Project 2

Technical Report

December 13, 2013

Team
Derek Kan
Lazar Stojkovic
Morgan Wallace
Andrew Win

Intro

Our team differentiates our project with styling and functionality. Our approach was to implement a MySQL database to provide a relational database backend allowing our minifier to add, edit and remove links and users with ease.

Early on, we added a theme of a magic monkey that minified the links for you. This theme carried throughout the duration of our project and affected the structure of our web app. For instance, we kept the web app to one page using AJAX so that we don't have to stop the monkey, which is moving, and reload him.

Database Implementation of MySQL

Problem

Pickle files are inadequate for the needs of a URL shortener where each user needs to maintain a list of their own links. It become increasingly difficult to work with these pickle files as they get larger.

Example

MySQL allows our shortener to let a user sign-up or log in and add links that will be saved for their return.

Technology

Python, MySQL, Harbinger server (where the database is hosted), MySQLdb (the specific library that we used in our implementation of app.py)

Challenges

At first we tried to use SQL Alchemy, a python library and extension to Flask, to ensure that our database inputs were sanitized. However, the API proved exceedingly difficult so we decided to rely upon MySQLdb which is a Python library that Derek had used before.

Even then, sanitizing the inputs took a bit of digging through documentation.

Alternatives

MySQLAlchemy

Future

The database could be expanded to include a guests table to allow users who did not want accounts but wanted to make a short url. Currently a user must have an account to user the url shortener which is a limitation or our current database implementation.

Delete Links

Problem

Needed to remove links

Example

Shortened URLs that were saved often become useless to the user so we needed a way to eliminate them from both the front-end and the database.

Technology

jQuery was used in the frontend to remove links from the DOM and an AJAX request is sent to the route “/delete” in the Flask app to remove that link from the ‘links’ table in the MySQL database.

Challenges

It was at first difficult to do this from the front-end and backend at the same time since they are separate actions. We had to make sure that both actions were performing successfully in all cases to make sure we could implement this in the final version.

Additionally, we had to re-bind the $(“.delete”).click to any new links that were added to the page since they were also added using AJAX. These new elements were not reacting to the .click method until we realized we had to add the .click function and all it’s contents after prepending to the links list.

Alternatives

The alternative to this would be to just route to the “/delete” in Flask and then have that delete from the database and then re-render the page. Thus, taking the user away from the page and fetching the documents again just to remove one item.

Future

Might be better to streamline the way that the deletion is implemented such that there is only one JavaScript function that called for delete instead of multiple

Get Page Title of Shortened URLs

Problem

Show something more informative than the long url in the list of shortened URLs.

Example

If we shorten the link “http://jblomo.github.io/webarch253/slides/Project2-Deliverables.html” we would get the title, “Project2-Deliverables” and then display that on the UI for that item.

Technology

Python (libraries: urllib2, lxml), HTML, MySQL, Flask
Once the user submits the form to shorten a URL the Flask app function for POST requests for “/shorts” calls a function called “get_url_title” with the argument of the full

url. This function uses urllib2 to open the long URL and then it parses it with lxml.html

Challenges

Encoding of many titles is in UTF-8 which uses icons, like a youtube URL where the user is playing a video shows this”▸ Sensor Fusion on Android Devices: A Revolution in Motion Processing - YouTube.” When Python get’s this, it throws and Unsupported Character error. This is resolved on my local machine by doing this complicated thing to the title which is saved as a string:

```
title.decode("utf-8").encode('ascii',"ignore")
```

However, you must also add this to the top of the script

```
# -*- coding: utf8 -*-
```

Even then we still got a few encoding errors; so, as a last resort, we added a try/except to make sure that if it erred then at least it would have long url as the title.

Alternatives

Alternatives to this would have been to use Beautiful soup but I don’t think that would have solved our problems with encoding.
Also, we could just leave the title as the long url but that doesn’t make for the best user experience - URLs are informative but not nice to read.

Future

In the future, we would display the Long URL underneath the title, just like google search results.

Styling

We wanted to give our application a distinct flavor. As its basic functionality revolves around an almost magical shortening of entered URLs, we have decided to make quirky wizardry our central theme. Medieval myths often talk about familiars - demons attending and obeying sorcerers, often said to assume the form of an animal - so we decided to make one of these our mascot. In particular, we decided it should be a monkey holding a scroll and that is how the name Magic Monkey Minifier was created.

Since the monkey was unquestionably the most important non-functional element in the app, we decided to pay extra attention to it. The goal was to give it a spooky and bizarre - almost taxidermic - appearance and yet let our users know it was alive. In order to achieve the effect, the monkey had to blink and wag its tail. We also wanted to make sure it is not the only animated thing in the page, so we decided that a candle in the darkness and a logo which evaporated into smoke would additionally enhance the atmosphere of supernatural. Of course, we also added some static elements, such as medieval Tarot cards and Runes.

Animating the monkey and the candle was very straightforward. Using Photoshop, we have created each of the necessary states, saved them as separate images, created respective CSS3 keyframe animations, set those up in the stylesheet, and made them loop indefinitely.

The evaporating logo was a little bit more technically complicated. First we put each of the letters in the logo in its own <span> tag. Next, we created a CSS3 keyframe animation of smoke using text-shadow and made the contents of each of the <span> tags evaporate into smoke at different rate - anywhere between 1 and 1.4 seconds - and fade in minimalistic app credits whenever the user hovers his mouse over the logo.