FriendSpot

Rob McNeil
40296075@live.napier.ac.uk
Edinburgh Napier University - Advanced Web Technologies (SET09203)

Abstract

FriendSpot is a micro social media platform which I have developed using Flask that allows users to create an account, make posts and follow their friends among other things. This report will be an overview of the design choices I made, how I developed the flask app, as well as exploring what features I would like to add to the app and evaluating the finished product and myself.

Keywords – Python, Flask, HTML, CSS, Javascript, Web app, SHH, Linux, Hashing, Twitter clone, Social network

1 Introduction

FriendSpot, at it's core, is a social networking app so there are a number of included features that one would expect from a social network. Upon entering the app URL the user will be redirected to the log in page, as a majority of the app is inaccessible to those who are not logged in. From here users can either log on if they have an account or they can create an account. If the user needs to create an account they are taken to a registration form where they must enter their; email address, first and last name, a unique username and a password (which needs to be entered twice so that users know they entered the wrong password). Upon signing up the user's password is hashed, the details are entered into the user database and they are granted access to the rest of the web app.

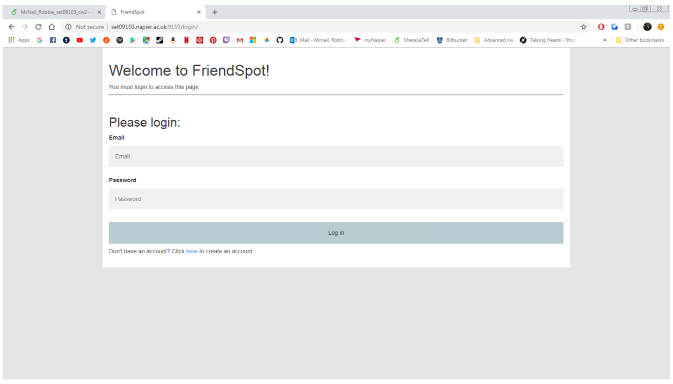

Welcome to FriendSpot!
You must login to access this page
Please login:
Email
Email
Password
Password
Log in
Don't have an account? Click here to create an account

Figure 1: **Login page** - users will be directed here if they aren't already logged in

Once signed in a user has access to all FriendSpot has to offer. The first page they will come across is the homepage. For a new user the homepage may seem pretty empty and boring, this is because the homepage is where the user can view posts from the accounts that they follow. If a user (such as a newly registered user) follows nobody then no posts can appear in the homepage except their own. Once a user starts following accounts their homepage will become filled with the posts of the people they follow in time descending order, meaning new posts show up at the top of the page and the oldest pages appear at the bottom. Users can also make and view their own posts from the homepage. To make a post the user must give the post a title and a body, similar to how posts on Reddit or Tumblr work. Posts will not only display the title and body but also includes other information such as the person who posted it, the user's avatar and the time and date at which it was posted. Users can click on the post's user to be taken to that user's profile.

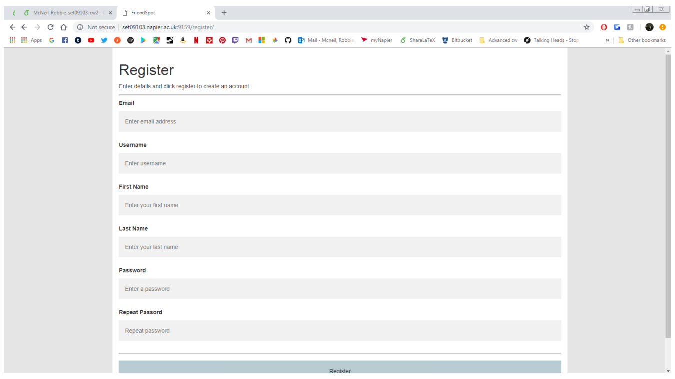
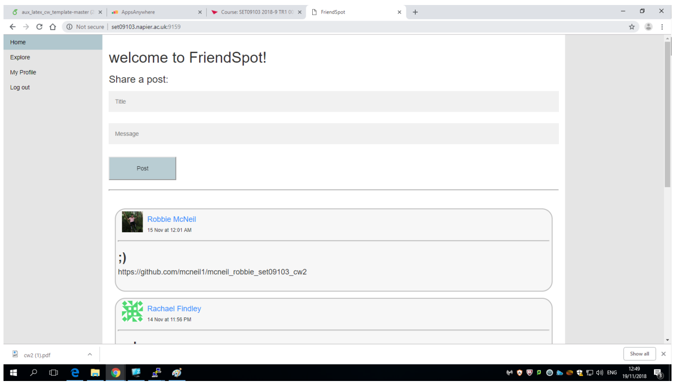


Figure 2: **FriendSpot's registration form**- here you can see the required information needed to create an account

If users want to find new users to follow they can go to the explore page. The explore page is very similar to the homepage with the exception of not being able to make a post from this page. The explore page includes all posts from all users across the app, regardless of whether the user follows them or not. So if a user has just registered and they need to follow some accounts they can go to the explore page and follow users who post content that they enjoy. This page makes navigating the app and finding new users a lot easier as prior to this the only way people could find users was by typing out the URL to the users profile, and if you did not know the user's username finding their page was more or less impossible.

Finally, users are able to access their own profile by navigating to the 'My Profile' section on the top left side of the app. your profile displays your name, username, avatar and your number of followers at the top of your page. Below your details are all of your posts and only your posts. If you are on another user's profile then you will be able to see all of the same information as your own profile except for you have the ability to follow or unfollow that user if you wish. Following/unfollowing a user will automatically update their follower count as well as add or remove that user's posts to your homepage.



welcome to FriendSpot!

Share a post:

Robbie McNeil
15 Nov at 12:01 AM

;)
https://github.com/mcneil1/mcneil_robbie_set09103_cw2

Rachael Findley
14 Nov at 11:56 PM

Figure 3: **FriendSpot homepage** - displaying a form at the top to create posts and followed posts below

Logging out can be done by clicking 'Log out' on the navigation bar, doing so will end your session and you will be redirected back to the log in page where you will be able to see a message telling the user they were successfully logged out. If a user want's to access the site again they will have to log back in.

2 Design

Coming off the back of my last flask web app wanted to improve my next project in a number of ways. Firstly I wanted to put some more emphasis on colour and style as I wanted the website to be more appealing and eye catching. The average person reportedly spends over 2 hours a day on social media[1], so if I wanted people to use my social media platform and continue to use it I would have to make the website look appealing. As well as this users would have to be able to find their way around and be able to use and understand how all the features worked.

This time around the URL hierarchy which I designed is much simpler with fewer total routes. There are the /login/ and /register/ routes, both of which can be accessed without being logged in. Those who are logged in can access the homepage at the base '/' route, the /explore/ route and the /profile/<username>/ route. most of these routes are pretty straight forward with the exception of the profile routes, which the users have to supply a username in order to get to that user's profile. Because this route only works if the username and corresponding user can be found in the user database we have to make sure that any incorrect usernames that are inputted return a 404 error. The same can be said for the /follow/<username>/ and /unfollow/<username>/ routes which are used to follow or unfollow a user and redirects the user to the corresponding users profile. Users must also be prevented from doing actions such as following and unfollowing themselves or users which do not exist.

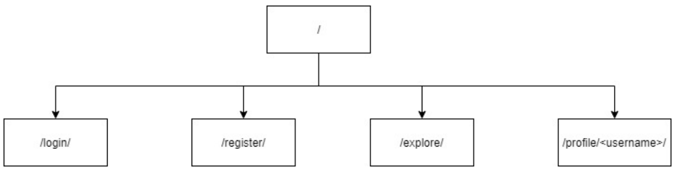


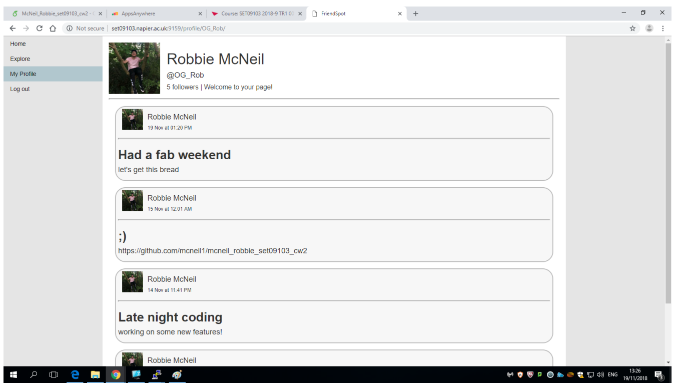
Figure 4: **URL hierarchy of FriendSpot**

Because users need to create an account to access a majority of FriendSpot we need to collect some of their information in order to create an account for them. To keep users safe we need to make sure that all data collected about them is relevant and that any sensitive data must be kept secret in case anybody tries to access this data. We know that it is bad practice to keep any passwords stored as plain text, so we will need to hash passwords at the very least to keep these users safe. When making an account we will need to get the users email address in case we need to contact them in anyway, their name so that other users know who the user is, a unique username so that each user can have their own personal profile and a password. This is all we will need to create a user so it is all we will collect from the user. Sensitive data such as racial origin, political beliefs, sexual orientation and credit card information is not needed and therefore will not be collected or stored[2].

As for the actual physical design of the web app (i.e. what the user sees) I wanted to make sure that each route felt similar, keeping all the pages consistent will allow users to associate each page with FriendSpot. To do this I will be employing the use of HTML templates, there will be a base template which will hold all relevant code that is used by each page, then corresponding page templates will be produced that inherit from the base template. Allowing all page templates inherit from one base template gives all inheriting templates the same look and feel and is extremely useful for making everything appear to be consistent. As well as using Jinja2 templates I am also going to use bootstrap to give the app a bit of polish and personality. To make FriendSpot feel more like a social media platform I'm going to be using some layout techniques such as giving the app a consistent colour scheme, utilizing margins on either side of the screen and including a navigation bar for quick and easy navigation around the site. Users will also be able to use a form in order to post messages to the site that any followers will be able to see on their homepage, these posts will also be posted to the poster's profile and the explore page. Users will be able to navigate to another user's profile by clicking on one of their posts, here you will be able to see all of their other posts and have the option to follow or unfollow them if you wish.

3 Enhancements

Although FriendSpot includes many common features that one would see social media websites there are many features that are missing or not as polished as I would have liked. For starters users have very little in the way of customization of their profile. While users can change their profile picture it is done so through a third party website named Gravatar[3], here users can create an account and upload an avatar, that avatar is then linked to their email account and will be used as your avatar for any website that uses the Gravatar service. This saved myself some time as I did not have to store or upload any pictures, however, it doesn't give the user much freedom to pick and choose new avatars at will. Changing a profile picture not only requires you to navigate to a separate website but also requires the user to create an account at that website as well which is far from ideal for users and may be a turn off point for some users. As well as this users have no way of editing a bio about themselves, which is a common feature for most social networks, this would definitely be a priority in the future as this allows users to make their profile a bit more personal and unique and can make FriendSpot appear more appealing.


McNeil_Robbie_set09103_cw2 - | AppsAnywhere | Course: SET09103 2018-9 TR1 00 | FriendSpot

Not secure | set09103.napier.ac.uk:9159/profile/OG_Rob/

Home
Explore
My Profile
Log out

Robbie McNeil
@OG_Rob
5 followers | Welcome to your page!

Robbie McNeil
19 Nov at 01:20 PM
Had a fab weekend
let's get this bread

Robbie McNeil
15 Nov at 12:01 AM
;)
https://github.com/mcneil1/mcneil_robbie_set09103_cw2

Robbie McNeil
14 Nov at 11:41 PM
Late night coding
working on some new features!

Robbie McNeil

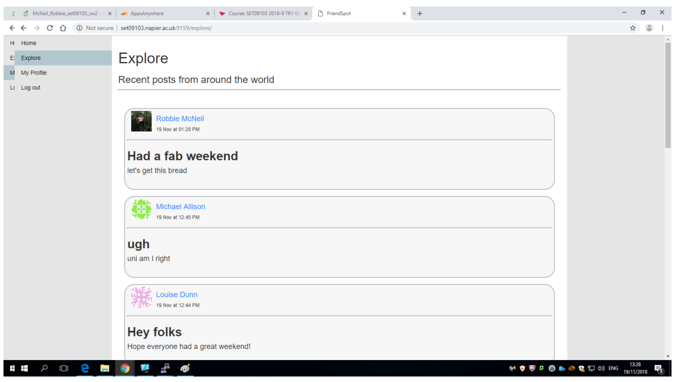
ENG 13:26 19/11/2018

Figure 5: **User profile**

The next enhancements I would make would be the ability to add multimedia to user posts. It's common these days on most social network website for a user to be able to upload images, gifs, videos and other types of media. I believe that allowing users to upload photos will make the web app more attractive to potential users, as this is yet another way for users to communicate and give their profile more personality. The reason that I was reluctant to add this feature and the ability to upload profile avatars from the app from the beginning is due to the fact that I did not want to store images (or any other multimedia content) in a database. In the future if I was to continue developing this project adding the ability to upload multiple media type and edit your personal page would be some of the first improvements made, as I believe these small changes would greatly increase people's interest in the service.

One feature that I was keen to implement and did attempt was instant messaging. Most social media platforms allow users to communicate privately either one to one or in a group via some sort of instant messaging service. In order to implement such a feature I installed SocketIO[4] which "gives Flask applications access to low latency bi-directional communications between the clients and the server". This plugin, if used and implemented correctly, would allow users to send messages to other users and receive messages from those other users as well. However, upon installing SocketIO via pip and adding SocketIO to my flask application the app refused to run. What once worked was now unable to run, so I removed all SocketIO code from my application, but the app still refused to run. I looked into a fix or a reason as to why my app would not run after the installation of the plugin but I was unable to find one. Unfortunately as a final resort I uninstalled SocketIO and my app started to work again, after this happened I placed the instant messaging feature to the side to work on other features. This feature showed promise and is definitely one that should be implemented at some point if FriendSpot would ever be able to compete with the big social networks.

Another common feature that is missing is the ability to like/favourite/up-vote posts. Not only does this allow users to support people and content which they like but it also acts as an incentive for users to make posts. If this system was in place then I could also make improvements to the explore feature of the web app. At the time of writing this the explore page displays all posts from FriendSpot in a time descending order, this means that users will see them most recent posts at the top of the pages and the further down you scroll the older the posts get. Although this does allow users to find other users there is no guarantee that the user will find any users that they want to follow as the posts they see might be spam or just generally uninteresting to the user. A better way to display posts on the explore page would be to have the most popular posts appear at the top of the page (i.e. the posts with the most likes). Preferably users would be able to select a timescale to sort these posts by such as daily, weekly, monthly or all time. This way users are more likely to find the content that they want to find.


McNeil_Robbie_set09103_cw2 - | AppsAnywhere | Course: SET09103 2018-9 TR1 00 | FriendSpot

Not secure | set09103.napier.ac.uk:9159/explore/

Home
Explore
My Profile
Log out

Explore
Recent posts from around the world

Robbie McNeil
19 Nov at 01:20 PM
Had a fab weekend
let's get this bread

Michael Allison
19 Nov at 12:45 PM
ugh
uni am I right

Louise Dunn
19 Nov at 12:44 PM
Hey folks
Hope everyone had a great weekend!

ENG 13:28 19/11/2018

Figure 6: **Explore page** - all posts from across the app are displayed here

Finally a small detail I would like to tweak would be the way posts are displayed. Currently posts are displayed in a box which is used to distinguish where one post ends and another begins. Each post is split into two with user info in the top half and the message appearing underneath. Although this works as intended I would like the border to perhaps be more subtle, at the moment even small posts take up a large amount of space and makes the UI feel clunky at times.

4 Critical Evaluation

The first thing I did when starting this project was to create templates and routes for each page. This meant making a base template that included all common template info such as the bootstrap code and any CSS. It also meant creating forms for login and registration, a homepage and profile pages. I used a colour scheme generating website[5] to find colours that worked well together and didn't come across overpowering or frustrating to look at for long periods of time. I put more effort into the colours used in this project as if users find the app irritating to look at or hard to read they'll likely not use the app agian.

After I had created basic templates for each route I had to get user login and registration working as I knew that I wanted the main website to be locked to non-users. Adding users meant taking inputted user info from the registration form and saving it in a database. At first I used SQLite3, which came pre-installed on the development server, however as databases became more complex and more databases came into play I decided to use the SQLAlchemy plugin instead[6], this plugin has a number of handy features that came into play later in development. Before a users info is saved to the database some info has to be validated. Firstly each user must have a unique email and username, as the email is a primary key and the username is linked to the user's unique profile. Secondly user's must type their password twice to validate that the inputted password is correct. Lastly each textbox must be filled out. If any of these requirements are not met then the account will not be made. Otherwise the user's password is hashed and the details are saved. To hash the user's password I used a plugin called Flask-Bcrypt[7] as there were no other password hashing plugins installed on the development server. Flask-Bcrypt allowed me to easily hash passwords, store the hashed passwords when a user created an account and compare hashed passwords against inputted passwords on the login page. With these feature implemented users could create an account or log in if they already had an account, their data would be stored in a database and their sensitive info would be hashed so users would not have to worry about anybody stealing their password. The account creation and logging in feature works well and as intended, information is validated to make sure no invalid data can be inputted and users can easily make an account and then log in with their credentials.

Next I had to implement a login-required method that would make sure that anybody who is not logged in would be able to gain access to the protected parts of the web app. I was able to do this with Flask-Session[8], which allows us to use server-side sessions. When a user logs in certain information is used by the session to validate that the user is indeed logged in. As well as this I used a website in order to generate a secret key[9] which is used to keep client-side sessions secure. I picked a strong password which was 'robust enough to keep your web hosting account secure', this ensured that nobody would be able to guess the secret key and users would be secure when using FriendSpot. By adding the login-required method to all relevant routes we are able to prevent unauthenticated users from accessing protected routes.

Now that users can log in view the website I had to add content to the pages. In order to store user posts I had to create a posts database. Each post has a unique number as well the title, body, time at which it was posted and the email address of the author. The database had to be a one-to-many database as one user can have many posts, so the author's email address is saved to link the post to the user who wrote it (the email address is used as a foreign key). I could not find any documentation on how to make a one-to-many relationships with SQLite3, however SQLAlchemy had a lot of documentation for these types of relationships which made it seamless to create this second database once I had switched from SQLite3 to SQLAlchemy. Once users were able to store posts in the posts database I had to print the contents of the posts database to the homescreen in time descending so that users could see the posts they were making and that others were making. This worked well but at the time all the user posts were printed to the homepage feed, even posts which users may not want to see, which was not ideal.

User profiles did not require much work to create as most the work was already done. The user database searches for the username inputted in the URL and returns the user. If the user exists then the user profile is filled with the returned users info including avatar, name, username and posts, if the username is not found then the user does not exist and we are redirected a custom 404 error page. After the user profile route could successfully generate a profile based on a given username I wanted to add some extra functionality in the form of following other users. This required a many-to-many database that linked users to other users. This table stored two values, the followers email and the email of person being followed. Once I had updated the databases and their methods I was able to add a follower count to each profile that would update automatically when a user clicked the users follow/unfollow button. Clicking the buttons would take you to the /follow/<username>/ route or the /unfollow/<username>/ route that would direct you back to the inputted user's profile if the user existed or direct you to a 404 error page if the user did not exist. Now when users visited the homepage they would only see the posts from people they followed and I believe this feature over any other makes FriendSpot seem most like a social network and it gives users the ability to see what they want to see.

The inclusion of being able to follow and unfollow users allowed me to change the homescreen to only show followers posts, solving the problem of unwanted posts showing up on users homepage. However, this also meant that any new users would have no posts to look at when they joined and no real way to find new users. To combat this I implemented the explore route, this used a very similar layout to the homepage with the exclusion of the form used to add new posts. The explore page needed to be a place for users to find other users so I made the explore page display all of the user posts rather than the homepage. Although I've already discussed how this feature would be better if posts were displayed based on popularity and not time posted, it does allow users to find each other.

Overall the features that FriendSpot include all work and they all appear to work reasonably well, however, some missing features do end up holding the app back in certain areas. As well as this the design isn't the best in a couple of areas despite the added emphasis on appearance this time around.

5 Personal Evaluation

When I started this project I had just come off the back of my previous Flask web app, which was used as a place for me to write music reviews. My last Flask project was quite frankly a bit boring, I used the default font through out, the only colour came from images and the app was generally pretty static. This time around I wanted to be more engaging with the user, I wanted to create something that users could interact with and would want to interact with, hence why I opted to develop a social media platform. Even though there were a number of features that I would have liked to have included which did not make the cut, either because I could not get them to work or I had no time to implement the feature, there were a number of features that I did manage to include.

A small difference between my previous work and FriendSpot is the inclusion of bootstrap. This time around I followed the Bootstrap example from the module workbook in order to give the website a more consistent and interesting look rather than just using default fonts like I have done in the past. This change was small and fairly easy to implement however it does make a difference, in order to have the bootstrap on every page I put the code in the base.html template which all other templates inherit from, which made the process even quicker and easier.

Because this was a very different direction from my last web app I had to research and use many new techniques and tools. First of all I had to utilize multiple databases to store different types of information and although I have worked with databases in the past actually implementing a database into a piece of software I was developing was a first for me. This is one area in particular that I was having trouble with during development as I could not get the many-to-many table needed for the following function to work for quite some time, however this was a feature that I desperately wanted to include and eventually I found an solution. SQLite3 was clearly not the best tool for the job so instead I installed SQLAlchemy, which included documentation which was extremely useful in guiding me towards a solution as it includes example code for both one-to-many (useful for linking users database to post database for posts) and many-to-many relationships (useful for linking user database to itself for following and unfollowing).

The Inclusion of a user base and the ability to make new accounts and have users log in and be able to view locked content was a feature that I tried to implement in my last web app that was not necessarily successful. Because I wanted to include this feature in FriendSpot I did research into how it might be done early into development. First thing I found was that most people recommended using Flask-login which is reportedly useful when you have a large user base, and although it may have been easier than other options I wanted to try use tools that were already at my disposal. This lead me to look into sessions, SQL and data hashing so I could not only add new users but I could also keep their data safe and secure.

When I was using the web app I found that the app would crash if you tried to go to a URL for a user that does not exist, this was because the user profile page would search the user database for that username and generate the profile base upon the information in the database. If the user did not exist then the page could not display information relevant to that user and would crash. Similarly the same would happen if you tried to follow or unfollow a user that did not exist. To prevent this crash I used created a custom error 404 page. If a user attempted to go to the page of a user that did not exist (or try to follow/unfollow that user) they would be redirected to the 404 error page, which prevents the crash from happening.

Despite learning about new tools and overcoming certain problems there were some problems that I couldn't figure out. I previously spoke about home I wanted to add instant messaging and how the Flask plugin that I installed to help me do this caused problems for me. I searched online for solutions and read the user documentation for a solution or explanation as to why installing the plugin made the app refuse to run, even without any code relating to the plugin being present in the app itself. This is one problem that I wanted to fix however, given that I could not find an answer and I still had other feature to implement I made the decision to scrap the feature for the time being as the app would be no use at all if it wasn't able to be run. Although I wasn't able to find a solution, moving away from the feature was, in my opinion, the best thing to do at the time.

Overall I think I worked well on this project, doing relevant research into the tools and features I was able to use definitely helped a lot in the beginning of the project for giving me a place to jump off at. Despite some mishaps along the way I managed to achieve most of what I set out to do, and although this may not mean that everything included in the web app looks perfect, everything that is included works as expected. There are some design choice I may have made differently if I did the project again and there are plenty more features that can be added in the future to make FriendSpot even better, but there is always room for improvement. I used the resources at my disposal to overcome a majority of the problems that were thrown my way, such as getting databases to work with each other, and put more thought into the finer details such as keeping users secure, the colours used and why users might use the app in the first place. However, given the time I had to get this web app off the ground I am proud of what I was able to produce. Based on my last project it feels and looks more professional, it's more polished and had more thought put into it.

References

[1] E. Asano, “How much time do people spend on social media?.”

[2] “Gdpr sensitive and non-sensitive data: A distinction with a difference.”

[3] “Gravatar.”

[4] “Flask-socketio user documentation.”

[5] “Colour scheme generator.”

[6] “Sqlalchmey user documentation.”

[7] “Flask-bcrypt user documentation.”

[8] “Flask-session user documentation.”

[9] “Random key generator.”