LINKshrink- A URL Shortener

Ross Langan
40276526@live.napier.ac.uk
Edinburgh Napier University - Advanced Web Technologies (SET09103)

Abstract

LINKshrink is a simple URL shortener. Providing a way to transform long URLs into short URLs, LINKshrink can be used for both home use and by professionals and boasts user accounts and an aesthetic design. In this report I will cover the design, implementation and evaluation of my web app, and include the extentsions I used to help me.

Keywords – Flask, Python, SQLAlchemy, Bootstrap, Web-app URL, Web, Recaptcha, Hashlib, Hashing, Security, Users

1 Introduction

LINKshrink is a URL shortening service providing users with a way of shortening URLs and is particularly useful for very long URLs. The user can paste the link of the target website into LINKshrink, which in turn generates a much shorter link. When the generates URL is used it will redirect the user to the target URL. The service is particularly useful for links used in social networking, SMS and email where there can be limits on the amount of characters.

The LINKshrink homepage is the main functional page of the web app and contains the input box used for inputting the target link and the output box for displaying the generated link. From the hompeage toolbar users are also able to reach pages to sign up and login to the app.

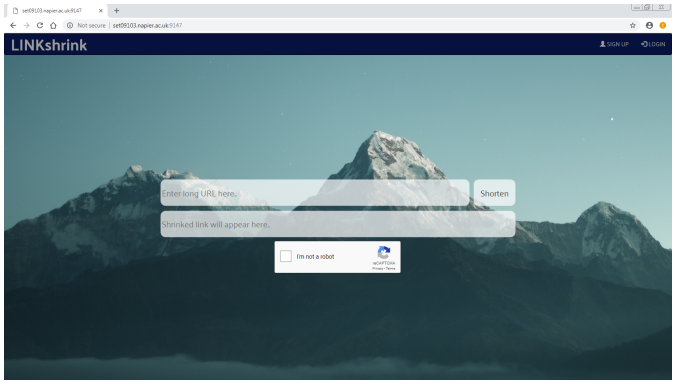

Figure 1: **LINKshrink homepage** - Aesthetic design containing URL input, output and navigation bar

Signing up to LINKshrink is easy and requires only a valid email address, username and password. When signing up the user must select a username that does not already appear in the system. Logging in to the app is even easier and only requires the username and password, both of which must appear in the database. I have made this possible by utilising Flask sessions. Sessions allow me to determine whether a user is logged in or not and direct them to the appropriate version of each page. For added security I have used Bcrypt [1] to hash the user's password. Unlike algorithms like 'MD5' and 'SHA1', Bcrypt adds extra security as it is designed to be slow and very hard to decrypt quickly.

LINKshrink also utilises Google's reCAPTCHA [2] service as an added security measure to prevent bots and automated services from spamming the web app. Users are required to complete the reCAPTCHA in order to generate a shortened link. If the reCAPTCHA is not completed then an error message will appear with the reason why. Users with an account who are logged in are directed to an exclusive homepage that does not require any reCAPTCHA completion.

The link generated by the system consists of the basic LINKshrink URL with a four character code on the end as seen below, and can be used by anyone who knows the link.

When the user wishes to shorten a link they must paste of type the link into the input box. Once the link has been entered and the reCAPTCHA has been completed the user must click the 'Shorten' button. This generates the new link which is output in the box below the input box. The generated link consists of the base link and a four digit combination of uppercase letters, lowercase letters and/or numbers. The four digit combination is generated at random and is unique to the target link. Following this link will always redirect the user straight to the original URL. After the URL has been shortened, the reCAPTCHA will disappear, and the user can click the 'Again?' button as seen below to reload the page and enter a new URL.

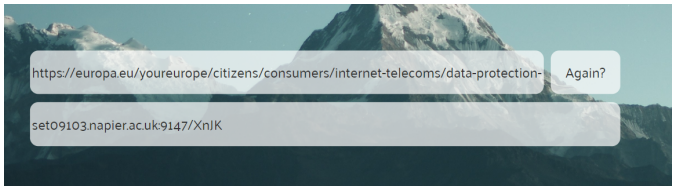

Figure 2: **Long and Short URLs** - the input box contains the old URL and the box below contains the new generated link

2 Design

2.1 Page Structure

Unlike my previous web app 'TV List-ting' I have not used a hierarchical structure for my pages. Because my web app provides a service and is not an informative website featuring many products or categories it has a very limited amount of separate pages. This means that the user will only ever be on the homepage, registration page or log in page. In researching similar websites I found that on some sites, the URL shortener is one of many services provided. The URL shortening service on some of these websites is only one single page. I designed my web app around this idea that the functionality should be mostly, if not all, on the same page. My initial design for this can be seen below.

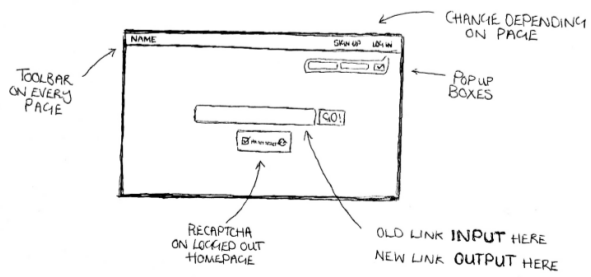

Figure 3: **Original Page Structure Plan** - features pop up log in and sign up fields

After deciding that I should include a separate page for the log in and sign up features, I chose against using a base template of which each page would inherit parts from, rather choosing to have four individual templates- one for each page. I decided against the base template because the bootstrap nav bar contains different buttons on every page, and the log in and sign up pages contain different fields for data input. The only part each page that is the same for the whole website is the background image.

Each of the four templates used contains HTML for the bootstrap toolbar, the background image and the appropriate text boxes. The CSS for the textboxes is contained in a stylesheet located in the static folder.

The pages are designed in a way where the user can reach the homepage from any page, and when logged out, access either the log in or sign up page. When logged in the user should only be able to reach the log out page.

2.2 URL Structure

The URL structure of my web app is based around a homepage with a registration and log in page accessible from the toolbar. The homepages appear differently depending on whether the user is logged in or logged out as a different template is rendered depending on the state. To show this I have included two separate homepages on the diagram.

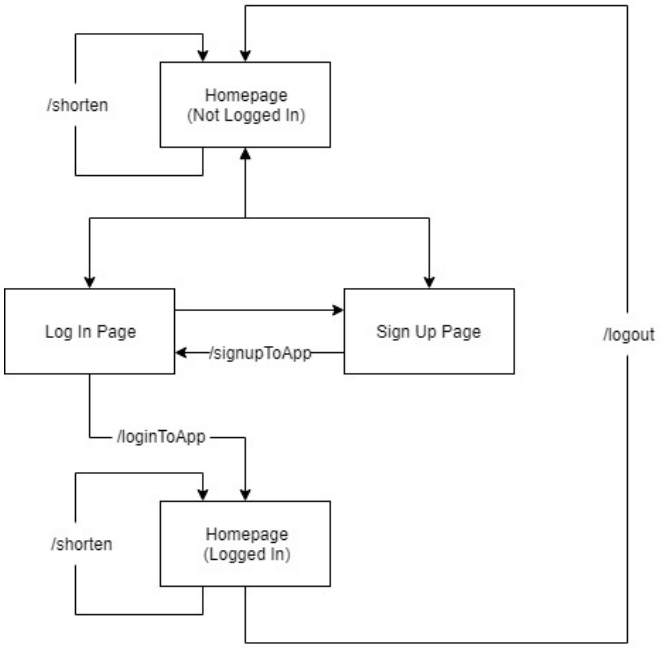

Figure 4: **URL Structure** - Possible paths that can be taken

The homepage of my web app for both logged out and logged in users is located at

set09103.napier.ac.uk:9147

This is the base URL for my web app and is the shortest and most basic URL that I can have right now. This means that is is easier for users who might navigate to my web app by typing in the address to the address bar.

When the user shortens a URL, the '/shorten' link is loaded. This route performs the security validation for the re-CAPTCHA, checks for a valid URL and generates the new link. Once these tasks have been completed, the user is redirected back to the homepage where the old link and new link will be visible. To the user, clicking the 'Shorten' button will appear just like refreshing the page and the shortened link showing up in the output box. The user is never able to see the '/shorten' page as such because it does not render any template.

Similarly when the user signs up to the web app or logs in, the processing of the entered data is performed in the '/signupToApp' and '/loginToApp' routes respectively. Once the data is processed the user is redirected to the next page. If the data is not valid, the page will be rendered with an error message below the fields.

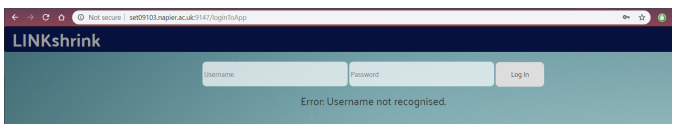

Figure 5: **Log In Page** - has returned an error because the username is not valid

When a user logs out of the web-app, the '/logout' route is loaded. This sets the 'logged_in' session to false and redirects the user to the homepage for logged out users. To access the

logged-in homepage, the user will have to log in to the app again.

2.3 Visual Design

To approach the visual design of my web app I researched and used several of the main URL shortening services including bit.ly [3], bit.do [4] and textmagic.com [5] which utilises Google's shortning service goo.gl. A common theme of these webpages as expected is a centered input box and button labelled 'Shorten'. Two of these sites offer user account services and paid features for tracking link usage and link management. Bit.ly in particular has a business-oriented look to it and features a motto 'Harness Every Click, Tap And Swipe'. It also contains links to its 'Enterprise' and 'Beyond the Basics' feature for paying users and businesses.

In researching smaller URL shortener projects I was able to find many more appealing web apps such as Shortify- a Python Flask URL Shortener [6]. Shortify features a striking high-res image as the background of the homepage and has a very personal, modern feel to it. Containing other features such as a To-Do list, weather information and the time, Shortify really feels like a modern hub for all the essential things a user might need.

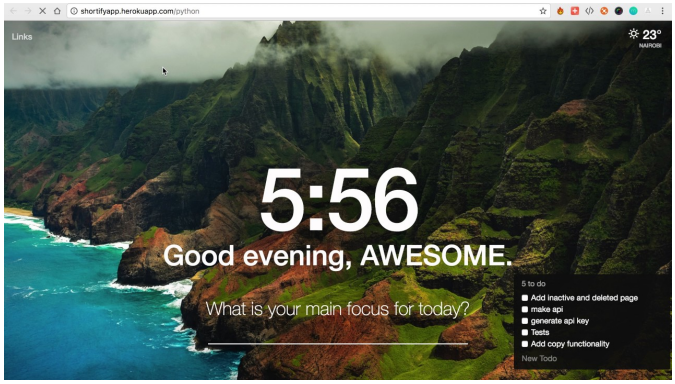

Figure 6: **Shortify Homepage** - A github project by user Awesome94

In designing my homepage, I first utilised the bootstrap toolbar for easy navigation around the site. It contains the name of the web app and links to the sign up and log in pages. I chose to use a dark blue colour scheme for the web app for a calm and sophisticated feel.

The background of my web app is a high-res blue/grey mountain scene. This picture is very calm and elegant and the toned-down colours allow for dark text to still be visible when in semi-transparent boxes. I chose to use this image after being inspired by the look and feel of the Shortify app.

In order to allow the input of the URL and the output of the shortened URL my web app features two large text boxes in the center of the screen. They appear semi-transparent complementing the aesthetics and the 'Shorten' button fits very nicely in the corner of the two textboxes. The placeholder inside the textboxes are grey, and the text when entered/generated is black, complying with the colour scheme.

Also important to the visual design of my homepage is the reCAPTCHA. This form is positioned just underneath the textboxes, and must be completed for the URL to be shortened. The reCAPTCHA is positioned here so that it is obvious to the user that it must be completed, and is in the same general area as the shorten button, reducing the distance the mouse has to travel.

The errors returned on the main page from an incomplete reCAPTCHA or invalid URL appear at the top of the page. They fit in with the colour scheme but can still be noticed easily when the page is loaded, and do not get in the way of the functional parts of the web app.

The log in and sign up pages include text boxes at the top of the page for data input. The text boxes on these pages appear at the top of the page just under the navigation bar. The reason for this is because the user navigates to these pages using the nav-bar, so the mouse is already going to be at the top of the screen. Having the boxes here increases the speed of using these pages and reduces the distance the mouse has to travel to use the boxes.

2.4 Security and Privacy

In order for my site to be secure and respect the privacy of its users, I have tried to implement some basic security features. I have deliberately avoided asking the user for more than their email address, a username and password at the sign up stage. This is so that I am not collecting unnecessary information that I do not need, such as address, mother's maiden name etc. If there was to be a data breach, the less personal information they can take the better.

In order to secure the password, I have used Bcrypt. Using this extension allows me to store ONLY the hashed password, and not the actual user's password. When the password is entered at the login stage it is hashed again, and the two hashed passwords are compared.

When a URL is entered for shortening, no security measures are required as this is not sensitive or personal data, and at this point is not attached to the users account at all. Because my application takes in user information it would be a sensible improvement to add a policy page, stating the policy for using and collecting the user's data.

3 Enchancements

A possible improvement to my web-app would be adding additional features such as link management tools. This would include the ability to view previously shortened links, delete shortened links in your history and view the amount of times the link has been clicked. Similar websites such as bit.ly offer these features, often as a paid upgrade of the service. A further improvement to this would be plotting the location of clicks on a map. This improvement is good because it gives users a way to retain previously entered links and gives users control over their links. Additionally, it is a good way of persuading users to sign up to the app as this feature can be restricted to registered users.

For registered users it is important to be able to view your account details. This should include information that the web app is storing on you- in this case the user's email address and username. Users should also be able to change the password

associated with their account, or reset the password if they have forgotten it. Expanding on this, in order to be GDPR compliant [7], my web app should ask for consent to store the user's data during the sign up process. This particular enhancement is essential.

A way of making the web app more personal is allowing users to create custom shortened URLs. As an alternative to the generated 4 characters at the end of the route, the user would be able to type in any combination of numbers and/or letters (e.g. set09103.napier.ac.uk:9147/FOOD). This would allow users to make the link relevant to the page such as 'FOOD' for a restaurant's website or 'shop' for an online shop. To implement this feature, it would be important to allow for more than 4 characters for custom links because each combination can only be entered once and with a large user-base users will attempt to use popular combinations regularly. This can be seen below on bit.do when I tried to enter 'FOOD'.

Figure 7: **Bit.do** - Chosen custom link has already been assigned to a long URL

The final change I would make to my app is the ability to enter a second URL to the input box immediately. I ran into issues trying to submit a second URL without reloading the page and added an 'Again?' button to solve this. However the ability to generate a second new URL without the action of clicking the again button would save time and clicks and improve the user experience, especially when the user is converting multiple links.

4 Critical Evaluation

The basis of my application before even thinking about the logged-in and logged-out features is the actual URL shortening. In order to shorten URLs it is essential to have a database which stores the original URL and the generated short URL. To achieve this I used the SQLAlchemy extension [8]. I chose to use this extension as it simplifies achieving common database tasks such as adding the databases and updating them. Because I did not do the web technologies course and I have never used python or html before creating my last web app, creating and using a database to support my web app was new to me. After careful consideration I decided to use SQLAlchemy and feel that I have used this extention to its full potential. I query this database several times to find particular values from rows from my URL database.

In order to add the user accounts, I had to create a second database. This database is called 'Users' and includes a column for the email address, username and hashed password. Creating this database was more simple because I learned how to create the first URL database. I query this database more than the URL database in order to log users in and sign them up to my web app.

To hash the user's passwords I used the BCrypt extension. I initially used the hashlib extension [9], however after experiencing problems using this, I switched to BCrypt which not only boasts better encryption but has simple documentation explaining how to use it. Using this extension helps makes my site more secure.

An additional security feature I added was reCAPTCHA. I had intended on using this feature from the start to help protect against bots and assure that the user is human. I think this feature works well with my web app because it performs just as expected and adds a more professional look to the site.

In order to allow the user to reach the logged in or logged out page I have utilised Flask sessions. Flask sessions works by giving the user a unique session cookie allowing them to reach certain pages when the session is set to certain values. I chose to use flask sessions as I have seen the use of this in several examples of user accounts in Flask applications and could not find an alternative that suited my needs. Flask sessions has worked well for me and since implementing it early on in the development, I have not ran into any problems with it.

The homepage of my website looks simplistic and modern and I am very happy with the way it has turned out. I think the toolbar works very well for allowing the user access to each different page. The input and output buttons fit perfectly next to each other and the captcha also fits in with these features very well. The background is not too harsh on the eyes and because it is dark colours and all very similar colours, it does not steal the attention away from the main purpose of the website.

The positioning of all the elements on the screen was made possible using CSS. To use the CSS I created a stylesheet and referenced it in my templates. The CSS has vastly improved the look of the elements on screen and without using this I think that the web app would have looked terrible.

5 Personal Evaluation

Working on my URL shortener was rewarding as I was able to build on the skills I learned in my last project. I was able to implement the basics of the web app much faster than I was with my previous app and could achieve more by using external extensions and add-ons.

At the planning stage of my project I chose on a dark blue and grey colour scheme. This later changed to just dark blues. I am really happy to have chosen this colour scheme because it looks good and is easy on the eyes. The background was also a major factor in this decision as I found the image around the time I was deciding on the colour scheme. I think that

I performed well on this visual aspect of the web app and I have produced a very aesthetically pleasing web page.

In choosing the extensions to use I feel like I have made the right choices and chosen the most appropriate and relevant ones to use. In order to make each of the extensions work I had to refer to their documentation and find examples on github. The biggest problems I faced were with reCAPTCHA. In order for the user to be verified the app must have a secret code provided by Google. This was not well stated in the documentation and for a long time I was unable to make it work properly. Aswell as this I found it difficult to position it on screen as it behaved differently to the input boxes for some reason. Eventually I was able to overcome these problems and the reCAPTCHA is now fully functional.

A simple error I had which cost me a huge amount of time was getting my stylesheet to work. I initially had the stylesheet in the same folder as the templates, and spent days changing the link statement at the top of the templates and changing the CSS in the sheet. I eventually found out that the stylesheet must be located in the static folder. This cost me a lot of time which could have been spent implementing further features and I could have found the solution faster if I had known what to look for.

A problem I had with my last coursework was not working fast enough. With this coursework I was able to work considerably faster because I am more familiar with putty and vim, and my skills in python and html have developed a lot further. There were parts of my program that I could have spent less time working on, such as positioning everything on the screen, however I feel like my time management for this project has been fairly good.

For the next time I would try and focus more on implementing features and adding new pages than the design of each page. My web app performs the task it was meant to do, however it would have been better with some link management tools or custom links.

Overall I feel that I have performed well on this task by creating an attractive, functional web app. I have supported my app with databases and security features and have managed to implement the ability to have user accounts. Along the way I have ran into several difficulties however I feel like I have achieved more than I thought possible at the start of this course. I have also thoroughly enjoyed being able to create my own web apps exactly how I designed them.

6 Appendices

6.1 SQLAlchemy

To install SQLAlchemy I used 'pip install SQLAlchemy'.

After installed, I created a file called 'tabledef.py' which stores the definitions for the table. It is here I create the users and url tables. When this file is ran, it creates the database, named 'database.db'. This is where all the database information is stored.

The documentation for SQLAlchemy can be found at https://www.sqlalchemy.org/

6.2 BCrypt

To install the BCrypt extension I used 'pip install bcrypt'.

After installed I use 'bcrypt.generate_password_hash()' and feed in the users password to generate a hashed password. In order to compare hashed passwords I use 'bcrypt.check_password_hash'.

6.3 reCAPTCHA

To install Google's reCAPTCHA I used 'pip install python-recaptcha'. After installed I had to visit the reCAPTCHA website to register my webapp and get a secret key. I then configure the reCAPTCHA in my code using

app.config.update({'RECAPTCHA_ENABLED': True, 'RECAPTCHA_SITE_KEY':'6LcEunkUAAAAAFHhAGgUWR2gFXdKlV7w
'RECAPTCHA_SECRET_KEY':'123456789secretkey'})

This includes the site key and secret key which are important to validate the user. The full documentation can be found at https://www.google.com/recaptcha/intro/v3.html

References

[1] "https://flask-bcrypt.readthedocs.io/en/latest/,"

[2] "http://bit.do/https://www.google.com/recaptcha/intro/v3.html,

[3] "http://bit.ly/,"

[4] "http://bit.do/,"

[5] "https://www.textmagic.com/free-tools/url-shortener,"

[6] "https://github.com/awesome94/shortify,"

[7] "https://ico.org.uk/for-organisations/guide-to-the-general-data-protection-regulation-gdpr/lawful-basis-for-processing/consent/,"

[8] "https://www.sqlalchemy.org/,"

[9] "https://docs.python.org/2/library/hashlib.html,"