COMP4 – Twitter Bookmarking System

Analysis

1. Introduction

1.1 Identifying the client

The system I am going to develop if for Mr Adam McNicol. The project is to create a system that analyses the tweets of the users and to provide information about the tweets specified. Also he wishes the app to facilitate the tracking of his colleagues' tweets and to provide suggestions of tweets and other people to follow. The system is to make following trends in the world of computing easier, finding resources and to make the searching of tweets of his own and of other users easier without the human errors encountered by manual searching of tweets. A possible aspect of the program would be to display detailed info-graphics about the most recent tweets in the public timeline such as topics most discussed based on keywords. Furthermore the process being automatic would ameliorate issues regarding the time used to search tweets and perform data queries about those tweets.

1.2 Define the current system

Currently to search the tweets of a timeline, Mr McNicol has to use the twitter web interface to search tweets on his twitter feed. On the home timeline twitter has a search bar into which a term or a *hashtag* can queried which twitter then uses to search all public tweets and displays them to the user. The user then has the option to limit the tweets from the top tweets tom all of twitter and to simply the users they follow. Previous searched are stored in a cookie in the web browser thus making past queries easily accessible for the user. Furthermore in the search for other resources and users to follow, Mr McNicol will go to sites such as 'Computing At Schools' (http://computingatschools.org.uk/) where teachers nation-wide use a forum to communicate and share data. Many of these users have Twitter accounts, which they use to post quick posts about new resources and other information regarding computing. If one of these users is interesting the Mr McNicol must then search for these users on Twitter before he is able to follow them. To find out the current trends within the computing sphere Mr McNicol must manually search for what *may* be trending on Twitter; there is no formal way of limiting the trends provided by twitter to a certain topic. Also many resources are housed at websites external to twitter; these have to be bookmarked in the web browser after finding them on Twitter from Users Mr McNicol follows, also from tweets that are from returned queries from Mr McNicol.

1.3 Describe the Problems

There are many problems with the current system that lead to the need for a stand-alone program to replace the system.

- The system is very un-reliable as with following many people on twitter one often misses tweets from users that could be very important.
- The system is tedious, as it requires high diligence in that one has to spend large amounts of time searching their twitter feed in order to find relevant tweets.
- The system is very difficult to manage as many resources come from many different sites across the World Wide Web, and thus keeping the resources in one centralised place can be difficult.

- Keeping track of separate blogs is also very difficult in the current system as they is no way to process links in tweets separate from manually sorting the links. Missing links through not being diligent enough is also a problem to the user with the current system.
- Following trends within the computing community is also impossible to follow using the current system using the Twitter Web Interface, and so a new system will be needed to make this feature available to the client.

1.4 Section Appendix

Kurt Hornett

Questions for Analysis:

1. For what purpose do you want a new system?
   - Want to keep track of trends in Computy + Colleagues
   - Resource to share, thoughts on practices.

2. What is your current system; what are the current processes and functions?
   - Check 'Computy at Schools' for ans
   - Most use Twitter → Say able twitter, follow tweet

3. What issues do you have with the current system and processes?
   - Miss tweets, too many tweets difficult to manage
   - Links to blogs.
   - Very tedious, to check timeline, needs vigilance

4. What are current inputs and outputs of the system?
   → Search for username → follow
   → Queries show related tweets
   → Tweet, blog-posts, resources et al.

5. What inputs and outputs do you want from the new system?
   → Remove some, similar info
   → Indicator on whether certain tags popular
   → See if follows should be suggested.

6. How often do you search Twitter or your Home Timeline?
   4/5 times a day

7. How many tweets do you plan to get info about in one query?
   → Stick with Lowest

8. Regarding to the new system, are there any features that you require?
   → Easily add bookmarks to wb
   → See blog-posts
   → Integrated w/ other services

9. What system are you planning to use?
   Macbook

Kurt Hornett

10. Are there any security concerns you may have with a system?
→ Standard twitter authentication

11. How many users will be using the system?
- Just 1, client

12. Do you get any errors in the current system?
→ None available in C.S.

13. How do you want these errors handled?
→ See doc.

14. Are there any key requirements for the new system?
→ Must link to twitter account (testing)

15. Are there any features you would like to see that are not necessary?
→ Interface must be intuitive
→ Homepage + Del.ic.ous
→ Graph of who connected to.

2. Investigation

2.1 The Current System

The current system is a manual system that requires the user to use the Twitter web interface to make any query regarding the tweets on their timeline. The user after logging in to Twitter from the twitter website (http://twitter.com) is then presented with their home timeline. Then the user inserts the term they wish search their tweets with into the search box at the top of the UI. Once sent to Twitter another web page is rendered which displays the results of the search, specifically the top tweets of twitter, the user does have the option to change the tweets displayed from this to either all of twitter or only people the user follows. Also usernames from forums around the web are taken and then searched on twitter in order to be able to follow the user.

2.1.1 Data Sources and Destinations

Data	Source	Destination
Username	User	Twitter API
Password	User	Twitter API
Home Timeline	Twitter API	Web Interface
Query	User	Twitter API
Query Results	Twitter API	Web Interface
Username (From other user)	Online Forum	Twitter API
Web Link (Resource)	Tweet	Bookmarks (Database)

2.2.2 Algorithms

Algorithm used for finding users on external websites:

```
IF User HAS Twitter Credentials THEN
     SEARCH Twitter FOR User
     IF User HAS Twitter THEN
          FOLLOW User
     END IF
END IF
```

Algorithm used for searching for interesting resources in timeline:

```
FOR tweets IN home_timeline THEN
```

```
     Resource_Useful ← False
     IF tweet CONTAINS 't.co/%' THEN
          Link ← 't.co/%'
          IF Link IS Useful THEN
               Resource_Useful ← True
          ELSE
               Resource_Useful ← False
          END IF
          IF Resource_Useful THEN
               ADD Link TO Bookmarks
          END IF
     END IF
END FOR
```

Algorithm used for finding interesting resources from a search:

```
OUTPUT "Enter search term: "
INPUT Query
Results ← Twitter API search call (Query)
FOR tweets IN Results THEN
     Resource_Useful ← False
     IF tweet CONTAINS 't.co/%' THEN
          Link ← 't.co/%'
          IF Link IS Useful THEN
               Resource_Useful ← True
          ELSE
               Resource_Useful ← False
          END IF
          IF Resource_Useful THEN
               ADD Link TO Bookmarks
          END IF
     END IF
END FOR
```

Algorithm used for searching tweets:

```
OUTPUT "Enter search term: "
INPUT Query
Results ← Twitter API search call (Query)
FOR tweets IN Results THEN
     OUTPUT tweets
END FOR
```

2.1.3 Data Flow Diagrams

Data Flow Diagrams

Key

Data Source or Destination
→ A line in the left corner indicates multiple appearances in DFD

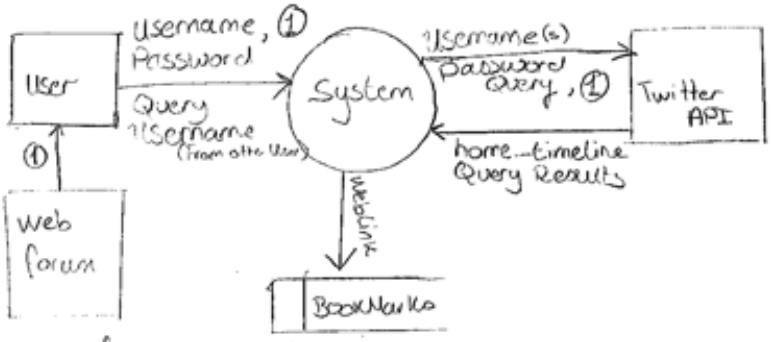
Process
→ Edges are rounded

Data Store

Data flow
→ Labelled with data being exchanged

For the Current System

Fig 1.1 Context Data Flow Diagram of Current System



①Username (from other user)

Fig 1.2 Level 1 Data Flow Diagram of logging into twitter

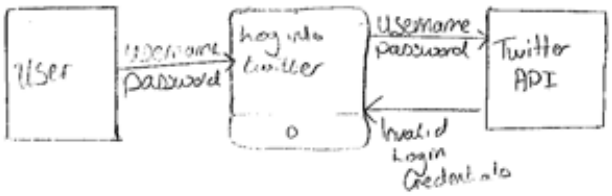


Fig 1.3 Level 1 DFD of Searching home timeline in Current System

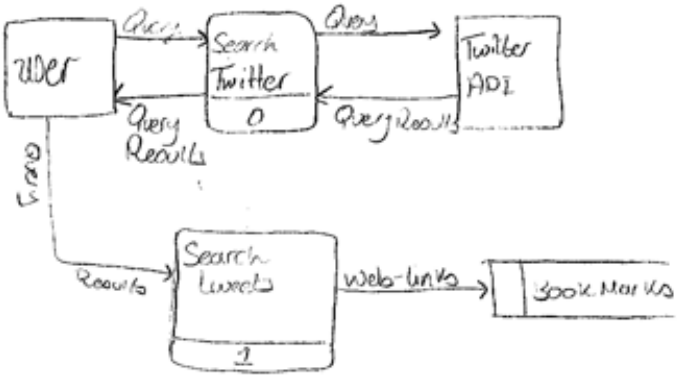
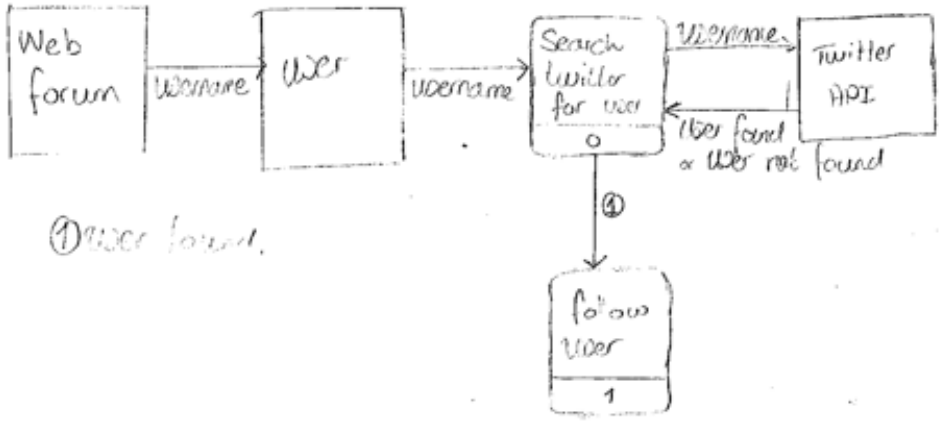


Fig 1.4 Level 1 DFD of Searching for users on External Websites



①User found.

2.2 The Proposed System

2.2.1 Data Sources and Destinations

In the program certain data will have required by the user. The data needed here would be the the users Twitter details (A username and a password) which, ideally, would only be needed once and the account would be tethered to the app thus allowing ease of use. The data provided by these details would be provided through the user interface of the application and would have to be stored securely somewhere on the system. Once the user has authenticated their details other information would have to be gathered from Twitter itself, using an implementation of the Twitter API in Python, regarding the tweets on the users public timeline. The Twitter API returns all data request in decoded JSON files, which in python, with one implementation of the API into python, appear as arrays while an option for XML data is available. The search of the users timeline could be processed by getting all the tweets from the users timeline and then searching the tweets locally on the computer; it could also be done by interacting with the Twitter API directly through a 'GET search/tweets' command. For the purpose of the analysis the first method shall be assumed. Furthermore links from Tweets will be parsed and aggregated so that a database or system to keep bookmarks is kept. Also the Twitter API will provide suggested followers to the user of the application and specify people related to computing.

Data	Source	Destination
Username	User	Secure file on computer
Password	User	Secure file on computer
User Timeline / Home Timeline	Twitter API	Computer / In Storage
Query	User	Twitter API
Returned tweets from query	User Timeline / Twitter API	Computer / In Storage
Graphic information	Program	Computer / In Storage
Web-Link	Tweet	Database
Bookmark	Program	Database
Suggested Followers	Twitter API	Database

2.2.2 Data Flow Diagrams

Data flow Diagrams

Fig 2.1 Context data flow diagram for proposed system

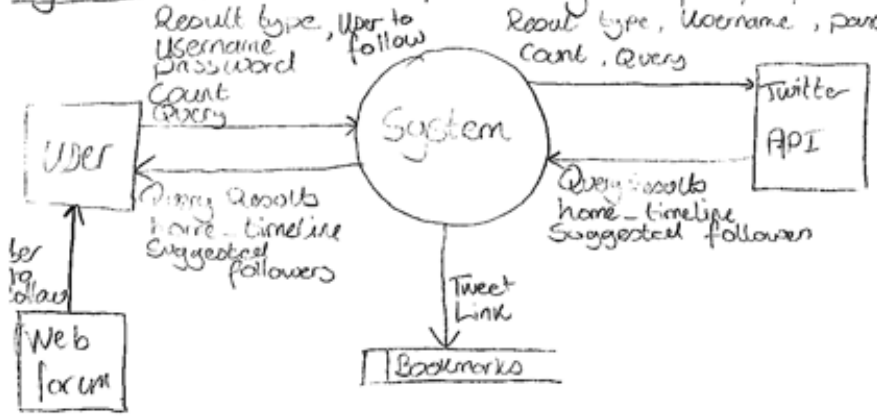


Fig 2.2 Level 1 DFD for logging into twitter

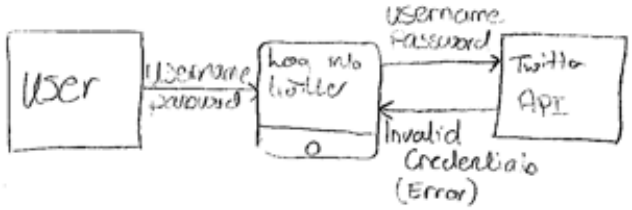


Fig 2.3 Level 1 DFD for obtaining home_timeline

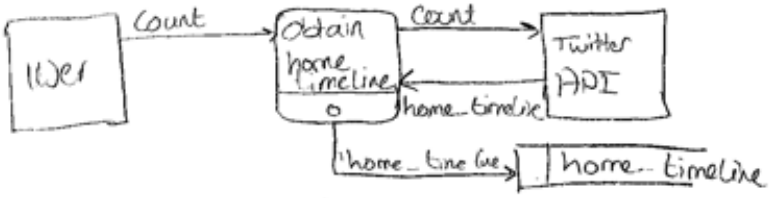


Fig 2.4 Level 1 DFD for Searching home_timeline tweets

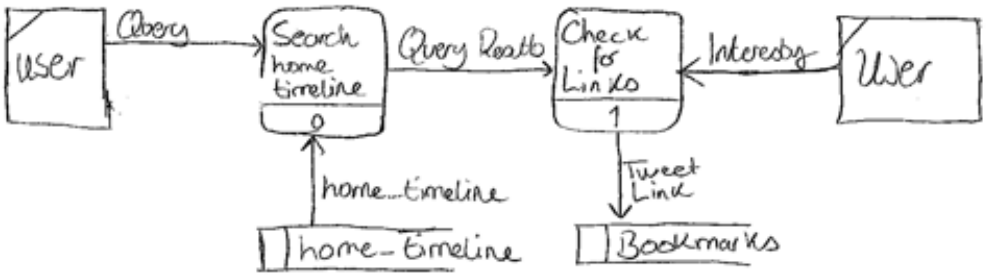


Fig 2.5 Level 1 DFD for getting Query Results from Search

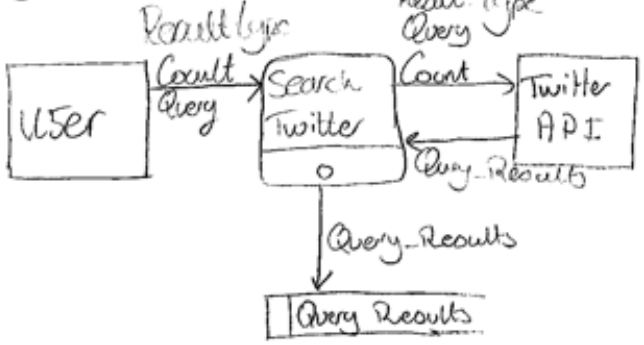


Fig 2.6 Level 1 DFD of Sorting Query Results for Links

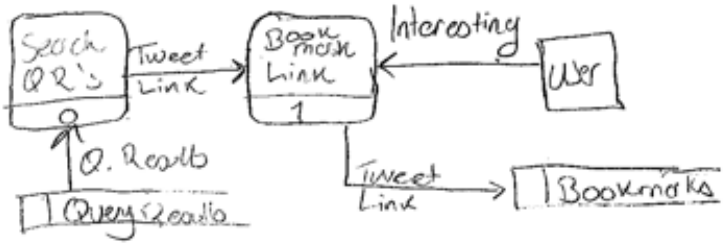


Fig 2.7 Level 1 DFD of Producing Info on tweets + Trends.

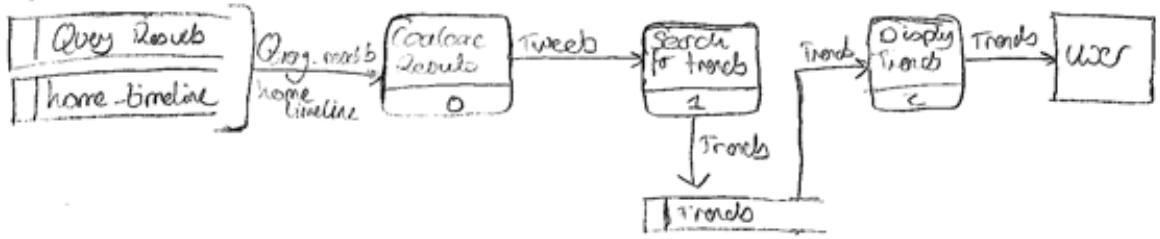
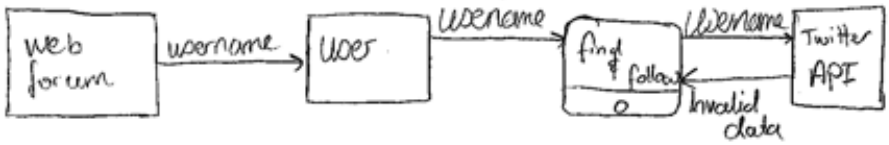


Fig 2.8 Level 1 DFD Showing finding user and following user



2.2.3 Data Dictionaries

Name	Data Type	Length	Example	Comment	Size
Username	String	max. 15 chars	JackieG		15 - 70 Bytes
Password	String	No limit	T1nb7Js		-
Tweet	String	140 chars	“This is a tweet”	Tweets are usually ~200 Bs	140 - 840 Bytes
(Tweet user name)	String	Max. 15 chars	@MrHornett French	Included in Tweet from API	15 - 70 Bytes
(Tweet Screen Name)	String	Max. 20 chars	John Smith	Included in Tweet from API	15 - 70 Bytes
count (GET user_timeline)	Integer	max. 3 digits	30	Limited to 200	28 Bytes
home_timeline	array	max. 800 entries	-		104 - 7064 Bytes
Query	String	Shouldn’t be more than 140 chars	‘Computing’	1 word or possibly a phrase	1 - 840 Bytes
count (GET search/tweets)	integer	max. 3 digits	100	Used to limit number of tweets returned with a search call.	28 Bytes
result_type	string	max 7 chars	mixed, recent, popular	Used to filter what type of tweets to return from search	~ 42 Bytes
Query Results	array	max 100 items	-		200 - 920 Bytes

Name	Data Type	Length	Example	Comment	Size
Link	String	~20 chars*	‘t.co/gLP0Ucg’	Link size varies accoriding to the URL shortener used. Example uses t.co.	120 - 126 Bytes max.
link_needed (‘Interesting’ in DFD’s)	string	1 character	‘y’ , ‘n’	Used to verify bookmarking of link	50 Bytes

2.2.4 Volumetric

To calculate a volumetric of the system based on the above data dictionary we add up the estimated amount of space that is used up by each piece of data. All data sizes for the strings is based on using ‘UTF-8’ encoding where each character can take between 1 - 6 bytes on the system. As there is only one instance of the username and password these values won’t have to be expanded for a volumetric however as the number of tweets called from the API can change a maximum and minimum variable should be drawn up regarding the size of the whole system. The count call for GET search/tweets is limited to 100, defaulted at 15. There is some data included with the Tweet such as a username and a Screen Name (or Real Name) that add to the size of the Tweet; in the dictionary these have been separated with the internal data showed by parentheses.

By that logic the system should use between 758 Bytes and 10’140 Bytes. However this is only an estimate.

3. Objectives

3.1 General Objectives

- User must be able to log into twitter
- Easy to understand intuitive interface
- Must receive tweets from the Twitter API
- Must be able to search tweets from timeline
- Must be able to store information from tweets such as links
- Must be able to follow trends
- Must obtain or generate suggested followers

3.2 Specific Objectives

- The user must be able to provide their own credentials to log onto twitter
- These credentials must be stored securely on the computer

- The user must be able to log out and also to stay logged in between sessions
- The interface must be easy to navigate and allow for ease of use
- Tweets must be received from the API
  - Whether this is through a call to receive the whole timeline
  - Or to receive only tweets pertaining to a search query

The user must be able to search tweets, however they were obtained, using a predefined query from the user

Results must then be displayed back to the user, through the graphical interface

Information from tweets, such as links to external resources, must be parsed from the tweets

These must then be stored in a database on the computer and must be easy to navigate by the user in order to find these resources.

The system must allow for the user to specify trends to follow and to find common themes between tweets in these trends (such as multiple people twitting about ‘Linux Con’ should be seen by the user)

The system must also be able to find these in the users timeline, among the users they follow, and not just the whole of twitter

The system must be able to get suggested followers from the Twitter API

The system then must either:

Display those follows intuitively to the User

Or search these follows for those specific to computing and then display them to the User

3.3 Core Objectives

The system *must* have an interface.

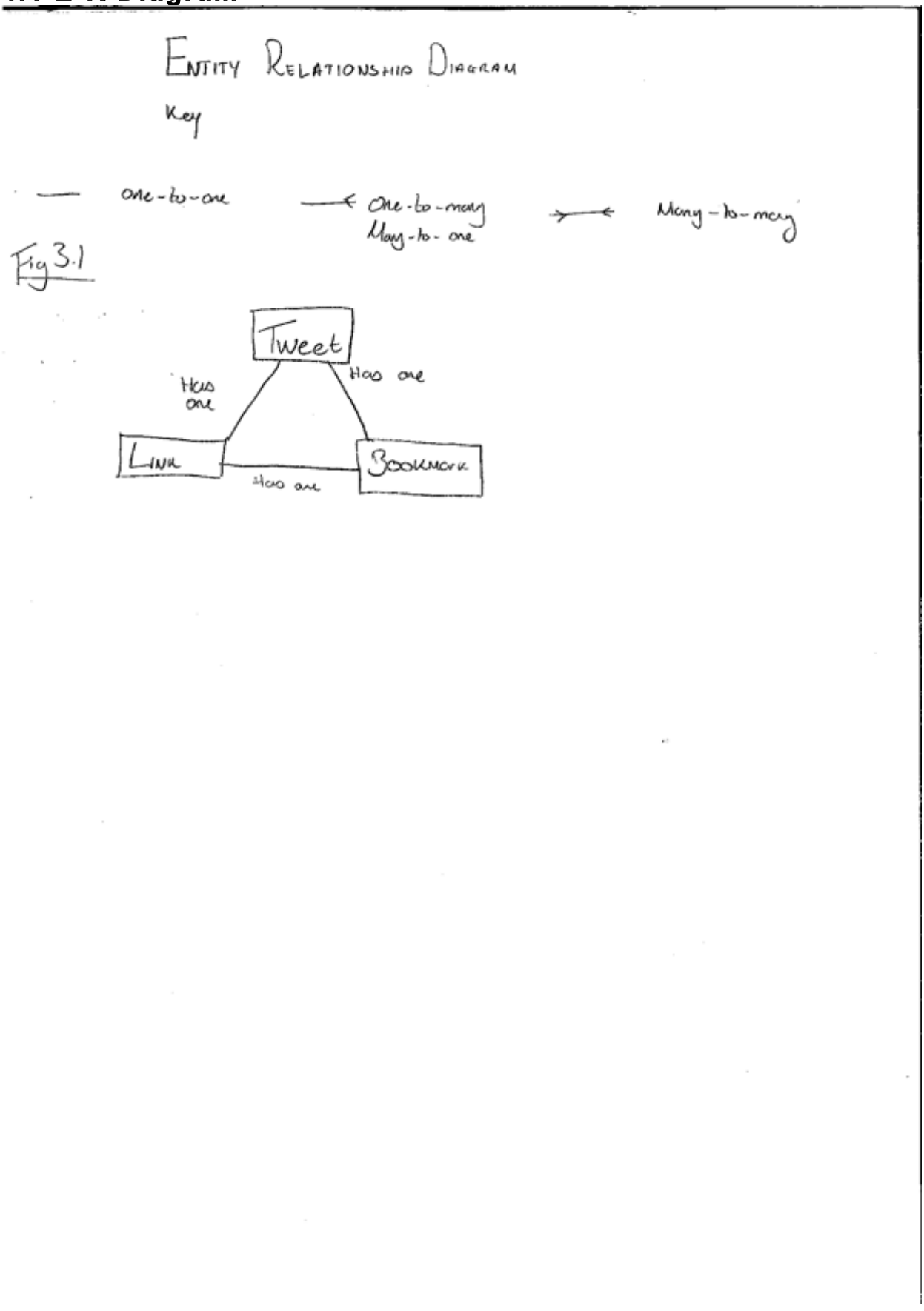
- The user *must* be able to log into twitter with their own Username and Password.
- The system *must* be able to connect to the Twitter API and be able to store the tweets retrieved.
- The system *must* allow the searching of tweets.

3.4 Other Objectives

- It could be possible for the system to display graphs or other graphical representations about the queries from the user.
- It could be possible to add functionality for multiple user accounts on the system.
- It could be possible to add support for external services such as ‘Del.icio.us’ or ‘Instapaper’ as a form of online bookmarking.

4. E-R Diagram and Descriptions

4.1 E-R Diagram



4.2 Entity Descriptions

Tweet(TweetID,user,screen_name,tweet_text,*LinkID,BookmarkID)*
Link(LinkID,link,site_URL,description,*TweetID*,*BookmarkID*)
Bookmark(BookmarkID,title,description,*TweetID*,*LinkID*)

5. Object Analysis

5.1 Object Listing

Tweet
Home Timeline
Query Results
User
Bookmark
Link

5.2 Relationship diagrams

Object Analysis

Fig 4.1 Simple Association Diagram

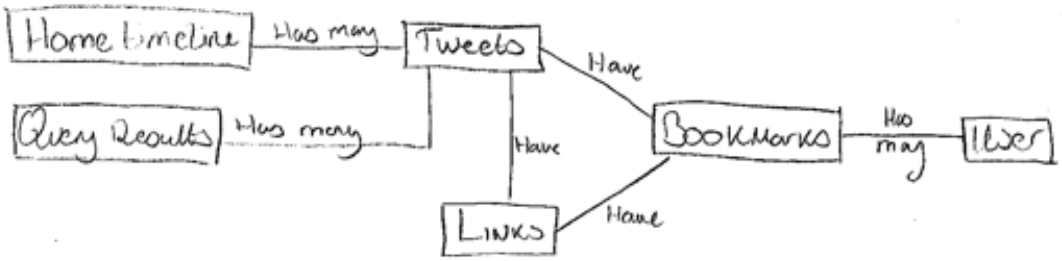


Fig 4.2 Association diagram with multiplicity

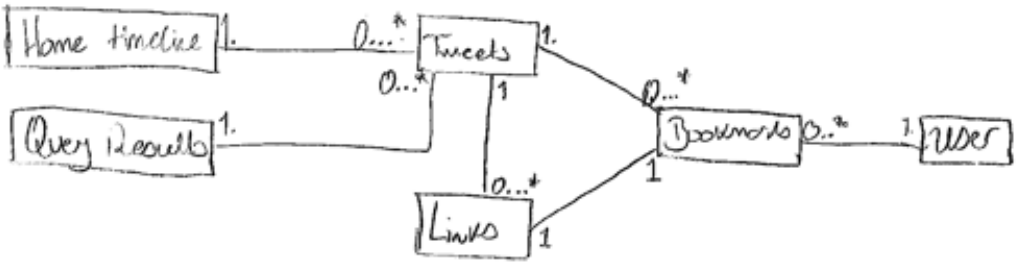
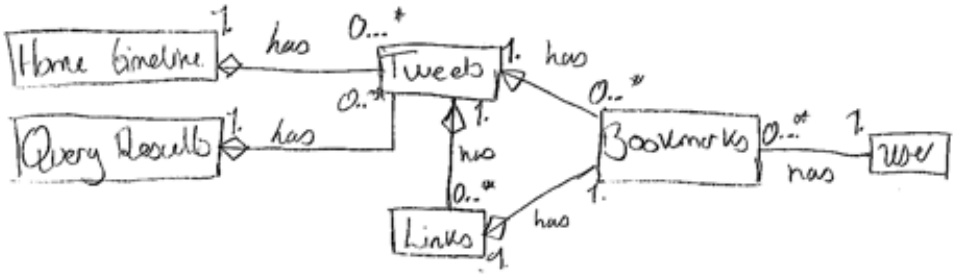


Fig 4.3 Association, Aggregation & Inheritance Diagram



5.3 Class Definitions

Home Timeline	Query Results	Tweet	User	Bookmark	Link
Tweets Count	Tweets Count Query	Username ScreenName TweetText	Username Password	Tweet Link User	Tweet Link Title URL
getTweets getCount storeHomeTimeline	getTweets getCount getQuery storeQueryResults	getUsername getScreenName getTweetText	getUsername getPassword storeSecurly login	getTweet getLink getUser storeBookmark	getTweet getLink getTitle getURL

6. Constraints

6.1 Hardware

- Has to run on latest version of OSX, Currently on a MacBook Pro.
    - This doesn't pose any great restraints on the development, only when packaging the application.

6.2 Software

One constraint in making this system is the Twitter API (REST API) itself as it restricts the amount of data one person or application can obtain at one time.

- Twitter API functions
    - When calling home_timeline maximum of 800 tweets per home_timeline, 200 per GET tweets/home_timeline call
    - When performing a GET search/tweets a maximum of 100 tweets per page, maximum 450 search request per app, 150 search requests per person.
    - All responses are sent in 'json' format, which has to be decoded.
    - Having to keep track of which tweets have already been received and queried. Functionality for this is provided within the latest REST API provided by Twitter.

6.3 Time

- Deadline of 14th of February 2014
    - Set by client

6.4 User Knowledge

Mr McNicol is proficient with computers and thus user knowledge of computer systems should pose no real constraint on implementation.

6.5 Access restrictions

- The client is the only person that will be using the application and thus only one level of user is needed
    - The users Username and Password must be stored safely, though.

7. Limitations

7.1 Areas which will not be included in computerisation

Areas most likely not to be included in the computerisation are:

- Integration with other online services, such as Instapaper or 'Del.icio.us'
- The creation of a graphic representation of what may be trending.

7.2 Areas considered for future computerisation

If it was the case there were unlimited resources for the making of the system then these could have been included:

- To include the aforementioned areas in the previous section above
- To add features for deeper integration with Twitter as a whole, such as allowing complete management of ones Twitter account.

8. Solution

8.1 Alternative Solutions

There are multiple solutions to the problems the client has with his current system:

- To use other Web Interfaces currently available on the internet
    - These will have additional features other than the twitter web interface that may solve some of the client's problems.
    - Also the client would be able to use the app on any system that is connected to the Internet.
    - This is also a very low cost way of using a new system.
    - However limitations will include that it is not tailored to the clients needs and desires and thus falls short on more specific requirements
    - The interfaces of these applications are only just reaching desktop application standards and thus are not as streamlined and intuitive
    - Also the data gained and stored is not necessarily what the client wants from the application and thus creates more unnecessary problems
    - Furthermore I am not able to program in HTML5 nor am I familiar with other web-app based languages
- To use a PyQt GUI application that is bespoke and tailored to his needs
    - The advantage being that most of his problems with the current system will be resolved as the system is tailored to his needs
    - Further it is a language that I am comfortable with using and developing software in
    - Also the data gained and manipulated is specifically what the client wants from the proposed system
    - The GUI is also of a higher standard than some web-apps and thus should be easy to navigate and to understand than web-based alternatives.
    - The disadvantage being the time needed to develop the software

    - Also the computer will need more resources to run the application than simply using a web-browser to run the app.
- To develop a cloud-based application with a web-interface that works similarly to the PyQt GUI.
    - Advantages include the fact it takes lees hard-drive space as it would be hosted online.
    - It also inherits some of the benefits of the PyQt GUI as it is also bespoke and fixes specific issues.
    - However it also relies on a web-interface.
    - It also requires a large start-up cost as it need to be stored on an online server, so costs involving buying server space and extra maintenance costs.

8.2 Justification of chosen solution

The solution chosen for the proposed system is to create a PyQt GUI application that may utilise a SQL database for bookmarking. The reason for choosing this system is that Python and PyQt are familiar language to programme in and so it should reduce the time needed to make the application. Also it is able to utilise the computers resources much more effectively than web-based applications, also being cheaper than implementing an online server-based web-interfaced system. Furthermore with the application being bespoke rather then general purpose I can tailor the system to deal specifically with the client's problems and thus reduce the creation of more issues with the current system.

Design

1. Overall System Design

1.1 Short Description of the mains parts of the system

Twitter Searching System.

- User Login Interface
- Initial Search Dialogue
- Tweet Search Results Interface
- Bookmark Creation Interface
- Bookmark Management Interface
- Suggested Users Interface
- Trending Tweet Interface

User Login Interface
- The user will be presented with a place to log into Twitter and thus be able to obtain tweet data from twitter by authenticating the application with the Twitter API
- The user will have fields for the Username and Password to be entered.
- The details the user has provided shall be stored in a hidden file in the computer so that it is secure.
- These details shall be sent to Twitter to interact with the API in order for the user to be logged in.

Initial Search Dialogue
- This interface will allow the user to enter search queries to find tweets from Twitter
- A space for entering the the search query shall be available.
- There shall also be a space for entering the number of Tweets to be searched for. This will have to be validated against the maximum number of tweets that can be retrieved in a request by the Twitter API.
- A pull down menu with the type of results to be shown can also be displayed, containing ‘Mixed’,‘Popular’, or ‘Recent’, as these are the types if tweets that can be retrieved in a search.
- This data shall be sent to the twitter API and storing the returned data on the system in order to be displayed to the user in the Tweet Search Results Interface, that should automatically be switched to for the user to see the resulting detail.

Tweet Search Results Interface
- This interface will provide a way to view the results of the tweets retrieved from the search stared by the user in the Initial Search Dialogue.
- It should show each tweet, their user, and other important information regarding each tweets.
- The lost of tweets should be scrollable.
- Their shall be buttons that allow for Tweets to be bookmarked, and this should open a Dialogue to add information needed for a bookmark.

Bookmark Creation Interface
- This shall have the facilities in order for the user to be able to create a bookmark of a tweet and the links that are contained inside the tweet.
- The information that is about to be bookmarked shall be shown to the user for clarity of what shall be saved.

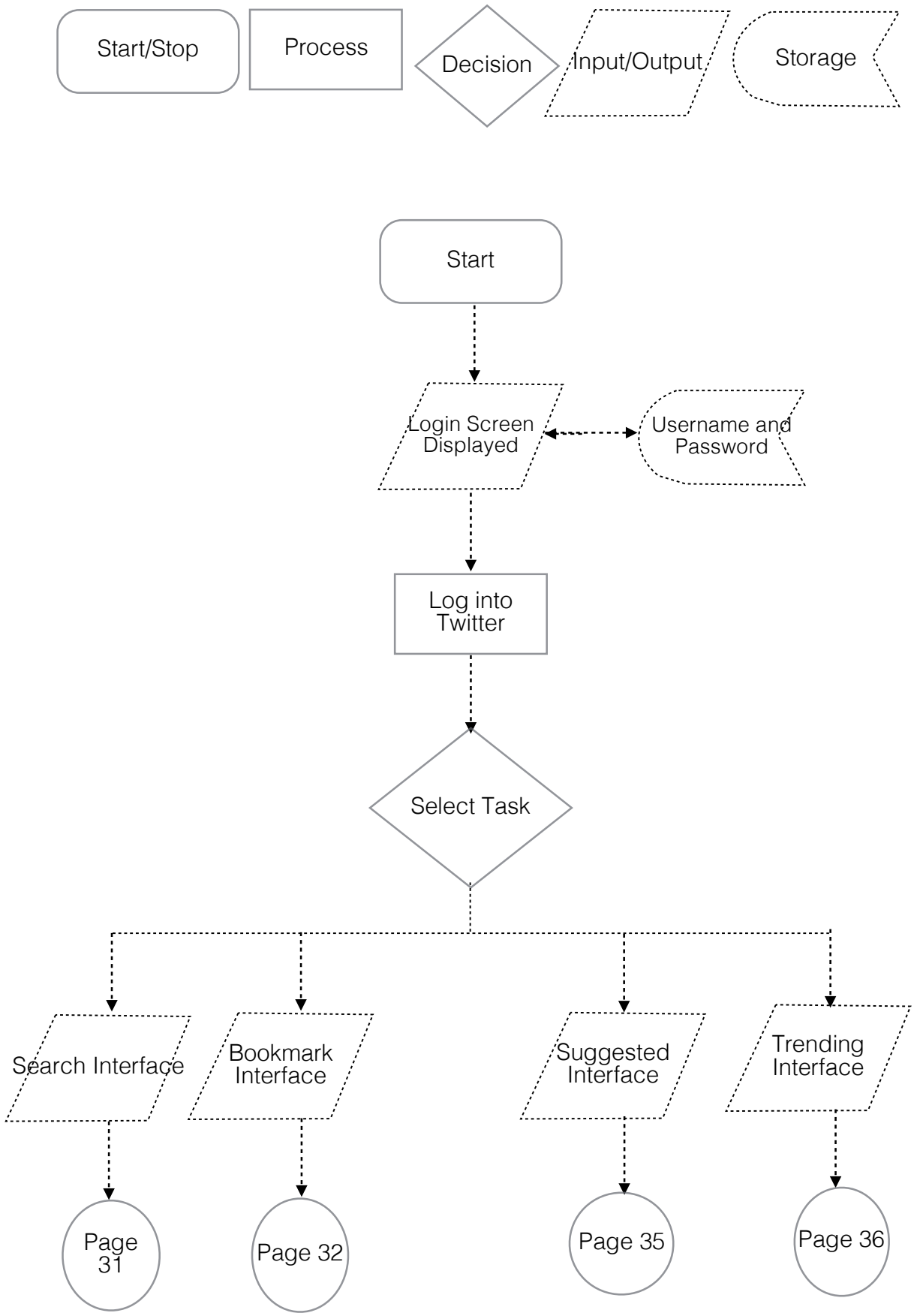
- There shall be some space to enter change established details or to add a separate description to the bookmark.
- This data shall be added to the database, with a confirmation message or error message, depending on whether there is an error in the users input or not.

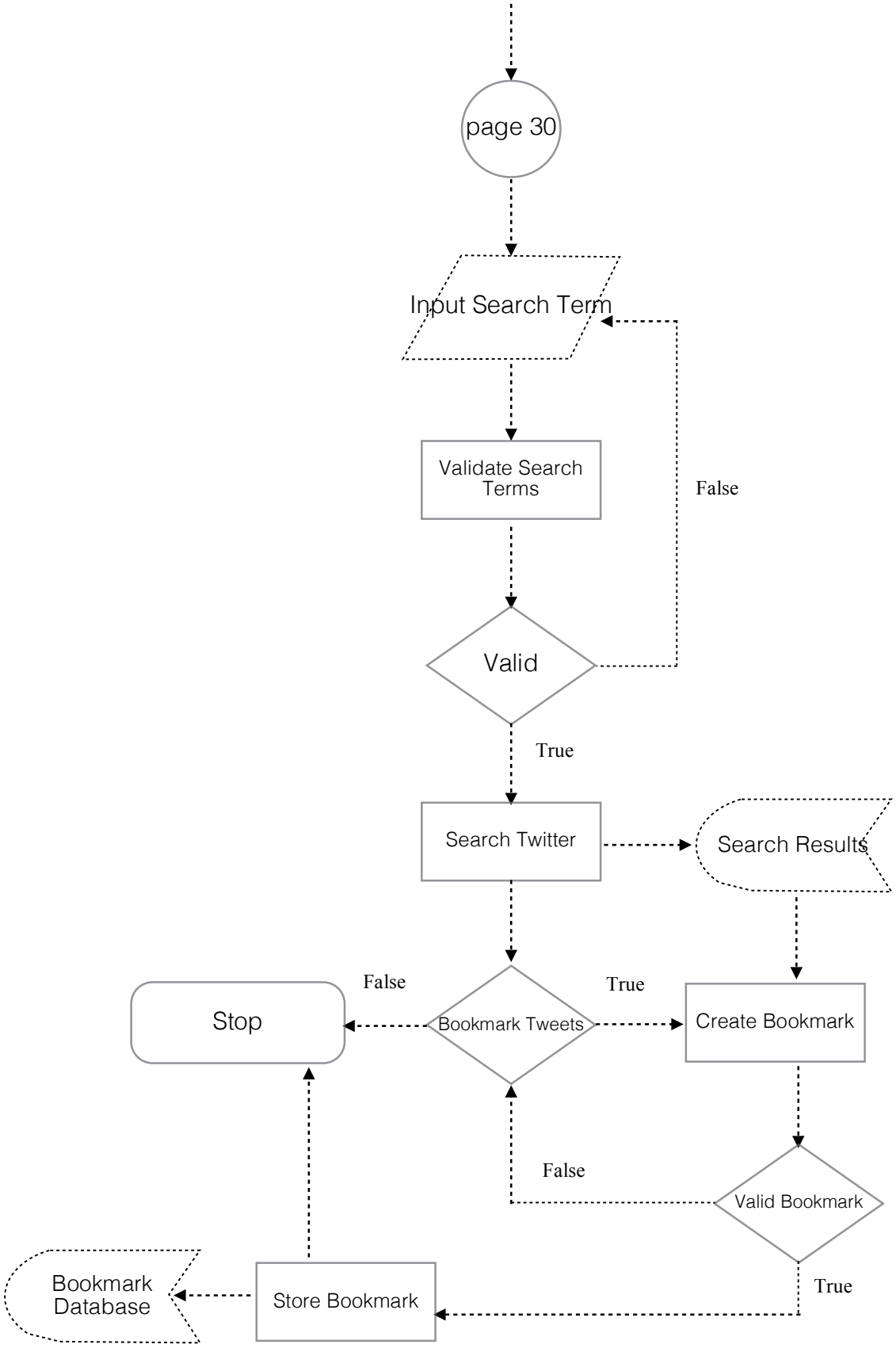
Bookmark Management Interface
- This interface shall allow the user to manipulate the databases of Tweets and Bookmarks and Links.
- It shall have a table view of each databases.
- This shall allow the user to see any changes made to the database and shall give the user the option to delete entries already in the table.
- There shall be a ‘commit changes’ or ‘submit’ button that shows a dialogue asking whether the user is sure to make the changes proposed by the user.

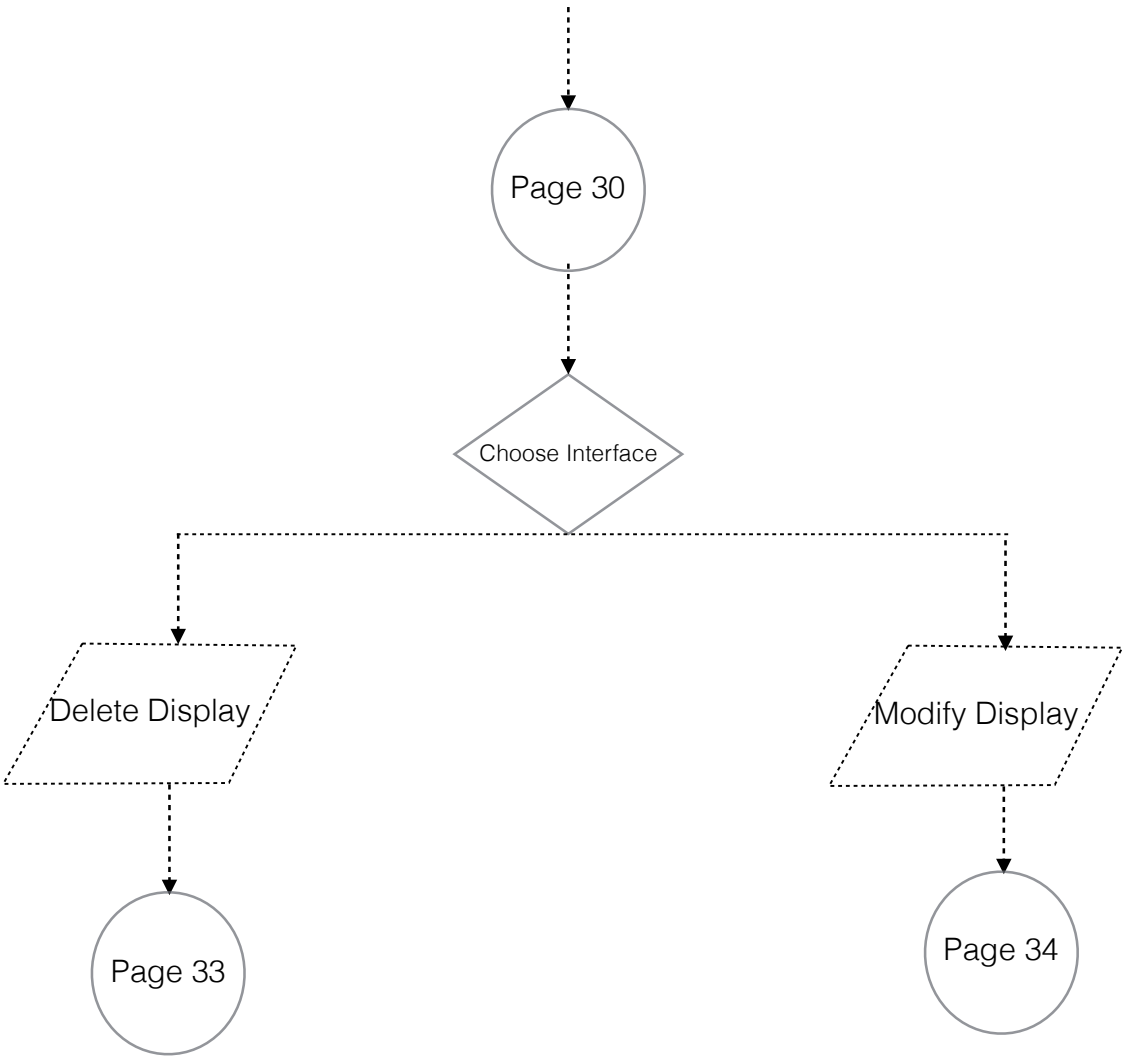
Suggested User Interface
- This interface shall have the list of users that are suggested to the User, being retrieved from the Twitter API and possibly generated in the Application.
- An option to follow users and to refresh shall be shown to the user.
- Any changes should be pushed to the Twitter API accordingly.

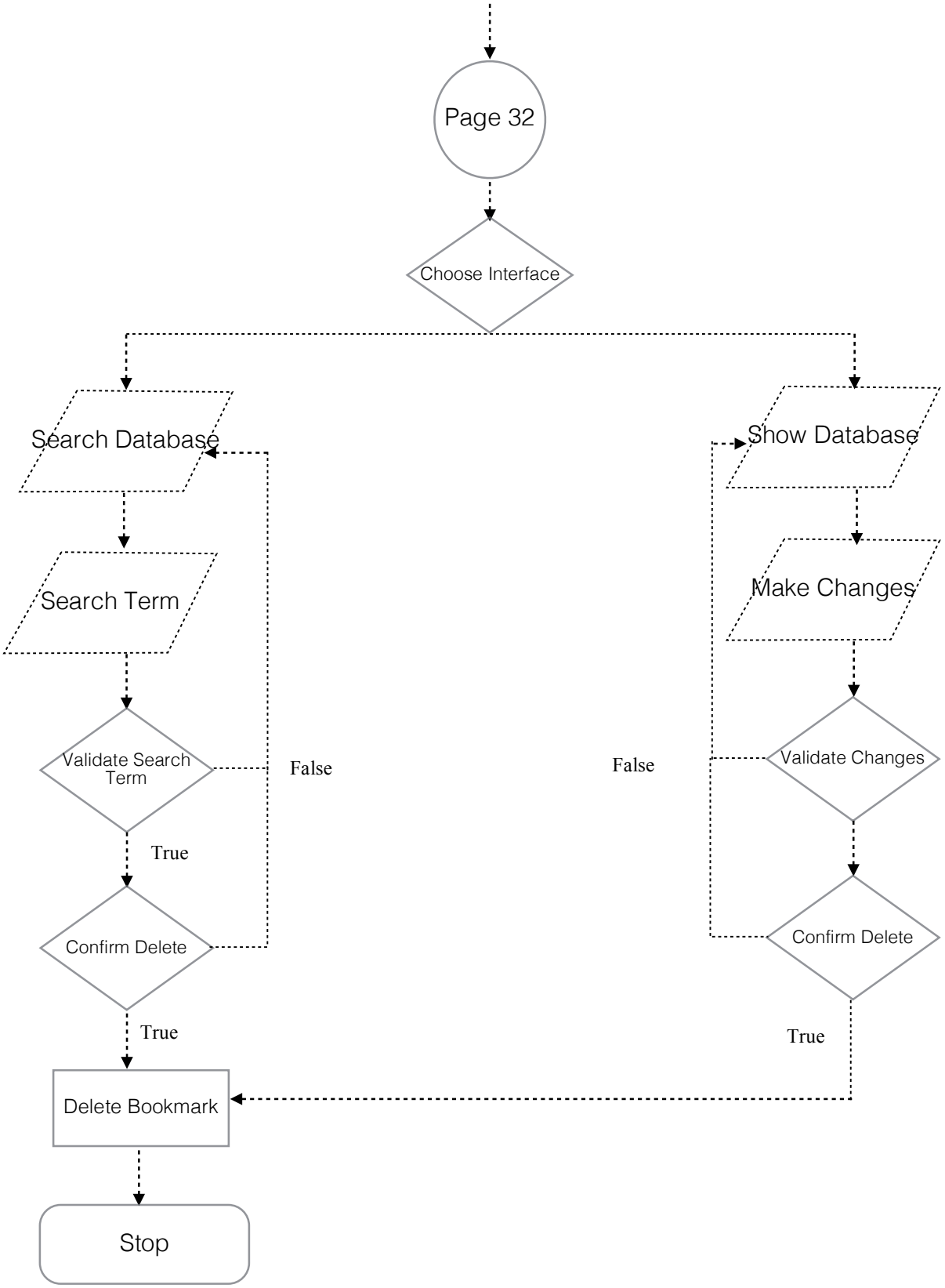
Trending Tweet Interface
- This interface shall allow the User of the application to be able to find out what tweets are trending on twitter based in their popularity. This would, whether it is through using the Twitter API or searching for trends in the user's Home Timeline itself.
- There should be a list that displays the tweets to the user and thus allow them to make decisions regarding the tweets. Such as the bookmarking that is defined above.
- Therefore there should be the tools needed to show how to bookmark tweets that are needed to be bookmarked by the user.
- It might therefore be useful for a searching function on the interface to allow the user to search within the trending tweets sent back from the Twitter API.

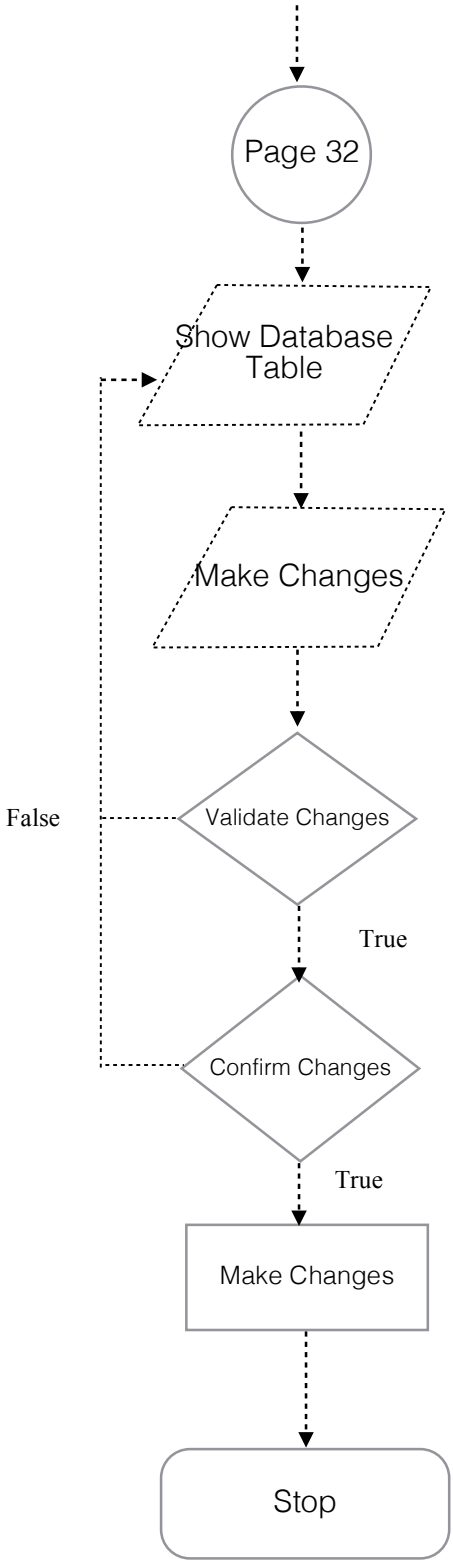
1.2 System flowcharts showing an overview of the complete system

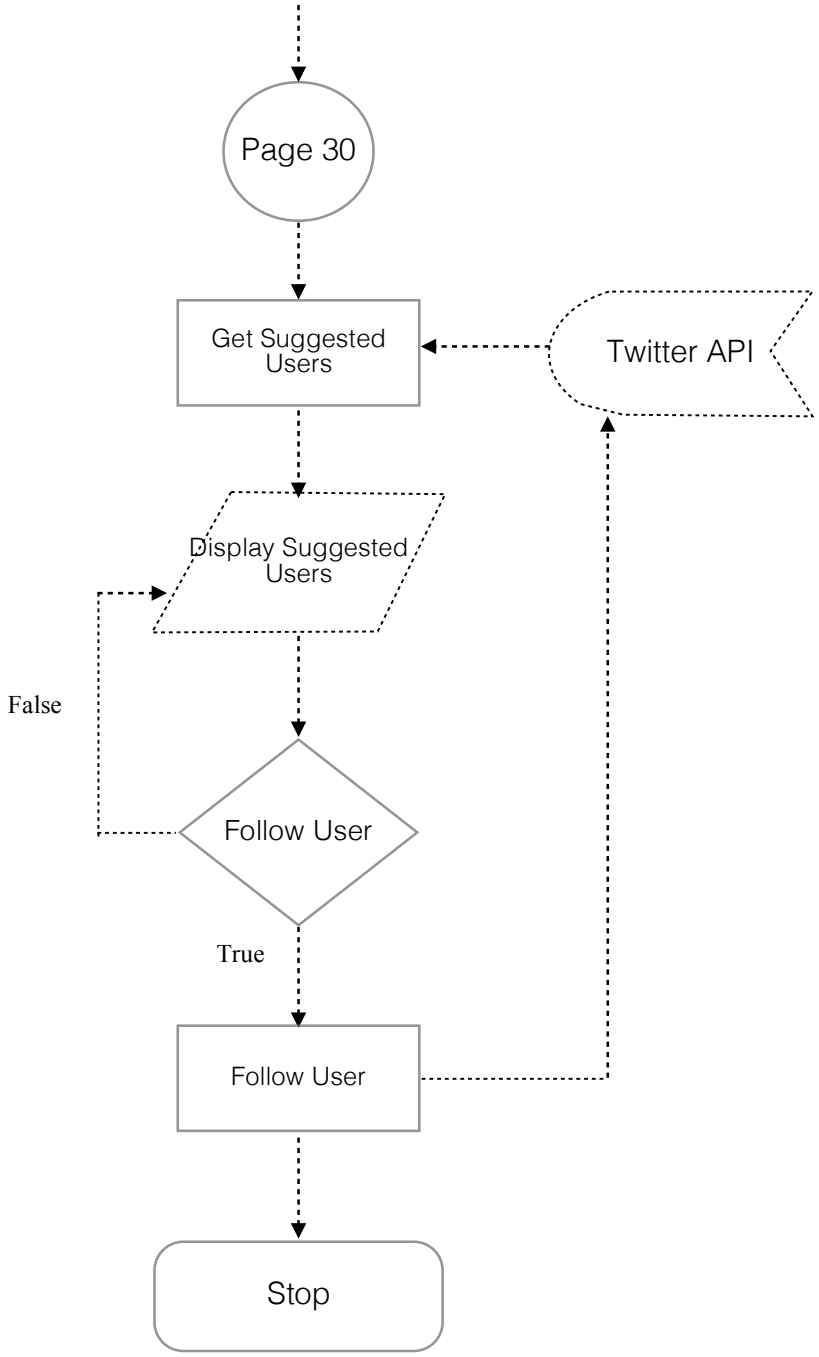


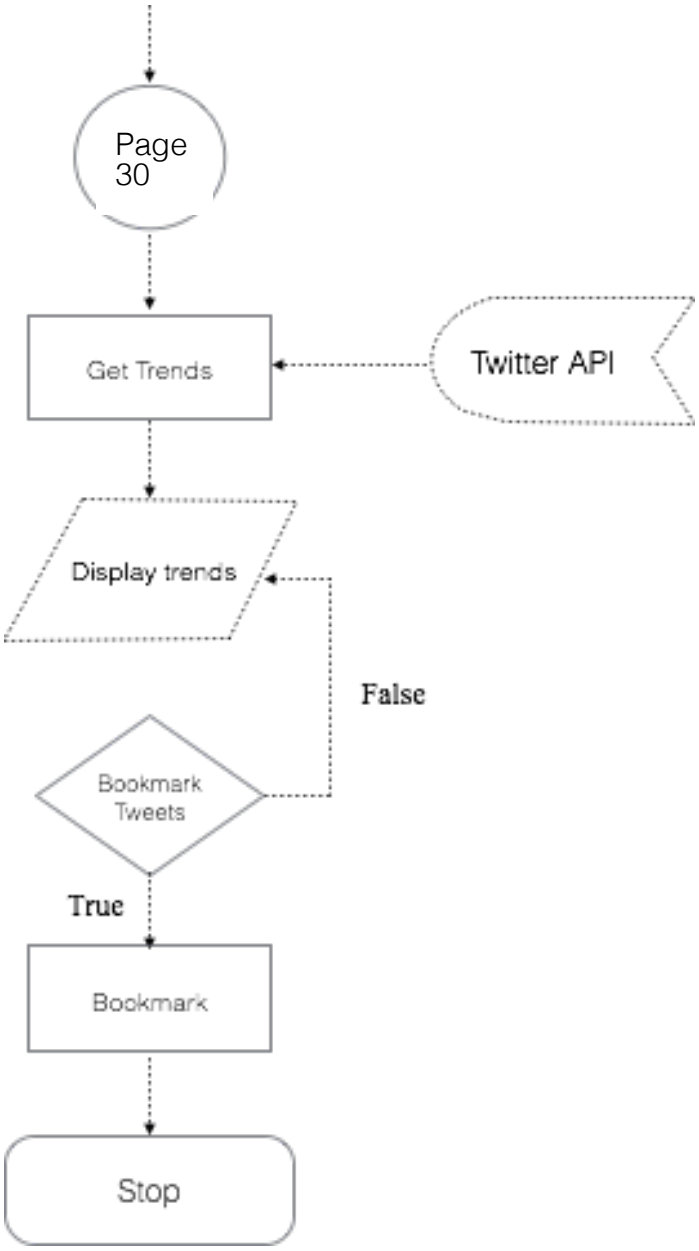

page 30
Input Search Term
Validate Search Terms
False
Valid
True
Search Twitter
Search Results
False
True
Stop
Bookmark Tweets
Create Bookmark
False
Valid Bookmark
True
Bookmark Database
Store Bookmark


Page 30
Choose Interface
Delete Display
Modify Display
Page 33
Page 34

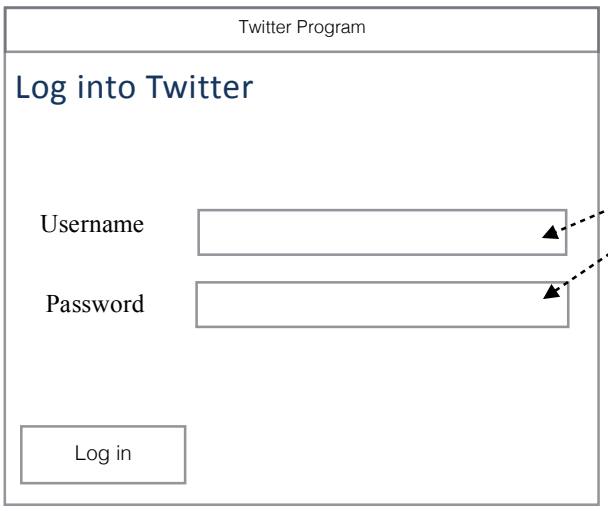

Page 32
Choose Interface
Search Database
Search Term
Validate Search Term
False
True
Confirm Delete
True
Delete Bookmark
Stop
Show Database
Make Changes
Validate Changes
False
Confirm Delete
True


Page 32
Show Database Table
Make Changes
Validate Changes
False
True
Confirm Changes
True
Make Changes
Stop

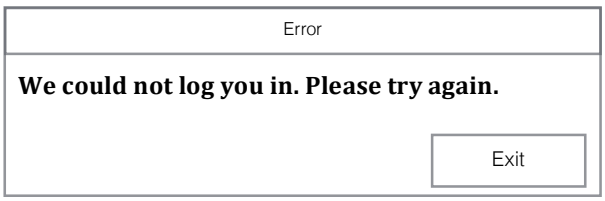

Page 30
Get Suggested Users
Twitter API
Display Suggested Users
False
Follow User
True
Follow User
Stop


Page 30
Get Trends
Twitter API
Display trends
False
Bookmark Tweets
True
Bookmark
Stop

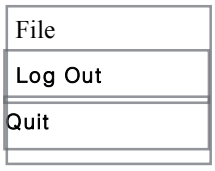
2. User Interface Design



These are the text edits that allow the user to place their login credentials.

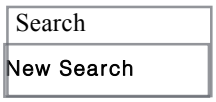


This is the window that would display if the client could not log the user into Twitter. Allowing a re-try

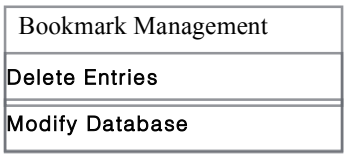


This allows the user to quit and return to the login screen

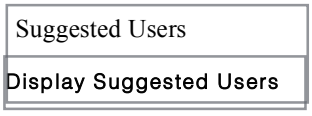
This allows the user to completely quit the program.



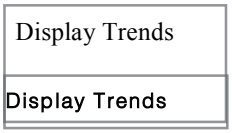
Page 38



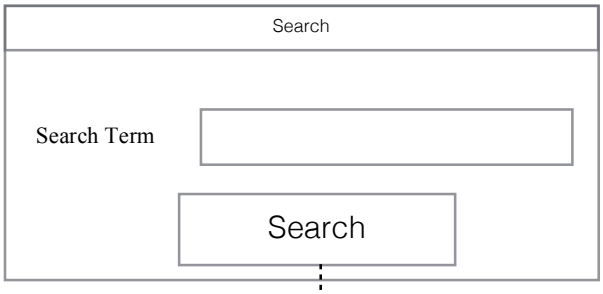
Page 39
Page 40

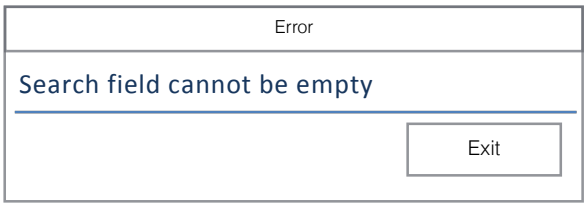


Page 42

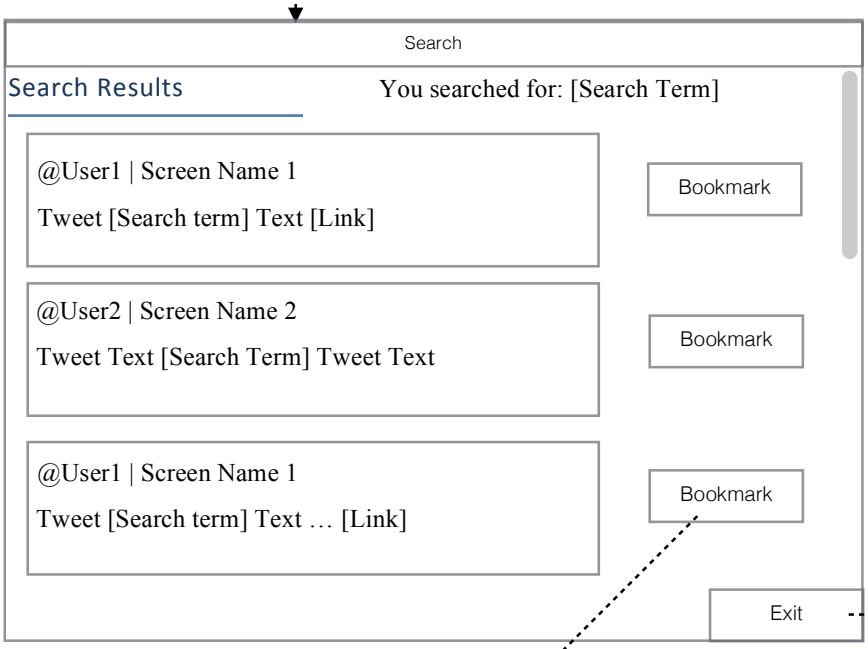


Page 41



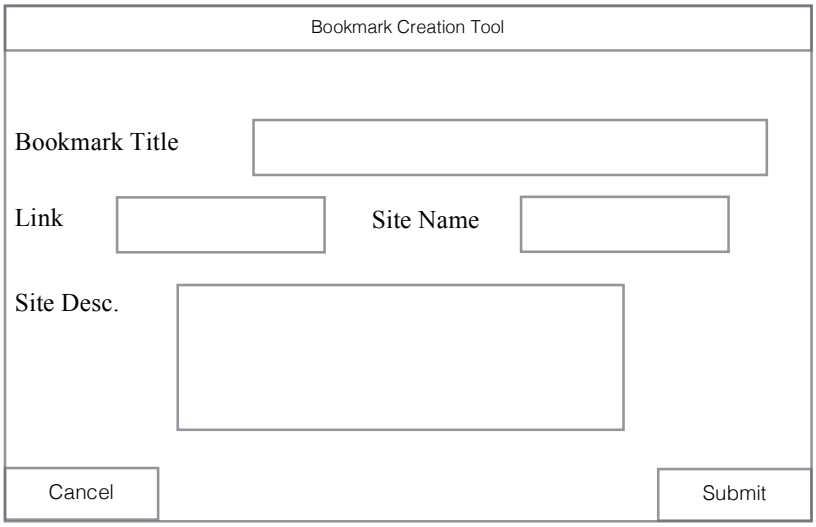


This would be displayed is the search field is empty

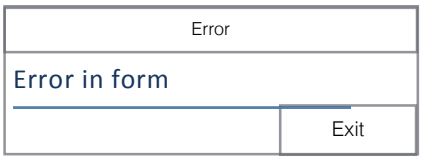


This interface shows the user all of the tweets returned from the search, thus allowing them to bookmark selected tweets.

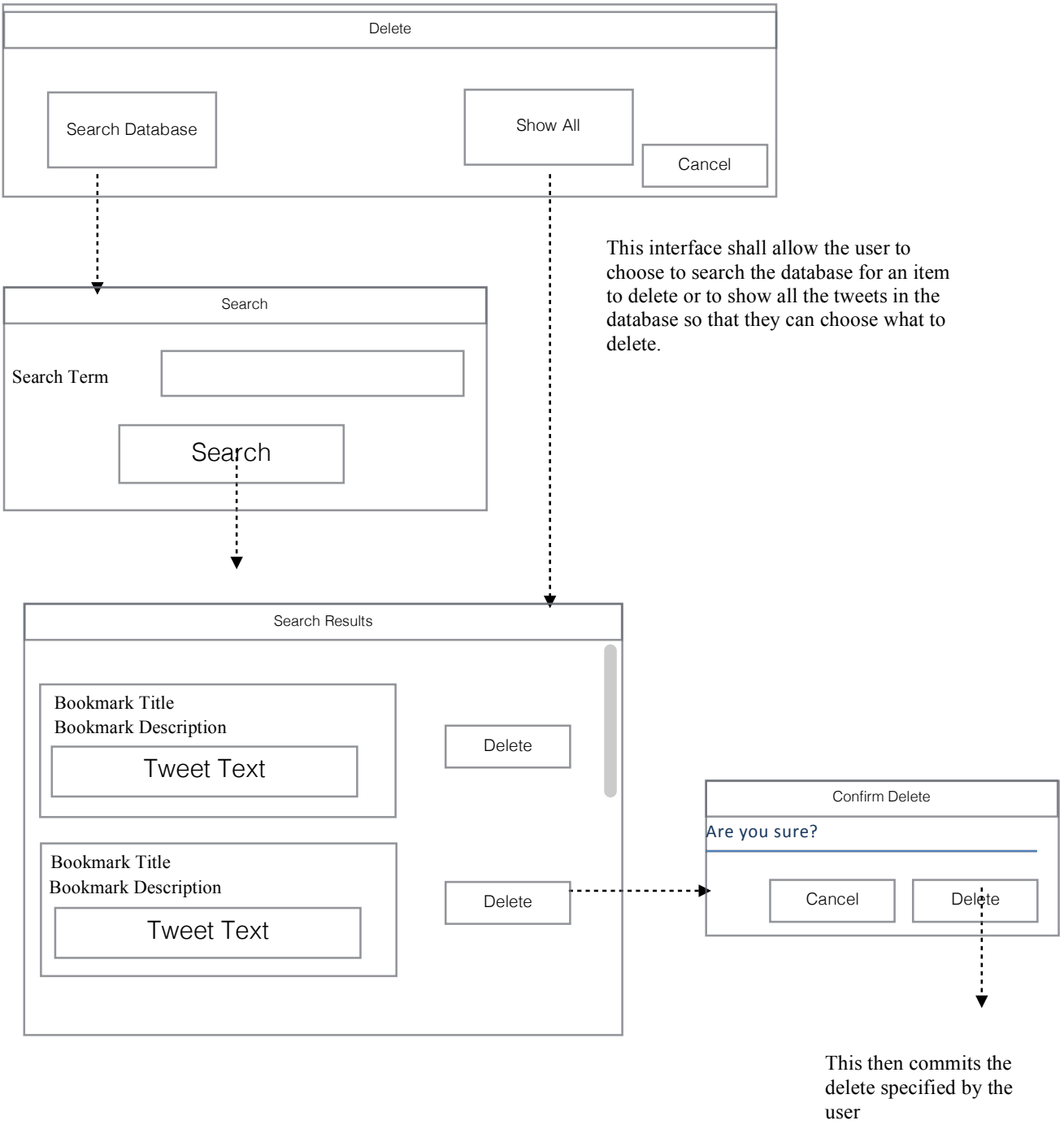
Returns to main interface



This display is used to add bookmark information to the database in the system.

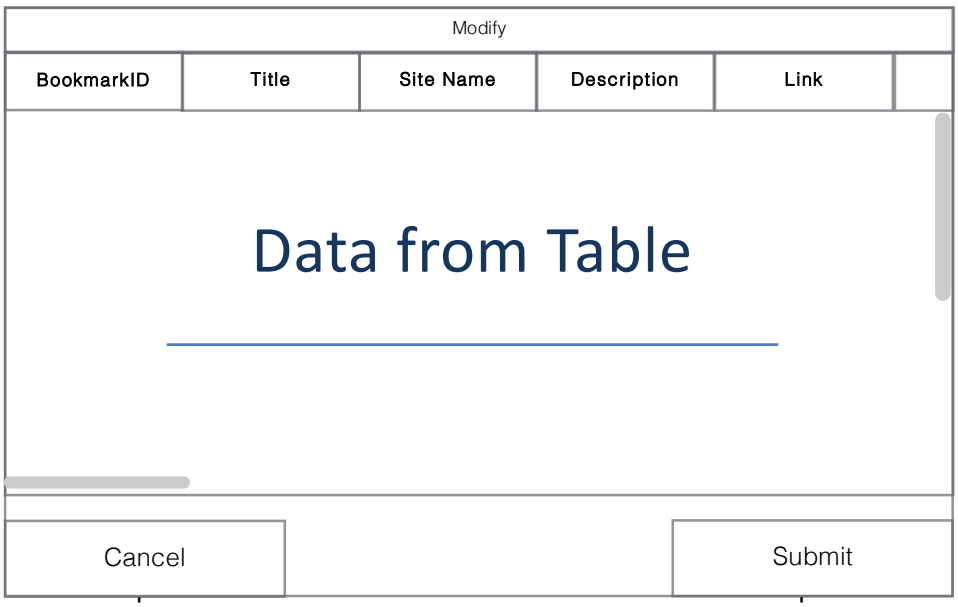


This is displayed when the form isn't valid



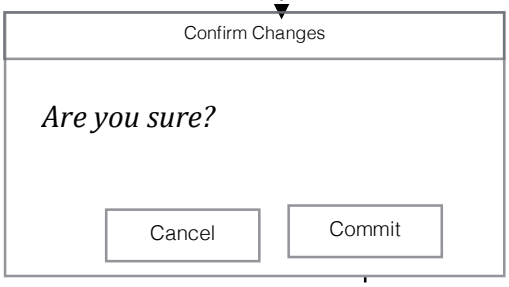
This interface shall allow the user to choose to search the database for an item to delete or to show all the tweets in the database so that they can choose what to delete.

This then commits the delete specified by the user

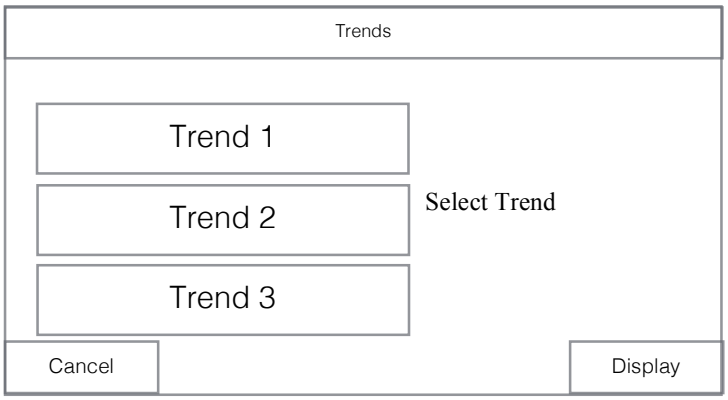


This interface shows all the data as a referential table, allowing for modification of the data in the table.

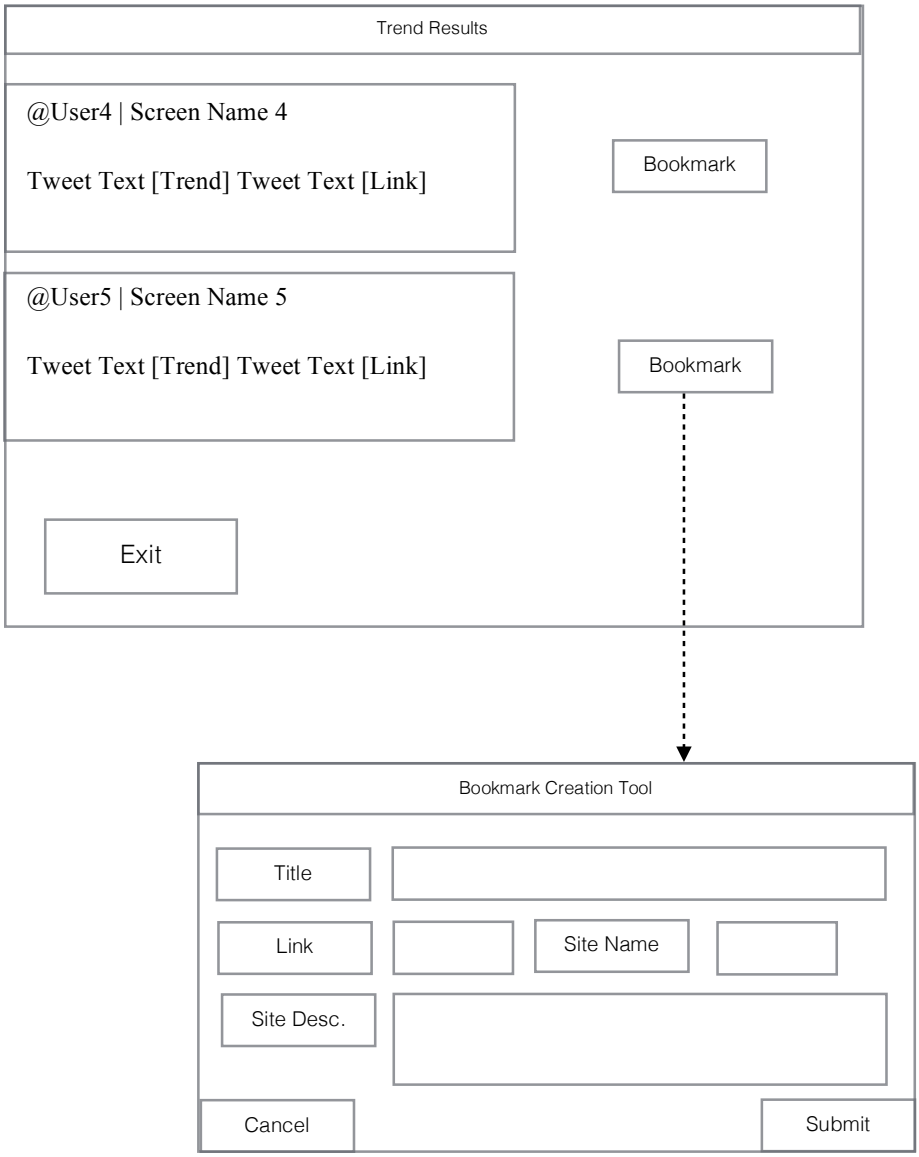
Returns user to main interface/ previous interface

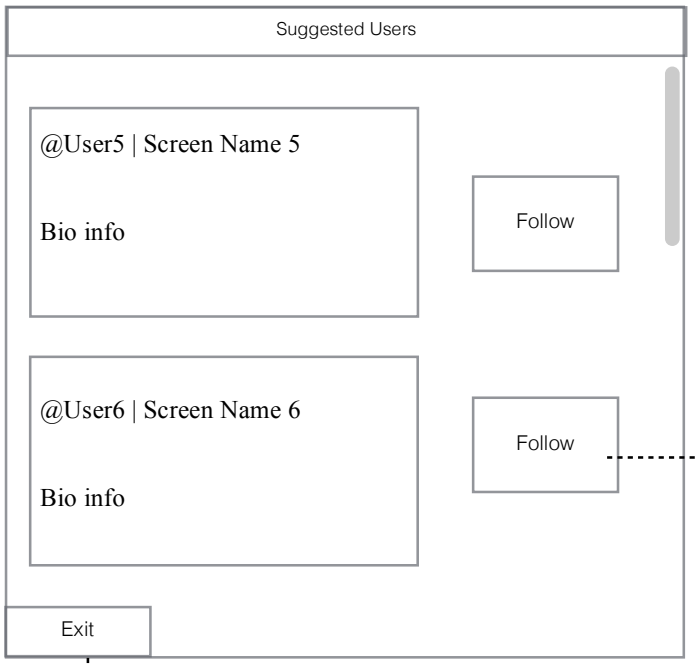


This would then commit the changes made in the Table view.



This interface will allow the user to select a trend that has been called from the Twitter API to search for tweets to bookmark from there.





This button allows the user to follow the user suggested to them.

This returns the user to the main/pervious interface

This interface shows the suggested users from the Twitter API so that the user can follow a new user and then pull their tweets and bookmark into the database.

3. Hardware Specification

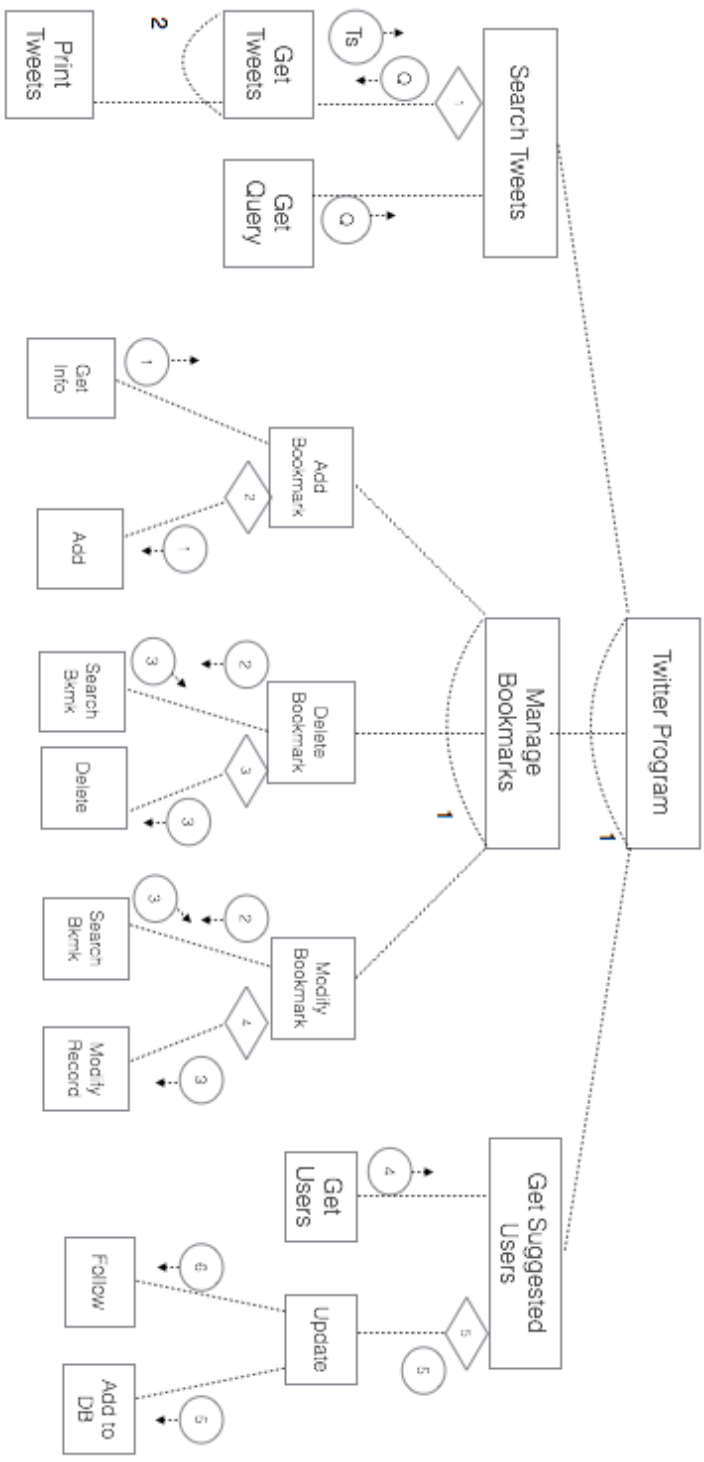
The program will be developed on and for a Macbook Air with the following specifications:

- Operating System: Mac OS X 10.9 (Mavericks)
- Processor: 1.8 GHz Intel Core i5
- Memory: 4GB 1600 MHz DDR3
- Storage: 120GB SSD Flash Storage
- Graphics: Intel HD 4000 1024MB Graphics
- Display: 13” (1440x900)
- Generic Keyboard and Trackpad

The justification for this set-up is that it is the client’s main system of use and thus should be developed for that system to keep the costs at a low and for efficiency. This set-up is entirely suitable for the program being developed as it requires no special hardware or software in order for the program to run.

4. Program Structure

4.1 Topdown design structure charts



Key

Ts — Tweets

Q — Query

1 — BookmarkTitle,
SiteName,
SiteDesc,
Link

2 — BookmarkID

3 — [Bookmark Info]

4 — List of Users

5 — Selected User
[User info]

6 — Username

1 — If Query IS Valid

2 — If Add Confirmed

3 — If Delete Confirmed

4 — If Modify Confirmed

5 — If Follow User

1 While Not Finished

2 For Tweets IN Search

4.2 Algorithms in pseudo-code for each data transformation process

```
FUNCTION Get Tweets (Query:STRING)
        QueryResults: ARRAY
        Count: INTEGER
        Tweets <— Twitter.Search(Query)
        WHILE Count <= Len(Tweets) DO
                QueryResults.Append(Tweets[Count])
        Return QueryResults
END FUNCTION
```

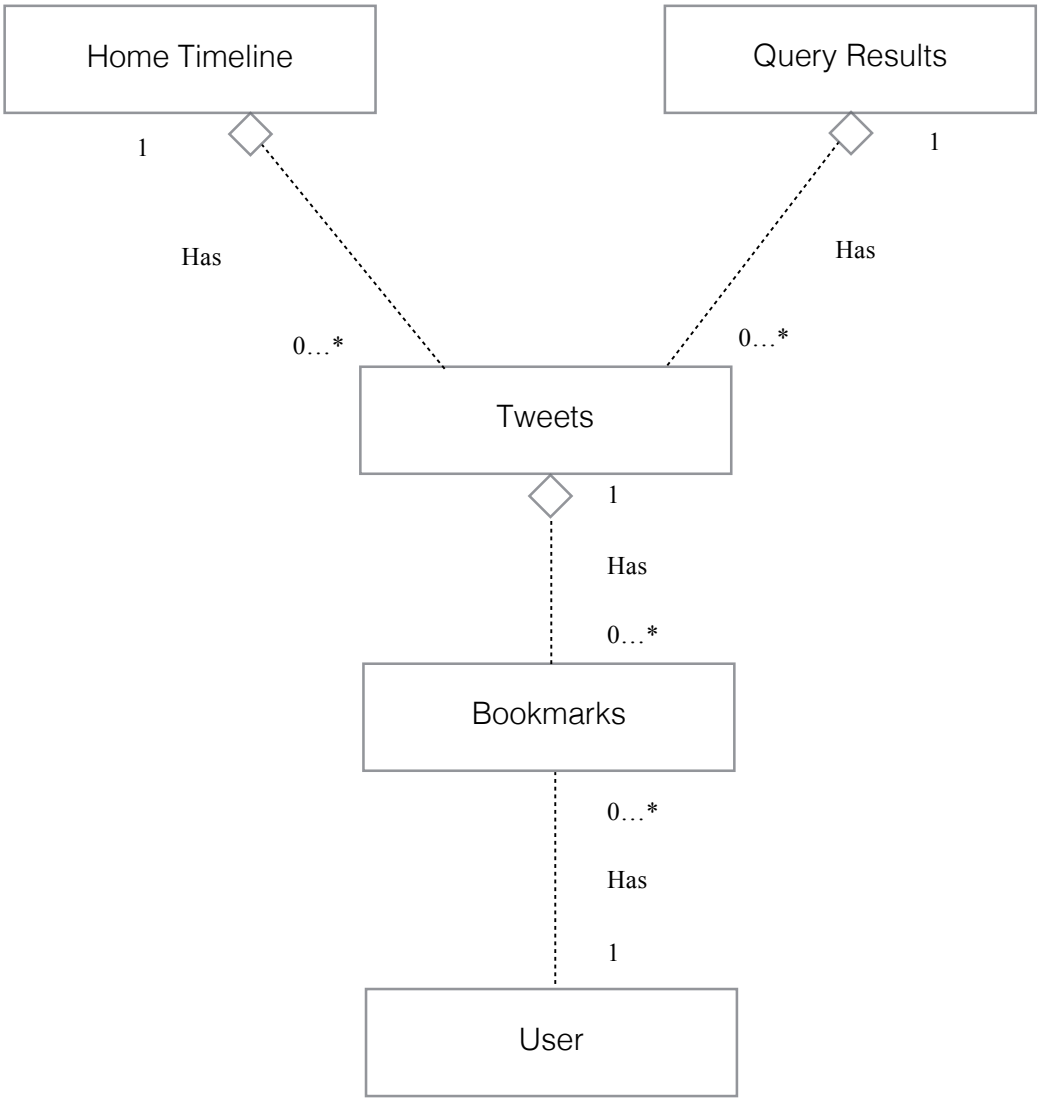
```
FUNCTION Get Query
        Query: STRING
        OUTPUT "Please enter a search term: "
        INPUT Query
        RETURN Query
END FUNCTION
```

```
FUNCTION Add Bookmarks (UserList:ARRAY)
        TweetList: ARRAY
        Choice: INTEGER
        TweetChoice: INTEGER
        FOR Users IN UserList THEN
                OUTPUT UserList[Users]
        OUTPUT "Select User: "
        INPUT Choice
        TweetList <— Twitter.statuses.user(UserList[Choice])
        FOR Tweets IN TweetList THEN
                OUTPUT TweetList[Tweets]
        OUTPUT "Select tweet to bookmark: "
        INPUT TweetChoice
        FOR Info IN TweetChoice THEN
                ADD TO DATABASE Bookmark
                        TweetChoice[Info]
END FUNCTION
```

```
FUNCTION Get Suggested ()
        SuggUsers: ARRAY
        Choice: INTEGER
        UserInfo: ARRAY
        Username: STRING
        SuggUsers <— Twitter.users.suggested()
        FOR User IN SuggUsers THEN
                OUTPUT SuggUsers[User]
        OUTPUT "Select User: "
        INPUT Choice
        FOR Info IN SuggUsers[Choice] THEN
                UserInfo.append(Info)
        Twitter.follow.user(Username)
        FOR Info IN UserInfo THEN
                ADD TO DATABASE User
```

```
                    UserInfo[Info]
END FUNCTION
```

4.3 Object diagrams



4.4 Class definitions

Home Timeline	Query Results	Tweet	User	Bookmark
Tweets Count	Tweets Count Query	Username ScreenName TweetText	Username Password	Tweet Link Title Site Name Site Descprition User
getTweets getCount storeHomeTimeline	getTweets getCount getQuery storeQueryResults	getUsername getScreenName getTweetText	getUsername getPassword storeSecurly login	getTweet getUser getTitle getSiteName getSiteDescription storeBookmark

5. Prototyping

5.1 Consideration of impact on design and development

Areas of the project that I shall prototype include all interfaces that require the user to modify the data in the database, as this will require some complex SQL Queries that should be tested before being implemented into the program. Therefore prototyping will allow the program to utilise only working SQL Queries.

Furthermore I shall prototype the methods and functions that require contacting and interacting with the Twitter API; such as logging in to Twitter, sending requests for different timelines and getting suggested users and trends. This must be prototyped to make sure that all functionality with Twitter is such that no issues occur unnecessarily while the user is using the program. Also to make sure that data received is the data required for the program.

Prototyping parts of the program in a CLI may also be useful in figuring out whether the algorithms created do in fact work and whether integration into Twitter works effectively in the program. Doing this before implementing or prototyping the UI means that the code required for the UI can be tested so as not to run into any further problems when the UI is then created =.

Also prototyping a UI for the program might be needed so as to make sure that elements within Python and Qt work correctly and also to get feedback from the client on whether the UI is suitable for their requirements discussed in the Analysis.

6. Definition of Data Requirements

6.1 Identification of all data input items

- Username
- Password

- Query
- Site Description
- Site Name

6.2 Identification of all data output items

- Query Results
- Home Timeline
- Suggested Users
- Trending Tweets
- Bookmarks

6.3 Explanation of how data output items are generated

Data Output Item	How it's generated
Query Results	Retrieved from Twitter API
Home Timeline	Retrieved from Twitter API
Suggested Users	Retrieved from Twitter API
Trending Tweets	Retrieved from Twitter API
Bookmarks (Database)	Created by user / Entries inputted by user

6.4 Data Dictionary

Name	Data Type	Length	Example	Comment	Size
Username	String	max. 15 chars	JackieG		15 - 70 Bytes
Password	String	No limit	T1nb7Js		-
Tweet	String	140 chars	"This is a tweet"	Tweets are usually ~200 Bs	140 - 840 Bytes
(Tweet user name)	String	Max. 15 chars	@MrHornett French	Included in Tweet from API	15 - 70 Bytes
(Tweet Screen Name)	String	Max. 20 chars	John Smith	Included in Tweet from API	15 - 70 Bytes
count (GET user_timeline)	Integer	max. 3 digits	30	Limited to 200	28 Bytes

Name	Data Type	Length	Example	Comment	Size
home_timeline	array	max. 800 entries	-		104 - 7064 Bytes
Query	String	Shouldn't be more than 140 chars	'Computing'	1 word or possibly a phrase	1 - 840 Bytes
count (GET search/tweets)	integer	max. 3 digits	100	Used to limit number of tweets returned with a search call.	28 Bytes
result_type	string	max 7 chars	mixed, recent, popular	Used to filter what type of tweets to return from search	~ 42 Bytes
Query Results	array	max 100 items	-		200 - 920 Bytes
Link	String	~20 chars*	't.co/gLP0Ucg'	Link size varies accoriding to the URL shortener used. Example uses t.co.	120 - 126 Bytes max.
link_needed ('Interesting' in DFD's)	string	1 character	'y' , 'n'	Used to verify bookmarking of link	50 Bytes
Site Name	string	-	CNet	The sites name, doesn't have a character limit but must have at least one character	-

Name	Data Type	Length	Example	Comment	Size
Site Descripton	string	200 chrs		A description given by the user, limited to 200 chars.	1200 - 1260 Bytes

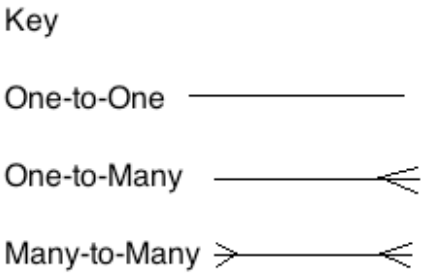
6.5 Identification of appropriate storage media

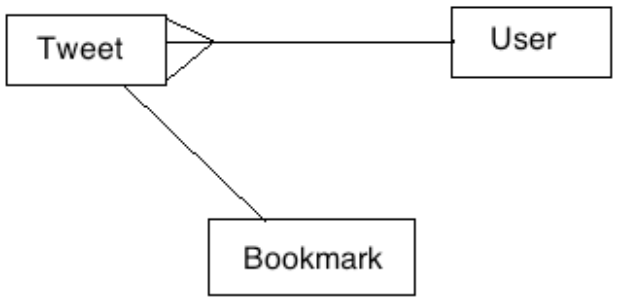
As all the operations shall only be perfumed on one computer, that being the clients main system, it is not necessary to adopt different means of storing the data but to use the main hard drive of the system itself. This is sensible as many of the data has to be re-accessed may times, such as retrieving tweets from twitter again, and thus using multiple storage media would be unfeasible as this could create data inconsistencies, especially regarding the bookmark databases. Despite this keeping a back-up if the data online could provide to be a useful medium, however it is not necessary to implement this into the program.

7. Database Design

7.1 Normalisation

7.1.1 ER Diagrams





UNF
Tweet ID
Tweet Text
Link
Site Name
Site Desc.
User ID
User Name
Screen Name
BookMark ID
Bookmark Title

1NF	
Non-Repeating	**Repeating**
TweetID	BookmarkID

1NF	
TweetText	TweetID
	Link
	Site Name
	Site Desc.
	UserID
	User Name
	Screen Name
	Bookmark Title

7.1.2 UNF to 3NF

2NF		
	Dependent	**Non-Dependent**
TweetID	BookmarkID	BookmarkID
TweetText	TweetID	Site Name
	Link	Site Desc.
		UserID
		User Name
		Screen Name
		Bookmark Title

3NF			
		Dependant	**Non-Dependant**
TweetID	BookmarkID	BookmarkID	UserID
TweetText	TweetID	*UserID*	User Name
	Link	Site Name	Screen Name
		Site Desc.	
		Bookmark Title	

Entity Descriptions

Tweet(TweetID,TweetText,*UserID*)
User(UserID,UserName,ScreenName)

Bookmark(BookmarkID,BookmarkTitle,SiteName,SiteDescription,Link,*TweetID*)

7.2 SQL Queries

For all the SQL Queries in this table, Python is being used to format all the stings that are to be processed by SQLite.

SQL Query	Description
"""CREATE TABLE Bookmark(BookmarkID INTEGER, BookmarkTitle TEXT, SiteName TEXT, SiteDescription TEXT, Link TEXT, TweetID INTEGER, PRIMARY KEY (BookmarkID), FOREIGN KEY (TweetID) REFERENCES Tweet(TweetID))"""	This is an example of a query that is used to create a database by using QSQLite in Python. What this statement does is list the attributes in the in table to be created, such as *Bookmark ID* and *Site Name*. It also sets the primary key to *Bookmark ID* and the foreign key to *TweetID*, by referencing the table *Tweet* and its primary key *TweetID*.
"""INSERT INTO Bookmark(BookmarkTitle, SiteName, SiteDescription, Link, TweetID) VALUES(?,?,?,?,?)"""	This statement is used to insert a value into the Bookmark Database. The values to be added are defined in the *Bookmark* field. The values that are to be inserted into these fields are expressed by question marks *?* as bind values will be used to insert data for a layer of security.
“"""SELECT Bookmark.BookmarkTitle, Bookmark.BookmarkDescription, Bookmark.Link, Tweet.TweetText FROM Bookmark,Tweet WHERE Bookmark.BookmarkTitle = ? OR Link = ? """	This shows how the searching feature works when the user searches for Bookmarks to delete. It shows what elements from which table to select (*Bookmark.BookmarkTitle*) and from which Databases (Bookmark) these tables are in. It also has bind values for where the user places what search terms they are.
“"""DELETE FROM Bookmark WHERE BookmarkID = ?"""	This shows the delete SQL Query to be used for deleting a bookmark where there user specifies the ID showed by a bind value in the example.
“SELECT Tweet.TweetText, Tweet.UserID, Tweet.TweetID, Bookmark.Link, User.Username FROM Tweet,User,Bookmark WHERE Bookmark.TweetID = Tweet.TweetID AND Tweet.UserID = User.UserID’”	This Select statement is used to find a specific bookmark from the bookmarks database, using information from all three tables of the database to find the entry.

8. Security and Integrity of the System and Data

8.1 Security and Integrity of Data

Both the user's Username and Password will have to stored securely on the user's computer, thus the information shall be stored in a hidden file or by encrypting the data.

As this program doesn't store information that can uniquely identify a person it does not come into conflict with legislation such as the Data Protection Act.

Referential Integrity of the data in the database has to be kept in check as well. This means that items that are connected through keys are updated throughout the database to ensure that all information is correct. In order to keep integrity when a bookmark is deleted from the database the related tweet shall also be deleted, but the Users in the database must be kept; therefore the Tweet and Bookmark tables will cascade when modified or deleted.

8.2 System Security

The program can only be accessed with someone with the correct Twitter login credentials and thus cannot be accessed by anyone without those details providing security

9. Validation

Data	Example	Validation used	Comment
Username	UserName110	Length Check Presence Check	Usernames have a max. of 15 charters and a minimum of 1 and so checking the length validates the input.
Password	Hfgj1*^$	Presence Check	Password have no upper or lower limit, however this field cannot be left blank.
Query	Computing'	Presence Check	This field in the program cannot be left blank.
Site Name	CNet	Presence Check	No upper limit but must bit be blank
Site Description	Website for computing	Presence Check Length Check	Length shall be limited.
Bookmark Title	Resource 1	Presence Check	No upper limit, must not be blank
Link	'http://t.co/XsdnUKma3'	Presence Check Length Check Format Check	t.co links (Twitters URL shortener) has a variable upper limit which can be obtained from the API. As of writing the upper limit is twenty. The space cannot be blank. It must be in the standard URL format.

10. Testing

10.1 Outline Plan

10.1.1 Identification and explanation of suitable test strategies

Test Series	Purpose of testing series	Testing Strategy	Strategy Rationale
1	Test the flow of data between the user and the database	Black-Box Testing	
2	Test the validation processes	Bottom-up Testing	
3	Test CLI operations and functions	Bottom-up Testing.	
4	Test GUI operations and Functions	Bottom-up Testing.	
5	Test whether program meets required specification	Acceptance Testing	

10.2 Detailed Plan

Test and series number	Purpose	Description	Test Data	Test Data Type	Expected Result	Actual Result	Evidence in Appendix
1,01	Test Adding Data to the database	Tests the flow of data to the database	Valid Bookmark Data Invalid Bookmark data	Normal Erroneous	Add Bookmark Error		
1,02	Test Modifying data in database	Tests the manipulating of database data.	Manipulated Bookmark Data Invalid Changes	Normal Erroneous			
1,03	Test Deleting Data from Database	Test removing of data from the database	Selected Bookmark Invalid Data	Normal Erroneous/ Boundary			
1,04	Test Logging into program (CLI)	Tests authorisation to API in CLI	*Uses the OAuth Authorisation, username and password.* *-> Invalid Data*	Normal Erroneous	*No Error to be presented,*		
2,01	Test selecting values from menu		*1* *x* *2014*	Normal Erroneous Boundary	*Select Menu Item* *Error* *Error*		
2,02	Test Validating Adding tweet	Tests whether the user wishes to add a tweet to database.	Y N 'x' '1'	Normal Normal Erroneous Erroneous	Tweet Added Tweet Not Added Error Occurs Error Occurs		
2,03	Test Validating Add User	Tests whether the user wishes to add a user from search to database,	Y N 'c' '2'	Normal Normal Erroneous Erroneous	Add User Doesn't Add User Error Occurs Error Occurs		
2,04	Testing Validating Deleting Tweet	Validates whether a user wishes a tweet selected to be deleted	Y N 'd' '4'	Normal Normal Erroneous Erroneous	Deletes Tweet Doesn't Delete Tweet Error Error		
2,05	Test validating Modifying data	Validates whether a user wants the changed data committed,	Y *N* *[Invalid data]*	Normal Normal Erroneous	Modifies data Doesn't modify data. Error		
2,06	Test Authorisation of application.	Validates the Username and password [through Twitter API] if successful then no error.	*Username and Password* *[Invalid Data]*	Normal Erroneous	*Log in to Twitter.* *Error*		

Test and series number	Purpose	Description	Test Data	Test Data Type	Expected Result	Actual Result	Evidence in Appendix
3,01	Test Logging in	Tests authorisation to API in CLI	see 1,04	see 1,04	see 1,04	see 1,04	see 1,04
3,02	Test searching for user	Tests connecting to API to get users.	TheHornettB'	Normal	Returns "Kurt Hornett"s Twitter account.		
			[Non-existant User name]	Erroneous	Error		
3,03	Test Adding user to database	Test adding aforesaid user to the database.	Uses data from 2,03	see 2,03	Confirmation of added data.		
3,04	Test Displaying user tweets	Tests obtaining and displaying tweets from user in database.	*Selects user from database 'Kurt Hornett'*	Normal	Displays tweets.		
3,05	Test Validating adding tweet.	Tests the validation process of adding tweets in CLI	Clicking 'Yes'	Normal	Adds Tweet		
			Clicking 'No'	Normal	Doesn't add Tweet.		
			Clicking yes with invalid data	Erroneous			
3,06	Test deleting tweet from database.	Test deleting tweet function in CLI	*Selects Tweet from Database.* '1'	Normal	Tweet '1' deleted.		
			Selecting a tweet not in the list '2014'	Boundary	Error		
			Invalid Data	Erroneous	Error		
3,07	Test modifying database.	Tests modifying data module in CLI	'New Title' *Change Bkmk Title.*	Normal	'New Title' *Change Title*		
			Invalid Data	Erroneous	*Error*		
3,08	Test getting suggested users.	Tests getting suggested users from API in CLI	-	-	No Error		
3,09	Test adding suggested user to database.	Test adding a user from the suggested users list to database.	User from returned suggested users list.		User added to database,		

Test Series	Purpose	Description	Test Data	Data Type	Expected Result	Actual Result	Evidence
4.01	Test Logging in	See 2.01 [with GUI]	*Username and Password*	Normal	Log-in Successful		
4.02	Test searching for user from GUI elements works	Test that pulling data from GUI elements works	TheHornettB'	Normal	Find; Kurt Hornett		
			Non-existant Username	Erroneous	Error		
4.03	Test Switching between layouts.	Switch between layouts/ screens	-	-	No Error		
4.04	Test Using push button to commit changes.	-	-	-	No Error		
4.05	Test RelationalTableView Functional	Test viewing data from database as a Relational Table.	-	-	Display Table		
4.06	Test Displaying Tweets on Window	Tests showing the list of tweets on the Window.	-	-	No Error		
4.07	Test Inputting data to add Bookmark to Database.	Tests the adding of data to a field to be pushed to database.	*[Bookmark Data]*	Normal	Bookmark added to Database.		
			[Invalid Bookmark Data]	Erroneous/ Boundary	Error		
4.08	Test Viewing Suggested Users	Tests the users being displayed to the user.	-	-	Users Dipayed		
4.09	Test adding user from Suggested Users	Tests adding a user from suggested users.	User from list	Normal	User added to database.		
			Invalid Data	Erroneous	Error		
4.10	Test modifying data	Tests modifying the data in the relational table	Valid changes	Normal	Changes made		
			Invalid Changes	Erroneous	Error		
4.11	Test Deleting data	Tests deleting data in the GUI Model	Select Bookmark	Normal	Delete Bookmark.		

Test Series	Purpose	Description	Test Data	Data Type	Expected Result	Actual Result	Evidence
5,0	Acceptance testing	Test to see program fits requirements set by the client in the analysis stange	-	-	-	-	-

This Page is Intentionally Blank

Testing

1. Test Plan

1.1 Detailed Test Plan

-Test Series

Test Series	Purpose of testing series	Testing Strategy	Strategy Rationale
1	Test the flow of control in CLI	Black-Box Testing	Tests the base functions of the whole CLI system.
2	Test the validation processes in CLI	Black-Box Testing	Tests the base functions of the whole CLI system.
3	Test CLI operations, functions, and flow of data.	Bottom-up Testing.	To test base units progressing through the system
4	Test GUI operations & functions	Bottom-up Testing.	To test base units progressing through the system
5	Test GUI Flow of data	Black-Box Testing	

- Detailed Test Plan

Test Series	Purpose	Description/Changes	Evidence
1.1	Test selecting choice on main menu.	Tests whether control of main menu is fluid	Figures 1a-e pp.72-4
1.2	Test selecting choice in *HomeUserMenu*	Tests whether control of *HomeUserMenu* is fluid	
1.3	Test selecting choice in *databaseMenu*	Tests whether control of *databaseMenu* is fluid	
1.4	Test selecting choice in *displayModifyChoice*	Tests whether control of *displayModifyChoice* is fluid	
1.5	Test selecting choice in *displaySlugs*	Tests whether control of *displaySlugs* is fluid	
2.1	Test validating adding Bookmark	Tests whether choosing to (not) add bookmarks to database works	
2.2	Test validating adding user	Tests whether choosing to (not) add user to database works	Figures 2a-e pp.74-6
2.3	Test validating Deleting Bookmark	Tests whether choosing to (not) to delete bkmk works	Figures 2f-k pp.75-8
2.4	Test Authorizing Application	Tests whether one can authorize the application.	Figures 3a-e pp.79-81

Test Series	Purpose	Description/Changes	Evidence
3.1	Test Searching for user	Tests whether the user can search for a user	
3.2	Test adding user to databse	Tests whether the user can add a user to the database	
3.3	Test displaying user tweets	Test whether the user can display the tweets from a user	
3.4	Test Adding bookmark	Tests whether the user can make a bkmk and add it to the databse	Figures 3.1a-g pp.81-3
3.4.1	Test adding Title	Tests that flow from adding title to next function works. There is no invalid input	Figures 3.1b-e
3.4.2	Test link info retreival	If a tweet has a link then the system should fetch the HTML source and find the site's name and title, adding it to the bookmark info. Tests whether this works	Figure 3.1c
3.4.3	Test adding site description	Tests that flow from adding title to next function works. There is no invalid input	Figure 3.1c
3.4.4	Test Adding Bookmark	Test whether the user can add a bookmark to the database	Figure 3.1d,f-g
3.5	Test Deleting Bookmarks	Tests whether the user is able to delete a bookmark	
3.6	Test Modifying Bookmarks	Tests whether a user is able to modify a bookmark's data.	
3.6.1	Test Modifying bookmark title	Tests whether a user is able to modify the bookmark title	
3.6.2	Test modifying Site name	Tests whether the user can edit the Bookmarks Site Name	
3.6.3	Test modifying Site Description	Tests whether the user is able to modify the site description	
3.6.4	Test modifying link	Tests whether the user is able to modify the link	
3.7	Test getting Suggested Users	Tests whether the user is able to obtain suggested users.	
4.1	Test switching to *Search User Tweet* layout	Tests whether the user is able to switch to the *search user tweets* layout.	Figures 4a-c p.84-5
4.2	Test *userMenuButton* displays list of users	Tests whether when the user selects the User Menu Button a list of selectable users is shown.	
4.3	Test Selecting A user from list	Tests whether the user can select a user from the	

Test Series	Purpose	Description/Changes	Evidence
		UserMenuButton list.	
4.4	Test return button works	Tests whether the return button sends the user to the previous screen.	Figures 5a,b p.85
4.5	Test switching to delete interface	Tests whether the user is able to switch to the delete entry interface.	
4.6	Test 'To Delete Button'	Tests whether when the user selects the 'to delete' button a list of options is displayed.	Figures 6a,b pp.86
4.7	Test Switching to Modify Interface	Tests whether the user is able to select the Modify Interface from the Menu.	
4.8	Tests switching to Suggested User interface	Tests whether the user is able to select switch to the suggested user interface from the Menu Bar	
4.9	Tests 'Select Topic' Button	Tests whether a suitable list of slugs is displayed when the user selects 'Select Topic'	Figures 7a,b p.87
5.1	Test searching for User	Tests whether the user is able to search and add a User to the database.	Figures 8a-d pp.87-8
5.2	Test showing Users Tweets	Tests whether the user is able to display the tweets from a user	
5.3	Test bookmarking tweet	Tests whether the user is able to bookmark a tweet from the list provided.	Figures 9a-f pp.89-91
5.3.1	Test selecting Tweet	Tests whether the user can successfully add a tweet	Figure 9b
5.3.2	Test title info into Bookmark Creation Tool	Tests whether the user can successfully add a title to bookmark creation tool.	Figures 9c,d
5.3.3	Test adding site description into Creation tool	Tests whether the user is able to add a site description to the bookmark tool	Figures 9c,d
5.3.4	Test adding bookmark	Tests whether the user is able to add a bookmark with the information provided.	Figures 9e,f
5.4	Test deleting bookmark	Tests whether the user is able to delete a bookmark	
5.4.1	Test selecting bookmark to delete	Tests whether the user is able to select a bookmark to delete	
5.4.2	Test confirming deleting bookmark	Tests whether the user can confirm deleting the bookmark	

Test Series	Purpose	Description/Changes	Evidence
5.5	Test modifying bookmark	Tests whether the user is able to modify bookmarks in the database	Figures 10a-d pp.91-2
5.5.1	Test inputting new data	Tests whether the user is able to add new data for a bookmark	Figures 10b,c
5.5.2	Test submitting changes	Tests whether the user is able to submit those changes to the databse	Figure 10d
5.6	Test adding a suggested user to the databse	Tests whether the user is able to add a suggested user to the database	Figures 11a-f pp.93-4

1.2 Changes From Original Plan

There are some fundamental changes to the test series and test plan, this is due to changes in the development and implementation of the system which have brought on different paradigms for using the system.

Test Series	Change	Justification
1	Tests the flow of control rather than the flow of data.	Shifted changes regarding the flow of data and validation of data to series involving the CLI and GUI
3	Added flow of data.	see change 1
4	Added flow of data. Spit into Functions and Operations, and Flow of Data	see change 1 Having a single test series didn't seem necessary, splits into smaller series.
5	Removed See Change to TS 4	Not feasible to carry out acceptance testing. See Change to TS 4
1.1 through 1.5	Tests control of menus	see change 1
2.1 through 2.4	Test series shifted -1	Removed original 2.1
3.1 through 3.7	Test series shifted -1	Removed original 3.1
3.4	Changed from Validation to Testing adding bkmk. Added sub-series	Validation handled in TS 2. No Bkmk testing in original plan.
3.6	Created sub-test series	Original test too vague; all relate to one

Test Series	Change	Justification
		overall test.

2. Test Data

2.1 Test Data

Test Series	Test Data	Data Type	Expected Result	Actual Result
1.1	1 2 2014 x	Normal Normal Boundary Erroneous	Choice selected Choice selected Re-enter choice Re-enter choice	Choice selected Choice selected Re-enter choice Re-enter choice
1.2	1 0 2014 x	Normal Normal Boundary Erroneous	Choice selected Exit to main menu Re-enter choice Re-enter choice	Choice selected Exits to main menu Re-enter choice Re-enter choice
1.3	1 0 2014 x	Normal Normal Boundary Erroneous	Choice selected Exit to main menu Re-enter choice Re-enter choice	Choice selected Exit to main menu Exits to main menu Re-enter choice
1.4	1 0 2014 x	Normal Normal Boundary Erroneous	Choice selected Exit to main menu Re-enter choice Re-enter choice	Choice selected Exit to main menu Re-enter choice Re-enter choice
1.5	1 0 2014 x	Normal Normal Boundary Erroneous	Choice selected Exit to main menu Re-enter choice Re-enter choice	Choice selected Exit to main menu Re-enter choice Re-enter choice
2.1	Y' 'N' 'x' '1'	Normal Normal Erroneous Erroneous	Adds bkmk to db Doesn't add bkmk to db Re-enter Option Re-enter Option	Adds bkmk to db Doesn't add nkmk to db Re-enter Option Re-enter Option
2.2	'musescore' 'TheHornettB' [blank space]	Normal Erroneous Exceptional	Adds user to db Raises 'user already in database error' Returns to previous menu	Adds user to db Raises 'user already in database error' Returns to previous menu
2.3	'Y' 'n' 'x'	Normal Normal Erroneous	Deletes Bkmk Doesn't delete bkmk Re-enter Option	Deletes Bkmk Doesn't delete bkmk Re-enter Option

Test Series	Test Data	Data Type	Expected Result	Actual Result
	'1'	Erroneous	Re-enter Option	Re-enter Option
2.4	*Enter PIN Displayed.* *Enter different number*	Normal Erroneous	Authorizes Error Raised	Authorizes Error Raised
3.1	TheHornettB [blank space]	Normal Exceptional	Returns to main menu Returns to main menu	Returns to main menu Returns to main menu
3.2	*see TS 2.2*	*see TS 2.2*	*see TS 2.2*	*see TS 2.2*
3.3	1 2 'x' 2014	Normal Normal Erroneous Boundary	Displays tweets Displays tweets Re-enter option Error Occurs	Displays tweets Displays tweets Re-enter option Error Occurs
3.4.1	'Title' [blank space]	Normal Normal	Move onto next function Move onto next function	Moves onto next function Moves onto next function
3.4.2	N/A	N/A	Site's data processed	Site's data processed
3.4.3	'Site Desc' [blank space]	Normal Normal	Move onto next function Move onto next function	Moves onto next function Moves onto next function
3.4.4	Y N 'x' 1	Normal Normal Erroneous Erroneous	Adds bookmark Doesn't add bkmrk Re-enter Option Re-enter Option	Adds bookmark Doesn't add bkmrk Re-enter Option Re-enter Option
3.5	Y N 'x' 1	Normal Normal Erroneous Erroneous	Deletes bookmark Doesn't delete bkmrk Re-enter Option Re-enter Option	Deletes bookmark Doesn't delete bkmrk Re-enter Option Re-enter Option
3.6.1	'New Title' [blank space]	Normal Normal	Adds new title Adds new title	Adds new title Adds new title
3.6.2	'New Site Name' [blank space]	Normal Normal	Adds new Site title Adds new site title	Adds new Site title Adds new site title

Test Series	Test Data	Data Type	Expected Result	Actual Result
3.6.3	'New Site Desc' [blank space]	Normal Normal	Adds new Site Description Adds new Site Description	Adds new Site Description Adds new Site Description
3.6.4	'www.gov.uk' [blank space]	Normal Normal	Adds new link Adds new link	Adds new link Adds new link
3.7	'1' '2' '5' 'c'	Normal Normal Boundary Erroneous	Makes selection Makes selection Return to main menu Re-enter choice	Makes selection Makes selection Return to main menu Re-enter choice
4.1	Click *Search user Tweets* Menu Action	Normal	Switches Layout	Switches Layout
4.2	Click User Menu Button	Normal	Shows User List	Shows User List
4.3	Select user from list; mouse click	Normal	Shows Users Tweets	Shows Users Tweets
4.4	Select return button	Normal	Returns to previous menu	Returns to previous menu.
4.5	Click 'Delete Entries' menu Action	Normal	Switches Layout	Switches Layout
4.6	Click 'To Delete' Button	Normal	Displays list	Displays List
4.7	Select 'Modify Entries' Menu Action	Normal	Switches Layout	Switches Layout
4.8	Select Suggested Users Action	Normal	Switches Layout	Switches Layout
4.9	Click 'Select Topic' button	Normal	List Displayed	List Displayed
5.1	'Musescore'	Normal Erroneous	User added to db Error Raised	User added to db Error Raised

Test Series	Test Data	Data Type	Expected Result	Actual Result
	'TheHornettB' [blank Space]	Exceptional	Error Raised	Error Raised
5.2	Select user from list	Normal	Shows users tweets	Shows users tweets
5.3.1	Select tweet from list	Normal	Shows Bookmark Creation Tool	Shows bookmark Creation tool.
5.3.2	'New Title'	Normal	Input into line edit	Input into line edit
5.3.3	'Site Desc'	Normal	Input into line edit	Input into line edit
5.3.4	Click 'Submit' button	Normal	Adds Bookmark	Adds Bookmark
5.4.1	Click number on list	Normal	Displays Dialog	Displays Dialog
5.4.2	Click Confirm Click close button	Normal Normal	Deletes bookmark Doesn't delete bookmark	Deletes Bookmark Deletes Bookmark
5.5.1	'New Title' 'New Site Desc'	Normal Normal	Input data Input data	Input Data Input data
5.5.2	Click Submit Button	Normal	Confirmation Dialog	Confirmation Dialog
5.6	Click Add user button	Normal	Confirmation Dialog	Confirmation Dialog

2.2 Changes From Original Test Data

As discussed above major changes in the implementation period have affected the way in which the system needs to be tested and so radical changes have been made to the structure of the test data.

Test Series	Change	Justification
2.2	Shifted from 'Y', 'N' to a username(s)	Change in function of interface
2.4	PIN entered instead of Username/Pass	System uses a PIN method.

3. Annotated Samples

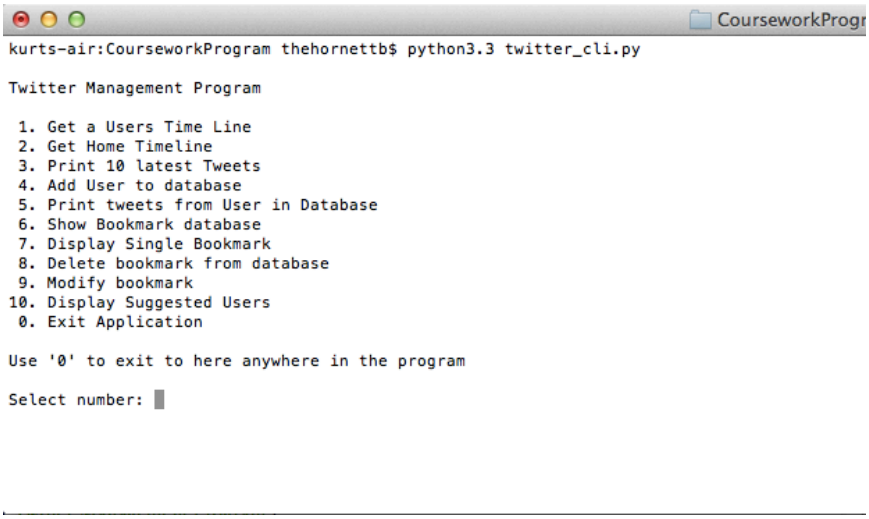
3.1 Actual Results

All actual results can be found in the Test Data table (§2.1)

3.2 Evidence

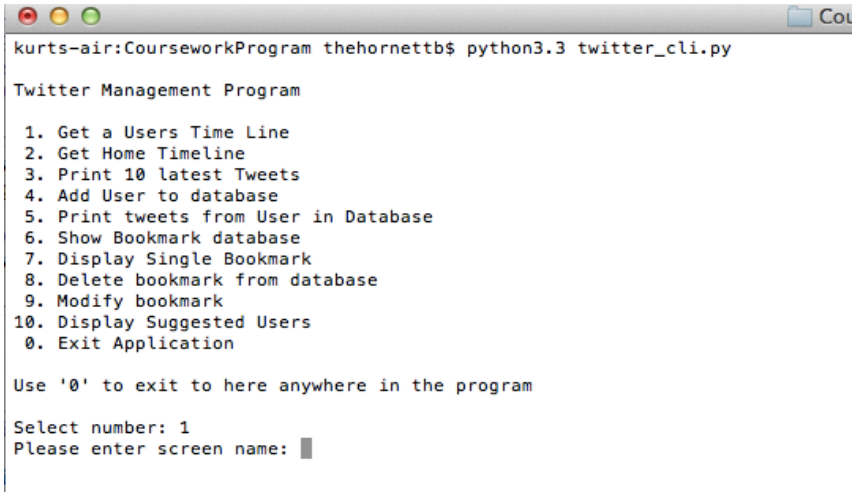
Test Series 1

Figure 1a - State of program before test



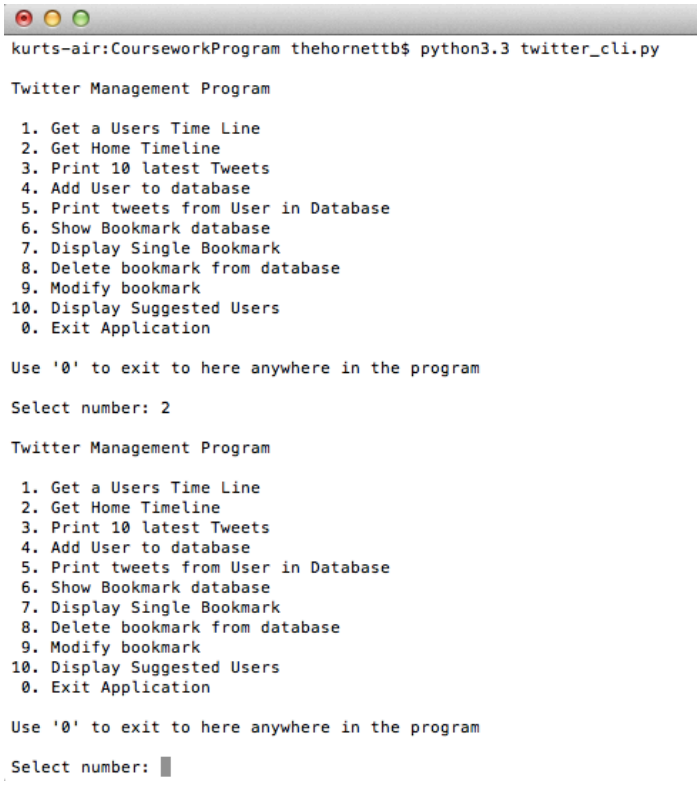
This figure shows the state of the program before entering an option to use. The main menu is displayed with instruction.

Figure 1b - Inputting choice 1 to menu



When selecting option 1 this screen is show, displaying the next menu which asks for a screen name for a users timeline.

Figure 1c - Inputting choice 2 into menu

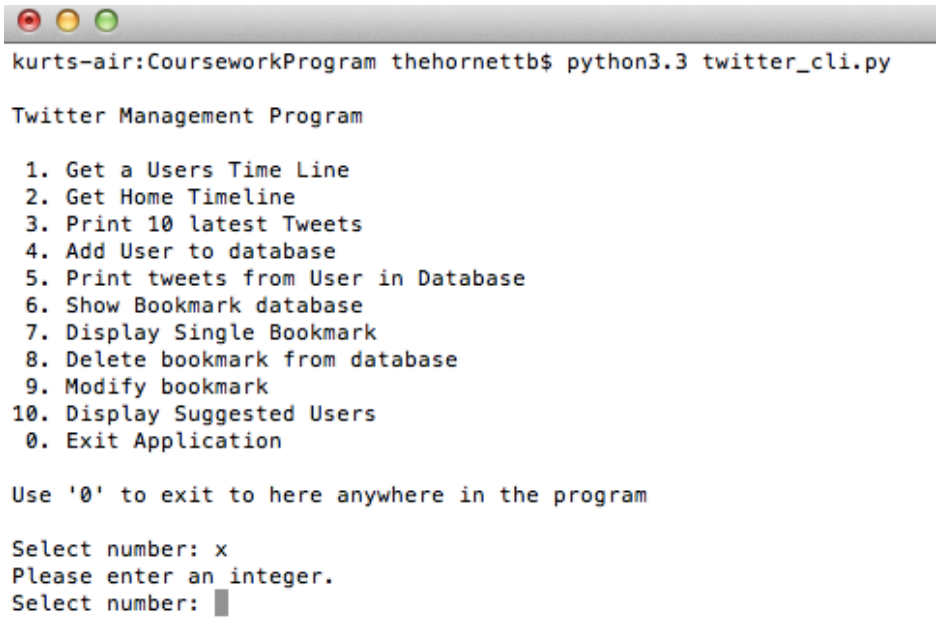


When selecting option 2 the main menu is displayed again, this is what is expected as the users timeline is fetched from Twitter. Because there is no error it is clear that the function was performed.

Figure 1d - Inputting a boundary error

This shows that when entering a boundary error, one is presented with the 'select number' dialogue again . This is due to the integer '2014' not being suitable.

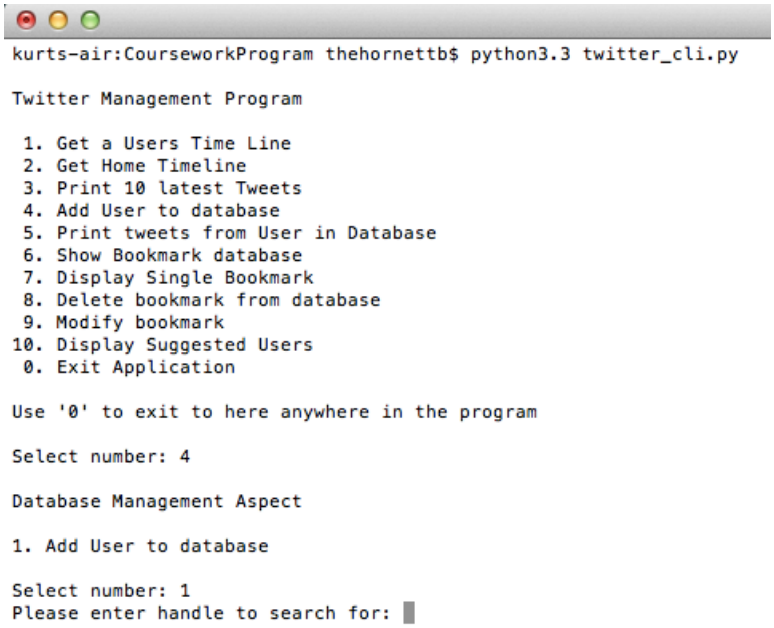
Figure 1e - Inputting erroneous data



This shows that when an integer is not entered then another dialogue is displayed that prompts the user to re-enter their choice as it is also not suitable.

Test Series 2

Figure 2a - State of program before input



This shows the program before typing in a username, the initial screen for the

Figure 2b - Adding a user 'musescore'

```
Database Management Aspect

1. Add User to database

Select number: 1
Please enter handle to search for: musescore

Twitter Management Program

 1. Get a Users Time Line
 2. Get Home Timeline
 3. Print 10 latest Tweets
 4. Add User to database
 5. Print tweets from User in Database
 6. Show Bookmark database
 7. Display Single Bookmark
 8. Delete bookmark from database
 9. Modify bookmark
10. Display Suggested Users
 0. Exit Application

Use '0' to exit to here anywhere in the program

Select number:
```

When an unknown username is entered and successfully added the main menu is displayed.

Figure 2c - Proof that user added

```
Users in database

1  Kurt Hornett
2  CardsAgainstHumanity
3  Adam McNicol
4  Anya Gritsenko
5  Bill Gates
6  Twitter
7  MuseScore

Select number:
```

Proof the user is now in the database.

Figure 2d - Adding an already existing user

```
Database Management Aspect

1. Add User to database

Select number: 1
Please enter handle to search for: TheHornettB
User already exits, not added

Twitter Management Program
```

When a know user is added then the user is told and the user isn't re-added.

Figure 2e - Adding exceptional data

```
Database Management Aspect

1. Add User to database

Select number: 1
Please enter handle to search for:
An error has occcured
An error occured

Twitter Management Program
```

When a blank space is input into the field then an error message occurs.

Figure 2f - State of program before test

```
Twitter Management Program

 1. Get a Users Time Line
 2. Get Home Timeline
 3. Print 10 latest Tweets
 4. Add User to database
 5. Print tweets from User in Database
 6. Show Bookmark database
 7. Display Single Bookmark
 8. Delete bookmark from database
 9. Modify bookmark
10. Display Suggested Users
 0. Exit Application

Use '0' to exit to here anywhere in the program

Select number: 8
N. ID Tweet Text
1  3  We'll be at the @nerdologues show at Sketchfest tonight g
2  4  RT @MaxTemkin: "Consider again that dot, you selfish fuck

Please Choose a Bookmark ID to delete, 0 to exit:

Select number:
```

This shows what is displayed when the users selects the delete bookmark menu.

For Full display see Appendix 1

Figure 2g - Validating user input

```
Please Choose a Bookmark ID to delete, 0 to exit:

Select number: 3

Bookmark Information

Title: Test
Site Name: Tumblr
Description: An Images on Tumblr
Link: https://t.co/JOUYymsmT4

Commit Action (Y/N):
```

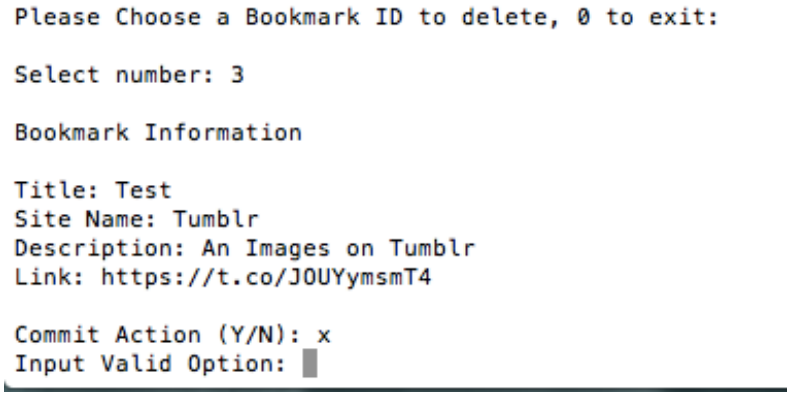
When a user selects a bookmark the validation dialog is displayed.

Figure 2h - Validating not adding bookmark

```
Please Choose a Bookmark ID to delete, 0 to exit:

Select number: 3

Bookmark Information

Title: Test
Site Name: Tumblr
Description: An Images on Tumblr
Link: https://t.co/JOUYymsmT4

Commit Action (Y/N): N
Bookmark not deleted

Twitter Management Program

 1. Get a Users Time Line
 2. Get Home Timeline
 3. Print 10 latest Tweets
 4. Add User to database
 5. Print tweets from User in Database
 6. Show Bookmark database
 7. Display Single Bookmark
 8. Delete bookmark from database
 9. Modify bookmark
10. Display Suggested Users
 0. Exit Application

Use '0' to exit to here anywhere in the program

Select number:
```

This shows that when N is selected the bkmk is not deleted.

Figure 2i - Inputting erroneous data



When invalid input x is entered a dialog raising the error is displayed

Figure 2j - Inputting erroneous data

```
Please Choose a Bookmark ID to delete, 0 to exit:

Select number: 3

Bookmark Information

Title: Test
Site Name: Tumblr
Description: An Images on Tumblr
Link: https://t.co/JOUYymsmT4

Commit Action (Y/N): 1
Input Valid Option:
```

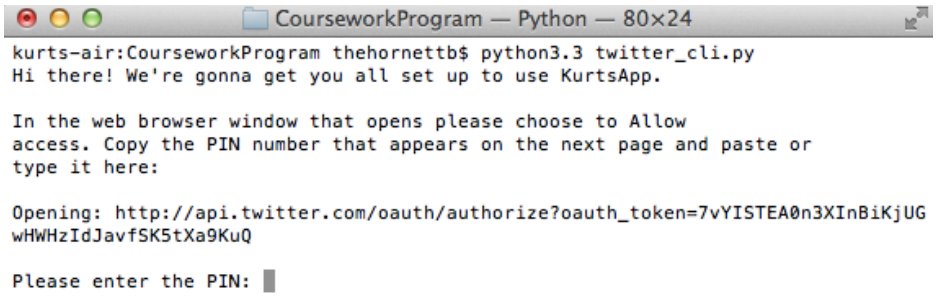
When '1' is entered an invalid input message is displayed

Figure 2k - Validating deleting bookmark.

```
Please Choose a Bookmark ID to delete, 0 to exit:

Select number: 3

Bookmark Information

Title: Test
Site Name: Tumblr
Description: An Images on Tumblr
Link: https://t.co/JOUYymsmT4

Commit Action (Y/N): Y

Twitter Management Program

 1. Get a Users Time Line
 2. Get Home Timeline
```

When 'Y' is entered the program simply continues as the bookmark is deleted.

Figure 3a - State before test

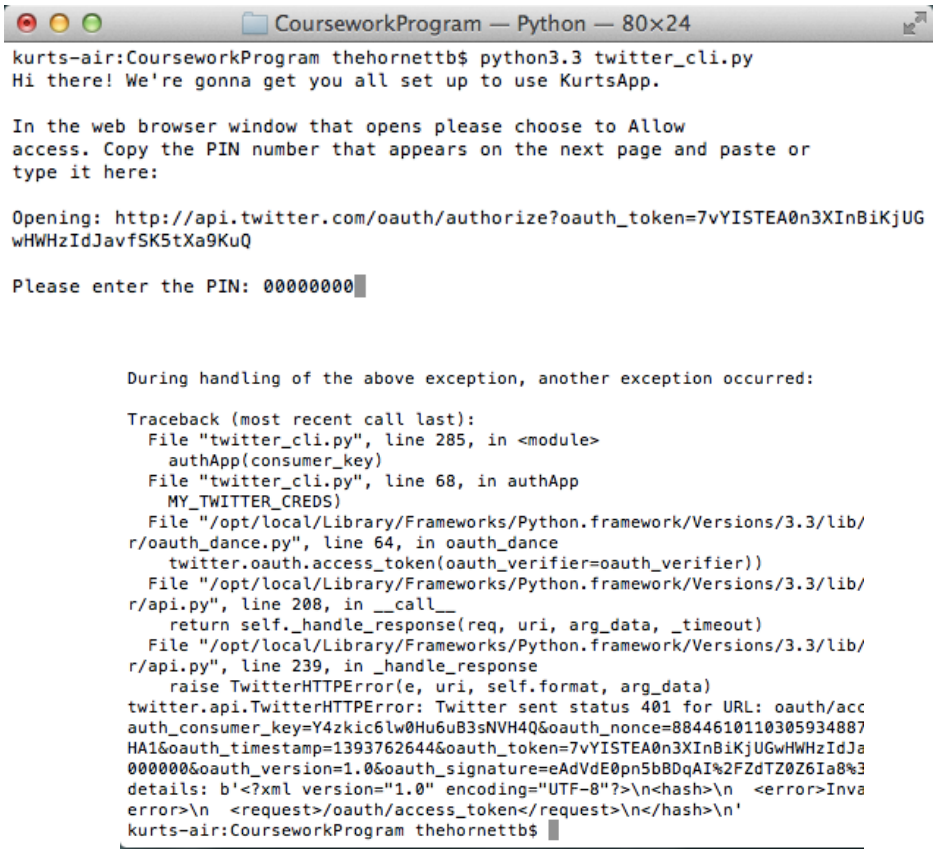


This shows the initialization of the application; if the user hasn't logged in before. This displays the dialog in the CLI.

Figure 3b - Website shown for Auth

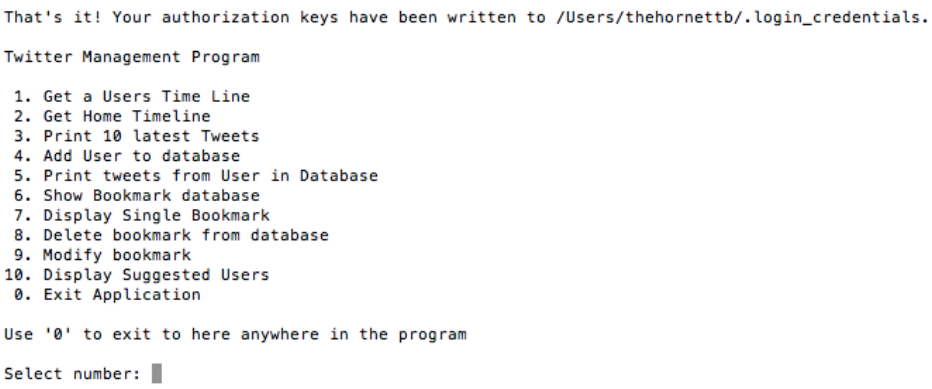
This image shows the initial page for OAuth; after clicking Authorize app the user is given a PIN to enter into the application.

Figure 3c,d - Entering an incorrect PIN



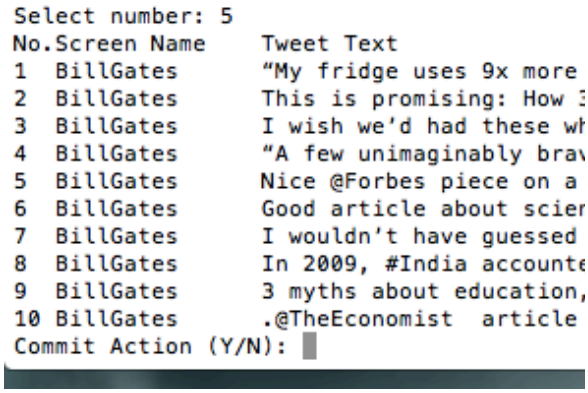
These images show what happens when an incorrect PIN is entered; an error messaged is raised and the program aborted.

Figure 3e - Entering correct PIN, program initialized



This images shows what happens when the correct PIN is entered and the main menu of the program.

Figure 3.1a - State of system before test



This shows the program before choosing a tweet to bookmark, The Tweets are from Bill Gates, a user in the database, for a full image see Appendix 2

Figure 3.1b - Selected Tweet; Add title dialog

```
Select number: 5
No.Screen Name    Tweet Text
1  BillGates      "My fridge uses 9x more ele
2  BillGates      This is promising: How 3D p
3  BillGates      I wish we'd had these when
4  BillGates      "A few unimaginably brave p
5  BillGates      Nice @Forbes piece on a hig
6  BillGates      Good article about scientis
7  BillGates      I wouldn't have guessed tha
8  BillGates      In 2009, #India accounted f
9  BillGates      3 myths about education, th
10 BillGates      .@TheEconomist  article of
Commit Action (Y/N): Y
Select number: 1
Please enter a title for the bookmark:
```

This image shows the program prompting the user for a title for the bookmark.

Figure 3.1c - Adding Site Description; Showing Link Info Retrieval

```
Commit Action (Y/N): Y
Select number: 1
Please enter a title for the bookmark: New Title
Please enter a short description of the site: Site Desc

Are you sure you wish to add this bookamrk?

Title: New Title
Site Name: Seven Graphics that Explain Energy Poverty and
Site Description: Site Desc
Link: http://t.co/LnwiU5K4CH
User: Bill Gates

Commit Action (Y/N):
```

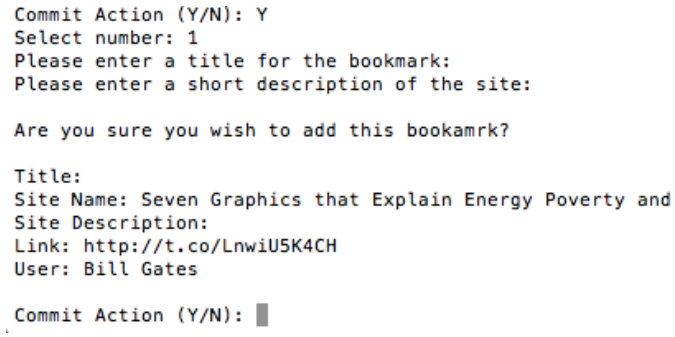
This image shows the validated bookmark information, the Title, site description and link info parsed from the tweet ready for the user to verify.

Figure 3.1d - Adding bookmark; returning to previous interface

```
Title: New Title
Site Name: Seven Graphics that Explain Energy Poverty and
Site Description: Site Desc
Link: http://t.co/LnwiU5K4CH
User: Bill Gates

Commit Action (Y/N): Y

No.Screen Name    Tweet Text
1  BillGates      "My fridge uses 9x more electricity tha
2  BillGates      This is promising: How 3D printing migh
3  BillGates      I wish we'd had these when I was in sch
4  BillGates      "A few unimaginably brave people took o
5  BillGates      Nice @Forbes piece on a high-tech pedic
6  BillGates      Good article about scientists who are t
```

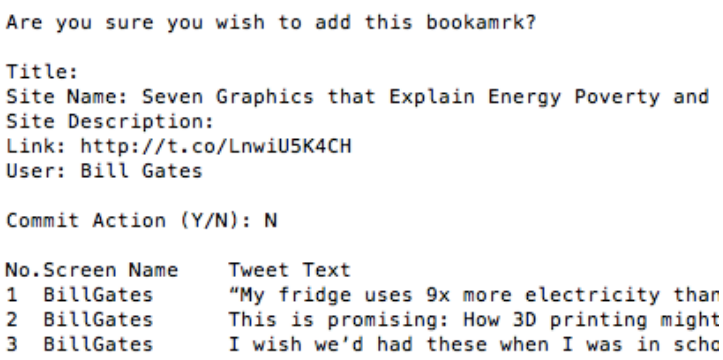
This image shows the user inputting 'Y' to add the bookmark, validating the add.

Figure 3.1e - Imputing blank spaces instead of data for Title and Site Description



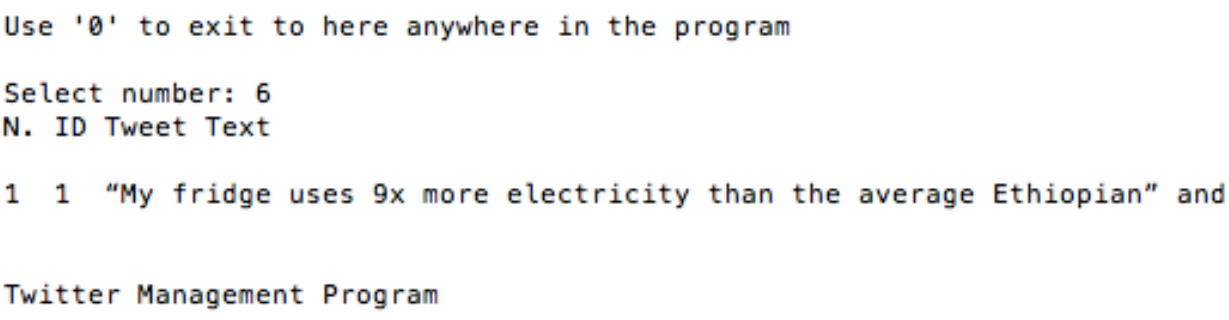
This image, similar to 3.1c, shows the validated data with blank spaces.

Figure 3.1f - Not adding the Bookmark; returning to previous interface



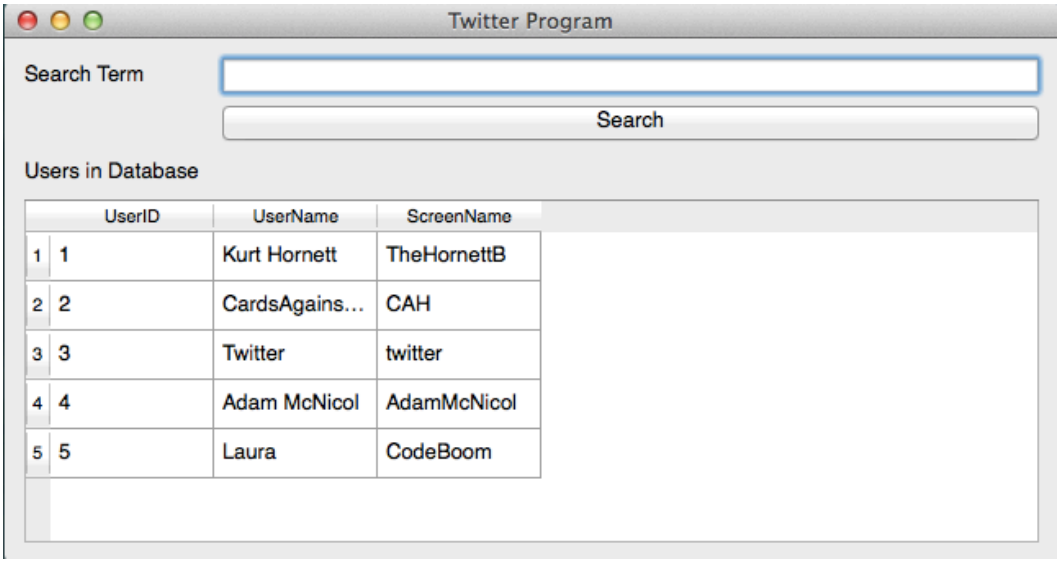
This images shows the user selecting 'N' instead of 'Y'.

Figure 3.1g - Proof of added bookmark



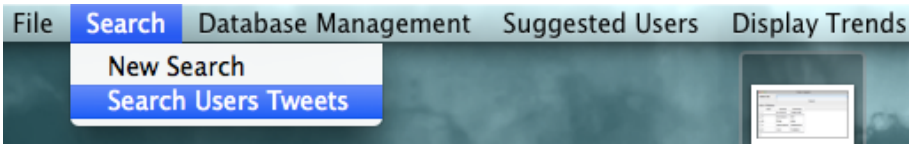
This image shows the tweet in the bookmark database

Figure 4a - State of System before Test



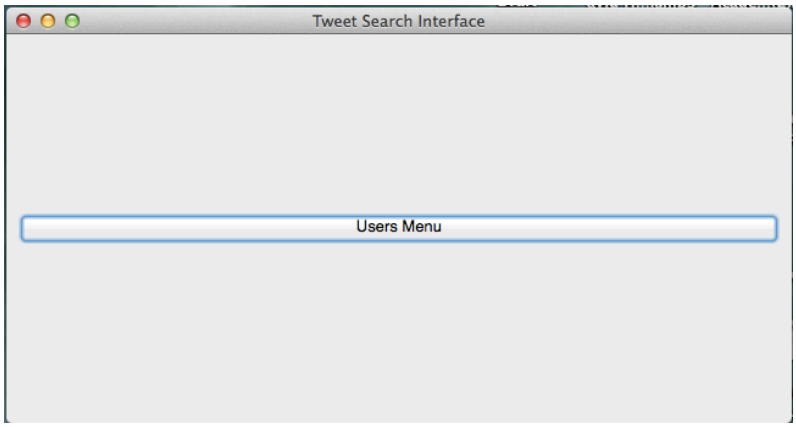
This image shows the state of the system before any test

Figure 4b - Selecting the Menu Action 'Search User Tweets'



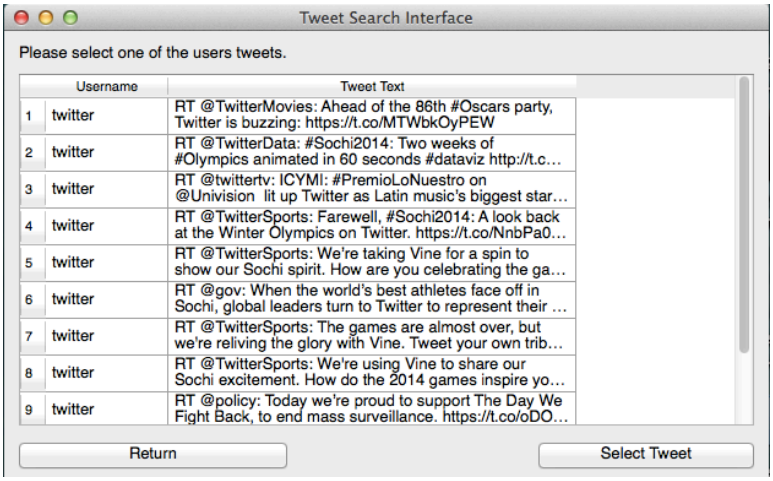
This images shows selecting the Menu Action to change the interface

Figure 4c - The Switched interface.



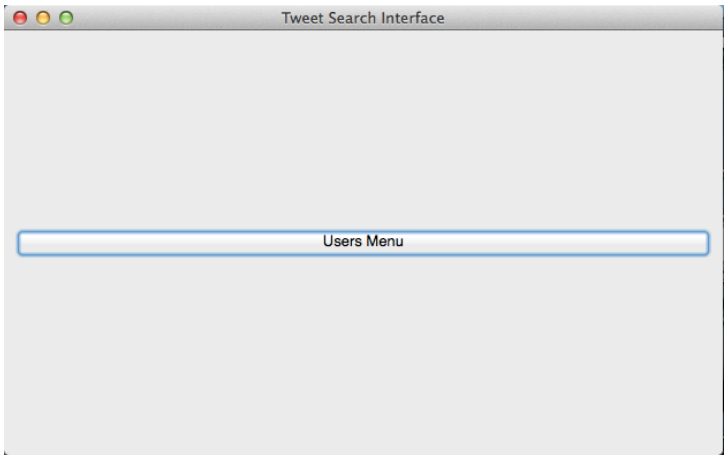
This image shows the changed interface

Figure 5a - State of system before test



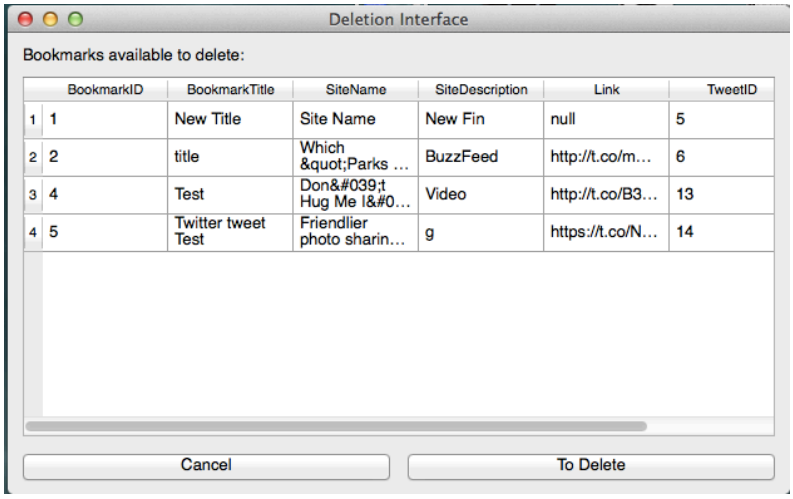
This images shows the state of the system before the test, with the return button in the lower left corner.

Figure 5b - The state of the system after selecting return



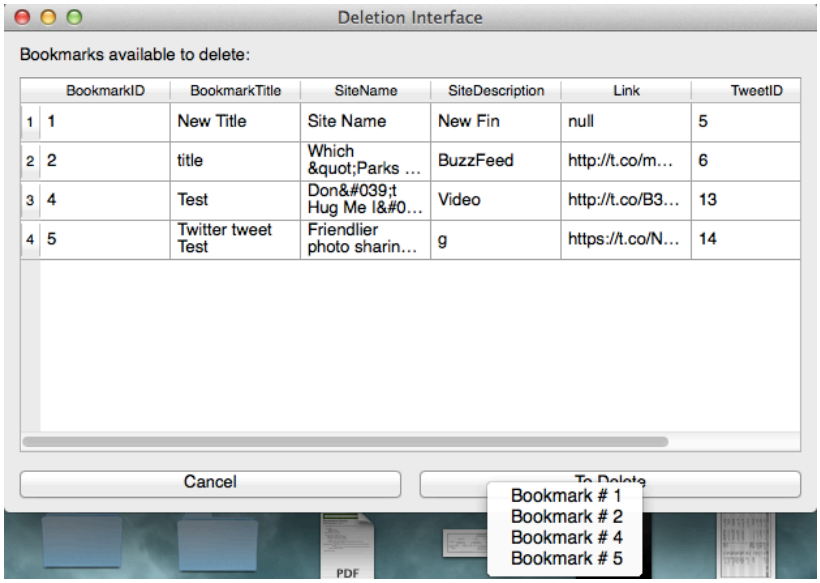
After return is selected the previous screen is shown

Figure 6a - State of system before test



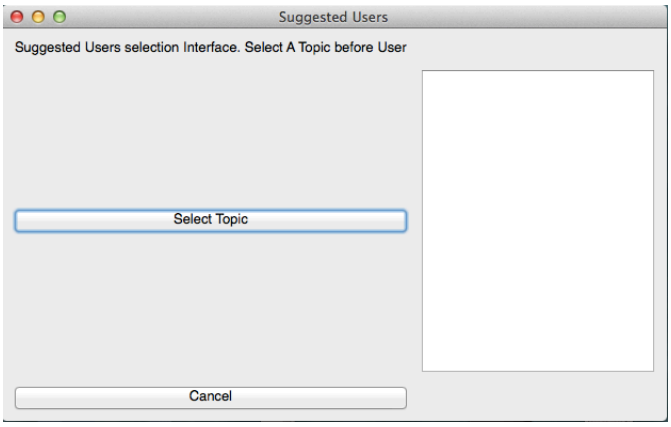
This shows the state of the system before the test takes place

Figure 6b - Clicking the 'To Delete Button'



This images shows the 'To Delete Button being selected and the list being shown.

Figure 7a - State of system before Test

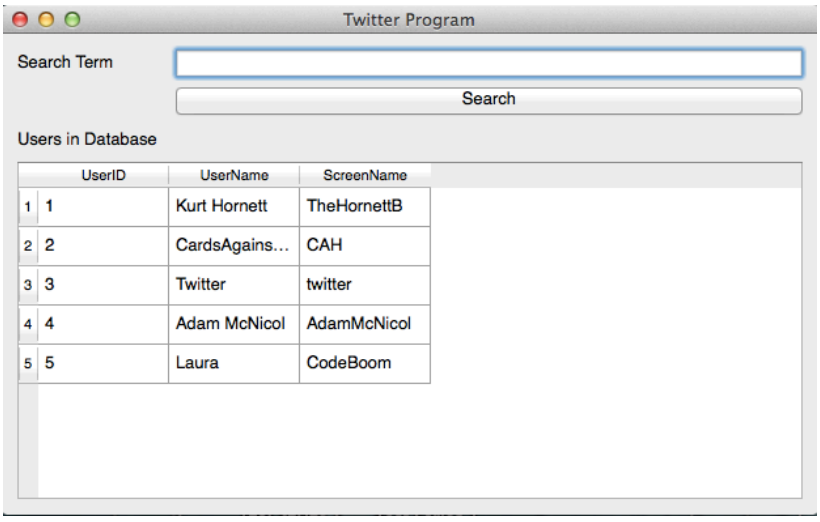


The system before the test began.

Figure 7b - Selecting the 'Select Topic Button

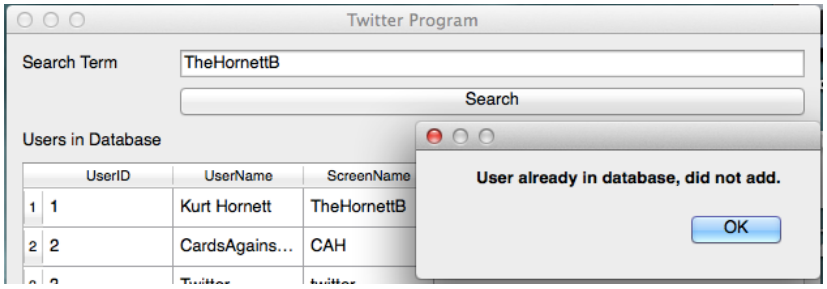
Selecting the 'Select Topic' button shows a list of topics

Figure 8a - State of system before test



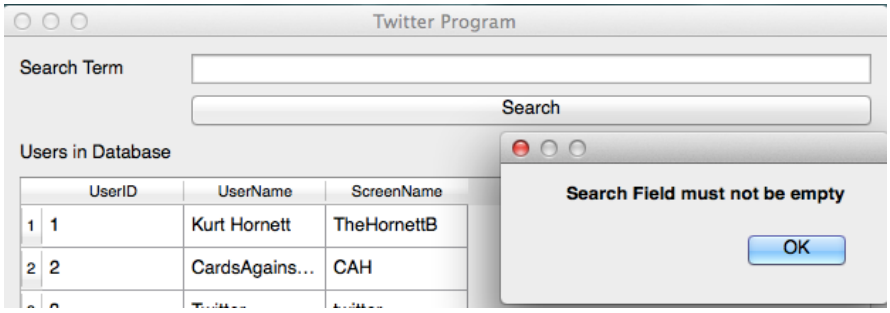
This image shows the interface before input.

Figure 8b - Inputting an already exiting user



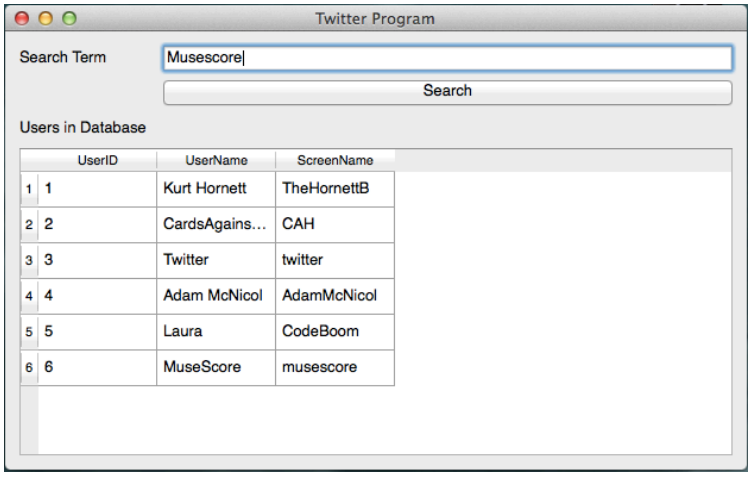
This images shows the error when adding an already existing user

Figure 8c - Inputting blank space



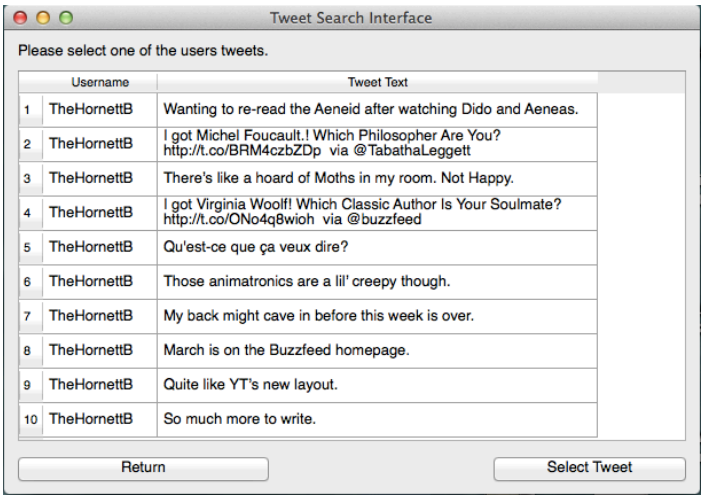
When a blank space is in-putted then this error is raised.

Figure 8d - Inputting the user 'Musescore'



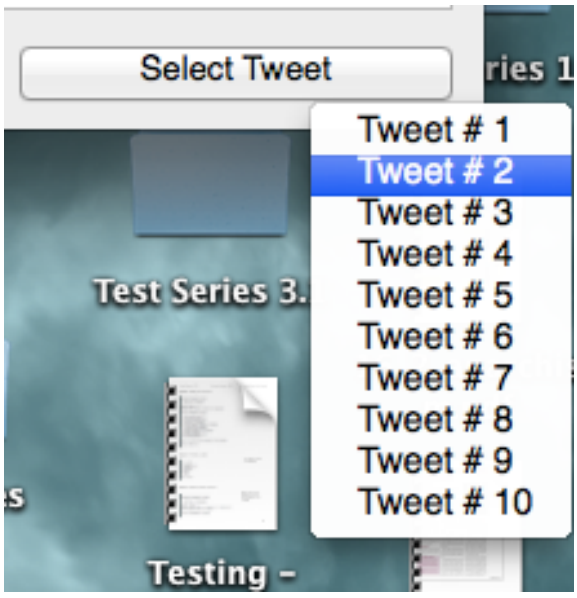
When a valid user is entered then it is added to the database as this images shows.

Figure 9a - State of program before test



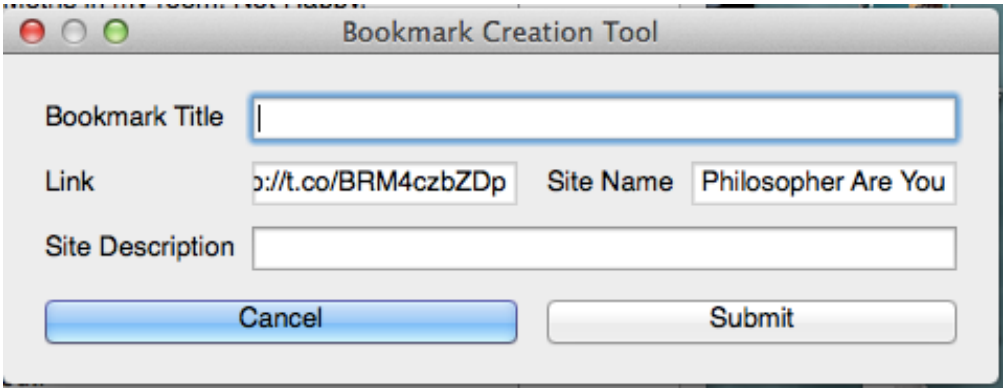
This image shows the state of the program before the test

Figure 9b - Selecting tweet



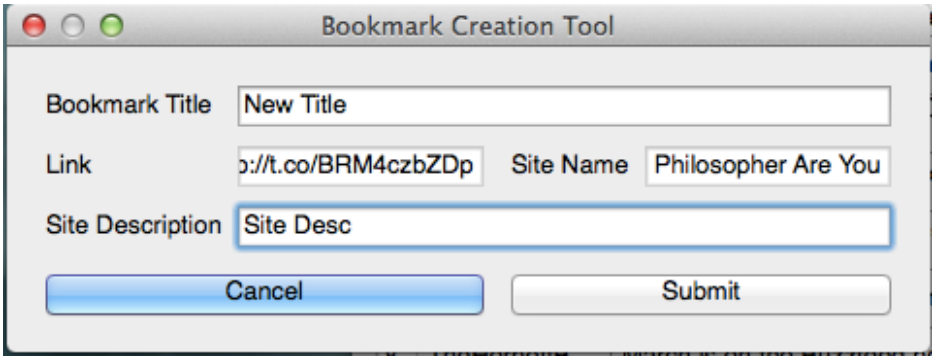
This image shows selecting the Tweet to bookmark.

Figure 9c - The Bookmark Creation Tool



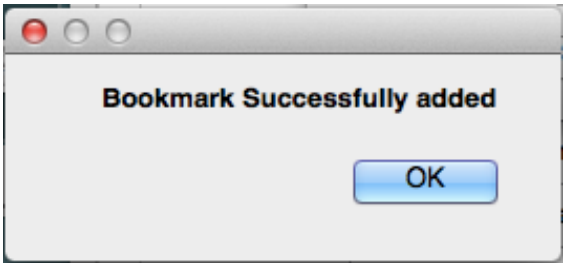
This shows the bookmark creation too that appears after selecting a tweet.

Figure 9d - Inputting information



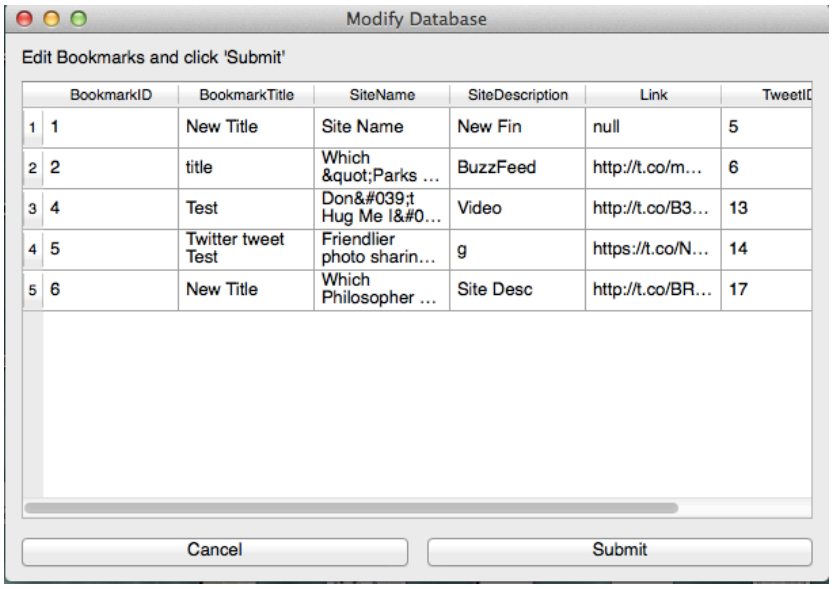
This image shows that user adding information for the bookmark.

Figure 9e - Clicking submit



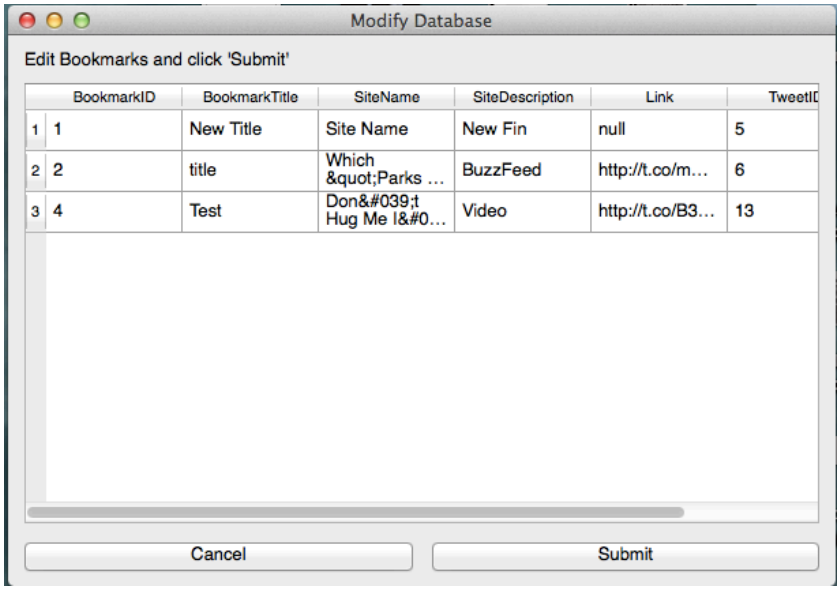
This image shows the dialog confirming that the bookmark was added

Figure 9f - Proof the bookmark was added



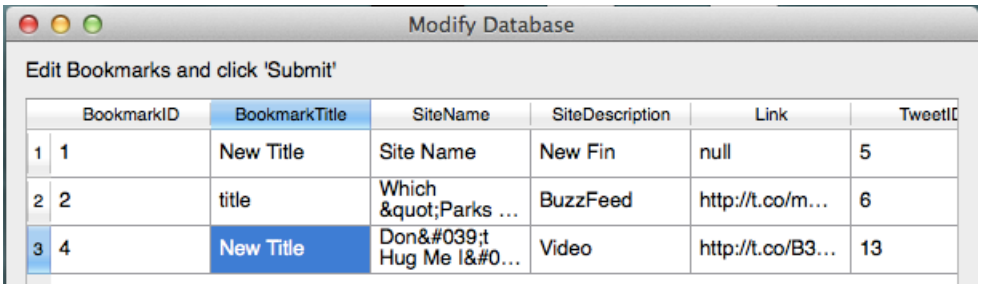
This image shows the Bookmark in the database with the ID 6

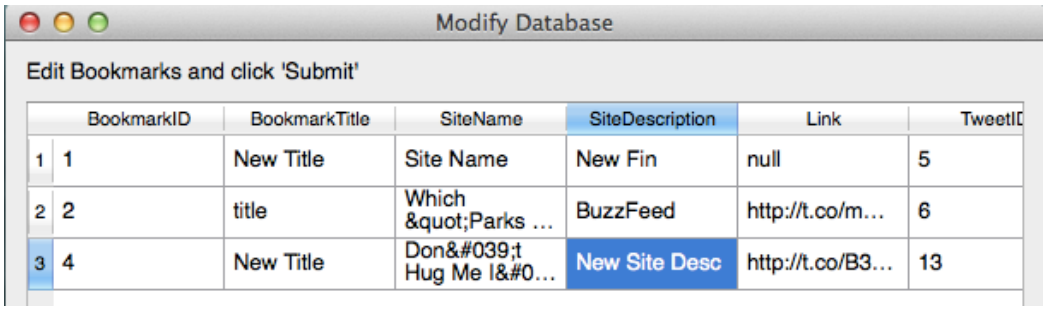
Figure 10a - The system before the test



This image shows the system before the test; the relational table view that the user can edit.

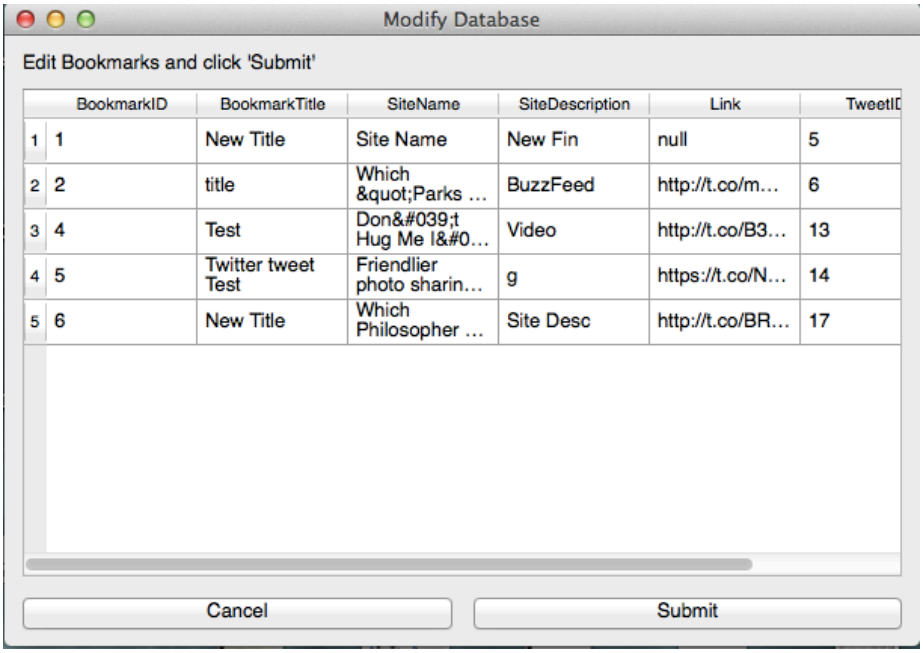
Figure 10b,c - Modifying the bookmarks





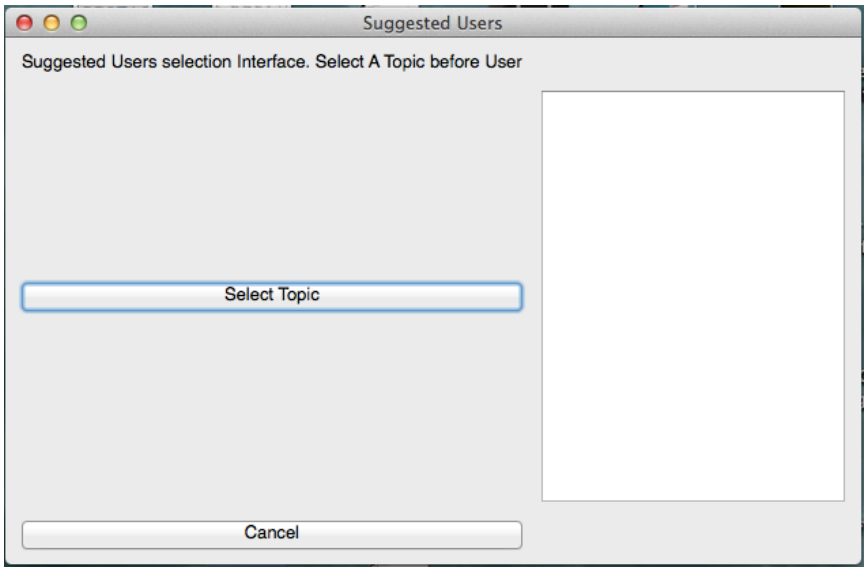
These images show the changed Title and Site Description

Figure 10d - Clicking Submit; the confirmation dialog



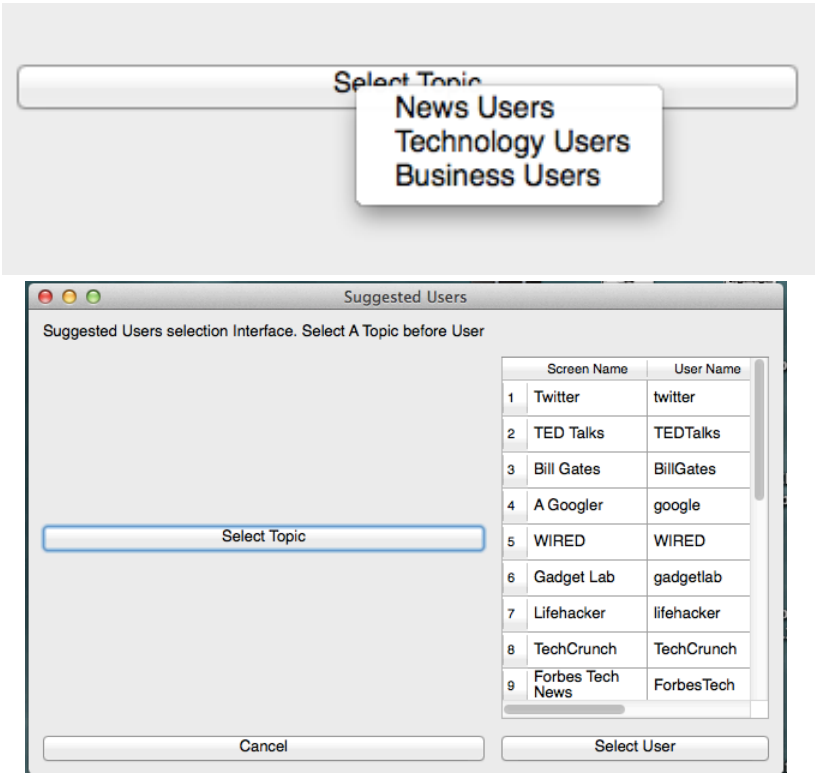
This dialog is displayed confirming the modification to the database

Figure 11a - State of system before test



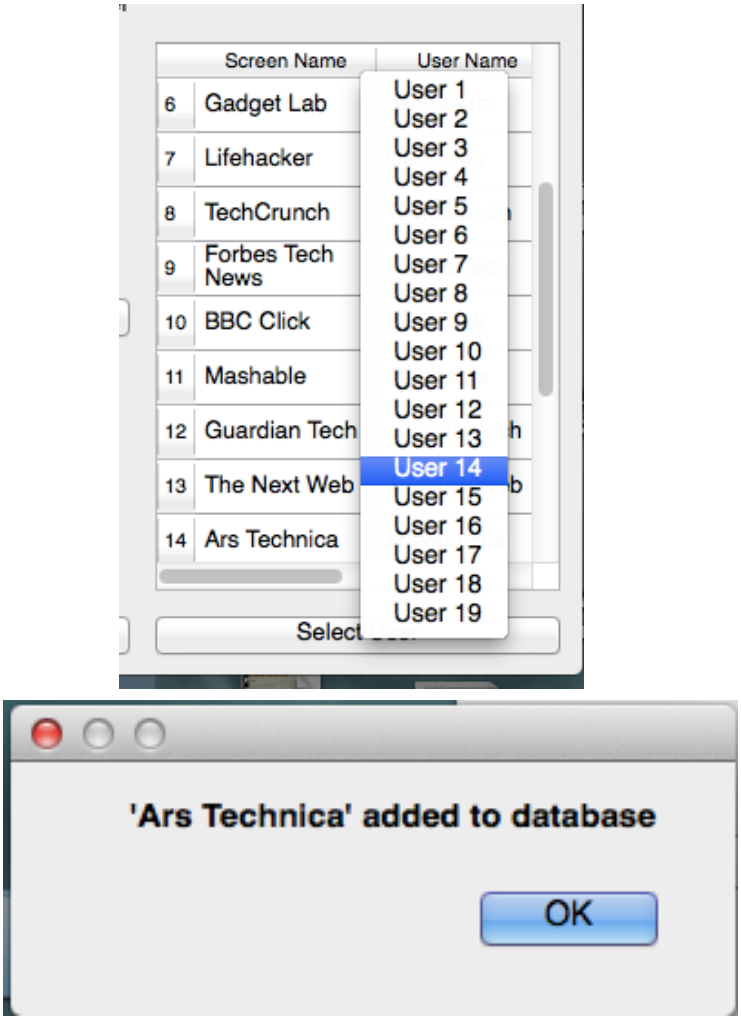
The system before the test

Figure 11b,c - Showing suggested users



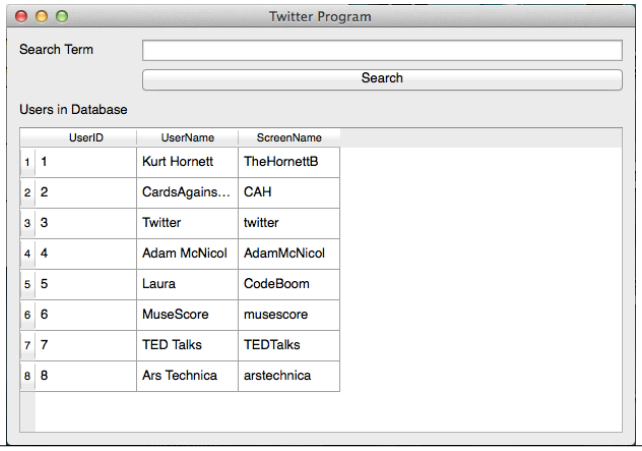
These images show selecting one of the topics to obtain suggested users from; the second shows the suggested users displayed in a table for the user to choose from.

Figures 11d,e- Adding the user to the database



These images show selecting a user from the list and then a confirmation that the user has been added to the database.

Figure 11f - Proof user is now in database



These images show selecting a user from the list and then a confirmation that the user has been added to the database.

4. Evaluation

4.1 Approach to Testing

In approaching the testing section I have used a mixed approach in that both bottom up and black box tests have been used. Black-box testing has been used to test rigorously the key aspects of the system and the validation checks in place. Bottom-up testing has been used to work through the base elements of the system followed by the next modules as the program was developed.

4.2 Problems

Problems during the testing phase occurred in only a few places and on the whole the system is rather stable. Test 5.4.2 failed in that instead of not deleting a bookmark it is deleted; this has come about through tricky programming on my part, as I have not written it so that the user is really given a choice in deletion, rather it acts more as a warning. Despite this, on the whole, all other tests have passed.

4.3 Strengths

The main strengths of my testing phase are, principally: the range of tests used that display a large proportion of what the system is capable of doing and thus giving a descriptive account of what the program is able to do and what it fails at in doing. Furthermore the methods of testing allowed for specific pinpointing of the problems that need to be fixed which allows for a meticulous re-working of parts of the system. All this shows an overall strong testing programme that highlights all the key points of the system. Lastly the use of sub test series has allowed for a greater insight into the key functions of the program from a testing perspective, insofar as testing the most specific elements possible.

4.4 Weaknesses

Contrary to the strengths pointed out above, there are also some key weaknesses with the testing programme and how testing was executed. Due to time constraints and testing methods it has been impossible to test all aspects of the system and so there is a lack of completeness to the testing programme, however a sufficient variety of tests from all aspects of the system have been included. Also the use of sub-test series has added to the overall amount of tests and thus increased the amount of time needed to carry out all the tests; this could be seen as a disadvantage as it may have taken away from testing other parts of the system, however, to re-iterate again, a wide range of tests have been included in the testing phase.

4.5 Reliability

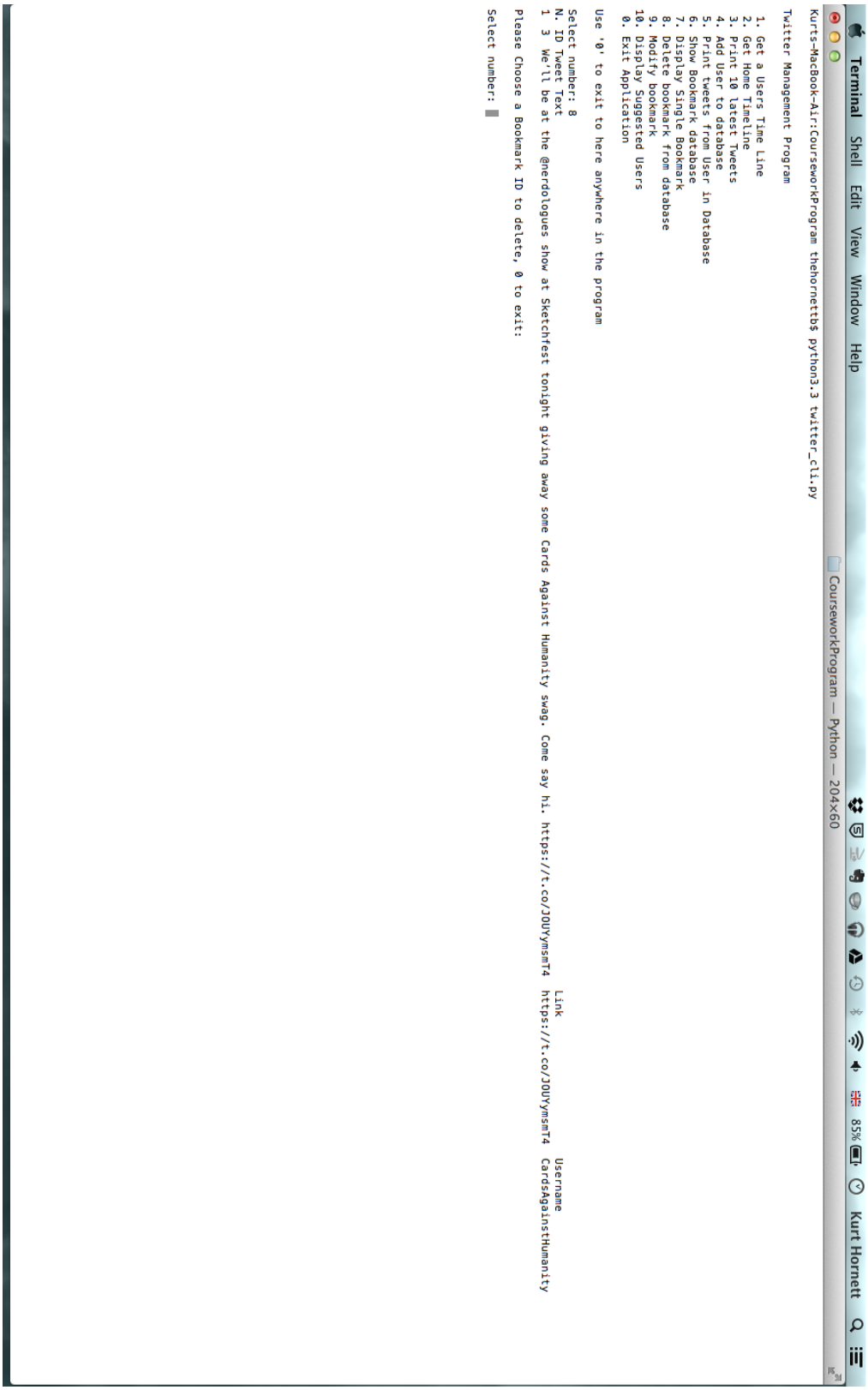
The reliability of the system refers to the system using and manipulating data without error and not allowing the user to create such errors in the program. On the whole, I do believe that the testing does show that the system is largely incredibly reliable, provided the user provides the correct inputs into the system. In all instances where some verification is needed and there is a limit to what the user should be allowed to input there are sufficient measure to make sure the user sticks within these predefined limitations of the system. The places where criticism of this may occur is in the ability to create Bookmarks with blank titles and descriptions as displayed in tests 3.4.1 & 3.4.3, and 5.3.2 & 5.3.3 which relies greatly on the user knowing what they are doing; however it is highly unlikely that such an error is to occur in the use of the system.

4.6 Robustness

In regards to the systems robustness, meaning the lack of infinite loops and errors that crash the program etc., the testing proves that the system is very robust, at no point is there an infinite loop that causes the user to have to quit the program, nor have there been any errors that crash the program requiring a restart of the system to continue use. A lack of robustness could be said to occur in test 5.4.2, in that the bookmark is deleted without a firm confirmation from the user, however this does not create any serious problem for the use and robustness of the overall system.

Appendices

Appendix 1



Appendix 2

```
Select number: 5
No.Screen Name    Tweet Text                                                                                                                       Links
1  BillGates      "My fridge uses 9x more electricity than the average Ethiopian" and other amazing energy stats http://t.co/LnwiU5K4CH http://t.co/JQzb3nq230http://t.co/LnwiU5K4CH
2  BillGates      This is promising: How 3D printing might unlock talent in poor countries: http://t.co/sd6qND1gBF http://t.co/pCW7z40Zcm           http://t.co/sd6qND1gBF
3  BillGates      I wish we'd had these when I was in school: students giving their own #TEDTalks: http://t.co/3Pn5diFR83 http://t.co/qi9dWxxLAF    http://t.co/3Pn5diFR83
4  BillGates      "A few unimaginably brave people took on the entire globe." @bhorowitz on why he's donating his book earnings: http://t.co/zb8caj9xg0  http://t.co/zb8caj9xg0
5  BillGates      Nice @Forbes piece on a high-tech pedicab and 6 other innovations from Africa: http://t.co/RMnTLiXiB2 http://t.co/gsPb1HaT3b      http://t.co/RMnTLiXiB2
6  BillGates      Good article about scientists who are trying to save lives by building a better potato: http://t.co/CsCEfzZe2m                  http://t.co/CsCEfzZe2m
7  BillGates      I wouldn't have guessed that a map of population &amp; income could be this beautiful: http://t.co/4UETKG9VeN via @NatGeo http://t.co/lUc8KxYqonhttp://t.co/4UETKG9VeN
8  BillGates      In 2009, #India accounted for 1/2 of the world's #polio cases. Here's how it became polio free, via @UNICEFIndia http://t.co/tWOVLZ7UtC     No Link
9  BillGates      3 myths about education, the Common Core, and why they're wrong: http://t.co/7WCOaKZdlr http://t.co/B7OFtSdZ0D                   http://t.co/7WCOaKZdlr
10 BillGates      .@TheEconomist  article of the week: How the Buffalo Bicycle symbolizes #Africa's potential: http://t.co/Wb0d2Mmb5o              http://t.co/Wb0d2Mmb5o
Commit Action (Y/N):
```

System Maintenance

1. Environment

1.1 Software

In implementing the system an array of software has been used:

- Python 3.3
- IDLE
- PyQt4
- Sqlite3
- RE module
- urllib module
- twitter
- SQLite Database Browser

1.2 Usage Explanation

Software	Usage Explanation
Python 3.3	Python, as a programming language, has been used due to my knowing this language principally as opposed to another. Further the language is easily understood compared to others.
IDLE	Easy to use editor for the Python Programming Language. Code highlighting aides understanding. Syntax error checking aides fixing issues.
PyQt4	Used to create GUIs. Works seamlessly with the python syntax. Creates effective and easy to use interfaces.
Sqlite3	Used for database management in the CLI. Simple to use and similar to native python modules in its use.
RE Module	Module used for regular expressions. Easy to use and runs efficiently.
urllib module	Used for fetching and manipulating HTML Documents.
twitter	Python wrapper for the Twitter API, used as it works with Python 3.3 and for its ease of use compared to other twitter wrappers.
SQLite Database Browser	Used to view the databases used in the program.

1.3 Features Used

Software	Features used
Python 3.3	This was used to develop and to run the program in both GUI and CLI form. Must be included to run the program on the Client's system
IDLE	This was used to write and run the Python scripts. Also used to save and update any file used by the system. Provided a highlighted syntax which allows ease of development.
PyQt4	Used in the Python to create a GUI. Multiple classes and functions of this have been used to create the components of the GUI
Sqlite3	Used principally in the CLI for interacting with the databases that are used to store data on the system. Simple to use DBMS
RE Module	Module included with the Python system and used in the program to implement regular expressions in order to parse HTML files pulled from the internet.
urllib Module	Module included with the Python system which allows the system to interact with HTML documents online by pulling and storing them in the system.
twitter	Downloaded module that has been used as a wrapper to interact with the Twitter API in the system. Chosen as it is simple to use and to understand and uses the same syntax that is used with the raw Twitter API documentation.
Sqlite Database Browser	Used to look at and edit databases in the system where the system itself does not handle the elements needed to be edited, for example the user of the system is unable to edit the Tweets added to the system, only the bookmarks and the users.

2. System Overview

2.1 Logging in

This part of the system is used to verify and allow the user to log into their twitter account, in order to use any of the features of the main system. The user is prompted to enter a key gained from Twitter by utilizing the user's computer's main Internet browser and the Command Line Interface of choice (Either the computer's terminal or the IDLE shell). The

system does not allow for the user to change those details at a later date, so only one user can use the program. Also the system cannot re-authorize the user and so it can be accessed by anyone after logging in. Despite this the program does not allow for any posting or changing the user's details at twitter and only allows access to what is stored locally in the computer (i.e. the database). Handled in the CLI as opposed to the GUI due to how the twitter wrapper works when interacting with Oauth.

2.2 User Search Interface

This interface allows the user of the system to be able to search for a user from the Twitter API to add to the database and to display a table of these users so that the user of the system is able to know that a user has been added and to know who is already in the database. It is a QMainWindow Class that contains a QLabel, or text box, with the text: 'Search Term', followed by a QLineEdit where the user is able to enter a screen name of a user of Twitter in order to be able to add them to the database. A QPushButton, is beneath this, which when clicked will search for a User from the Twitter API and add it to the database.

On this interface is also a QSQLTableModel that displays the User Table of the database so that the user knows who has already been added to the database. This is updated every time that a user is added to the Database, after the QPushButton is clicked.
The interface primarily is used in order to add new users to the database, by entering relevant information – a username – into a search field above a table, which contains the users already stored in the database. Regular buttons allow the user to navigate this part of the interface, especially the use of the main button, which initiates the search and interaction with the API, which adds a new user to the database.

2.3 Search User Tweets Interface

This interface is used to search for the tweets of a user already stored in the database. It allows the user of the system to select a user from the system's database and display some of the latest tweets from that user in order to add them to the database of bookmarks that contains what the user wishes to add to that database.

This interface has multiple layouts that are combined into a QStackedLayout. It has inherited the QMainWindow Class. On the first interface is simply a QPushButton that when clicked displays a QMenu that contains a list of QActions of the users in the database that the user is able to search the latest tweets of.

After clicking a QAction from the list a new interface is displayed that contains a QTableWidget with the information from the Twitter User obtained from the API in a request. On this interface are also two QPushButtons which allow the user to return to the previous interface, 'Cancel', or select a Tweet for Bookmarking, 'Select Tweet'. When the latter is selected another QMenu wih QActions is displayed with a list of the tweets that can be bookmarked.

Put simply this interface allows the user to search the tweets of a user already stored in the database; this is done easily through the use of button that displays a list of the users in the database which the user can then select. After this a table is shown with the information of the latest tweets of the user, which the user can then select to bookmark. This is initiated by the use of buttons on the interface that display a list of the bookmark-able tweets. Buttons are there that also allow the user to exit this process to the main interface – The User Search Interface.

2.4 Bookmark Creation Tool

This part of the interface is inherited from QDialog and appears when the user clicks a QAction in the Search User Tweets Interface. It contains 4 QLabels and 4 QLineEdits for each QLabel, each pertaining to some aspect of the Bookmark to be created (Title,SiteName,SiteDesc,Link). Two of these Line Edits (Line and SiteName) are read only as they contain information that has been retrieved from the Twitter API.

Also there are two QPushButtons, one that allows the user to cancel the operation and one that adds the bookmark with the information in the QLineEdits to the database. After pressing on the latter QPushButton either a QMessageBox confirming the added bookmark appears or a QErrorMesssage appears notifying there was an error, and the bookmark wasn't added.

This interface is used to add bookmarks to the database. It has many text boxes with relevant labels that allow the user, for some, to be able to enter information that is used for the bookmark, including the Bookmark Title and the Site Description. The user is then given the choice to cancel the process or to go ahead and add the bookmark to the database. This is a dialog box that is separate from the main interface.

2.5 Modify Bookmark Interface

This interface, also a QMainWindow class, is used for modifying bookmarks in the database. The interface contains, primarily, a QSqlRelationalTable that allows for foreign keys to be displayed correctly in the table. To make changes the user simply has to click on the field they wish to edit and press enter. In this table model is the Bookmark Table from the database. Underneath this element are two QPushButtons, one that allows the user return to the main interface, the other when clicked pushes the changes made in the QSqlRealationalTable to the database, if successful the a QMesssageBox appears that explains that the operation was successful, if not the a QErrorMessage appears explaining that there was an error.

Simply, this interface allows the user, through the table that constitutes the majority of the space of this interface, to edit the bookmarks that have already been installed in the interface. By editing the fields in the table and pressing the return button on the keyboard the user can make changes to any aspect of the bookmark they may wish, pushing the changes made with a press o the submit button found in the bottom corner of the interface. Here again is a means of returning to the main interface by pressing a cancel button.

2.6 Delete Bookmark Interface

This interface, on the whole, is almost identical to the Modify Bookmark Interface and thus needs not a detailed explanation. It also contains a QSqlRelationalTable and two push buttons, however this time the user cannot edit the table and the second QPushButton displays a QMenu of QActions which contains the bookmarks that can be deleted from the database. When one of these QActions is clicked a QMessageBox appears that tells the user that the Bookmark is about to be deleted subsequently deleting the bookmark.

This interface, identical to the modify interface, allow the user to delete bookmarks they wish from the database. Instead of a submit button a delete button, which displays a menu of the delete-able bookmarks, is included.

2.7 Suggested User Interface

This interface is used to add new users to the database maybe unbeknown to the system's user by utilizing Twitter's Suggested Users. When instantiated it has a QLabel at the top of the QGridLayout, This is followed by a QPushButton on the left hand side of the interface and an empty QTableWidget on the right hand side. There is one QPushButton on the left hand side of the interface that allows the user to return to the main interface.
The main QPushButton that allows the user to select a QAction from a QMenu after having pressed the QPushButton collect a list of Users from the Twitter API; only three options are selectable; Business; Technology; and news. After selecting an option the interface gains some new features.

After selecting an option the QTableWidget becomes populated with a list of users from the call to the Twitter API, furthermore a new QPushButton appears that is populated by the list of users generated by the Twitter API. By selecting a QAction from this QMenu a QMessageBox appears informing the user who is being added to the database and then adds the user to the database. An error message appears if necessary.

Possibly the most complex of the interface in this system, as it has some features that are not available until other actions have been performed, this interface allows the user of the system to find and add suggested users to their database, in order to bookmark their tweets at a late date. The suggested users are found by clicking a button that displays the topics of which people can be found – 3 are included in the system. After doing this a table appears with the information of the users that can be added to the system, along with a new button that allows the user to select a user to add to the database. All throughout this interface a cancel button is apparent that allows the user to return to the initial interface.

3. Code Structure

3.2.1 Database Query Functions

Database interactions were handled by two specially made functions that allow the database to be accessed without rewriting the code necessary to do so, Found in §10.2 'twitter_databse.py' and §10.11 'sql_gui_misc.py'. The following example will be taken from the former of the two from lines 22-7 and 29-35

```
22. def query(sql,data,db):
23.     with sqlite3.connect(db) as db:
24.         cursor = db.cursor()
25.         cursor.execute('PRAGMA foreign_keys = ON')
26.         cursor.execute(sql,data)
27.         db.commit()
```

And:

```
29. def searchQuery(sql,data,db):
30.     with sqlite3.connect(db) as db:
31.         cursor = db.cursor()
32.         cursor.execute('PRAGMA foreign_keys = ON')
33.         cursor.execute(sql,data)
34.         results = cursor.fetchall()
35.         return results
```

What both of these functions show is the use of the use of a function to access the database, which either requires a commit or search results to be returned. The code has been structured in this way to allow the versatility of other functions that require calls to the database, in such a way that they can be used anywhere in the program and can take any data put into them through the parameters they take.

Furthermore, the name of the database is a variable that is passed into the function rather than being pre-defined and unchangeable, this is because bother the CLI and the GUI interfaces use these two functions but use different database files. What passing the 'db' variable in allows is for the many different aspects of the program to use the same function while using different database files.

3.2.2 The MainWindow Class

This is taken from §10.3 'MainWindow.py', Lines 22-117. It is used as a meta-window that implements and instantiates all of the other windows in the program. This is has been done so that any amount of new interfaces can be easily implemented into the program, if new functions were ever to be added to the program.

```
22. class MainWindow(QMainWindow):
23.     def __init__(self):
24.         super().__init__()
25.         #Set Window Title
```

```
26.         self.setWindowTitle('Twitter Program')
27.
28.         #Create Twitter Object
29.         self.createTwitterObject()
30.
31.         #Create Menubar for interface - Add Each Menu
32.         self.menuBar = QMenuBar()
33.         self.fileMenu = self.menuBar.addMenu('File')
34.         self.searchMenu = self.menuBar.addMenu('Search')
35.         self.managementMenu = self.menuBar.addMenu('Database Management')
36.         self.suggestedMenu = self.menuBar.addMenu('Suggested Users')
37.         self.trendsMenu = self.menuBar.addMenu('Display Trends')
38.
39.         #Add Actions to each Menu
40.         self.logOutAction = self.fileMenu.addAction('Log Out')
41.         self.quitAction = self.fileMenu.addAction('Quit')
42.         self.searchAction = self.searchMenu.addAction('New Search')
43.         self.tweetAction = self.searchMenu.addAction('Search Users Tweets')
44.         self.deleteAction = self.managementMenu.addAction('Delete Entries')
45.         self.modifyAction = self.managementMenu.addAction('Modity Databases')
46.         self.suggestedAction = self.suggestedMenu.addAction('Display Suggested
    Users')
47.         self.trendsAction = self.trendsMenu.addAction('Display Trends')
48.
49.         #Set Menu Bar
50.         self.setMenuBar(self.menuBar)
51.
52.         #Import Layouts
53.         self.searchInterface = SearchWindow(self.twitter)
54.         self.tableInterface = TableViewWindow()
55.         self.tweetSearchInterface = TweetInterface(self.twitter)
56.         self.deleteInterface = DeleteInterface()
57.         self.suggestedInterface = SuggestedInterface(self.twitter)
58.
59.         #Create Layout
60.         self.layout = QStackedLayout()
61.         self.layout.addWidget(self.searchInterface.searchWidget)
62.         self.layout.addWidget(self.tableInterface.tableWidget)
63.         self.layout.addWidget(self.tweetSearchInterface.tweetsWidget)
64.         self.layout.addWidget(self.deleteInterface.deleteWidget)
65.         self.layout.addWidget(self.suggestedInterface.suggUserWidget)
66.
67.         #Set Main Widget
68.         self.widget = QWidget()
69.         self.widget.setLayout(self.layout)
70.         self.setCentralWidget(self.widget)
71.
72.         #Add Action Actions
73.         self.searchAction.triggered.connect(self.newSearch)
74.         self.tweetAction.triggered.connect(self.tweetSearch)
75.         self.modifyAction.triggered.connect(self.modifyBookmarks)
76.         self.deleteAction.triggered.connect(self.deleteBookmarks)
77.         self.suggestedAction.triggered.connect(self.displaySuggested)
78.         self.quitAction.triggered.connect(self.quit_)
79.
80.         #Actions for sub-classes
81.         self.tableInterface.cancelButton.clicked.connect(self.cancelAction)
82.         self.deleteInterface.cancelButton.clicked.connect(self.cancelAction)
83.         self.suggestedInterface.cancelButton.clicked.connect(self.cancelAction)
84.
85.     def newSearch(self):
86.         self.layout.setCurrentWidget(self.searchInterface.searchWidget)
87.         self.setWindowTitle('Twitter Program')
```

```
88.     def tweetSearch(self):
89.         self.layout.setCurrentWidget(self.tweetSearchInterface.tweetsWidget)
90.         self.setWindowTitle('Tweet Search Interface')
91.     def cancelAction(self):
92.         self.layout.setCurrentWidget(self.searchInterface.searchWidget)
93.         self.setWindowTitle('New Search')
94.     def modifyBookmarks(self):
95.         self.tableInterface.createTableModel()
96.         self.layout.setCurrentWidget(self.tableInterface.tableWidget)
97.         self.setWindowTitle('Modify Database')
98.     def deleteBookmarks(self):
99.         self.deleteInterface.createTableModel()
100.            self.deleteInterface.createDeleteAction()
101.            self.layout.setCurrentWidget(self.deleteInterface.deleteWidget)
102.            self.setWindowTitle('Deletion Interface')
103.        def displaySuggested(self):
104.            self.layout.setCurrentWidget(self.suggestedInterface.suggUserWidget)
105.            self.setWindowTitle('Suggested Users')
106.        def createTwitterObject(self):
107.            consumer_key = getConsumerKey()
108.            MY_TWITTER_CREDS = os.path.expanduser('~/.login_credentials')
109.            if not os.path.exists(MY_TWITTER_CREDS):
110.                oauth_dance("KurtsApp",'Y4zkic6lw0Hu6uB3sNVH4Q', consumer_key,
111.                            MY_TWITTER_CREDS)
112.            oauth_token, oauth_secret = read_token_file(MY_TWITTER_CREDS)
113.            #Creation of twitter object with authorized detailes.
114.            self.twitter = Twitter(auth=OAuth(
115.                oauth_token, oauth_secret, 'Y4zkic6lw0Hu6uB3sNVH4Q', consumer_key))
116.        def quit_(self):
117.            window.close()
```

The bulk of the information in this function, that is necessary to understand why it has been put into its own separate class, is from before the creation of methods in the class. By importing from external files, which have within them other QMainWindow classes, makes it easy to add new interfaces into the program. By simply importing the module, creating a new menu action, instantiating the main widget, and adding it into the QStackedLayout, means that there is a great amount of flexibility for expanding the system beyond what it currently does.

Further it allows greater control and debugging of specific elements of the program as all the information about the layouts are dispersed among the multiple files of the system, which have been structured in a way so that all relevant pieces of code are grouped together. This also means that if something breaks in a local section of code, for example in the 'TweetIterface.py' module it would not break the whole system at once as a system that was combined into one file would do.

It also allows for some continuity within the separate interfaces, to exemplify this, the cancelAction function of the system is used for buttons that exist throughout the system and not in one single interface. The use of a class to bring all of these together means that you do not have re-write familiar functions, making the system much easier to read, and to comprehend.

3.2.3 The Main Function

This is a function that appears at the end of almost every document, the example here taken from §10.3, Lines 121-6:

```
121.   if __name__ == '__main__':
122.       app = QApplication(sys.argv)
123.       window = MainWindow()
124.       window.show()
125.       window.raise_()
126.       app.exec()
```

What this function allows is for the running of all individual modules separately from the main window class; this allows for testing and debugging to be carried out much more effectively, especially for specific parts of the system, independent of system as a whole.

4. Variable Listing

The Design section, pp30-90, contains a data dictionary of which this is based of.

Variable Name	Purpose	Section	Line Number
choice	Stores the users choice	10.1 pp125-134	50,53,130,270,287
consumer_key	Stores the keys needed from Twitter to Authorize	10.1 pp125-134	59,283
home_timeline	Stores the users Home timeline	10.1 pp125-143	80
timeline	Gets the timeline of a user from twitter	10.1 pp125-134	102
count	Used as an iterator	10.1 pp125-134	115,118,125,151,155
YN	Stores a user's choice when a string is needed	10.1 pp125-134	140,142-3,
IDList	Stores list of ID's from the database	10.1 pp125-134 10.2 pp134-139	158,160,161,375 40
changes	This stores a string of the changes to be made to a bookmark	10.1 pp125-134	184,186-7
suggUserSlugs	A list of 'slugs' which are used to obtain suggested users	10.1 pp125-134	190,196
SuggUserChoice	An implementation of choice specifically for the suggested users	10.1 pp125-134	209
userList	List which stores the choice of the user to check the database.	10.1 pp125-134	220-1,234-5
suggUsers	A dictionary of all the suggested users from Twitter	10.1 pp125-134	274
users	A list of users from the database	10.1 pp125-134	315,330
List	A list of all the information of the bookmarks in the database	10.1 pp125-134	343
SingleBookmark	Stores information of a single bookmark	10.1 pp125-134	352
sql	Used in various locations to store sql queries	10.2 pp134-139	46,53,60,67,73,90,118,125

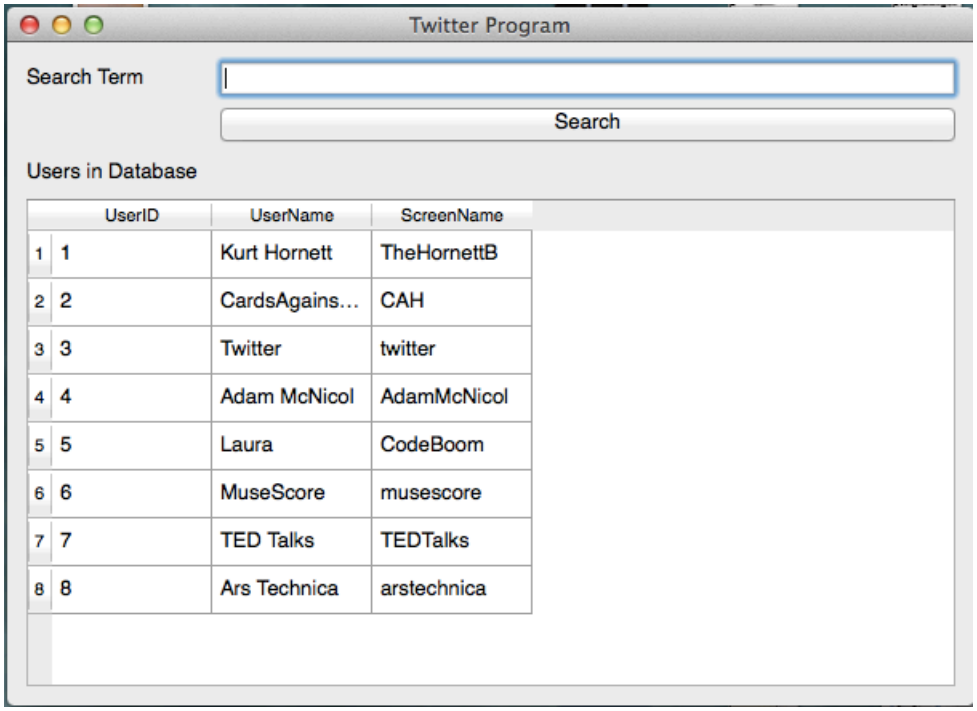
self.userChoice	Used as iterator to find users choice	10.4 10.8	pp142-146 pp156-160	81,83 100,102
tweetListLabels	Stores labels to be used for table	10.4	pp142-146	90
self.timeline	Stores a user's timeline information	10.4	pp142-146	87,96,127
self.userList	Stores a list of uses from the database	10.4 10.7	pp142-146 pp152-156	114 66
self.userNumber	Creates the number of user from length of self.userList	10.4	pp142-146	115
bookmarkInfo	Stores certain information for bookmark Tool	10.4	pp142-146	129,133
regex	Stores the regular expression for finding title	10.4	pp142-146	138
url	Stores information from a HTTPRequest	10.4	pp142-146	139
html	Converted url	10.4	pp142-146	140-1
self.SiteName	Stores the title from the regular expression search	10.4	pp142-146	142
num	Stores number of bookmarks	10.6	pp148-152	66
self.bookmarkList	Stores list of bookmarks from the database	10.6	pp148-152	66
self.deleteChoice	Stores the required integer to delete bookmark	10.6	pp148-152	71,75
columnNames	Stores the names for the table to be produced	10.8	pp156-160	75
Self.suggUsers	Stores the suggested users gathered from Twitter	10.8	pp156-160	65,67,69,74,89
data	Tuple of information needed for suggested users	10.8	pp156-160	104
tweetData	Tuple with the information needed for adding bookmarks	10.10	pp162-166	75
tweetID	Stores the latest tweetID from the	10.10	pp162-166	77

	database		
response	Stores users response in the create_database file	10.12 pp169-172	14

5. System Evidence

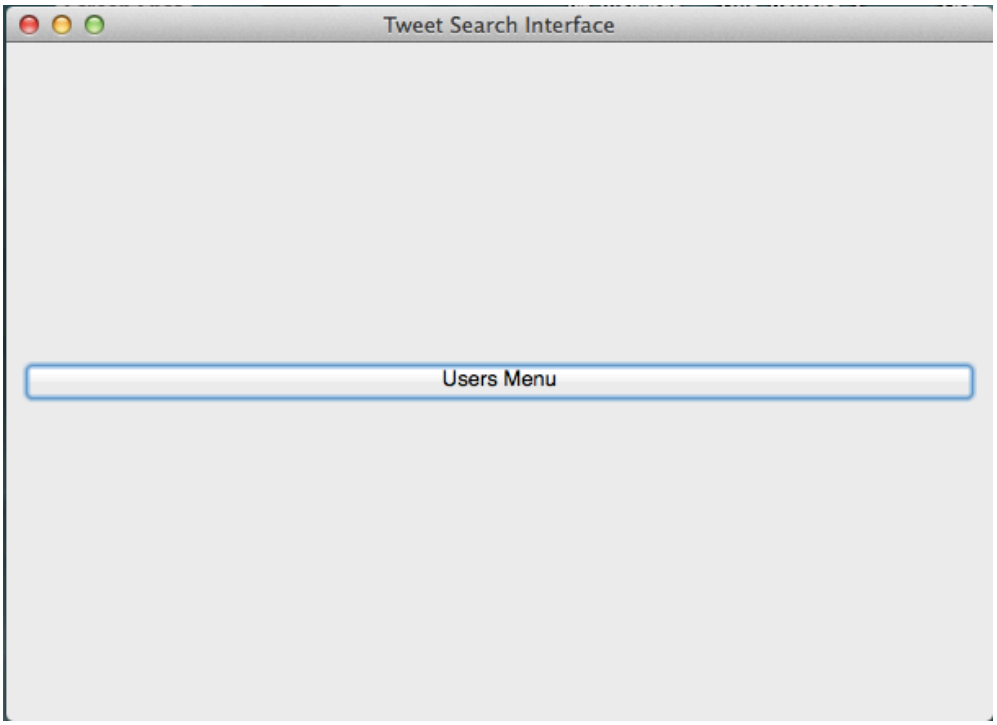
5.1 User Interface

5.1.1 – The Search User Interface

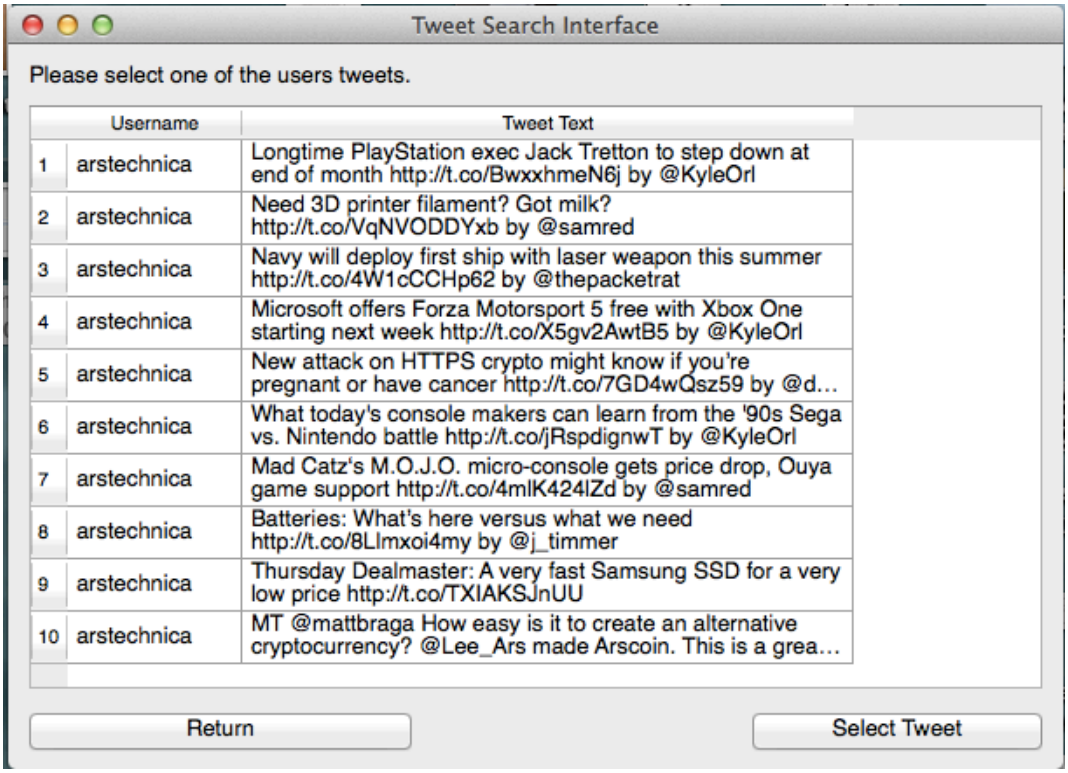


The first screen of the interface after logging in, allows for the searching of users and displaying already existing users in the user database.

5.1.2 – Components of the Tweet Search Interface

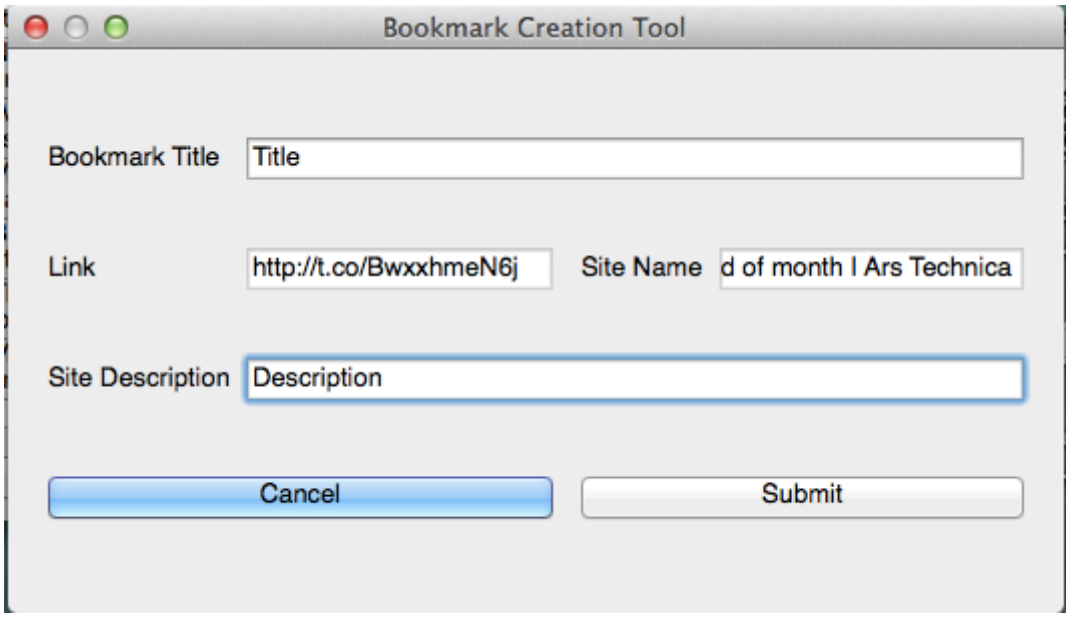


The Tweet search interface's first screen, allows the user to select a user to display their tweets.



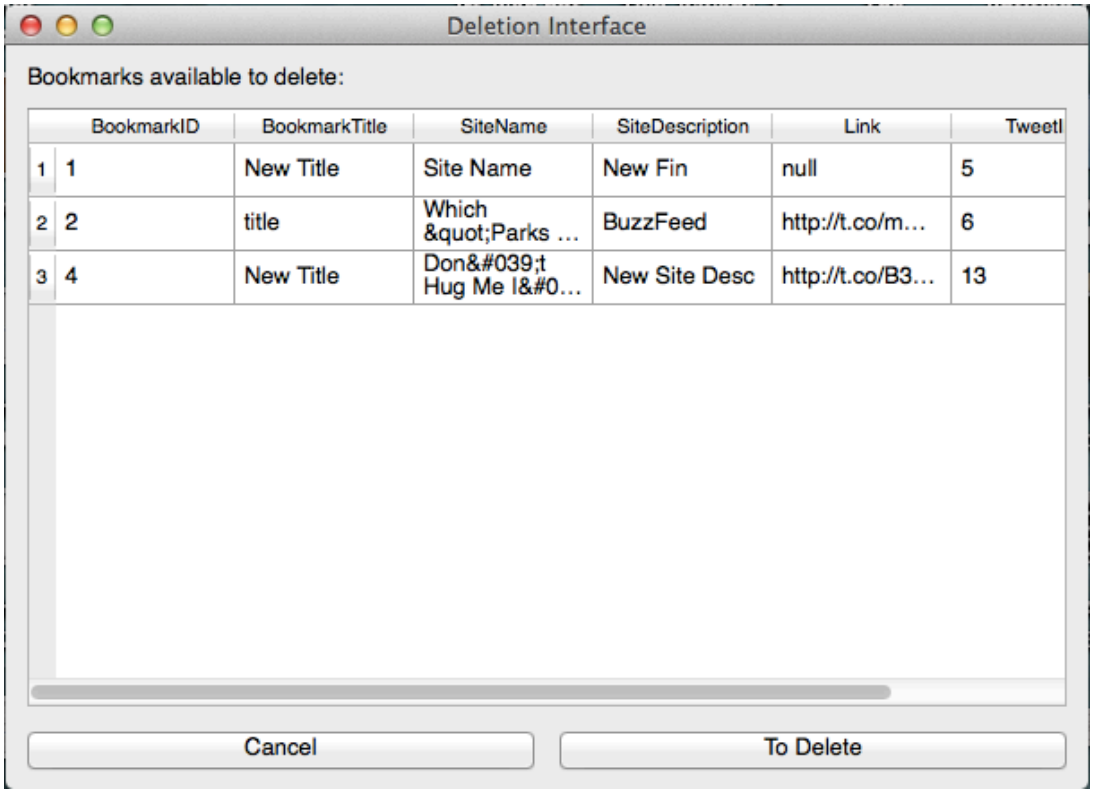
This Interface displays the tweets from the chosen user, which allows the user to select a tweet to save.

5.1.3 – The Bookmark Creation Tool



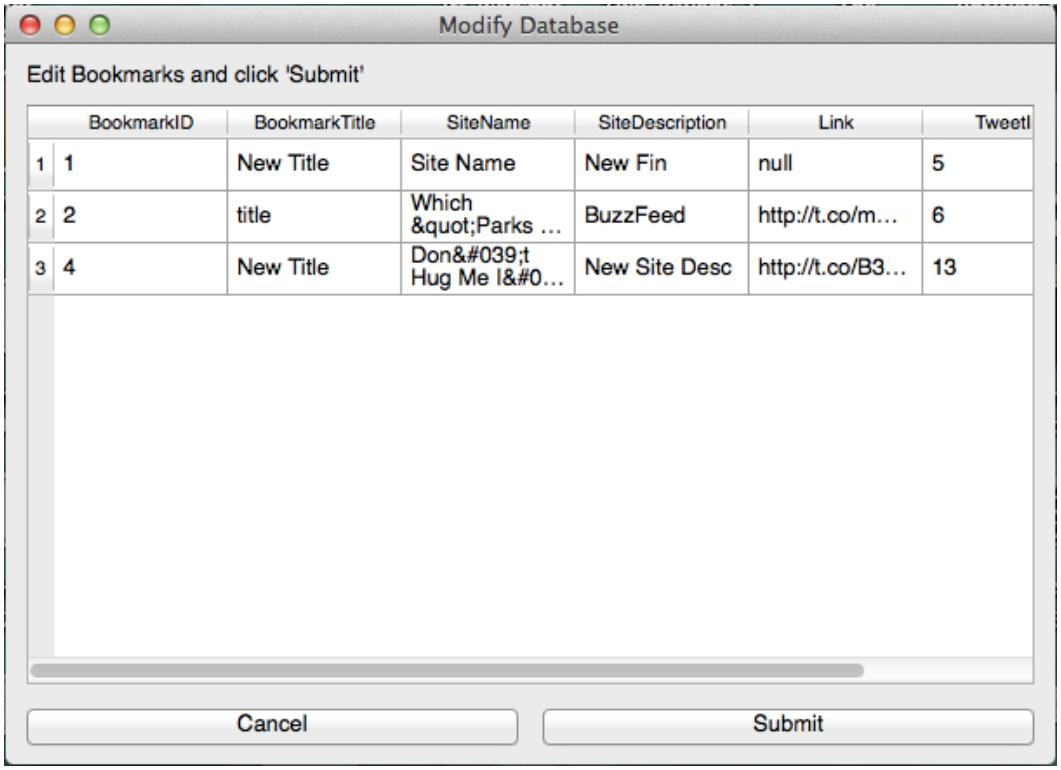
The Bookmark Creation Tool which allows the user to create and add a bookmark to the system's database.

5.1.4 – The Deletion Interface



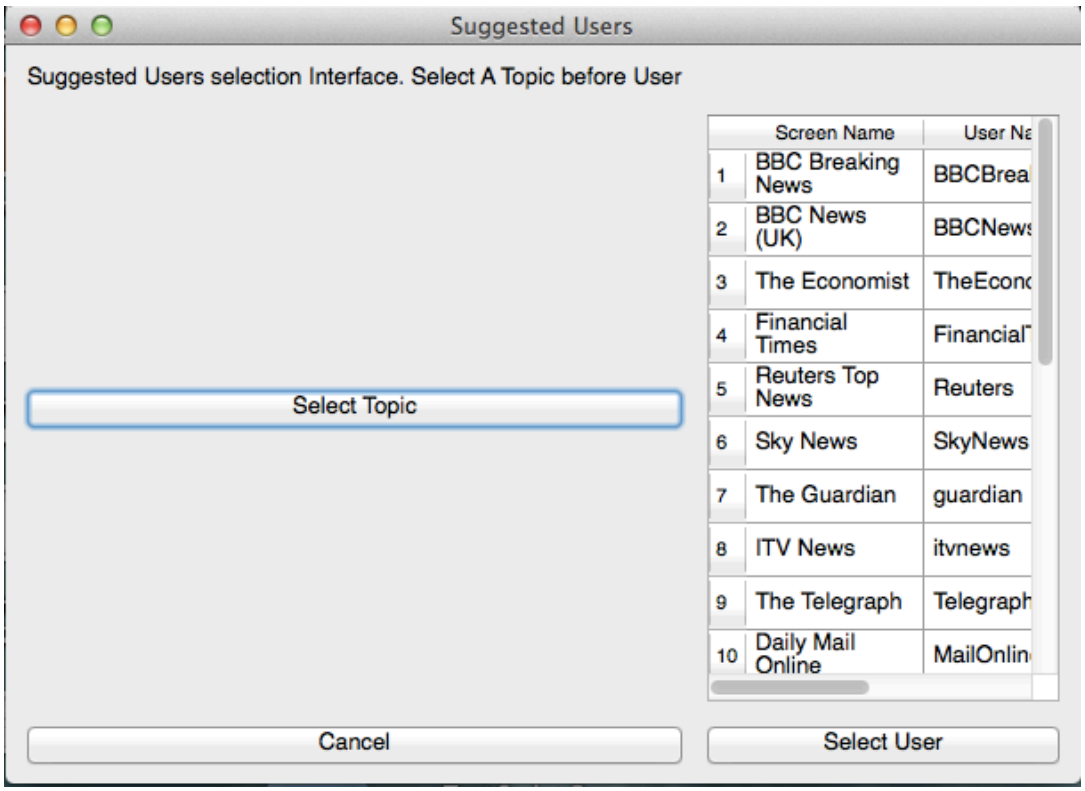
This is the Bookmark Deletion interface, which allows the user to choose a bookmark to delete from the database.

5.1.5 – The Modification Interface



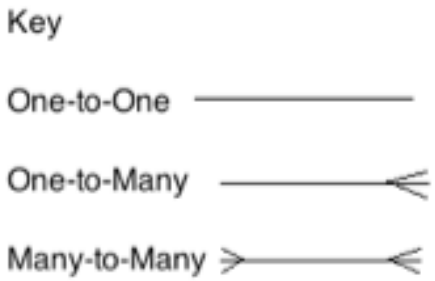
The Modify Bookmark Interface which allows the user to modify an already existing bookmark in the database.

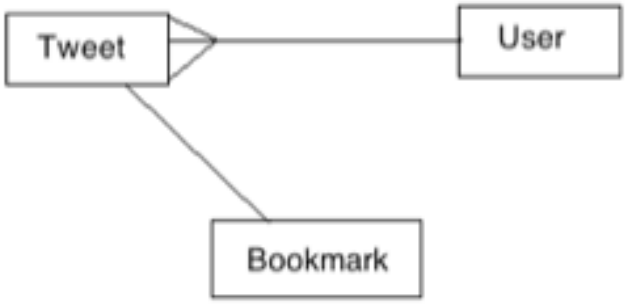
5.1.6 – The Suggested User Interface



The Suggested User Interface which allows the user to be able to add a suggested user to the database of the system.

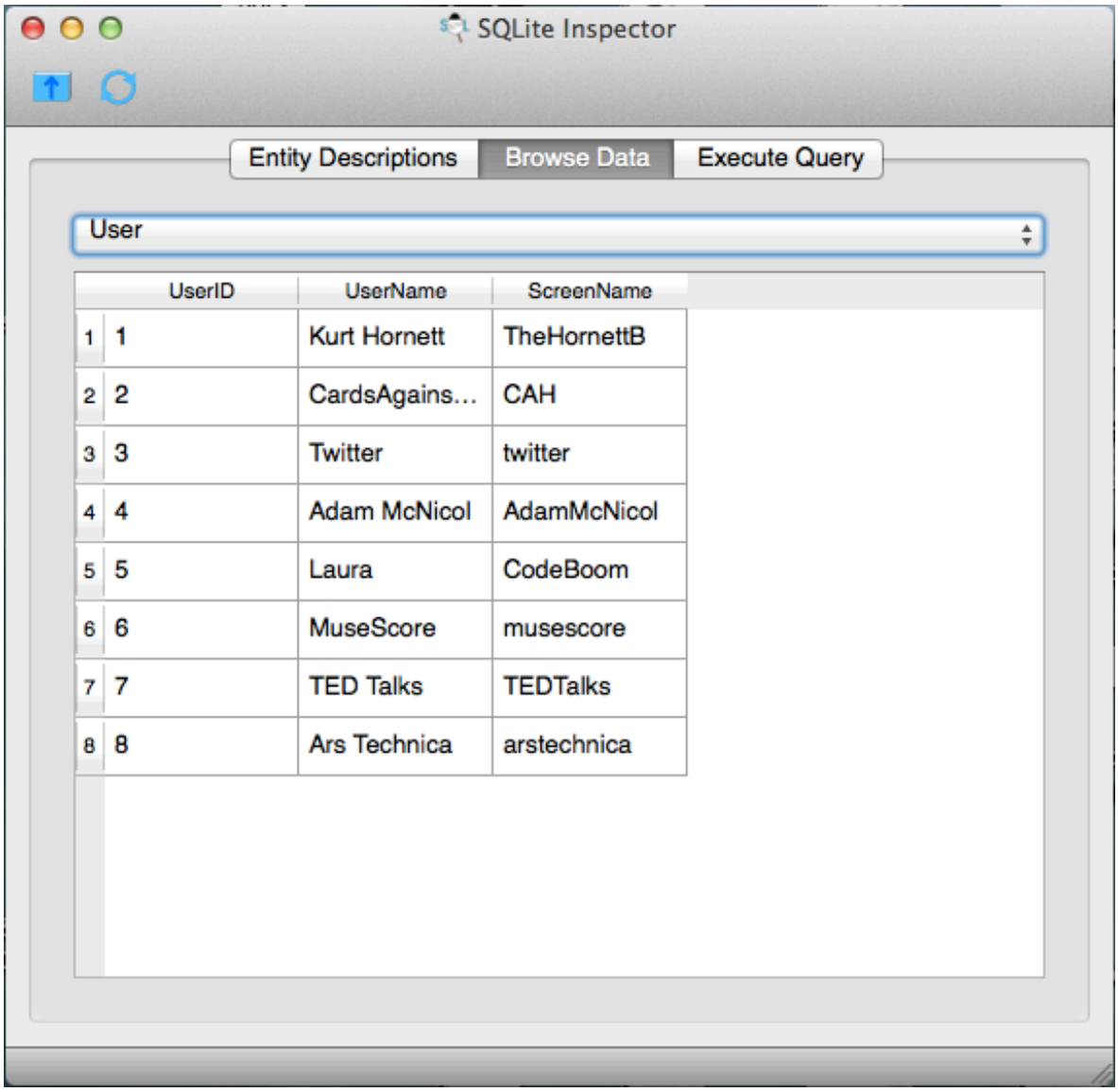
5.2 ER Diagram





5.3 Database Table Views

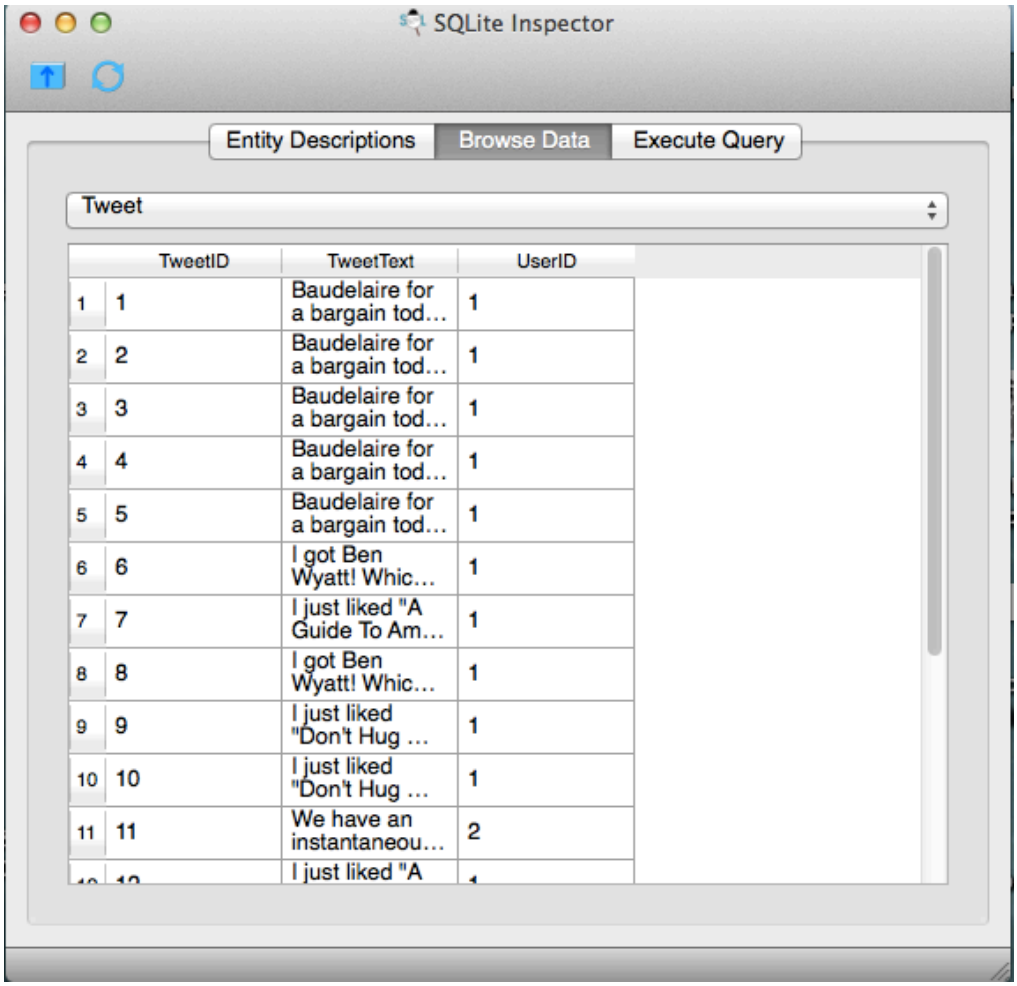
5.3.1 'User' Entity



	UserID	UserName	ScreenName
1	1	Kurt Hornett	TheHornettB
2	2	CardsAgains…	CAH
3	3	Twitter	twitter
4	4	Adam McNicol	AdamMcNicol
5	5	Laura	CodeBoom
6	6	MuseScore	musescore
7	7	TED Talks	TEDTalks
8	8	Ars Technica	arstechnica

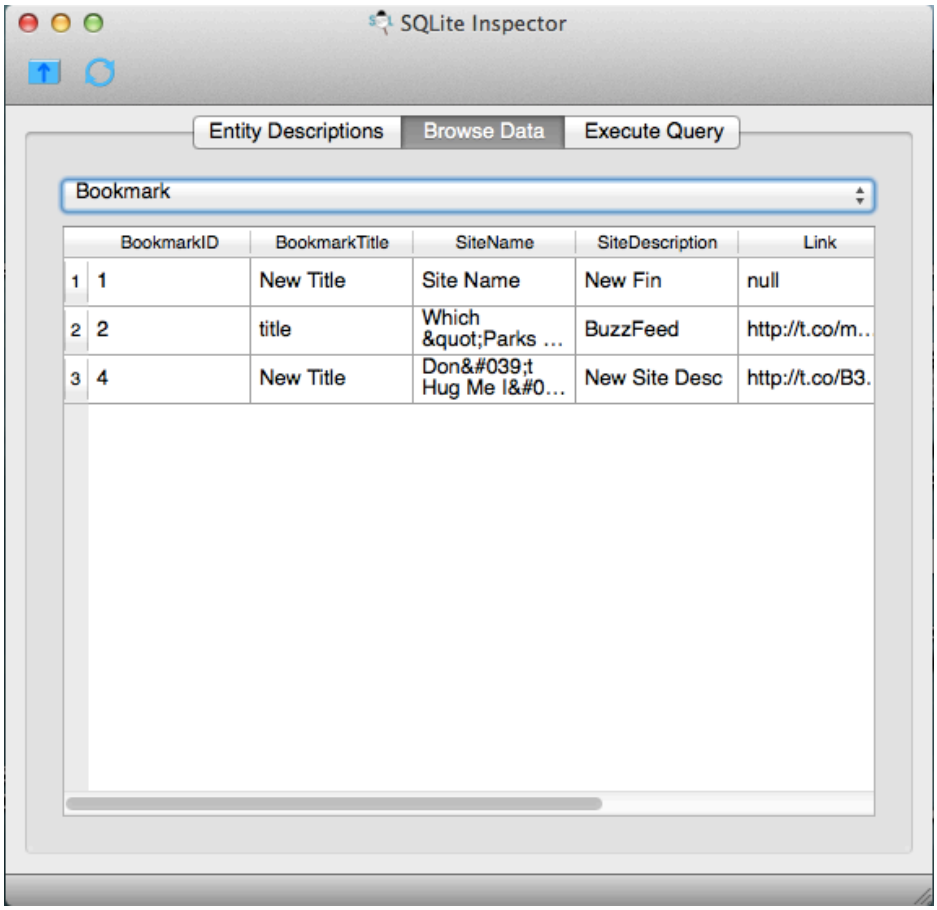
The 'User' entity of the database; it has 3 fields, the UserID used to identify each user, the UserName field which stores the username of the added user and the ScreenName field which stores the screen name of the added user.

5.3.2 'Tweet' Entity



This is the Tweet entity, it also has three fields; the TweetID field uniquely identifies each tweet, the TweetText field stores the text of each tweet and the UserID is a foreign key that links the Tweet to the User.

5.3.3 'Bookmark' Entity



This shows the Bookmark entity of the database, it has five fields; the BookmarkID is the primary key, which identifies each bookmark, the BookmarkTitle field stores the title of each bookmark, the SiteDescription field stores the user defined site description, the Link field stores the link pulled from the tweet and the TweetID field is a foreign key, which stores the primary of the Tweet each bookmark is linked to.

5.4 Database SQL

5.4.1 User Table

```
CREATE TABLE User
        (UserID INTEGER,
        UserName TEXT,
        ScreenName TEXT,
        PRIMARY KEY(UserID))
```

5.4.2 Tweet Table

```
CREATE TABLE Tweet
        (TweetID INTEGER,
        TweetText TEXT,
        UserID INTEGER,
        PRIMARY KEY(TweetID),
        FOREIGN KEY(UserID) REFERENCES User(UserID) ON UPDATE CASCADE ON
DELETE CASCADE)
```

5.4.3 Bookmark Table

```
CREATE TABLE Bookmark
        (BookmarkID INTEGER,
        BookmarkTitle TEXT,
        SiteName TEXT,
        SiteDescription TEXT,
        Link TEXT,
        TweetID INTEGER,
        PRIMARY KEY(BookmarkID),
        FOREIGN KEY(TweetID) REFERENCES Tweet(TweetID) ON UPDATE CASCADE ON
DELETE CASCADE)
```

5.5 SQL Queries

SQL Query	Functionality
SELECT Username, ScreenName, UserID From User	This Query is used to retrieve all the information about the users from the database.
INSERT INTO User (UserName, ScreenName) VALUES(?,?)	This Query is used to add a user to the database, the ?s represent unknown values that are handled by the system.
INSERT INTO Tweet (TweetText, UserID) VALUES(?,?)	This Query is used to add a tweet to the database, the ?s representing unknown values.
INSERT INTO Bookmark (BookmarkTitle,	This Query is used to add a new bookmark into the

SiteName, SiteDescription, Link, TweetID) VALUES(?,?,?,?,?)	database, the ?s represent unknown variables.
SELECT TweetID FROM Tweet WHERE TweetID = (SELECT MAX(TweetID) FROM Tweet)	This query is used to get the max TweetID from the Tweet Table in order to know how many tweets are in the table.
SELECT Tweet.TweetText, Tweet.UserID, Tweet.TweetID, Bookmark.Link, User.Username, Bookmark.BookmarkID FROM Tweet,User,Bookmark WHERE Bookmark.TweetID = Tweet.TweetID AND Tweet.UserID = User.UserID	This query is used to return all the bookmark in the database.
DELETE FROM Bookmark WHERE BookmarkID = ?	This query is used to delete a bookmark from the database, the ?s are unknown values.
SELECT Username FROM User,Tweet WHERE Tweet.TweetID = ? AND Tweet.UserID = User.UserID	This query is used to get a single user from the database, the ?s are unknown values.
UPDATE Bookmark SET BookmarkTitle = ? WHERE BookmarkID = ? UPDATE Bookmark SET SiteName = ? WHERE BookmarkID = ? UPDATE Bookmark SET SiteDescription = ? WHERE BookmarkID = ? UPDATE Bookmark SET Link = ? WHERE BookmarkID = ?	These queries are all used to update bookmarks already existent in the database, the ?s represent unknown variables.

All the SQL statements in this table use a Q Mark interface to input unknown values into the database, the ?s in the text are used as placeholders for unknown value that have information inputted into them. This is for security as using invariable tuples limits the

chances of erroneous data being entered into the database, rather than using python's inbuilt syntax.

6. Testing

6.1 Summary of Testing

The Full testing results can be found in the testing section, pp62-99

Overall the result of the testing was very good and there are no known errors in the system that are not caused by the user entering erroneous data. The testing section has proved that all the key functions of both the CLI and GUI interfaces are in working condition that are easy for the user to understand.

The majority of the key objectives of the client have been met with a few minor shortcomings in the system. Firstly the amount of tweet a user is able to view at one time is 10, however this can easily be changed with a small change in the loops in the system. Furthermore the Trends section of the program has completely been cut due to timing constraints. Apart from these few exceptions the program completes most of the other requirements of the client.

6.2 Known Issues

Currently there are no known major issues in the program. Most issues of great importance were fixed in the early testing and implementation stages of the development of the system, and as such the system is largely robust and secure.

Despite this there are some issues regarding *how* a user reaches a new interface and what functions are run when this is done. For example when the user adds a suggested user then uses the cancel button to return to the initial interface the new user is not found in the table view; however if the user uses the menu bar of the program the necessary functions are run and the user is displayed. These errors, although, are not common in the program with this being the only currently identified instance.

7. Code Explanations

7.1 Difficult Sections

There are a few more difficult sections within this code that may do with some explanation in order to clarify what exactly they are doing.

7.1.1 Displaying Tweets in the GUI (§10.4, Line 86-104, pp142-146)

```
86.    def displayTweet(self):
```

```
87.         self.timeline =
self.twitter.statuses.user_timeline(id=self.userList[self.userChoice][1])
88.         self.tweetLayout.addWidget(self.returnButton,2,0)
89.         self.tweetTableWidget = QTableWidget(10,2)
90.         tweetListLabels = ['Username','Tweet Text']
91.         self.tweetTableWidget.setHorizontalHeaderLabels(tweetListLabels)
92.         tweetTextList = []
93.         count = 0
94.         self.tweetTableWidget.setSortingEnabled(False)
95.         while count < 10:
96.             newTableWidget = QTableWidgetItem(self.timeline[count]['text'])
97.             self.tweetTableWidget.setItem(count,1,newTableWidget)
98.             userTableWidget = QTableWidgetItem(self.userList[self.userChoice][1])
99.             self.tweetTableWidget.setItem(count,0,userTableWidget)
100.            count += 1
101.        self.tweetSelectMenuFunc()
102.        self.tweetLayout.addWidget(self.selectTweetButton,2,2)
103.        self.tweetLayout.addWidget(self.tweetTableWidget,1,0,1,3)
104.        self.mainLayout.setCurrentWidget(self.tweetWidget)
105.
```

The difficulty in this part of code comes from the use of a loop to help create a table widget that displays the tweets correctly in the interface. Lines 86-89 focus on instantiating and gathering the information needed to add to the table. After this the list labels are created, and are set as the titles for the columns in line 91. Starting from line 95 there is a while loop that adds the necessary information for the table by each time until the stepper variable has reached ten. Line 96 has the creation of a new temporary item for the table widget (hence there not being a 'self.' At the beginning of the variable name) that contains the tweet text; the next step is to add that to the table, in the second column ('1' is used as '0' is the first column), with 'count being the row number. This is repeated in lines 98-9 for usernames rather than tweets. Line 100 is iterating the stepper so that there is not an infinite loop. Lines 101-4 are adding the newly created layout to the stacked main layout and then setting that as the current widget.

7.1.2 Regular Expressions (§10.4, Lines 138-42; §10.2, Lines 102-7)

```
138.        regex = re.compile('<title>(.*?)</title>')
139.        url = urllib.request.urlopen(link) #Gets the data from the url
140.        html = url.read() #Converts HTTPRequest into text
141.        html = str(html) #Changes from 'bytes' to str so re will work
142.        self.siteName = regex.search(html).group(1)
```

This code is from §10.4 'TweetInterface.py' (It also exists in the CLI version at §10.2 'twitter_database.py', Lines 102-7) which demonstrates the use of a regular expression to get the title of webpage for the bookmark information. Line 138 shows the compilation of the regular expression, using the re module, which from the regular expression itself it is evident that it shall be used to get the title of a webpage from the use of HTML tags (specifically the title tags that are used to store the websites name). Line 139 shows that use of the urllib.request module to get an html file from the internet, the 'link' value would be from the tweet that the user is trying to bookmark, if one exists, as it is stored in the meta-information about a tweet it does not require a regular expression to parse it from

the tweet text. Line 140 converts this into a text file that is readable, while line 141 demonstrates a conversion from the bytes format to a string one, which re will work with. Line 142 shows the setting on the variable 'self.siteName' with the result of the re search of the text file, which shall be the title of the webpage – if he page doesn't have a title then the space is left blank.

7.1.3 Using OAuth and oauth_dance (§10.3 Lines, 107-16)

```
107.        consumer_key = getConsumerKey()
108.        MY_TWITTER_CREDS = os.path.expanduser('~/.login_credentials')
109.        if not os.path.exists(MY_TWITTER_CREDS):
110.            oauth_dance("KurtsApp",'Y4zkic6lw0Hu6uB3sNVH4Q', consumer_key,
111.                        MY_TWITTER_CREDS)
112.        oauth_token, oauth_secret = read_token_file(MY_TWITTER_CREDS)
113.        #Creation of twitter object with authorized detailes.
114.        self.twitter = Twitter(auth=OAuth(
115.            oauth_token, oauth_secret, 'Y4zkic6lw0Hu6uB3sNVH4Q', consumer_key))
```

What this part of the code shows is the interaction between the system and the Twitter API, specifically for the authorization of the application in order to carry out any of its functions. The first line of this code shows the consumer_key being obtained from a function, this is in face the API secret key that mustn't be human readable in the program and so has been converted to binary data and stored in a hidden file. The second line is used to create the directory, also a hidden file, where the login credentials of the user shall be stored, these are special OAuth tokens that are obtained from Twitter and bear no resemblance to the username and password and are never readable in the application due to the oauth_dance tool that is provided by Python Twitter Tools, which is used in the next couple of lines in order to obtain these items. The last of the lines in this part of the code is the instantiation of the Twitter class which is used to make any interaction with the Twitter API throughout the application, using the keys that have just been generated.

7.2 Self-created Algorithms

7.2.1 Testing whether a user is in the database already

(From §10.1 'twitter_cli.py', Lines238-245)

```
238.            for users in users:
239.                if users[1] == SuggUsers[SuggUserChoice-1]['screen_name']:
240.                    check = True
241.            if check == False:
242.                addUserFromSearch(userList)
243.                print('User Added')
244.            else:
245.                print('User already exits, not added')
```

What this algorithm does is determine whether a user already exists in the database to reduce the chance of duplication in the database to 0. It does this by checking the value 'user' against the relevant information in the list 'users', taken from the database. If it is the

case then the value ‘check’ is set to true ready for the rest of the function to do what is required to add the user, or not, to the database.

8. Settings

In order to run this program a few modules are needed. Firstly the standard Python 3.3 installation is needed, on top of the PyQt 4 is required if the client wishes to run the GUI version of the system. RE and Urllib modules are included in the Python 3.3 installation and so do not need to be installed separately from the main Python installation.

In order to be able to interact with the Twitter API and perform all the useful functions of the system, it is necessary for the user to install the twitter-1.13.1 module. Information on how to install this module can be found at https://pypi.python.org/pypi/twitter/1.13.1.

9. Acknowledgements

This system contains no code taken directly from an Internet source that is not a part of the documentation of Python, PyQt4, or Python Twitter Tool. This was in order to successfully implement the user of regular expressions, using urllib, and utilizing Oauth to connect and use the Twitter API.

10. Code Listing Appendix

10.1 twitter_cli.py

```
1. #Kurt Hornett
2. #COMP4 - Implemetation
3. #Twitter Implemetation work - MarkII
4. #Dec 2013
5.
6. #External Modules
7. from twitter import *
8. from sqlite3 import *
9.
10. #User Made Maodules
11. from twitter_database import *
12.
13. #System Modules
14. import os
15. import pickle
16.
17. #Creation of a CLI Inteface for interacting with twitter.
18. #Limiting to viewing 10 latest tweets from timeline for testing purposes.
19.
20. def mainMenu():
21.     #Typical main menu
22.     print()
23.     print('Twitter Management Program')
24.     print()
25.     print(' 1. Get a Users Time Line')
26.     print(' 2. Get Home Timeline')
27.     print(' 3. Print 10 latest Tweets')
28.     print(' 4. Add User to database')
29.     print(' 5. Print tweets from User in Database')
30.     print(' 6. Show Bookmark database')
31.     print(' 7. Display Single Bookmark')
32.     print(' 8. Delete bookmark from database')
33.     print(' 9. Modify bookmark')
34.     print('10. Display Suggested Users')
35.     print(' 0. Exit Application')
36.     print()
37.     print('Use \'0\' to exit to here anywhere in the program')
38.     print()
39.
```

```
40. def HomeUserMenu():
41.     print()
42.     print('Please select Home Timeline or User Timeline')
43.     print()
44.     print('1. Home Timeline')
45.     print('2. User Timeline')
46.     print()
47. 
48. def getChoice():
49.     try:
50.         choice = int(input('Select number: '))
51.     except ValueError:
52.         print('Please enter an integer.')
53.         choice = int(input('Select number: '))
54.     return choice
55. 
56. def getConsumerKey():
57.     #Get Consumer key from hidden file - API advises it's kept secret
58.     with open('.consumer_key.txt','rb') as key:
59.         consumer_key = pickle.load(key)
60.     return consumer_key
61. 
62. def authApp(consumer_key):
63.     #All necessary proceedures for authorising app and therefore loggin into twitter, doesn't allow log out.
64.     #Uses PIN authorization  - could change to UserPass later in app stages and implemetation
65.     #MY_TWITTER_CREDS = os.path.expanduser('~/.login_credentials')
66.     if not os.path.exists(MY_TWITTER_CREDS):
67.         oauth_dance("KurtsApp",'Y4zkic6lw0Hu6uB3sNVH4Q', consumer_key,
68.                         MY_TWITTER_CREDS)
69. 
70. 
71. def createTwitterObject():
72.     oauth_token, oauth_secret = read_token_file(MY_TWITTER_CREDS)
73.     #Creation of twitter object with authorized detailes.
74.     twitter = Twitter(auth=OAuth(
75.         oauth_token, oauth_secret, 'Y4zkic6lw0Hu6uB3sNVH4Q', consumer_key))
76.     return twitter
77. 
78. def getHomeTimeline(twitter):
79.     #Creates home_timeline object with all available tweets from the home timeline.
80.     home_timeline = twitter.statuses.home_timeline()
81.     return home_timeline
82. 
83. def displayLatestTweets(hm,count):
84.     #Print information of 10 latest tweets as a table
```

```
    print('{0:<3}{1:<15}{2:<140}{3:<21}'.format('No.','Screen Name','Tweet Text','Links'))
    while count < 10:
        #Checks to see that the tweets have links; if there are no links then the print statement crashes the
        #programme.
        if hm[count]['entities']['urls'] != []:
            print('{0:<3}{1:<15}{2:<140}{3:<21}'.format(count+1,hm[count]['user']['screen_name'],
                                                        hm[count]['text'],
                                                        hm[count]['entities']['urls'][0]['url']))
        else:
            print('{0:<3}{1:<15}{2:<140}{3:<21}'.format(count+1,hm[count]['user']['screen_name'],
                                                        hm[count]['text'],
                                                        'No Link'))

        count += 1

    def getUserTimeLine():
        try:
            timeline = twitter.statuses.user_timeline(id=input('Please enter screen name: ')) #Gets a users
timeline from the API
            return timeline
        except:
            print('An error occured')

    def displayLinks(hm,count):
        #Prints links as a table
        print('{0:<3} {1:<21}'.format('No.','Link'))
        while count < 10:
            if hm[count]['entities']['urls'] != []:
                print('{0:<3} {1:<21}'.format(count+1,hm[count]['entities']['urls'][0]['url']))
            else:
                 print('{0:<3} {1:<21}'.format(count+1,'No Link'))
            count += 1

    def displayUsers(users):
        count = 1
        print()
        print('Users in database')
        print()
        #Prints the users in the list
        for user in users:
                print('{0:<3}{1:<21}'.format(count,user[0]))
                count += 1
        print()

    def displayUsersTimeline(choice,users,twitter):
```

```
129.        try:
130.            userTm = twitter.statuses.user_timeline(id=users[choice-1][1])
131.            count = 0
132.            displayLatestTweets(userTm,count)
133.            return userTm
134.        except:
135.            print('An error has occured.')
136.
137.
138.    def getYN():
139.        try:
140.            YN = input('Commit Action (Y/N): ')
141.            while YN not in ['y','n','Y','N']:
142.                YN = input('Input Valid Option: ')
143.            YN = YN.upper()
144.            return YN
145.        except:
146.            print('An error occured')
147.
148.    def displayBookmarks(List):
149.        print('{0:<3}{1:<3}{2:<140}{3:<25}{4:<21}'.format(
150.            'N.','ID','Tweet Text','Link','Username'))
151.        count = 0
152.        while count < len(List):
153.            print('{0:<3}{1:<3}{2:<140}{3:<25}{4:<25}'.format(
154.                count+1,List[count][5],List[count][0],List[count][3],List[count][4]))
155.            count += 1
156.
157.    def createIDlist(List):
158.        IDlist = []
159.        for count in range(len(List)):
160.            IDlist.append(List[count][5])
161.        return IDlist
162.
163.    def DisplaySingleBookmark(List):
164.        print()
165.        print('Bookmark Information')
166.        print()
167.        print('Title: {0}'.format(List[0][0]))
168.        print('Site Name: {0}'.format(List[0][1]))
169.        print('Description: {0}'.format(List[0][2]))
170.        print('Link: {0}'.format(List[0][3]))
171.        print()
172.
173.    def displayModifyChoices():
```

```
174.        print()
175.        print('What to modify?')
176.        print()
177.        print('1. Bookmark Title')
178.        print('2. Site Name')
179.        print('3. Site Description')
180.        print('4. Link')
181.        print()
182.
183.    def getChanges():
184.        changes = input('Please enter new data: ')
185.        while changes == '':
186.            changes = input('Some data must be entered: ')
187.        return changes
188.
189.    def displaySlugs():
190.        suggUserSlugs = ['News','Technology','Business']
191.        count = 0
192.        print()
193.        print('Please select the Catergry to get Suggested Users From.')
194.        print()
195.        while count < len(suggUserSlugs):
196.            print('{0}. {1}'.format(count+1,suggUserSlugs[count]))
197.            count += 1
198.        print()
199.
200.    def displaySuggestedUsers(suggUsers,twitter):
201.        count = 0
202.        print()
203.        print('Here are the selected users')
204.        print()
205.        while count < len(suggUsers):
206.            print('{0}. {1}'.format(count+1,suggUsers[count]['name']))
207.            count += 1
208.        print()
209.        SuggUserChoice = getChoice()
210.        if SuggUserChoice != 0:
211.            FAChoice = chooseFollowAdd()
212.            if FAChoice != 0:
213.                FollowAddUser(FAChoice,SuggUserChoice,suggUsers,twitter)
214.
215.    def FollowAddUser(FAChoice,SuggUserChoice,SuggUsers,twitter):
216.        if FAChoice == 1:
217.            twitter.friendships.create(id=SuggUsers[SuggUserChoice-1]['screen_name'])
218.            print('User Followed')
```

```
219.        elif FAChoice == 2:
220.            userList = []
221.            userList.append(SuggUsers[SuggUserChoice-1])
222.            users = getUsersFromDatabase()
223.            check=False
224.            for users in users:
225.                if users[1] == SuggUsers[SuggUserChoice-1]['screen_name']:
226.                    check = True
227.            if check == False:
228.                addUserFromSearch(userList)
229.                print('User Added')
230.            else:
231.                print('User already exits, not added')
232.        elif FAChoice == 3:
233.            twitter.friendships.create(id=SuggUsers[SuggUserChoice-1]['screen_name'])
234.            userList = []
235.            userList.append(SuggUsers[SuggUserChoice-1])
236.            users = getUsersFromDatabase()
237.            check=False
238.            for users in users:
239.                if users[1] == SuggUsers[SuggUserChoice-1]['screen_name']:
240.                        check = True
241.                if check == False:
242.                    addUserFromSearch(userList)
243.                else:
244.                    print('User already exits, not added')
245.            print('User Added & Followed')
246.        else:
247.            print('Nothing Happened')
248.
249.    def displayForSuggUsers(twitter,choice):
250.        if choice == 1:
251.            suggUsers = twitter.users.suggestions.news.members()
252.            displaySuggestedUsers(suggUsers,twitter)
253.        elif choice == 2:
254.            suggUsers = twitter.users.suggestions.technology.members()
255.            displaySuggestedUsers(suggUsers,twitter)
256.        elif choice == 3:
257.            suggUsers = twitter.users.suggestion.business.members()
258.            displaySuggestedUsers(suggUsers,twitter)
259.        else:
260.            print('Invalid Choice')
261.
262.    def chooseFollowAdd():
263.        print()
```

```
264.        print('Would you like to follow or add a user?')
265.        print()
266.        print('1. Follow')
267.        print('2. Add to Database')
268.        print('3. Both')
269.        print()
270.        choice = getChoice()
271.        return choice
272.
273.    def checkingStuff(twitter):
274.        suggUsers = twitter.users.suggestions.news.members()
275.        count = 0
276.        while count < len(suggUsers):
277.            print(suggUsers[count]['screen_name'])
278.            count += 1
279.        #print(suggUsers)
280.
281.    if __name__ == "__main__":
282.        #Consumer key required regardless
283.        consumer_key = getConsumerKey()
284.        MY_TWITTER_CREDS = os.path.expanduser('~/.login_credentials')
285.        authApp(consumer_key)
286.        twitter = createTwitterObject()
287.        choice = 1
288.        count = 0
289.        #Typical while loop for managing program.
290.        #0 being catch-out for exiting program.
291.        while choice != 0:
292.            mainMenu()
293.            choice = getChoice()
294.            while choice not in [0,1,2,3,4,5,6,7,8,9,10,99]:
295.                choice = getChoice()
296.            if choice == 1:
297.                userTm = getUserTimeLine()
298.            elif choice == 2:
299.                home_timeline = getHomeTimeline(twitter)
300.                hm = home_timeline
301.            elif choice == 3:
302.                HomeUserMenu()
303.                HoMo = getChoice()
304.                if HoMo == 1:
305.                    displayLatestTweets(home_timeline,count)
306.                else:
307.                    displayLatestTweets(userTm,count)
308.            elif choice == 4:
```

```
                databaseMenu()
                choice1 = getChoice()
                while choice1 != 0:
                    if choice1 == 1:
                        try:
                            user = getUser(twitter)
                            users = getUsersFromDatabase()
                            check=False
                            for users in users:
                                if users[1] == user[0]['screen_name']:
                                    check = True
                            if check == False:
                                addUserFromSearch(user)
                            else:
                                print('User already exits, not added')
                        except:
                            print('An error occur\'d')
                        choice1 = 0
            elif choice == 5:
                users = getUsersFromDatabase()
                displayUsers(users)
                Uchoice = getChoice()
                while Uchoice != 0:
                    userTm = displayUsersTimeline(Uchoice,users,twitter)
                    YN = getYN()
                    if YN == 'Y':
                        Tnumber = getChoice()
                        addTweet(users,userTm,Tnumber,Uchoice)
                        addBookmark(userTm,Tnumber)
                    elif YN == 'N':
                        Uchoice  = 0
            elif choice == 6:
                List = getBookmarks()
                displayBookmarks(List)
            elif choice == 7:
                Bookmarks = getBookmarks()
                displayBookmarks(Bookmarks)
                BookChoice = getChoice()
                while BookChoice > len(Bookmarks):
                    print('Please choose integer in list: ')
                    BookChoice = getChoice()
                SingleBookmark = getDataBookmark(BookChoice)
                DisplaySingleBookmark(SingleBookmark)
            elif choice == 8:
                List = getBookmarks()
```

```
354.                displayBookmarks(List)
355.                print()
356.                print('Please Choose a Bookmark ID to delete, 0 to exit: ')
357.                print()
358.                Dchoice = getChoice()
359.                if Dchoice != 0:
360.                    BList = getDataBookmark(Dchoice)
361.                    DisplaySingleBookmark(BList)
362.                    DlYN = getYN()
363.                    if DlYN == 'Y':
364.                        deleteBookamrk(Dchoice)
365.                    else:
366.                        print('Bookmark not deleted')
367.                else:
368.                    pass
369.            elif choice == 9:
370.                Bookmarks = getBookmarks()
371.                displayBookmarks(Bookmarks)
372.                Mchoice = getChoice()
373.                IDlist = createIDlist(Bookmarks)
374.                while Mchoice not in IDlist:
375.                    Mchoice = getChoice()
376.                if Mchoice != 0:
377.                    SingleBookmark = getDataBookmark(Mchoice)
378.                    DisplaySingleBookmark(SingleBookmark)
379.                    displayModifyChoices()
380.                    ModChoice = getChoice()
381.                    while ModChoice not in [0,1,2,3,4]:
382.                        ModChoice = getChoice()
383.                    if ModChoice != 0:
384.                        changes = getChanges()
385.                        ModifyBookmark(Mchoice, ModChoice, changes)
386.            elif choice == 10:
387.                displaySlugs()
388.                SuggChoice = getChoice()
389.                displayForSuggUsers(twitter,SuggChoice)
390.            elif choice == 99:
391.                checkingStuff(twitter)
392.            else:
393.                pass
```

10.2 twitter_database.py

```
1.  #Kurt Hornett
2.  #COMP4 - Implemetation
3.  #Twitter CLI - Database management
4.  #Dec 2013
5.
6.  from twitter import *
7.
8.  import re
9.  import urllib.request
10.
11. from twitter_cli import *
12.
13. import sqlite3
14.
15. def databaseMenu():
16.     print()
17.     print('Database Management Aspect')
18.     print()
19.     print('1. Add User to database')
20.     print()
21.
22. def query(sql,data,db):
23.     with sqlite3.connect(db) as db:
24.         cursor = db.cursor()
25.         cursor.execute('PRAGMA foreign_keys = ON')
26.         cursor.execute(sql,data)
27.         db.commit()
28.
29. def searchQuery(sql,data,db):
30.     with sqlite3.connect(db) as db:
31.         cursor = db.cursor()
32.         cursor.execute('PRAGMA foreign_keys = ON')
33.         cursor.execute(sql,data)
34.         results = cursor.fetchall()
35.         return results
36.
37. def getUser(twitter):
38.     try:
39.         search = input('Please enter handle to search for: ')
40.         userList = twitter.users.lookup(screen_name=search)
```

```
41.         return userList
42.     except:
43.         print('An error has occcured')
44.
45. def getUsersFromDatabase():
46.     sql = '''SELECT Username,ScreenName,UserID
47.                 From User'''
48.     db = 'Bookmark_Database-cli-test-2.db'
49.     users = searchQuery(sql,(),db)
50.     return users
51.
52. def addUserFromSearch(userList):
53.     sql = '''INSERT INTO User(UserName,ScreenName)
54.                 VALUES(?,?)'''
55.     data = (userList[0]['name'],userList[0]['screen_name'])
56.     db = 'Bookmark_Database-cli-test-2.db'
57.     query(sql,data,db)
58.
59. def addUserFromTweet(hm,number):
60.     sql = '''INSERT INTO User(UserName,ScreenName)
61.                 Values(?,?)'''
62.     data = (hm[number]['user']['name'],hm[number]['user']['screen_name'])
63.     db = 'Bookmark_Database-cli-test-2.db'
64.     query(sql,data,db)
65.
66. def addTweet(Users,UserTm,Tnumber,Uchoice):
67.     sql = '''INSERT INTO Tweet(TweetText,UserID) VALUES(?,?)'''
68.     data = (UserTm[Tnumber-1]['text'],Users[Uchoice-1][2])
69.     db = 'Bookmark_Database-cli-test-2.db'
70.     query(sql,data,db)
71.
72. def addBookmark(hm,number):
73.     sql = '''INSERT INTO Bookmark(BookmarkTitle,SiteName,SiteDescription,Link,TweetID)
74.                 VALUES(?,?,?,?,?)'''
75.     tweetID = getLatestTweet()
76.     if hm[number-1]['entities']['urls'] != []:
77.         link = hm[number-1]['entities']['urls'][0]['url']
78.         bookmarkTitle,siteName,siteDesc = getBookmarkData(link)
79.     else:
80.         link = 'No Link'
81.         bookmarkTitle,siteName,siteDesc = getBookmarkData(link)
82.     data = (bookmarkTitle,siteName,siteDesc,link,tweetID[0][0])
83.     db = 'Bookmark_Database-cli-test-2.db'
84.     if checkAdd(data,tweetID):
85.         query(sql,data,db)
```

```
    else:
        pass

def getLatestTweet():
    sql = '''SELECT TweetID FROM Tweet WHERE TweetID = (SELECT MAX(TweetID) FROM Tweet)'''
    db = 'Bookmark_Database-cli-test-2.db'
    maxId = searchQuery(sql,(),db)
    return maxId

def getBookmarkData(link):
    bookmarkTitle = input('Please enter a title for the bookmark: ')
    siteName = 'Null'
    siteDesc = 'Null'
    if link != 'No Link':
                try:
                    #Create RE for finding title
                    regex = re.compile('<title>(.*?)</title>')
                    url = urllib.request.urlopen(link) #Gets the data from the url
                    html = url.read() #Converts HTTPRequest into text
                    html = str(html) #Changes from 'bytes' to str so re will work
                    siteName = regex.search(html).group(1)
                    siteDesc = input('Please enter a short description of the site: ')
                except:
                    print()
                    print('No HTML Title Found')
                    print()
            return bookmarkTitle,siteName,siteDesc

        def getBookmarks():
        ##    sql = '''SELECT TweetText,UserID, TweetID FROM Tweet'''
        ##    sql2 = '''SELECT Link FROM Bookmark WHERE TweetID = ?'''
        ##    sql3 = '''SELECT Username FROM User WHERE UserID = ?'''
            sql = '''SELECT
    Tweet.TweetText,Tweet.UserID,Tweet.TweetID,Bookmark.Link,User.Username,Bookmark.BookmarkID FROM
                     Tweet,User,Bookmark WHERE Bookmark.TweetID = Tweet.TweetID AND Tweet.UserID = User.UserID'''
            db = 'Bookmark_Database-cli-test-2.db'
            tweetList = searchQuery(sql,(),db)
            return tweetList

        def deleteBookamrk(Dchoice):
            sql = '''DELETE FROM Bookmark WHERE BookmarkID = ?'''
            db = 'Bookmark_Database-cli-test-2.db'
            query(sql,(Dchoice,),db)

        def checkAdd(data,tweetID):
```

```
130.          print()
131.          print('Are you sure you wish to add this bookamrk?')
132.          print()
133.          print('Title: {0}'.format(data[0]))
134.          print('Site Name: {0}'.format(data[1]))
135.          print('Site Description: {0}'.format(data[2]))
136.          print('Link: {0}'.format(data[3]))
137.          username = getIndieUser(tweetID)
138.          print('User: {0}'.format(username[0][0]))
139.          print()
140.          YNadd = getYN()
141.          print()
142.          if YNadd == 'Y':
143.              return True
144.          else:
145.              return False
146.
147.      def getIndieUser(tweetID):
148.          sql = '''SELECT Username FROM User,Tweet WHERE Tweet.TweetID = ? AND Tweet.UserID = User.UserID'''
149.          db = 'Bookmark_Database-cli-test-2.db'
150.          username = searchQuery(sql,(tweetID[0][0],),db)
151.          return username
152.
153.      def getDataBookmark(choice):
154.          sql = '''SELECT BookmarkTitle,SiteName,SiteDescription,Link FROM Bookmark
155.                   WHERE BookmarkID = ?'''
156.          data = (choice,)
157.          db = 'Bookmark_Database-cli-test-2.db'
158.          bookmarks = searchQuery(sql,data,db)
159.          return bookmarks
160.
161.      def ModifyBookmark(Mchoice,ModChoice,changes):
162.          if ModChoice == 1:
163.              sql = '''UPDATE Bookmark
164.                       SET BookmarkTitle = ?
165.                       WHERE BookmarkID = ?'''
166.          elif ModChoice == 2:
167.              sql = '''UPDATE Bookmark
168.                       SET SiteName = ?
169.                       WHERE BookmarkID = ?'''
170.          elif ModChoice == 3:
171.              sql = '''UPDATE Bookmark
172.                       SET SiteDescription = ?
173.                       WHERE BookmarkID = ?'''
174.          elif ModChoice == 4:
```

```
175.             sql = '''UPDATE Bookmark
176.                      SET Link = ?
177.                      WHERE BookmarkID = ?'''
178.         data = (changes,Mchoice)
179.         db = 'Bookmark_Database-cli-test-2.db'
180.         query(sql,data,db)
181.
182.     if __name__ == '__main__':
183.         user = getUser(twitter)
184.         addUserFromSearch(user)
185.
```

10.3 MainWindow.py

```
1.  # Kurt Hornett
2.  # Main Interface
3.  # Implemetation
4.
5.  from PyQt4.QtCore import *
6.  from PyQt4.QtGui import *
7.  from PyQt4.QtSql import *
8.
9.  from SearchWindow import *
10. from BookmarkTool import *
11. from TableView import *
12. from TweetInterface import *
13. from DeleteInterface import *
14. from SuggestedInterface import *
15.
16. #CLI Modules
17. from twitter_cli import *
18. from twitter_database import *
19.
20. import sys
21.
22. class MainWindow(QMainWindow):
23.     def __init__(self):
24.         super().__init__()
25.         #Set Window Title
26.         self.setWindowTitle('Twitter Program')
27.
28.         #Create Twitter Object
29.         self.createTwitterObject()
30.
31.         #Create Menubar for interface - Add Each Menu
32.         self.menuBar = QMenuBar()
33.         self.fileMenu = self.menuBar.addMenu('File')
34.         self.searchMenu = self.menuBar.addMenu('Search')
35.         self.managementMenu = self.menuBar.addMenu('Database Management')
36.         self.suggestedMenu = self.menuBar.addMenu('Suggested Users')
37.         self.trendsMenu = self.menuBar.addMenu('Display Trends')
38.
39.         #Add Actions to each Menu
40.         self.logOutAction = self.fileMenu.addAction('Log Out')
41.         self.quitAction = self.fileMenu.addAction('Quit')
42.         self.searchAction = self.searchMenu.addAction('New Search')
43.         self.tweetAction = self.searchMenu.addAction('Search Users Tweets')
```

```
44.         self.deleteAction = self.managementMenu.addAction('Delete Entries')
45.         self.modifyAction = self.managementMenu.addAction('Modity Databases')
46.         self.suggestedAction = self.suggestedMenu.addAction('Display Suggested Users')
47.         self.trendsAction = self.trendsMenu.addAction('Display Trends')
48.
49.         #Set Menu Bar
50.         self.setMenuBar(self.menuBar)
51.
52.         #Import Layouts
53.         self.searchInterface = SearchWindow(self.twitter)
54.         self.tableInterface = TableViewWindow()
55.         self.tweetSearchInterface = TweetInterface(self.twitter)
56.         self.deleteInterface = DeleteInterface()
57.         self.suggestedInterface = SuggestedInterface(self.twitter)
58.
59.         #Create Layout
60.         self.layout = QStackedLayout()
61.         self.layout.addWidget(self.searchInterface.searchWidget)
62.         self.layout.addWidget(self.tableInterface.tableWidget)
63.         self.layout.addWidget(self.tweetSearchInterface.tweetsWidget)
64.         self.layout.addWidget(self.deleteInterface.deleteWidget)
65.         self.layout.addWidget(self.suggestedInterface.suggUserWidget)
66.
67.         #Set Main Widget
68.         self.widget = QWidget()
69.         self.widget.setLayout(self.layout)
70.         self.setCentralWidget(self.widget)
71.
72.         #Add Action Actions
73.         self.searchAction.triggered.connect(self.newSearch)
74.         self.tweetAction.triggered.connect(self.tweetSearch)
75.         self.modifyAction.triggered.connect(self.modifyBookmarks)
76.         self.deleteAction.triggered.connect(self.deleteBookmarks)
77.         self.suggestedAction.triggered.connect(self.displaySuggested)
78.         self.quitAction.triggered.connect(self.quit_)
79.
80.         #Actions for sub-classes
81.         self.tableInterface.cancelButton.clicked.connect(self.cancelAction)
82.         self.deleteInterface.cancelButton.clicked.connect(self.cancelAction)
83.         self.suggestedInterface.cancelButton.clicked.connect(self.cancelAction)
84.
85.     def newSearch(self):
86.         self.layout.setCurrentWidget(self.searchInterface.searchWidget)
87.         self.setWindowTitle('Twitter Program')
88.     def tweetSearch(self):
```

```
89.         self.layout.setCurrentWidget(self.tweetSearchInterface.tweetsWidget)
90.         self.setWindowTitle('Tweet Search Interface')
91.     def cancelAction(self):
92.         self.layout.setCurrentWidget(self.searchInterface.searchWidget)
93.         self.setWindowTitle('New Search')
94.     def modifyBookmarks(self):
95.         self.tableInterface.createTableModel()
96.         self.layout.setCurrentWidget(self.tableInterface.tableWidget)
97.         self.setWindowTitle('Modify Database')
98.     def deleteBookmarks(self):
99.         self.deleteInterface.createTableModel()
100.            self.deleteInterface.createDeleteAction()
101.            self.layout.setCurrentWidget(self.deleteInterface.deleteWidget)
102.            self.setWindowTitle('Deletion Interface')
103.        def displaySuggested(self):
104.            self.layout.setCurrentWidget(self.suggestedInterface.suggUserWidget)
105.            self.setWindowTitle('Suggested Users')
106.        def createTwitterObject(self):
107.            consumer_key = getConsumerKey()
108.            MY_TWITTER_CREDS = os.path.expanduser('~/.login_credentials')
109.            if not os.path.exists(MY_TWITTER_CREDS):
110.                oauth_dance("KurtsApp",'Y4zkic6lw0Hu6uB3sNVH4Q', consumer_key,
111.                            MY_TWITTER_CREDS)
112.            oauth_token, oauth_secret = read_token_file(MY_TWITTER_CREDS)
113.            #Creation of twitter object with authorized detailes.
114.            self.twitter = Twitter(auth=OAuth(
115.                oauth_token, oauth_secret, 'Y4zkic6lw0Hu6uB3sNVH4Q', consumer_key))
116.        def quit_(self):
117.            window.close()
118.
119.
120.
121.    if __name__ == '__main__':
122.        app = QApplication(sys.argv)
123.        window = MainWindow()
124.        window.show()
125.        window.raise_()
126.        app.exec()
```

10.4 TweetInterface.py

```
1. #Kurt Hornett
2. #Show Tweet Tool
3. #Implemetation - Jan '14
4.
5. from PyQt4.QtCore import *
6. from PyQt4.QtGui import *
7. from PyQt4.QtSql import *
8.
9. from twitter_database import *
10. from sql_gui_misc import *
11. from BookmarkTool import *
12.
13. import sys
14. import re
15. import urllib.request
16.
17. class TweetInterface(QMainWindow):
18.     def __init__(self,twitter):
19.         super().__init__()
20.         #Set Window Titile
21.         self.setWindowTitle('Show User Tweets')
22.
23.         self.twitter = twitter
24.
25.         #Create Tweet Layout
26.         self.tweetLayout = QGridLayout()
27.         self.helpLabel = QLabel('Please select one of the users tweets.')
28.         self.returnButton = QPushButton('Return')
29.         self.tweetLayout.addWidget(self.helpLabel,0,0)
30.         self.selectTweetButton = QPushButton('Select Tweet')
31.
32.         #Create Menu
33.         self.usersButton = QPushButton('Users Menu')
34.         self.usersMenu = QMenu()
35.
36.         #Add Actions
37.         self.getUsersNumber()
38.         self.addActions()
39.         self.usersMenu.setTitle('Users Menu')
40.
41.         #Create User Layout
42.         self.userLayout = QGridLayout()
43.
```

```
        #Add Widgets
        self.userLayout.addWidget(self.usersButton,0,0)

        #Create Stacked Layout
        self.userWidget = QWidget()
        self.userWidget.setLayout(self.userLayout)
        self.tweetWidget = QWidget()
        self.tweetWidget.setLayout(self.tweetLayout)
        self.mainLayout = QStackedLayout()
        self.mainLayout.addWidget(self.userWidget)
        self.mainLayout.addWidget(self.tweetWidget)

        #Create central widger
        self.tweetsWidget = QWidget()
        self.tweetsWidget.setLayout(self.mainLayout)

        #Set Centarl Layout
        self.setCentralWidget(self.tweetsWidget)

        #Connexion
        self.usersButton.clicked.connect(self.showMenu)
        self.returnButton.clicked.connect(self.showUserMenu)
        self.selectTweetButton.clicked.connect(self.showTweetMenu)

    def showUserMenu(self):
        self.mainLayout.setCurrentWidget(self.userWidget)

    def showMenu(self):
        self.usersMenu.exec_(QCursor.pos())
        if self.sender().text() != 'User Menu':
            self.printUser()

    def showTweetMenu(self):
        self.tweetSelectMenu.exec_(QCursor.pos())

    def printUser(self):
        if self.sender().text() != self.usersButton.text():
            self.userChoice = 0
            while self.actionsList[self.userChoice].text() != self.sender().text():
                self.userChoice += 1
            self.displayTweet()

    def displayTweet(self):
        self.timeline = self.twitter.statuses.user_timeline(id=self.userList[self.userChoice][1])
        self.tweetLayout.addWidget(self.returnButton,2,0)
```

```
        self.tweetTableWidget = QTableWidget(10,2)
        tweetListLabels = ['Username','Tweet Text']
        self.tweetTableWidget.setHorizontalHeaderLabels(tweetListLabels)
        tweetTextList = []
        count = 0
        self.tweetTableWidget.setSortingEnabled(False)
        while count < 10:
            newTableWidget = QTableWidgetItem(self.timeline[count]['text'])
            self.tweetTableWidget.setItem(count,1,newTableWidget)
            userTableWidget = QTableWidgetItem(self.userList[self.userChoice][1])
            self.tweetTableWidget.setItem(count,0,userTableWidget)
                count += 1
            self.tweetSelectMenuFunc()
            self.tweetLayout.addWidget(self.selectTweetButton,2,2)
            self.tweetLayout.addWidget(self.tweetTableWidget,1,0,1,3)
            self.mainLayout.setCurrentWidget(self.tweetWidget)


    def tweetSelectMenuFunc(self):
        self.tweetSelectMenu = QMenu()
        for count in range(10):
            self.tweetSelectMenu.addAction('Tweet # {0}'.format(count+1)).triggered.connect(self.selectedTweet)
        self.tweetActionsList = self.tweetSelectMenu.actions()

    def getUsersNumber(self):
        self.userList = getUsersFromDatabaseGUI()
        self.userNumber = len(self.userList)

    def addActions(self):
        for count in range(self.userNumber):
            self.usersMenu.addAction('{0}'.format(self.userList[count][0])).triggered.connect(self.printUser)
        self.actionsList = self.usersMenu.actions()

    def selectedTweet(self):
        self.tweetChoice = 0
        while self.tweetActionsList[self.tweetChoice].text() != self.sender().text():
            self.tweetChoice += 1
        self.BookmarkToolWindow = BookmarkWindow(self.timeline,self.tweetChoice,self.userList,self.userChoice)
        if self.timeline[self.tweetChoice]['entities']['urls'] != []:
            self.getLinkInfo(self.timeline[self.tweetChoice]['entities']['urls'][0]['url'])
            bookmarkInfo = [self.timeline[self.tweetChoice]['entities']['urls'][0]['url'],self.siteName]
            self.BookmarkToolWindow.setData(bookmarkInfo)
            self.BookmarkToolWindow.show()
        else:
            bookmarkInfo = ['null','null']
```

```
134.              self.BookmarkToolWindow.setData(bookmarkInfo)
135.              self.BookmarkToolWindow.show()
136.
137.      def getLinkInfo(self,link):
138.          regex = re.compile('<title>(.*?)</title>')
139.          url = urllib.request.urlopen(link) #Gets the data from the url
140.          html = url.read() #Converts HTTPRequest into text
141.          html = str(html) #Changes from 'bytes' to str so re will work
142.          self.siteName = regex.search(html).group(1)
143.
144.
145.  if __name__ == '__main__':
146.      app = QApplication(sys.argv)
147.      window = TweetInterface()
148.      window.show()
149.      window.raise_()
150.       app.exec_()
```

10.5 TableView.py

```
1.  #Kurt Hornett
2.  #Comp4 Practical Project
3.  #Realtional Table view
4.  #Interfaces - Implemetation
5.
6.  from PyQt4.QtCore import *
7.  from PyQt4.QtGui import *
8.  from PyQt4.QtSql import *
9.
10. import sys
11.
12. class TableViewWindow(QMainWindow):
13.     def __init__(self):
14.         super().__init__()
15.
16.         #Set window title
17.         self.setWindowTitle('Database Management')
18.
19.         #Open and connect database
20.         self.createConnectedDatabase()
21.
22.         #Create Table View & Buttons
23.         self.tableView = QTableView()
24.         self.label = QLabel('Edit Bookmarks and click \'Submit\'')
25.         self.cancelButton = QPushButton('Cancel')
26.         self.submitButton = QPushButton('Submit')
27.
28.         #Create Layout
29.         self.layout = QGridLayout()
30.         self.layout.addWidget(self.label,0,0)
31.         self.layout.addWidget(self.tableView,1,0,1,2)
32.         self.layout.addWidget(self.cancelButton,2,0)
33.         self.layout.addWidget(self.submitButton,2,1)
34.
35.         #Create Central Widget
36.         self.tableWidget = QWidget()
37.         self.tableWidget.setLayout(self.layout)
38.         self.setCentralWidget(self.tableWidget)
39.
40.         #Create Table Model
41.         self.createTableModel()
42.
43.         #Connexions
```

```
44.         self.submitButton.clicked.connect(self.updateModify)
45.
46.     def createTableModel(self):
47.         self.model = QSqlRelationalTableModel()
48.         self.model.setEditStrategy(QSqlRelationalTableModel.OnManualSubmit)
49.         self.model.setTable('Bookmark')
50.         self.tableView.setModel(self.model)
51.         self.tableView.model().select()
52.
53.     def createConnectedDatabase(self):
54.         self.db = QSqlDatabase.addDatabase('QSQLITE')
55.         self.db.setDatabaseName('Bookmark_Database.db')
56.         self.db.open()
57.
58.     def updateModify(self):
59.         try:
60.             self.model.submitAll()
61.             doneMessage = QMessageBox()
62.             doneMessage.setWindowTitle('Confimation Message')
63.             doneMessage.setText('Database modified')
64.             doneMessage.exec_()
65.         except:
66.             error = QErrorMessage()
67.             error.showMessage('An Eror Occured')
68.             error.exec_()
69.
70.
71.
72. if __name__ == '__main__':
73.     app = QApplication(sys.argv)
74.     window = TableViewWindow()
75.     window.show()
76.     window.raise_()
77.     app.exec_()
```

10.6 DeleteInterface.py

```
1.  #Kurt Honrtt
2.  #Twitter Implemenatuib
3.  # Feb 2014
4.
5.  from PyQt4.QtCore import *
6.  from PyQt4.QtGui import *
7.  from PyQt4.QtSql import *
8.
9.  from sql_gui_misc import *
10. from twitter_database import *
11. from ConfirmDeleteDialog import *
12.
13. import sys
14.
15. class DeleteInterface(QMainWindow):
16.     def __init__(self):
17.         super().__init__()
18.         #Set Window Title
19.         self.setWindowTitle('Deletion Interface')
20.
21.         #Open and connect database
22.         self.createConnectedDatabase()
23.
24.         #Create Objects
25.         self.helpLabel = QLabel('Bookmarks available to delete: ')
26.         self.cancelButton = QPushButton('Cancel')
27.         self.deleteButton = QPushButton('To Delete')
28.         self.deleteTable = QTableView()
29.
30.         #Create Layout
31.         self.layout = QGridLayout()
32.         self.layout.addWidget(self.helpLabel,0,0)
33.         self.layout.addWidget(self.deleteTable,1,0,1,2)
34.         self.layout.addWidget(self.cancelButton,2,0)
35.         self.layout.addWidget(self.deleteButton,2,1)
36.
37.         #Set Central Widget
38.         self.deleteWidget = QWidget()
39.         self.deleteWidget.setLayout(self.layout)
40.         self.setCentralWidget(self.deleteWidget)
41.
42.         #Create Tabel Model
43.         self.createTableModel()
```

```
44.         self.createDeleteAction()
45.
46.         #connexions
47.         self.deleteButton.clicked.connect(self.showDeleteMenu)
48.
49.     def showDeleteMenu(self):
50.         self.deleteMenu.exec_(QCursor.pos())
51.
52.     def createTableModel(self):
53.         self.model = QSqlRelationalTableModel()
54.         self.model.setEditStrategy(QSqlRelationalTableModel.OnManualSubmit)
55.         self.model.setTable('Bookmark')
56.         self.deleteTable.setModel(self.model)
57.         self.deleteTable.model().select()
58.
59.     def createConnectedDatabase(self):
60.         self.db = QSqlDatabase.addDatabase('QSQLITE')
61.         self.db.setDatabaseName('Bookmark_Database.db')
62.         self.db.open()
63.
64.     def createDeleteAction(self):
65.         self.deleteMenu = QMenu()
66.         num, self.bookmarkList = getBookmarkNumber()
67.         for count in range(num):
68.             self.deleteMenu.addAction('Bookmark #
    {0}'.format(self.bookmarkList[count][0])).triggered.connect(self.getDeleteOption)
69.         self.deleteActionList = self.deleteMenu.actions()
70.     def getDeleteOption(self):
71.         self.deleteChoice = 0
72.         count = 0
73.         while self.deleteActionList[count].text() != self.sender().text():
74.             count +=1
75.             self.deleteChoice += 1
76.         self.confirmDelete()
77.
78.     def confirmDelete(self):
79.         try:
80.             self.confirmDialog = ConfirmDeleteDialog(self.bookmarkList,self.deleteChoice)
81.             self.confirmDialog.exec_()
82.             self.delete()
83.         except:
84.             error = QMessageBox()
85.             error.setText('An Error Occured')
86.             error.exec_()
87.
```

```
88.     def delete(self):
89.         deleteBookmarkGUI(self.bookmarkList[self.deleteChoice][0])
90.         self.createTableModel()
91.         self.createDeleteAction()
92.
93.
94.
95.
96.
97. if __name__ == '__main__':
98.     app = QApplication(sys.argv)
99.     window = DeleteInterface()
100.          window.show()
101.          window.raise_()
102.          app.exec_()
103.
```

10.7 SearchWindow.py

```
1.  #Kurt Honrtt
2.  #Twitter Implemenatuib
3.  # Feb 2014
4.
5.  from PyQt4.QtCore import *
6.  from PyQt4.QtGui import *
7.  from PyQt4.QtSql import *
8.
9.  from sql_gui_misc import *
10. from twitter_database import *
11. from ConfirmDeleteDialog import *
12.
13. import sys
14.
15. class DeleteInterface(QMainWindow):
16.     def __init__(self):
17.         super().__init__()
18.         #Set Window Title
19.         self.setWindowTitle('Deletion Interface')
20.
21.         #Open and connect database
22.         self.createConnectedDatabase()
23.
24.         #Create Objects
25.         self.helpLabel = QLabel('Bookmarks available to delete: ')
26.         self.cancelButton = QPushButton('Cancel')
27.         self.deleteButton = QPushButton('To Delete')
28.         self.deleteTable = QTableView()
29.
30.         #Create Layout
31.         self.layout = QGridLayout()
```

```
32.         self.layout.addWidget(self.helpLabel,0,0)
33.         self.layout.addWidget(self.deleteTable,1,0,1,2)
34.         self.layout.addWidget(self.cancelButton,2,0)
35.         self.layout.addWidget(self.deleteButton,2,1)
36.
37.         #Set Central Widget
38.         self.deleteWidget = QWidget()
39.         self.deleteWidget.setLayout(self.layout)
40.         self.setCentralWidget(self.deleteWidget)
41.
42.         #Create Tabel Model
43.         self.createTableModel()
44.         self.createDeleteAction()
45.
46.         #connexions
47.         self.deleteButton.clicked.connect(self.showDeleteMenu)
48.
49.     def showDeleteMenu(self):
50.         self.deleteMenu.exec_(QCursor.pos())
51.
52.     def createTableModel(self):
53.         self.model = QSqlRelationalTableModel()
54.         self.model.setEditStrategy(QSqlRelationalTableModel.OnManualSubmit)
55.         self.model.setTable('Bookmark')
56.         self.deleteTable.setModel(self.model)
57.         self.deleteTable.model().select()
58.
59.     def createConnectedDatabase(self):
60.         self.db = QSqlDatabase.addDatabase('QSQLITE')
61.         self.db.setDatabaseName('Bookmark_Database.db')
62.         self.db.open()
63.
64.     def createDeleteAction(self):
```

```
65.         self.deleteMenu = QMenu()
66.         num, self.bookmarkList = getBookmarkNumber()
67.         for count in range(num):
68.             self.deleteMenu.addAction('Bookmark #
    {0}'.format(self.bookmarkList[count][0])).triggered.connect(self.getDeleteOption)
69.         self.deleteActionList = self.deleteMenu.actions()
70.     def getDeleteOption(self):
71.         self.deleteChoice = 0
72.         count = 0
73.         while self.deleteActionList[count].text() != self.sender().text():
74.             count +=1
75.             self.deleteChoice += 1
76.         self.confirmDelete()
77.
78.     def confirmDelete(self):
79.         try:
80.             self.confirmDialog = ConfirmDeleteDialog(self.bookmarkList,self.deleteChoice)
81.             self.confirmDialog.exec_()
82.             self.delete()
83.         except:
84.             error = QMessageBox()
85.             error.setText('An Error Occured')
86.             error.exec_()
87.
88.     def delete(self):
89.         deleteBookmarkGUI(self.bookmarkList[self.deleteChoice][0])
90.         self.createTableModel()
91.         self.createDeleteAction()
92.
93.
94.
95.
96.
```

```
97. if __name__ == '__main__':
98.     app = QApplication(sys.argv)
99.     window = DeleteInterface()
100.            window.show()
101.            window.raise_()
102.            app.exec_()
103.
```

10.8 SuggetsedInterface.py

```
1.  #Kurt Honrtt
2.  #Twitter Implemenatuib
3.  # Feb 2014
4.
5.  from PyQt4.QtCore import *
6.  from PyQt4.QtGui import *
7.  from PyQt4.QtSql import *
8.
9.  from sql_gui_misc import *
10. from twitter_database import *
11. from ConfirmDeleteDialog import *
12.
13. import sys
14.
15. class DeleteInterface(QMainWindow):
16.     def __init__(self):
17.         super().__init__()
18.         #Set Window Title
19.         self.setWindowTitle('Deletion Interface')
20.
21.         #Open and connect database
22.         self.createConnectedDatabase()
23.
24.         #Create Objects
25.         self.helpLabel = QLabel('Bookmarks available to delete: ')
26.         self.cancelButton = QPushButton('Cancel')
27.         self.deleteButton = QPushButton('To Delete')
28.         self.deleteTable = QTableView()
29.
30.         #Create Layout
31.         self.layout = QGridLayout()
```

```
32.         self.layout.addWidget(self.helpLabel,0,0)
33.         self.layout.addWidget(self.deleteTable,1,0,1,2)
34.         self.layout.addWidget(self.cancelButton,2,0)
35.         self.layout.addWidget(self.deleteButton,2,1)
36.
37.         #Set Central Widget
38.         self.deleteWidget = QWidget()
39.         self.deleteWidget.setLayout(self.layout)
40.         self.setCentralWidget(self.deleteWidget)
41.
42.         #Create Tabel Model
43.         self.createTableModel()
44.         self.createDeleteAction()
45.
46.         #connexions
47.         self.deleteButton.clicked.connect(self.showDeleteMenu)
48.
49.     def showDeleteMenu(self):
50.         self.deleteMenu.exec_(QCursor.pos())
51.
52.     def createTableModel(self):
53.         self.model = QSqlRelationalTableModel()
54.         self.model.setEditStrategy(QSqlRelationalTableModel.OnManualSubmit)
55.         self.model.setTable('Bookmark')
56.         self.deleteTable.setModel(self.model)
57.         self.deleteTable.model().select()
58.
59.     def createConnectedDatabase(self):
60.         self.db = QSqlDatabase.addDatabase('QSQLITE')
61.         self.db.setDatabaseName('Bookmark_Database.db')
62.         self.db.open()
63.
64.     def createDeleteAction(self):
```

```
65.         self.deleteMenu = QMenu()
66.         num, self.bookmarkList = getBookmarkNumber()
67.         for count in range(num):
68.             self.deleteMenu.addAction('Bookmark #
    {0}'.format(self.bookmarkList[count][0])).triggered.connect(self.getDeleteOption)
69.         self.deleteActionList = self.deleteMenu.actions()
70.     def getDeleteOption(self):
71.         self.deleteChoice = 0
72.         count = 0
73.         while self.deleteActionList[count].text() != self.sender().text():
74.             count +=1
75.             self.deleteChoice += 1
76.         self.confirmDelete()
77.
78.     def confirmDelete(self):
79.         try:
80.             self.confirmDialog = ConfirmDeleteDialog(self.bookmarkList,self.deleteChoice)
81.             self.confirmDialog.exec_()
82.             self.delete()
83.         except:
84.             error = QMessageBox()
85.             error.setText('An Error Occured')
86.             error.exec_()
87.
88.     def delete(self):
89.         deleteBookmarkGUI(self.bookmarkList[self.deleteChoice][0])
90.         self.createTableModel()
91.         self.createDeleteAction()
92.
93.
94.
95.
96.
```

```
97. if __name__ == '__main__':
98.     app = QApplication(sys.argv)
99.     window = DeleteInterface()
100.            window.show()
101.            window.raise_()
102.            app.exec_()
103.
```

10.9 ConfirmDeleteDialog.py

```
1. #Kuyrt Hornrtg
2. #Teittwer Impeplemratuoneo
3. #Feb 2014
4.
5. from PyQt4.QtCore import *
6. from PyQt4.QtGui import *
7. from PyQt4.QtSql import *
8.
9. import sys
10.
11.
12. class ConfirmDeleteDialog(QDialog):
13.     def __init__(self,list_,choice):
14.         super().__init__()
15.         #Set Window Title
16.         self.setWindowTitle('Confimr Deletion')
17.
18.         self.list = list_
19.         self.choice = choice
20.
21.         #Create Layout
22.         self.helpLabel = QLabel('This is going to be deleted')
23.         self.submitButton = QPushButton('Submit')
24.         self.titleLabel = QLabel('Title')
25.         self.titleLineEdit = QLineEdit()
26.         self.titleLineEdit.setReadOnly(True)
27.         self.linkLabel = QLabel('Link')
28.         self.linkLineEdit = QLineEdit()
29.         self.linkLineEdit.setReadOnly(True)
30.         self.siteNameLabel = QLabel('Site Name')
31.         self.siteNameLineEdit = QLineEdit()
```

```
32.         self.siteNameLineEdit.setReadOnly(True)
33.         self.siteDescLabel = QLabel('Site Description')
34.         self.siteDescLineEdit = QLineEdit()
35.         self.siteDescLineEdit.setReadOnly(True)
36.
37.         self.confirmLayout = QGridLayout()
38.         self.confirmLayout.addWidget(self.helpLabel,0,0)
39.         self.confirmLayout.addWidget(self.titleLabel,1,0)
40.         self.confirmLayout.addWidget(self.titleLineEdit,1,1,1,3)
41.         self.confirmLayout.addWidget(self.linkLabel,2,0)
42.         self.confirmLayout.addWidget(self.linkLineEdit,2,1)
43.         self.confirmLayout.addWidget(self.siteNameLabel,2,2)
44.         self.confirmLayout.addWidget(self.siteNameLineEdit,2,3)
45.         self.confirmLayout.addWidget(self.siteDescLabel,3,0)
46.         self.confirmLayout.addWidget(self.siteDescLineEdit,3,1,1,3)
47.         self.confirmLayout.addWidget(self.submitButton,4,3)
48.
49.         self.setLayout(self.confirmLayout)
50.
51.         self.setData()
52.
53.         self.submitButton.clicked.connect(self.close)
54.
55.     def setData(self):
56.         self.titleLineEdit.setText(self.list[self.choice][1])
57.         self.linkLineEdit.setText(self.list[self.choice][4])
58.         self.siteNameLineEdit.setText(self.list[self.choice][2])
59.         self.siteDescLineEdit.setText(self.list[self.choice][3])
```

10.10 BookmarkTool.py

```
1.  #Kurt Hornett
2.  # COMP4 Practical Project
3.  # Implemetation - Interfaces
4.  # Bookmark Creation interface
5.
6.  from PyQt4.QtCore import *
7.  from PyQt4.QtGui import *
8.  from PyQt4.QtSql import *
9.
10. from twitter_database import *
11. from sql_gui_misc import *
12.
13. import sys
14.
15. class BookmarkWindow(QDialog):
16.     def __init__(self,timeline,choice,userList,userChoice):
17.         super().__init__()
18.
19.         #Set Window Title
20.         self.setWindowTitle('Bookmark Creation Tool')
21.
22.         #Set Data
23.         self.timeline = timeline
24.         self.choice = choice
25.         self.userList = userList
26.         self.userChoice = userChoice
27.
28.         #Create Labels
29.         self.titleLabel = QLabel('Bookmark Title')
30.         self.linkLabel = QLabel('Link')
31.         self.siteNameLabel = QLabel('Site Name')
```

```
        self.siteDescLabel = QLabel('Site Description')

        #Create Line Edits
        self.titleLineEdit = QLineEdit()
        self.linkLineEdit = QLineEdit()
        self.linkLineEdit.setReadOnly(True)
        self.siteNameLineEdit = QLineEdit()
        self.siteNameLineEdit.setReadOnly(True)
        self.siteDescLineEdit = QLineEdit()

        #Create Push Button
        self.cancelButton = QPushButton('Cancel')
        self.submitPushButton = QPushButton('Submit')

        #Create Layout
        self.BookmarkLayout = QGridLayout()
        self.BookmarkLayout.addWidget(self.titleLabel,0,0)
        self.BookmarkLayout.addWidget(self.titleLineEdit,0,1,1,3)
        self.BookmarkLayout.addWidget(self.linkLabel,1,0)
        self.BookmarkLayout.addWidget(self.linkLineEdit,1,1)
        self.BookmarkLayout.addWidget(self.siteNameLabel,1,2)
        self.BookmarkLayout.addWidget(self.siteNameLineEdit,1,3)
        self.BookmarkLayout.addWidget(self.siteDescLabel,2,0,)
        self.BookmarkLayout.addWidget(self.siteDescLineEdit,2,1,1,3)
        self.BookmarkLayout.addWidget(self.cancelButton,3,0,1,2)
        self.BookmarkLayout.addWidget(self.submitPushButton,3,2,1,2)

        self.bookmarkWidget = QWidget()
        self.bookmarkWidget.setLayout(self.BookmarkLayout)
        self.setLayout(self.BookmarkLayout)

        #Set conexions
        self.cancelButton.clicked.connect(self.return_)
```

```
        self.submitPushButton.clicked.connect(self.addBookmark)

    def setData(self,bookmarkInfo):
        self.bookmarkInfo = bookmarkInfo
        self.linkLineEdit.setText(bookmarkInfo[0])
        self.siteNameLineEdit.setText(bookmarkInfo[1])

    def addBookmark(self,):
        try:
            db = 'Bookmark_database.db'
            tweetData = (self.timeline[self.choice]['text'],self.userList[self.userChoice][2])
            addTweetGUI(tweetData)
            tweetID = getLatestTweetGUI()
            bookmarkData =
(self.titleLineEdit.text(),self.siteNameLineEdit.text(),self.siteDescLineEdit.text(),self.linkLineEdit.text(),tweetID[0][0])
            addBookmarkGUI(bookmarkData,db)
            doneMessage = QMessageBox()
            doneMessage.setText('Bookmark Successfully added')
            doneMessage.exec_()
        except:
            error = QErrorMessage()
            error.showMessage('An Error Occured')
            error.exec_()

    def return_(self):
        self.close()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = BookmarkWindow()
    window.show()
    window.raise_()
    app.exec_()
```

97.
98.
99.

10.11 sql_gui_misc.py

```
1. #SQL GUI Miscellany
2. #Kurt Hornett
3. #GUI Implemetation
4.
5. import sqlite3
6. from twitter_database import *
7.
8. def databaseUserList():
9.     sql = '''SELECT Username,ScreenName,UserID
10.          From User'''
11.    data = ()
12.    users = textQuery(sql,data)
13.    return users
14.
15. def textQuery(sql,data):
16.    with sqlite3.connect('Bookmark_database.db') as db:
17.        cursor = db.cursor()
18.        cursor.execute('PRAGMA foreign_keys = ON')
19.        cursor.execute(sql,data)
20.        results = cursor.fetchall()
21.    return results
22.
23. def getUsersFromDatabaseGUI():
24.    sql = '''SELECT Username,ScreenName,UserID
25.             From User'''
26.    db = 'Bookmark_Database.db'
27.    users = textQuery(sql,())
28.    return users
29.
30. def addBookmarkGUI(data,db):
31.    sql = '''INSERT INTO Bookmark(BookmarkTitle,SiteName,SiteDescription,Link,TweetID)
```

```
32.             VALUES(?,?,?,?,?)'''
33.     query(sql,data,db)
34.
35. def addTweetGUI(data):
36.     sql = '''INSERT INTO Tweet(TweetText,UserID) VALUES(?,?)'''
37.     db = 'Bookmark_Database.db'
38.     query(sql,data,db)
39.
40. def getLatestTweetGUI():
41.     sql = '''SELECT TweetID FROM Tweet WHERE TweetID = (SELECT MAX(TweetID) FROM Tweet)'''
42.     db = 'Bookmark_Database.db'
43.     maxId = searchQuery(sql,(),db)
44.     return maxId
45.
46. def getBookmarkNumber():
47.     sql = '''SELECT * FROM Bookmark'''
48.     db = 'Bookmark_Database.db'
49.     data = ()
50.     list_ = textQuery(sql,data)
51.     num = len(list_)
52.     return num, list_
53.
54. def deleteBookmarkGUI(Dchoice):
55.     sql = '''DELETE FROM Bookmark WHERE BookmarkID = ?'''
56.     db = 'Bookmark_Database.db'
57.     query(sql,(Dchoice,),db)
58.
59. def addSuggestedUser(data):
60.     sql = '''INSERT INTO User(UserName,ScreenName)
61.             VALUES(?,?)'''
62.     db = 'Bookmark_Database.db'
63.     query(sql,data,db)
64.
```

65.
66.

10.12 create_database.py

```
1. # Kurt Hornett
2. # Database
3. # Sep 2013
4.
5. import sqlite3
6.
7. def create_database(db_name,table_name,sql):
8.     with sqlite3.connect(db_name) as db:
9.         cursor = db.cursor()
10.        cursor.execute('select name from sqlite_master where name=?',(table_name,))
11.        result = cursor.fetchall()
12.        keep_table = True
13.        if len(result) == 1:
14.            response = input("Table {0} alread exists, recreate? (y/n): ".format(table_name))
15.            if response == 'y':
16.                keep_table = False
17.                print('Table {0} recreated - all previous data shall be lost.'.format(table_name))
18.                cursor.execute("drop table if exists {0}".format(table_name))
19.                db.commit()
20.            else:
21.                print('Existing table kept')
22.        else:
23.            keep_table = False
24.        if not keep_table:
25.            cursor.execute(sql)
26.            db.commit()
27.
28. if __name__ == "__main__":
29.
30.     db_name = "Bookmark_Database.db"
31.     sql = """CREATE TABLE User
```

```
32.             (UserID INTEGER,
33.             UserName TEXT,
34.             ScreenName TEXT,
35.             PRIMARY KEY(UserID))"""
36.     sql2 = "'CREATE TABLE Tweet
37.             (TweetID INTEGER,
38.             TweetText TEXT,
39.             UserID INTEGER,
40.             PRIMARY KEY(TweetID),
41.             FOREIGN KEY(UserID) REFERENCES User(UserID) ON UPDATE CASCADE ON DELETE
    CASCADE)"'
42.     sql3 = "'CREATE TABLE Bookmark
43.             (BookmarkID INTEGER,
44.             BookmarkTitle TEXT,
45.             SiteName TEXT,
46.             SiteDescription TEXT,
47.             Link TEXT,
48.             TweetID INTEGER,
49.             PRIMARY KEY(BookmarkID),
50.             FOREIGN KEY(TweetID) REFERENCES Tweet(TweetID) ON UPDATE CASCADE ON DELETE
    CASCADE)
51.             "'
52.     create_database(db_name,"User",sql)
53.     create_database(db_name,"Tweet",sql2)
54.     create_database(db_name,"Bookmark",sql3
```

User Manual

1. Table of Contents

Contents

2. Introduction

The purpose of this program is to provide the user with a means of accessing Twitter in order to bookmark tweets and other information about the tweets locally, from users which they have specified themselves, that have also been stored locally. It allows adding users to a database, searching their latest tweets and storing those that have been selected in a database – along with any information those tweets hold, such as links and site names – and searching for suggested users to add to the program's database. Further it allows editing and deleting of bookmarks the user has made. This user manual shall go through the many processes of the system.
The system's intended audience are those who require a system that allows for the bookmarking in tweets in such a way that allows information regarding the tweets to be stored locally; developed specifically for the client, Mr. Adam McNicol, a computing teacher from a Sixth Form College.

3. Installation

3.1 Pre-requisites

The installation of the system on one's local computer requires a few tools in order to fully run the system without significant error. The following pieces of software are necessary for running the program:

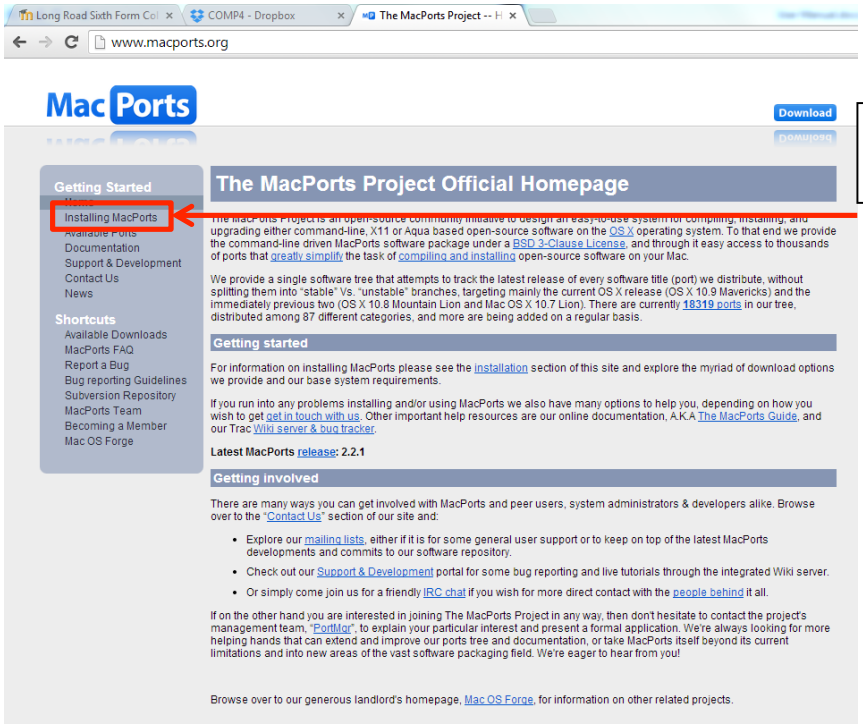
1. MacPorts
2. Python 3.3
3. IDLE
4. PyQt 4
5. PIP
6. Python Twitter Tools

In order to help the user install all the necessary requirements for the system a short guide to installing these components shall be given. All of these installation guides assume that the operating system being used in an Intel based version of Mac OS X.

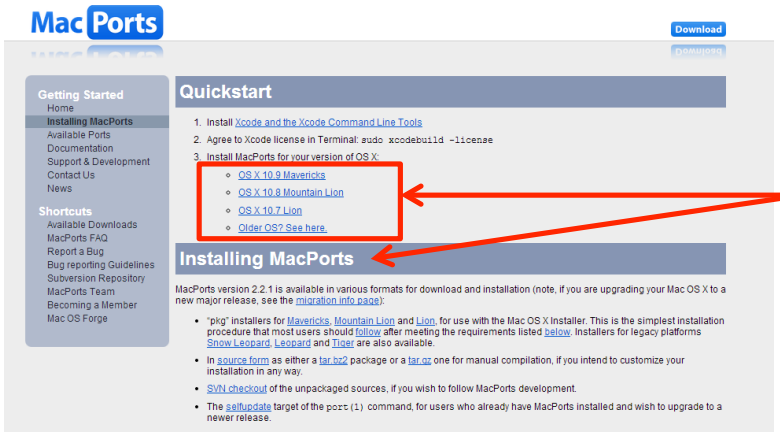
3.1.1 MacPorts

In order to successfully install many of the system's components MacPorts is needed, this allows for all the components to be stored in one place on the computer, which is far more streamlined than the alternative of downloading each component separately.

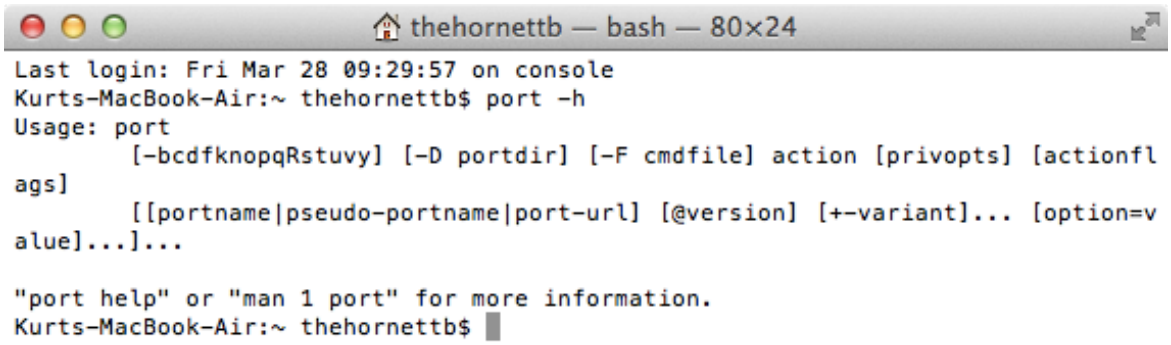
1. Go to macports.org.
2. On this website there is a section which has instructions on how to install the system, called 'Installing MacPorts'



3. Installing and using MacPorts requires XCode and certain utilities from XCode, ensure you have these from the Mac App Store.
4. On this subsection are the necessary packages (.pkg files) needed to install the program based on what version of Mac OS X you are using. Install the relevant version.



5. The MacPorts system is used in the Terminal.
    a. To test whether the installation has been completed successfully, run `port -h` in the terminal, which should display a list of commands the program takes.

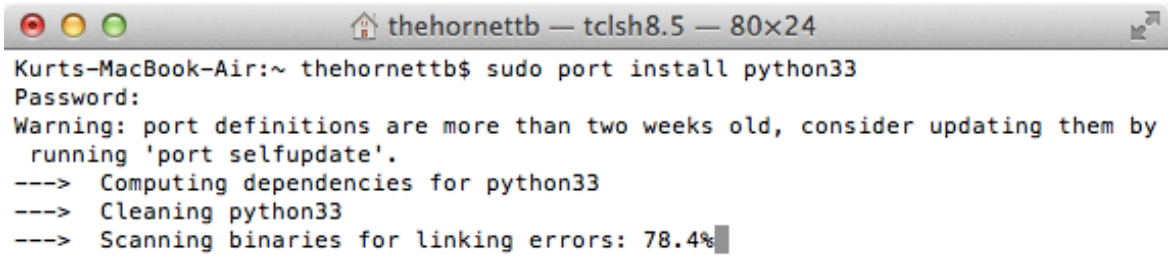


This is the screen with the command run, which confirms the installation of the program.

3.1.2 Python 3.3 & IDLE

Python 3.3 and IDLE are both installed using the CLI MacPorts interface. Assuming the above installation went correctly:

1. Open the Terminal Application on your computer.
2. Run `port install python33` in the shell in order to install python 3.3 through MacPorts.
    a. You may wish to run `sudo port install python33` to install the program in the root directory and thus system-wide. This will require the root user's password.



This is the screen with the installation running, if the program installs correctly it should show you.

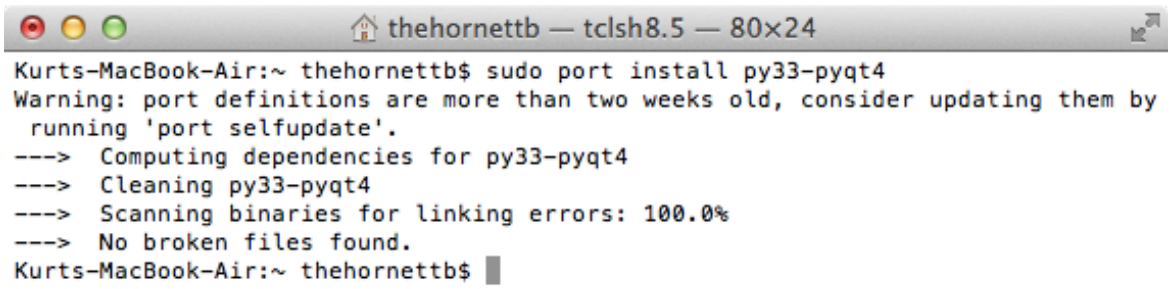
3. After running these commands python shall be installed in the `/Applications/MacPorts` folder on your system.

This is the directory with Python installed, within the applications folder.

3.1.3 PyQt4

PyQt4 is also installed as a MacPort and thus has similar steps as installing Python and IDLE.

1. Open the Terminal Application on your computer
2. Run `port install py33-pyqt4` in the shell in order to install PyQt4 through MacPorts
    a. You may have to run `sudo port install py33-pyqt4` to install the program in the root directory and thus system-wide. This will require the root user's password.



This is the terminal installing the program, but as it is already installed the whole process is not shown here.

3. After running these commands PyQt4 should be installed on your computer and can be found in the `/Applications/MacPorts` folder on your computer.

3.1.4 PIP

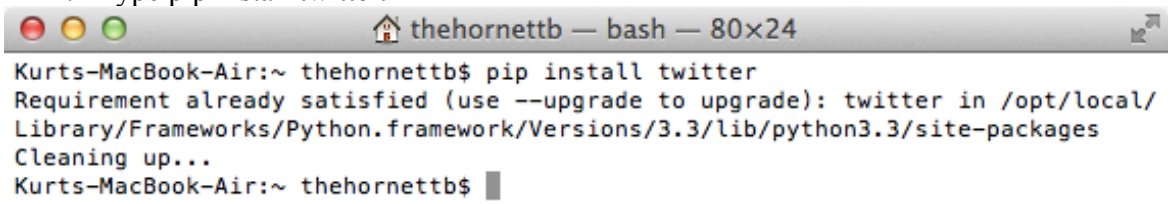
Pip is a tool that allows for managing and updating python modules from the Python Package Index (pypi.python.org/pypi).

1. First go to the PIP website at pip-installer.org, this site includes all the information needed to install PIP.
2. Click on *Install or Upgrade PIP* to find the instructions on how to install PIP.
    a. This requires downloading a python script and then running the script in the Terminal using the correct version of python, which may have to be run using the command `python3.3` rather than `python`.
3. Once that is done PIP should be ready to use.

3.1.5 Python Twitter Tools

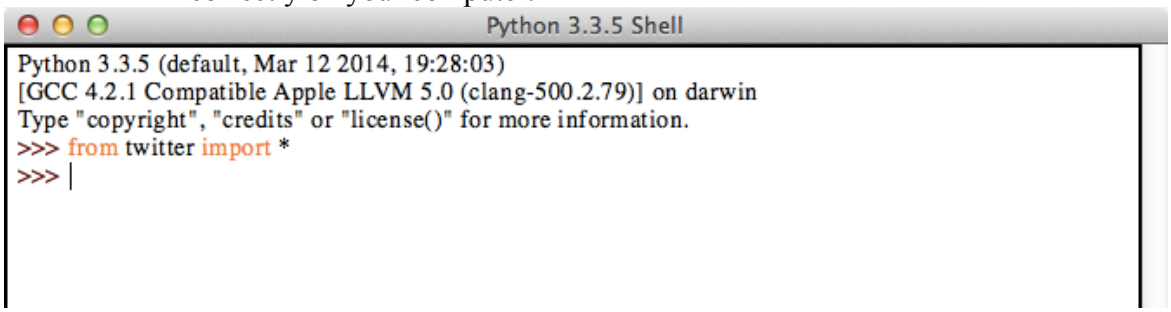
This is the module required to run all of the Twitter modules in python, the program cannot do anything without this module and is therefore very important. Information about this module can be found at mike.verdone.ca/twitter/

1. Open the Terminal.
2. Type pip install twitter.



This is the screen with the pip install dialogue, as it is already install it is not installed here.

3. After the installation is complete Python Twitter Tools should run on your system.
    a. To test this, open the correct version of IDLE installed earlier and run, in the python shell, `from twitter import * .`
    b. If this runs without error then Python Twitter Tools has been installed correctly on your computer.



This is the python command running without error, this is how it should appear after installing Python Twitter Tools.
After completing these steps your system shall be ready to run the program on your computer.

3.1.6 Hardware Requirements

The hardware required to run this program is not of great importance as long as the above pieces of software are installed on their operating system in the right way. This manual has assumed that the user will be using a Mac OS X based system and has therefore been tailored to that system, as it is of the clients wishes. However there are some base needs including:

- A Keyboard and Mouse

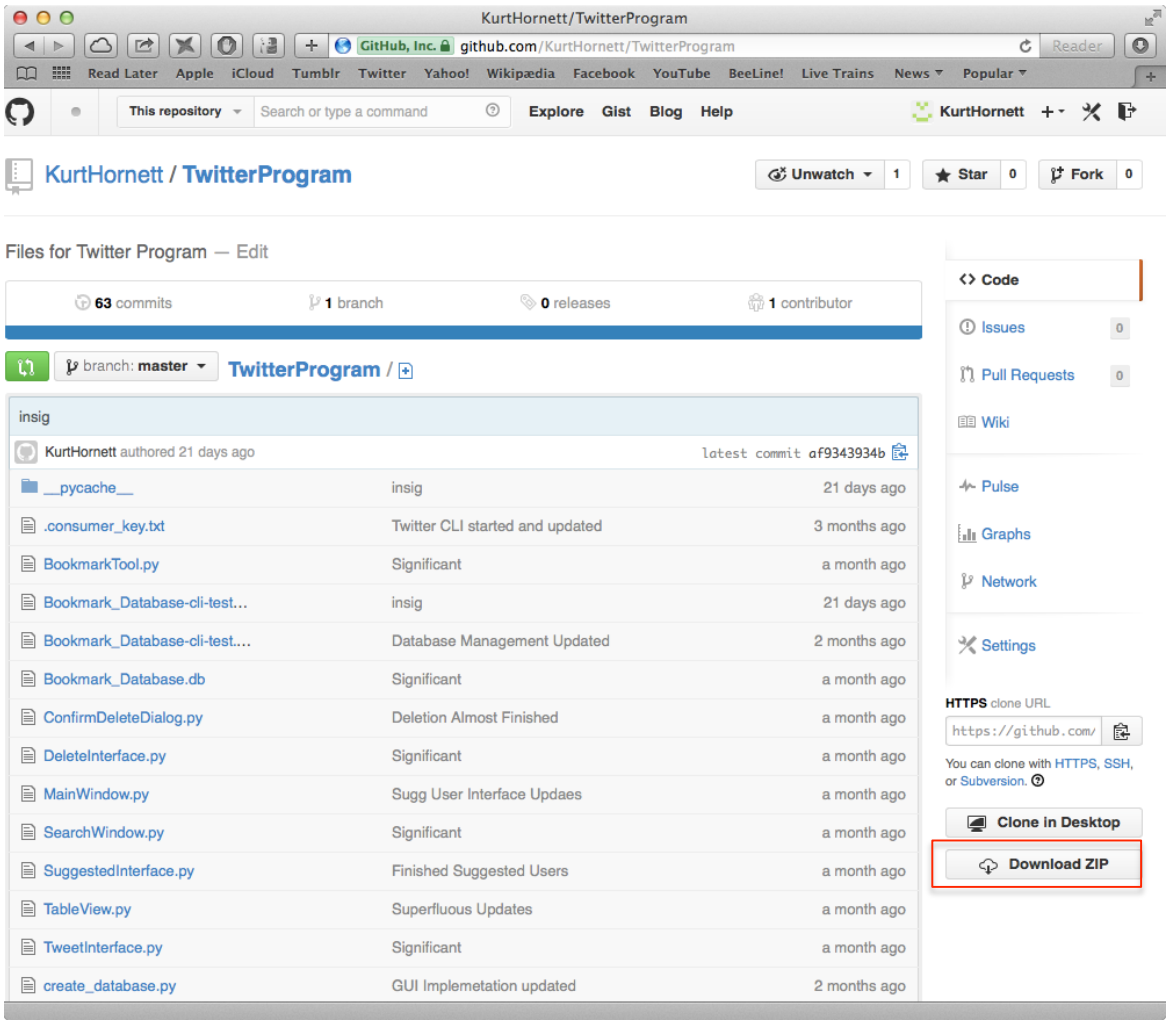
- A Monitor
- Hard Disk Storage for the Database
- A minimum of 512MBs of RAM
- An Internet connection, preferably without download limits.

Provided the user has these it should be easy enough to use the program as any other.

3.2 System Installation

In order to install the system to your computer requires a few steps that include downloading the system's source code from GitHub. This can be done in many ways but it is preferred that you do it by downloading a zip file from the web interface.
In order to do this:

1. Go to 'https://github.com/KurtHornett/TwitterProgram' where the source code is



   hosted. It should look like this in your browser:

2. On the above image is a highlighted area, click this to download the program to your computer; If you are using a mac then it shall automatically unzip itself.

3. Take the folder or zip and place it where you would like it on your computer.

3.3 Running the System

Depending on how you wish to run the program, as a CLI or GUI is up to the user, to use the CLI version of the program, in the terminal run `python3.3 twitter_cli.py` while in the directory it is stored. However this guide shall assume the user wishes to use the GUI and all the tutorials ahead shall pertain to this version only.
To run the GUI version you can either double-click on the file and then press F5 (or fn + F5 if you're using a laptop or MacBook, and it is setup to use function keys in that way) on your keyboard to run the program, or you can enter, in the correct directory, in the terminal `python3.3 MainWindow.py` which shall proceed to run the program.

4. Tutorial

4.1 Introduction

This section of the User Manual shall be used to explain and illustrate how to use all of the main features of the program in such a way that is easy to understand and facilitates the use of the program; it shall also explain common errors and difficulties that arise out of using the program.
Although this program has been developed and has been tested to run on a Mac, some of the screenshots below may contain a Windows environment. The steps however are exactly the same for the Mac OS X version, albeit menu controls are handled by universal menu bars on the Operating System and not by the program itself on the Mac interface.

4.2 Assumptions

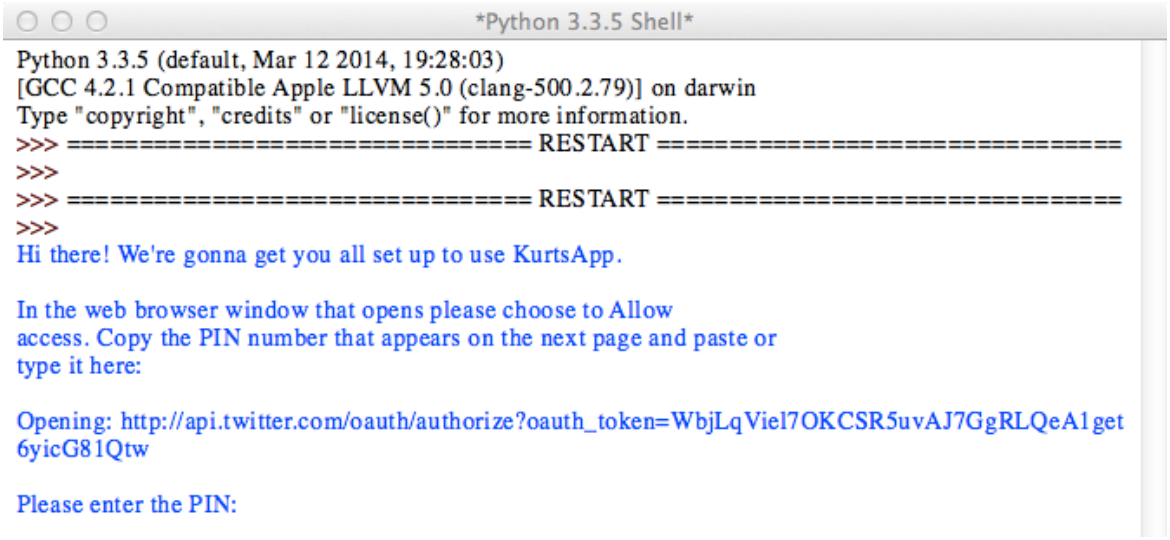
It has been assumed that there is much prior experience of using the Mac OS X environment and using computers in general, and so to the beginner it may not be so easy to follow, however attempts to make this guide easy enough to be understood by many shall be made. It is expected that the user have a working knowledge of the Mac UI and its universal menu bar, further it has been assumed that the user have a working knowledge of using the Terminal in order to install the necessary components required to run the program. The Tutorial section shall be split up on a question basis so that the user knows what to look for when they have an issue with the program.
This Tutorials Question:

4.3 The Tutorial

4.3.1 How do I login into Twitter to use the program?

This is the very first step needed in order to be able to use the program at all, as without Authorisation from Twitter you will not be able to pull information from their API.
This process will run as soon as the user runs the program for the first time and will not be displayed to the user again after it has been completed. A description of how to delete the user's credentials will be included at the end of this tutorial section.
This process is handled by the CLI, whether that is in the Terminal or through python's IDLE interface. This tutorial shall show how this is done though the IDLE shell.

1. When the program is first run this screen shall be shown, also the default Internet browser shall be displayed asking the user to Log in to Twitter.



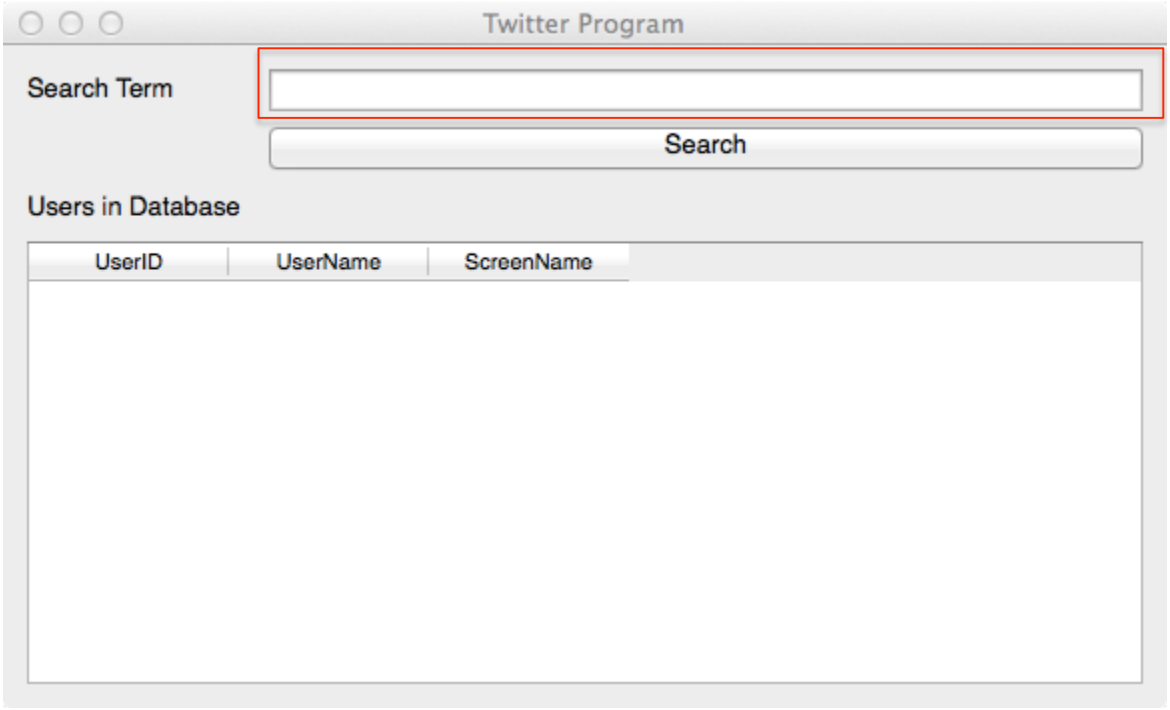
Here is the CLI interface in which the user must enter a PIN provided by the Twitter API in a browser window.

2. When you enter the PIN given to you from the Internet Browser Login the system shall be authorized and the initial Search User Interface shall be displayed.
3. It also displays where the login credentials file has been stored. If you wish to use a different user later in time this file will have to be deleted; so keep note of its location.

4.3.2 How do I add a user to the database?

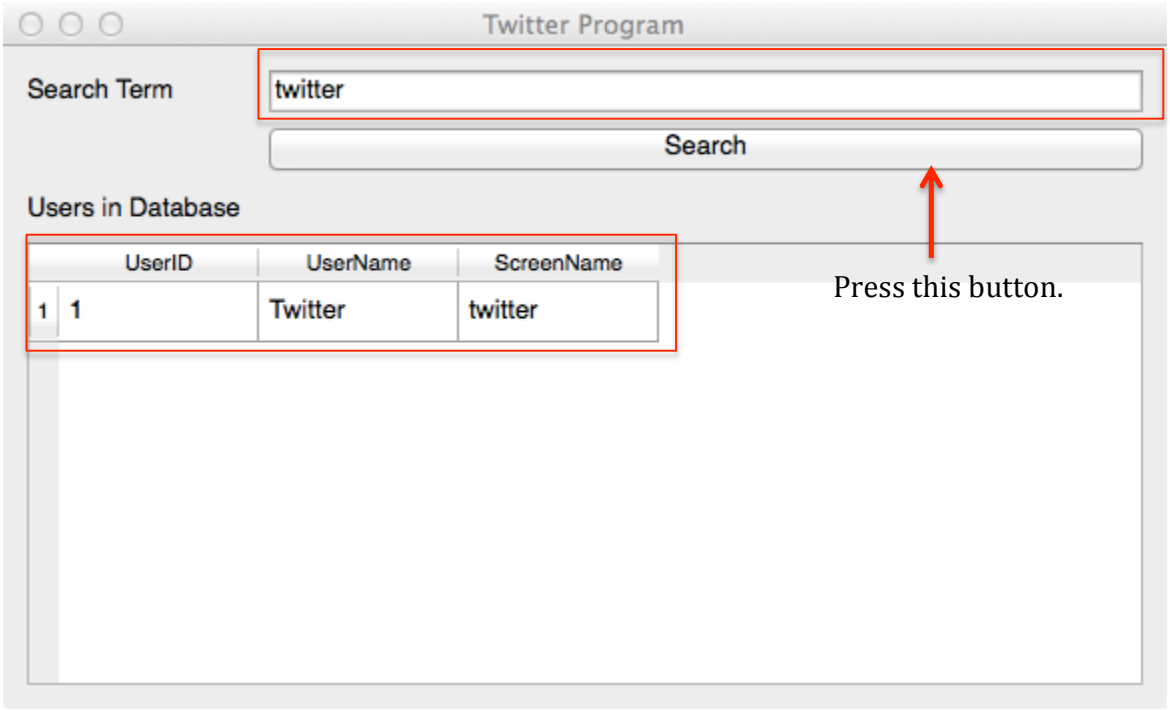
This is a very simple process and is required if you wish to add bookmarks from a user's tweets.

1. Firstly, on the main interface there is a search pane in which you can enter the name of a Twitter User that can be added to the database. It is highlighted in the images below.



2. In this field the user can enter a name of a user to add to the database; if the field is left empty then a message will appear asking the user to enter some details. When a valid twitter handle is entered and the search button pressed the Table View underneath will be updated and show the user just added; illustrated in the below screenshot.

3. After completing this step the user has been added to the database. If you try to add a

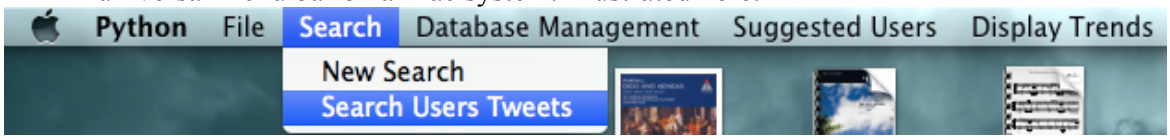


user that already exists in the database, then you shall be given a message that the user already exists and therefore cannot be added to the database.

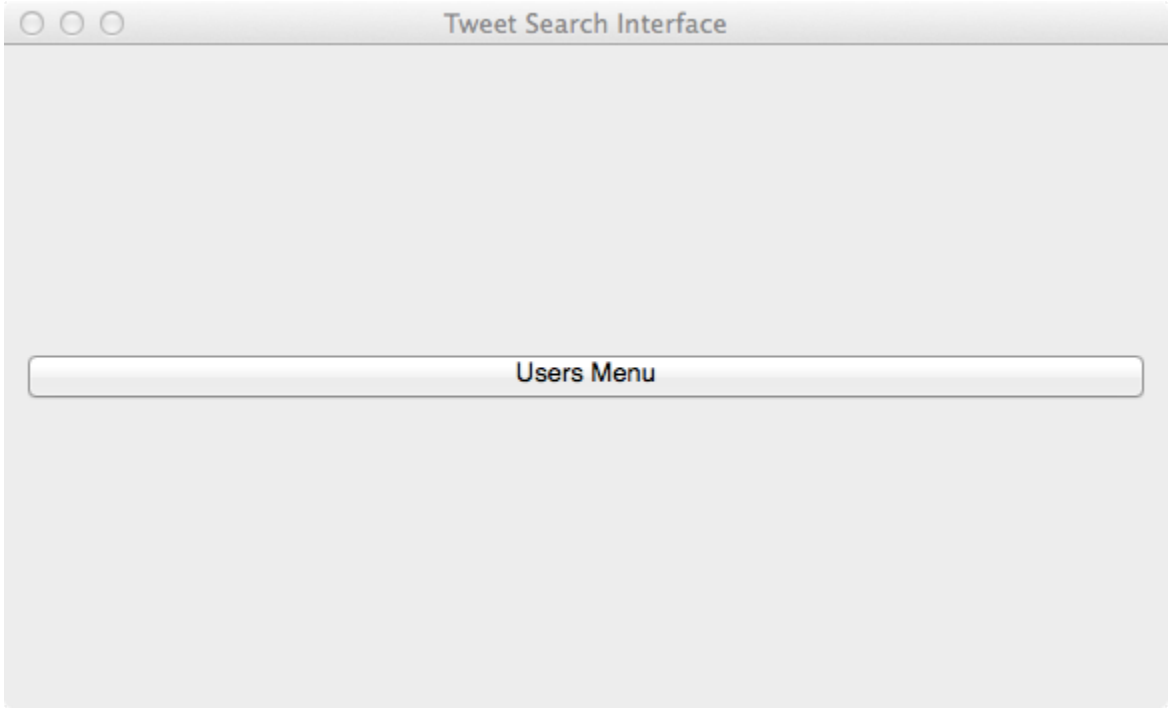
4.3.3 How do I bookmark a tweet from an added user?

This is a rather simple process as well and does not require much technical knowledge.

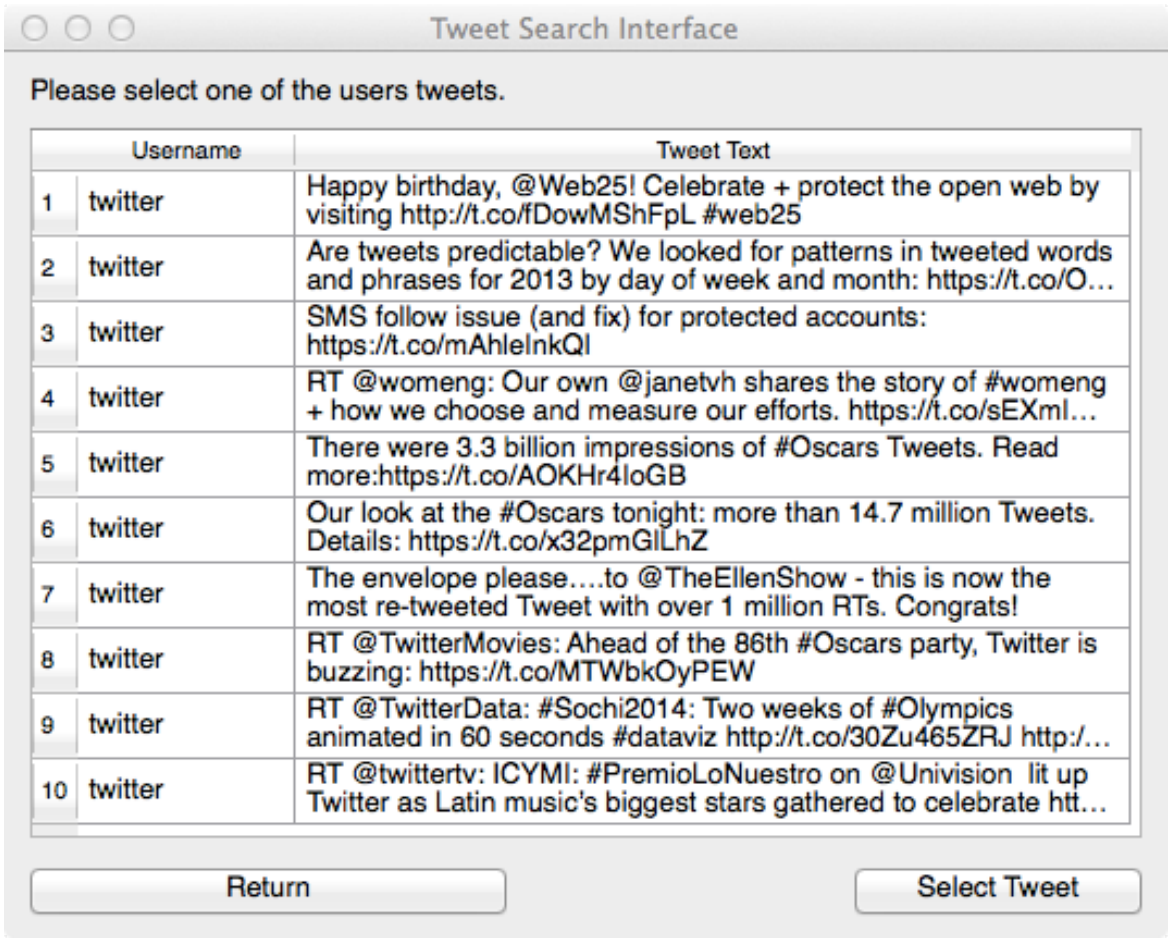
1. First you need to switch to the Search User Tweets Interface, this is done through the universal menu bar on a Mac system. Illustrated here:



After this has been clicked the user should see this:

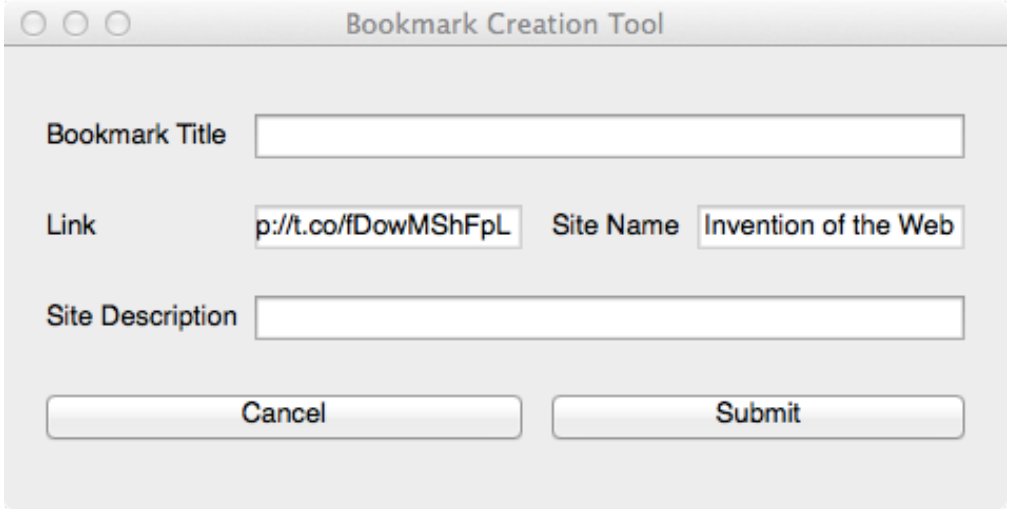


2. You should then click on the button on the middle of the screen, a list of all the users on the system will be displayed and you must click one of these to display their 10 latest tweets. After doing so this interface shall be presented to you:

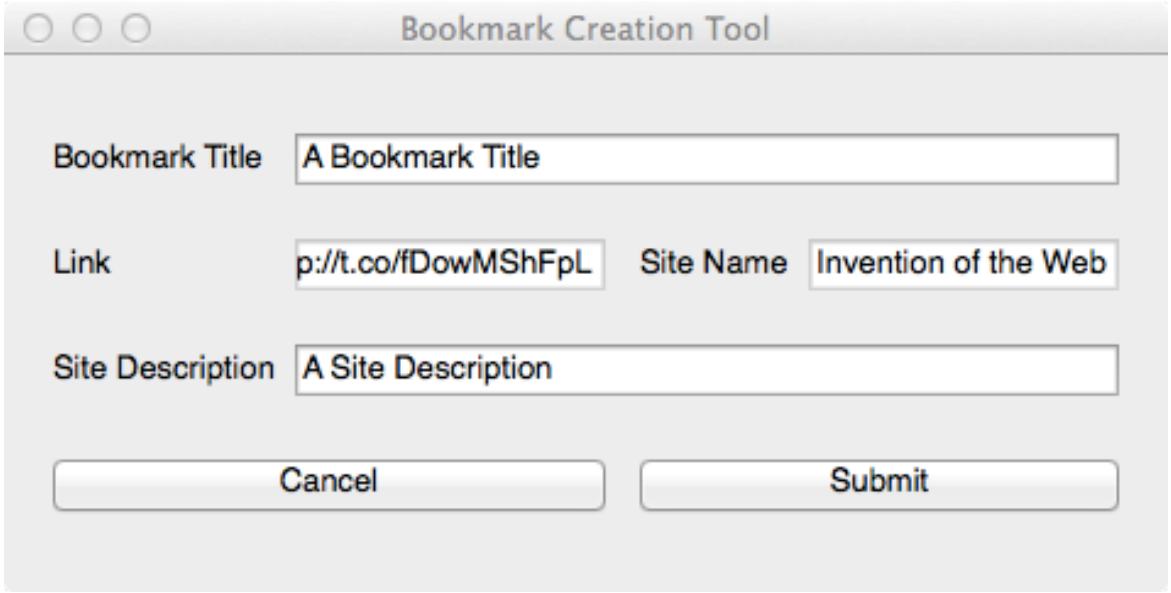


This images show the interface with the second column expanded, you must do this yourself, as it is not displayed at full width.

3. You must then select a tweet to save to the database. Once you have done this you must click the 'Select Tweet' button where a list of the tweets is displayed corresponding to their numbers on the table. After selecting this this window appears:



4. On this interface you can enter a new bookmark title and a site description, however the Link and Site Name fields are read-only because they ought not be edited. An example of some information is here:

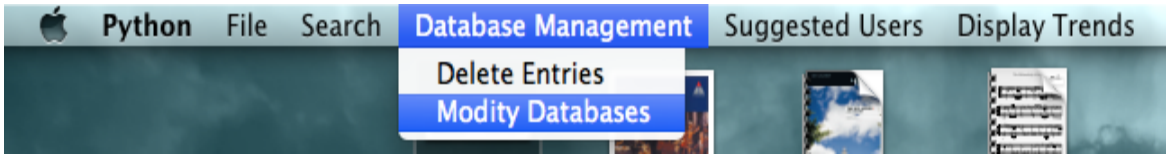


5. After doing this you can the click 'Submit' to add the bookmark to the database, click 'cancel' to exit the Bookmark Creation Tool.
6. That is all that is needed to add a bookmark to the database.

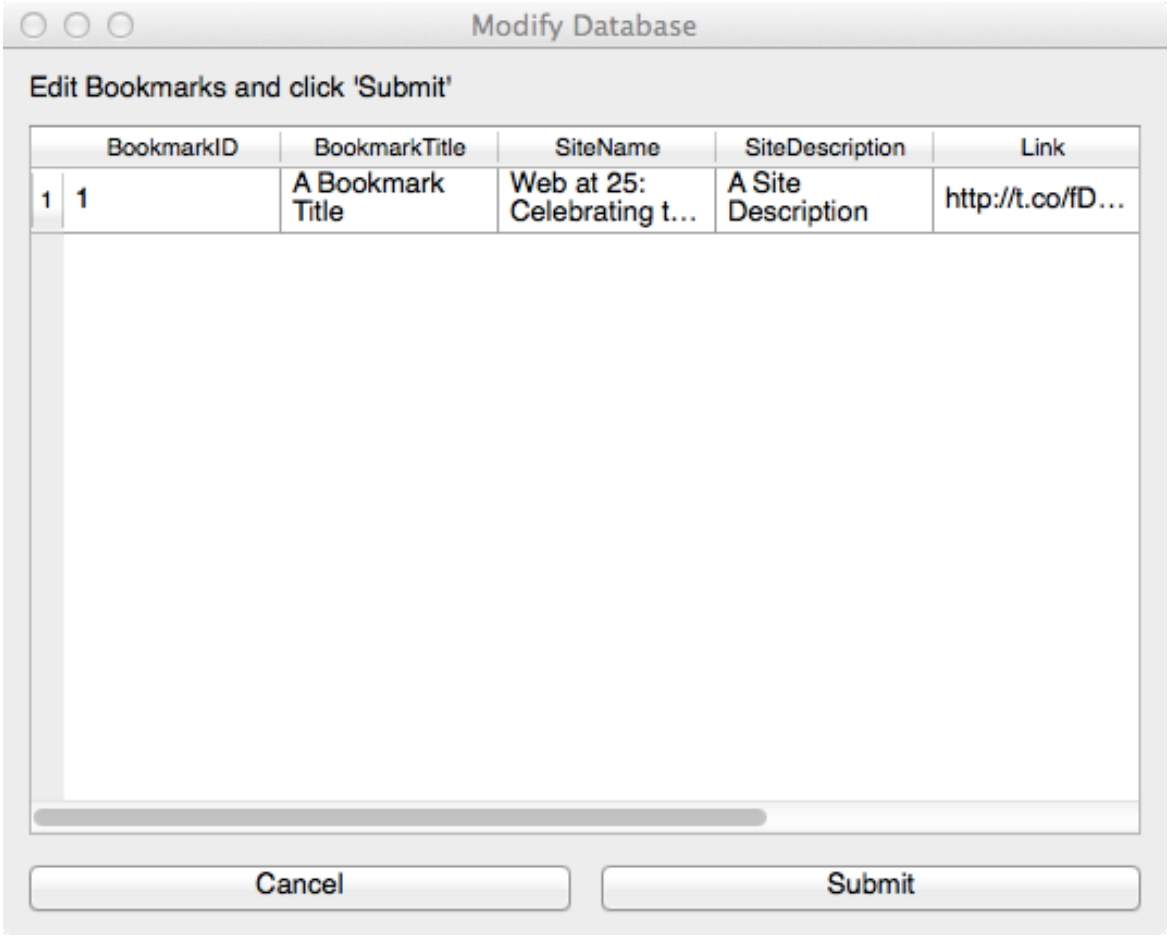
4.3.4 How do I modify a bookmark already in the database?

This is again a very simple process that does not require detailed technical knowledge.

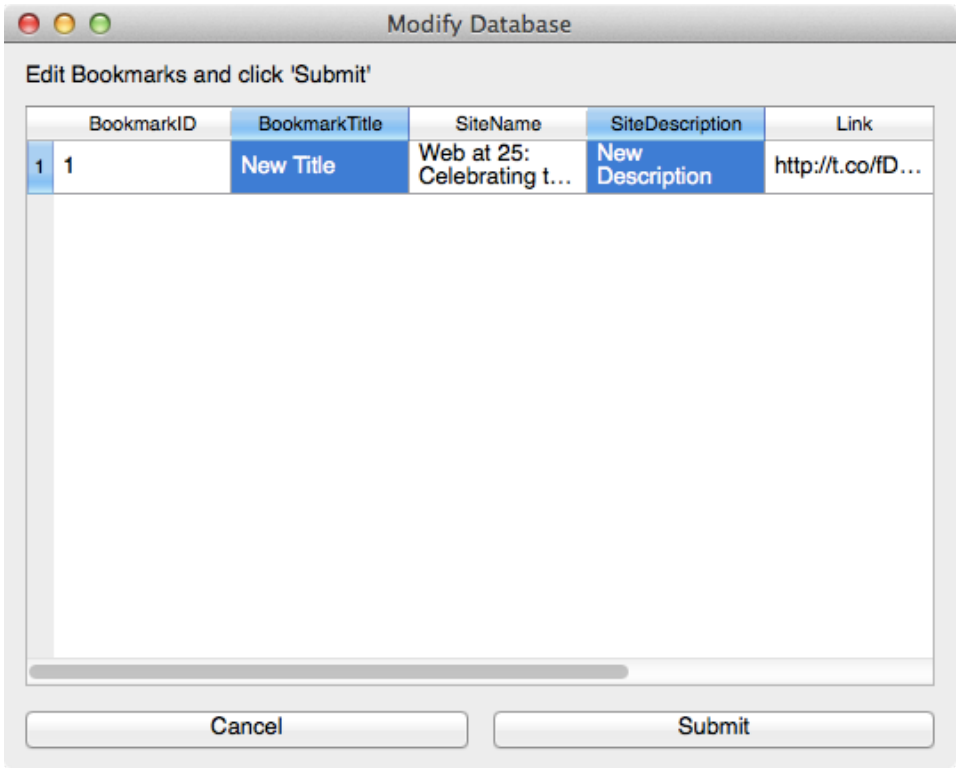
1. First, you must switch to the bookmark modify interface from the universal menu bar, as with the previous section.



After clicking that, this interface will appear:

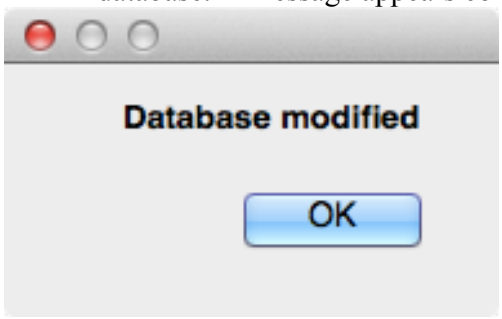


2. To modify a bookmark in the database, double click on any field you wish to edit, type in the new information, and you ***must*** press the return key on your keyboard. An example of edited information:



Highlighted is the changed information, this is not how the window is displayed on your computer unless you select both after having edited the information.

3. After making the desired changes, press 'Submit', which pushes the changes to the database. A message appears confirming the changes:

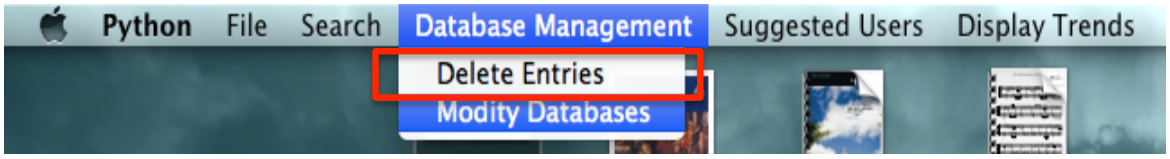


4. This means that the database has been successfully modified.

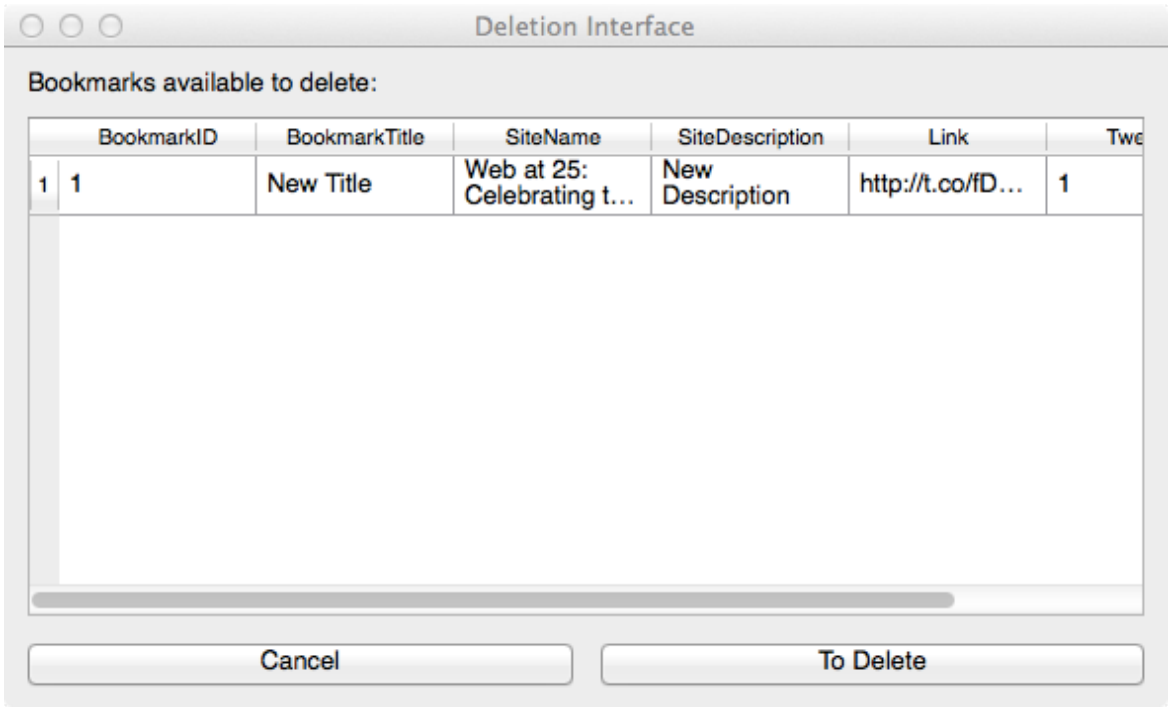
4.3.5 How do I delete a bookmark from the database?

This process is very similar to the modify tutorial, but it rather simpler and requires fewer steps.

1. Switch to the delete interface from the universal menu bar:

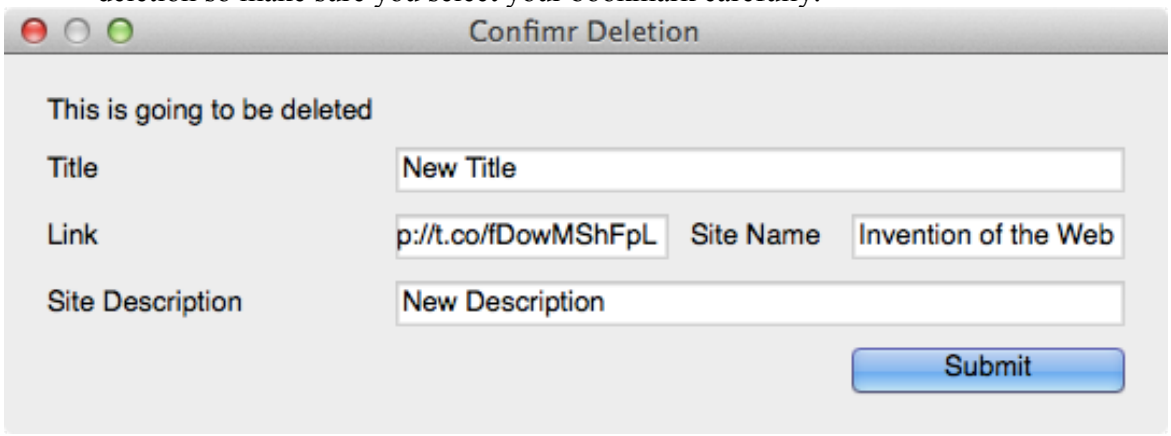


Which displays this interface:



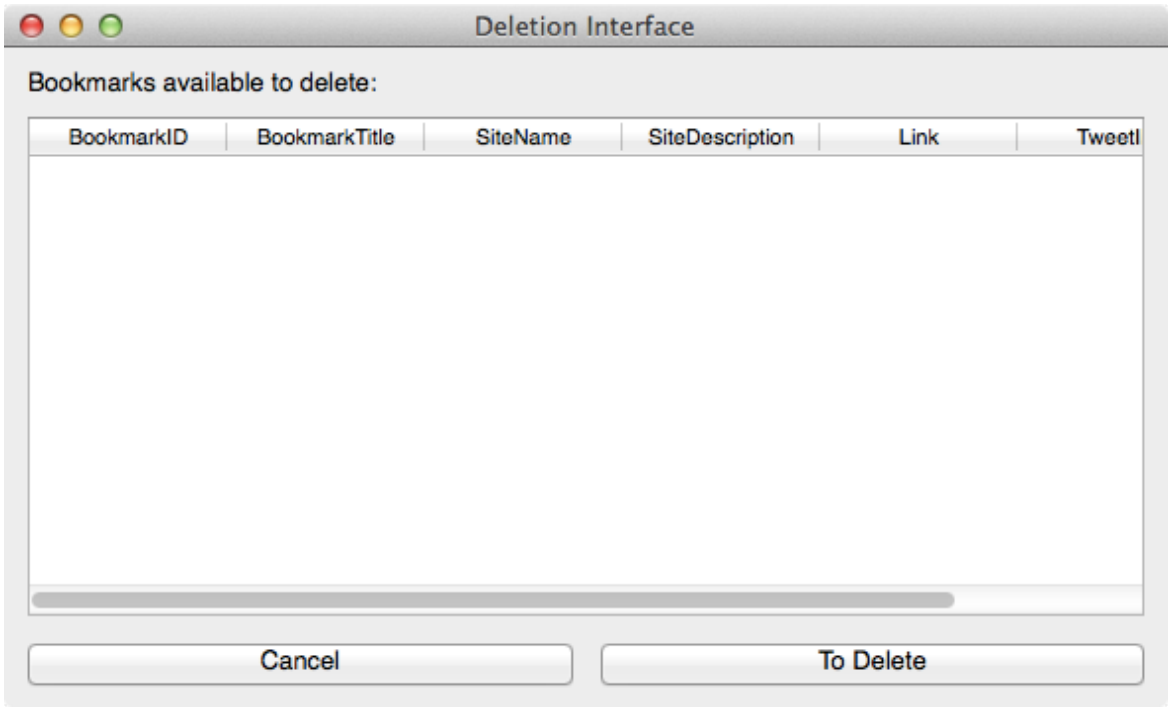
As you can see it is very similar to the previous modify interface.

2. Once you have decided what bookmark you wish to delete, click the 'To Delete' button and select the bookmark with the corresponding *BookmarkID*.
3. After that a screen confirming the deletion appears, this **will not** let you cancel the deletion so make sure you select your bookmark carefully.



This is the confirmation screen that has the bookmark information.

4. After clicking 'Submit' the bookmark will be deleted and the Table will be updated to show this fact.

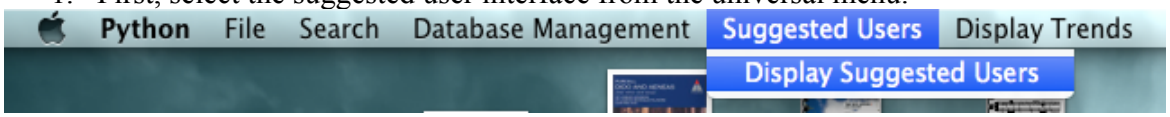


The empty table after deleting a bookmark.
And that is how a bookmark is deleted in the program.

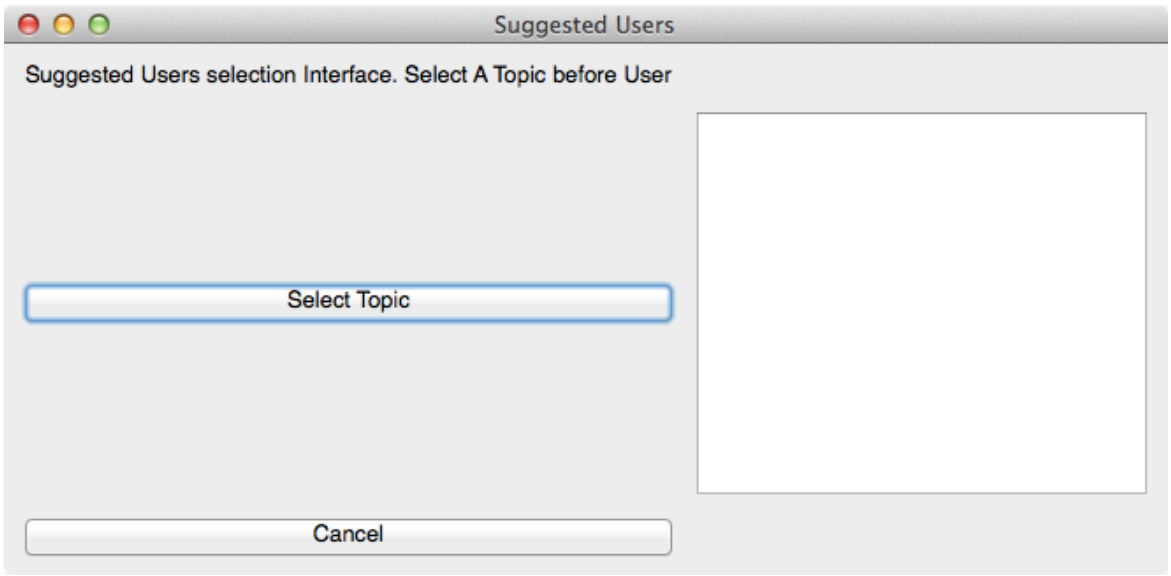
4.3.6 How do I add a suggested user to the database?

This is the final process that this tutorial shall explain to you. It is also a simple process that requires not much work to complete.

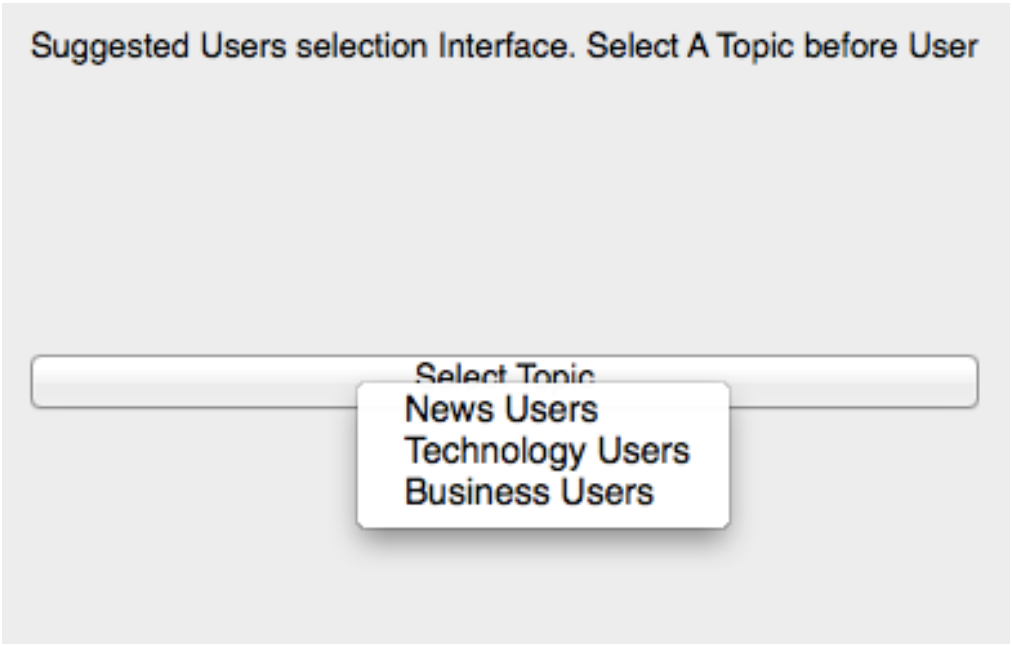
1. First, select the suggested user interface from the universal menu.



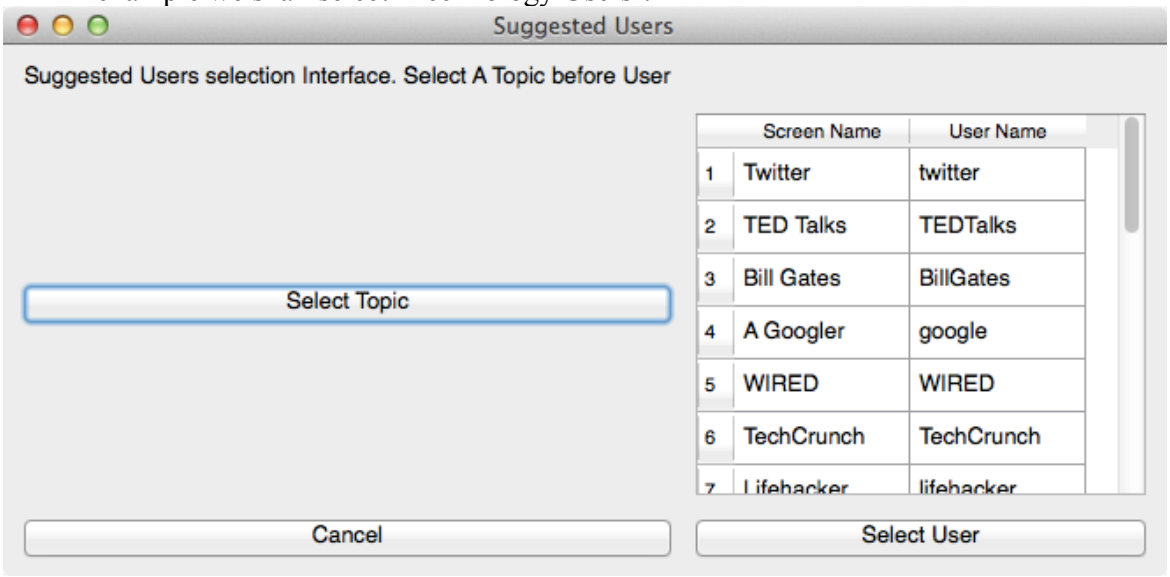
Which changes to this interface:



2. In order to see the users which can be added to the database in the field to the right you must select an option from the 'Select Topic' menu; Press the button and choose from the selection of 3:

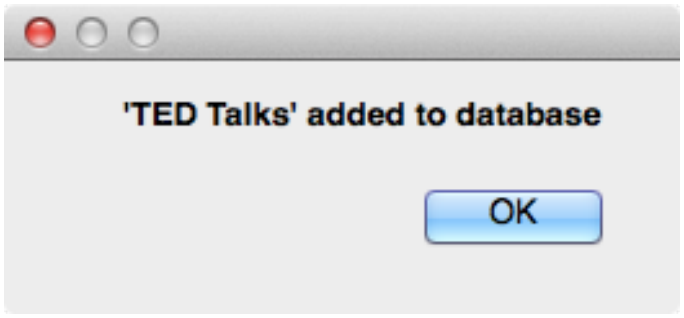


3. After selecting an option the users will appear in the table to the right, for this example we shall select 'Technology Users':



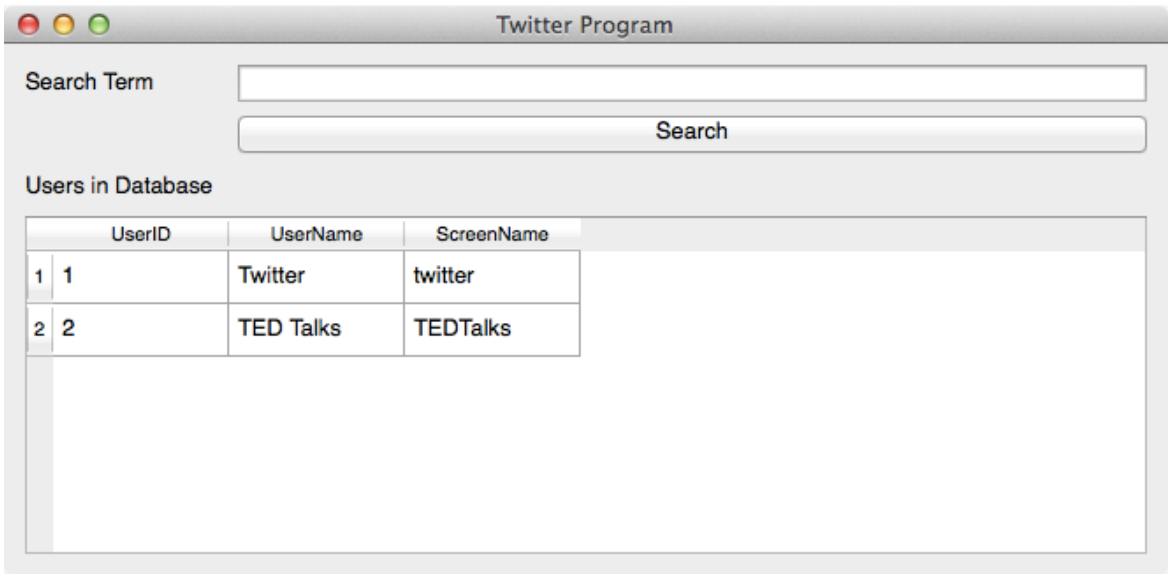
The Table on the right populated by users.

4. Choose a user from the list, and then press 'Select User' where you will again find a menu with the respective numbers of each user, select the correct number and that user will be added to the database. For this example we have selected 'TED Talks'. A message will appear saying the user has been added.



5. To see the user in the database, select 'New Search from the universal menu.
    a. N.B. If 'Cancel' is selected and not 'New Search' from the menu then the new user will not appear in the database as desired.

6. On that screen shall be the user in the database:



And so you have added a new user to the database, which can be used to store bookmarks.

5. Limitations

There are a few limitations to the system that must be addressed in this tutorial so that the user understands why these sections may not work.
Firstly, on the system's menu in the universal menu bar there is a section that is called 'Display Trends' with an action that does not do anything. This is due to this part of the system not being developed and therefore is a remnant of what was initially planned for the system.
Furthermore, as already stated, in the bookmark deletion interface, after having made a choice of which bookmark to delete, one cannot reverse the changes made to the database, one cannot cancel the bookmark deletion. Therefore the user needs to be prudent when going to delete bookmarks.

7. Errors

7.1.1 Trouble with the 'Search User Tweets interface'

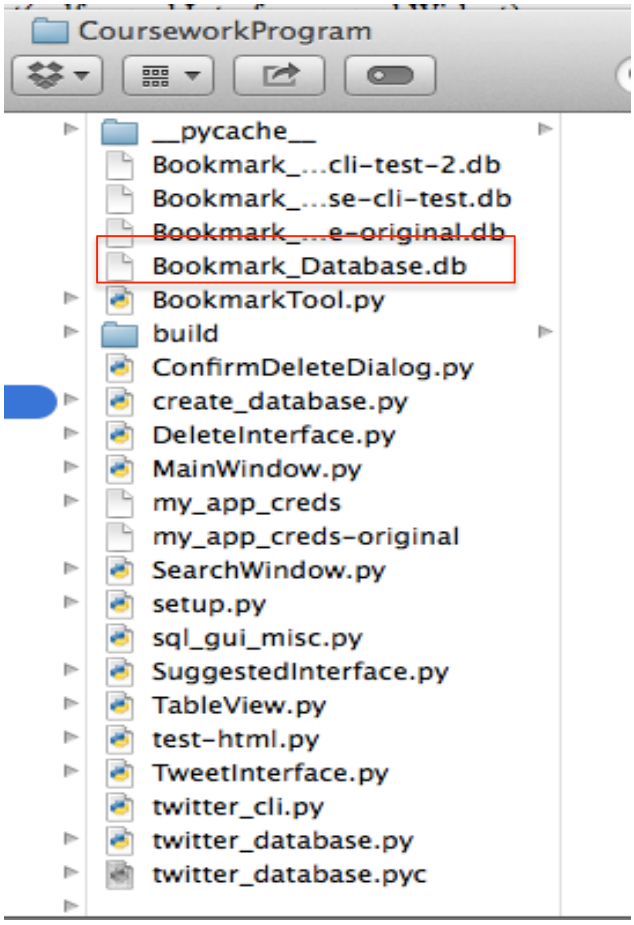
One issue with the system is that it often occurs that after adding a user to the database they do not appear on the list of users to search the tweets of. If this happens then one must restart the system by closing the window and the re-running the system.

8. System recovery

This system has no inbuilt method of backing up and restoring the database or information that is created on the system. However that does not mean that it is impossible to do this. Here are some steps on how to back up and then restored the data.

8.1 Backing up

1. In order to back up the systems database you will have to find the .db file on the computer.
2. Go to the folder where you have installed the program, in this folder is a file called 'Bookmark-Database.db'



3. This file can be copied and placed into an online storage system that you use (such a ‘Dropbox’ or ‘SkyDrive’) for safe keeping until use need to use it again.

8.2 Restoring data

1. In order to restore the data, simple put the ‘Bookmark-Database.db’ file back into the programs folder, as long as it has the same name it shall work fine in the program

Evaluation

1. Customer Requirements

In this section I shall go through the general and specific requirements that were made in the analysis of the system. A summary of the objectives can be found in the Analysis section (pp.7-28, §3), which contains them as a list; in this section only the general and specific objectives shall be referenced to as the core and other objectives are not necessarily relevant to the implementation of the program.

1.1 General Objectives

1.1.1 Logging in to Twitter

Objective:

User must be able to log into twitter

This Objective has been met.

Fulfilled:

This objective has been successfully fulfilled; this is done through the user of the Twitter wrapper in Python. This interacts with the Twitter API and causes a browser window to be displayed where the user of the system is able to enter their user credentials.

Evidence:

Below is an image of the login dialogue that is displayed on the Internet browser. Furthermore in the evidence section the client has confirmed that he has been able to log into twitter with his own credentials. Evidence of this is also in the Testing Section from Figures 3a-e; the program successfully passed the test and therefore proves that the program can do this function.

This image shows the webpage shown when initializing the program for the first time, pressing 'Authorize App' allows the user to see a PIN for the program.

1.1.2 Easy Interface

Objective:

Easy to understand intuitive interface

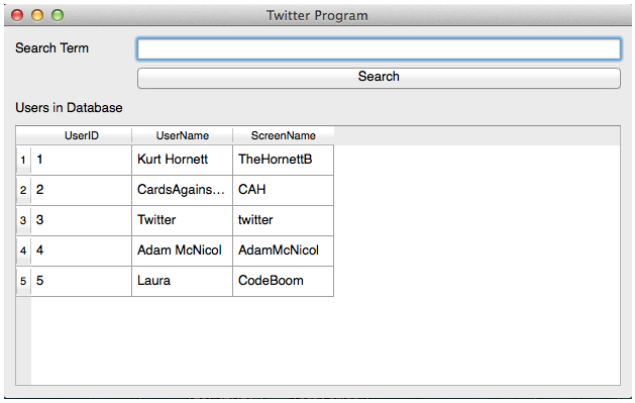
This Objective has been met.

Fulfilled:

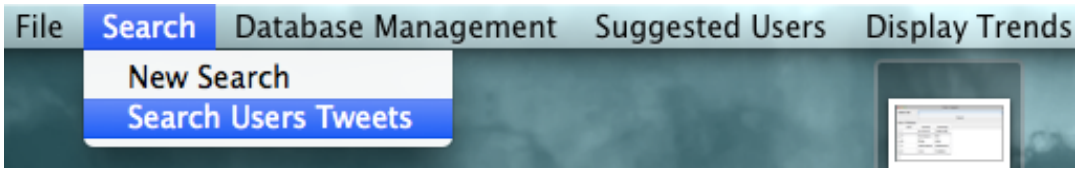
This objective has been satisfied very well. The system has an interface that follows many of the conventions of interfaces in use today, such as using push buttons and utilizing universal/local menu bars, the former being the case with Mac OS X, of which the system has been developed for.

Evidence:

Evidence of this is found in the User feedback (Criticism to be expanded upon in §2), appended at the end of this Evaluation section, and from the Testing section. Screenshots of the system below also demonstrate the system's familiarity.



These two images show the main interface and part of the universal menu bar on the Mac system that is used for navigation of the majority of the interface.



1.1.3 Getting Tweets from the API

Objective:

Must receive tweets from the Twitter API

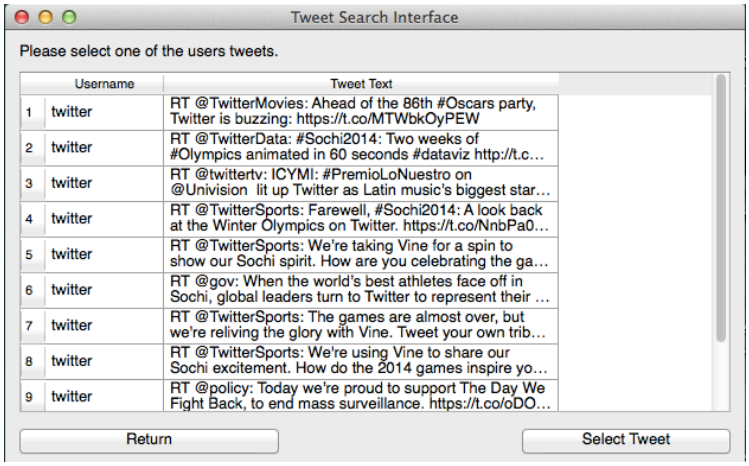
This Objective has been met.

Fulfilled:

This objective has been successfully fulfilled; the user of the system can easily display the tweets from users stored in the database. This is done through a system in the interface that pulls the tweets from twitter and then displays them in a table in the interface.

Evidence:

The evidence for this being completed can be found in the evidence appendix, where the client has stated that he has been able to carry out the function. Furthermore the system has been able to do this, proved by the testing section, where a test was carried out to explain this module. Some screenshots have also been included here to show exactly how this has been achieved.



This image shows the table with the tweets pulled from the Twitter API displayed to the user, confirming that the objective has been met.

1.1.4 Searching the Timeline

Objective:

Must be able to search tweets from timeline

This Objective has not been met.

Fulfilled:

This objective has not been met; this is due to a change in the focus of the program from being focussed on the finding of tweets in the user's timeline to a focus on the user's tweets themselves. Instead of finding a tweet before a user, a user is found before the tweet, and therefore this has become unnecessary. Furthermore as not a large amount of tweets are displayed on the interface it would not be necessary to search those, however if more were displayed then it might make sense to implement this function.

1.1.5 Storing Tweet Information

Objective:

Must be able to store information from tweets such as links

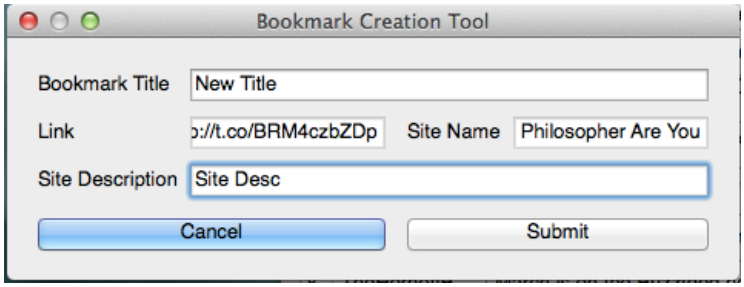
This Objective has been met.

Fulfilled:

This objective has been met. This is handled by the bookmarking system that has been implemented in the system overall. This allows the tweets from a user to be added to a database that includes the links and the site information of a tweet from a Twitter User.

Evidence:

Evidence of the fulfilment of this part of the system can be found in the Evidence Appendix, where again the client states being able to carry out this function, further in the Testing section this has been tested and has proven to be successful.



Here is the Bookmark Creation tool that is used to add bookmarks to the database; it is elaborated

1.1.6 Following Trends

Objective:

Must be able to follow trends

This Objective has not been met.

Fulfilled:

This objective has not been met. This is due to the time constraints on the implementation stage of the system. It could not be implemented because of the time spend creating a fully functioning system in relation to the many other elements of the program. It is therefore that the trends section still appears as an option in the program, but is however not in use in the program.

1.1.7 Suggested Users

Objective:

Must obtain or generate suggested followers

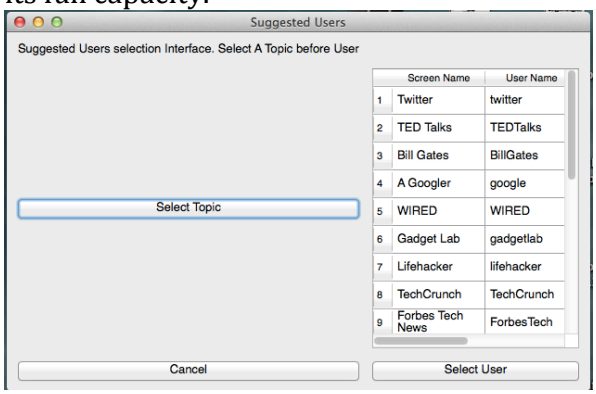
This Objective has been met.

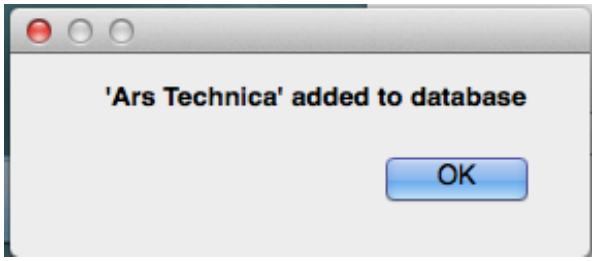
Fulfilled:

This objective has been met in the implementation stage. It is handled by its own specific interface that allows the user to select a topic and obtain users based on that topic that can then be added to the database easily through the use of menus on the interface.

Evidence:

Evidence for this can be found again in the evidence appendix, which contains answers from the client about the suggested users function, and from the testing section. Below are screenshots of the interface to show it working to its full capacity.





What these two windows show is the suggested user interface that is included in the program. The first shows the full interface that is used to select the users from the list in the left hand side of the image. The second show the confirmation message of a user being added to the database, from the suggested list. This displays clear evidence that the suggested users part of the system is fully functional. More information can be found

1.2 Specific Objectives

1.2.1 Login Credentials

Objective:

The user must be able to provide his/her own credentials to log onto twitter

This Objective has been met.

Fulfilled:

This objective has been met. This is done through the use of OAuth Verification that means that the user must enter his/her own credentials to be able to use the application at all. Therefore it is not possible for the user to access the system without using his/her own credentials.

Evidence:

Evidence for this can be found in the Testing Section (pp.62-99, §3.5) and from the client's feedback, from Question 1, answered affirmatively, asking whether they are able to login with his or her own credentials. No screenshots are provided as to access the page, login credential are required, however §1.1.1 of this document provides an image of the page before entering credentials, which can act of proof of what is discussed here.

1.2.2 Storing the user credentials

Objective:

These credentials must be stored securely on the computer

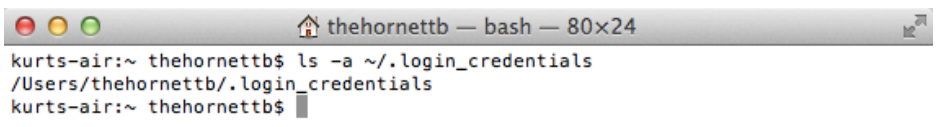
This Objective has been met.

Fulfilled:

This has been fulfilled. The Python Twitter Tools module comes with a function called 'oauth_dance' which allows the details of the user, their OAuth keys, to be stored in a file on the computer in such a way that is not human readable. Furthermore, they are not stored in the program's directory but in the user's home directory, adding a layer to the security of where the details are held. Also the file to which they are stored is hidden and not displayed by default in the file manager, but can easily be accessed by the command 'ls' in the terminal.

Evidence:

As evidence the file (called '.login_credentials') as it is in the home folder of the user shall be shown. As the program does not crash when using this function it must be the case that the details have been stored successfully.



What is shown in this image is the use of the terminal to display a search for the file, as it is a hidden file, it is not displayed by default in the file manager, hence the use of the terminal.

1.2.3 Logging out between sessions

Objective:
The user must be able to log out and also to stay logged in between sessions

This Objective has not been met.

Fulfilled:
This objective has not been fulfilled by the implementation of the system. Largely due to the method used to authorize the system and due to time constraints this has not come about in the implementation.

1.2.4 Easy to navigate interface

Objective:
The interface must be easy to navigate and allow for ease of use

This Objective has been met.

Fulfilled:
This objective has been fulfilled, but the extent of this claim shall be expanded upon in the next section (§2.2.4).

Evidence:
In §7 it is shown that the client was able to use the interface, and it passed many user interface tests in the Testing section (pp.62-99).

1.2.5 Interacting with the API

Objective:
Tweets must be received from the API

This objective was split into two separate parts on the Analysis section, however only the second of the two is relevant and thus has been simply expressed by the umbrella title.

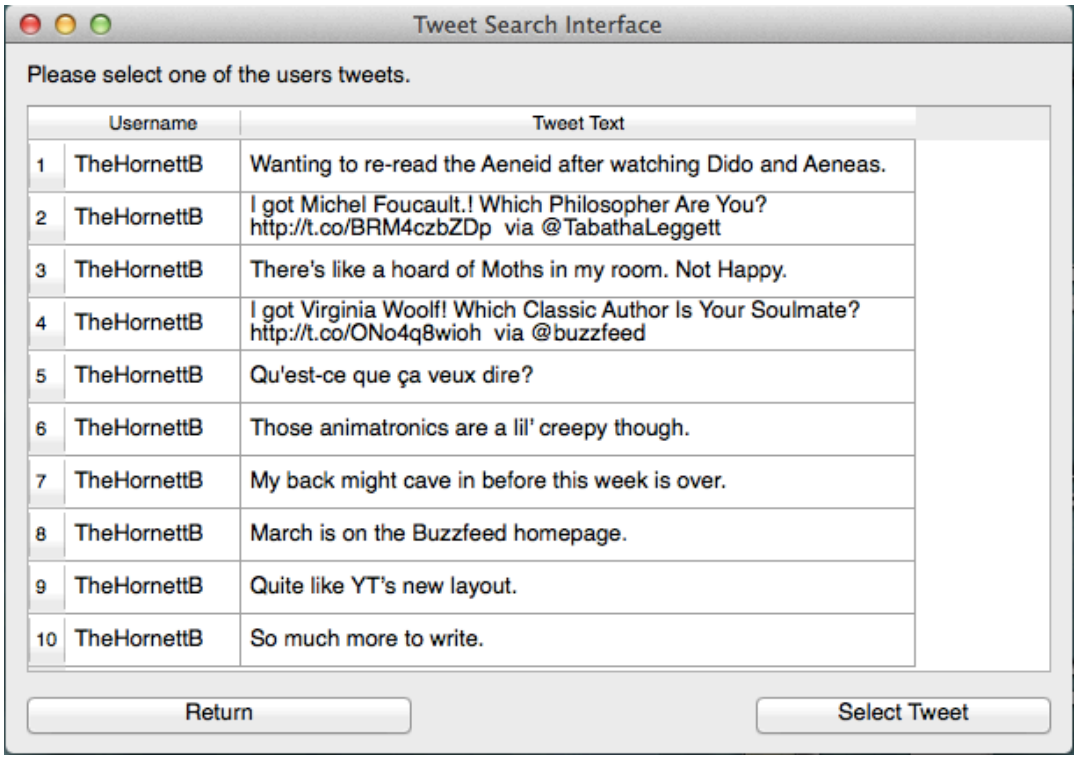
This Objective has been met.

Fulfilled:

This objective has been successfully fulfilled. The user, by using both the CLI and GUI versions of the program is able to see some of the latest tweets that a user has posted, ready to be bookmarked.

Evidence:

For this claim evidence can be found, again, within the Testing section, as there are many tests, which prove that the user can display tweets received from the API; further an image below shall show exactly the display that contains the tweets garnered from the Internet.



This screenshot shows the tweets that have been obtained by the API, thus acting as evidence for the claim that the system can interact with the Twitter API.

1.2.6 Searching Tweets

Objectives:

- The user must be able to search tweets; however they were obtained, using a predefined query from the user
- Results must then be displayed back to the user, through the graphical interface

This Objective has not been met.

Fulfilled:

This objective has not been fulfilled due to changes in the nature of the program, rather than finding tweets from the home timeline, the user searches

specific user's tweets that are added to the database. This change has led to the dismissal of this as an objective of the system.

1.2.7 Parsing the Tweets

Objective:

Information from tweets, such as links to external resources, must be parsed from the tweets

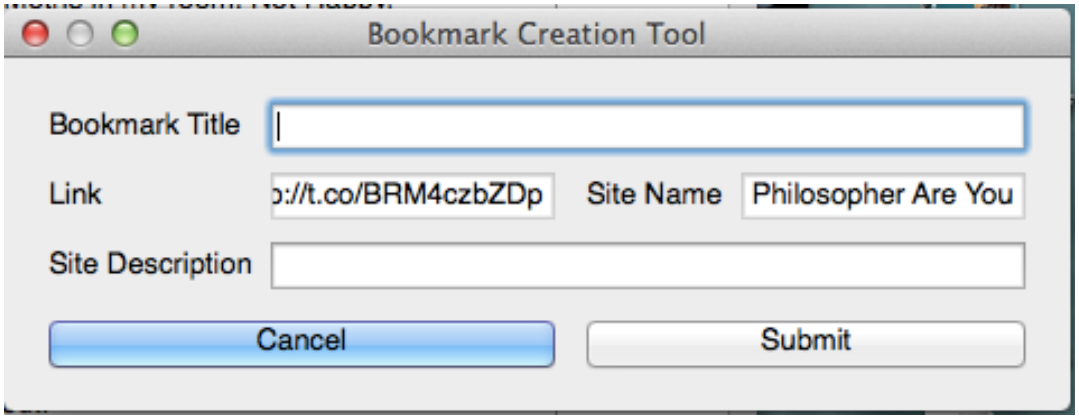
This Objective has been met.

Fulfilled:

This objective has been successfully fulfilled; handled by the bookmarking interface, this displays the link(s) that is stored in the tweets meta-information, and furthermore the user information is also parsed from the meta-information provided by the tweets.

Evidence:

Evidence of the completion of this part of the interface is provided in the Testing section, which has passed tests that prove that the system is capable of doing the defined objective. Below are screenshots of this process.



This screenshot shows the bookmark creation interface in which the Link and Site Name have not been added by the user but obtained from the Tweet and from a web page, respectively. This acts as proof that what is claimed, that the system is able to parse tweets, and in this case web pages also.

1.2.8 Storing to a Database

Objective:

These must then be stored in a database on the computer and must be easy to navigate by the user in order to find these resources.

This Objective has been met.

Fulfilled:

This objective has been satisfied, also done through the bookmarking interface. This allows the user to store all he required information about the tweets and the users in the bookmark database.

Evidence:

Evidence for this can be found in the System Maintenance section in §5, where the use of SQL is elaborated on, also in the Testing section (§1.1, Test series 5.3,pp.62-99) where it has been tested where the use of bookmarking is, and the system has passed these tests. Below are some screenshots of this part of the system in action.

	BookmarkID	BookmarkTitle	SiteName	SiteDescription	Link	TweetID
1	1	New Title	Site Name	New Fin	null	5
2	2	title	Which "Parks …	BuzzFeed	http://t.co/m…	6
3	4	New Title	Don't Hug Me I&#0…	New Site Desc	http://t.co/B3…	13

This is a screenshot from inside the program that displays part of the database table view. It shows the bookmarks created for testing purposes and is evidence of the user being able to add information to the database of the system.

1.2.9 Using Trends

Objective:

The system must allow for the user to specify trends to follow and to find common themes between tweets in these trends

This Objective has not been met

Fulfilled:

This objective has not been fulfilled within the main implementation of the program. This is due to time constraints not allowing this to be implemented; if there was more time then it may have been possible to implement this fully into the program.

1.2.10 Suggested Users

Objective:

The system must be able to get suggested followers from the Twitter API

This Objective has been met

Fulfilled:

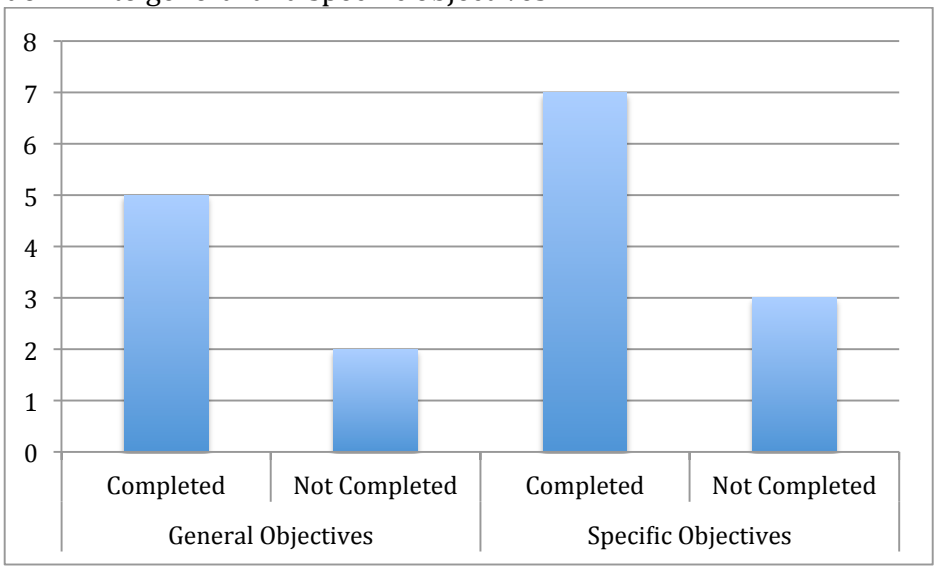
This objective has been successfully fulfilled in the implementation period. It forms part of both the CLI and GUI environments that have been developed for this system. The system allows the user to check from 3 different topics where the suggested users are aggregated.

Evidence:

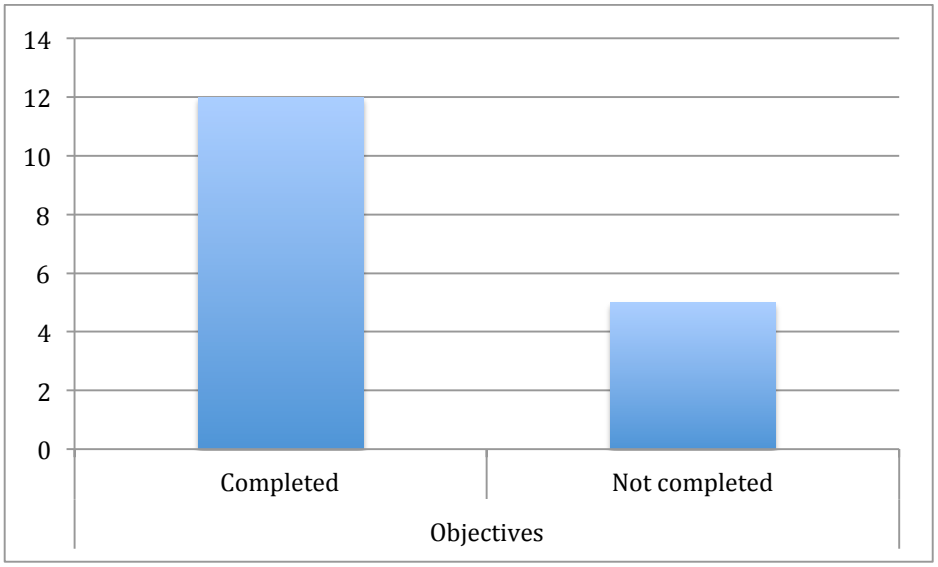
Evidence of this function working can be found in the testing section (§1.1, Test series 5.6, pp62-99), where the system passes tests that require the use of suggested user functions. In the End User Evidence appendix, the client answers Qs 13 & 14, which asks whether the user is able to add suggested users to the database, affirmatively, which reinforces the evidence for this objective being met.

1.3 Tally of Complete Objectives

A Graph showing the number of completed and uncompleted objectives broken down into general and specific objectives



A Graph showing the total of all the completed and uncompleted objectives.


14
12
10
8
6
4
2
0
Completed
Not completed
Objectives

2. Effectiveness

In this section I shall evaluate the effectiveness of all the completed objectives in the system in order to better evaluate how well the implementation has occurred. The names of the objectives can be found in §1 of this section, where the titles of the sub-sections are corresponding. Evidence pertaining to each objective's evaluation shall be included in the *Judgement and Evaluation* sections within each of the sub-sections within this section.

2.1 General Objectives

2.1.1 Logging into Twitter

Evaluation Criteria:

Whether the interface used is intuitive

Whether the end user found it suitable

Judgement and Evaluation

Overall, the effectiveness of this completed objective is at question, although it has been successfully implemented and has no issues of use – the user is able to log into Twitter easily – the method gone about this could have been improved. The interface uses the CLI to get the users PIN number needed to log in; this could have been implemented into the GUI rather than relying on the CLI. The Client in §7 of this document has noted this concern, where in Q2 he responds,

"Possibly have the pin entry in a GUI window"

which demonstrates that the user is aware of the fall back in this part of the implementation. Therefore it can be concluded that this has been done effectively but not to its full potential.

2.1.2 *Easy Interface*

Evaluation Criteria

Whether the user finds the interface easy to understand

Whether the interface's text is legible

Whether how to use the system can be easily intuited

Judgement and Evaluation

This objective has been met to a good standard, but again it could be done more effectively to better facilitate standards within the computing industry, and to better facilitate understanding of interface elements. To back up this claim,

within the End User Feedback in §7 of this section in Q3 the client responds to a question pertaining to first sight of the interface, in which he responds:

> *"[Interface] is quite sparse – unsure what the search box is for"*

which clearly shows that the user found the interface as it is in the implementation as being not up to an easy to understand standard and thus the interface should be updated to reflect this; this could be done by more explanations within the system and a better use of images to create metaphor rather than relying on a text heavy interface. In evaluation, this part of the system, although met, could have been done much more effectively than it is.

Regarding the legibility of the text of the system (which shall be further elaborated upon in §4) the system on a whole is largely legible and provides text of a standard size coupled with high contrast layouts, evidence of these claims can be found in §4 of this section, under 'Readability.' The interface, however, is not the easiest to intuit the use of; due to the use of text rather than metaphor, and a lack of explanation within the interfaces of what each button and command does, there is a significant lack of easily understandable elements. Better use should have been made metaphor and images rather than a text heavy system.

2.1.3 Getting Tweets from the API

Evaluation Criteria

Whether the user could successfully do the operation

The speed of the operation

Judgement and Evaluation

This objective has been completed rather effectively, and without too much room for improvement. The user was successfully able to complete the operation, evident by the client's feedback in §7, where in Q6 he replies affirmatively as to whether he was able to display user tweets. Furthermore, this operation takes no noticeable amount of time to complete, as it does not require the downloading of large files across the Internet, but rather kilobytes of text, the time taken for the operation to complete is negligible. And so, conclusively, it can be said that this has been completed rather effectively

2.1.4 Storing Tweet Information

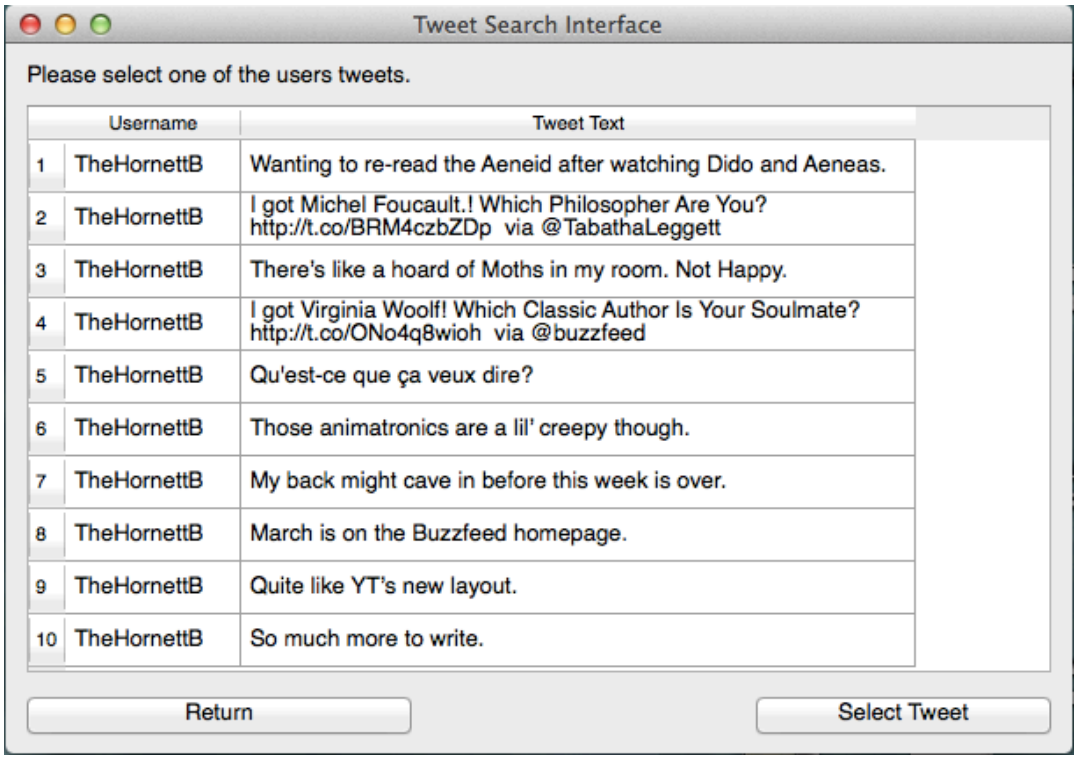
Evaluation Criteria

Whether the user was able to do the process

Whether the user found the implementation satisfactory to their needs.

Judgement and Evaluation

This objective has been completed rather successfully within the implementation system, still with room for improvement, but on the whole very good. Addressing the first of the criteria, the end user feedback in §7 was affirmative regarding whether the user was able to add the tweets to the database, evident from question 8. The user also gave feedback regarding the improvement of this part of the interface, wishing that the Table Views within the program were more interactive than they currently are; the client responds: *"Should make use of Table Selection"*. Thus it can be concluded that this could have been implemented much better than it is in its current state.



This image shows the table view used in the system. The items within this table are not interactive in any way and the user must use the push button in the bottom right corner in order to interact with the system, one of the criticisms of the system.

2.1.5 Suggested Users

Evaluation Criteria

Whether the user is able to complete the action

Whether the user found the interface to their standard

Judgement and Evaluation

This objective has been met satisfactorily, again with much room for improvement in how the system goes about performing the functions. This objective has been met, as the user is able to do the action supposed in the

objective title, however there is still the pertinent issue of the interactivity of the table views in the interface, a concern of the client reiterated in Q15 of §7 of this section, where the user says: *"[...] selection of items for Table View"* which is a reference to the table views in the interface not being incredibly interactive (apart from the delete and modify interfaces, to an extent.)

2.2 Specific Objectives

2.2.1 Login Credentials

Evaluation Criteria

Whether there is an effective way of entering one's own credentials

Whether the user found the method suitable

Judgement and Evaluation

This part of the system could have been implemented in a better state than as it currently is in the system. Although the user is able to do the action quite easily, as the client responds in §7 – the user feedback – that he has been able to login to Twitter with his own credentials in question 1, the system could have handled the retrieval of the information from the user in a way more consistent with the design and interface of the overall system rather than relying on a CLI, an interface foreign to most users (although, to the client, not so.) The client, in the feedback, did also propose changes to the system that might aid in facilitating the effectiveness of this system, such as making the PIN entering screen part of the GUI, paraphrased from Q2 of the feedback. Therefore this part of system could have been developed and implemented in a more effective manner, based on the evaluation criteria.

2.2.2 Storing the user credentials

Evaluation Criteria

Whether the user credentials are stored

Whether this is done securely

Judgement and Evaluation

This part of the system has been implemented rather effectively, based on the chosen evaluation criteria. The system successfully stores the user credentials, as the user, in the feedback, reports being able to use the system, which is proof of the credentials being successfully stored; furthermore the system passed Test Series 2.4 in the testing section (pp.62-99) and therefore proves further that the system is able to stored the user credentials effectively.

The system also stores the user credentials securely, firstly, as the user's username and password are not stored on the computer but their OAuth tokens, there is a reduced chance of someone being able to find the users credentials.

Secondly, the users tokens are stored using a function that comes with Python Twitter Tools called OAuth dance, which stores the users tokens on a file in the computer in such a way that is not human readable. Lastly the credentials are stored in a hidden file in the computer, which is not accessible by default in the main file manager, and therefore reduces again the chance that someone would be able to read the users credentials. This is elaborated upon in the System Maintenance section (pp.99-172), which acts as evidence for this claim.

It can therefore be concluded that the system fulfils this objective to a high standard and therefore completes this objective effectively.

2.2.3 Easy to navigate interface

Evaluation Criteria

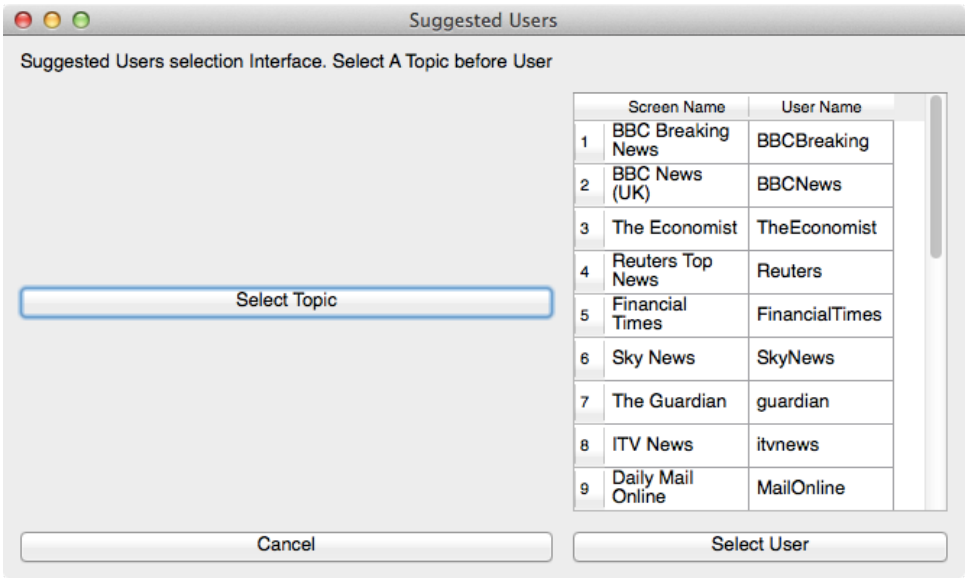
Whether the client found the interface navigable

Whether the interface is easily learnt

Judgement and Evaluation

Much of the evaluation of this specific objective can be found in §2.1.2 of this section, which explains how easy the interface is to use and whether it meets the evaluation criteria of that section, along with evidence to back up those claims. It has been concluded, in that section, that the interface could have had more measures to facilitate easy use, based on feedback from the user that criticises some of the less obvious aspects of the system; this impedes on the learnability of the system and therefore it can be said that the program, though not overly difficult to learn, has some ambiguities that could have been better handled. Therefore it can be concluded from the previous section, referenced above, that this could have been implemented more effectively than it has been.

As the above Screenshot shows, much of the system is navigated through the menu bars at the top of the interface; this allows a simple and universal way to navigate all aspects of the system in a quick and easy



The above images shows a screenshot of one of the interfaces in the system, it has instructions on the top of the interface and each button has a clear label telling the user what each does, this aids the learnability of the system.

2.2.4 Interacting with the API

Evaluation Criteria

Whether the user is able to interact with the API

Whether there are pertinent latency issues

Whether the user is satisfied with the interaction

Judgement and Evaluation

The effectiveness of this part of the implementation, in relation to the objective stated, is on the whole not bad. The user can successfully interact with the Twitter API, allowing them to view tweets from users from the whole of twitter, after they have been added to the database containing users; evidence of this claim is available from the user feedback in §7, where the client responds affirmatively to a question regarding whether he was able to retrieve tweets from the API. There are also no noticeable latency issues regarding this part of the system, as demonstrated in the testing of the system, and by lack of comment on this from the client, which is assumed to be from a lack of latency issues. Regarding the interaction and the users satisfaction with this part of the system, there is possible improvement. In the questionnaire the client responds asking for more interactivity with the graphical elements within the interface, especially for clearer toolbars. Therefore, mostly an effective implementation, but with room for improvement regarding the interface elements of the system.

2.2.5 Parsing the tweets

Evaluation Criteria

Whether the user is able to complete the action

Whether the user was satisfied with the action

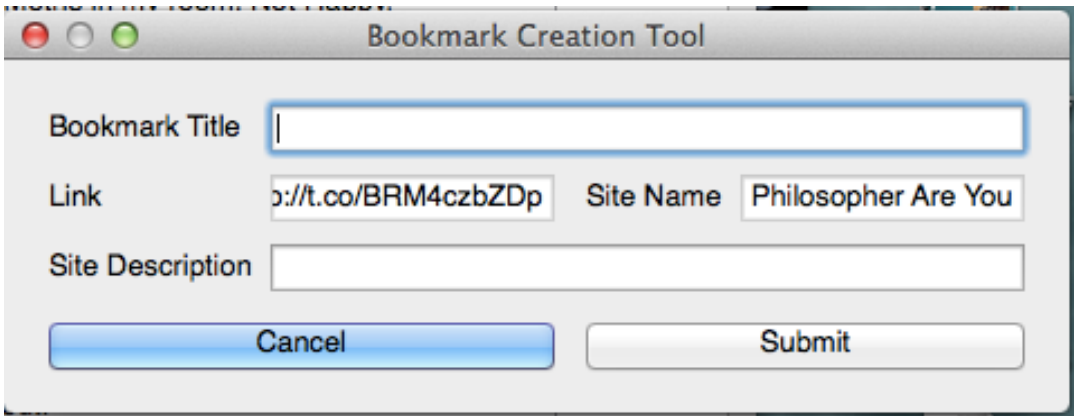
Whether the information is clearly displayed

Judgement and Evaluation

This objective has been completed very effectively in the implementation of the system and has allowed for the user to easily retrieve and display the information that is part of the tweet and of links associated with the tweet. The system handles this in the user tweets interface and further in the Bookmark Creation Tool, which displays link and web page information as well as the information from the tweets, such as the username of the tweet's author.

The action is completed without hassle, as proven by the Testing Section, which in Test Series 5.3 passes the operation easily. Furthermore in the user feedback in §7 of this section, where in Q8 the client responds affirmatively to questions regarding the completion of the action. Here criticism of the interface arises, in that the Table Views are not interactive, a commonalty throughout the system, except for the interfaces that handle the modification of tweets. This acts as evidence of the effectiveness and improvements needed to be made regarding the parsing and displaying of tweets in the system

Regarding whether the system displays the information gained effectively is another matter that has to be consulted separately. The system uses line edits, (which are described in more detail in the System Maintenance section – pp.99-172 – in §3&7) which displays the information as plain text. This is done sufficiently and clearly, but many not have been the best way to do the action, this is due to the edits not automatically resize and so it can obscure the information and not make it clear what is there. This is an issue that can be, and ought to be fixed. Evidence of this claim is below in an annotated screenshot.



What this screenshot shows is the Bookmark Creation Tool with the generated Link and Site Name, this shows the obscured text in both of the line edits; this can be fixed by the user manually, but should be done automatically in order to facilitate understanding of what the system is

2.2.6 Storing to a Database

Evaluation Criteria

Whether the user can store information in the database

Whether the database is normalised

Whether there is security against duplicate information

Judgement and Evaluation

This objective has been completed rather effectively, with database management being on the whole fluid and easy to use. The database has been completely normalised, as it has been shown in the Design section (pp.28-56) in which, in §4, there are ER Diagrams and UNF to 3NF tables demonstrating the normalisation of the database, which keeps consistency between the many tables in the database.

Regarding the user being able to store information within the database, this is also possible, as demonstrated in the Testing section (pp.62-99) in Test Series 5.3, in which tests are done to check whether adding data to the database works or not, in which the tests are passed without error.

Lastly regarding duplication of data unnecessarily; this is a more contentious issues as its conclusion differs based on what is looked at, for users added to the database there are clear functions which do not allow for the user to add a user that already exists in the database, this is proven through tests in the Testing section in test series 5.1, where the test is passed. However, regarding Tweets, these can be added multiple times when the user creates multiple bookmarks. The tweets table is less of an issue, as the user never directly engages with the table while using the program but the user can make duplicate bookmarks, this is a problem with the system and highlights a lack of effectiveness in the implementation stage.

In conclusion, on the whole this has been completed very effectively, however there are some drawbacks in the way it has been implemented that call for better ways of handling the duplication of data within the system.

2.2.7 Suggested Users

Evaluation Criteria

Whether the user is able to add suggested users

Whether the interface is suitable for the user

Whether the interface is easily intuited

Judgement and Evaluation

This part of the system has been effectively implemented. The user is able to effectively add users to the database through the suggested user dialog of the system, evident by the end user's responses in the feedback in §7, where in Q14 the user confirms that he was able to add suggested users effectively, with no criticism as is provided with other question. It can therefore be inferred that the user also found the interface not too inhibiting to use, although the same criticism of the table views not being interactive.

The interface contains on it more instructions than the other interfaces and therefore can be said to be more easily intuited than other aspects of the system, however the same issues remain as to the use of metaphor rather than plain text and other decisions made in the implementation that could have been different to make the system more intuitive than it is in its current state.

2.3 Summary

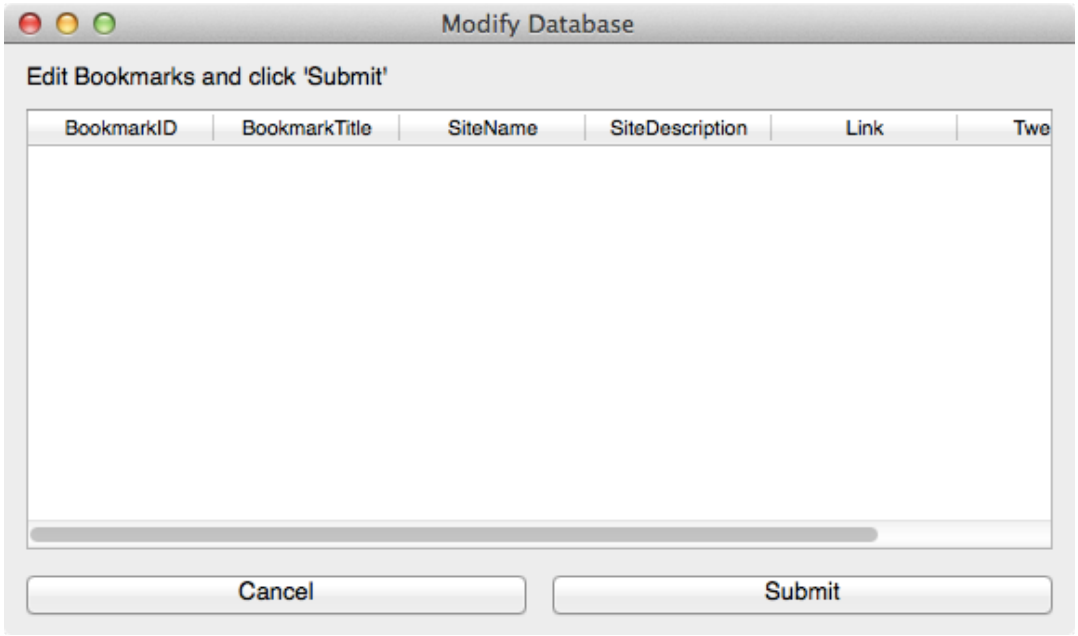
Overall the system has been done quite effectively, many of the aspects of the system have been implemented in such as way that is usable by many that already have a working knowledge of familiar computer systems that use basic GUI elements. There is, however, much criticism of the GUI and the many choices made in the implementation stage that could have been different in order to make the system easier to use at first glance without having to resort to another in order to understand what to do next. Many of the tables within the system are not interactive, which if they were would have facilitated the use of the system, and there are lots of ambiguities regarding what to do next with the system as it not immediately obvious how to go about the next operation the user wishes to do, again without consorting the user manual or some other external authority on the program. Still despite these criticisms the program is very workable and, although there is room for definite improvement, the implementation has been rather effective in achieving the aims of each of the objectives created during the analysis stage of the project.

3. Learnability

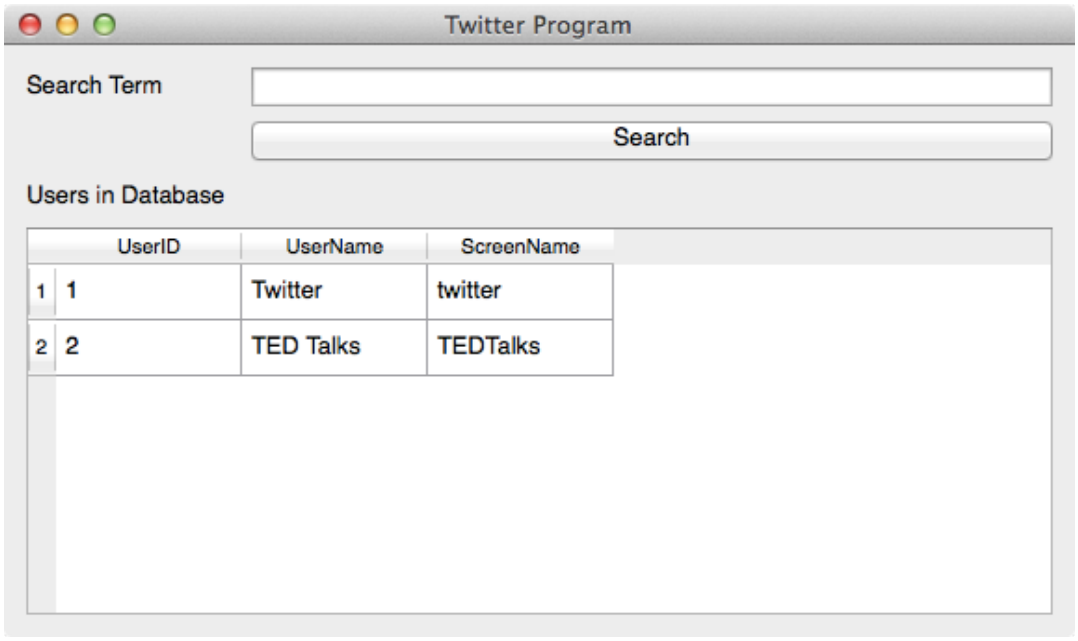
Overall the learnability of the system is quite low and not so difficult as to deny people the easily intuit the system's controls. There have been some measures to increase the usability and the learnability of the system, but there is work that has to be done to better make the interface easier to use.

All the buttons in the interface have clear and concise names that aid the user understanding what each button does on the interface. For example the buttons that cancel or stop an operation are all called 'Cancel' and when the user wishes to commit changes in the database the modify button is called 'Modify'. These help the user intuit the meaning of many of the buttons in the interface and also reduce the learning curve of the system. However it can also be

ambiguous within the system what some buttons do because of bad design decisions and some ambiguous language that can actually increase the learning curve of the system. For example on the main interface shown to the user at the beginning of the system there is a text edit with the label 'Search Term'; this is ambiguous and lacks clarity and does not therefore aid the user in understanding what the interface is meant to do. The conclusion of this point is that on the whole the interface is learnable but there are some minor difficulties in the program.



The above images shows the modify interface that has push buttons that have clear titles that aid the learnability of the system. There are also instructions on the top of the interface that tell the user what they have to do, thus aiding the understanding of the system.



The image above shows the initial interface with the ambiguous terminology used which detracts from the users learning of the system and creates difficulties regarding understanding what to do next.

These features make the system sometimes quite difficult to learn and there are some steps the user will have to take to get a complete understanding of the system, such as reading a user manual or by consulting another on how to use the system. Overall however the system is not so difficult to grasp to render it unusable and has therefore, to an extent, been done reasonably effectively.

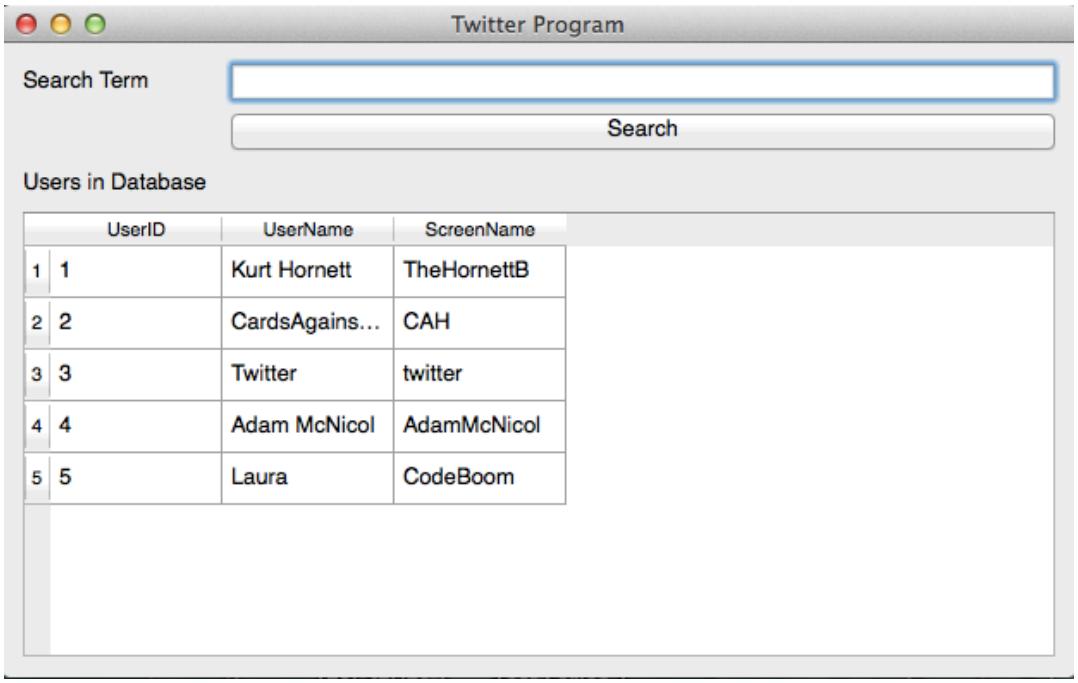
4. Usability

4.1 Target Acquisition Time

This is a measure which determines how fast the user is able to commit an action within the system, based on factors such as the relative size of the button used on the interface and whether the icons and text are clearly visible to the user.

Overall this system is relatively fast to use, in regard to the menu bars used in the application, accessing the items should be similar to the average time used on a Mac as the universal menu bars are utilised within the application. Despite this it can be said that many of the interface buttons on the interface are rather small compared to some other systems, also that there is some ambiguity in the use of names for the windows, for the buttons, and for the menu items that all reduce the time that could be spent using the program. Many more instructions within the interface could have been utilised to make this less of an issue, for example in the initial interface of the program it could have been that the search bar was labelled with 'User Name' rather than 'Search Term' as this would reduce the ambiguity of the interface; furthermore the use of more labels to instruct the user on what each button or function does could have been used, such as tooltips for many the interface elements.

Target acquisition time in the system is probably just below average and is one of the weaknesses of the system developed, however on the whole the interface is rather fluid and poses no obvious difficulties that may arise from using an unfamiliar interface.



The above image is the main interface that is displayed to the user directly after the application is authorised and authenticated. The Target Acquisition Time of this interface is rather low, and much slower than other parts of the system. There are almost no clues on this part of the interface as to what each button does and each thing can do, and it relies on the use of the menu bar on the Mac OS to navigate to other parts of the system, which is a rather slow way of doing this when the use of buttons on the interface of a larger size

4.2 Latency

This section evaluates not the time needed for the user to acquire the knowledge to use the interface but the time necessary for the actions within the system to occur such as pushing a button or by executing an SQL statement.

For many of the push buttons on the interface it is immediately obvious when a background action is occurring, as for many of these GUI elements, when clicked often are highlighted blue, and often stay highlighted until the action has finished. And for most actions that do not require an SQL statement or access of the Twitter API these actions take fractional amount of time to complete and have no noticeable waiting times.

However, when an action takes longer, especially when the application is accessing the Internet there is a longer, more noticeable wait that occurs. It is obvious that an action is occurring for a longer amount of time as on the Mac version of the program that highlighted buttons stay highlighted for the duration of the task. Thus it is visible that there is an action going on. However, a side effect of this is an unresponsive program that may seem a little alienating to the user, still the wait often lasts not longer than a few seconds and the program does not crash when this freeze occurs.
When doing an action that requires an SQL statement to be done, there is no obvious lag in the system; it is a rather smooth process that poses no slowing down in the programs running time. When an action that requires SQL to be used it is often handled again by the push buttons on the interface that stay highlighted when an action occurs.

Throughout the system there are no messages or sign that display what the length of a wait may be. This can be seen as an issue in the program, which needs to be addressed, as it can be quite ambiguous as to what exactly is happening in the program. This is a weakness in the program, but on the while latency is not a massive issue in the program.

4.3 Readability

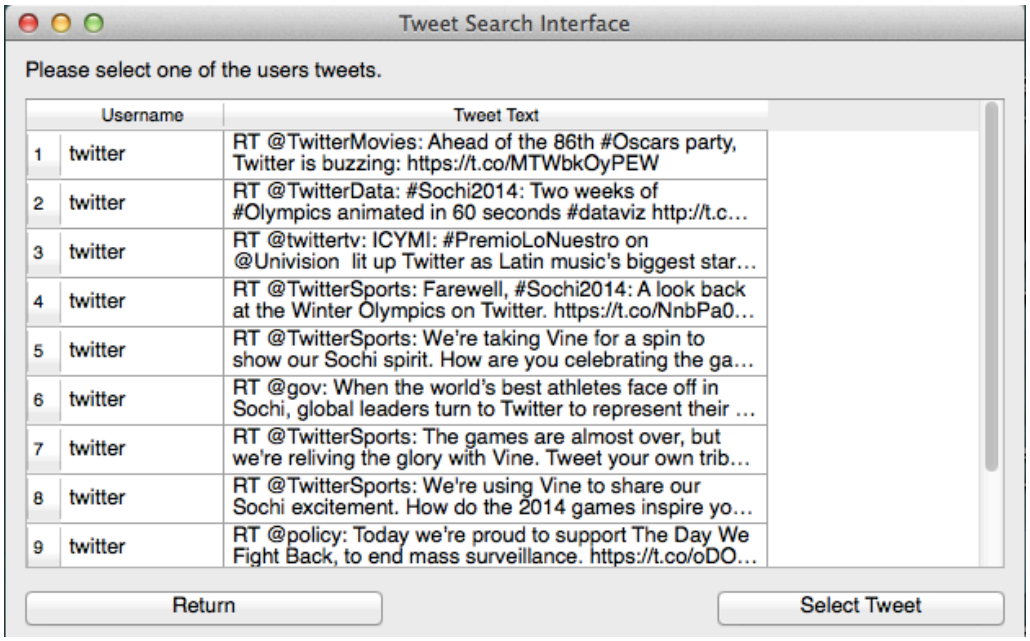
Throughout the program many of the interfaces and graphical elements are of the main colour of the system, when using a Mac this is a silver colour with all or most of the text being a standard black. Most of the text therefore is of a high contrast and can be read easily by anybody who decides to use the system. This enhances the overall readability of the system and makes for a smooth process.

All the font sizes on the system are rather small, and could have been enlarged to such an extent that makes them clearer and slightly more readable. When error messages are displayed the font becomes bold, which enforces the immediacy of an error and makes clear that attention needs to be paid to the message.

Most of the time when numbers are displayed in the system, they are with the same font and style of that of regular text. This means that they are generally very readable and understandable to the eye of the user.

Overall the system remains very readable, some of the fonts could be larger to aid readability, but the system, as it is, is not illegible and could be read by most people with ease.

Below are some screenshots of the program, which have some annotations of what is shown on each screenshot pertaining to the readability of the system:



This screenshot shows the Tweet Search Interface and is a good example of the readability of the text in the system as a whole.

1) The text, especially within the Table is of very high contrast; the black text is on a white background, which aids user readability greatly.
2) The font size is also not so small it is unreadable, also not too large it is obtrusive or 'in the way'
3) The numbers used within the table are also clearly readable, and the numbers are separate and distinguishable from the rest of the text, allowing the user to clearly see what each number is.

4.4 Use of Metaphor

Because the system does not make use of many graphic representations and relies heavily on text to bring across the message of what each graphical element does, it is not of much relevance how metaphor has been used in the system. This does however highlight the possibility of using more graphics and metaphors within the system as this is a common trait of many programs and helps aid the user by being clearer about what each object does.

4.5 Navigability

Navigating the system is largely completely handled by the universal menu bars provided by the Mac Operating System, which means that the user will have a familiar way of navigating the system as almost all application on a Mac system use the same menu bars for navigating through the majority of features within the system. This relies not on metaphoric images but relies rather on good use of unambiguous text titles for each of the functions each button does. This is done effectively in the system as the titles are largely clear and denote a key aspect of the system and what each menu action does, for example under the 'Database Management' tab the user will only find functions that relate to that task, namely the 'modify' and 'delete' functions of the system. This allows for a very easily navigated system.

The image above show the use of the menu system that is part of the Mac operating system that is used by many other interfaces in the majority of applications for the Mac system.

Nonetheless there is some criticism to this approach; by using this system I have not used many controls within the environment of the program itself, there aren't very many images or buttons that allow access to these function, such as use of toolbars within the window. By using some of these feature instead navigating the system might have been much easier and needed much less time to find the correct element to navigate to. Therefore there are some key improvements that can be made to this part of the system.

5. Maintainability

In this part of the evaluation I shall discuss how maintainable the system is.

5.1 Fixing Bugs

Fixing bugs that arise in the system, despite the lack of many in the testing section, should not be such an issue to the programmer who seeks to fix any of

these bugs. Should ever a need for fixing a bug in the system appear, due to these reasons it should not be an issue for the next programmer to fix them:

- The code of the system is largely self-documenting, insofar as suitable variable names have been used throughout the program to clearly identify what each variable does, using abstract names, such as ‘a’ or ‘b’, for any mathematical function has been avoided throughout the implementation so as not to come across any difficulty in understanding the program.
    - An example of this is can be found in §4 of the System Maintenance section (pp.99-172) that contains the Variable Listing, which has many of the systems variables and demonstrates what each variable does. For example, the variable *self.bookmarkList* by its name obviously points to a list containing a bookmark.
- Commenting has been used extensively throughout the code. Wherever there is a more difficult section of code to understand compared to the rest of the code, there have been extensive line-by-line comments that explain succinctly what exactly is going on with the code. This allows for a greater comprehension of the code so that whoever comes to program for the system further knows what each variable does and how this relates to the whole program.
    - Evidence of this claim can be found in §7.1.2 in the System Maintenance (pp.99-172) which shows the use of regular expressions within the system, the code is shown below to demonstrate the use of commenting in the system.

```
138.        regex = re.compile('<title>(.*?)</title>')
139.        url = urllib.request.urlopen(link) #Gets the data from the url
140.        html = url.read() #Converts HTTPRequest into text
141.        html = str(html) #Changes from 'bytes' to str so re will work
142.        self.siteName = regex.search(html).group(1)
```

- The program avoids the use of any global variables throughout; it has been kept in mind while implementing the system to keep the use of global variables to a minimum and to maximize the use of local variable, so that when an error does occur it does not affect much of the system beyond the error. This will help the next developer of the system to maintain it in relation to bugs.
- The modular structure of the code, especially in the GUI implementation, means that functions and methods are largely self-contained and therefore bugs are very localized; this helps find bugs quickly and bring an end to bugs quickly as it would not take much time to find where an error was occurring.
    - Evidence concerning this claim can be found in the system maintenance section (pp-99-172) in §3&7 in which the structure of the code is expounded upon and elaborated, explaining how the modular structure was achieved and how it has effected the system and its management.

5.2 Changing Parameters

Changing the parameter of the system may be necessary at some point in the further development of the program. This has been facilitated to an extent as

well, to make it as easy as possible for whoever next wishes to develop the system further.

- The use of local variable, and not global ones, as stated before, help make changing variable very easy, without major disruptions in the rest of the program. For example, in the system maintenance section (pp.99-172) a section explaining how tweets are displayed is shown to use a stepper variable; this variable is a local one that can be easily changed to manipulate the amount of times the function runs.
- The data dictionaries in the documentation, especially in the system maintenance section allow for a good understanding of the variables in the system and therefore allow the developer to know exactly what each variable does in the system and how a change in the parameter might do to the system as a whole and not just as a local experiment.

5.3 Responding to new Requirements

Should ever the need to expand the system arise there have been many precautions made in order to facilitate any expansion to the functionality of the program. Adding features such as the 'Trends' section that didn't make it to the final product can be easily implemented to the GUI version of the system.

- One way this has been facilitated is the use of a modular code system that allows for the greatest amount of expansion possible; because the system is structured as such it allows for expansion by simply adding new modules to the system, for example with the MainWindow class, any new feature can be easily implemented into the system.
    - Evidence of this claim can be again found within the Maintenance section (pp99-172), which in §3.2.2 explains how the use of the Main Window class is used to create a modular structure to the code, and how by importing modules errors can be localised.
- Again the localisation of many of the variables allows for a great amount of flexibility in adding new units to the program, the limited use of global variables means that major changes to the system do not necessarily entail a major change to the functioning of the program, compared to an extensive use of global variables.

5.4 Conclusion

Overall, it can be concluded, from this demonstration, that the code of the system I have developed is very maintainable, the use of a modular code structure that relies more on local variable as opposed to global variables, variables that are on the whole self-documenting and the structure within the system that allow for a great deal of flexibility, it can be said that the system is very maintainable and ready for any due expansion to the system.

6. Suggestions for Improvements

6.1 Elements of the Search User Interface

This stems from criticism, found in the Evidence Appendix, of the main interface of the program, displayed when it is started; the client says '*unsure what the search box is for.*' Some of the elements on this interface are not necessarily clear at first sight. For example the Line edit (text box) has the label 'Search Term' but it is not clear from a new user's standpoint that these are for bringing users from Twitter to the database. More notes and instructions could be used on the interface to make it more clear what exactly what each element of the program does, without necessarily having to refer to the User Manual. Furthermore the names of the grouping on the menu bar could be more explanatory as it may not be clear what is where without a look at the User Manual.

6.2 Selecting Users for Tweets

This is a criticism of the way in which users are selected for their tweets to be displayed. The client, in the evidence appendix, says *'[...] should make use of table section,'* in reference to this point. The system as it has been developed requires the user to switch to a separate interface, in which s/he has to access a menu that contains a list of the users in the database, which can be clicked to gain the users tweets. It may have been more sensible, and made the process much easier, according to the clients responses on the questionnaire, to have the table in the initial interface play this role, by being interactive by allowing the clicking of it elements to display and to produce the results required; for example, by clicking the username of a user in the database to display the tweets, rather than the long process that is currently used.

6.3 The Bookmark Creation Tool

There is some minor criticism of this part of the interface, which leads to some improvements to be made in the system. Currently when a bookmark is added to the system the interface does not exit, nor does the information in the table disappear. This is something of a problem as it can be quite confusing what is meant to happen. It should be the case that the window closes and removes the data already stored in the interface.

6.4 Finishing Unfinished Objectives

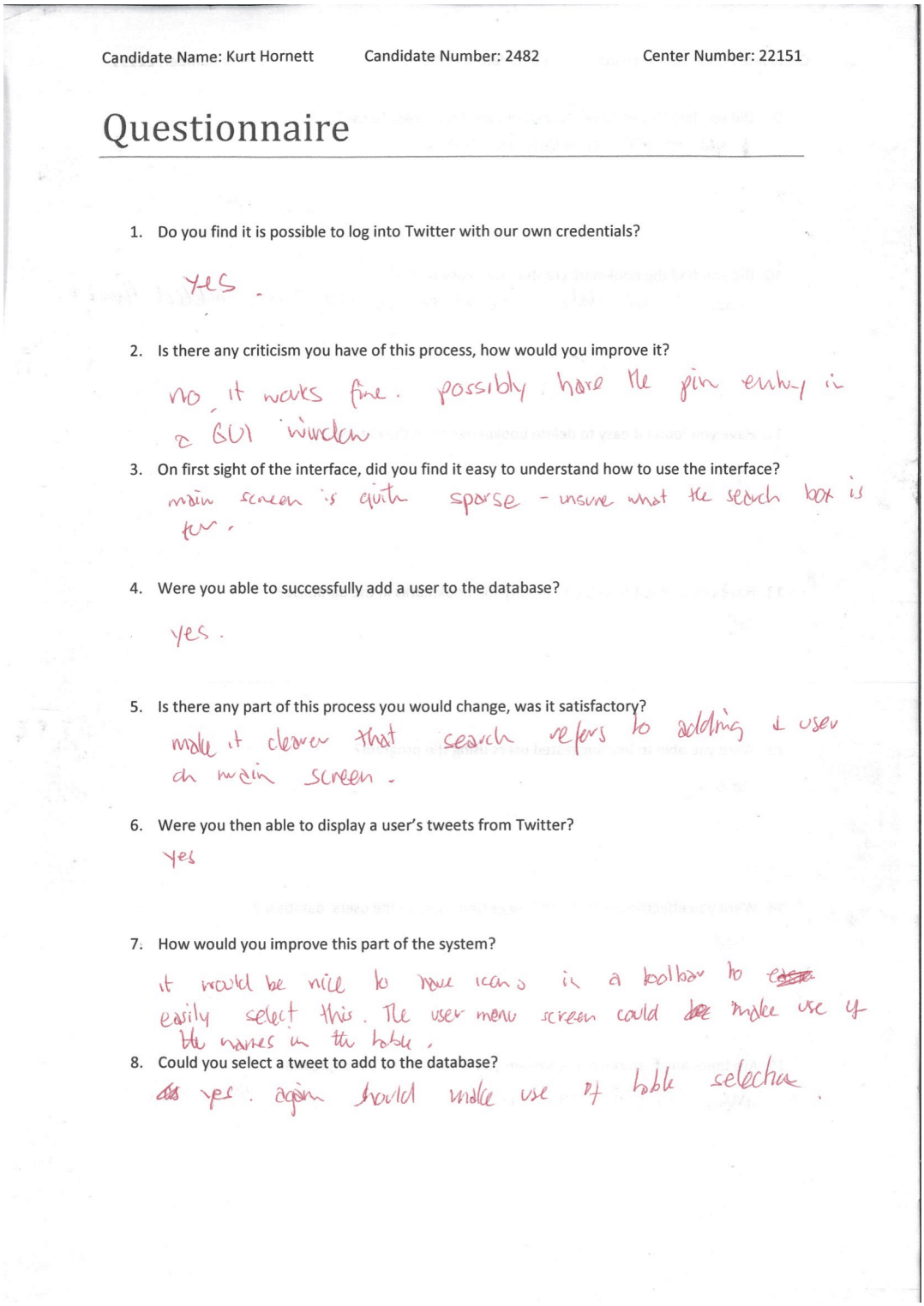
Another aspect that could be improved is the completion of many of the tasks that were not completed in the initial implementation of the program. Specifically regarding the whole 'Trends' section that was not completed before the end of the implementation period. Including these elements would have made for a much more well-rounded system that stuck more to what the client requested, which is developed in the Analysis section (pp7-28).

6.5 Adding Extra Features

Another feature that could have been in the system is the use of features included in the Analysis section (pp7-28, §3.4) that include a few ideas for a second version of the program. Features such as more use of graphics to display information about tweets and more management functions that directly relate to

twitter; furthermore a multiple user system could have been put in place, both in relation to the local databases and to the Twitter Authorization.

7. End User Evidence Appendix



Candidate Name: Kurt Hornett Candidate Number: 2482 Center Number: 22151

Questionnaire

1. Do you find it is possible to log into Twitter with our own credentials?

   yes.

2. Is there any criticism you have of this process, how would you improve it?

   no it works fine. possibly have the pin entry in a GUI window

3. On first sight of the interface, did you find it easy to understand how to use the interface?

   main screen is quite sparse - unsure what the search box is for.

4. Were you able to successfully add a user to the database?

   yes.

5. Is there any part of this process you would change, was it satisfactory?

   make it clearer that search refers to adding a user on main screen.

6. Were you then able to display a user's tweets from Twitter?

   yes

7. How would you improve this part of the system?

   it would be nice to have icons in a toolbar to ~~easy~~ easily select this. The user menu screen could ~~be~~ make use of the names in the table.

8. Could you select a tweet to add to the database?

   ~~yes~~ yes. again should make use of table selection.

Candidate Name: Kurt Hornett Candidate Number: 2842 Center number: 22151

9. Did you find this an easy process; did you find it easy to use?

yes – see comments in Qs.

10. Did you find the bookmark creation tool easy to use?

- yes. current data should be cleared when added though.

11. Have you found it easy to delete bookmarks from the database?

yes.

12. Have you found it possible to modify the bookmarks in the database?

yes.

13. Were you able to find suggested users using the program?

yes.

14. Were you effectively able to add suggested users to the users' database?

yes.

15. Are there any features of the system you would like to see improved?

Main thing is the selection of items for table view.

Candidate Name: Kurt Hornett Candidate Number: 2482 Center Number: 22151

I confirm that these answers are my own.

Name (Print)

Adam McNicol

Signature

Date

1/4/14