Product Brief - Team Software Project 2

RED - changes made due to other comments e.g Highlighted words and sentences in bold to improve readability

GREEN - changes made by our own decision e.g Gantt chart

Overview

Within our project, we aim to create a threat-analysing scanner for Twitter. The scanner will be created as a web application that will allow users to scan accounts for behaviour that is considered to be anti-social. After viewing what the algorithm has determined to be dangerous the user will then have the option to report the account and add it to our database.

Our web application will be named Tweet Guard Threat Scanner, and will aim to provide a user-friendly environment that gives an accurate and detailed reflection of the threat, or lack thereof that may lie within the user's Twitter followers.

Overall, the aim is to create a simple, sleek and minimalistic web app that offers users an easy way to filter out any negativity, antisocial behaviour or danger from their twitter feed with the click of a button. We aim to create an algorithm that is accurate, helpful and will grow stronger as more users use it.

Why Is It Needed?

In digital times, digital crimes are also rocketing. People need a way to keep themselves safe from online threats like sexual predators, fraud accounts, internet trolls, etc.This will be perfect for people such as children, women and others who face the most threat online.

Twitter is an open platform where people can look at and talk about current affairs across the globe. With differing points of view, some people may have a perspective on a topic which might be controversial or offensive to other users. Some may also exploit the platform for malicious purposes.

Twitter has a system in place where if a user believes something violates the platform’s rules they can report it. This helps to indicate to the Twitter team that something is going on and they can then decide whether or not to take corrective measures. E.g. They may

enforce a user to delete their Tweet, or suspend their account permanently (if a more serious/repeat offence is committed).
Violations are grouped under three separate categories:

1. Safety
This includes violent threats, terrorism, child sexual exploitation, harrassment, discriminational hate (e.g racism), intentions/encouragement of self-harm or suicide, sensitive media, and illegal activity.

2. Privacy
Includes disclosing other people's private information, and non-consensual nudity.

3. Authenticity
Includes spam, civic integrity, impersonation, synthetic media, and copyright/trademark.

Although the reporting of these violations on Twitter helps to make the platform a safer place, there are a few problems that it does not address:

- It is inefficient to depend on Twitter staff to go through each user report and determine what action should be taken. This takes too long for threats to be identified and eliminated, meaning more users are vulnerable for a longer period.
- Users will be completely oblivious to a harmful account as they cannot see if it has been previously reported before.

The web application will quickly solve these problems:

- Reported information is stored in a database, so threatening accounts can immediately be identified by users.
- Users will now be able to scan for harmful accounts and see all details of previous reports.
- Whenever a threatening appearing account tries to interact with a user, the user will be able to quickly discover if other people have had the same problem, or if our algorithms have detected said threatening behaviour.
- The web application will show a danger level so that threats of all sizes are accounted for. This allows users to make their own decisions on how restrictive they would like their account to be.

[3]

So What Does It Do, And How Does It Do It?

In our web app, the main function that we are choosing to implement is a “scan” function, which uses sentiment analysis and natural language processing to provide a full report on a user’s Twitter followers. We aim to implement at least one criteria on which a user can be determined as dangerous, with racism within the tweets of a user at the forefront as our principle area of development.

The scan function can be used in two different ways:

- The user can search for a specific account and receive a full report on the account.
- The user can scan their own account and be given analysis of up to 10 of their most recent followers, documented as a full report which highlights the dangerous users, and, if possible, will give an option to unfollow a specific user or mass unfollow all threatening accounts.

When an account is scanned, the algorithm will check it’s recent activity and return an analysis of possible threats to the user. The user can then report this activity and add it to the database for future users to see. The returned analysis will contain information such as an overall “danger level” of the account, the amount of reports/flags our system or other uses have made on an account in the last month, the common reason for reporting and even some of the tweets in question that may contain said threatening qualities or anti-social behaviour.

Updated how BERT is used in our model

We first introduce a transfer learning approach for hate speech detection based on an existing pre-trained language model called BERT (Bidirectional Encoder Representations from Transformers) given preproccessed tweets pulled from a users profile, the tweets are fed into the model to be fine-tuned according to different strategies. At the end, using the trained classifiers we predict labels of the tweets and display the flagged tweets.

Added information on dataset selection

Our model is trained from 6 publicly available datasets which have been fine tuned through public opinion to mitigate bias in classification. The Davidson dataset for example Davidson crawled 84.4 million tweets from 33,458 twitter users. To annotate collected tweets as “Hate”, “Offensive” or “Neither”, they randomly sampled 25k tweets and asked users of CrowdFlower crowdsourcing platform to label them. After labeling each tweet by annotators, if their agreement was low, the tweet was eliminated from the sampled data [11]

Dataset	Source	Target	Hate	Non-Hate	Total
Davidson [5]	Twitter	Groups or members of a group	1,430	4,163	5,593
Gibert [6]	Stormfront	N/A	1,196	9,748	10,944
Waseem [7]	Twitter	Minorities	759	5,545	6,304
Basile [8]	Twitter	Women or immigrants	5,390	7,415	12,805
Ousidhoum [9]	Twitter	N/A	1,278	661	1,939
Founta [10]	Twitter	Race, Religion, Sexual Orientation, disability or gender	4,948	53,790	58,738

The user will then even be given a chance based on the mentioned tweets to agree or disagree if there was indeed antisocial behaviour involved and to block/mute/unfollow the account. Users will also be able to use the report function, to alert others of threatening accounts that may have flown under the radar of our algorithm.

We would like to add more features as time goes on, and one main feature we would like to add in future is a fact checker. With this feature, a user could enter a tweet URL and be returned an answer on whether what is stated is true or false. We feel it would be relevant in today's society, with reports of "fake news" being broadcasted on a daily basis. Again, it would be implemented with our natural language processing and machine learning techniques.

What Other Solutions Are Out There?

There are currently no solutions on the market that deal with this problem. Software security companies such as Proofpoint, FraudWatch International and Mandiant offer solutions to a problem that comes underneath the same category of social media threat protection. However they only protect branded social media accounts (different target market) and only solve a subset of the problems our product deals with.

Their security solutions monitor a wide range of social media sites for fake or fraudulent accounts impersonating a client's brand. They monitor and detect specific threats across these platforms, alerting the client when a potential impersonator is found which then must be confirmed by the client before the account is subject to removal.

Our product aims to give protection to all users of the Twitter platform specifically because it is where the problem we are addressing is most prominent.

In the past, there has been research done into similar topics that has produced varying results, with some reports of bias towards certain groups of people. We feel that due to the nature of our web app, with its main aim of producing a report for a user for them to then decide if what they have seen is untrustworthy or dangerous, this bias is obsolete. Only the outside bias of a user could influence the quality of our service.

Added information how other classifiers fail and how ours differ

Other solutions fail to mitigate racial bias in detection. In theory the racial bias stems from the training set used to train the model. Training sets label text as hate without considering the context of the text, which proves how the test data that feeds these algorithms are biased from the start. We tackled this issue by using a pretrained model that uses datasets which have been publicly constructed using various experiments to alleviate false positives in detection [9] [10]

Project objectives & flexibility/constraints

	Target	Tolerance
Scope	Web Application Reporting System Database of reported threatening accounts Danger Level Scan function (Search specific account for report) Sentiment analysis & machine learning	If time left over after necessary features are completed: Can extend functionality to scan all of your followers/ mutual accounts/ unknown accounts sending message requests Fact Checker to determine if a tweet provides true or false information Threat alert
Time	7 weeks	None as we may finish early but then scope/functionality will be extended
Quality	Front end of application implemented in HTML, CSS, JavaScript, etc. Use the Flask Web framework to speed up development Back end database implemented with SQLite Twitter API used for filtering keywords	Can use Bootstrap to implement front end Database implementation is flexible Tweepy has what we need but pull limit of 3k followers
Risks	Team member becomes ill or cannot complete task for other reason	Spread the remainder of the task workload among other members
Benefits	There are benefits to making the design look nicer or extending functionality to add more useful features	If time is an issue it is more beneficial to keep to a simpler design and to focus on required features only

Our Team

Overview - Roles

We analysed our strengths and weaknesses and for the next 2 weeks have allocated roles which will be reviewed and re-evaluated as the project progresses, when people find out what they enjoy/ are struggling with.

- Evan Dunbar - Bert Model Implementation
- Conor McGavin - Back-end, infrastructure using Flask and SQL Alchemy
- Conor Heeney - Front-end design and development, Styling Scanning Pages
- Mark Daly - Front-end design and development, Login and Admin system
- Joel McKenna - Front-end design and development, Report user page

Added information on each members role

Conor's M role was mainly concerned with back-end, as well as document editing. For the initial few weeks, he worked on setting up the database structure, the functionality of the report function, the database search function, as well as the total reports list. He and Evan have both been in charge of the editing of the project brief and other documents. In later weeks, Conor worked with Evan's scan function and embedded it into the site, using the TweePy API in the process to access users' profiles and data and giving this information to the front-end developers to output to the site.

Evan's role for the project was mainly with the back-end, For the first few weeks he worked on implementing functions for pulling twitter users tweets, details, followers and following using the TweePy API. Initially he implemented a Logistic Regression to detect hate speech which was later updated to a BERT model when faced with the issue of racial bias in the initial model. He wrote functions to return the Tweet information that was considered a threat for the front ends team to display. Throughout the project Evan made edits to the brief and Gantt charts recording project evaluation and changes made throughout the project.

Mark started off with front-end design of the application, implementing the basic design from the prototype that was proposed. He then moved onto setting up the login system to allow users to log in with Twitter's API and to restrict access to certain pages. Since the application now knew the current user's details, Mark could change the previous implementation of entering your username to scan your account to a click of a button. He then implemented an admin system to enable manipulation of the database to authenticated users, and allowed a regular user to withdraw their reports.

Conor H's role has mainly been continuous front-end development. He started off in conjunction with Mark in putting together the initial designs of each page. To create a full front-end implementation, he needed to correspond with Conor M to import back-end data such as scan results and report data from the database. Later in the project, as requirements advanced, Conor and Mark split the styling of pages between each other and Conor was primarily responsible for the design of the scan pages. This front-end work is still ongoing and evolves as the weeks progress. Conor is currently working with Conor M in implementing a feature that's in development where a user can unfollow a scanned follower. The ideal approach is to use ajax to send a request to a back end function which will complete the action. Using ajax prevents reload and re-scanning. Conor H, having experience with ajax, is currently assisting Conor M with this implementation to make the task more streamlined.

Our Approach

Removed comments such as "about halfway through the week"

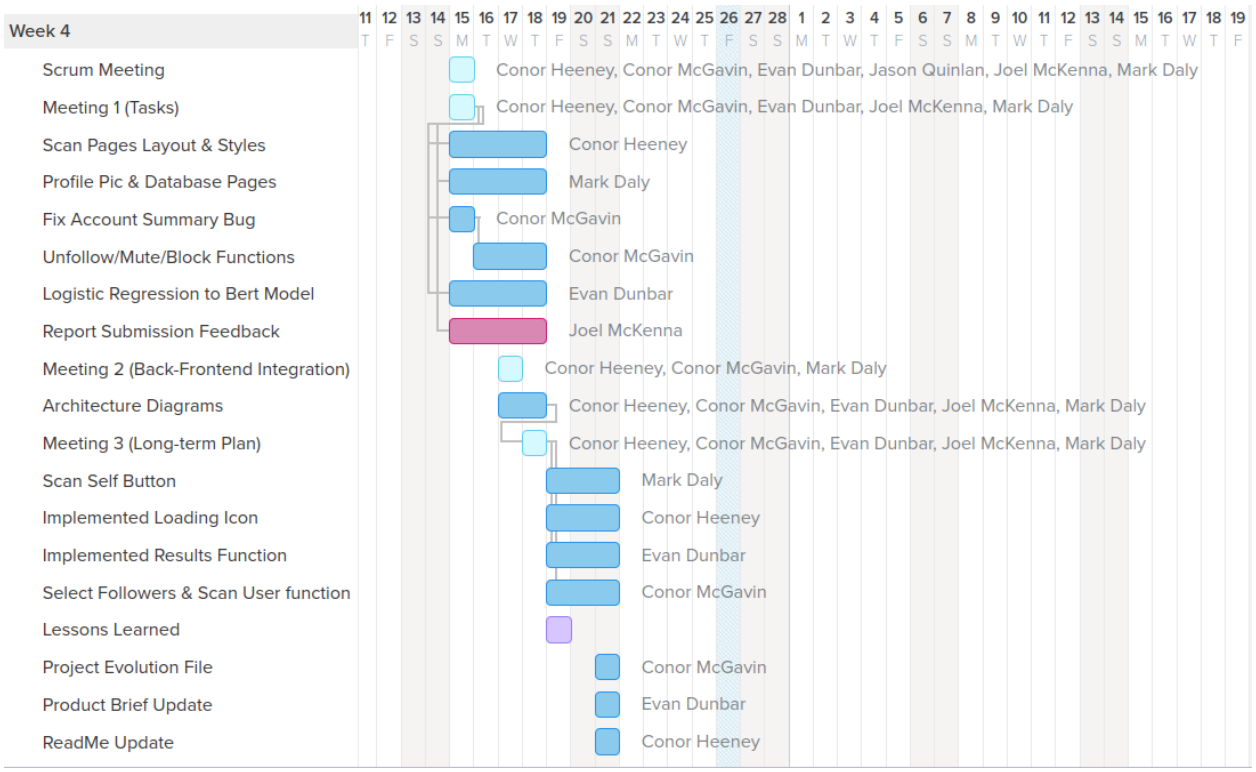
We will be taking an Agile approach to the project using Scrum techniques. The typical routine for a week involves our Scrum meeting with the product owner and the whole team. This is where we discuss reflections on the previous week and what needs to be done in the upcoming week. We then have a meeting with the development team, where we discuss the distribution of tasks among members. We then work on our own tasks, separately for the most part. the end of the week, we create the Gantt chart for the week and update the product brief to reflect any changes to the specification.

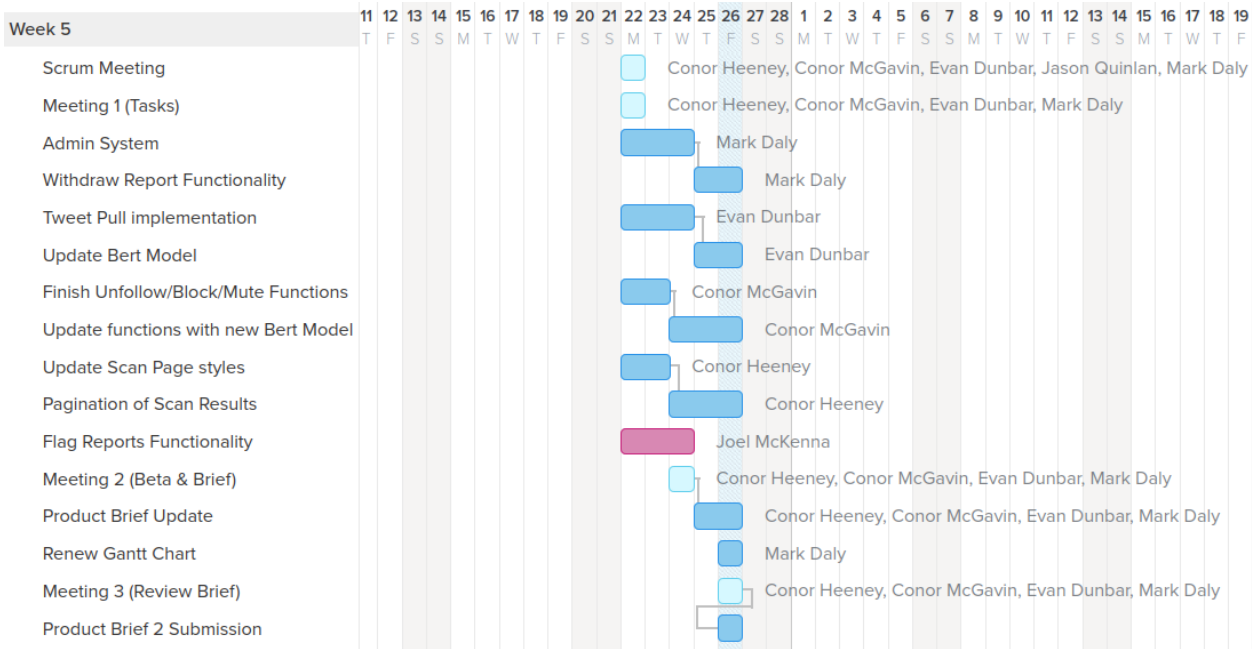
We aim to understand each other's roles and work closely and collaboratively to produce an environment where one team member will be able to work independently of another, but know what is expected of them to produce from other team members. This ensures fairness and that there is never one person that feels they are doing everything. We aim to closely follow a plan and reassess on a weekly basis on where the project is, and where it needs to be.

Gantt Chart

Light blue - Meetings Dark blue - Tasks

Red - Not completed Purple - Individual Deliverables

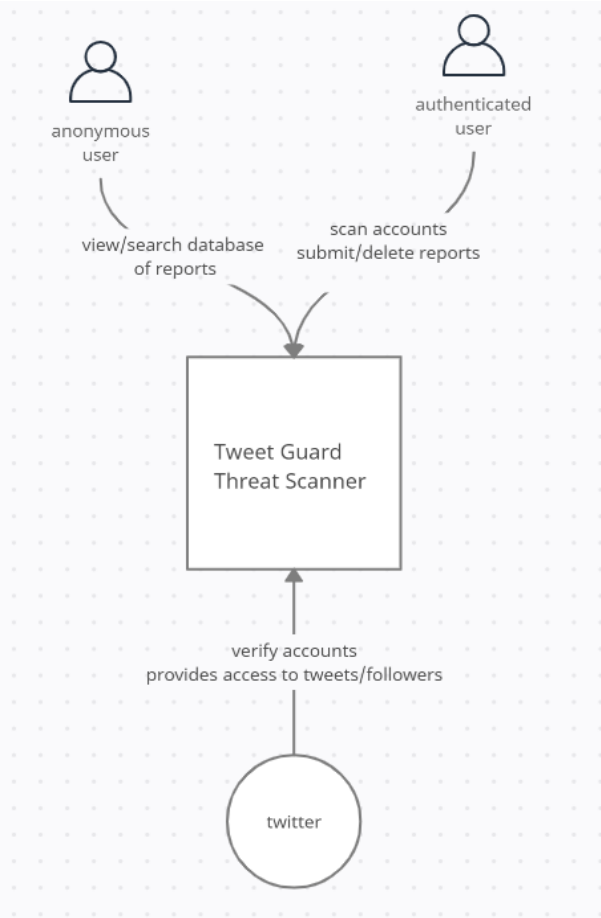




Added descriptions and diagrams outlining the application architecture

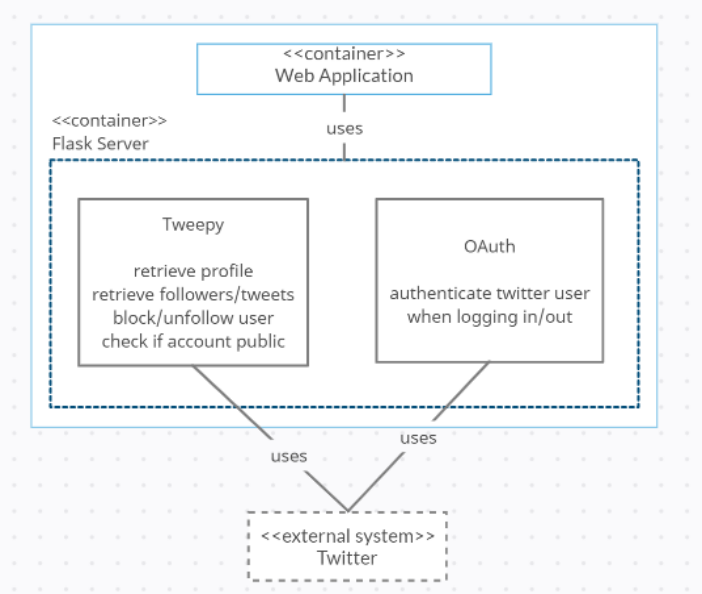
Architecture

We have a number of diagrams outlining the architecture of both the components that make up the project, and the interactions a user can have with it. These technologies and services form the backbone of the application, and are used in a variety of ways to build a full fledged web app. These components range from external APIs to local databases to our own defined functions



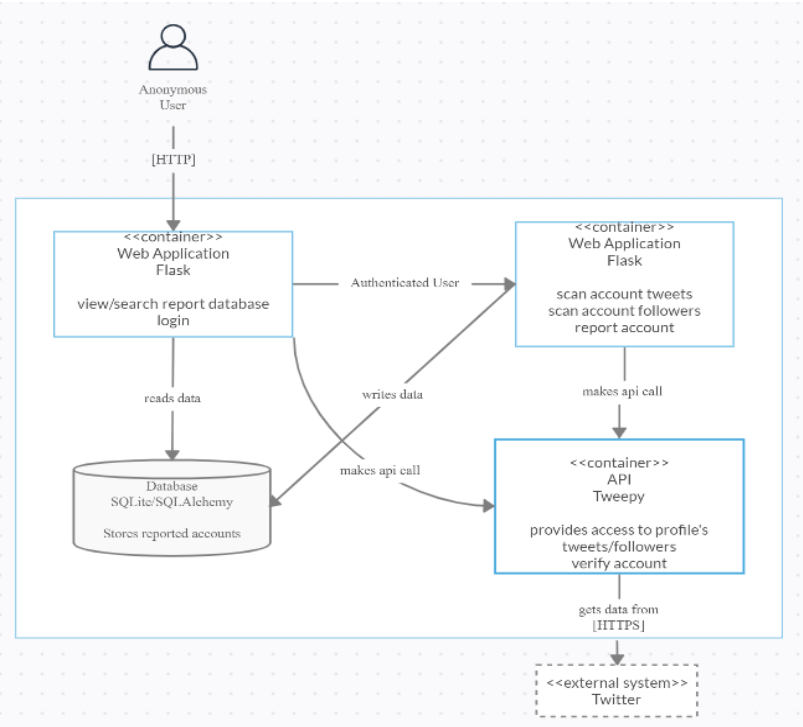
The application uses Tweepy, a well known Python library, to access the Twitter API. This is useful throughout the application. It's used by the scanning algorithms to pull

tweets, for the OAuth authentication, to implement the unfollow/block functionality and it's used throughout the application to pull user information



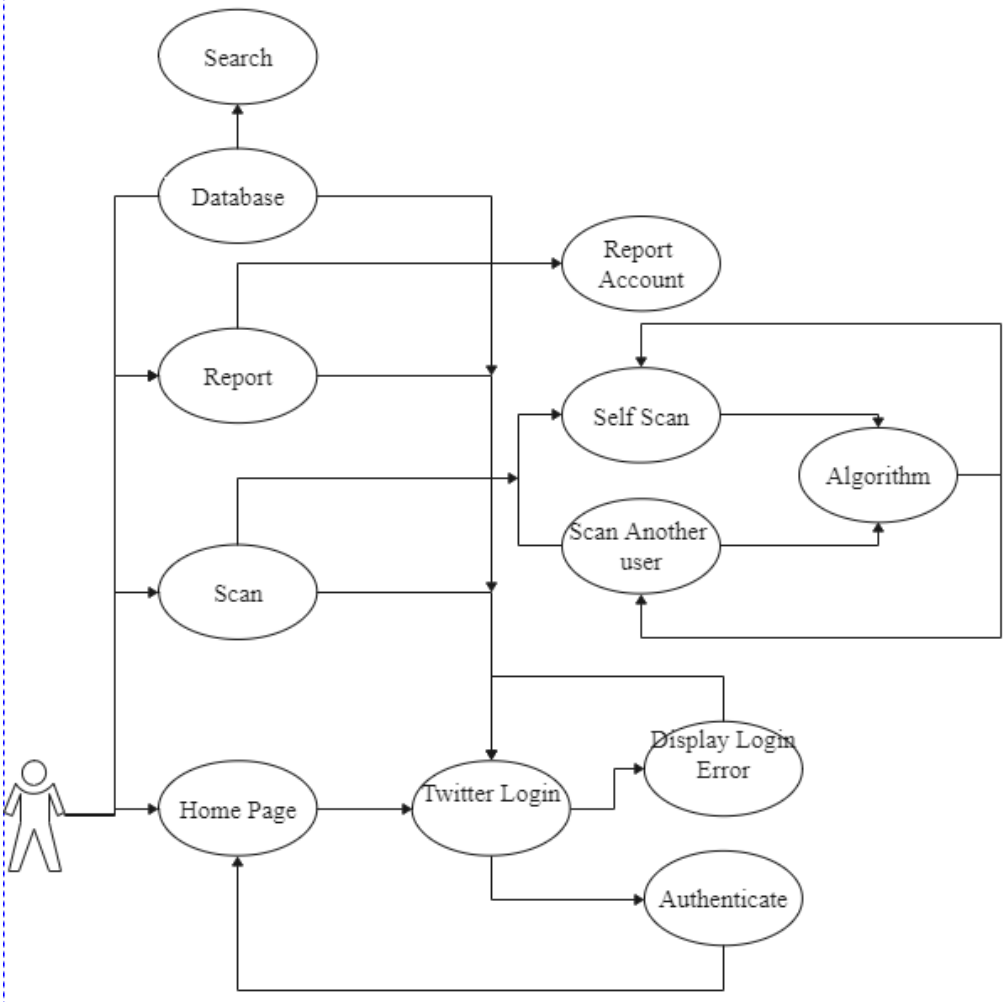
Visualisation of the interactions with Tweepy

The application uses an SQL database to store information for manually reported Twitter users. This data is used so Tweet Guard users can search for reports submitted by humans to get a more opinionated perspective from other users.

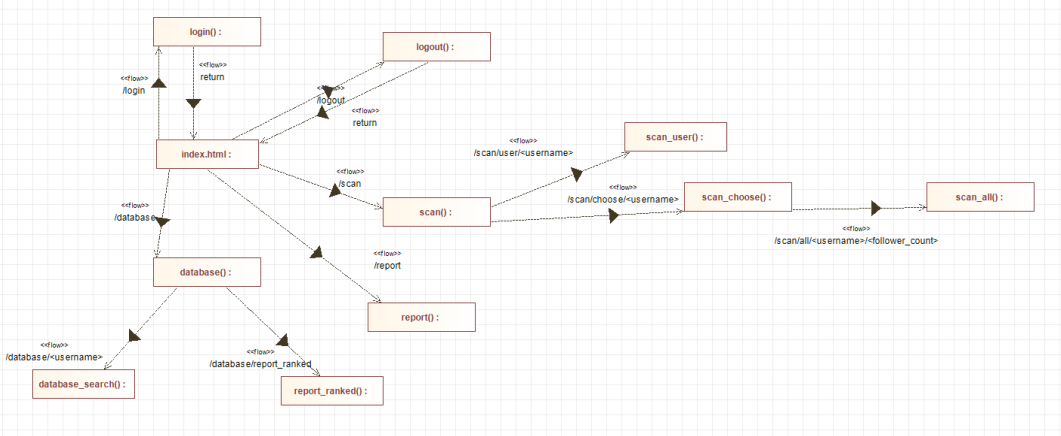


Visualisation of interactions with the SQL database

The following two diagrams show how the user can interact with our application at a higher level. The first is a use-case diagram, showcasing the paths the user can take and what options are available from each page. The second is a communication diagram showing a high level overview of the functions contained in the back-end, and the routes that are used to get there.



Use-case diagram showcasing the possible paths a user can take



Communication diagram showcasing how functions are called

MoSCoW - Updated

Must Have:
Scan Function - scan up to 10 followers and scan specific person for all threats
Report Function
Database of Reports
Login functionality w/ twitter integration
“Total reports by user” page
User lookup with reports
No stored scan results to keep database size low
BERT model to make scan function more effective

Should Have:
Account summary / profile with calculated danger level and other statistics
Javascript to click and select users to scan
Option to unfollow / block à user that is scanned as bad
Loading screens
Paginated results on all pages
Ability for an admin.moderator to take down reports
Ability for à user to delete their own report

Could Have:
Fact checker if time at end
Lower waiting time for scan results
Option to report a user that has been scanned
Like/dislike button on report?

Won’t Have (this time around):
Scan unlimited amount of accounts
Very fast scan results
Trust function that calculates if a report made by à user could be invalid/ lying

Conclusion

We hope this document puts into perspective both our vision of our product and our vision for good team communication and planning.

We are happy with our current product specifications and although we feel we have a lot to cover, we hope this product can continue to evolve, with more functionality and polish than is outlined in this document.

Thank you for taking the time to read this.

Updated references removing unused references and fixed links to referenced text

References

[1] Help.twitter.com. 2021. *The Twitter Rules*. [online] Available at: <https://help.twitter.com/en/rules-and-policies/twitter-rules> [Accessed 21 February 2021].

[2] Davidson, T., Warmsley, D., Macy, M., Weber, I.: Automated hate speech detection and the problem of offensive language. In: Eleventh international aaai conference on web and social media (2017)

[3]. de Gibert, O., Perez, N., Pablos, A.G., Cuadros, M.: Hate speech dataset from a white supremacy forum. In: Proceedings of the 2nd Workshop on Abusive Language Online (ALW2). pp. 11–20 (2018)

[4] Waseem, Z., Hovy, D.: Hateful symbols or hateful people? predictive features for hate speech detection on twitter. In: Proceedings of the NAACL student research workshop. pp. 88–93 (2016)

[5] Basile, V., Bosco, C., Fersini, E., Nozza, D., Patti, V., Pardo, F.M.R., Rosso, P., Sanguinetti, M.: Semeval-2019 task 5: Multilingual detection of hate speech against immigrants and women in twitter. In: Proceedings of the 13th International Workshop on Semantic Evaluation. pp. 54–63 (2019)

[6] Ousidhoum, N., Lin, Z., Zhang, H., Song, Y., Yeung, D.Y.: Multilingual and multi aspect hate speech analysis. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). pp. 4667–4676 (2019)

[7]. Founta, A.M., Djouvas, C., Chatzakou, D., Leontiadis, I., Blackburn, J., Stringhini, G., Vakali, A., Sirivianos, M., Kourtellis, N.: Large scale crowdsourcing and

characterization of twitter abusive behavior. In: Twelfth International AAAI Conference on Web and Social Media (2018)

[8] Mozafari, Marzieh, et al. “Hate Speech Detection and Racial Bias Mitigation in Social Media Based on BERT Model.” *PLOS ONE*, vol. 15, no. 8, 27 Aug. 2020, p. e0237861, 10.1371/journal.pone.0237861. Accessed 7 Nov. 2020.

[9] Aluru, Sai Saketh, et al. “Deep Learning Models for Multilingual Hate Speech Detection.” *ArXiv:2004.06465 [Cs]*, 9 Dec. 2020, arxiv.org/abs/2004.06465. Accessed 28 Feb. 2021.

[10] “Hate-Speech-CNERG/Dehatebert-Mono-English · Hugging Face.” *Huggingface.co*, huggingface.co/Hate-speech-CNERG/dehatebert-mono-english.