

Virtual Assistants

Sam Palmer

Contents

Analysis.....	4
Problem definition	4
Background.....	4
Users.....	7
Objectives	9
Diagrams.....	11
Research	11
Design.....	12
Technology	12
Algorithms	12
Data structures	12
File and class structure	12
Assistant.txt	13
Csvworker.py	13
Encryption.py.....	14
Main.py	14
Notesreminders.py.....	14
Settings.py	16
Twitter.py	16
Userdata.db	17
Weather.py	17
API queries.....	18
Weather	18
Twitter	19
Libraries and packages.....	19
Database design.....	20
Notes table.....	20
Reminders table	20
UserInfo table.....	20
Queries.....	20
Screens	22
Setup screen	22

Home screen	23
Weather screen	23
Twitter screen	23
Notes screen	23
Reminders screen.....	23
Settings screen.....	24
Research	24
Technical Solution	25
Assistant.txt.....	25
Csvworker.py	33
Encryption.py.....	34
Main.py	35
NotesReminders.py	48
Settings.py.....	50
Twitter.py	52
Weather.py	52
Testing.....	56
Test strategy.....	56
Test plan	58
Test evidence.....	63
Evaluation.....	65
Objective evaluation	65
Objective 1: "The app should be easy to use"	65
Objective 2: "The app should show information when it is relevant"	65
Objective 3: "The app should integrate with a social media"	66
Objective 4: "Data should be kept secure"	66
Objective 5: "The app should run as above on both Android and iOS"	66
Objective 6: "The user should be able to export their data as a CSV file"	66
User feedback	66
Interview	66
Analysis.....	67
Extensions.....	67

Analysis

Problem definition

Virtual assistants are a growing part of today's society, with offerings from Google, Amazon, and Apple dominating the field- serving reminders, weather, and other relevant information. However, these services are impersonal, inaccurate and often feel designed to harvest the user's data.

For my project I have chosen to create a virtual assistant that assists the user with basic daily tasks without feeling corporate and robotic. There are many competing virtual assistants available currently however almost all of them are produced by large companies that monitor user data.

This will improve my technical skills by providing programming practice and will also allow me to expand my skillset. I will have the opportunity to learn more about mobile development, SQL databases, API usage, and encryption.

To research this problem I have spoken to friends and created a survey designed to collect opinions about virtual assistants.

Background

I have investigated an existing popular virtual assistant, Google Assistant. This service is now available on most android devices and a dedicated physical "smart speaker" device is available to purchase. Google Assistant can be activated by saying "OK Google" but can also be woken by opening the app or tapping the top of the smart speaker. Questions can be typed or spoken on Android devices, but can only be spoken on the smart speaker.

I have personally used Google Assistant daily for over six months and have found that it offers weather information, travel times, news and important reminders. It is incredibly convenient to have this information always available in one place, however the app does have some issues where the information is not always displayed or is displayed at the wrong time – for example, the app does not show weather information in the morning.

Besides Google Assistant, two other popular virtual assistants are Siri and Amazon Alexa. Siri, developed by Apple, has been included with all iPhones since the release of the iPhone 4S in 2011. It has since expanded to be included on all modern Apple devices, including a new dedicated smart speaker, the HomePod. Siri is an example of one of the earliest mainstream virtual assistants, with competitor Google Now (now Google Assistant) released in 2012, and Alexa being released in 2014.

All three of these virtual assistants can search the web for basic information such as weather and can set reminders and calendar events. They also all make use of voice recognition for user queries and will read answers using text to speech. Both Google and Amazon claim that their assistants can distinguish between individual voices to provide a more personalised experience, however an article

by Cnet reveals that this feature is easily fooled, which can pose a security risk especially as they are both capable of making online purchases at the user's request.

Both Siri and Google Assistant are primarily smartphone based, or at the very least are intended to be connected to a smartphone. This means that the app versions of these assistants can control system functions such as volume, navigate to a place from the user's current location, and make and receive calls and texts.

Google Assistant is part of the Google ecosystem so if the user has a connected Gmail account the assistant can detect information in emails such as appointments, flights, events, and shipment tracking. It will also learn from the user's search history, providing information on relevant sports teams, music releases, and other news. This has raised some privacy concerns- although Google's privacy policy claims that the data is never shared with third parties without the user's consent, this data could still be exposed in a data breach or other security failure.

In a more recent innovation, all three of these assistants can control "smart home" devices such as lightbulbs, thermostats, and security systems.

Notably, only Alexa directly supports social media actions, for example posting to Twitter. Workarounds using third party apps are available for Google Assistant.

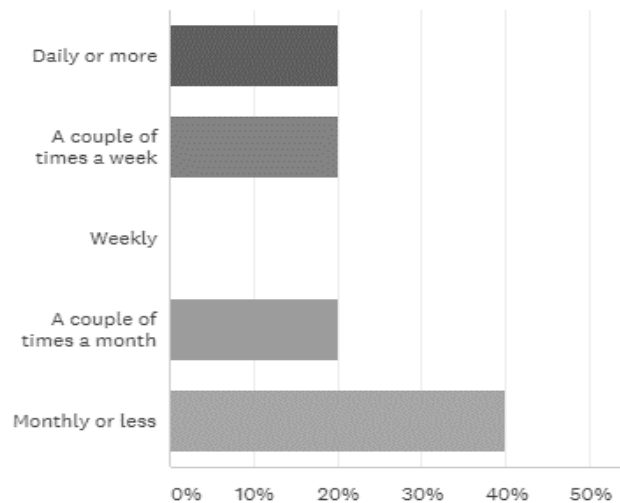
Many of these features are beyond A level standard, such as voice command recognition, so I will not implement these in my program as they would be too time-consuming and complex for me to complete alone. The smart home controls would also be impossible for me to test as I do not have access to any "smart" devices, so I will not implement this.

I have asked people to complete a survey about virtual assistants. The following questions were asked:

- How would you describe your experience with virtual assistants?

2 respondents said "alright", another said "good" and another said "some useful (Google, Alexa) - some terrible (S Voice, Cortana)".

- How often do you use virtual assistants?



This shows that while some users use virtual assistants very often, many use them much less frequently.

- What would encourage you to use virtual assistants more often?

Responses included "better voice recognition", "ease of use" and "relevance of results". Another said "It's currently easier to just not use them".

I also asked users to state how strongly they agreed with the following statements:

- Virtual assistants are useful to me
- Virtual assistants are specific to my needs
- Virtual assistants are fun
- Virtual assistants keep my data safe
- Virtual assistants are smart
- Virtual assistants make my life easier

The results were as follows:

	STRONGLY DISAGREE	DISAGREE	NEITHER AGREE NOT DISAGREE	AGREE	STRONGLY AGREE
Virtual assistants are useful to me	0.00% 0	40.00% 2	20.00% 1	40.00% 2	0.00% 0
Virtual assistants are specific to my needs	20.00% 1	20.00% 1	60.00% 3	0.00% 0	0.00% 0
Virtual assistants are fun	0.00% 0	0.00% 0	0.00% 0	80.00% 4	20.00% 1
Virtual assistants keep my data safe	0.00% 0	20.00% 1	80.00% 4	0.00% 0	0.00% 0
Virtual assistants are smart	0.00% 0	40.00% 2	0.00% 0	40.00% 2	20.00% 1
Virtual assistants make my life easier	0.00% 0	40.00% 2	0.00% 0	40.00% 2	20.00% 1

This shows that while many find that virtual assistants are useful, they do not think that their lives are made easier by them or that they are smart. Interestingly, all users said that virtual assistants were fun.

Users

My intended audience is all smartphone users, although the app will be particularly useful to those who frequently need to set reminders and who have a commute.

I interviewed my friend Jake in order to determine what features of a virtual assistant he likes and dislikes. Jake has an iPhone, so has access to Siri, and his family owns a Google Home smart speaker, which he has also used.

s: You're an iPhone user, so you have access to Siri. This could clearly help you with reminders etc., but I've never seen you use it. Why is that?

j: The voice recognition is bad.

s: But you can type your queries.

j: If I type I may as well just use the reminders app.

s: Did you think the Google home is any better? That recognises our questions quite well.

j: Yeah, I would use that if I had it for myself.

s: Are you familiar with the Google assistant setup? if you swipe left on the home screen it displays cards with weather, commute time, news, and reminders and relevant times.

j: I've seen that. It's pretty decent actually.

s: If Siri did the same stuff would you use it even without voice recognition? If everything was in the same place like that?

j: Probably, yeah.

s: Because it's a Google service it automatically connects to the other Google apps, so it shows your calendar events. However, it also connects to Gmail and scans your emails for plane tickets, events, parcel tracking numbers etc.

j: Oh, I didn't know that.

s: There's no easy way to disable this that I can see in the app. Have any thoughts about this?

j: I think you should have to manually enable it.

s: What features would you use in a card system like this? I've seen you use the notes app often.

j: Alarms and weather and notes mostly. I don't like the reminders app, so I use notes instead.

s: So, most of your problem here is that you don't like the existing app. Would having driving directions here be useful?

j: I wouldn't use directions.

s: What about social media integration? Only Alexa has this. What's the social media you use the most?

j: Probably Twitter.

In summary, the features Jake has requested are:

- All information in the same place
- Either better voice recognition or minimal typing
- Fewer privacy concerns than Google Assistant
- Alarms, weather, notes, and reminders
- Social media integration, preferably Twitter

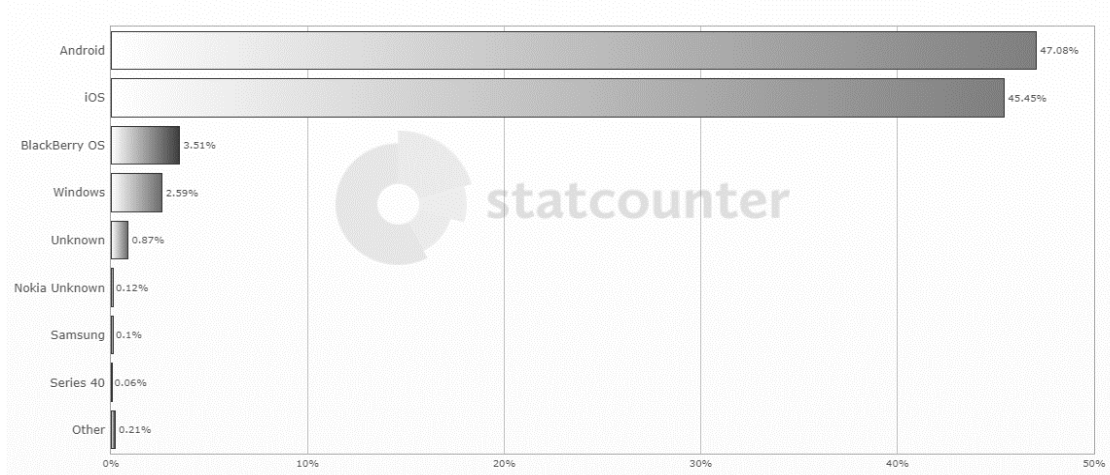
Voice recognition will be difficult to implement. Most speech APIs require an expensive subscription to use, and are often reserved for research or commercial purposes. Due to the limitations of my testing environment, I will also be unable to access a microphone, and I will also be unable to test the use of a microphone on a mobile device as this will require the use of app permissions and complex emulators.

I have chosen to merge the reminders and alarms functions. Much like the use of a microphone, the ability to make a mobile device sound an alarm will require access to these permissions, which I am unable to test. However, reminders and alarms have very similar purposes, so very little functionality is lost for the user.

The app should also be accessible and easy to use, to maximise the number of people that can use it. The elderly and visually impaired in particular can benefit from this kind of service as they may struggle to use keyboards and read text on a small screen.

Data from statcounter.com shows that 47% of users in the UK use Android, whereas 45% use iOS. This means that I will develop the app for both of these operating systems to allow a wide range of people to use the app. Other operating systems are used by only about 7.5% of users.

Mobile Operating System Market Share United Kingdom
Mar 2016 - Mar 2017



In order to create a python program that will run on mobile devices, I will use the Kivy library. Kivy enables python code to be easily run on both Android and iOS devices, whilst also supporting Windows and Linux PCs for ease of testing. There is also a simple markup language called the kv language that allows easy creation of layouts and styles for the app's screens.

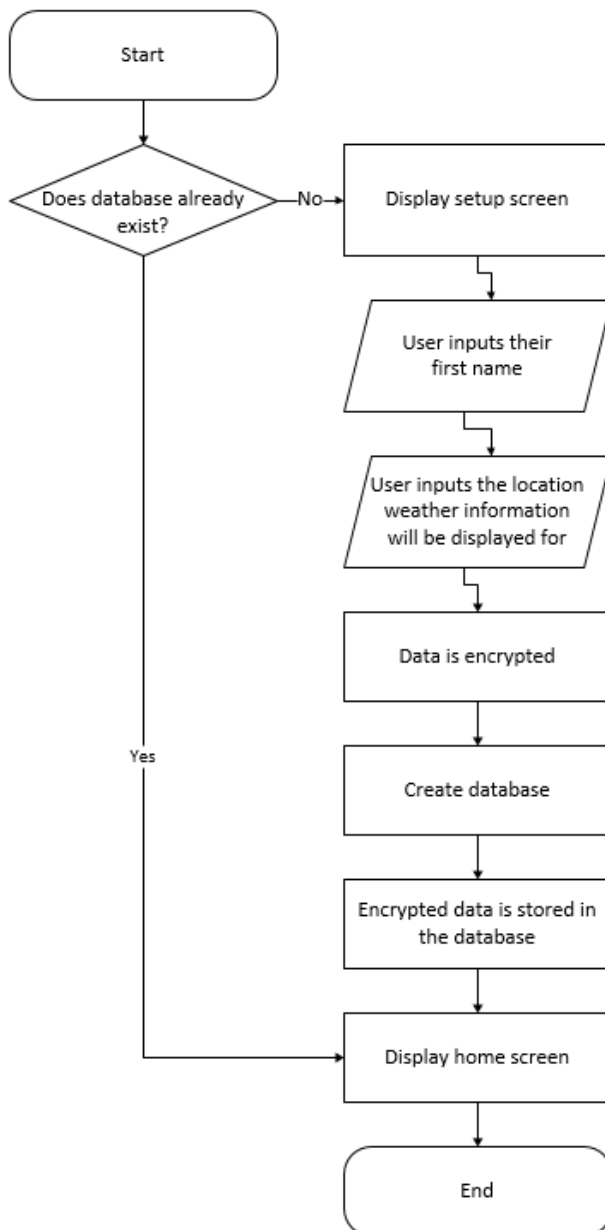
I have also decided to add additional functionality in the form of a csv export function. This will allow users to export their notes or reminders as a csv file, so that the data may be used in another program if the user wishes. I will also include the ability to email the created csv to an email address of the user's choice.

Objectives

Based on both the interview and the survey results, I have decided upon the following objectives:

- The app should be easy to use
 - The app should use simple, friendly language and address the user by their name
- The app should show information when it is relevant
 - Weather information should be retrieved from online API and displayed when appropriate

- The user should be able to define target locations for this
- The app should find new weather information for the current day in the morning and display this all day
- The user should be able to request weather information for any location at any time
- Reminders should be shown at the correct time
 - Existing reminders should be able to be searched for ease of use
- The app should display the last few notes taken
 - The user should also be able to view a sorted list of all their notes
 - All notes should be searchable based on the text inside them
- The app should integrate with a social media
 - Twitter provides an easy to use API
 - Users should be able to view tweets by accounts they follow
- Data should be kept secure
 - As little data as possible should be transferred to the internet for API usage
 - For example, only basic user defined location data should be needed to retrieve weather information
 - Data should be held securely in databases
 - This data should be encrypted
 - The user should be able to edit and delete information at any time
- The app should run as above on both Android and iOS
 - This will allow the app to be useful to as many people as possible
- The user should be able to export their data as a csv file
 - They should then be able to email the file using their own account to a target email of their choice



Diagrams

To the left is a flowchart representing the logic every time the app is opened. First, the program checks to see if the database that will hold the user's data already exists. If no such database is found, the user is shown the setup screen, where they can enter their name and location. This data is then encrypted and stored in the database, and then the home screen is displayed. If the database already exists, the setup has already been completed so the home screen is displayed.

Research

For research I have used the following sources:

<https://www.apple.com/uk/ios/siri/>

<https://assistant.google.com>

<https://developer.twitter.com/en/docs.html>

<https://www.amazon.com/Amazon-Echo-And-Alexa-Devices/b?ie=UTF8&node=9818047011>

<https://www.google.com/policies/privacy/>

<https://www.cnet.com/news/fooling-amazon-and-googles-voice-recognition-isnt-hard/>

<https://techcrunch.com/2011/10/04/apple-reveals-siri-voice-interface-the-intelligent-assistant/>

<http://gs.statcounter.com/os-market-share/mobile/united-kingdom/#monthly-201603-201703-bar>

<https://www.w3.org/WAI/mobile/>

Design

Technology

For my project I am going to use the Python programming language. I have chosen Python because it is easy to use, and I am already familiar with the language. Python has many tools and libraries available, such as Kivy, a library for cross-platform app development.

Kivy allows easy creation and testing of both simple and complex apps and includes an easy way to create and alter the app's layout and appearance using the kv language. Kivy programs can run on a PC for easy testing and there is an Android app called the Kivy Launcher which can run python files natively on the device, without the use of an emulator on a PC. Unfortunately, no such app exists for iOS, so I will be unable to test my app on those devices.

Algorithms

One of my objectives is that the user data will be encrypted. Any algorithm I could write myself will not be cryptographically secure, however as the user's data is not stored online and no sensitive information is stored, my intention is only to obfuscate the data rather than make it completely inaccessible. For this I will use a simple caesar cipher, with the key generated from the user's name:

```
alphabet ← "abcdefghijklmnopqrstuvwxyz"
key ← len name
for c in text:
    if c in alphabet:
        i ← index of c
        i ← i + key
        if i > 25:
            i ← i - len alphabet
        ciphertext ← ciphertext + alpha[i]
    else:
        ciphertext = ciphertext + c
```

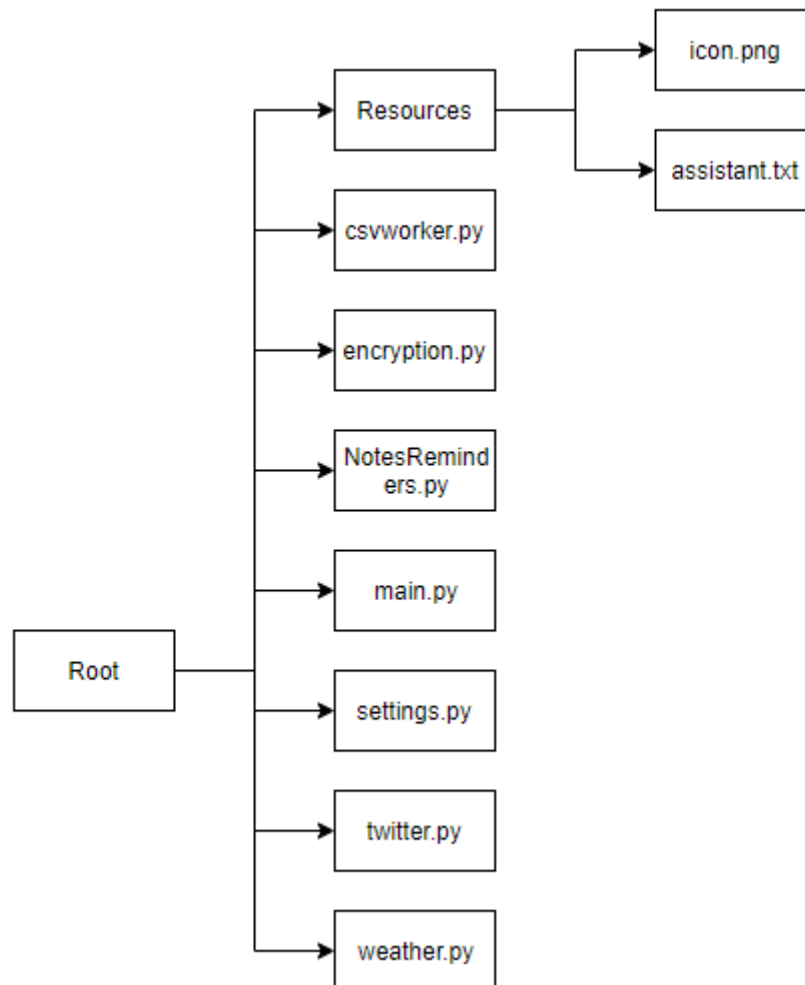
Data structures

The main data structure in my program is a database to store the user's details, notes, and reminders. This data is stored externally rather than in variables as it needs to be persistent, even if the program is closed.

Working data will be held in variables: for example, data will need to be retrieved from the database and decrypted in order to be worked with in the app. If more than one note or reminder is retrieved from the database in a single query, the data will be stored in a list of lists of strings, so it can be iterated over and displayed on the screen in the correct order.

File and class structure

I have decided to split my code across multiple files for simplicity and better code readability. The file structure looks like this:



The files, and the classes I will include within them are:

[Assistant.txt](#)

This file will contain kv language code, and will provide kivy with layout, style and hierarchy information for the widgets and screens.

[Csvworker.py](#)

This file contains a single class, csvworker, that contains two methods: one to handle exporting the csv, and another to email the file.

Function	Parameters	Returns	Purpose
init			Sets up a database connection and initialises the encryption class
exportcsv	Idtype (string)		Creates a csv file from the user's selected database table (notes or reminders)
email	Username (string), password (string), target (string)		Sends an email with the exported csv file attached over gmail servers using the user's email and password, to a user defined target address

Csvworker
+ alpha: string
+ key: integer
- init()
+ exportcsv(idtype)
+ email(username, password, target): boolean

Encryption.py

The file contains a single class, crypto, that contains methods to encrypt and decrypt text.

Function	Parameters	Returns	Purpose
Init	Setup (boolean), length (integer)	None	Determines if the program is in setup mode, if not then the key is the length of the user's name in the database, if it is then the key is the number given
Encrypt	Ciphertext (string)	Text (string)	Encrypts the given ciphertext with a simple caesar cipher
Decrypt	Ciphertext (string)	Text (strings)	Decrypts the given ciphertext with a simple caesar cipher

Crypto
+ alpha: string
+ key: integer
- init (setup, length)
+ encrypt(ciphertext): string
+ decrypt(ciphertext): string

Main.py

This is the main file, acting as a "hub" for the classes in the other files to interact with each other and be displayed. This will also contain the code required by kivy to create and display screens.


Notesreminders.py

This file contains classes for the creation, display, and retrieval of notes and reminders. Although the two sets of data are very similar to each other, the SQL queries are distinct enough from each other that the classes must remain separate, rather than inheriting from a common class. This also avoids any potential conflicts with duplicate ID numbers and the differing table columns.

Notes class:


Function	Parameters	Returns	Purpose
Create	Title (string), content (string)		Creates a database entry using the given data. The ID number is automatically generated and added.

Mostrecent		Title (string), Content (string)	Returns the most recent note the user created.
Edit	Noteid (integer), title (string), content (string)		Replaces the information in the chosen note with the new title and content.
Delete	Noteid (integer)		Deletes the selected note
Search	Searchterm (string)	Data (list of strings)	Returns all notes with the search term in their content.
Sort	Type (string)	Data (list of strings)	Returns all notes, sorted by the given column (Date or title)

 Notes
<pre> - init() + create(title, content) + mostrecent(): list + delete(noteid) + edit(noteid, title, content) + search(searchterm): list + sort(type): list </pre>

Reminders class:

Function	Parameters	Returns	Purpose
Create	Title (string), content (string), year, month, day, hour, minute, second (integers)		Creates a database entry using the given data. The ID number is automatically generated and added.
Mostrecent		Title (string), Content (string)	Returns the most recent reminder the user created
Edit	Title (string), content (string)		Allows the user to edit a chosen reminder.
Delete	Reminderid (integer)		Deletes the selected reminder
Search	Searchterm (string)	Data (list of strings)	Returns all reminders with the search term in their content.
Sort	Type (string)	Data (list of strings)	Returns all reminders, sorted by the given column (Date or title)

 Reminders
<pre> - init() + create(title, content, year, month, day, hour, minute, second) + mostrecent(): list + delete(reminderid) + edit(reminderid, title, content) + search(searchterm): list + sort(type): list </pre>

Settings.py

This file contains two classes, settings and setup. The settings class contains methods relating to the settings screen, and setup contains methods used in the setup process.

Settings class:

Function	Parameters	Returns	Purpose
Init			Sets up and authorises a Twitter API request, ready to be called
Userlatest	Username (string)	Text (string)	Gets the latest tweet from a given username from the Twitter API
User10	Username (string)	Text (list of strings)	Gets the previous 10 tweets from a given username from the Twitter API

Settings
- init()
+ changename(name)
+ changelocation(country, city)

Setup class:

Function	Parameters	Returns	Purpose
Completesetup	Name (string), country (string), city (string)	None	Creates the database tables and inserts the user's information into them.

Setup
+ completesetup(name, country, city)

Twitter.py

This file will contain a single class, Twitter, that will contain all the relevant methods for use of the Twitter API. This includes authenticating the requests and retrieving tweets.

Function	Parameters	Returns	Purpose
Init			Sets up and authorises a Twitter API request, ready to be called
Userlatest	Username (string)	Text (string)	Gets the latest tweet from a given username from the Twitter API
User10	Username (string)	Text (list of strings)	Gets the previous 10 tweets from a given username from the Twitter API

Twitter
+ twurl: string
- init()
+ userlatest(username): string
+ user10(username): list

Userdata.db

This file is a database file, used to store the user data and their notes, and reminders. It contains individual tables for each of these items.

Weather.py

This file will contain two classes, `weather4day` and `weather10day`, and is used to send requests to the wunderground.com API, and the methods within it return the relevant sections of data that wunderground.com returns. The methods extract the forecast for the current day, following day, or for the whole week from the json file provided by the API.

Weather4day class:

Function	Parameters	Returns	Purpose
init	Country (string), city (string)	None	Makes an api call for the next 4 days of weather using the country and city arguments, then parses the data as json format.
Forecasttodaytext		Forecast (string)	Returns the text version of today's forecast. Data included varies.
Forecasttodayhigh		Temperature (integer)	Returns the highest forecast temperature for the current day
Forecasttodaylow		Temperature (integer)	Returns the lowest forecast temperature for the current day

Weather4Day
+ data: json as dictionary
- init(country, city)
+ forecasttodaytext(): string
+ forecasttodayhigh(): integer
+ forecasttodaylow(): integer

Weather10day class:

Function	Parameters	Returns	Purpose
init	Country (string), city (string)	None	Makes an api call for the next 10 days of weather using the country and city arguments, then parses the data as json format.
Forecast10daystext		Forecast (string)	Returns the next 10 days of the text version of the forecast. Data included varies.
Forecast10dayshigh		Forecast (list of integers)	Returns the next 10 days of the highest forecast temperatures
Forecast10dayslow		Forecast (list of integers)	Returns the next 10 days of the lowest forecast temperatures
Daylist		Days (list of strings)	Returns the names of the next 10 days

Weather10Day
+ data: json as dictionary
- init(country, city)
+ forecast10daystext(): list
+ forecast10dayshigh(): list
+ forecast10dayslow(): list
+ daylight(): list of strings

API queries

My program contains multiple queries to online APIs. In order to handle these, I will use the requests (<http://docs.python-requests.org/en/master/>) library to simplify the process of using HTTP requests. The requests library links to requests-oauthlib (<https://requests-oauthlib.readthedocs.io/en/latest/>), a library designed to allow developers to authenticate their HTTP requests, which is needed for some of my API requests.

Weather

The API I will use for weather queries is wunderground.com. I have chosen this because it is free, accurate, and easy to use. The API offers many functions, such as astronomy and tide information, but I will only use the forecasting query. The request to the API server requires a key, which I will have to sign up for, and a location. The query format for a 3-day forecast is in the following format:

`http://api.wunderground.com/api/(key)/forecast/q/(state or country)/(city).json`

And the query for a 10-day forecast is in this format:

`http://api.wunderground.com/api/(key)/forecast10day/q/(state or country)/(city).json`

The data is returned in JSON format and must be parsed and filtered by my program. The JSON contains data I do not need, such as humidity and wind direction, but does contain other useful information such as the day of the week the forecast is for and the predicted temperature at both day and night. The data typically looks like this:

```
"forecast":{
  "txt_forecast":{
    "date":"5:59 AM PDT",
    "forecastday":[
      {
        "period":0,
        "icon":"partlycloudy",
        "icon_url":"http://icons.wxug.com/i/c/k/partlycloudy.gif",
        "title":"Tuesday",
        "fcttext":"Cloudy early with peaks of sunshine expected late. High
62F. Winds WSW at 10 to 20 mph.",
        "fcttext_metric":"Cloudy early with peaks of sunshine expected late.
High 16C. Winds WSW at 15 to 30 km/h.",
        "pop":"10"
      },

```

Twitter

The free Twitter API allows tweets to be searched up to 7 days ago, which is a reasonable timescale to construct a small timeline of the accounts the user wants to see. Python libraries for ease of use of the Twitter API exist, though I will avoid using these to reduce including unnecessary code and to retain control over my program.

Due to the limitations of the free Twitter API and the scope of the project, all Twitter requests are handled through a developer account made specifically for this project. At this scale the API does allow users to post to their account, but the developer account is the only account authenticated so the user will not have access to this feature.

A sample query to the API looks like this:

[https://api.twitter.com/1.1/statuses/user_timeline.json?screen_name=\(username\)](https://api.twitter.com/1.1/statuses/user_timeline.json?screen_name=(username))

Twitter requires all API requests to be authenticated using OAuth. For this OAuth needs an app token, app key, app secret and a token secret. These are all generated by Twitter and accessible from within the developer account. They are all hard-coded into the program so they can be used for authentication.

Libraries and packages

I intend to use the following external modules:

- Requests
 - Requests is a library designed to simplify HTTP requests in Python. I am using this for my API requests. It also includes a json decoder, which is useful for my project as the weather API returns data in json format
- Requests-OAuthlib
 - This library requires the use of the requests library and can authenticate HTTP requests. This is used to authenticate some of my API requests, but many do not require it.
- Kivy
 - Kivy is the basis for my entire program. It allows me to create an app that will run on most operating systems, including mobile devices.

And the following built in python modules:

- Sqlite3
 - Used for database requests. Allows control of databases using standard SQL requests.
- Csv
 - Used for writing to the csv file.
- Email
 - This is used for composing, attaching a file to, and then sending an email.
- Datetime
 - This is used for converting the user's date and time input to a standardised format.

- `Os.path`
 - Used to check if the database exists.

Database design

Notes table

Field	Data type	Sample data	Purpose
NoteID	Integer	52	Primary key
Title	String	"My note title"	Allows the user to title their notes for ease of navigation
Content	String	"My note content"	Contains the content of the user's note
Date	Float	1539980712.0	Date/time the note was created in seconds since the epoch, for sorting purposes.

Reminders table

Field	Data type	Sample data	Purpose
ReminderID	Integer	24	Primary key
Title	String	"My reminder title"	Allows the user to title their reminders for ease of navigation
Content	String	"My reminder content"	Contains the content of the user's reminder
Date	Float	1539941344.0	Contains the time the reminder will display at, in seconds since the epoch

UserInfo table

Field	Data type	Sample data	Purpose
Name	String	"John Smith"	The user's name
Country	String	United Kingdom	The user's country so weather information can be retrieved
City	String	London	The user's country so weather information can be retrieved
LastTwitterSearch	String	"@kedst"	The last twitter username the user searched for. Defaults to @kedst

Queries

The setup screen and its associated methods use SQL queries to create the tables, set the user's name and location in the database, as well as provide the default value for lasttwittersearch. The queries used for this are:

Query	Purpose
-------	---------

CREATE TABLE userInfo (Name text, Country text, City text, LastTwitterSearch text, primary key(Name))	Creates the userinfo table
CREATE TABLE Notes (NoteID integer, Title text, Content text, Date float, primary key(NoteID))	Creates the notes table
CREATE TABLE Reminders (ReminderID integer, Title text, Content text, Days text, Time time, Date float, Repeats boolean, primary key(ReminderID))	Creates the reminders table
INSERT INTO userInfo (Name, Country, City, LastTwitterSearch) VALUES ('}', '{', '{', '{')	Inserts the user's given data into the userinfo table

Notesreminders.py uses SQL queries to retrieve, create, edit, and delete notes and reminders.

Query	Purpose
INSERT INTO Notes (Title, Content, Date) VALUES ('}', '{', {);	Creates a note with the information given by the user. The date is automatically generated.
SELECT Max(Date) FROM Notes	Selects the date of the most recently created note
SELECT Title, Content FROM Notes WHERE Date='{'	Selects the title and content of the note with the given date (will be the most recent note)
DELETE FROM Notes WHERE NoteID='{'	Deletes the note with the given noteid
UPDATE Notes SET Title='{', Content='{ WHERE NoteID='{	Updates the title and content of the note with the given noteid with the new data
SELECT NoteID, Title, Content FROM Notes WHERE Content LIKE '%{'%	Selects the noteids, titles and contents of any notes that have the given search term in their content
SELECT NoteID, Title, Content FROM Notes ORDER BY {	Sorts all notes by the given sort type (alphabetical or by date)
INSERT INTO Reminders (Title, Content, Date) VALUES ('}', '{', {);	Creates a reminder with the information given by the user.
SELECT Min(Date) FROM Reminders	Selects the date of the reminder with the oldest date
SELECT Title, Content FROM Reminders WHERE Date='{'	Selects the title and content of the reminder with the given date
DELETE FROM Reminders WHERE ReminderID='{	Deletes the note with the given reminderid
UPDATE Reminders SET Title='{', Content='{ WHERE ReminderID='{;	Updates the title and content of the reminder with the given reminderid with the new data
SELECT ReminderID, Title, Content, Date FROM Reminders WHERE Content LIKE '%{'%	Selects the reminderids, titles, contents and dates of any reminders that have the given search term in their content
SELECT ReminderID, Title, Content, Date FROM Reminders ORDER BY {	Sorts all reminders by the given sort type (alphabetical or by date)

Settings.py also uses SQL queries to alter the user's name and location.

Query	Purpose
UPDATE userInfo SET Name='{	Updates the user's name
UPDATE userInfo SET Country='{	Updates the user's country
UPDATE userInfo SET City='{	Updates the user's city

In addition, main.py contains SQL queries used in various screens to retrieve data so it can be displayed.

Query	Purpose
SELECT Name FROM userInfo	Gets the user's name
SELECT city FROM userInfo	Gets the user's city
SELECT country FROM userInfo	Gets the user's country
UPDATE userInfo SET LastTwitterSearch='{'	Updates the last twitter username the user searched for

Screens

The structure of my program's screens is as follows:

- Setup
- Home
 - Weather
 - More weather
 - Twitter
 - More tweets
 - Notes
 - Create note
 - View notes by title
 - Edit note
 - View notes by date
 - Edit note
 - Reminders
 - Create reminder
 - View reminders by title
 - Edit reminder
 - View reminders by date
 - Edit reminder
 - Settings
 - Export
 - Email

Setup screen

This screen is shown to the user when they start the app for the first time. It contains text input boxes for the user's name, country, and city, and labels showing what each box is for. There is a single button which creates a database, encrypts the user's data, and then adds it to the database. The user is then shown the home screen.

Home screen

This screen is shown to the user by default, if the database is found. It displays a brief welcome message addressing the user by name, retrieved from the database. There are 5 buttons: weather, twitter, notes, reminders, and settings. Clicking each one will display the corresponding screen.

Weather screen

The main weather screen displays the user's set location and the text version of the current day's forecast for that location. It also displays the high and low temperatures for this forecast. Below this, there are text input boxes for country and city and labels marking each box. A button allows the user to see an extended weather forecast for this given location, which loads the "more weather" screen. There is also a button that when clicked displays the home screen.

The "more weather" screen displays the name of the day and the text forecast, as well as high and low forecast temperatures for the following 10 days. At the bottom there is a "back" button that returns the user to the main weather screen.

Twitter screen

The main Twitter screen displays the username the user last searched for and the most recent tweet from this username. There is a text input box for the username to be searched for with a corresponding label, and a button to execute the search. This loads the "more Twitter" screen. There is also a button to return to the home screen.

The "more Twitter" screen displays the username the user has searched for, and the last ten tweets from that username. There is a "back" button that displays the main Twitter screen.

Notes screen

The main notes screen shows the title and content of the last note the user created. Below this, there are two buttons: one to view all notes sorted by creation date, and another to view all notes sorted alphabetically by title. There is also a text input box and a "search" button, which will search all notes for the text in the text input box. Finally, there is a back button that displays the home screen.

The "more notes" screen is the basis for viewing the notes sorted by title and date, and for search results. It displays the title and content for each note along with buttons to edit or delete the note. At the bottom is a button to return to the main notes screen.

Reminders screen

The reminders screens are almost identical to the notes screens, with a few differences. When the user is creating a reminder, there are 6 text boxes corresponding to year, month, day, hour, minute, and second, which will not accept non-numeric input. This date and time is then displayed alongside the reminder in the more reminders section. The user can also edit this date when editing the rest of the reminder.

Viewing reminders sorted by date sorts them by the user given date, and the reminder displayed on the main reminders screen is the one with the oldest given date.

Settings screen

The settings screen contains a text box and a corresponding label for the user to change their name. There is a "change name" button that updates the userinfo table in the database when clicked. If the text box is empty, clicking the button should not do anything. There are also text boxes and a button to change the user's location.

Below this is an "export data" button which when clicked displays three buttons: "export notes", "export reminders" and a "back" button. Clicking either of the export buttons writes the user's data from the corresponding database table to a csv file. The email screen is then displayed.

The email screen has three text boxes with labels: "Email", "password", and "target". The password field displays any text input as asterisks. Below this is a "send" button that attempts to send an email with the csv file generated previously attached. If any of the input boxes are empty, clicking the button will not do anything. If the email is successful, the home screen is displayed.

Research

I have used the following sources:

<https://developer.twitter.com/en/docs>

<https://www.wunderground.com/weather/api/d/docs>

<https://cloud.google.com/speech-to-text/>

<http://docs.python-requests.org/en/master/>

<https://docs.python.org/3.6/>

Technical Solution

Assistant.txt

```

1. <HomeScreen>
2.     lblName: lblName
3.     BoxLayout:
4.         Label:
5.             id: lblName
6.             text: "Welcome, name!"
7.         Button:
8.             id: btnWeather
9.             text: "Weather"
10.            on_press: app.root.current = "weather"
11.        Button:
12.            id: btnTwitter
13.            text: "Twitter"
14.            on_press: app.root.current = "twitter"
15.        Button:
16.            id: btnNotes
17.            text: "Notes"
18.            on_press: app.root.current = "notes"
19.        Button:
20.            id: btnReminders
21.            text: "Reminders"
22.            on_press: app.root.current = "reminders"
23.        Button:
24.            id: btnSettings
25.            text: "Settings"
26.            on_press: app.root.current = "settings"
27.
28. <WeatherScreen>
29.     name: "weather"
30.     lblLocation: lblLocation
31.     lblWeatherText: lblWeatherText
32.     lblWeatherHigh: lblWeatherHigh
33.     lblWeatherLow: lblWeatherLow
34.     inputCity: inputCity
35.     inputCountry: inputCountry
36.
37.     BoxLayout:
38.         Label:
39.             id: lblLocation
40.             text: "latestLocation"
41.         Label:
42.             id: lblWeatherText
43.             text_size: self.size
44.             text: "latestWeatherText"
45.
46.         GridLayout:
47.             cols: 2
48.             Label:
49.                 text: "High:"
50.                 halign: "right"
51.             Label:
52.                 id: lblWeatherHigh
53.                 halign: "left"
54.                 text: "latestWeatherHigh"
55.             Label:
56.                 text: "Low:"
57.                 halign: "right"
58.             Label:
59.                 id: lblWeatherLow
60.                 halign: "left"
61.                 text: "latestWeatherLow"

```

```

62.
63.     GridLayout:
64.         cols: 2
65.         Label:
66.             text: "Country:"
67.         TextInput:
68.             id: inputCountry
69.         Label:
70.             text: "City:"
71.         TextInput:
72.             id: inputCity
73.
74.         Button:
75.             id: btnUpdateLocation
76.             text: "Get weather"
77.             on_press: root.getmoreweather()
78.         Button:
79.             id: btnBack
80.             text: "Back"
81.             on_press: app.root.current = "home"
82.
83. <MoreWeatherScreen>
84.     name: "moreweather"
85.     layoutMoreWeather: layoutMoreWeather
86.
87.     ScrollView:
88.         GridLayout:
89.             id: layoutMoreWeather
90.             size_hint_y: None
91.             height: self.minimum_height
92.             cols: 1
93.
94. <TwitterScreen>
95.     name: "twitter"
96.     inputTwitterUsername: inputTwitterUsername
97.     lblRecentUsername: lblRecentUsername
98.     lblRecentTweet: lblRecentTweet
99.
100.     BoxLayout:
101.         Label:
102.             id: lblRecentUsername
103.             text: "recentUsername"
104.         Label:
105.             id: lblRecentTweet
106.             text: "recentTweet"
107.
108.         GridLayout:
109.             cols: 2
110.             Label:
111.                 text: "@Username:"
112.             TextInput:
113.                 id: inputTwitterUsername
114.
115.         Button:
116.             id: btnSearchTweets
117.             text: "Search username"
118.             on_press: root.getmoretweets()
119.         Button:
120.             id: btnBack
121.             text: "Back"
122.             on_press: app.root.current = "home"
123.
124. <MoreTwitterScreen>
125.     name: "moretwitter"
126.     layoutMoreTwitter: layoutMoreTwitter
127.

```

```

128.     ScrollView:
129.         GridLayout:
130.             size_hint_y: None
131.             height: self.minimum_height
132.             id: layoutMoreTwitter
133.             cols: 1
134.
135.     <NotesScreen>
136.         name: "notes"
137.         inputSearchNotes: inputSearchNotes
138.         lblLastNoteContent: lblLastNoteContent
139.         lblLastNoteTitle: lblLastNoteTitle
140.
141.     BoxLayout:
142.         Label:
143.             id: lblLastNoteTitle
144.             text: "Most recent note title"
145.         Label:
146.             id: lblLastNoteContent
147.             text: "Most recent note content"
148.         Button:
149.             id: btnNewNote
150.             text: "New note"
151.             on_press: root.newnote()
152.         GridLayout:
153.             cols: 2
154.             Button:
155.                 id: btnAllNotes
156.                 text: "Notes by time"
157.                 on_press: root.notesbytime()
158.             Button:
159.                 id: btnAllNotes
160.                 text: "Notes by title"
161.                 on_press: root.notesbytitle()
162.         GridLayout:
163.             cols: 2
164.             TextInput:
165.                 id: inputSearchNotes
166.             Button:
167.                 id: btnSearchNotes
168.                 text: "Search notes"
169.                 on_press: root.searchnotes()
170.         Button:
171.             id: btnBack
172.             text: "Back"
173.             on_press: app.root.current = "home"
174.
175.     <NewNotesScreen>
176.         name: "newnotes"
177.         inputNewNoteTitle: inputNewNoteTitle
178.         inputNewNoteContent: inputNewNoteContent
179.
180.     BoxLayout:
181.         GridLayout:
182.             cols: 2
183.             Label:
184.                 id: lblNewNoteTitle
185.                 text: "Title:"
186.             TextInput:
187.                 id: inputNewNoteTitle
188.
189.         Label:
190.             id: lblNewNoteTitle
191.             text: "Content:"
192.         TextInput:
193.             id: inputNewNoteContent

```

```

194.         multiline: True
195.         Button:
196.             id: btnCreateNote
197.             text: "Create"
198.             on_press: root.createnote()
199.         Button:
200.             id: btnBack
201.             text: "Discard"
202.             on_press: app.root.current = "notes"
203.
204.     <MoreNotesScreen>
205.         name: "morenotes"
206.         layoutMoreNotes: layoutMoreNotes
207.         ScrollView:
208.             GridLayout:
209.                 id: layoutMoreNotes
210.                 cols: 1
211.                 size_hint_y: None
212.                 height: self.minimum_height
213.
214.     <EditNotesScreen>
215.         name: "editnotes"
216.         inputEditNoteTitle: inputEditNoteTitle
217.         inputEditNoteContent: inputEditNoteContent
218.         layoutEditNotes: layoutEditNotes
219.
220.         BoxLayout:
221.             id: layoutEditNotes
222.             GridLayout:
223.                 cols: 2
224.                 Label:
225.                     id: lblEditNoteTitle
226.                     text: "Title:"
227.                 TextInput:
228.                     id: inputEditNoteTitle
229.
230.                 Label:
231.                     id: lblEditNoteTitle
232.                     text: "Content:"
233.                 TextInput:
234.                     id: inputEditNoteContent
235.                     multiline: True
236.                 Button:
237.                     id: btnEditNote
238.                     text: "Save"
239.                     on_press: root.editnote()
240.                 Button:
241.                     id: btnBack
242.                     text: "Discard"
243.                     on_press: app.root.current = "morenotes"
244.
245.     <RemindersScreen>
246.         name: "reminders"
247.         inputSearchReminders: inputSearchReminders
248.         lblLastReminderContent: lblLastReminderContent
249.         lblLastReminderTitle: lblLastReminderTitle
250.
251.         BoxLayout:
252.             Label:
253.                 id: lblLastReminderTitle
254.                 text: "Most recent reminder title"
255.             Label:
256.                 id: lblLastReminderContent
257.                 text: "Most recent reminder content"
258.             Button:
259.                 id: btnNewReminder

```

```

260.         text: "New reminder"
261.         on_press: root.newreminder()
262.     GridLayout:
263.         cols: 2
264.         Button:
265.             id: btnAllReminders
266.             text: "Reminders by time"
267.             on_press: root.remindersbytime()
268.         Button:
269.             id: btnAllReminders
270.             text: "Reminders by title"
271.             on_press: root.remindersbytitle()
272.     GridLayout:
273.         cols: 2
274.         TextInput:
275.             id: inputSearchReminders
276.         Button:
277.             id: btnSearchReminders
278.             text: "Search reminders"
279.             on_press: root.searchreminders()
280.     Button:
281.         id: btnBack
282.         text: "Back"
283.         on_press: app.root.current = "home"
284.
285. <NewRemindersScreen>
286.     name: "newreminders"
287.     inputNewReminderTitle: inputNewReminderTitle
288.     inputNewReminderContent: inputNewReminderContent
289.     inputNewReminderYear: inputNewReminderYear
290.     inputNewReminderMonth: inputNewReminderMonth
291.     inputNewReminderDay: inputNewReminderDay
292.     inputNewReminderHour: inputNewReminderHour
293.     inputNewReminderMinute: inputNewReminderMinute
294.     inputNewReminderSecond: inputNewReminderSecond
295.
296.     BoxLayout:
297.         GridLayout:
298.             cols: 2
299.             Label:
300.                 id: lblNewReminderTitle
301.                 text: "Title:"
302.             TextInput:
303.                 id: inputNewReminderTitle
304.
305.             Label:
306.                 id: lblNewReminderTitle
307.                 text: "Content:"
308.             TextInput:
309.                 id: inputNewReminderContent
310.                 multiline: True
311.
312.             Label:
313.                 id: lblNewReminderDateTime
314.                 text: "Content:"
315.
316.             GridLayout:
317.                 cols: 3
318.                 Label:
319.                     id: lblNewReminderYear
320.                     text: "Year"
321.                 Label:
322.                     id: lblNewReminderMonth
323.                     text: "Month"
324.                 Label:
325.                     id: lblNewReminderDay
326.                     text: "Day"
327.                 TextInput:

```

```

326.         id: inputNewReminderYear
327.     TextInput:
328.         id: inputNewReminderMonth
329.     TextInput:
330.         id: inputNewReminderDay
331.     GridLayout:
332.         cols: 3
333.         Label:
334.             id: lblNewReminderHour
335.             text: "Hour"
336.         Label:
337.             id: lblNewReminderMinute
338.             text: "Min"
339.         Label:
340.             id: lblNewReminderSecond
341.             text: "Sec"
342.     TextInput:
343.         id: inputNewReminderHour
344.     TextInput:
345.         id: inputNewReminderMinute
346.     TextInput:
347.         id: inputNewReminderSecond
348.
349.     Button:
350.         id: btnCreateReminder
351.         text: "Create"
352.         on_press: root.createreminder()
353.     Button:
354.         id: btnBack
355.         text: "Discard"
356.         on_press: app.root.current = "reminders"
357.
358.     <MoreRemindersScreen>
359.         name: "morereminders"
360.         layoutMoreReminders: layoutMoreReminders
361.     ScrollView:
362.         GridLayout:
363.             id: layoutMoreReminders
364.             cols: 1
365.             size_hint_y: None
366.             height: self.minimum_height
367.
368.     <EditRemindersScreen>
369.         name: "editreminders"
370.         inputEditNReminderTitle: inputEditReminderTitle
371.         inputEditReminderContent: inputEditReminderContent
372.         layoutEditReminder: layoutEditReminder
373.
374.     BoxLayout:
375.         id: layoutEditReminder
376.         GridLayout:
377.             cols: 2
378.             Label:
379.                 id: lblEditReminderTitle
380.                 text: "Title:"
381.             TextInput:
382.                 id: inputEditReminderTitle
383.
384.             Label:
385.                 id: lblEditReminderTitle
386.                 text: "Content:"
387.             TextInput:
388.                 id: inputEditReminderContent
389.                 multiline: True
390.         Button:
391.             id: btnEditReminder

```

```

392.         text: "Save"
393.         on_press: root.editReminder()
394.     Button:
395.         id: btnBack
396.         text: "Discard"
397.         on_press: app.root.current = "morereminder"
398.
399.     <SettingsScreen>
400.         name: "settings"
401.         inputNewName: inputNewName
402.         inputNewCountry: inputNewCountry
403.         inputNewCity: inputNewCity
404.
405.     BoxLayout:
406.         Label:
407.             text: "Changes will take effect after restart"
408.         GridLayout:
409.             cols: 2
410.             Label:
411.                 text: "Name:"
412.             TextInput:
413.                 id: inputNewName
414.
415.         Button:
416.             id: btnChangeName
417.             text: "Change name"
418.             on_press: root.changename()
419.
420.         GridLayout:
421.             cols: 2
422.             Label:
423.                 text: "Country:"
424.             TextInput:
425.                 id: inputNewCountry
426.             Label:
427.                 text: "City:"
428.             TextInput:
429.                 id: inputNewCity
430.
431.         Button:
432.             id: btnChangeLocation
433.             text: "Change location"
434.             on_press: root.changelocation()
435.         Button:
436.             id: btnExport
437.             text: "Export Data"
438.             on_press: app.root.current = "export"
439.         Button:
440.             id: btnBack
441.             text: "Back"
442.             on_press: app.root.current = "home"
443.
444.     <SetupScreen>
445.         name: "setup"
446.         inputName: inputName
447.         inputCountry: inputCountry
448.         inputCity: inputCity
449.
450.     BoxLayout:
451.         GridLayout:
452.             cols: 2
453.             Label:
454.                 text: "Name:"
455.             TextInput:
456.                 id: inputName
457.             Label:

```



```

458.             text: "Country:"
459.             TextInput:
460.                 id: inputCountry
461.             Label:
462.                 text: "City:"
463.             TextInput:
464.                 id: inputCity
465.             Button:
466.                 id: btnConfirm
467.                 text: "Confirm"
468.                 on_press: root.completesetup()
469.
470.         <ExportScreen>
471.             name: "export"
472.             BoxLayout:
473.                 Button:
474.                     id: btnExportNotes
475.                     text: "Export Notes"
476.                     on_press: root.exportnotes()
477.                 Button:
478.                     id: btnExportReminders
479.                     text: "Export Reminders"
480.                     on_press: root.exportreminders()
481.                 Button:
482.                     id: btnBack
483.                     text: "Cancel"
484.                     on_press: app.root.current = "settings"
485.
486.         <EmailScreen>
487.             name: "email"
488.             inputEmailUsername: inputEmailUsername
489.             inputEmailPassword: inputEmailPassword
490.             inputEmailTarget: inputEmailTarget
491.
492.             BoxLayout:
493.                 GridLayout:
494.                     cols: 2
495.                     Label:
496.                         text: "Username"
497.                     TextInput:
498.                         id: inputEmailUsername
499.                     Label:
500.                         text: "Password"
501.                     TextInput:
502.                         id: inputEmailPassword
503.                         password: True
504.                     Label:
505.                         text: "Target email"
506.                     TextInput:
507.                         id: inputEmailTarget
508.                 Button:
509.                     id: btnSendEmail
510.                     text: "Send"
511.                     on_press: root.sendemail()
512.                 Button:
513.                     id: btnBack
514.                     text: "Cancel"
515.                     on_press: app.root.current = "settings"
516.
517.         <Label>
518.             text_size: self.width, None
519.             padding_x: "20dp"
520.             padding_y: "20dp"
521.             halign: "center"
522.             valign: "center"
523.             font_size: "15dp"

```

```

524.
525.     <BoxLayout>
526.         orientation: "vertical"
527.
528.     <TextInput>
529.         multiline: False

```

Csvworker.py

```

1. import csv
2. import sqlite3
3. import smtplib
4. from email.mime.multipart import MIMEMultipart
5. from email.mime.text import MIMEText
6. from email.mime.base import MIMEBase
7. from email import encoders
8. from encryption import Crypto
9.
10.
11. class csvworker:
12.     def __init__(self):
13.         self.db = sqlite3.connect("resources/UserData.db")
14.         self.cursor = self.db.cursor()
15.         self.c = Crypto(False, 0)
16.
17.     # Method - exportcsv
18.     # Parameters - idtype: string
19.     # Return - None
20.     # Purpose - Creates a csv file from the user's selected database table (notes or
reminders)
21.     def exportcsv(self, idtype):
22.         file = open("resources/output.csv", "w")
23.         writer = csv.writer(file)
24.         sql = """SELECT {}ID, Title, Content, Date FROM {}s ORDER BY Title""".format(
idtype, idtype)
25.         self.cursor.execute(sql)
26.         data = self.cursor.fetchall()
27.         writer.writerow(["ID", "Title", "Content", "Date"])
28.         for i in range(len(data)):
29.             writer.writerow([data[i][0], self.c.decrypt(data[i][1]), self.c.decrypt(d
ata[i][2]), data[i][3]])
30.
31.     # Method - email
32.     # Parameters - username: string, password: string, target: string
33.     # Return - None
34.     # Purpose - Sends an email with the exported csv file attached over gmail servers
using the user's email and
35.                 password, to a user defined target address
36.     def email(self, username, password, target):
37.         try:
38.             msg = MIMEMultipart()
39.             msg['From'] = username
40.             msg['To'] = target
41.             msg['Subject'] = "Assistant - Data output"
42.             body = "Attached is the output data in csv format, as created by Assistan
t"
43.             msg.attach(MIMEText(body, 'plain'))
44.             filename = "resources/output.csv"
45.             attachment = open("resources/output.csv", "rb")
46.             file = MIMEBase('application', 'octet-stream')
47.             file.set_payload(attachment.read())
48.             encoders.encode_base64(file)

```

```

49.         file.add_header('Content-
Disposition', "attachment; filename= %s" % filename)
50.         msg.attach(file)
51.         server = smtplib.SMTP('smtp.gmail.com', 587)
52.         server.starttls()
53.         server.login(username, password)
54.         text = msg.as_string()
55.         server.sendmail(username, target, text)
56.         server.quit()
57.         return True
58.     except:
59.         return False

```

Encryption.py

```

1. import sqlite3
2.
3.
4. class Crypto:
5.     # Method - Crypto init
6.     # Parameters - setup: boolean, length: integer
7.     # Return - None
8.     # Purpose - Determines if the program is in setup mode, if not then the key is the
length of the user's name in the
9.     # database, if it is then the key is the length argument
10.    def __init__(self, setup, length):
11.        self.alpha = "abcdefghijklmnopqrstuvwxyz"
12.        if setup is True:
13.            self.key = length
14.
15.        else:
16.            with sqlite3.connect("resources/UserData.db") as db:
17.                cursor = db.cursor()
18.                cursor.execute("SELECT Name FROM userInfo")
19.                name = cursor.fetchone()
20.                name = name[0]
21.                self.key = len(name)
22.
23.    # Method - encrypt
24.    # Parameters - ciphertext: string
25.    # Return - text: string
26.    # Purpose - Encrypts the given ciphertext with a simple caesar cipher
27.    def encrypt(self, ciphertext):
28.        text = ""
29.        for c in ciphertext:
30.            if c in self.alpha:
31.                i = self.alpha.find(c)
32.                i = i + self.key
33.                if i > 25:
34.                    i = i - len(self.alpha)
35.                text = text + self.alpha[i]
36.            else:
37.                text = text + c
38.        return text
39.
40.    # Method - decrypt
41.    # Parameters - ciphertext: string
42.    # Return - text: string
43.    # Purpose - Decrypts the given ciphertext with a simple caesar cipher
44.    def decrypt(self, ciphertext):
45.        text = ""
46.        for c in ciphertext:
47.            if c in self.alpha:

```

```

48.         i = self.alpha.find(c)
49.         i = i - self.key
50.         if i < 0:
51.             i = i + len(self.alpha)
52.         text = text + self.alpha[i]
53.     else:
54.         text = text + c
55.     return text

```

Main.py

```

1.     # -*- coding: utf-8 -*-
2.
3.     import sqlite3
4.     import os.path
5.     import datetime
6.
7.     from kivy.app import App
8.     from kivy.lang import Builder
9.     from kivy.uix.screenmanager import ScreenManager, Screen
10.
11.     from weather import Weather4Day, Weather10Day
12.     from settings import Setup, Settings
13.     from twitter import Twitter
14.     from encryption import Crypto
15.     from csvworker import csvworker
16.     from NotesReminders import Notes, Reminders
17.
18.     from kivy.core.window import Window
19.     from kivy.uix.label import Label
20.     from kivy.uix.button import Button
21.     from kivy.uix.gridlayout import GridLayout
22.     from kivy.metrics import dp
23.
24.     # Assistant.txt contains layout, widget and formatting information for all of the
25.     # screens
26.     Builder.load_file("resources/assistant.txt")
27.
28.     class HomeScreen(Screen):
29.         # Method - HomeScreen init
30.         # Parameters - username: string
31.         # Return - None
32.         # Purpose - Retrieves the user's name from the database and displays it in a
33.         # label when the screen is built by kivy
34.         def __init__(self, **kwargs):
35.             super(HomeScreen, self).__init__(**kwargs)
36.
37.             with sqlite3.connect("resources/UserData.db") as db:
38.                 cursor = db.cursor()
39.                 cursor.execute("SELECT Name FROM userInfo")
40.                 username = cursor.fetchone()
41.                 c = Crypto(False, 0)
42.                 username = c.decrypt(username[0])
43.                 self.lblName.text = "Welcome, {}".format(username)
44.
45.         #####
46.
47.     class WeatherScreen(Screen):
48.         # Method - WeatherScreen init
49.         # Parameters - None

```

```

50.         # Return - None
51.         # Purpose - Calls getweather function when the screen is built by kivy
52.         def __init__(self, **kwargs):
53.             super(WeatherScreen, self).__init__(**kwargs)
54.             self.getweather()
55.
56.         # Method - getweather
57.         # Parameters - city: string, country: string
58.         # Return - None
59.         # Purpose - Gets the weather forecast for the current day for the user's set
location and displays them in labels
60.         def getweather(self):
61.             with sqlite3.connect("resources/UserData.db") as db:
62.                 cursor = db.cursor()
63.                 cursor.execute("SELECT city FROM userInfo")
64.                 city = cursor.fetchone()
65.                 cursor.execute("SELECT country FROM userInfo")
66.                 country = cursor.fetchone()
67.                 c = Crypto(False, 0)
68.                 city = c.decrypt(city[0])
69.                 country = c.decrypt(country[0])
70.
71.                 w = Weather4Day(country, city)
72.
73.                 self.lblLocation.text = "The weather in {}, {} is".format(city, country)
74.
75.                 self.lblWeatherText.text = w.forecasttodaytext()
76.                 self.lblWeatherHigh.text = w.forecasttodayhigh() + "C"
77.                 self.lblWeatherLow.text = w.forecasttodaylow() + "C"
78.
79.         # Method - getmoreweather
80.         # Parameters - city: string, country: string, textdata: string, highdata: str
ing, lowdata: string
81.         # Return - None
82.         # Purpose - Obtains and displays the data on the moreweather screen using the
user's chosen location
83.         def getmoreweather(self):
84.             city = self.inputCity.text
85.             country = self.inputCountry.text
86.
87.             if city == "" or country == "":
88.                 pass
89.             else:
90.                 moreweather = self.manager.get_screen("moreweather")
91.                 moreweather.layoutMoreWeather.clear_widgets(moreweather.layoutMoreWea
ther.children)
92.
93.                 w = Weather10Day(country, city)
94.                 textdata = w.forecast10daystext()
95.                 highdata = w.forecast10dayshigh()
96.                 lowdata = w.forecast10dayslow()
97.                 days = w.daylist()
98.
99.                 for i in range(len(days)):
100.                     lblday = Label(text=days[i], size_hint_y=None)
101.                     lblday.texture_update()
102.                     moreweather.layoutMoreWeather.add_widget(lblday)
103.
104.                     lbltext = Label(text=textdata[i], size_hint_y=None)
105.                     lbltext.texture_update()
106.                     moreweather.layoutMoreWeather.add_widget(lbltext)
107.
108.                     grid = GridLayout(cols=2, size_hint_y=None)
109.                     lblhigh = Label(text=highdata[i], size_hint_y=None)
110.                     lbllow = Label(text=lowdata[i], size_hint_y=None)

```

```

111.         lblhigh.texture_update()
112.         lbllow.texture_update()
113.         grid.add_widget(lblhigh)
114.         grid.add_widget(lbllow)
115.         moreweather.layoutMoreWeather.add_widget(grid)
116.
117.         btnback = Button(text="Back", height=dp(40), size_hint_y=None, on_pre
ss=lambda a: self.back())
118.         moreweather.layoutMoreWeather.add_widget(btnback)
119.         self.manager.current = "moreweather"
120.
121.         # Method - back
122.         # Parameters - None
123.         # Return - None
124.         # Purpose - Displays the weather screen
125.         def back(self):
126.             self.manager.current = "weather"
127.
128.
129.         class MoreWeatherScreen(Screen):
130.             def __init__(self, **kwargs):
131.                 super(MoreWeatherScreen, self).__init__(**kwargs)
132.
133.         #####
134.
135.
136.         class TwitterScreen(Screen):
137.
138.             # Method - TwitterScreen init
139.             # Parameters - None
140.             # Return - None
141.             # Purpose - When kivy has built the screen, initialises instances of the Cryp
to and Twitter classes for this class,
142.             # then calls the latesttweet function
143.             def __init__(self, **kwargs):
144.                 super(TwitterScreen, self).__init__(**kwargs)
145.                 self.db = sqlite3.connect("resources/UserData.db")
146.                 self.cursor = self.db.cursor()
147.                 self.c = Crypto(False, 0)
148.                 self.t = Twitter()
149.                 self.latesttweet()
150.
151.             # Method - latesttweet
152.             # Parameters - username
153.             # Return - None
154.             # Purpose - Retrieves the username the user last searched for, fetches their
latest tweet from the API, then
155.             # displays their username latest tweet in labels
156.             def latesttweet(self):
157.                 self.cursor.execute("SELECT LastTwitterSearch FROM userInfo")
158.                 username = self.cursor.fetchone()
159.                 username = self.c.decrypt(username[0])
160.                 self.lblRecentTweet.text = self.t.userlatest(username)
161.                 self.lblRecentUsername.text = "Latest tweet from @" + username
162.
163.             # Method - getmoretweets
164.             # Parameters - un:string
165.             # Return - None
166.             # Purpose - Obtains and displays the data on the moretweets screen using the
user's chosen location
167.             def getmoretweets(self):
168.                 un = self.inputTwitterUsername.text
169.                 if un == "":
170.                     pass
171.                 else:

```

```

172.         moretwitter = self.manager.get_screen("moretwitter")
173.         moretwitter.layoutMoreTwitter.clear_widgets(moretwitter.layoutMoreTwitter.children)
174.
175.         tweets = self.t.user10(un)
176.         lblun = Label(text="Latest tweets from @" + un), size_hint_y=None)
177.         moretwitter.layoutMoreTwitter.add_widget(lblun)
178.
179.         for i in range(len(tweets)):
180.             lbltweet = Label(text=tweets[i], size_hint_y=None)
181.             lbltweet.texture_update()
182.             moretwitter.layoutMoreTwitter.add_widget(lbltweet)
183.
184.         btnback = Button(text="Back", height=dp(40), size_hint_y=None, on_press=
ss=lambda a: self.back())
185.         moretwitter.layoutMoreTwitter.add_widget(btnback)
186.         self.manager.current = "moretwitter"
187.
188.         # Method - back
189.         # Parameters - username: string, secureusername: string
190.         # Return - None
191.         # Purpose - Updates the main twitter screen with the latest tweet from the username the user last searched for,
192.         #             then displays the twitter screen.
193.         def back(self):
194.             username = self.inputTwitterUsername.text
195.             secureusername = self.c.encrypt(username)
196.             sql = "UPDATE userInfo SET LastTwitterSearch='{0}'".format(secureusername)
197.             self.cursor.execute(sql)
198.             self.db.commit()
199.             self.lblRecentUsername.text = "Latest tweet from {0}".format(username)
200.             self.lblRecentTweet.text = self.t.userlatest(username)
201.             self.manager.current = "twitter"
202.
203.
204.         class MoreTwitterScreen(Screen):
205.             def __init__(self, **kwargs):
206.                 super(MoreTwitterScreen, self).__init__(**kwargs)
207.
208.             #####
209.
210.
211.         class NotesScreen(Screen):
212.             # Method - NoteScreen init
213.             # Parameters - None
214.             # Return - None
215.             # Purpose - Initialises instances of the notes and crypto classes, connects to the database, and runs the latestnote method
216.             #             latestnote method
217.             def __init__(self, **kwargs):
218.                 super(NotesScreen, self).__init__(**kwargs)
219.                 self.c = Crypto(False, 0)
220.                 self.n = Notes()
221.                 self.latestnote()
222.                 self.db = sqlite3.connect("resources/UserData.db")
223.                 self.cursor = self.db.cursor()
224.
225.             # Method - latestnote
226.             # Parameters - data: list of strings
227.             # Return - None
228.             # Purpose - If the user has any notes, display the most recent one in labels
229.
230.             def latestnote(self):
231.                 data = self.n.mostrecent()

```

```

231.         if data is False:
232.             self.lblLastNoteTitle.text = "No notes found!"
233.             self.lblLastNoteContent.text = " "
234.         else:
235.             recenttitle = data[0]
236.             recentcontent = data[1]
237.             recenttitle = self.c.decrypt(recenttitle)
238.             recentcontent = self.c.decrypt(recentcontent)
239.             self.lblLastNoteTitle.text = recenttitle
240.             self.lblLastNoteContent.text = recentcontent
241.
242.     # Method - newnote
243.     # Parameters - None
244.     # Return - None
245.     # Purpose - Clears the content of the text inputs of the newnotes screen, the
n displays it
246.     def newnote(self):
247.         newnotes = self.manager.get_screen("newnotes")
248.         newnotes.inputNewNoteTitle.text = ""
249.         newnotes.inputNewNoteContent.text = ""
250.         self.parent.current = "newnotes"
251.
252.     # Method - notesbytime
253.     # Parameters - data: list of strings
254.     # Return - None
255.     # Purpose - Retrieves a list of the user's notes sorted by time, then passes
it to the setupmorenotes function
256.     def notesbytime(self):
257.         data = self.n.sort("Date")
258.         count = len(data)
259.         self.setupmorenotes(count, data)
260.
261.     # Method - notesbytitle
262.     # Parameters - data: list of strings
263.     # Return - None
264.     # Purpose - Retrieves a list of the user's notes sorted by title alphabetical
ly, then passes it to the
265.         setupmorenotes function
266.     def notesbytitle(self):
267.         data = self.n.sort("Title")
268.         count = len(data)
269.         self.setupmorenotes(count, data)
270.
271.     # Method - searchnotes
272.     # Parameters - searchterm: string, data: list of strings
273.     # Return - None
274.     # Purpose - Gets a list of the user's notes containing a given search term, t
hen passes it to the setupmorenotes
275.         function
276.     def searchnotes(self):
277.         searchterm = self.inputSearchNotes.text
278.         if searchterm == "":
279.             pass
280.         else:
281.             searchterm = self.c.encrypt(searchterm)
282.             data = self.n.search(searchterm)
283.             count = len(data)
284.             self.setupmorenotes(count, data)
285.
286.     # Method - back
287.     # Parameters - None
288.     # Return - None
289.     # Purpose - Runs the latestnote method, then displays the notes screen
290.     def back(self):
291.         self.latestnote()
292.         self.manager.current = "notes"

```



```

293.
294.     # Method - delete
295.     # Parameters - noteid: string
296.     # Return - None
297.     # Purpose - Deletes the note corresponding to the given noteid, then calls the
298.     #           latestnote method and displays the
299.     #           notes screen
300.     def delete(self, noteid):
301.         self.n.delete(noteid)
302.         self.latestnote()
303.         self.manager.current = "notes"
304.
305.     # Method - edit
306.     # Parameters - noteid: string
307.     # Return - None
308.     # Purpose - Displays the editnotes screen, and passes noteid to it
309.     def edit(self, noteid):
310.         editnotes = self.manager.get_screen("editnotes")
311.         editnotes.currentnoteid = noteid
312.         self.manager.current = "editnotes"
313.
314.     # Method - setupmorenotes
315.     # Parameters - count: integer , data: list of strings
316.     # Return - None
317.     # Purpose - Adds widgets displaying the title and content of each note within
318.     #           the given list, as well as their
319.     #           corresponding edit and delete buttons to the morenotes screen. A "back" but
320.     #           ton is then added and the screen is
321.     #           displayed.
322.     def setupmorenotes(self, count, data):
323.         morenotes = self.manager.get_screen("morenotes")
324.         morenotes.layoutMoreNotes.clear_widgets(morenotes.layoutMoreNotes.childre
325.         n)
326.         for i in range(count):
327.             noteid = data[i][0]
328.             title = data[i][1]
329.             title = self.c.decrypt(title)
330.             content = data[i][2]
331.             content = self.c.decrypt(content)
332.
333.             lbltitle = Label(text=title, size_hint_y=None)
334.             lbltitle.texture_update()
335.             morenotes.layoutMoreNotes.add_widget(lbltitle)
336.
337.             lbltext = Label(text=content, size_hint_y=None)
338.             lbltext.texture_update()
339.             morenotes.layoutMoreNotes.add_widget(lbltext)
340.
341.             grid = GridLayout(cols=2, size_hint_y=None)
342.
343.             btncedit = Button(text="Edit", size_hint_y=None, on_press=lambda a: se
344.             lf.edit(noteid))
345.             btncdelete = Button(text="Delete", size_hint_y=None, on_press=lambda a
346.             : self.delete(noteid))
347.             btncedit.texture_update()
348.             btncdelete.texture_update()
349.             grid.add_widget(btncedit)
350.             grid.add_widget(btncdelete)
351.             morenotes.layoutMoreNotes.add_widget(grid)
352.
353.             grid = GridLayout(cols=2, size_hint_y=None)
354.             btnback = (Button(text="Back", height=dp(80), on_press=lambda a: self.bac
355.             k()))
356.             grid.add_widget(btnback)
357.             morenotes.layoutMoreNotes.add_widget(grid)
358.             self.manager.current = "morenotes"

```

```

352.
353.
354. class NewNotesScreen(Screen):
355.     def __init__(self, **kwargs):
356.         super(NewNotesScreen, self).__init__(**kwargs)
357.
358.         # Method - createnote
359.         # Parameters - title: string, content: string
360.         # Return - None
361.         # Purpose - Encrypts the title and content and passes them to the create meth
od, then displays the notes screen
362.     def createnote(self):
363.         title = self.inputNewNoteTitle.text
364.         content = self.inputNewNoteContent.text
365.
366.         if title == "" or content == "":
367.             pass
368.         else:
369.             c = Crypto(False, 0)
370.             title = c.encrypt(title)
371.             content = c.encrypt(content)
372.             n = Notes()
373.             n.create(title, content)
374.             notes = self.manager.get_screen("notes")
375.             notes.latestnote()
376.             self.manager.current = "notes"
377.
378.
379. class MoreNotesScreen(Screen):
380.     pass
381.
382.
383. class EditNotesScreen(Screen):
384.     # Method - editnote
385.     # Parameters - noteid: string, title: string, content: string
386.     # Return - None
387.     # Purpose - Encrypts the title and content and passes them to the edit method
, then displays the notes screen
388.     def editnote(self):
389.         noteid = self.currentnoteid
390.         title = self.inputEditNoteTitle.text
391.         content = self.inputEditNoteContent.text
392.         if title == "" or content == "":
393.             pass
394.         else:
395.             c = Crypto(False, 0)
396.             title = c.encrypt(title)
397.             content = c.encrypt(content)
398.             n = Notes()
399.             n.edit(noteid, title, content)
400.             notes = self.manager.get_screen("notes")
401.             notes.latestnote()
402.             self.manager.current = "notes"
403.
404. #####
405. #####
406.
407. class RemindersScreen(Screen):
408.     # Method - RemindersScreen init
409.     # Parameters - None
410.     # Return - None
411.     # Purpose - Initialises instances of the reminders and crypto classes, conne
cts to the database, and runs the
412.     # latestreminder method
413.     def __init__(self, **kwargs):

```

```

414.         super(RemindersScreen, self).__init__(**kwargs)
415.         self.c = Crypto(False, 0)
416.         self.r = Reminders()
417.         self.latestreminder()
418.         self.db = sqlite3.connect("resources/UserData.db")
419.         self.cursor = self.db.cursor()
420.
421.         # Method - latestreminder
422.         # Parameters - data: list of strings
423.         # Return - None
424.         # Purpose - If the user has any reminders, display the most recent one in lab
    else
425.         def latestreminder(self):
426.             data = self.r.mostrecent()
427.             if data is False:
428.                 self.lblLastReminderTitle.text = "No reminders found!"
429.                 self.lblLastReminderContent.text = " "
430.             else:
431.                 recenttitle = data[0]
432.                 recentcontent = data[1]
433.                 recenttitle = self.c.decrypt(recenttitle)
434.                 recentcontent = self.c.decrypt(recentcontent)
435.                 self.lblLastReminderTitle.text = recenttitle
436.                 self.lblLastReminderContent.text = recentcontent
437.
438.         # Method - newreminder
439.         # Parameters - None
440.         # Return - None
441.         # Purpose - Clears the content of the text inputs of the newreminder screen,
    then displays it
442.         def newreminder(self):
443.             newreminders = self.manager.get_screen("newreminders")
444.             newreminders.inputNewReminderTitle.text = ""
445.             newreminders.inputNewReminderContent.text = ""
446.             newreminders.inputNewReminderYear.text = ""
447.             newreminders.inputNewReminderMonth.text = ""
448.             newreminders.inputNewReminderDay.text = ""
449.             newreminders.inputNewReminderHour.text = ""
450.             newreminders.inputNewReminderMinute.text = ""
451.             newreminders.inputNewReminderSecond.text = ""
452.             self.parent.current = "newreminders"
453.
454.         # Method - remindersbytime
455.         # Parameters - data: list of strings
456.         # Return - None
457.         # Purpose - Retrieves a list of the user's reminders sorted by time, then pas
    ses it to the setupmorereminders
458.         # function
459.         def remindersbytime(self):
460.             data = self.r.sort("Date")
461.             count = len(data)
462.             self.setupmorereminders(count, data)
463.
464.         # Method - remindersbytime
465.         # Parameters - data: list of strings
466.         # Return - None
467.         # Purpose - Retrieves a list of the user's reminders sorted by title alphabet
    ically, then passes it to the
468.         # setupmorereminders function
469.         def remindersbytitle(self):
470.             data = self.r.sort("Title")
471.             count = len(data)
472.             self.setupmorereminders(count, data)
473.
474.         # Method - searchreminders
475.         # Parameters - searchterm: string, data: list of strings

```

```

476.         # Return - None
477.         # Purpose - Gets a list of the user's reminders containing a given search term, then passes it to the
478.         # setupmorereminders function
479.         def searchreminders(self):
480.             searchterm = self.inputSearchReminders.text
481.             if searchterm == "":
482.                 pass
483.             else:
484.                 searchterm = self.c.encrypt(searchterm)
485.                 data = self.r.search(searchterm)
486.                 count = len(data)
487.                 self.setupmorereminders(count, data)
488.
489.         # Method - setupmorereminders
490.         # Parameters - count: integer , data: list of strings
491.         # Return - None
492.         # Purpose - Adds widgets displaying the title and content of each reminder within the given list, as well as their
493.         # corresponding edit and delete buttons to the morereminders screen. A "back" button is then added and the screen is
494.         # displayed.
495.         def setupmorereminders(self, count, data):
496.             morereminders = self.manager.get_screen("morereminders")
497.             morereminders.layoutMoreReminders.clear_widgets(morereminders.layoutMoreReminders.children)
498.
499.             for i in range(count):
500.                 reminderid = data[i][0]
501.                 title = data[i][1]
502.                 title = self.c.decrypt(title)
503.                 content = data[i][2]
504.                 content = self.c.decrypt(content)
505.                 date = data[i][3]
506.                 date = datetime.datetime.fromtimestamp(date)
507.
508.                 lbltitle = Label(text=title, size_hint_y=None)
509.                 lbltitle.texture_update()
510.                 morereminders.layoutMoreReminders.add_widget(lbltitle)
511.
512.                 lbltext = Label(text=content, size_hint_y=None)
513.                 lbltext.texture_update()
514.                 morereminders.layoutMoreReminders.add_widget(lbltext)
515.
516.                 lbldate = Label(text=str(date), size_hint_y=None)
517.                 lbldate.texture_update()
518.                 morereminders.layoutMoreReminders.add_widget(lbldate)
519.
520.                 grid = GridLayout(cols=2, size_hint_y=None)
521.                 btncedit = Button(text="Edit", size_hint_y=None, on_press=lambda a: self.edit(reminderid))
522.                 btncdelete = Button(text="Delete", size_hint_y=None, on_press=lambda a: self.delete(reminderid))
523.                 btncedit.texture_update()
524.                 btncdelete.texture_update()
525.                 grid.add_widget(btncedit)
526.                 grid.add_widget(btncdelete)
527.                 morereminders.layoutMoreReminders.add_widget(grid)
528.
529.                 grid = GridLayout(cols=2, size_hint_y=None)
530.                 btnback = (Button(text="Back", height=dp(80), on_press=lambda a: self.back()))
531.                 grid.add_widget(btnback)
532.                 morereminders.layoutMoreReminders.add_widget(grid)
533.                 self.manager.current = "morereminders"
534.

```

```

535.         # Method - back
536.         # Parameters - None
537.         # Return - None
538.         # Purpose - Runs the latestreminder method, then displays the reminders screen
539.         n
540.         def back(self):
541.             self.latestreminder()
542.             self.manager.current = "reminders"
543.         # Method - delete
544.         # Parameters - reminderid: string
545.         # Return - None
546.         # Purpose - Deletes the reminder corresponding to the given reminderid, then
547.         # calls the latestreminder method and
548.         # displays the reminders screen
549.         def delete(self, reminderid):
550.             self.r.delete(reminderid)
551.             self.latestreminder()
552.             self.manager.current = "reminders"
553.         # Method - edit
554.         # Parameters - reminderid: string
555.         # Return - None
556.         # Purpose - Displays the editreminders screen, and passes reminderid to it
557.         def edit(self, reminderid):
558.             editreminders = self.manager.get_screen("editreminders")
559.             editreminders.currentreminderid = reminderid
560.             self.manager.current = "editreminders"
561.
562.
563.         class NewRemindersScreen(Screen):
564.             def __init__(self, **kwargs):
565.                 super(NewRemindersScreen, self).__init__(**kwargs)
566.
567.             # Method - createreminder
568.             # Parameters - title: string, content: string
569.             # Return - None
570.             # Purpose - Encrypts the title and content and passes them and the given time
571.             # s to the create method, then displays
572.             # the reminders screen
573.             def createreminder(self):
574.                 title = self.inputNewReminderTitle.text
575.                 content = self.inputNewReminderContent.text
576.                 year = self.inputNewReminderYear.text
577.                 month = self.inputNewReminderMonth.text
578.                 day = self.inputNewReminderDay.text
579.                 hour = self.inputNewReminderHour.text
580.                 minute = self.inputNewReminderMinute.text
581.                 second = self.inputNewReminderSecond.text
582.
583.                 if title == "" or content == "" or year == "" or month == "" or day == ""
584.                 or hour == "" or minute == "" or second == "":
585.                     pass
586.                 else:
587.                     if year.isnumeric is False or month.isnumeric is False or day.isnumer
588.                     ic is False or hour.isnumeric is False or minute.isnumeric is False or second.isn
589.                     umeric is False:
590.                         pass
591.                     else:
592.                         c = Crypto(False, 0)
593.                         title = c.encrypt(title)
594.                         content = c.encrypt(content)
595.                         r = Reminders()
596.                         r.create(title, content, int(year), int(month), int(day), int(hou
597.                         r), int(minute), int(second))
598.                         reminders = self.manager.get_screen("reminders")

```

```

594.         reminders.latestreminder()
595.         self.manager.current = "reminders"
596.
597.
598.     class MoreRemindersScreen(Screen):
599.         pass
600.
601.
602.     class EditRemindersScreen(Screen):
603.         # Method - editreminder
604.         # Parameters - reminderid: string, title: string, content: string
605.         # Return - None
606.         # Purpose - Encrypts the title and content and passes them to the edit method
        , then displays the reminders screen
607.         def editreminder(self):
608.             reminderid = self.currentreminderid
609.             title = self.inputEditReminderTitle.text
610.             content = self.inputEditReminderContent.text
611.             if title == "" or content == "":
612.                 pass
613.             else:
614.                 c = Crypto(False, 0)
615.                 title = c.encrypt(title)
616.                 content = c.encrypt(content)
617.                 n = Reminders()
618.                 n.edit(reminderid, title, content)
619.                 reminders = self.manager.get_screen("reminders")
620.                 reminders.latestreminder()
621.                 self.manager.current = "reminders"
622.
623.         #####
624.         #####
625.
626.     class SettingsScreen(Screen):
627.         def __init__(self, **kwargs):
628.             super(SettingsScreen, self).__init__(**kwargs)
629.
630.         # Method - changename
631.         # Parameters - name: string
632.         # Return - None
633.         # Purpose - Passes the user's given name to the changename method in settings
        .py, if a name has been given
634.         def changename(self):
635.             name = self.inputNewName.text
636.             if name == "":
637.                 pass
638.             else:
639.                 s = Settings()
640.                 s.changename(name)
641.
642.         # Method - changelocation
643.         # Parameters - city: string, country: string
644.         # Return - None
645.         # Purpose - If a city or country has been given, pass them to the changelocat
        ion method in settings.py
646.         def changelocation(self):
647.             city = self.inputNewCity.text
648.             country = self.inputNewCountry.text
649.             if city == "" or country == "":
650.                 pass
651.             else:
652.                 s = Settings()
653.                 s.changelocation(country, city)
654.
655.

```

```

656. class SetupScreen(Screen):
657.     def __init__(self, **kwargs):
658.         super(SetupScreen, self).__init__(**kwargs)
659.
660.         # Method - completesetup
661.         # Parameters - name: string, city: string, country: string
662.         # Return - None
663.         # Purpose - If a name, city, and country have been given, pass them to the co
664.         #           mpletesetup method in settings.py, then
665.         #           add the screens that require completesetup to have run, then show
666.         #           the home screen
667.         def completesetup(self):
668.             name = self.inputName.text
669.             city = self.inputCity.text
670.             country = self.inputCountry.text
671.
672.             if name == "" or city == "" or country == "":
673.                 pass
674.             else:
675.                 setup = Setup()
676.                 setup.completesetup(name, country, city)
677.                 sm.add_widget(HomeScreen(name="home"))
678.                 sm.add_widget(WeatherScreen(name="weather"))
679.                 sm.add_widget(TwitterScreen(name="twitter"))
680.                 sm.add_widget(NotesScreen(name="notes"))
681.                 sm.add_widget(RemindersScreen(name="reminders"))
682.                 sm.add_widget(ExportScreen(name="export"))
683.                 self.parent.current = "home"
684.
685. class ExportScreen(Screen):
686.     # Method - ExportScreen init
687.     # Parameters - None
688.     # Return - None
689.     # Purpose - Initialises an instance of csvworker for this screen
690.     def __init__(self, **kwargs):
691.         super(ExportScreen, self).__init__(**kwargs)
692.         self.c = csvworker()
693.
694.         # Method - exportnotes
695.         # Parameters - None
696.         # Return - None
697.         # Purpose - Pass "Note" to the exportcsv method, then display the email scree
698.         n
699.         def exportnotes(self):
700.             self.c.exportcsv("Note")
701.             self.parent.current = "email"
702.
703.         # Method - exportreminders
704.         # Parameters - None
705.         # Return - None
706.         # Purpose - Pass "Reminder" to the exportcsv method, then display the email s
707.         creen
708.         def exportreminders(self):
709.             self.c.exportcsv("Reminder")
710.             self.parent.current = "email"
711.
712. class EmailScreen(Screen):
713.     def __init__(self, **kwargs):
714.         super(EmailScreen, self).__init__(**kwargs)
715.
716.         # Method - sendemail
717.         # Parameters - username: string, password: string, target: string
718.         # Return - None

```

```

717.     # Purpose - Initialises csvworker, fetches the user's username, password, and
       target email, then passes them to the
718.     #         email method
719.     def sendemail(self):
720.         c = csvworker()
721.         username = self.inputEmailUsername.text
722.         password = self.inputEmailPassword.text
723.         target = self.inputEmailTarget.text
724.         if username == "" or password == "" or target == "":
725.             pass
726.         else:
727.             if c.email(username, password, target) == False:
728.                 self.parent.current = "home"
729.
730.     #####
       #####
731.
732.     # Initialise a screenmanager
733.     sm = ScreenManager()
734.
735.
736.     class AssistantApp(App):
737.         # Sets the app title to "Assistant"
738.         title = "Assistant"
739.
740.         # Method - addscreens
741.         # Parameters - None
742.         # Return - None
743.         # Purpose - Adds screens that do not require setup to be completed
744.         def addscreens(self):
745.             sm.add_widget(MoreWeatherScreen(name="moreweather"))
746.             sm.add_widget(MoreTwitterScreen(name="moretwitter"))
747.             sm.add_widget(NewNotesScreen(name="newnotes"))
748.             sm.add_widget(MoreNotesScreen(name="morenotes"))
749.             sm.add_widget(EditNotesScreen(name="editnotes"))
750.             sm.add_widget(NewRemindersScreen(name="newreminders"))
751.             sm.add_widget(MoreRemindersScreen(name="morereminders"))
752.             sm.add_widget(EditRemindersScreen(name="editreminders"))
753.             sm.add_widget(SettingsScreen(name="settings"))
754.             sm.add_widget(EmailScreen(name="email"))
755.
756.         # Method - build
757.         # Parameters - None
758.         # Return - sm (screen manager)
759.         # Purpose - Kivy method. Adds the screen manager which contains all the scree
       ns to be displayed. Also used to
760.         #         set the app icon.
761.         def build(self):
762.             self.icon = "resources/icon.png"
763.             return sm
764.
765.         # Method - on_start
766.         # Parameters - None
767.         # Return - None
768.         # Purpose - Kivy method. Checks if the database exists- if yes, all screens a
       re added and the home screen is
769.         #         displayed. If no, the setup screen and the screens in addscreens
       are added, then the setup screen is
770.         #         displayed.
771.         def on_start(self):
772.             if os.path.exists("resources/UserData.db") is True:
773.                 sm.add_widget(HomeScreen(name="home"))
774.                 sm.add_widget(WeatherScreen(name="weather"))
775.                 sm.add_widget(TwitterScreen(name="twitter"))
776.                 sm.add_widget(NotesScreen(name="notes"))
777.                 sm.add_widget(RemindersScreen(name="reminders"))

```



```

778.         sm.add_widget(ExportScreen(name="export"))
779.         self.addscreens()
780.         self.root.current = "home"
781.     else:
782.         sm.add_widget(SetupScreen(name="setup"))
783.         self.addscreens()
784.         self.root.current = "setup"
785.
786. # Create the app, set the window size, then run the app.
787. if __name__ == "__main__":
788.     app = AssistantApp()
789.     Window.size = (360, 640)
790.     app.run()

```

NotesReminders.py

```

1. import sqlite3
2. import time
3. import datetime
4.
5.
6. # Notes class
7. class Notes:
8.     def __init__(self):
9.         self.db = sqlite3.connect("resources/UserData.db")
10.        self.cursor = self.db.cursor()
11.
12.    # Method - create
13.    # Parameters - title: string, content: string
14.    # Return - None
15.    # Purpose - Inserts the title, content, and current datetime into the database
16.    def create(self, title, content):
17.        date = time.time()
18.        sql = """insert into Notes (Title, Content, Date) values ('{}', '{}', {});"""
19.        .format(title, content, date)
20.        self.cursor.execute(sql)
21.        self.db.commit()
22.
23.    # Method - mostrecent
24.    # Parameters - None
25.    # Return - data: list of strings
26.    # Purpose - Fetches the most recently created note from the database
27.    def mostrecent(self):
28.        sql = """SELECT Max(Date) FROM Notes"""
29.        self.cursor.execute(sql)
30.        data = self.cursor.fetchall()
31.        data = data[0][0]
32.        sql = """SELECT Title, Content FROM Notes WHERE Date='{}'""".format(data)
33.        self.cursor.execute(sql)
34.        data = self.cursor.fetchall()
35.        if data:
36.            return data[0]
37.        else:
38.            return False
39.
40.    # Method - delete
41.    # Parameters - noteid: integer
42.    # Return - None
43.    # Purpose - Deletes the note corresponding to the given noteid
44.    def delete(self, noteid):
45.        sql = """delete from Notes where NoteID='{}'""".format(noteid)
46.        self.cursor.execute(sql)
47.        self.db.commit()

```

```

47.
48.     # Method - edit
49.     # Parameters - noteid: integer, title: string, content: string
50.     # Return - None
51.     # Purpose - Edits the note corresponding to the given noteid, replacing the current values with those in title and
52.                 content
53.     def edit(self, noteid, title, content):
54.         sql = """UPDATE Notes SET Title='{ }', Content='{ }' WHERE NoteID='{ }';""".format(title, content, noteid)
55.         self.cursor.execute(sql)
56.         self.db.commit()
57.
58.     # Method - search
59.     # Parameters - searchterm: string
60.     # Return - data: list of strings
61.     # Purpose - Returns the notes that have the search term in their title
62.     def search(self, searchterm):
63.         sql = """SELECT NoteID, Title, Content FROM Notes WHERE Content LIKE '%{ }%'"""
64.         self.cursor.execute(sql)
65.         data = self.cursor.fetchall()
66.         return data
67.
68.     # Method - sort
69.     # Parameters - type: string
70.     # Return - data: list of strings
71.     # Purpose - Returns the user's notes, sorted by the given sort type (alphabetical or by date)
72.     def sort(self, type):
73.         sql = """SELECT NoteID, Title, Content FROM Notes ORDER BY { }""".format(type)
74.
75.         self.cursor.execute(sql)
76.         data = self.cursor.fetchall()
77.         return data
78.
79. # Reminders class
80. class Reminders:
81.     def __init__(self):
82.         self.db = sqlite3.connect("resources/UserData.db")
83.         self.cursor = self.db.cursor()
84.
85.     # Method - create
86.     # Parameters - title: string, content: string, year: integer, month: integer, day: integer, hour: integer,
87.                 minute: integer, second: integer
88.     # Return - None
89.     # Purpose - Inserts the title, content, and given datetime into the database
90.     def create(self, title, content, year, month, day, hour, minute, second):
91.         date = datetime.datetime(year, month, day, hour, minute, second)
92.         date = date.timestamp()
93.         sql = """insert into Reminders (Title, Content, Date) values ('{ }', '{ }', { })
94.         ;""".format(title, content, date)
95.         self.cursor.execute(sql)
96.         self.db.commit()
97.
98.     # Method - mostrecent
99.     # Parameters - None
100.    # Return - data: list of strings
101.    # Purpose - Fetches the next reminder from the database
102.    def mostrecent(self):
103.        sql = """SELECT Min(Date) FROM Reminders"""
104.        self.cursor.execute(sql)
105.        data = self.cursor.fetchall()
106.        data = data[0][0]

```

```

106.         sql = """SELECT Title, Content FROM Reminders WHERE Date='{ }'""".format(d
    ata)
107.         self.cursor.execute(sql)
108.         data = self.cursor.fetchall()
109.         if data:
110.             return data[0]
111.         else:
112.             return False
113.
114.     # Method - delete
115.     # Parameters - reminderid: integer
116.     # Return - None
117.     # Purpose - Deletes the reminder corresponding to the given reminderid
118.     def delete(self, reminderid):
119.         sql = """delete from Reminders where ReminderID='{ }'""".format(reminderid
    )
120.         self.cursor.execute(sql)
121.         self.db.commit()
122.
123.     # Method - edit
124.     # Parameters - reminderid: integer, title: string, content: string
125.     # Return - None
126.     # Purpose - Edits the reminder corresponding to the given reminderid, replaci
    ng the current values with those in
127.     #           title and content
128.     def edit(self, reminderid, title, content):
129.         sql = """UPDATE Reminders SET Title='{ }', Content='{ }' WHERE ReminderID='
    { }';""".format(title, content, reminderid)
130.         self.cursor.execute(sql)
131.         self.db.commit()
132.
133.     # Method - search
134.     # Parameters - searchterm: string
135.     # Return - data: list of strings
136.     # Purpose - Returns the reminders that have the search term in their title
137.     def search(self, searchterm):
138.         sql = """SELECT ReminderID, Title, Content, Date FROM Reminders WHERE Con
    tent LIKE '%{ }%'""".format(searchterm)
139.         self.cursor.execute(sql)
140.         data = self.cursor.fetchall()
141.         return data
142.
143.     # Method - sort
144.     # Parameters - type: string
145.     # Return - data: list of strings
146.     # Purpose - Returns the user's reminders, sorted by the given sort type (alph
    abetical or by date)
147.     def sort(self, type):
148.         sql = """SELECT ReminderID, Title, Content, Date FROM Reminders ORDER BY
    { }""".format(type)
149.         self.cursor.execute(sql)
150.         data = self.cursor.fetchall()
151.         return data

```

Settings.py

```

1. import sqlite3
2. from encryption import Crypto
3.
4.
5. class Settings:
6.     # Method - Settings init
7.     # Parameters - None

```

```

8.     # Return - None
9.     # Purpose - Initialises an instance of the crypto class for this class and sets u
p the database for editing
10.    def __init__(self):
11.        self.db = sqlite3.connect("resources/UserData.db")
12.        self.cursor = self.db.cursor()
13.        self.c = Crypto(None, 0)
14.
15.    # Method - changename
16.    # Parameters - name: string
17.    # Return - None
18.    # Purpose - Encrypts the user's chosen name then sets the user's name in the data
base
19.    def changename(self, name):
20.        name = self.c.encrypt(name)
21.        self.cursor.execute("""UPDATE userInfo SET Name='{ }'""".format(name))
22.        self.db.commit()
23.
24.    # Method - changelocation
25.    # Parameters - country: string, city: string
26.    # Return - None
27.    # Purpose - Sets the user's location in the database to the country and city argu
ments
28.    def changelocation(self, country, city):
29.        country = self.c.encrypt(country)
30.        city = self.c.encrypt(city)
31.        self.cursor.execute("""UPDATE userInfo SET Country='{ }'""".format(country))
32.        self.cursor.execute("""UPDATE userInfo SET City='{ }'""".format(city))
33.        self.db.commit()
34.
35.
36. class Setup:
37.     # Method - completesetup
38.     # Parameters - name: string, country: string, city: string
39.     # Return - None
40.     # Purpose - Creates the database ables and adds the user's information to the dat
abase
41.     def completesetup(self, name, country, city):
42.         c = Crypto(True, len(name))
43.         name = c.encrypt(name)
44.         country = c.encrypt(country)
45.         city = c.encrypt(city)
46.         un = c.encrypt("kedst")
47.         db = sqlite3.connect("resources/UserData.db")
48.         cursor = db.cursor()
49.
50.         sql = "CREATE TABLE userInfo (Name text, Country text, City text, LastTwitter
Search text, primary key(Name))"
51.         cursor.execute(sql)
52.
53.         sql = """INSERT INTO userInfo (Name, Country, City, LastTwitterSearch) VALUES
('{ }', '{ }', '{ }', '{ }')""".format(name, country, city, un)
54.         cursor.execute(sql)
55.
56.         # Notes table
57.         sql = """CREATE TABLE Notes (NoteID integer, Title text, Content text, Date f
loat, primary key(NoteID))"""
58.         cursor.execute(sql)
59.
60.         # Reminders table
61.         sql = """CREATE TABLE Reminders (ReminderID integer, Title text, Content text
, Date float, primary key(ReminderID))"""
62.         cursor.execute(sql)
63.         db.commit()

```

Twitter.py

```

1. import requests
2. from requests_oauthlib import OAuth1
3.
4.
5. class Twitter:
6.     # Method - Twitter init
7.     # Parameters - token: string, appkey: string, appsecret: string, tokensecret:
    string
8.     # Return - Non
9.     # Purpose - Sets up and authorises a Twitter API request, ready to be called
10.
11.     def __init__(self):
12.         token = "989416304108064770-HwtB1mm13xiKBOMq1Tjq2roq9SZCh0"
13.         appkey = "RcuKZbyWJiSXsb57Pvuprkzr"
14.         appsecret = "CSdnGZ1ZZkPfMrWF9qTR98sRAFMPiuTf10SouAHV0d8hCKHmN"
15.         tokensecret = "pRhsJmMozHLycb7e6rc2wy0Xrk74e2yvVIZkfaGUYCFLU"
16.         self.auth = OAuth1(appkey, appsecret, token, tokensecret)
17.         self.twurl = "https://api.twitter.com/1.1/statuses/user_timeline.json?scr
    een_name="
18.
19.     # Method - userlatest
20.     # Parameters - username: string
21.     # Return - text: string
22.     # Purpose - Gets the latest tweet from a given username from the Twitter API
23.
24.     def userlatest(self, username):
25.         try:
26.             url = self.twurl + username + "&count=1"
27.             response = requests.get(url, auth=self.auth)
28.             f = response.json()
29.             q = f[0]
30.             return q["text"]
31.         except:
32.             return "No tweets found!"
33.
34.     # Method - user10
35.     # Parameters - username: string
36.     # Return - s: string
37.     # Purpose - Gets the previous 10 tweets from a given username from the Twitte
    r API
38.
39.     def user10(self, username):
40.         try:
41.             url = self.twurl + username + "&count=10"
42.             response = requests.get(url, auth=self.auth)
43.             f = response.json()
44.             s = []
45.             for tweet in f:
46.                 s.append(tweet["text"])
47.             return s
48.         except:
49.             return ["No tweets found!"]

```

Weather.py

```

1. import requests
2.
3.
4. class Weather4Day:
5.     # Method - Weather4Day init

```

```

6.     # Parameters - country: string, city: string
7.     # Return - None
8.     # Purpose - Makes an api call for the next 4 days of weather using the country and city arguments, then parses the
9.     #             data as json format.
10.    def __init__(self, country, city):
11.        try:
12.            url = "http://api.wunderground.com/api/034fa2c65c35e441/forecast/q/{}/{}.json".format(country, city)
13.            response = requests.get(url)
14.            self.data = response.json()
15.            print(self.data)
16.        except:
17.            self.data = "Can't connect"
18.
19.    # Method - forecasttodaytext
20.    # Parameters - None
21.    # Return - s: string
22.    # Purpose - Gets the text forecast for the current day
23.    def forecasttodaytext(self):
24.        try:
25.            for day in self.data["forecast"]["txt_forecast"]["forecastday"]:
26.                if day["period"] == 0:
27.                    s = day["fcttext_metric"]
28.            return s
29.        except:
30.            return "No forecast found!"
31.
32.    # Method - forecasttodayhigh
33.    # Parameters - None
34.    # Return - s: integer
35.    # Purpose - Gets the highest forecast temperature for the current day
36.    def forecasttodayhigh(self):
37.        try:
38.            for day in self.data["forecast"]["simpleforecast"]["forecastday"]:
39.                if day["period"] == 1:
40.                    s = day["high"]["celsius"]
41.            return s
42.        except:
43.            return ""
44.
45.    # Method - forecasttodaylow
46.    # Parameters - None
47.    # Return - s: integer
48.    # Purpose - Gets the highest forecast temperature for the current day
49.    def forecasttodaylow(self):
50.        try:
51.            for day in self.data["forecast"]["simpleforecast"]["forecastday"]:
52.                if day["period"] == 1:
53.                    s = day["low"]["celsius"]
54.            return s
55.        except:
56.            return ""
57.
58.
59. class Weather10Day:
60.     # Method - Weather10Day init
61.     # Parameters - country: string, city: string
62.     # Return - None
63.     # Purpose - Makes an api call for the next 10 days of weather using the country and city arguments, then parses the
64.     #             data as json format.
65.     def __init__(self, country, city):
66.         try:
67.             url = "http://api.wunderground.com/api/034fa2c65c35e441/forecast10day/q/{}/{}.json".format(country, city)

```

```

68.         response = requests.get(url)
69.         self.data = response.json()
70.     except:
71.         self.data = "Can't connect"
72.
73.     # Method - forecast10daystext
74.     # Parameters - None
75.     # Return - s: list of strings
76.     # Purpose - Gets the next 10 days of text forecasts
77.     def forecast10daystext(self):
78.         s = []
79.         try:
80.             for day in self.data["forecast"]["txt_forecast"]["forecastday"]:
81.                 # Only days, not nights
82.                 if day["period"] % 2 == 0:
83.                     s.append(day["fcttext_metric"])
84.             return s
85.         except:
86.             return ["No forecast found!"]
87.
88.     # Method - forecast10dayshigh
89.     # Parameters - None
90.     # Return - s: list of strings
91.     # Purpose - Gets the next 10 days of highest forecast temperatures
92.     def forecast10dayshigh(self):
93.         s = []
94.         try:
95.             for day in self.data["forecast"]["simpleforecast"]["forecastday"]:
96.                 s.append(day["high"]["celsius"])
97.             return s
98.         except:
99.             return [""]
100.
101.     # Method - forecast10dayslow
102.     # Parameters - None
103.     # Return - s: list of strings
104.     # Purpose - Gets the next 10 days of lowest forecast temperatures
105.     def forecast10dayslow(self):
106.         s = []
107.         try:
108.             for day in self.data["forecast"]["simpleforecast"]["forecastday"]:
109.                 s.append(day["low"]["celsius"])
110.             return s
111.         except:
112.             return [""]
113.
114.     # Method - daylist
115.     # Parameters - None
116.     # Return - s: list of strings
117.     # Purpose - Gets the names of the 10 days included in the forecast
118.     def daylist(self):
119.         days = []
120.         try:
121.             for day in self.data["forecast"]["txt_forecast"]["forecastday"]:
122.                 # only days, no nights
123.                 if day["period"] % 2 == 0:
124.                     days.append(day["title"])
125.             return days
126.         except:
127.             return [""]
128.

```


Testing

Test strategy

My program consists of several key areas: The setup process, the home screen, the weather section, the twitter section, the notes section, the reminders section, the settings screen and the csv and email process. Many small, insignificant tests have been conducted during the programming process however major system testing has not yet been completed before this stage. Testing will allow me to ensure the app runs smoothly and meets my objectives.

For the setup process, the program needs to check if the user has already completed the setup. This is done by checking to see if the database already exists, under the assumption that it otherwise would not be present, so I will need to test my program both with and without the database existing. The setup screen itself contains labels, text input boxes and a button- the labels should display hard-coded text labelling the text input boxes. These boxes should allow the user to input any text they wish from a mobile device; however they often do behave strangely when used on a PC. The input does not need any special validation beyond checking that they are not blank and what Kivy provides by default as the data is not processed, only stored and displayed. Validation of the user's location happens via the weather API in the weather section. The button should add the screens that do not require the database to exist, create the database and its tables then insert the user's data, before adding the remainder of the screens and displaying the home screen. If the database does exist, all screens should be added, and the home screen displayed.

The home screen should display the user's name, retrieved from the database, along with a predefined message in a label. There are then 6 buttons that when clicked should display the corresponding screen: weather, twitter, notes, reminders, or settings.

The main weather screen should retrieve the user's city and country from the database and display it in a label within some predefined text. In another label, the text forecast for the current day at the given location should be displayed. If the location is not found or there is no network connection, an error message should be displayed instead. The high and low forecast temperatures should appear below this, and should also be replaced with an error if the forecast cannot be found. The user should also be able to input a country and city of their choice into two text input boxes and can then click a button to be taken to the "more weather" screen with forecast data for the given location. If the user has not provided a city and country, they will be unable to proceed. There is also a "back" button, that should allow the user to return to the home screen.

The more weather screen, only accessible via the main weather screen, displays 10 days' worth of forecast information in individual labels. This information includes the name of the day, the text forecast, and the high and low temperatures. This information will most likely exceed the length of the screen, so it should be possible to scroll down to see the rest of the labels. If no information can be found for the given location, or the user has entered an invalid location, an error message should be displayed instead. A button should be added to the bottom of the screen, allowing the user to return to the main weather screen.

The main twitter screen is similar to the main weather screen. The last username the user searched for should be displayed above the latest tweet from that username. The default username is @kedst. If the username does not exist or no tweets are found, an error message should be displayed instead. Below this, the user is able to input a username to search for in a text input box and can click a button to begin the search. This also stores the searched username in the database. If the user has not entered a username, clicking the button should not do anything. There is also a button that should display the home screen when clicked.

The more twitter screen is only accessible through the main twitter screen. It displays up to ten tweets from a given username, however if no tweets can be found an error message should be displayed instead. A button should be added at the end of the labels that when clicked displays the main twitter screen.

The main notes screen also follows the same basic structure as the previous two sections – the title of the last note the user created and its content in two separate labels. If no notes are found, an error message should be displayed instead. There are then two buttons: one should display all notes sorted by creation date, and the other should display all notes sorted alphabetically by title. There is also a text input box, allowing the user to input a search phrase, and a button that begins the search. If the user has not entered a search term, clicking the button will not do anything. A final button displays the “create note” screen, where the user is able to input a title and content for their new note. Clicking the “create” button on this screen should add this data to the notes table in the database, but if either title or content has not been provided clicking the button should not do anything. There is also a “cancel” button that should display the main notes screen.

The more notes screen is the basis for the “notes by time” screen, the “notes by title” screen and the “search notes” results. Each note’s titles and contents are added to individual labels. The labels should be created dynamically so there should never be more labels than needed. Because the screen is dynamic in this way, the program should remain stable if no notes are found. Below each note is 2 buttons, one allowing the user to edit the corresponding note and one to delete it. A back button is added to the end that displays the main notes screen when clicked.

The reminders section is almost identical to the notes section, so will require similar testing, with one key difference. When creating a reminder, the user should be able to enter a date and time in a series of text input boxes. When the reminder is created, this should be converted to seconds since the epoch. If a digit is missing, or the user has entered a non-numeric input, the reminder should not be able to be created. The main reminders screen should display the reminder with the oldest date associated to it. The more reminders screen should display the reminder’s title and content and its date, in a standard format.

The settings screen contains a label with predefined text informing the user that some of their changes will not be effective until the app is restarted. There is then a text input box with a label allowing the user to input their name. There is a “change name” button below this, which should replace the user’s name in the database with the given one when clicked. If the user has not input any text, clicking the button should not do anything. Below this are additional text boxes where the user can input their country and city, and a button that should replace the location in the database with

the one given. Nothing should happen if either box is empty when the button is clicked. The text input boxes on this page do not need any additional validation beyond checking that data has been entered because the text is only stored and displayed, not processed in any way. Checking that the user's location is valid should be handled in the weather section. There are also buttons, one displays the export screen, and another displays the home screen.

The export screen contains just three buttons: one allows the user to export their notes, another exports the reminders, and another displays the settings screen. If the user chooses to export, the data should be read from the relevant database table and written line by line to a file at a predetermined location. The email screen should then be displayed, which has text input boxes allowing the user to input their email username, password and the email the message should be sent to. The password text input box should only display asterisks to obscure the password. Below this is a "send" button and a "back" button. The "cancel" button displays the settings screen, and the "send" button should piece the email together with a predefined message and attaching the csv file just created. The app should then authenticate with the Gmail server. If the email cannot be sent for any reason, for example in the absence of a network connection, the user should be returned to the home screen.

These tests will demonstrate the reliability, robustness, and user friendliness of my app by ensuring that the app does not crash and that all of its functions are useable.

Test plan

Test No.	Purpose of the test	Test Data	Expected Outcome	Actual Outcome	Changes Needed	Time stamp
1.1	The app should correctly detect if the user has not completed setup	Run app with no database present	The setup screen is displayed	As expected	None	0:00
1.2	The app should complete the setup process	"Name", "London", "UK" in setup screen	The database is created with the correct tables and the user's data is inserted	As expected	None	0:00
1.3	The user should not be able to complete setup if any of the input boxes are empty	All boxes empty on setup screen	Nothing happens when button is pressed	As expected	None	0:00
1.4	The app should correctly detect if the user has completed setup	Run app with database present	The home screen is displayed	As expected	None	0:00
2.1	The user's name should be displayed on the home screen	"Name" in the name column of the	The name is shown along with a predefined message	As expected	None	0:39

		userinfo table				
2.2	Each button on the home screen should lead to its corresponding screen	Click each button on the home screen	The correct screens are displayed	As expected	None	0:39
2.3	The "back" button on each screen should lead to the previous screen	Click the "back" button on each main screen	The home screen is displayed	As expected	None	0:39
3.1	The main weather screen should display the text forecast and high and low temperatures for the location in the database	"UK" and "London" in userinfo table	The information is displayed	As expected	None	1:14
3.2	The user should be unable to proceed if the text input boxes are empty	Text input boxes empty	Nothing happens	As expected	None	1:14
3.3	The more weather screen should display 10 days of forecasts for a given location	"France" and "Paris" in text input boxes	10 days of forecast information is shown	As expected	None	1:14
3.4	The more weather screen should handle an invalid location	Gibberish text in text input boxes	An error message is displayed	As expected	None	1:14
4.1	The main twitter screen should display the latest tweet from the given username	"kedst" in userinfo table	The latest tweet is displayed	As expected	None	2:07
4.2	The user should be unable to proceed if the text input box is empty	Text input box empty	Nothing happens	As expected	None	2:07
4.3	The more twitter screen should display the previous 10 tweets from the given username	"twitter" in text input box	The previous 10 tweets are displayed	As expected	None	2:07
4.4	The last username searched should be updated in the database	"twitter" in text input box	The new username is shown on the main twitter screen	As expected	None	2:07

4.5	The more twitter screen should handle an invalid username	Gibberish text in text input box	An error message is displayed	As expected	None	2:07
5.1	The most recently created note's title and content should be displayed on the main notes page	"Title 1" and "content 1" in the database"	The title and content are displayed	As expected	None	2:57
5.2	Clicking the "create note" button should display the create notes screen	Click the button	The create note screen is displayed	As expected	None	2:57
5.3	Clicking the "create" button should add the note to the database	"A Title 2" and "A content" in text input boxes	The note is added to the database	As expected	None	2:57
5.4	The user should be unable to proceed if any text input box is empty	Text input boxes empty	Nothing happens	As expected	None	2:57
5.5	Clicking the "notes by title" buttons displays all notes alphabetically	The previous 2 notes in database	The note with title "a title 2" is displayed first	As expected	None	2:57
5.6	Clicking the "notes by time" button displays all notes chronologically	The previous 2 notes in database	The note with title "title 1" is displayed first	As expected	None	2:57
5.7	Clicking "edit" displays the edit note screen	Click "edit" button on "title 1" note	The edit note screen is displayed	As expected	None	2:57
5.8	If any of the text boxes are empty, the user should be unable to proceed	Text input boxes empty	Nothing happens	As expected	None	2:57
5.9	Clicking "edit" updates the note's information in the database	"Title 1", "New content"	The note is updated	As expected	None	2:57
5.10	Clicking "delete" deletes the note	Click delete button on "title 1" note	The note is deleted	As expected	None	2:57

5.11	The "search notes" button should not do anything if the text input box is empty	Text input box empty	Nothing happens	As expected	None	2:57
5.12	The search notes button should display the notes that contain the given search term	"Content" in search box	The "A Title 2" note is displayed	As expected	None	2:57
6.1	The reminder with the oldest date should be displayed on the main reminders screen	"Title 1" and "content 1" in the database, with a date/time in the past	The title and content are displayed	As expected	None	5:02
6.2	Clicking the "create reminder" button should display the create reminders screen	Click the button	The create reminder screen is displayed	As expected	None	5:02
6.3	Clicking the "create" button should add the reminder to the database	"A Title 2" and "A content" in text input boxes, with a date/time in the future	The reminder is added to the database	As expected	None	5:02
6.4	The user should be unable to proceed if any text input box is empty	Text input boxes empty	Nothing happens	As expected	None	5:02
6.5	Clicking the "reminders by title" buttons displays all reminders alphabetically	The previous 2 reminders in database	The reminder with title "a title 2" is displayed first	As expected	None	5:02
6.6	Clicking the "reminders by time" button displays all reminders chronologically	The previous 2 reminders in database	The reminder with title "title 1" is displayed first	As expected	None	5:02
6.7	Clicking "edit" displays the edit reminder screen	Click "edit" button	The edit reminder screen is displayed	As expected	None	5:02

6.8	If any of the text boxes are empty, the user should be unable to proceed	Text input boxes empty	Nothing happens	As expected	None	5:02
6.9	Clicking "edit" updates the reminder's information in the database	"Title 1", "New content"	The reminder is updated	As expected	None	5:02
6.10	Clicking "delete" deletes the reminder	Click delete button	The reminder is deleted	As expected	None	5:02
6.11	The "search reminders" button should not do anything if the text input box is empty	Text input box empty	Nothing happens	As expected	None	5:02
6.12	The search reminders button should display the reminders that contain the given search term	"Content" in search box	The "Title 1" reminder is displayed	As expected	None	5:02
7.1	The "change name" button should do nothing if the text input box is empty	Text input box empty	Nothing happens	As expected	None	6:52
7.2	Clicking "change name" should update the user's name in the database	"Jake" in text input box	The name column is updated	As expected	None	6:52
7.3	The "change location" button should do nothing if the text input boxes are empty	Text input boxes empty	Nothing happens	As expected	None	6:52
7.4	Clicking "change location" should update the user's location in the database	"Paris" and "France" in text input boxes	The city and country columns are updated	As expected	None	6:52
7.5	Clicking "export data" should show the export screen	Click "export data"	The export screen is displayed	As expected	None	6:52
7.6	Clicking "export notes" should export the notes as a csv, then	Click "export notes"	The csv file is generated, and the email screen is displayed	As expected	None	6:52

	display the email screen					
7.7	Clicking "export reminders" should export the reminders as a csv file, then display the email screen	Click "export reminders"	The csv file is generated, and the email screen is displayed	As expected	None	6:52
7.8	If any text input boxes are empty, clicking the "email" button should do nothing	Text input boxes empty	Nothing happens	As expected	None	6:52
7.9	The "password" text input box should display all input as asterisks	Any text in password text input box	The text is displayed as asterisks	As expected	None	6:52
7.10	Clicking "Send email" should send an email with the csv file attached	Click "send email"	The email is sent and received, with the csv file attached	As expected	None	6:52

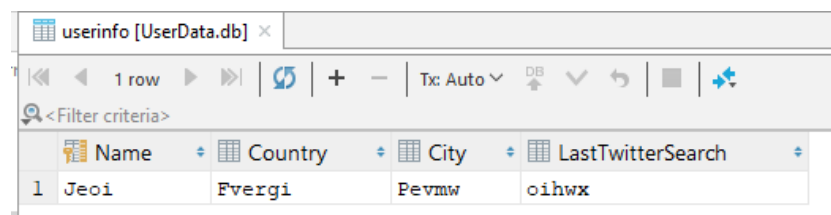
Test evidence

Video footage of my tests on a Windows computer can be found at the following link:

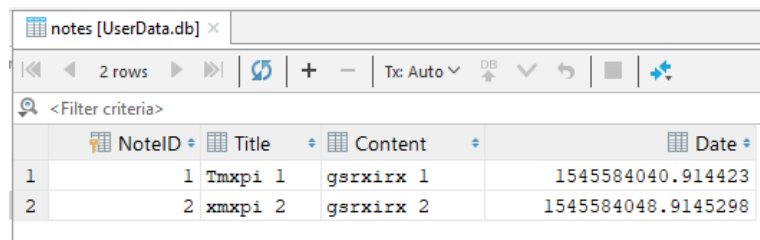
<https://youtu.bsettinse/Hewjw3fMVDc>

I have also repeated these tests on an Android device via the Kivy launcher, with an almost identical version of the program. The csv export functionality, and its testing, was omitted because Android's version of Python does not include the csv module. Other than this, the app worked exactly as above, despite some minor unavoidable display issues that did not affect functionality.

In order to prove that my databases are encrypted, I have included some screenshots of the database:

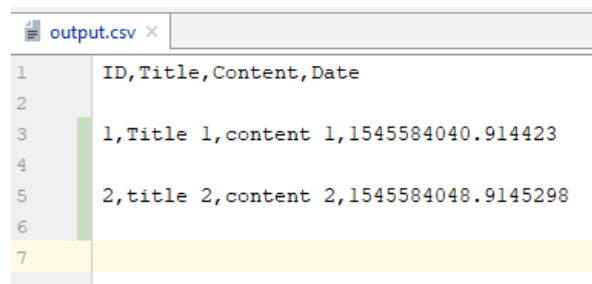


userinfo [UserData.db] x				
1 row				
Name	Country	City	LastTwitterSearch	
Jeoi	Fvergi	Pevmw	oihwx	



	NoteID	Title	Content	Date
1	1	Tmxpi 1	gsrxirx 1	1545584040.914423
2	2	xmxpi 2	gsrxirx 2	1545584048.9145298

In addition, I have included a screenshot of the csv file when the user's notes are exported:



ID,Title,Content,Date
1,Title 1,content 1,1545584040.914423
2,title 2,content 2,1545584048.9145298

Evaluation

Objective evaluation

Objective 1: "The app should be easy to use"

This has been met. The text in my app is large and clearly visible, and the buttons are large enough that they are easy to click. All my buttons and text input boxes are labelled so it is clear to the user what they must do to get to see the information they want.

This objective included the secondary objective of "The app should use simple, friendly language and address the user by their name". I believe this has also been met; on the home screen the first thing the user sees is a friendly greeting addressing them by name and the language used throughout the app is simple.

Objective 2: "The app should show information when it is relevant"

This included the secondary objective "weather information should be retrieved from an online API and displayed when appropriate". This has been met- my testing shows that weather information is correctly displayed from the Wunderground API. I did not need to include any additional logic in my app to display the correct information at the correct time, as by default the API returns the forecast for the current day first. My tertiary objectives, "The user should be able to define target locations", "The app should find new weather information for the current day in the morning and display this all day", and "The user should be able to request weather information for any location at any time" have all been met. When the app is opened for the first time, the user is required to enter a default location for weather forecasts to be retrieved for, and they are then able to change this location at any time from the settings screen. The app refreshes the weather information every time the weather screen is displayed, so information always remains current. Finally, the user is able to see an extended forecast for any city and country that they choose, accessible at any time from the main weather screen.

Another secondary objective, "Reminders should be shown at the correct time", has also been met. When creating a reminder, the user is able to set a date and time for it, and the main reminder screen displays the reminder with the oldest date first, with the intention that the user will delete it when they are finished with it. This included the tertiary objective of "Existing reminders should be searched for ease of use", which has been met because my testing shows that the user is able to search their reminders for any search term.

The third and final secondary objective was "The app should display the last few reminders taken". This has been met: the main reminders screen displays the most recently created reminder, and the user is able to view all of their reminders sorted by their creation date. The tertiary objectives, "The user should be able to view a sorted list of all of their reminders" and "All reminders should be searchable based on the text inside them", have been met. My testing shows that the user can view their reminders sorted by both creation date and alphabetically by title, and that reminders can be searched for any given search term.

Objective 3: "The app should integrate with a social media"

The app does integrate with a social media, and in line with the secondary objective "Twitter provides an easy to use API" it integrates with Twitter. This also included a tertiary objective, "Users should be able to view tweets by accounts they follow" which has been met. The main twitter screen displays the latest tweet from the account the user last searched for. The user can search for a Twitter username and see the last ten tweets from that account, so they are not restricted to just accounts they follow.

Objective 4: "Data should be kept secure"

I believe that the user's data is kept secure, so this objective has been met. This objective had a secondary objective of "As little data as possible should be transferred to the internet". This has been met because only the user's country and city are transferred to the weather API, and only the username they have searched for is sent to Twitter.

The other secondary objective was "Data should be held securely in databases". The user's data is held in a single, locally stored database. This included two tertiary objectives: the first, "This data should be encrypted" has been met because the user's data is encrypted with a caesar cipher to obfuscate the data to anyone who may be attempting to access the files from outside the app. The second, "The user should be able to edit and delete information at any time" has also been met because my testing shows that the user can delete and edit their reminders and reminders and can also change their name and location at any time.

Objective 5: "The app should run as above on both Android and iOS"

It is impossible to verify if I have met this objective. I do not have access to a testing environment for iOS devices, but I have no reason to believe that my app would not work on iOS because I have not used any OS-specific features.

Kivy features a launcher for Android, and my testing shows that a modified version of my app does run on an Android device. However, this version has CSV features removed because the version of python included in Android does not include the CSV module. However, if I were able to package my app as an APK, able to be run directly on Android, I would be able to include this module and all functionality would be available.

Objective 6: "The user should be able to export their data as a CSV file"

I believe I have met this objective. My testing shows that user is able to export both their reminders and reminders as a CSV file. There was a secondary objective of "They should then be able to email the file using their own account to a target email of their choice". I have met this objective - the user is able to send an email with the CSV file attached through their own email address to a target email of their choice, provided that their own email address is with Gmail.

User feedback

Interview

Does the app fit your needs?

Yes. It allows me to look at twitter and stores all my reminders and notes in the same place.

Would you use this day to day?

Probably. It would be useful for organising my meetings and appointments

Any criticism?

If the note is too long, it does not appear correctly on the main notes screen.

Do you have any suggestions for improvements or extensions?

I would like to be able to sign into my own Twitter account, or use different social medias. I would also like a notification for when reminders are due.

Analysis

I interviewed the initial client, Jake. He did not provide extensive feedback, but it was valuable. He reacted positively to the app when given the chance to use it, though was frustrated at some of the display issues when the app is run from an Android device. The criticism given - that some notes do not always display fully – is a limitation of the Kivy app and may be a compatibility issue with the device used for demonstration. This issue does not occur when the app is run from a PC.

Extensions

If I were able to develop my program for more time, I would like to include some additional features that I think would drastically improve the quality of my program.

First, I would like to improve the Twitter section by allowing the user to sign in to their account and view the tweets from all of the accounts that they follow in one place. This was mentioned in the user feedback as a desired extension. I did not include this within my final program because it would require the use of three-legged oauth, which is incredibly complex to implement and beyond A-level standard. However, the inclusion of this feature would make my program more in line the objective of “Users should be able to view tweets by accounts they follow”.

In addition to this, I would also like to include the alarm functionality outlined in my analysis. This would include a third section - similar to the reminders and reminders sections - that would allow the user to set repeating alarms that would ring at a specified time, even if the app was running in the background. I did not include this in my program because it would require the use of an Android emulator in order to access the required permissions, which was not available to me during development. Without these permissions, the alarms would not ring, so the functionality was too similar to reminders.

This was brought up in the user feedback, where my client requested a notification when reminders were due. As with the alarms functionality, this is possible but would require access to a development environment which was not available. I would include this feature given the time and resources.

Finally, I would like to extend the email section to include the ability to send emails from more services than just Gmail. Although Gmail is free, it would be more convenient to the user to be able to use their own personal email address. I did not include this in my final program due to time limitations.