Webarch253 – Final Project Technical Report

Team Members: Evie Phan, Shreyas

Features that our group implemented in the URL shortener project are: (1) database, (2) Bulk URL shortener, (3) ShortURL Removal, (4) URL bundling, (5) stats, and (6) mobile interface.

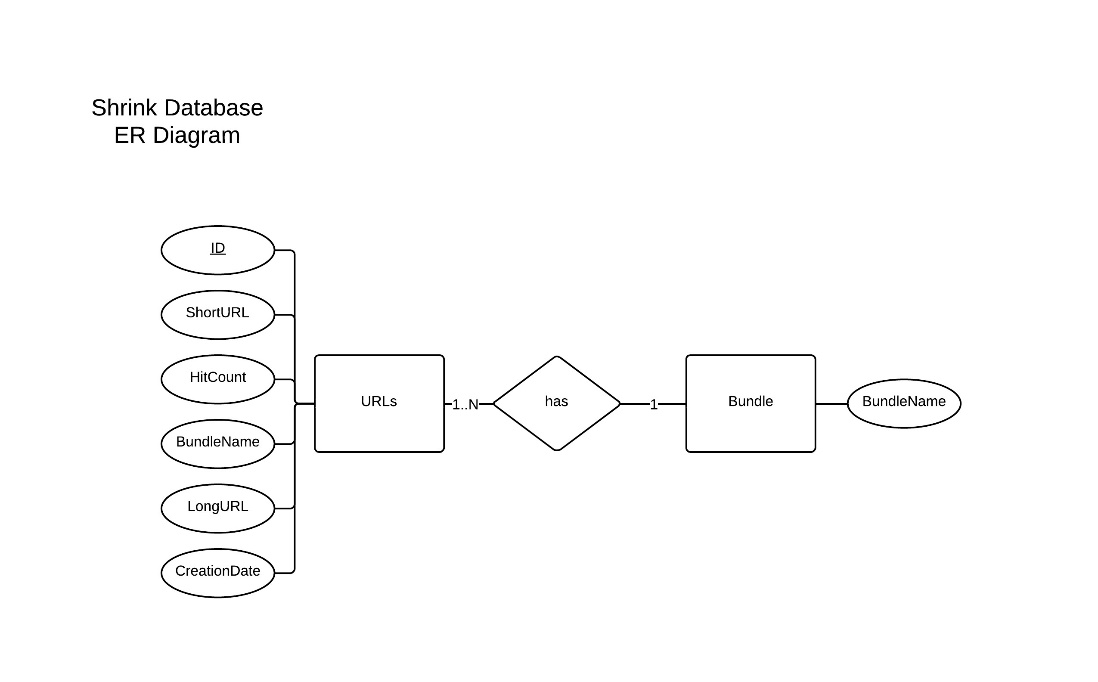
**Database**.

The database stores records that were submitted through the back end. Without the database the short urls would not be saved after the python script finishes running. After creating the environment for the database, we connected the server to Navicat for MySQL to create the table to store information in. However, we then switched over to Postgres with Heroku because we found that it was easier to update information. The only hurdle we had with creating the database was learning how Postgres works, as we never used it before.

We used the development practices as discussed in the last lecture like virtualenv and generating requirements.txt for easy collaboration.

The challenge was in creating a local instance to work with and in setting up a remote database for production. Both were implemented and managed using Heroku. Other challenges was the learning curve of Postgres which both of us did not have any experience in. Working with ACID property was a challenge which was different from the shelve module that was implemented in part one.

We set up an EC2 with a mysql instance initially which we were more familiar with, but in the end rejected it for the ease of use of Heroku as a platform in terms of development and deployment. This is the link to our work: <http://sshr-shrink.herokuapp.com/home>. The following ER Diagram is a representation of what our database looks like.



**Bulk URL shortener**.

The Bulk URL shortener allows a user to convert multiple long urls into short urls in one submission. This is a great feature because it saves time for the user if he/she has more than one long url to shorten. To illustrate further, a user may click on the plus sign (+) on the right of the text input fields to add urls to shorten. Once the user is ready to generate the short urls the user can then click submit. Bulk inputs allows an unlimited number of inputs. We implemented an AJAX based bulk form submit for creation of multiple short urls. The second problem was populating the bundle fields because there was no restrictions on the bundles. Bundles could be chosen from an already existing list or a user can choose to create a new one. Some problems we faced were based on maintaining database sessions. We rolled back the changes that we had made in the session. Currently we are loading separate pages for response, ideally the bulk url creation could be asynchronous.

**Short URL Removal**

If the user no longer wishes to maintain a short url, we allow him to remove it from the system. This is accomplished by deleting the relevant database record. This feature is simple to implement compared to others, but it is very important to users. One concern we have is with security. What if a user attempts to delete another user’s short url? We don’t have a solution to this, but an improvement could include individual accounts that control access to urls.

**URL bundling**.

The URL bundling feature allows a user to organize his/her short urls into categories. This helps the user keep short url links organized and easy to locate. To add urls to a bundle, a user may create their bundle in the text field that indicates bundle. The bundle feature has an auto suggest feature so if the user types in a bundle that already exists in the database a recommended bundle will show for the user to select from. From a database point of view, we created two separate tables and used JOIN to work with them. A problem we overcame was the Object Relationship Model for doing database queries via Flask SQL Alchemy. We highly recommend reading the material before proceeding.

**Stats**.

The stats feature allows the user to see the hit count of all existing short urls in the database. This feature allows the user to see how popular a given short url is based on the number of times a short url was asked for by a browser. A user may see the hit counts for each url next to the short url link. We can do that because each time a website visitor types a short url in his browser, our web server knows about it (since it has to redirect to the correct url). Therefore, we can collect information about the number of hits without any additional HTML or Javascript code in the web page itself.

**Mobile interface**.

The mobile interface feature allows for the website to scale to a cell phone or tablet. We optimized and tested for different mobile resolutions and tried to keep the page size small (for people on slow connections) by using icon type faces instead of images for icons. We used html5 boil plate templates for laying out the page.

Testing was a challenge because there are many mobile devices to test on.

Future improvements could include applying front end speed optimization as discussed in class.