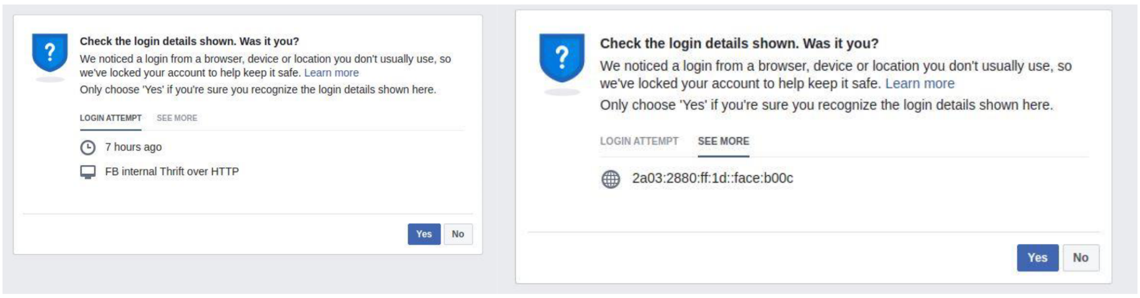
In this last section, we focus on the period between October 1st and January 1st, which marked the end of the project. Following the VPN breakdown discussed in Section 6.1,

Username	Country	VPN	Creation	Deactivation	Lifetime	Reason	Actions
Ella Mila	BE	be-bru.pvdata.host	14/12/2020	11/02/2021	59	Suspicious Login	142
Jonas Mozar	AT	at-wie.pvdata.host	18/11/2020	29/03/2021	131	Picture identification required	237
Malcom Sinneti	IT	it-mil.pvdata.host	22/12/2020	16/06/2021	176	Phone number verification	381
Anarmy	ES	None	29/12/2020	08/09/2021	253		570
David Rico	ES	None	23/12/2020	08/09/2021	259		559
Nieves Maria	ES	es-mad.pvdata.host	19/12/2020	08/09/2021	263		594
Peque Con	ES	Personal VPN	08/12/2020	08/09/2021	274		608
Alton Hester	DE	de-fra.pvdata.host	03/12/2020	08/09/2021	279		587
M.B	ES	es-bcn.pvdata.host	30/11/2020	08/09/2021	282		621
Roberto Perez	GR	gr-ath.pvdata.host	27/11/2020	08/09/2021	285		652
Felix R	ES	es-mad.pvdata.host	12/11/2020	08/09/2021	300		653
Otto Herrera	US	us-dal.pvdata.host	10/11/2020	08/09/2021	302		649
Javier Rodriguez	ES	es-mad.pvdata.host	01/09/2020	08/09/2021	372		839
Savoir Marie	FR	fr-par.pvdata.host	28/08/2020	08/09/2021	376		830

TABLE 5.2. ACCOUNT STATUS SECOND ITERATION

we faced significant challenges as all our existing accounts were flagged, blocked, and banned. To ensure the timely retrieval of essential data for the project, we needed to assess the extent of the damages incurred.

Initially, we attempted to recover some of the accounts and succeeded in restoring seven of them. However, when we attempted to set up the system again using these accounts, we encountered the "FB Internal Thrift over HTTP" error (see Figure 5.4). Lacking a clear solution, we conducted experiments and performed detailed analyses and reports to understand the nature of this error. Our findings revealed that accounts associated with the breached system had a significantly low tolerance for normal logins. After a few login attempts, these accounts would periodically trigger the error and request a password update.


Check the login details shown. Was it you?
We noticed a login from a browser, device or location you don't usually use, so we've locked your account to help keep it safe. Learn more
Only choose 'Yes' if you're sure you recognize the login details shown here.

7 hours ago
FB internal Thrift over HTTP

Check the login details shown. Was it you?
We noticed a login from a browser, device or location you don't usually use, so we've locked your account to help keep it safe. Learn more
Only choose 'Yes' if you're sure you recognize the login details shown here.

2a03:2880:ff:1d::face:b00c

Fig. 5.4. Error shown by Facebook

Considering this situation, we made the strategic decision to create an entirely new set of accounts from scratch. We established a new environment by deploying our system on a completely new virtual machine, ensuring no connection could be made to previous accounts by Facebook or any potential identifiers (e.g., IP addresses, locations). While this decision prolonged the process of setting up our system, it was essential to maintain the integrity and autonomy of the newly established accounts. This approach effectively mitigated the risk of Facebook associating the new accounts with the previous ones.

Username	Lifetime	Amigos
Ella Mila	59	-
Jonas Mozar	131	-
Malcom Sinneti	176	-
Anarmy	**253**	**39**
David Rico	**259**	**202**
Nieves Maria	**263**	**59**
Peque Con	274	25
Alton Hester	279	13
M.B	282	23
Roberto Perez	**285**	**52**
Felix R	**300**	**101**
Otto Herrera	302	22
Javier Rodriguez	372	19
Savoir Marie	376	**44**

TABLE 5.3. FRIENDS AT THE END OF SECOND ROUND

Despite the challenges faced, we successfully built a stable set of six accounts within two months, and this time all of them were operative until the end of the process, obtaining an impressive 100% success (we must also keep in mind that this last set of account were tested on a significantly smaller period of time, and this could contribute to increasing the success rates). The careful implementation of a fresh environment, new virtual machine, and new IP addresses ensured the smooth operation of our system. During this period, we also made experiments on creating pages for these accounts. This new set of accounts provided the necessary foundation for us to resume our activities and achieve stability in our operations.

By overcoming the setbacks caused by the VPN breakdown and implementing a robust strategy for account creation, we were able to navigate the obstacles and establish a reliable system once again. The resilience and adaptability demonstrated during this phase highlight the effectiveness of our approach and reaffirm our commitment to the project's objectives.

Username	Lifetime	Actions	Expected
Ella Mila	59	142	147
Jonas Mozar	131	237	327
Malcom Sinneti	176	381	440
Anarmy	253	570	632
David Rico	259	559	647
Nieves Maria	263	594	657
Peque Con	274	608	685
Alton Hester	279	587	697
M.B	282	621	705
Roberto Perez	285	652	712
Felix R	300	653	750
Otto Herrera	302	649	755
Javier Rodriguez	372	839	930
Savoir Marie	376	830	940

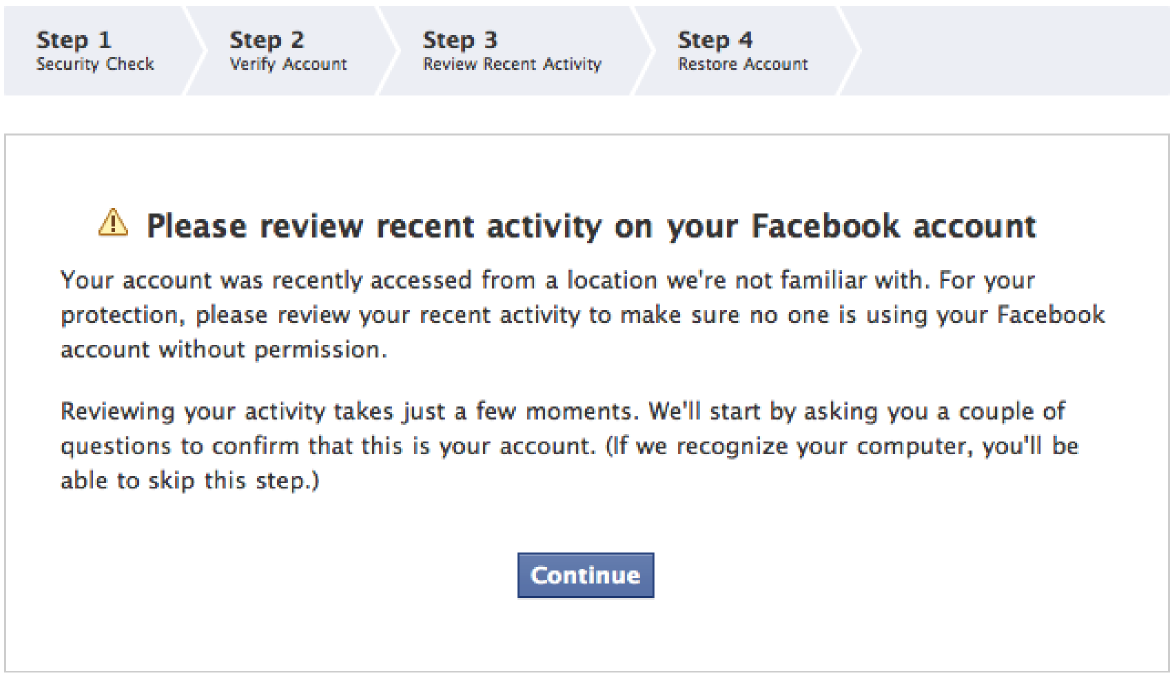
TABLE 5.4. ACTIONS PERFORMED VS EXPECTED

6. ISSUES OVERCOME DURING THE BACHELOR THESIS

6.1. VPN failure and account blocking

During the lifetime of our automated system, which had been performing at its peak in terms of stability and functionality as shown in the previous section, we encountered a significant setback around September 10, 2021. Suddenly, approximately 75% of the accounts within our system began experiencing issues such as dropping accounts, blocked access, and overall malfunctioning. Since our script system operated autonomously, with minimal human intervention needed, it was crucial to rely on regular log checks to identify any anomalies. Fortunately, due to my routine practice of reviewing the output log every two days, I quickly noticed something unusual occurring within the system.

Within 24 hours of detecting the problem, we made the decision to completely shut down the system to prevent further damage to the reliability of the affected accounts, and try to understand what was going on. Our immediate priority was to investigate the cause of the mass account banning by Facebook. Through a deep analysis of our detailed log system, we were able to identify the cause of the issue. It was discovered that our VPN service subscription had not been renewed, resulting in a loss of access to our proxy servers. Consequently, when our accounts attempted to select a proxy, they encountered a new type of exception that had not been handled (this was a mistake, and we should have handled any error into the stop of the access), leading the operation to continue with the server's default proxy. As a result, all the accounts were accessing and performing actions on the same IP domain. This, added to the fact that many of these accounts were typically accessed from different countries, resulting in a significant location change, triggering Facebook's security alerts.


Step 1 Security Check

Step 2 Verify Account

Step 3 Review Recent Activity

Step 4 Restore Account

⚠ Please review recent activity on your Facebook account

Your account was recently accessed from a location we're not familiar with. For your protection, please review your recent activity to make sure no one is using your Facebook account without permission.

Reviewing your activity takes just a few moments. We'll start by asking you a couple of questions to confirm that this is your account. (If we recognize your computer, you'll be able to skip this step.)

Fig. 6.1. Unfamiliar access lock notification

Within less than four days, we permanently lost 12 accounts, while an additional five accounts were temporarily blocked. Despite the unfortunate circumstances, this incident provided valuable insights into Facebook's account blocking procedures, thanks to the detailed logs we maintained. The account blocking process on Facebook typically occurs in three stages:

- Flagging: Accounts are marked with a flag, and their performance per hour is reduced by 50-75% (the token is also updated). However, access to the account remains unaffected during this stage.
- Warning Pop-up: When accessing the account, a new pop-up notification is displayed, alerting the user that their account may have been used by someone suspicious. (See Figure 6.1)
- Deactivation: Facebook deactivates the account and requires confirmation through one of its verification methods, such as phone verification or identity/picture recognition.

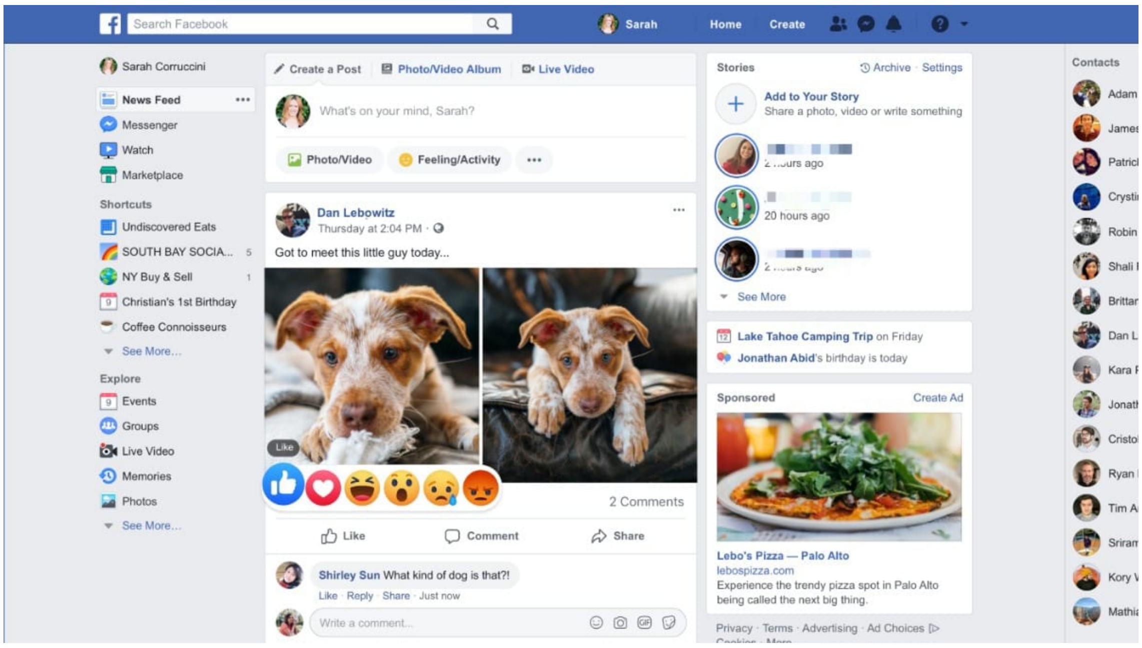
In light of this incident, we made the decision to start anew with a fresh network to avoid the risk of using the potentially compromised remaining accounts within the new system. This was essential to ensure the continued development of our automated system while minimizing dragging any potential issues from this unfortunate event.

Overall, this experience served as a valuable lesson, highlighting the importance of maintaining a robust and up-to-date VPN service to ensure the smooth operation of our automated system. Additionally, the incident provided us with valuable insights into Facebook's account blocking procedures, enabling us to refine our approach and enhance the system's security measures.

6.2. Facebook update and CSS tracking problem

During the last quarter of 2020, Facebook underwent a significant update, introducing a new style for its pages (See Figure 6.3 versus the classic style on Figure 6.2). This update was gradually rolled out to user accounts throughout the year, with some accounts receiving the new version earlier than others. As a result of this update, our automated system faced challenges as it heavily relied on CSS identification to locate and interact with elements on the Facebook pages.

With the introduction of the new style, the CSS identifiers for various components and elements changed, making our existing script ineffective on accounts with the updated pages. To ensure the continued functionality of our system, we needed to adapt and modify our script to work with the new CSS identifiers.



Got to meet this little guy today...

What kind of dog is that?!

Experience the trendy pizza spot in Palo Alto being called the next big thing.

Fig. 6.2. Facebook classic/legacy style from [33]

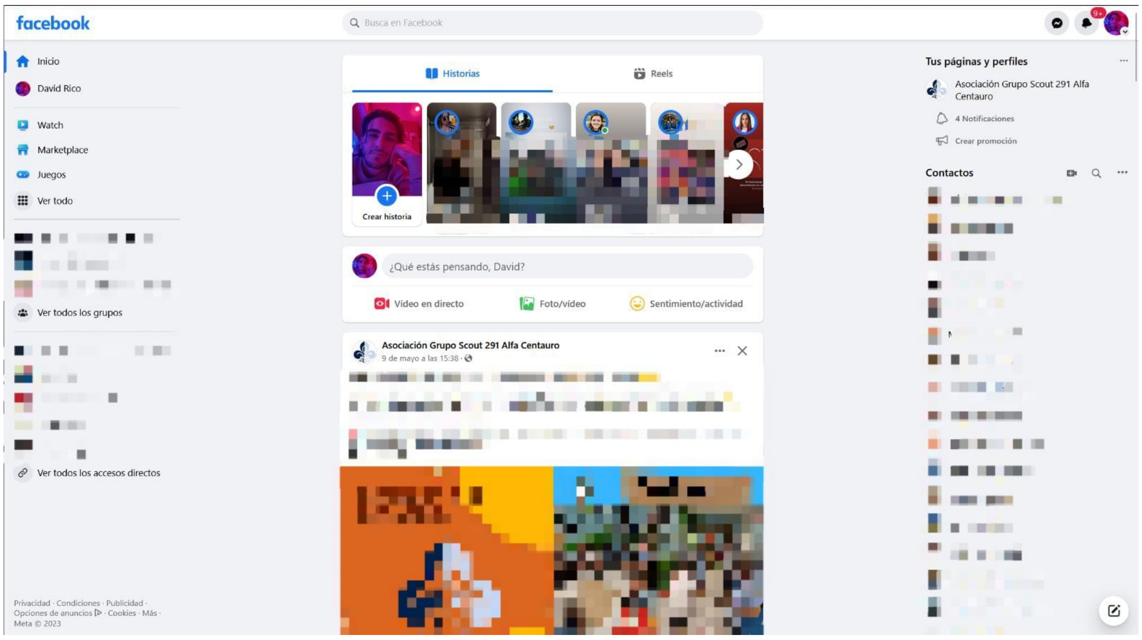


Fig. 6.3. Facebook Updated View and style

The process of fixing this issue involved meticulously retracing each component and element within the new version of Facebook's pages. We had to identify the updated CSS identifiers and incorporate them into our script accordingly. This allowed our system to accurately locate and interact with the elements on the new pages.

Furthermore, it was important to consider that not all accounts received the updated version of Facebook simultaneously. The roll-out was gradual and depended on factors such as location and other variables. To address this variation, we implemented a dynamic identification system. By analyzing the CSS IDs of both the classic and new login pages, our system could determine which version of Facebook was being used for each account every time we try to access. Based on this information, the appropriate script version would be selected right from the landing page, ensuring compatibility and functionality regardless of the version being utilized.

By adapting to the Facebook update and implementing dynamic CSS tracking, we were able to overcome the challenges posed by the changes in the platform's style. This allowed our automated system to continue operating seamlessly across both the old and new versions of Facebook, ensuring consistent performance and functionality for our users.

7. FUTURE WORK

The development of any software system is a dynamic process, and there are often opportunities for improvement even after the project is completed. In this chapter, I will discuss some possible improvements that I have identified for the automated system described in the previous chapters.

Some of these improvements were identified during the development process but could not be implemented due to time constraints, while others have emerged in the years since the project was completed, as new technologies and use cases have emerged. By exploring these potential avenues for improvement, we can gain a deeper understanding of the possibilities and limitations of automated systems, as well as identify opportunities for future research and development.

7.1. Auto-add friends (discontinued)

During the development of our automated system, we explored various features to enhance its functionality and mimic human-like interactions on Facebook. One such feature we initially started building was the "Auto-add Friends" tool. The aim of this tool was to automate the process of accepting friend requests and, in future stages, actively searching for new friends. However, we decided to discontinue the development of this feature and redirect our focus to more relevant sections that had a direct impact on the performance of the automated system.

The decision to discontinue the development of the "Auto-add Friends" feature was driven by two primary factors. Firstly, ethical concerns emerged as we delved deeper into the implications of automating friend requests. We realized that this feature had the potential to generate a significant number of new friend connections, which raised questions about the boundaries of online social interactions and the potential unintended consequences. To ensure responsible usage, we paused the development of this feature to thoroughly evaluate and address these ethical concerns. The second factor was the limited human resources to support the development.

In its initial implementation, the "Auto-add Friends" feature successfully accepted friend requests and expanded the bot's friends community as intended. However, we observed an unexpected phenomenon during our testing phase. Facebook's algorithms seemed to push our automated system towards larger communities of real human users, even though it was not actively publishing in those languages or targeting specific groups. We think it is probably related to a location based recommendation/ interest based feature. For instance, our user named “Savoir Marie”, which accessed Facebook through a Paris VPN, became deeply integrated into a local Arabic community and started receiving numerous direct messages from active users within that community.

While the ability to add new users to the friends network was considered crucial for our automated system to appear as part of our task to make our system appear to be more human, it became evident that a balance needed to be struck to prevent any harm or disruption to real user communities. To address this concern, we propose a cautious approach that involves regulating and limiting new friend connections. A solution to this would be to filter and only establish new friendships among the existing members of the automated system. However, this approach carries its own risks, as the detection of one account could raise suspicions about the authenticity of all accounts within the network.

A possible patch to the proposed solution would be that, as discussed in Section 5, it is critical of having a few real human accounts to support the automated system. These human accounts could help maintain a level of credibility and trustworthiness, while also mitigating the risks associated with relying solely on automated accounts. By carefully managing the friend network and implementing measures to regulate new friend requests, we could strike a balance between simulating human-like interactions and respecting the integrity of real user communities.

Although the “Auto-add Friends” feature was discontinued in the current implementation of our automated system, it serves as a valuable lesson in considering the ethical implications and potential consequences of automated actions on social media platforms. Our decision to prioritize user safety, ethical considerations, and the preservation of authentic online communities led us to set aside this feature until a more comprehensive and responsible approach can be developed. Anyway, we believe in its potential if well and meticulously developed as a big improvement for the system.

7.2. LL for generating text content

Large Language Models (LLMs) are a type of language model that use deep learning techniques to learn from massive amounts of text data. They can generate human-like text and perform various natural language processing tasks, such as translation, summarization, question answering, and more. In the last four years, LLMs have made significant advances in terms of size, accuracy, and diversity of domains. These models can now generate high-quality text content for various purposes and audiences, making them a valuable tool for automated systems. In this subsection, we will discuss the advantages and disadvantages of using LLMs for creating the text content our users will publish.

Language models such as LSTMs (long short-term memory networks), GRUs (gated recurrent units) and BERT (Bidirectional Encoder Representations from Transformers) have been used for years to generate text content. However, it wasn’t until the advent of large-scale pretraining techniques such as GPT (Generative Pre-training Transformer) that LLMs (large language models) really became a game changer in the field of natural language processing (NLP).

In the last four years, LLMs have made significant progress in generating text content that is virtually indistinguishable from human-written text. GPT-2, for example, is capable of generating news articles, poems, and even short stories that are often mistaken for being written by a human. Later versions, such as GPT-3, GPT 3.5 and GPT 4, has shown even more impressive capabilities, including the ability to translate languages, answer questions, and even generate computer code, even expanding their base knowledge being able to search the internet through beta implemented plugins.

Now, with the advancements made in LLMs and their ability to generate highly sophisticated and realistic text, it is possible to use them to generate text content for our automated system. By doing so, we can significantly increase the human-like behavior of our users, making them even more difficult to distinguish from real human users.

I have evaluated possible advantages and drawbacks, that can be summarized as follows:

- Advantages
  - LLMs can generate text in multiple languages, increasing the reach and versatility of our users. This will be extremely helpful regarding our limited reach to generate/search specific content for the different users, and align that content to their geographic location, matching not only language, but also other contextual knowledge.
  - LLMs can generate text based on simple prompts, simplifying the interaction and control of our users. By implementing these models, we will be able to give simple prompts in order to obtain complex text outputs. We could even generate multiple outputs at once, and post them over the time. The last step would be to completely automate the system by having an automated prompt generation that will later feed the models.
  - LLMs can generate coherent, relevant, and engaging text, improving the realism and appeal of our users. Regarding last trends on LLMs, using an internet-connected model will allow our automated user to discuss and comment recent events or real-time news, which will add a completely new layer of credibility.
- Disadvantages
  - LLMs require a lot of computational resources and data to train and run, limiting the scalability and accessibility of our users. In order to meet the computational requirements for running LLMs, we currently rely on third-party services such as computational cloud providers to run our own models or token systems like OpenAI's GPT prompt service.
  - LLMs can generate biased, inaccurate, or harmful text, posing ethical and social risks for our automated system and their users. Ethical concerns will be discussed later in section 10.2. But since these models are trained on a

large language basis, we must be careful and establish limits in order to avoid breaking the platform rules.

- LLMs can generate unpredictable or inconsistent text, affecting the reliability and quality of our automated system. This point could look like it is incoherent with the advantages mentioned, but since any text will be generated from the prompt given, we must be extremely careful and consistent while writing or generating those. This may require a certain level of expertise in using the technology, which may take some time to acquire, especially since the technology is still evolving. It is important to keep in mind that the quality of the generated text heavily relies on the quality of the prompts given.

In conclusion, after considering the advantages and disadvantages mentioned, it can be concluded that with careful and consistent development and implementation, LLMs would have a positive impact on our automated system by providing credibility and consistency to our content while simplifying the data collection process through automation.

7.3. Generative Neural Networks for images creation

This section shares some similarities with the previous ones, but also some conceptual differences due to the completely different nature of the models.

Generative Neural Networks (GNNs) are a type of deep learning model that can generate new data by learning from existing data. In recent years, GNNs have been used to generate realistic images that are often indistinguishable from images created by human artists. These models can be trained on large datasets of images and then generate new images that have similar styles, features, and textures. In this section, we will explore the advantages and disadvantages of using GNNs for image creation in our automated system.

One of the most popular types of GNNs for image creation is the Generative Adversarial Network (GAN). GANs consist of two neural networks, a generator and a discriminator, that compete against each other during training. The generator learns to create images that are similar to the training dataset, while the discriminator learns to distinguish between real and generated images. Over time, the generator becomes better at creating realistic images that fool the discriminator.

Using GNNs for image creation in our automated system can have several advantages. For our concrete use case, GNNs could generate high-quality images quickly and efficiently, and we won't need to rely on scrapping images from the internet, as we did. By training a GAN on a large dataset of images, we can generate new images in real-time, which is essential for our automated system to be able to respond quickly to user input.

Also, since we can train the images with personalized sets, GNNs could generate images that are tailored for specific "users". By training the GAN on certain "automated user preferences", we can generate images that match the user's preferred style, geographical location, or just to match the text on the post.

The potential drawbacks are quite similar to the ones from LLMs on previous section, such as an even bigger computational capability (even though LLMs are larger and more expensive to train, they are often less computationally expensive to deploy). We will also need to deal with biased or inappropriate images problem, for the same reasons as previously stated.

In conclusion, GNNs are a powerful tool for generating high-quality images that can be personalized and restriction-less. The use of GANs in particular allows for the creation of realistic images that can be generated quickly and efficiently. However, as with LLMs, there are potential drawbacks such as the need for significant computational power and the risk of biased or inappropriate images. These factors must be carefully considered when incorporating GNNs into an automated system.

8. COSTS

In this chapter, we will conduct an economic analysis of the automated system, taking into account various cost factors and resource requirements. By analyzing the costs associated with human resources, materials, and assets, we can gain insights into the financial implications of implementing and maintaining the system.

8.1. Estimation

During this section, we will make an estimation of the real cost of developing this project. The overall costs calculated would be €15000. A detailed analysis can be found in the following sections.

8.1.1. Human Resources cost

The first aspect we will consider is the cost of human resources involved in the development and operation of the automated system. The table below presents the estimated time spent and the corresponding salaries for the parties involved.

Employee	**Time Spend (h)**	**Salary**	**Total**
David Rico	960	11.25€/h	10800€
Ángel Cuevas	100	15€/h	1500€
			12300€

TABLE 8.1. HUMAN RESOURCES EXPENSES

It is important to note that these costs are specific to the development and initial operation phase of the automated system.

8.1.2. Materials and assets

In addition to human resources, the economic analysis also considers the costs associated with materials and assets required for the implementation of the automated system. This includes hardware, software licenses, cloud computing services, and any other tangible or intangible resources necessary for the system's functioning.

Employee	**Cost**	**Depreciation**	**Total**
Mi pc	900€	12 months	850€
Server	2400€	12 months	2350€
			3250€

TABLE 8.2. MATERIALS EXPENSES

In this concrete project, the main assets were my personal device and the server we used to run our script.

By considering both human resources and materials/assets costs, we can gain a comprehensive understanding of the economic implications of implementing and operating the automated system.

9. PLANNING

To illustrate the various tasks involved in this project and their corresponding time requirements, a comprehensive Gantt diagram has been created (see Figure 9.1). The official commencement of this thesis project was on November 1st, and the project exited its testing and evaluation stage in October next year.

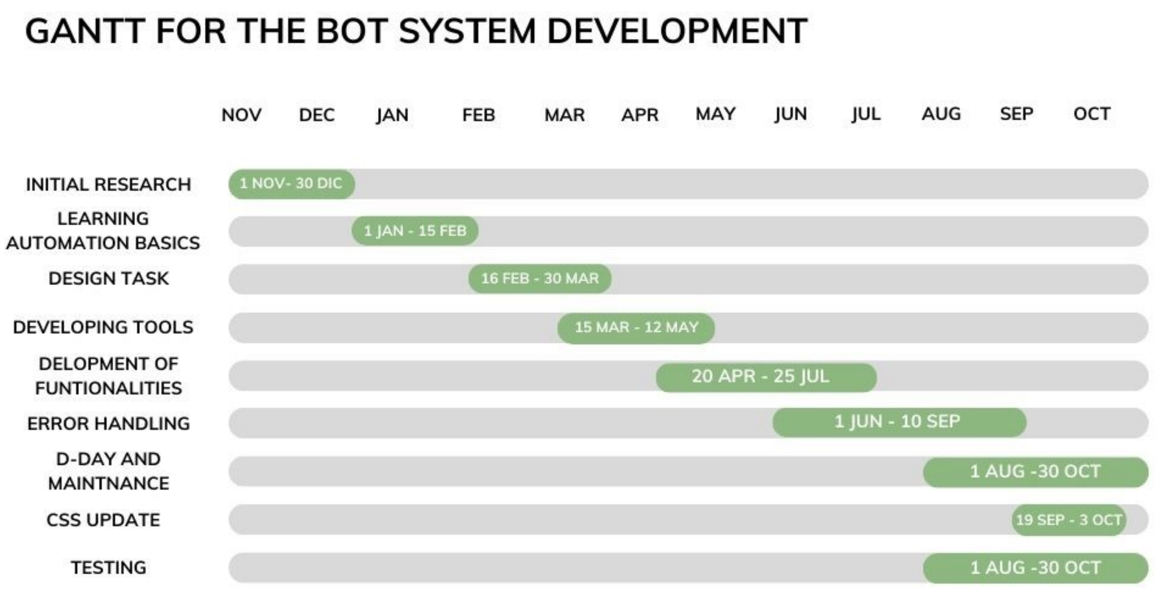


Fig. 9.1. Gantt diagram with the phases of the project

The presented timeline accurately represents the year-long development process that occurred between 2020 and 2021. It is important to note that the provided dates are approximate, as the initial plan had to be adjusted and modified to accommodate the specific requirements and demands of each project stage.

10. SOCIO-ECONOMIC ANALYSIS

In this chapter, we will explore the social analysis and ethical concerns associated with the development and deployment of an automated system for Facebook interaction. As technology continues to advance, it is essential to critically examine the potential social implications and ethical considerations that arise from the integration of automated system into social media platforms like Facebook. We will not only take into account concerns, but also the benefits this tools can provide.

10.1. Social Analysis

Social analysis involves examining the impact of technology on society and understanding how it influences human behavior, relationships, and communication dynamics. In the context of an automated system for Facebook interaction, social analysis plays a crucial role in evaluating both the positive and negative effects it may have on individuals and communities, as well as its potential to enhance the research process and shed light on the workings of social media platforms.

The development of an automated system for Facebook interaction has the potential to significantly impact the research landscape, particularly in the field of social media analysis. Here are some key points to consider:

1. Enhanced Research Efficiency: The use of automated systems can streamline the research process by automating data collection, content analysis, and information retrieval. This will allow the researchers to focus more on data analysis and interpretation.

2. Improved Data Accuracy and Consistency: Automated systems can ensure consistent and standardized data collection by following predefined protocols and methodologies. This reduces the potential for human error or bias, resulting in more reliable research outcomes. Also, due to the lack of natural bias, will allow researchers to introduce, when needed, an artificial bias to, for example, see how platforms filters, ban or push certain contents in an easy, automated and massive way. Researchers could gain deeper insights into social media phenomena.

3. Access to Hidden Algorithms: Automated systems provide researchers with a unique opportunity to study the algorithms and inner workings of social media platforms. By interacting with the platform through automated systems, researchers can gain insights into the recommendation systems, content prioritization, and targeted advertising mechanisms, tools that are kept in secret from many companies, and can represent a huge threat when misused or incorrectly development or implemented.

This knowledge can contribute to not only a better understanding of how social media platforms shape user experiences and influence online behavior, but also detect and prevent algorithms to advertise trends that can endanger users.

4. Study of User Engagement and Reactions: Automated systems can help researchers explore user engagement patterns, sentiments, and reactions to various types of content on the desired platform. By collecting and analyzing data on likes, shares, comments, and interactions, researchers can gain valuable insights into user preferences, social dynamics, and the impact of content on audience engagement. Obtaining valuable data on user preferences to perform social analysis was the reason this tool was developed after (in our particular case, the data was not obtained from interaction, but from Facebook ads service), and proved to be useful.

In addition to the research benefits, the wider public can also benefit from the insights gained through the analysis of social media. By understanding the algorithms and mechanisms behind these platforms, individuals can make informed decisions about their social media usage and be more aware of the potential influence and biases embedded within the system.

In today's society, where social media plays a key role in shaping relationships and interactions worldwide, specially in younger generations, we must take the initiative to study and understand the underlying algorithms that govern social media platforms. By doing so, we can gain insights into their mechanisms and evaluate their potential effects on individuals and communities, and not just trust developers and companies to make a fair use with them.

10.2. Ethical Concerns

Ethical concerns arise from the potential implications and consequences of deploying automated systems, and given that Facebook is one of the platforms more used worldwide, we can assume that the extent of these concerns is critical. Examining the ethical dimensions helps identify and address the ethical challenges associated with their use, ensuring a responsible implementation.

- User Data Protection
  - The algorithm used in the automated system has the capability to retrieve large amounts of data from the platform, surpassing what a human user can access. This raises concerns about user data privacy and protection.
  - Due to the fact vast quantities of data can be retrieved, including potentially sensitive information if published online, it poses a risk if not handled and stored securely.

  - The tool allows obtaining healthy tokens for many "users", and these results into the capacity of querying many data from internal platforms tools like advertisement data, which can be misused for targeted marketing or other purposes that may infringe on user privacy.

- Ethical Interactions with Humans
  - As the network of users within the system appears highly human-like and interacts with other users, it becomes crucial to consider the ethical implications of these interactions.
  - Users may assume they are engaging with other human users and form expectations accordingly. Failing to disclose the automated nature of the users can lead to misleading and potentially harmful interactions. Ensuring neutrality and minimizing harm in interactions should be a priority.

- Influence on User Opinions
  - The extensive reach and persuasive capabilities (due to the fact it could appear many people share a same idea or opinion, and this could result in social pressure for a user) of an automated system raise concerns about its potential to manipulate or influence user opinions.
  - Using the network of automated users to spread specific political views or biased/fake information can lead to the distortion of public discourse, the creation and raise of social movements around fake data or undermine a democratic process.

- Introduction of Bias
  - As mentioned in section 10.1, bias can intentionally be introduced into the automated system for research purpose, and due to the nature of social media and connected sites, this will impact not only the research conducted but also the experiences of users on the platform.
  - of course, unintentional or malicious biased automated user networks may perpetuate existing societal biases, discriminate against certain individuals or groups, and reinforce systemic inequalities. This concern is not solely related to the nature of the tool itself, but also to human behavior. While any misused tool can potentially be harmful, the impact becomes more significant as the tool's reach expands.

The social analysis and ethical concerns surrounding the development and implementation of an automated system highlight the importance of responsible and conscientious deployment, as well as the understanding of the reach and threats it could imply. By considering the social implications and addressing ethical challenges, we can ensure that the automated system contributes positively to the social media ecosystem while safeguarding user privacy, promoting transparency, and upholding ethical standards.

Ensuring 100% safety is challenging when developing a tool that, when used responsibly and ethically, can contribute to a direct increase in social knowledge and safety by gaining a deeper understanding of not only social media dynamics, but also our own society and tendencies. However, it is crucial to recognize that if the tool is misused or not properly managed, it can become a potential danger. Thus, proactive measures must be taken to mitigate any potential risks or negative consequences that may arise from its use.

11. CONCLUSION

The main objective of this thesis was to develop a successful automated tool that facilitates research by providing investigators worldwide with access to proven significant data, regardless of their background. This objective was achieved through a comprehensive year-long effort focused on the development, debugging, testing, and overcoming various challenges associated with our automated system.

The journey involved tackling critical aspects such as evasion of detection mechanisms, enhancing robustness, and ensuring reliability through the implementation of intelligent solutions within the designed system. By addressing these challenges, we aimed to establish a solid foundation that would enable the construction of smarter and more advanced technical solutions.

It is essential to recognize that this research serves as a starting point, laying the groundwork for future advancements in the field. Our intention is to foster continuous progress and adaptability by leveraging these initial findings and technical infrastructure. As the landscape of social media continues to evolve, our goal is to create responsive systems that can evolve alongside us, accommodating the changing nature of our interactions with these platforms.

By providing researchers with an efficient and accessible tool, we aim to empower the academic community and contribute to the advancement of knowledge in various domains. We believe that through the application of technological solutions and the utilization of proven data, researchers will be able to conduct their investigations more effectively, regardless of their individual backgrounds.

In summary, this thesis represents a significant step forward in the pursuit of facilitating research and leveraging technology to overcome barriers in data access. The foundation we have established will serve as a springboard for future developments, ensuring that our tools remain at the forefront of the evolving research landscape.

BIBLIOGRAPHY

[1] *Marketing api - documentation - facebook for developers*, Accessed: 2023-05-05, 2022. [Online]. Available: https://developers.facebook.com/docs/marketing-api/.

[2] M. Facebook, *Meta reports first quarter 2023 results*. [Online]. Available: https://investor.fb.com/investor-news/press-release-details/2023/Meta-Reports-First-Quarter-2023-Results/default.aspx.

[3] *About facebook ads*, Accessed: 2023-05-05, 2022. [Online]. Available: https://www.facebook.com/business/ads.

[4] F. META, *Facebook mau worldwide 2022*, Apr. 2023. [Online]. Available: https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/.

[5] P. Gupta, *Do you know facebook generates 4 petabytes of data per day*. [Online]. Available: https://www.linkedin.com/pulse/do-you-know-facebook-generates-4-petabytes-data-per-day-pandav-gupta.

[6] A. Joy, "Web automation with python: A beginner's guide," *Pythonista Planet*, Dec. 2020, Accessed: 2023-05-05. [Online]. Available: https://pythonistaplanet.com/web-automation-with-python/.

[7] Workato, "Python automation: 9 scripts to automate critical workflows," *Workato*, Nov. 2020, Accessed: 2023-05-05. [Online]. Available: https://www.workato.com/blog/python-automation-scripts/.

[8] *Selenium - documentation*. [Online]. Available: https://www.selenium.dev/documentation/overview/.

[9] T. Araújo, M. Fatehkia, D. Garcia, and B. Stewart, "Using facebook advertising data to predict obesity and diabetes prevalence," *BMC public health*, vol. 17, no. 1, p. 119, 2017.

[10] M. Fatehkia, T. Araújo, D. Garcia, and B. Stewart, "Measuring digital gender inequalities using facebook advertising data," *Social Science Computer Review*, vol. 36, no. 2, pp. 244–259, 2018.

[11] M. Fatehkia, T. Araújo, D. Garcia, B. Stewart, and A. Palotti, "Mapping urban socioeconomic inequalities in developing countries through facebook advertising data," *Frontiers in data science*, vol. 7, p. 63 352, 2020.

[12] A. Palotti, T. Araújo, M. Fatehkia, D. Garcia, and B. Stewart, "Using facebook advertising data to track mass migration," *Social Science Computer Review*, vol. 38, no. 1, pp. 103–116, 2020.

[13] F. Rampazzo, A. De Stefano, and M. Fortunato, "Measuring the impact of migration on local economies using facebook advertising data," *Social Science Computer Review*, vol. 39, no. 4, pp. 577–594, 2021.

[14] J. Vieira, T. Araújo, M. Fatehkia, D. Garcia, and B. Stewart, "Using facebook advertising data to describe the socio-economic situation of syrian refugees in lebanon," *Frontiers in data science*, vol. 9, p. 3530, 2022.

[15] H. Allcott and M. Gentzkow, "Social media and fake news in the 2016 election," *Journal of Economic Perspectives*, vol. 31, no. 2, pp. 211–236, 2017.

[16] D. Garcia *et al.*, "Analyzing gender inequality through large-scale facebook advertising data," *Proceedings of the National Academy of Sciences*, vol. 115, no. 27, pp. 6958–6963, 2018. DOI: 10.1073/pnas.1717781115.

[17] A. Mehrjoo, R. Cuevas, and Á. Cuevas, "A new methodology to measure faultlines at scale leveraging digital traces," *EPJ Data Science*, vol. 11, no. 1, 2022. DOI: 10.1140/epjds/s13688-022-00350-w.

[18] N. Obradovich *et al.*, *Expanding the measurement of culture with a sample of two billion humans*, 2020. DOI: 10.3386/w27827.

[19] J. Harfield, "What is a web crawler/spider and how does it work?" *MUO*, 2021. [Online]. Available: https://www.makeuseof.com/what-is-a-web-crawlerspider-and-how-does-it-work/.

[20] *Python web scraping tutorials – real python*, Web page. [Online]. Available: https://realpython.com/tutorials/web-scraping/.

[21] *Github topics, web-scraper*, Web page. [Online]. Available: https://github.com/topics/web-scraper.

[22] K. Rungta, *What is selenium? introduction to selenium automation testing*, May 2023. [Online]. Available: https://www.guru99.com/introduction-to-selenium.html.

[23] *Pros and cons of selenium as an automation testing tool*, May 2023. [Online]. Available: https://testsigma.com/blog/selenium-automation-testing-pros-cons/.

[24] D. Rico, "Design and development of a scalable web scraper for large-scale data collection," Ph.D. dissertation, 2023.

[25] *Facebook help center: Facebook*, 2019. [Online]. Available: http://web.archive.org/web/20190731153051/https://www.facebook.com/help/.

[26] H. Zhu, E. Chen, and X. Xie, "User identity linkage across online social networks: A review," *ACM Transactions on Internet Technology (TOIT)*, vol. 14, no. 3, p. 34, 2014.

[27] M. Duggan, N. B. Ellison, C. Lampe, A. Lenhart, and M. Madden, "Mobile messaging and social media 2015," *Pew Research Center*, 2015.

[28] R. Gómez and J. Pavón, “Introducing social randomization to reduce the detection of autonomous bot-based accounts in online social networks,” *Sensors*, vol. 19, no. 7, p. 1554, 2019.

[29] T. Barnes and S. Srinivasan, “Proxy use and the risk of detection in bot-based online social systems,” *Information Systems Research*, vol. 29, no. 3, pp. 599–616, 2018.

[30] J. Wright, E. De Cristofaro, and G. Stringhini, “Using proxies and randomized accounts to evade twitter’s bot detection,” pp. 2949–2955, 2019.

[31] U. Mahmood, *Random imgur*, https://github.com/umahmood/random-imgur, 2020.

[32] P. S. Foundation, *random — Generate pseudo-random numbers*, https://docs.python.org/3/library/random.html, Accessed: 2023-05-05, 2021.

[33] S. Staff, *“facebook classic” will close in september, facebook confirms*, Aug. 2020. [Online]. Available: https://rakitaplikasi.com/en/blog/facebook-classic-will-close-in-september-facebook-confirms.