Annex Social Network App

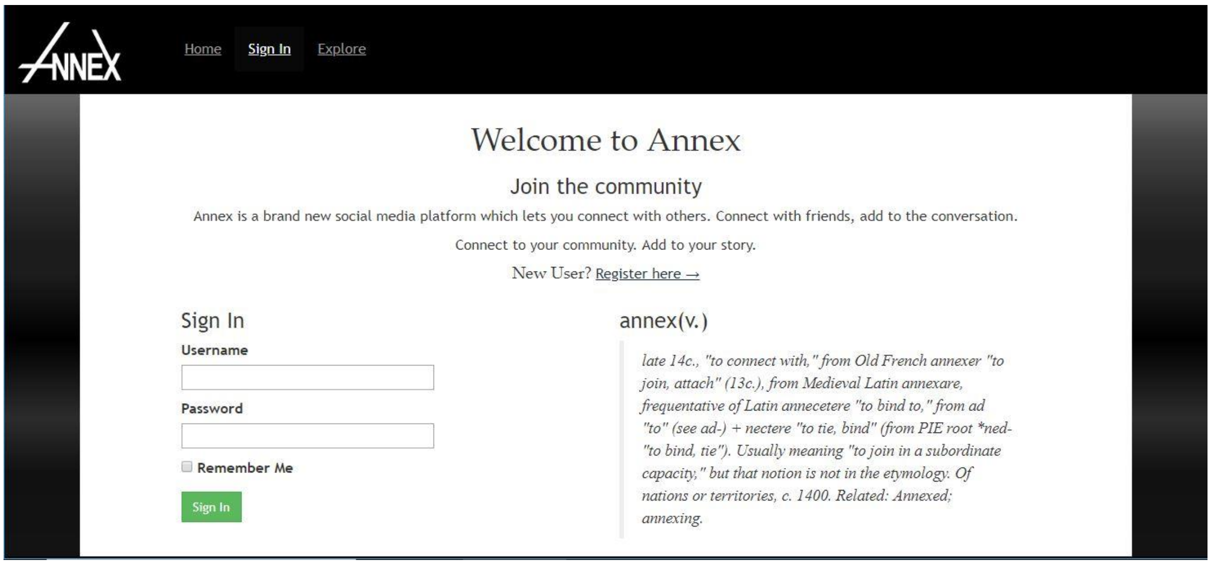
Christine Griffiths

40402000@live.anpier.ac.uk

Edinburgh Napier university – Advanced Web Technology (SET09103)

Introduction

The Annex Social Network App is a simple Twitter clone which allows registered users to share text-based posts with other registered users, users can be 'followed' and 'unfollowed'. Built using the bootstrap 3 framework and HTML templates, dynamic content is delivered to the user via a fully responsive interface. The application is database driven with a secure login and registration system, the database also delivers dynamic content in the form of posts, user profile data and a feed of posts which can all be edited according to the user's preferences The Python Flask microframework is used to organise and manage the resources for the app, such as packages and functions, which are then served as required.



Welcome to Annex

Join the community

Annex is a brand new social media platform which lets you connect with others. Connect with friends, add to the conversation.

Connect to your community. Add to your story.

New User? Register here →

Sign In

Username

Password

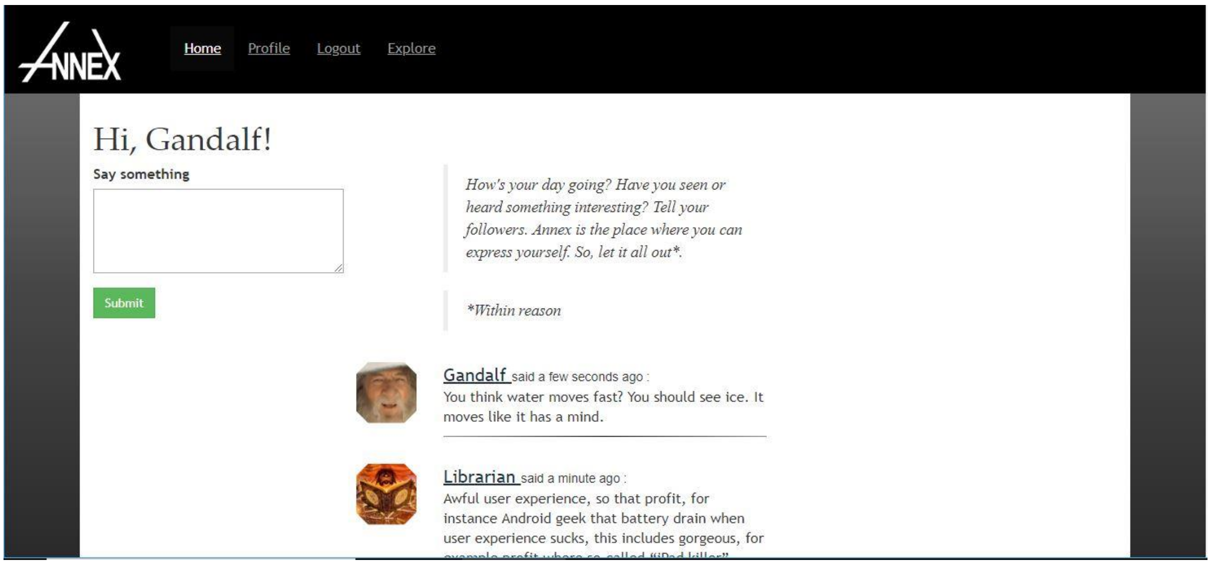
Remember Me

Sign In

annex(v.)

late 14c., "to connect with," from Old French annexer "to join, attach" (13c.), from Medieval Latin annexare, frequentative of Latin annecetere "to bind to," from ad "to" (see ad-) + nectere "to tie, bind" (from PIE root *ned- "to bind, tie"). Usually meaning "to join in a subordinate capacity," but that notion is not in the etymology. Of nations or territories, c. 1400. Related: Annexed; annexing.

Figure 1 Index page – defaults to login page



Hi, Gandalf!

Say something

How's your day going? Have you seen or heard something interesting? Tell your followers. Annex is the place where you can express yourself. So, let it all out.*

**Within reason*

Gandalf said a few seconds ago :
You think water moves fast? You should see ice. It moves like it has a mind.

Librarian said a minute ago :
Awful user experience, so that profit, for instance Android geek that battery drain when user experience sucks, this includes gorgeous, for

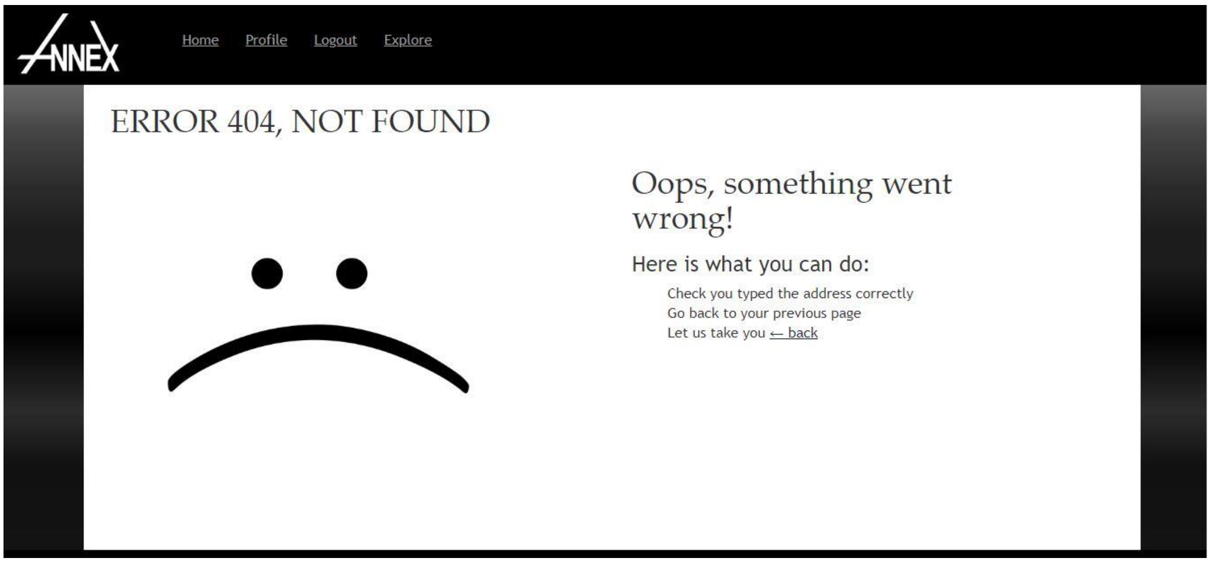
Figure 2 Index page – user logged in

Design

The application archetecture has been designed to produce, as much as possible a Rich Internet Applications (RIA). This means the server works as a web service or application programming interface (API), and its primary function is to retrieve and store the data which will be used for the client application.

The protocol with which the application communicates with the web service is designed to follow Representational State Transfer (REST) architechture. 'REST is a very popular approach to building APIs because it emphasizes simplicity, extensibility, reliability, and performance'. (F., 2017) The Flask microframework's root() decorator (combined with the methods argument where necessary) made creating a RESTful API quite simple. All of the resources, in this case resources are posts and users, have a unique identifier, as the application is built with HTML this means a unique URL. The URL identifiers can then be used by the client application with the request method to complete operations. The application's URLs are designed to aid the client in finding new resources, for example: user/'username' (where username is that of any user) will take you directly to that user's profile page.

When requests are sent from the client a response is returned from the web service in the form of an HTTP status code. Should these status codes be the error 404 or 500 the user is provided with error handler pages, rather than the defaults provided by Flask, which offer them information about what went wrong and how to resolve the problem. 'Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution'. (Nielsen, 1994)



ERROR 404, NOT FOUND

Oops, something went wrong!

Here is what you can do:

Check you typed the address correctly
Go back to your previous page
Let us take you ← back

Figure 3 - Error 404 with instructions for user

The protection of information, particularly user data, is provided via user authentication in the form of a login system. This system is facilitated by Flask-Login which stores the user's data client side (in a cookie) meaning the authentication system complies with the RESTful requirement of being stateless, or essentially not storing client data between requests.

The design of the application's interface was intended to aid the user in carrying out the tasks they wished to complete when using it. The navigation is simple and clear and with the active item of the navigation showing where the user is in the application, the user should never feel lost. The application uses consistent fonts, colours and layouts, so the user recognises the app immediately and that they are still within it. 'Users should not have to wonder whether different words, situations, or actions mean the same thing'. (Nielsen, 1994) A logical hierarchy of the page elements means the user can find what they are looking for immediately, headings describe page content, white space emphasises content and similar elements are grouped into columns and rows using Bootstrap's grid system.

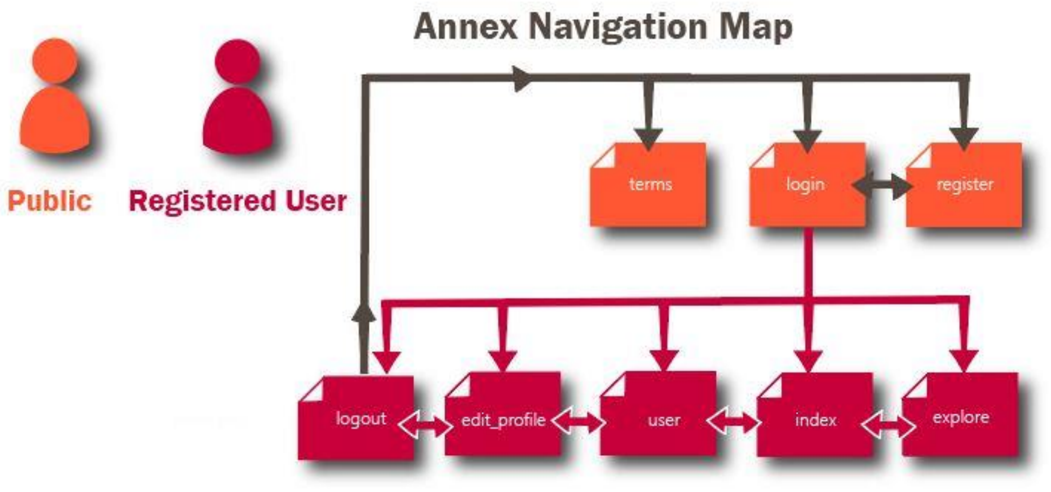


Figure 4 Navigation Map

Enhancements

Originally it was planned that the application would be far more complex than it is; particularly with more interactions between users. A comment system allowing users to comment on one another's posts would be a great enhancement as it would allow more user interaction and more user generated content, increasing user engagement.

A search bar should have been part of the navigation bar, allowing the user to search for other users, such as acquaintances or people they were interested in hearing from.

Email authentication of users would be a great enhancement, improving security. Email password reset tokens would also be an enhancement, as it is users have no way of resetting their password.

The app's profile pictures are currently avatars provided by the Gravatar avatar service. It would be good to have an image uploader for users to choose their own profile pictures. However, the storage and validation of image files would have to be carefully considered.

Unit testing would automate testing and save time testing manually as well as providing a simple way of checking that newly added code does not stop older code functioning the way it should.

Blueprints could have been implemented to improve the simplicity of the application's structure and a better organisation of the application's resources.

Critical Evaluation

Dynamically generated content is implemented well in the application. Jinja2's template engine is used to generate HTML with inheritances being implemented correctly. Storage and serving of data using a database, means only the data requested by the API is served, saving and managing resources for a better running application.

The URL paths are well designed, meaning the user can find the information for individual resources quickly, easily and intuitively.

The user authentication, login, works well in restricting access to personal information and not allowing access to the application by any unauthorised parties.

A JavaScript function is used in pages with a lot of content to allow the user, upon scrolling, to return to the top of the page by clicking a button which is fixed to the side of the content. This is a useful feature for returning to the top of a long page and to the navigation.

Although effective in allowing the user to carry out the tasks they may wish to perform, the user interface is overly simple and lacking in content. The interface does not make use of any of bootstrap's themes, which could have improved the overall appearance.

Personal Evaluation

Personal expectations when beginning this coursework were far too high. Expecting to be able to produce a professional-level application which was ready for deployment after only two months of using the Flask Python microframework was far from realistic. During the building of the application it quickly became clear that there were limitations to just how professional it could be, such as personal experience, knowledge, skills and time. Many frustrations were encountered whilst trying to implement an application package factory and blueprints into the application, particularly errors which hadn't been encountered before. As such a lot of time was wasted trying to make the application package factory work correctly and eventually the idea was abandoned, but not before most of the development time had been used.

Initial research was time consuming, leaving less time for the development of the application. However, deeper understanding of object orientated programming, building RESTful applications and server-side technologies in general is invaluable, not only in completing this project, but also in future projects and employment.

The application meets the coursework criteria, but is a personal disappointment, in that it is far simpler than planned in design, structure and features.

Table of Figures

References

F., X. F. (2017, 12 21). *How to design a RESTful API architecture from a human-language spec*. Retrieved from O'Reilly Learning: https://www.oreilly.com/learning/how-to-design-a-restful-api-architecture-from-a-human-language-spec

Grinberg, M. (March 2018). *Flask Web Development (2nd).* O'REILLY.

Jinja2. (2008). *Jinja2 Documentation (2.10)*. Retrieved from Jinja 2 Documentation: http://jinja.pocoo.org/docs/2.10/

Nielsen, J. (1994). Enhancing the Explanitory Power of Usability Heuristics. *CHI '94 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, (pp. 152-158). Boston, Massachusetts, USA.

Smyth, P. (2018, 4 2). *Creating Web APIs with Python and Flask*. Retrieved from The Programming Historian: https://programminghistorian.org/en/lessons/creating-apis-with-python-and-flask

Appendices

Bootsrap 3 was used an HTML framework for creating the user interface. Version 3 was used opposed to version 4 as it is far more stable and more widely supported and documented.

Flask-Login - a small extension used to manage user sessions for logged in users Install the extension with pip:

```
$ pip install flask-login
```

Flask-SQLAlchemy was used to simplify the use of SQLAlchemy within the application. SQLAlchemy was used to simplify the code and save repitition of almost identical code. SQLAchemy also handles database changes well, without having to completely rewrite code. Install the extension with pip:

```
$ pip install flask-sqlalchemy
```

Flaks-Migrate - a lightweight wrapper for using Alembic database migration framework in Flask. Supports a variety of databases, especially useful in deployment. . Install the extension with pip:

```
$ pip install flask-migrate
```

Flask-WTF – automises the creation of web forms, saving time and repitative code. Validation features are useful in removing the tedious, repetative and error-prone process of coding validation. Install the extension with pip:

```
$ pip install flask-wtf
```

Flask-moment - integrates Moment.js into Flask for handling times and dates in the browser and rendering with JavaScript. Install the extension with pip:

```
$ pip install flask-moment
```