BirdCatcher

Initial Technical Report

Team Bravo

Ryan Taylor (Team Leader), Robin Graf, Wai Kai Wu

Jack Nicol & Michael Graver

Critical Use Case Model

Core Idea

The Vision is to have a system that lets you control and manage social media, this would be useful for Organisations and Events. Specifically aimed at twitter but for possible future expansions to other social networks, such as Facebook or Instagram and others. Admins would be able to set conditions for which tweets could be displayed.

User Requirements

To have a clear understanding of what our design and eventually also the system needs to do, we wrote down a list of functional and non-functional requirements for the system. We then reviewed these requirements together with the client to ensure we had the correct idea in mind.

Functional:

1. A user can view the tweets that satisfy the admins filter conditions.
2. An admin can set a filter for hashtags.
3. An admin can set a filter for twitter handle.
4. An admin can remove a tweet from view.
5. A user can *interact with a tweet.*
6. An admin can remove a twitter handle from the *approved list*.
7. An admin can login using username and password.
8. A user can filter by admin defined conditions.
9. An admin can *sticky* a tweet.
10. An admin can *sticky* a hashtag.
11. A user can search for specific tweet within the system.
12. A user can view a geotagged tweet on the map from handles on the *approved list.*

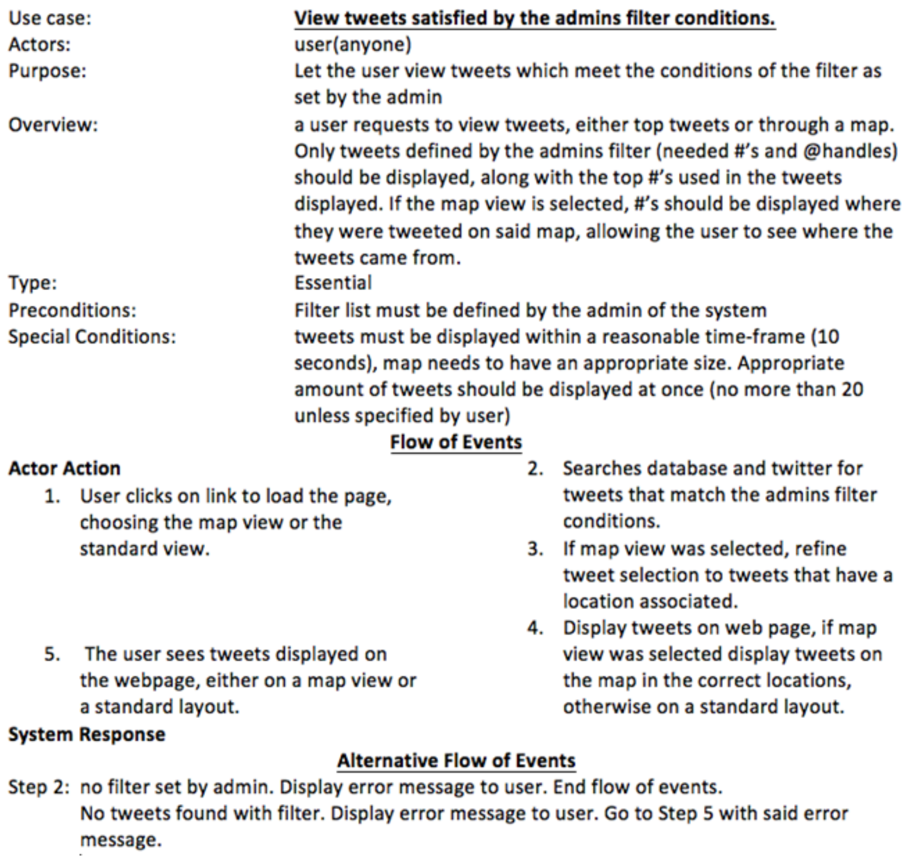
Non-Functional:

- The size of the map should be appropriate. (10m to 1000km)
- An appropriate number of tweets should be displayed at a time, unless specified (0-10 tweets).
- There will be a mobile and a desktop version of the application.
- Tweets should be displayed within a reasonable time frame (10 seconds, depending on internet connection).
- The application will work with latest versions of Firefox, Safari, Chrome (which support HTML5).
- The tweets will be "lazy loaded".

Meaning of Words

sticky – To display what has been "stickied" always at the top of the page.
approved list – A list of hashtags and usernames as set by the admin.
interact – To reply, re-tweet and like the tweets displayed on the page.
Lazy loaded - loaded on command, i.e. when scrolling.

Core Use Cases



Use case:	**View tweets satisfied by the admins filter conditions.**
Actors:	user(anyone)
Purpose:	Let the user view tweets which meet the conditions of the filter as set by the admin
Overview:	a user requests to view tweets, either top tweets or through a map. Only tweets defined by the admins filter (needed #'s and @handles) should be displayed, along with the top #'s used in the tweets displayed. If the map view is selected, #'s should be displayed where they were tweeted on said map, allowing the user to see where the tweets came from.
Type:	Essential
Preconditions:	Filter list must be defined by the admin of the system
Special Conditions:	tweets must be displayed within a reasonable time-frame (10 seconds), map needs to have an appropriate size. Appropriate amount of tweets should be displayed at once (no more than 20 unless specified by user)

Flow of Events

Actor Action	
1. User clicks on link to load the page, choosing the map view or the standard view.	2. Searches database and twitter for tweets that match the admins filter conditions.
	3. If map view was selected, refine tweet selection to tweets that have a location associated.
5. The user sees tweets displayed on the webpage, either on a map view or a standard layout.	4. Display tweets on web page, if map view was selected display tweets on the map in the correct locations, otherwise on a standard layout.
System Response	

Alternative Flow of Events

Step 2: no filter set by admin. Display error message to user. End flow of events.
No tweets found with filter. Display error message to user. Go to Step 5 with said error message.

Use case:	**Receiving a new Tweet from Twitter that matches the filter condition**
Actors:	Twitter
Purpose:	Put new Tweet into database to display to users at a later time
Overview:	a preapproved twitter handle makes a tweet which satisfies the filter condition set by the admin. The twitter API then passes the tweet onto the system which stores the tweet in the database.
Type:	Essential
Preconditions:	Filter list and approved handles must be defined by the admin of the system
Special Conditions:	system shouldn't miss 99% of tweets which satisfy the filter condition

Flow of Events

Actor Action	**System Response**
1. A new Tweet is created on Twitter	2. System finds tweet. 3. System checks tweet to make sure it satisfies filter condition and handle is in approved list. 4. Analyse and put tweet into storage

Alternative Flow of Events

Step 2: No tweet found (because it was deleted or another reason). End flow of events.

Step 3: no filter set by admin. Notify admin, End flow of events.

Use case:	**Setting a filter and approved handles**
Actors:	Admin
Purpose:	Creating a new filter for incoming tweets and setting approved handles
Overview:	admin decides to create or edit the filter and change approved handles. Views current filter and approved handles, then makes changes, which are saved to the database (to be used for finding tweets later).
Type:	Essential
Preconditions:	none
Special Conditions:	filter should use #'s

Flow of Events

Actor Action	**System Response**
1. Admin decides to change filter and/or approved handles.	2. System displays existing filter and approved handles.
3. Admin makes changes to filter, removes or adds handles.	
	4. System saves and stores changed filter and list of approved handles in database.
6. Admin reviews changes to filter and approved handles.	5. System displays changed filter and list of handles.

Alternative Flow of Events

Step 2: No existing filter and approved handles, display empty filter and empty list, continue from step 3.

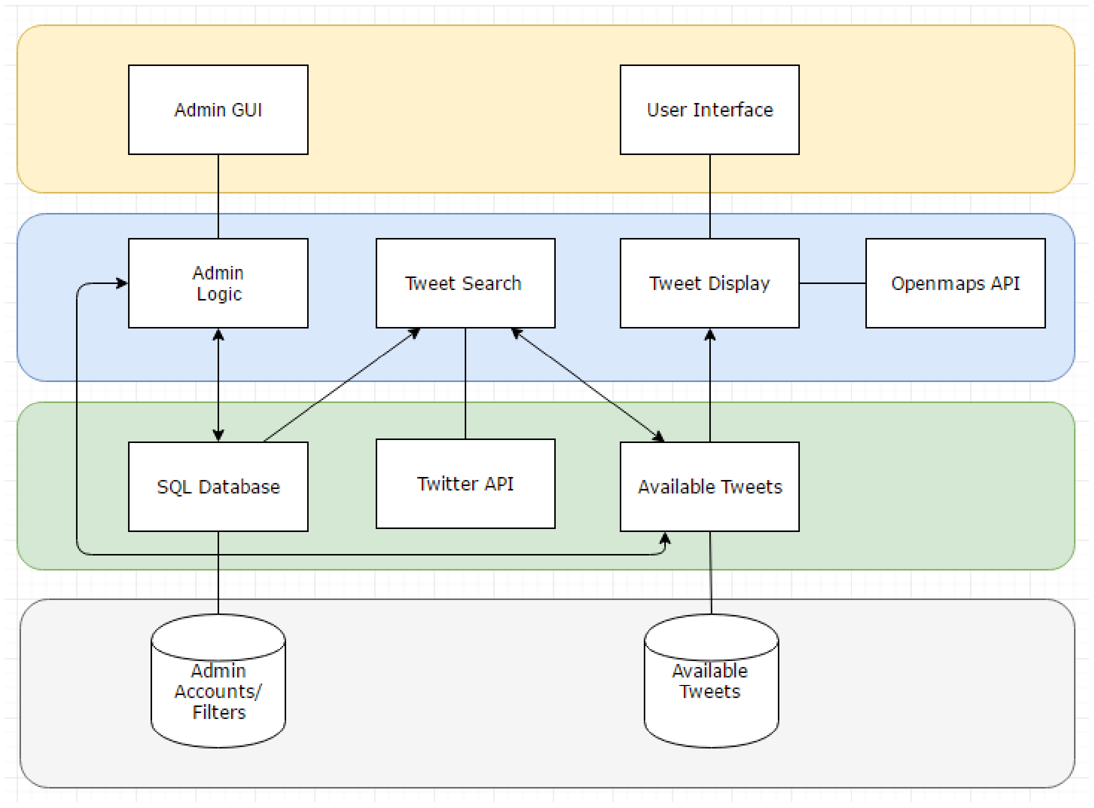
Step 4: no changes, specify no changes made, redo step 3.

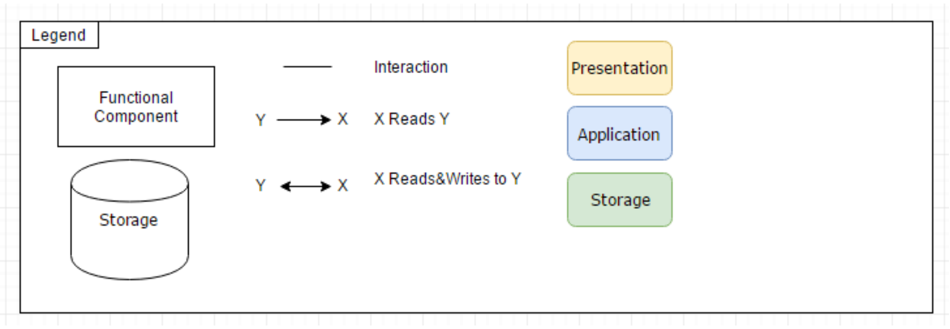
Step 6: admin is unhappy with changes, go back to step1.

System Architecture

Three Tier Structure

We decided a three-tier layered architecture because it made the most sense and is already a common structure for web systems. Below is an image of what our rough architecture looks like.



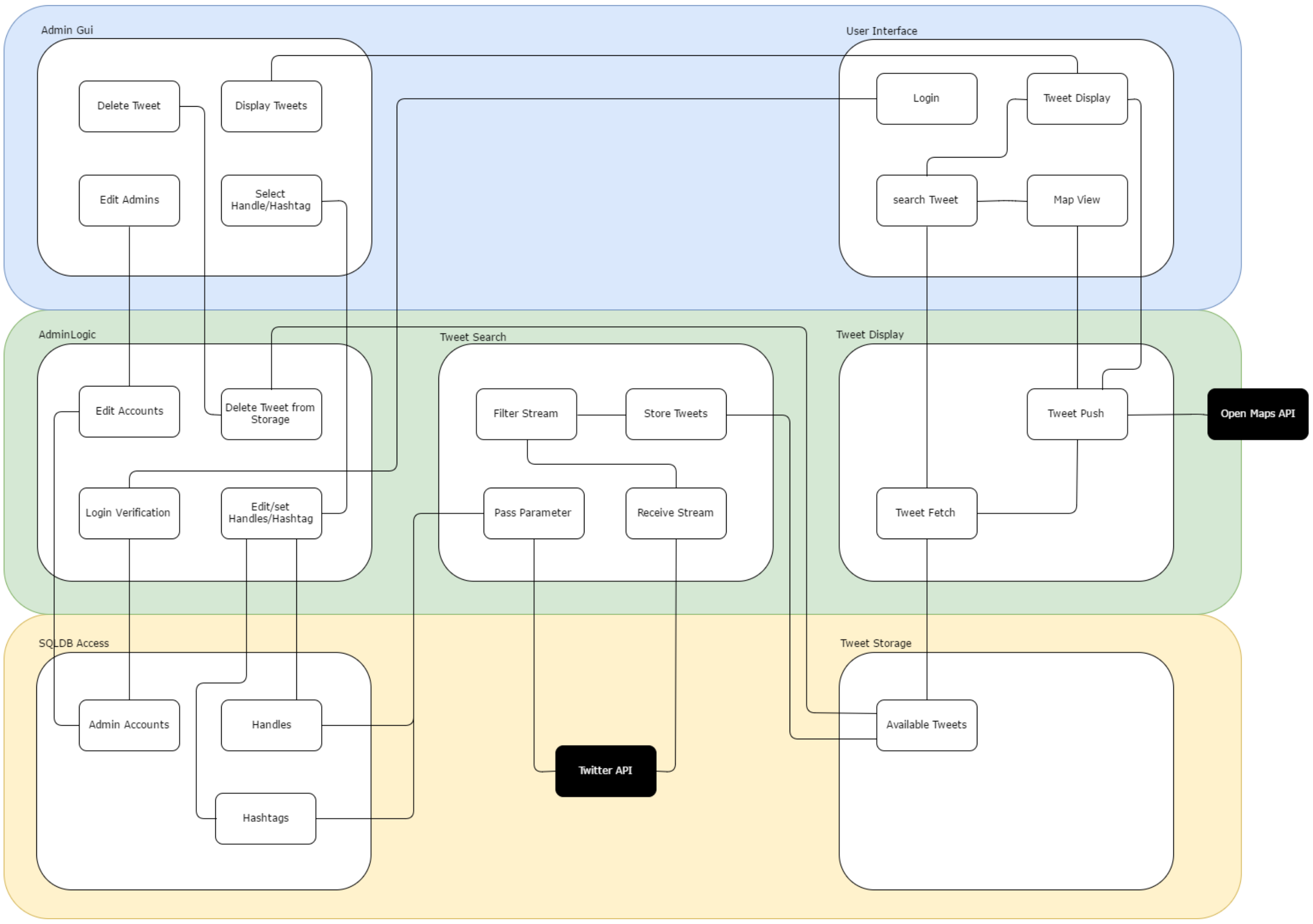


At the bottom, we have our storage layer, with 2 different storage methods (one Database, the other NoSQL), and 2 external API's.
"Admin Accounts/Filters" DB stores the login information for each admin and the different hashtags and handles for which we want to fetch the tweets with.
"Available Tweets" Storage has all the tweets that meets the admin's filter conditions and that the users can see.
The Twitter API is passed parameters per the admins filter from Tweet search, then returns a stream of tweets.
The Open Maps API passes the map which is used for presenting the tweets on a map, given their location.

In the middle layer, we have 3 main applications.
"Admin Logic" controls which handles and hashtags to pass through search and which tweets to display. It also allows the admin to delete tweets from the already available tweets.
"Tweet search" passes the parameters to the Twitter API using the admins filter condition. It then also receives the stream of tweets from the Twitter API and stores these tweets in the NoSQL DB system.
"Tweet Display" on request from the user interface it takes tweets from "Available Tweets", ranks and sorts them then passes them to the user interface where they are displayed to the user.

On the top layer, we have 2 different interfaces.
"Admin GUI" lets an admin interact with the system; setting filters etc.

"User Interface" this is where any user can view tweets, either on a map or in a standard layout.



We had thought about other architectural patterns but decided that they were less appropriate. We decided to go more in-depth with our design, to make sure it would be low coupling and high cohesion, obviously, it would be impossible to maximise cohesion without making the system be just one module, so we did what we could. In terms of coupling there are at most 3 links per module and one module has 4 links if you count the api as a linkage. The twitter API is in the storage layer because it provides data, like a database would. The Open Maps API is in the Logic layer as it behaves more like a library.

We decided for 2 different storage solutions. Having the tweets stored in a noSQL database for faster access when a user requests to view a tweet (which we expect to be the most frequent action on the system). Anything related to the Filters set by the admin, or details of the Admin accounts are stored in a SQL database system.

Risk Assessment

Possible Risks

Risk	Risk level	Impact
Lack of experience in team	Medium	More time spent learning the system. This could increase time to implement and therefore impact quality negatively.
Mismanagement of funds	Low-Medium	High -Running out of money to develop Project.
Time Constraints	Low-Medium	Impact quality negatively as team will have to work faster to meet deadlines.
Inadequate Design	Low	Creating a sub-par system, or having to go back and completely redo Design.
Security	High	System being susceptible to hackers accessing protected data.
Problematic APIs	Very Low	Twitter API unlikely to be problematic. However, if it was, it would lead to the failure of project.
Twitter downtime	low	High -Twitter being down, would mean our system would be useless, and twitter permanently being down would make our software obsolete.
Change in prominent social media	Low-Medium	There is a possibility that Twitter becomes unpopular, meaning we

		would have to keep up and adapt our software to the most popular social network. This would impact our development team; however, we have plans for this potential change.

Important Risks

The most important risks are those that have a high impact and a high chance of happening and or medium chance and high impact, or high chance and medium impact.

Security

To keep security risks to a limit we should implement a good and secure method for anything related to an Admin function, since that would be where the sensitive information might be.

Mismanagement of funds

Running out of money would be the surest way to spell the end for the project. For development we are using Heroku to host the service, meaning while in development we can save some of the funds. If there is a complication in development and more time needs to be spent on developing certain features, either more funds have to be spent on development time or there will be less features in the finished system.

Twitter Downtime

There is no real way of reducing the risk of twitter downtime, but in our design we can account for the downtime. This is done in that we store tweets as they are created, meaning they can still be viewed during the downtime.

Project Plan

Initial Schedule

Below follows the weekly meetings and updates from the beginning of planning through development. This includes what we expected to have completed and what we needed to complete next.

Inception Phase

Week 1

- Introduced to team members.
- Booked meeting with guide, Bruce Scharlau.
- Decided to meet every Tuesday.

Week 2

- Decided on Team Leader (Ryan Taylor).
- Decided on Deputy Leader (Robin Graf).
- Met with Bruce Scharlau.
- Brainstormed ideas for project.
- Agreed upon an Agile approach.

Week 3

- Agreed upon a project.
- Created a business plan.
- Considered Ruby/Rails or Python/Django.
- Interviewed Clients.
- Received requirements from clients.

Elaboration Phase

Week 4

- Discussed functional requirements.
- Created user stories.
- Prioritized user stories.
- Decided on Django framework for implementation.

Week 5

- Finalised financial plan.
- Decided on non-functional requirements.
- Changed functional requirements.

Design Phase

Week 6

- First draft of Architecture diagram.
- Showed Technical documents to Bruce Scharlau for feedback.
- Made a use case diagram.
- Made a sequence Diagram.

Week 7

- Minor changes to class diagram.
- Reviewed architectural design.

Week 8

- Played around with twitter API, how to get a feed from a user and from a certain hashtag.
- Created a diagram with submodules and their connections.
- Discussed about the page layout.

Week 9

- Minor changes to architectural design.
- Fixed some submodules and their connections.
- Created some design patterns for a submodule.

- Worked on the Twitter API.

Implementation Phase

Week 10

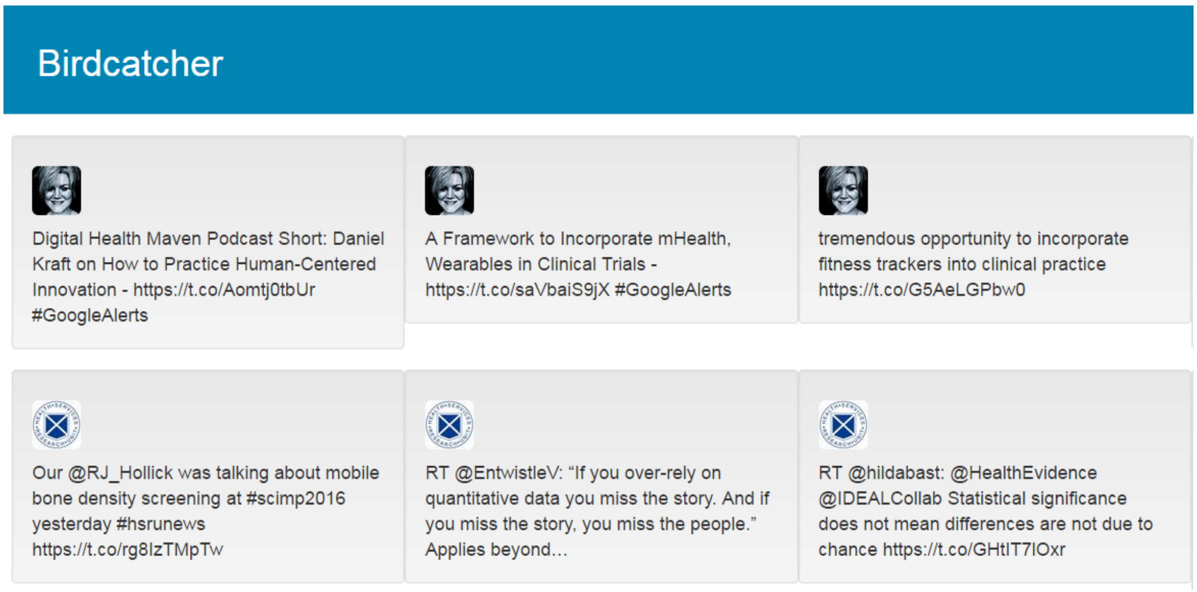
- Started writing White paper and Technical reports.
- Set up workspace on Cloud9.
- Wrote out some test cases.
- Used twitter API to display selected tweets.

Week 11

- Refactored the implementation of the stream functionality.
- Implemented a script to take the JSON from the Twitter API and write it to a file.
- Implemented a script to read the JSON file and assign the correct values to our arrays.
- Implemented the html template which iterated through the arrays to generate more divs in the html template.
- Implemented a basic CSS file for the main page with Bootstrap.
- Push project to local git.
- Push local git to deploy on Heroku.

Proof of Concept

Screenshot of our prototype


Birdcatcher

Digital Health Maven Podcast Short: Daniel Kraft on How to Practice Human-Centered Innovation - https://t.co/Aomtj0tbUr #GoogleAlerts

A Framework to Incorporate mHealth, Wearables in Clinical Trials - https://t.co/saVbaiS9jX #GoogleAlerts

tremendous opportunity to incorporate fitness trackers into clinical practice https://t.co/G5AeLGPbw0

Our @RJ_Hollick was talking about mobile bone density screening at #scimp2016 yesterday #hsrunews https://t.co/rg8IzTMpTw

RT @EntwistleV: "If you over-rely on quantitative data you miss the story. And if you miss the story, you miss the people." Applies beyond…

RT @hildabast: @HealthEvidence @IDEALCollab Statistical significance does not mean differences are not due to chance https://t.co/GHtIT7IOxr

Software development so far

For the proof of concept, we have implemented the most important feature of our system, the Twitter stream. The system will return and display a batch of tweets controlled by a set of parameters we send to the Twitter API. Right now, we are using the REST API, which only returns tweets when requested. Later, we will change it to the Streaming API, which will return a tweet whenever they are posted.
As for the rest of the functionalities, we have already planned how they would be implemented and how they would look like.

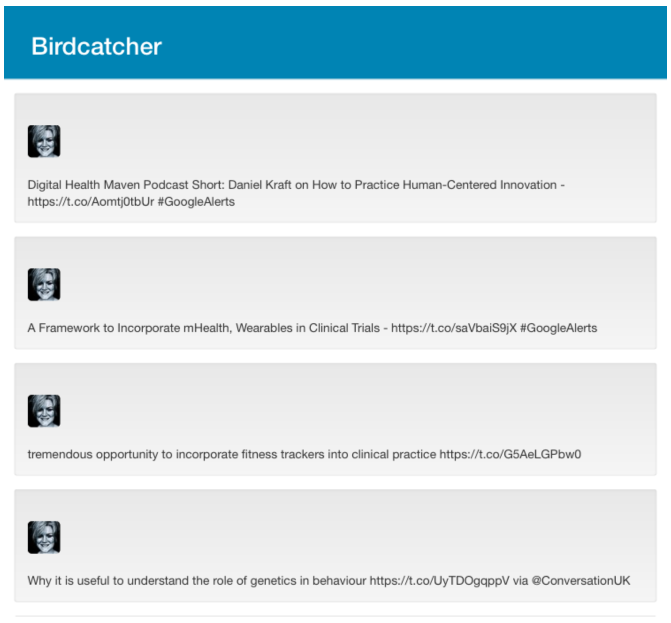
Problems

We have encountered some issues during our development, but it is mainly because we have few to no experience with web development in general. Creating the project with a Django framework was straightforward. Importing the Twitter API took some time, but we got a hang of quickly. When we wanted to deploy it to Heroku though, we have encountered numerous problems, which took us a considerable amount of time to solve. The C9 cloud IDE came with a lot of packages pre-installed, but the majority we required were missing or outdated, namely gunicorn, which hindered our system from starting up. Once we installed and updated everything, it worked like we wanted it to.

Achievements and what worked as expected

We have laid out the groundwork for our project. The Twitter API works seamlessly, just like we expected. For now, we have to input the parameters directly into the code, but for the final project we will have implemented an admin system, which allows an admin to input them from the website.

Another feature we have achieved concerns the user interface. We have implemented bootstrap so our website adapts to different devices. It will work on mobile and desktop.


Birdcatcher

Digital Health Maven Podcast Short: Daniel Kraft on How to Practice Human-Centered Innovation - https://t.co/Aomtj0tbUr #GoogleAlerts

A Framework to Incorporate mHealth, Wearables in Clinical Trials - https://t.co/saVbaiS9jX #GoogleAlerts

tremendous opportunity to incorporate fitness trackers into clinical practice https://t.co/G5AeLGPbw0

Why it is useful to understand the role of genetics in behaviour https://t.co/UyTDOgqppV via @ConversationUK

The rest of the implementation should be straightforward too unless we stumble upon some unanticipated problems.

What needs to be changed or redone

Since we have only done the basis of our system, there will be quite some work left. The difficulty should be moderate, but the amount might be more substantial.

First, we need to change the Twitter API from an on-request version to the stream version, which return a tweet as soon as it is posted. Then we also need to connect our system to a database. Luckily the cloud9 cloud IDE has an integrated database, but for simplicity’s sake we are using a text file to store the returned tweets right now. Django also has an admin system by default, so we just need to figure out how to process an admin’s input and pass it to the Twitter API. Other Twitter features we are going to add are the retweet/favourite/share buttons. Furthermore, we need to add a search field for the user to search through the available tweets.

Optional functionalities that we might include are the Tweet Map function and other social network services integration. The map function requires the OpenMaps API, which we have no experience with at all so we might have some difficulties. When we eventually arrive at that point, we will revaluate it, discuss with our client and decide whether to include it or not. Since the user interface is not our top priority, we will polish it up at the end of our development.

Conclusion

Having finished the initial planning and development of the project we have several lessons which will help us be better equipped next semester. One of the main lessons learned would be to take a closer look at the risk assessment and take more time to fully assess certain parts.
We also found that working on the project as a group with everyone present was very beneficial to the development of the project as everyone could keep up to speed and provide some input.
Overall the project was well managed but as always there were many things to evaluate and learn from. The groundwork has been completed and we will be able to complete the project with relative ease.