Eötvös Loránd University

Faculty of Informatics

Department of Software Technology

Mastodon social media threat alert extension

Supervisor:
Zoltán Gera
tanársegéd,programozó matematikus MSc

Author:
Kajdomcaj Dion
Computer Science BSc

Budapest, 2022

EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

Thesis Registration Form

Student's Data:

Student's Name: Kajdomcaj Dion

Student's Neptun code: E9TZ67

Course Data:

Student's Major: Computer Science BSc

I have an internal supervisor

Internal Supervisor's Name: *Gera Zoltán*

Supervisor's Home Institution: *ELTE Faculty of Informatics*

Address of Supervisor's Home Institution: *1117 Pázmány Péter stny 1/C*

Supervisor's Position and Degree: *egyetemi tanársegéd, programozó matematikus MSc*

Thesis Title: Mastodon Social Media Threat Alert Extension

Topic of the Thesis:

(Upon consulting with your supervisor, give a 150-300-word-long synopsis os your planned thesis.)

Social media thread alert extension is an extension which is planned to improve the security of Mastodon social media platform by letting the users know when there is a possibility for a threat towards them in this social media. This extension will provide social media threat alert for the users, for example when an account texts you it will detect and let you know whether that account has a possibility of being a fake account and afterwards will allow you to decide whether you want to be still connected with that account or not. Besides this it will prevent these kind of users getting the attention they seek in this social media and by that it will start to get them off the network since they will not get the attention they seek for in it, other than this it will encourage students and professors to use their university social media even more than they actually do.

Budapest, 2021. 11. 30.

Contents

Chapter 1

Introduction

1.1 Motivation

Nowadays we all know someone who has been affected negatively by the bullying happening on social media, which is becoming a very common problem in our society. Sometimes those cases escalated to the point where it turned into a serious mental health problem for the ones who have been affected. I was always wondering for a way to help people prevent similar cases and feel safe while surfing the social media feed. Everyone has it's rights to feel safe while doing a certain activity, but bringing safety to the social media in these kind of cases has been neglected throughout these last years.
The main reasons mentioned before have pushed me to brainstorm an idea to help people feel safe on Mastodon, which is open source, and enjoy the beauty of connecting with real profiles and use the social media to the point where it changes their life for good.

1.2 Thesis structure

This thesis consists of 4 chapters, which help users understand the installation steps of the software,the correct way of using the software and the possible behavior that can lead to errors, as well as help developers understand the functionality behind the software, the software architecture and the testing made on the software. Besides those main chapters it also contains a bibliography, a list of figures, a list of codes and a list of tables.

Chapter 2

User documentation

In this chapter we will discuss the installation steps, the correct way of using the software and a brief description about the software.

2.1 Project Description

My project is a desktop application that is meant to run on the background while using Mastodon social media. The whole project was built using the latest *Python* 3 version. The main goal of the project is to detect possible threats coming from other accounts in form of direct messages and tags, after warning the user about possible threats it let's the user decide the kind of action he wants to take against the account that may be a threat and the domain where the account came from.

As a prerequisite for using this desktop application is a stable internet connection and a Mastodon account.

2.2 Installation guide

As earlier mentioned, in order to use Mastodon social media threat alert application we need to have a stable internet connection and a Mastodon account in any server and *Python* 3.10.2.

The application is currently supporting Windows and Linux but the goal is to extend it as a mobile application which supports IOS and Android. Hence, the installation steps are the same for both, Windows and Linux, but we will go through the steps in Windows specifically.

To download the application we need to clone the following repository: `https://github.com/DionKajdomcaj/Mastodon-Social-Threat-Alert.git`.

Prior to cloning the repository we need to make sure that we have git. If git is missing, the user can dowload it at the following url: `https://git-scm.com/download/win` If the git prerequisite is met, we can clone the repository by entering the following command in the command prompt:

```
C:\Users\dionk>git clone https://github.com/DionKajdomcaj/
    Mastodon-Social-Threat-Alert.git
```

Code 2.1: Cloning Repository

After succeeding to clone the repository we need to install the requirements for our environment. In order to install the requirements we need to make sure we have pip command. If the pip command is not installed, the user can install it by clicking the following url: `https://phoenixnap.com/kb/install-pip-windows`.

In the application folder there is a file called requirements, installing it will download all the libraries needed to run the application.

We can install them by using the following command in command prompt:

```
C:\Users\dionk\Mastodon-Social-Threat-Alert>pip install -r
    requirements.txt
```

Code 2.2: Installing requirements

Now we are ready to run the application.

2.2.1 Running the application

In order to run the application we need to make sure that we are in the correct directory and then run the following command:

```
C:\Users\dionk\Mastodon-Social-Threat-Alert>python ThreatAlert.py
```

Code 2.3: Running the application

After running the command, if all the prerequisites are met, we can see our application log in page.

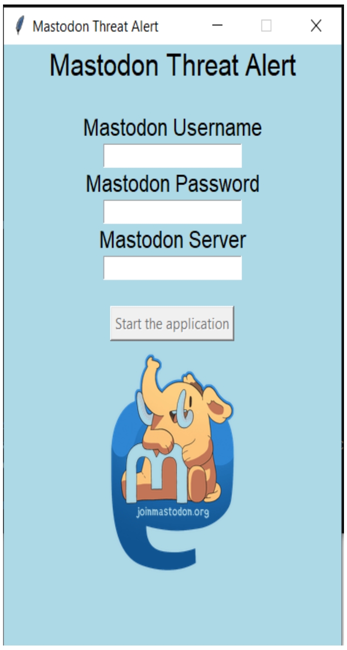


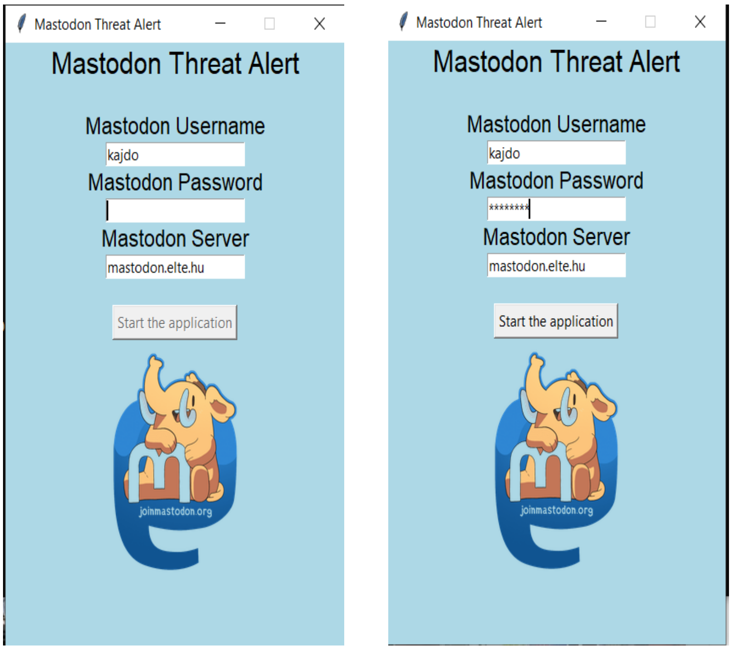
Figure 2.1: Application main page

2.3 Logging in

As we saw in Figure 2.1 the button to actually start the application is disabled. In order to enable it we must fill all the necessary fields which are:

- **Mastodon username**, which is your original Mastodon account username.
- **Mastodon password**, which is your original Mastodon account password, that you use to log in to Mastodon.
- **Mastodon server**, which is the server your account is currently registered.

If only one of them is missing then the user will not be able to start the application nor use it.



Mastodon Threat Alert

Mastodon Username: kajdo

Mastodon Password:

Mastodon Server: mastodon.elte.hu

Mastodon Threat Alert

Mastodon Username: kajdo

Mastodon Password: *******

Mastodon Server: mastodon.elte.hu

(a) Missing password field - Button disabled

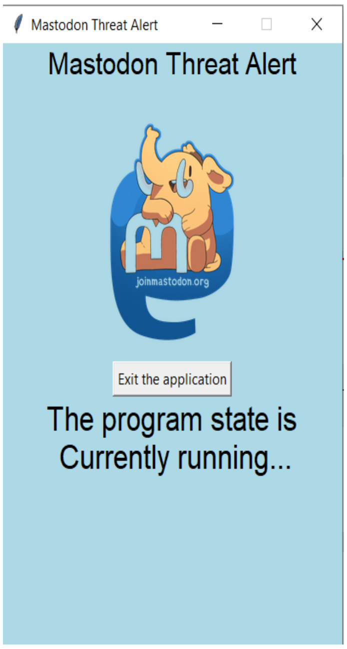
(b) No missing field - Button enabled

Figure 2.2: Button disabled and enabled

The fields must be filled with the Mastodon account information. In the username field you must enter your Mastodon username, and the same goes up to password. About the server you need to know which server are you in and only type the domain, for example in case there is an account like: example@mastodon.social then the username is example and the server is mastodon.social. All of the data are case sensitive, so you must give them exactly as they are originally.

2.3.1 Valid log in data

If every data is correct then the application will be connected to the Mastodon API after we click on the button. The program will let us now that it is running, and it will not change it's state until it recognizes a possible threat for the logged in user.



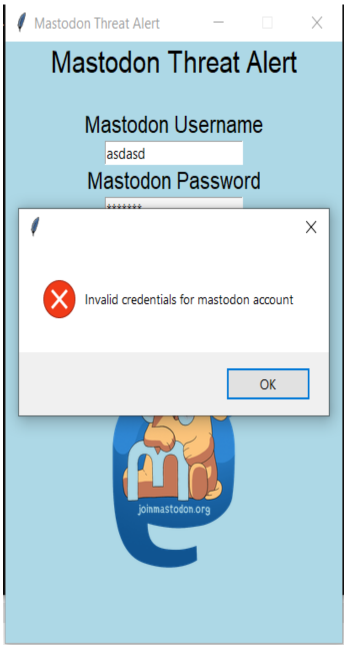
Mastodon Threat Alert

The program state is
Currently running...

Figure 2.3: Application running

2.3.2 Invalid log in data

Even if the button is enabled it does not mean that the log in data entered by the user are correct. So, if the user does not give the valid log in data then the application will not be connected to Mastodon API. Hence, it will give the user an error message. The following figure is going to show you the message you will receive for giving invalid log in data. After receiving the message you can just press the message button and try again as many times as you need.



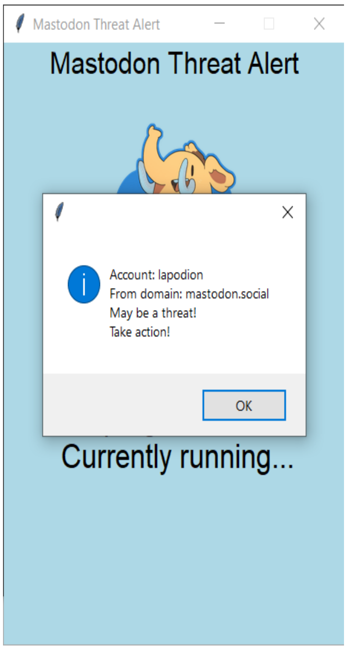
Invalid credentials for mastodon account

Figure 2.4: Incorrect log in data

2.4 Actions for the possible threat account

Now that we logged in successfully, we can start using Mastodon as usually, but this time we have the Mastodon threat alert application running on the background and looking for possible threats.

Every time that we are going to receive a direct message or a tag notification, that account's data is going to be checked whether it has a possibility to be a threat or not. After checking if the account, that was trying to reach you, is considered to be a possible threat, the application will show you a warning message containing the possible threat account's username and domain, and will ask you to take a certain action against the account.



Account: lapodion
From domain: mastodon.social
May be a threat!
Take action!

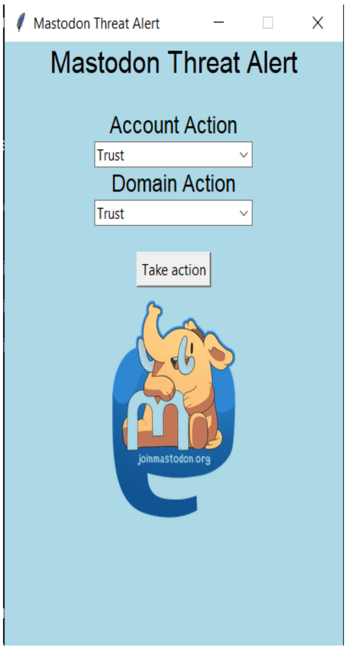
Figure 2.5: Possible threat notification

We have three kind of actions supported in our application depending whether we want to take them against an account or a domain. These actions are

- **Trust**
- **Block**
- **Mute**

The default value for both of them is Trust, which can be changed.

We have to simply choose the action from a combo box for both, possible threat account and it's domain, and click the button in order to perform the actions.

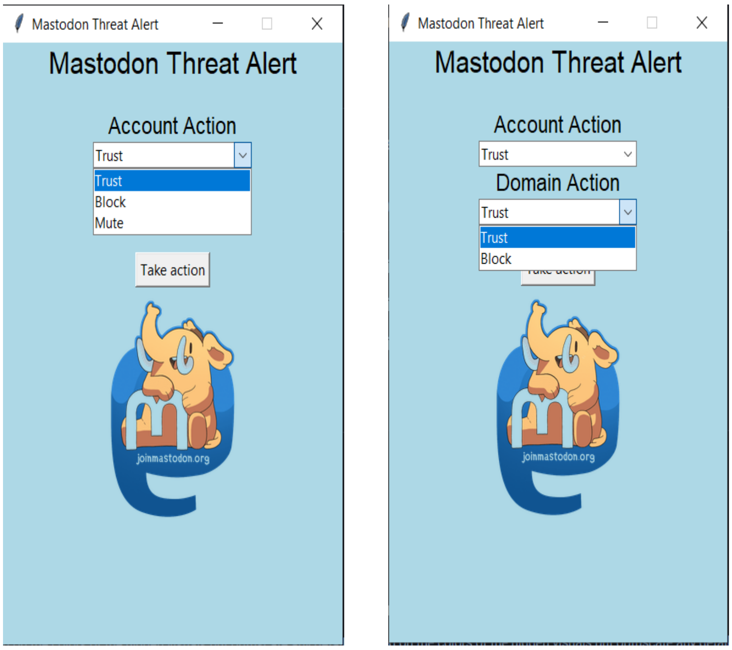


Account Action: Trust

Domain Action: Trust

Figure 2.6: Action window

Now we can choose independently the type of action for both, the account and it's domain.

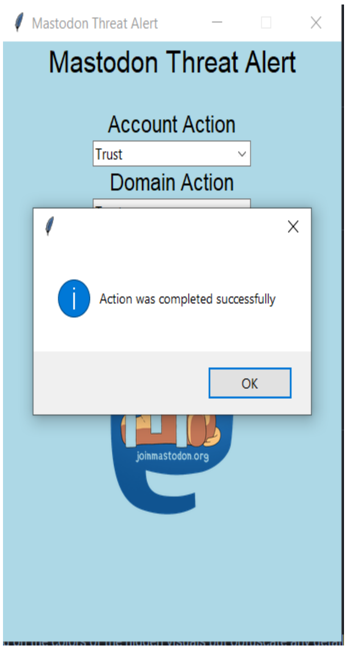


(a) Account Actions

(b) Domain Actions

Figure 2.7: Actions available for account and domain

When we click the button, a pop up window will show up, letting you know that the actions were successful.



Mastodon Threat Alert

Account Action

Trust

Domain Action

Action was completed successfully

Figure 2.8: Pop up window to confirm the actions

After closing the pop up window, the application will go back to it's running state as in Figure 2.3.

2.4.1 Account action

As it was mentioned before, we can choose the actions separately for the possible threat account and it's domain. In this section we are going to talk about the possible threat account's action.

We have 3 types of actions for the account, which are: Trust, Block, Mute. The following table will describe each action.

Account Actions	
Trust	This action is very simple in context. When the user takes this function it means that he is trusting the possible threat account, and the application will not check the same account again, if it is trying to reach the user again.

Block	This action blocks the possible threat account, meaning it does not let the account reach the user anymore, not only by direct messaging or tagging, but even following, liking or any other social media activity. Besides those things the user will not see any activity from the possible threat account. To sum it up, the possible threat account will be non existent for the user.
Mute	Muting an account is same as ignoring the account, since the user will not receive notifications from the possible threat account anytime it tries to reach the user, but the user will still be able to check it's activity and the possible threat account will be able to try and reach the user, but simply the user will not be notified.

Table 2.1: Description of every action that can be taken against an account

However, every action that has been taken against the possible threat account can be easily reverted in Mastodon.

2.4.2 Domain action

In case of the possible threat account's domain we have less actions that can be taken against it.

The actions are Trust and Block. They might seem the same as the account actions described in table 2.1, but blocking a domain is completely different, since here we are working with the whole domain. However, in case of Trust, the functionality is the same as it was in the possible threat account's action.

In the following table you can understand their functionality.

Domain Actions	
Trust	This action works the same way as it works to trust an account. However, trusting the domain will not take any actual action against the domain. So, simply the accounts that are from that domain can freely reach the user in any form and without any restrictions.
Block	In case of Block the whole domain is blocked, meaning that any account that belongs in that domain can't reach the user. But, the user should always be very careful and is recommended not to block domains unless he/she was disturbed many times by the accounts coming from the same domain.

Table 2.2: Description of every action that can be taken against a domain

Same as in the account actions, the domain actions can be reverted in Mastodon.

2.5 Exiting the application

As we can see in figure 2.3, we can exit the application in two ways, by clicking the button named *Exit the application* or by simply closing the window. It is recommended to exit the application by clicking the button because it is terminated safely, but sometimes when we want to exit it while we are not on the running page, we can exit the application by closing the window as well. However, it is not recommended to close the application while taking actions, like in figure 2.6, because the possible threat account's will not be saved in the database.

Chapter 3

Developer documentation

In this chapter, we are going to get into details about the software architecture, the solutions to certain problems encountered during implementation and how everything came up together in the end.

In order to build and run the application as a developer, you must follow the same steps as in 2.2.

As it was mentioned before the application is completely connected to Mastodon [1]. So, it is useless and not the correct way to use it without using Mastodon.

3.1 Mastodon API

As it was mentioned in 2.1, this application check whether an account that is trying to reach you is a possible threat or not. To do this checking we need the data of the account. So, we have to connect to a API that get the data from the live Mastodon server in order to check the possibility of a threat account. Other than that, the API is used to let us take the actions mentioned in 2.4 directly from our application, without the need of doing it through Mastodon. For this application we used an already created API for Python called *Mastodon API* [2].

Below we can see a scheme of how our this API operates.

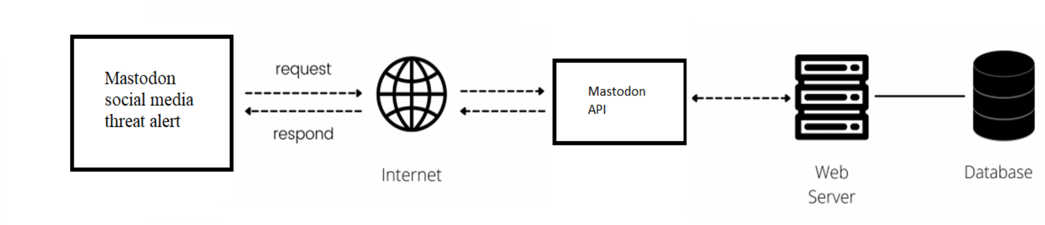


Figure 3.1: Mastodon API scheme

3.1.1 API rate limits

This API has a fixed rate-limit that is 300 requests per 5 minutes, in order not to violate the rate-limit and make the user wait for 5 minutes until the rate-limit resets, we needed to get data from the live server every 2.4 second so the user can get the full experience of Mastodon. This being said, user has enough time to surf the feed without having to wait for 5 minutes.

3.1.2 API initialization

To use this API we need to use the user's credentials to log into Mastodon account through API, and those credentials will be saved in *.secret* files in the directory where we call the API, and that is the reason why the user must give the credentials in the starting of the application. Below is a the snippet for logging in to the Mastodon account through API.

```
def __init__(self):
  self._user = Mastodon()
  self._userApiInstance = Mastodon()
  self._mastodonServer = ""

def createApp(self, mastodon_server):
  if(mastodon_server.split("://")[0] != "https"):
    self._mastodonServer = "https://" + mastodon_server
  else:
    self._mastodonServer = mastodon_server

Mastodon.create_app(
  "mastodonApiAppUser",
  api_base_url=self._mastodonServer,
  to_file='app/secretFolder/mastodonApiAppUser.secret'
)

def setUpAccounts(self):
  self._user = Mastodon(
    client_id='app/secretFolder/mastodonApiAppUser.secret',
    api_base_url=self._mastodonServer
  )

def loginAccount(self, username, password, user=True):
  if user:
    self._user.log_in(
      username,
      password,
      to_file='app/secretFolder/usercredentials.secret'
    )
```

Code 3.1: Logging in to Mastodon account through API

3.1.3 API instance

Logging in to the account is not enough in order to use this API, but we can say it is a very important step and a must. To actually start using the API we need to create an API instance for that account. Below you can see the snippet of creating the API instance for both user account or admin account.

```
def createApiInstance(self):
  self._userApiInstance = Mastodon(
      access_token='app/secretFolder/usercredentials.secret',
      api_base_url=self._mastodonServer
  )
```

Code 3.2: Creating API instance

So, we have to create a access token from our the user credentials saved in *.secret* file, and after creating the API instance we can start using the API methods. The main methods that this application uses are:

- **Get the notifications**
- **Clear the notifications**
- **Block an account**
- **Block a domain**
- **Get certain account's data**
- **Get user's following accounts**
- **Mute an account**

We will deep dive into each of the methods in the following sections.

3.1.4 Notifications

To get the accounts that are trying to reach the user, we first need to get the notifications and filter them to only get the direct messages and tags, which is done by setting the parameter named **mentions** to true. To get the notifications through the API we can use the following method:

```
def getNotifications(self):
  return self._userApiInstance.notifications(mentions_only=True)
```

Code 3.3: Getting the notifications by API

And in order to filter them we need to go through each and filter them by their type and existence in the applications database.

```
def startSession(self):
  try:
    notifications = self.api.getNotifications()
    accounts_reaching_user = []
    for notification in notifications:
        account_id = notification['account']['id']

        inDatabase = self.isAccountInDatabase(int(account_id))

        if (account_id not in accounts_reaching_user and not
           inDatabase):
          accounts_reaching_user.append(account_id)

      return accounts_reaching_user
    except Exception:
      return []
```

Code 3.4: Filtering the notifications

3.1.5 Getting account's data

As it was mentioned in 3.1 we need to get the account's data in order to check for the possibility of the threat. To get the data of a certain account, we need that account's id and we can get public data for that account such as:

- **Account ID**
- **Username**
- **Domain**
- **Followers Count**
- **Followings Count**
- **Statuses count**
- **Avatar**
- **Header**
- **Date of creation**

And in order to fetch those data we can simply call the API method which is as below:

```
getAccountData(self, account_id, admin=False):
  if not admin:
    return self._userApiInstance.account(account_id)
```

Code 3.5: Fetching certain account’s public data

This method will return the data as a dictionary where the keys are the public data names and the values are their values.

3.1.6 Getting user’s following accounts

We know that if we follow someone that means we most probably know him. Hence, we need to trust the accounts we follow when we start our application. To trust them we need to get the list of the accounts which is done by the following method:

```
def getFollowingAccounts(self):
  my_id = self._userApiInstance.me()[’id’]
  return self._userApiInstance.account_following(my_id)
```

Code 3.6: Method to get the list of the accounts we follow

As we can see, we first need to get our account id and then get the list of the accounts we follow. After having the list of the accounts we follow, we need to trust by inserting each in the application’s database so we don’t have to check them if they try to reach us.

```
def trustFollowings(self):
  following_accounts = self.api.getFollowingAccounts()
  for account_data in following_accounts:
    try:
      account_id = int(account_data[’id’])
      username = str(account_data[’username’])
      domain = str(account_data[’url’].split(’/’)[2])
      self.__database.insertAccount(account_id, username, domain,
          False)
      self.__database.insertDomain(domain, False)
    except Exception:
        print("Error")
```

Code 3.7: Inserting the accounts we follow immediately in the application database

3.1.7 Blocking an account

As it was mentioned in 2.4.1 we have the option to block an account and of course the only way to do it outside the application is by using the API. So, in order to block an account we need to call the API method and pass the account's id as a parameter and the account with that id will be blocked.

```
def blockAccount(self, account_id):
    self._userApiInstance.account_block(account_id)
    print("done")
    return True
```

Code 3.8: Blocking an account method

3.1.8 Muting an account

Same as blocking an account, when we want to mute a certain account we need to call the API method and pass the account's id as a parameter.

```
def muteAccount(self, account_id):
    self._userApiInstance.account_mute(account_id)
    return True
```

Code 3.9: Muting an account method

3.1.9 Blocking a domain

When it comes to blocking a domain the parameter changes, since the domains are not classified with ids. If we want to block a domain we need to pass the domain's name as a parameter, but keep in mind that the domain is case-sensitive.

```
def blockDomain(self, domain):
    self._userApiInstance.domain_block(domain)
    return True
```

Code 3.10: Blockin a domain method

3.2 Predictive Model

3.3 Database

3.4 GUI

3.5 Testing

Chapter 4

Conclusion

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In eu egestas mauris. Quisque nisl elit, varius in erat eu, dictum commodo lorem. Sed commodo libero et sem laoreet consectetur. Fusce ligula arcu, vestibulum et sodales vel, venenatis at velit. Aliquam erat volutpat. Proin condimentum accumsan velit id hendrerit. Cras egestas arcu quis felis placerat, ut sodales velit malesuada. Maecenas et turpis eu turpis placerat euismod. Maecenas a urna viverra, scelerisque nibh ut, malesuada ex.

Aliquam suscipit dignissim tempor. Praesent tortor libero, feugiat et tellus porttitor, malesuada eleifend felis. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nullam eleifend imperdiet lorem, sit amet imperdiet metus pellentesque vitae. Donec nec ligula urna. Aliquam bibendum tempor diam, sed lacinia eros dapibus id. Donec sed vehicula turpis. Aliquam hendrerit sed nulla vitae convallis. Etiam libero quam, pharetra ac est nec, sodales placerat augue. Praesent eu consequat purus.

Appendix A

Simulation results

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque facilisis in nibh auctor molestie. Donec porta tortor mauris. Cras in lacus in purus ultricies blandit. Proin dolor erat, pulvinar posuere orci ac, eleifend ultrices libero. Donec elementum et elit a ullamcorper. Nunc tincidunt, lorem et consectetur tincidunt, ante sapien scelerisque neque, eu bibendum felis augue non est. Maecenas nibh arcu, ultrices et libero id, egestas tempus mauris. Etiam iaculis dui nec augue venenatis, fermentum posuere justo congue. Nullam sit amet porttitor sem, at porttitor augue. Proin bibendum justo at ornare efficitur. Donec tempor turpis ligula, vitae viverra felis finibus eu. Curabitur sed libero ac urna condimentum gravida. Donec tincidunt neque sit amet neque luctus auctor vel eget tortor. Integer dignissim, urna ut lobortis volutpat, justo nunc convallis diam, sit amet vulputate erat eros eu velit. Mauris porttitor dictum ante, commodo facilisis ex suscipit sed.

Sed egestas dapibus nisl, vitae fringilla justo. Donec eget condimentum lectus, molestie mattis nunc. Nulla ac faucibus dui. Nullam a congue erat. Ut accumsan sed sapien quis porttitor. Ut pellentesque, est ac posuere pulvinar, tortor mauris fermentum nulla, sit amet fringilla sapien sapien quis velit. Integer accumsan placerat lorem, eu aliquam urna consectetur eget. In ligula orci, dignissim sed consequat ac, porta at metus. Phasellus ipsum tellus, molestie ut lacus tempus, rutrum convallis elit. Suspendisse arcu orci, luctus vitae ultricies quis, bibendum sed elit. Vivamus at sem maximus leo placerat gravida semper vel mi. Etiam hendrerit sed massa ut lacinia. Morbi varius libero odio, sit amet auctor nunc interdum sit amet.

Aenean non mauris accumsan, rutrum nisi non, porttitor enim. Maecenas vel tortor ex. Proin vulputate tellus luctus egestas fermentum. In nec lobortis risus, sit amet tincidunt purus. Nam id turpis venenatis, vehicula nisl sed, ultricies nibh. Suspendisse in libero nec nisi tempor vestibulum. Integer eu dui congue enim venenatis lobortis. Donec sed elementum nunc. Nulla facilisi. Maecenas cursus id lorem et finibus. Sed fermentum molestie erat, nec tempor lorem facilisis cursus. In vel nulla id orci fringilla facilisis. Cras non bibendum odio, ac vestibulum ex. Donec turpis urna, tincidunt ut mi eu, finibus facilisis lorem. Praesent posuere nisl nec dui accumsan, sed interdum odio malesuada.

Bibliography

[1] Mastodon org. “Mastodon open source social media”. In: (). URL: `https://www.joinmastodon.org`.

[2] Lorenz Diener. *Mastodon API for Python*. URL: `https://github.com/halcy/Mastodon.py`.

List of Figures

List of Tables

List of Algorithms

List of Codes