Patrick Canny
Ellis Springe
Liam Ormiston

Landscrape Sprint Log

Overview
The primary goal of the Landscrape web application is to provide users with information on the best things around them. What differentiates our application from competitors is support for a variety of search queries spanning several popular searching sites in order to assist users in developing a list of top attractions in the area. This allows for a simple application that can be easily accessed and utilized by many users.

Project Planning
Development of our application will adhere to the agile methodology and the principles of iterative design. After our initial prototyping sprint (2 weeks) we will focus on one-week sprints with the goal of adding a single feature with each sprint. We will utilize both a gantt chart alongside of the Busybot Slack integration to manage task delegation and progress tracking.

Sprint 1 | Prototyping | 10/22-11/5
Prototype Goals
The prototype of our application will focus on baseline functionality, that is focusing on a single query and scraping results from a single site (Yelp).

We chose to use the Flask microframework for development of our application. Flask is a bare-bones web development framework based in Python. Flask allows for a high level of flexibility and customization for small-scale applications. We knew that our web-scraper would most easily be implemented with Python as an engine, which is a primary reason that we went with Flask. We also wanted the option to use ReactJS for our frontend, which is not as easily supported in a more full-featured Python framework such as Django. Though our prototype doesn’t use React for the frontend, we are hoping to add this in a future iteration.

Prototype Features
Landing Page: The landing page of the application is simple, and allows the user to directly enter and search quickly and easily.

Search Page: Barebones search page with a string entry field. The query entered in this field is then thrown to the Python backend, which handles all the heavy lifting.

About Page: Documentation Landing Page

Search Functionality: The search function is implemented in Python, and queries the Yelp API in order to return results and display them to the frontend.

Navbar: We learned about HTML templating in development of this application, and utilized templating to allow for a persistent navbar layout on each page.

Sprint 2 | Optimizing, Adding Small Features | 11/5-11/15

Sprint Goals

1. Optimize and Improve Scraper Function
2. Add a Loader to the UI
3. Support form addition for multi-query support that's more intuitive
4. Style Output

Updates

1. The Scraper is now far more efficient thanks to Ellis' efforts. The time to load a query is noticeably shorter. We decided to display less information and add a div option to show more information.
2. Pat added a simple loader to the UI using jQuery and a custom Pac-Man .gif.
3. Form addition proves to be fairly complicated with WTForms. Pat worked for a while on getting an option to duplicate an instance of a form with little success.
4. Liam is still working on styling the output appropriately.

New Tasks for Sprint 3

1. Add another site to compare Yelp results to, and integrate some sort of data analytics to rank results accordingly.
2. Loader is pretty good, but possibly modify the gif or link to a cool game
3. Pat will either work on completing this or reassign to Liam
4. Liam or Pat will finish this ASAP
5. Create Scraper Unit Tests
6. Determine some way to test the frontend

Sprint 3 | Finalization of Processes and Deploy | 11/15-11/30

Sprint Goals

1. Add another site to compare Yelp results to, and integrate some sort of data analytics to rank results accordingly.
2. Loader is pretty good, but possibly modify the gif or link to a cool game
3. Multi-Query Input that's pretty intuitive
4. Style Output
5. Create Scraper Unit Tests
6. Determine some way to test the frontend

Updates

1. Ellis completed and integrated Foursquare comparison on 11/30/17. The Data Analytics were fairly simple, as we just looked at "stars" from each site to compare the results
2. Loader stayed the same, despite looking into using an open-source html5 Pacman implementation. The limiting factor was the jinja templating engine

3. We implemented this as a cue-text box, since using form addition/subtraction would require some fancy WTForms knowledge. Overall, it's a pretty good solution. We also added options for input of a city and state which impacts the result of our query.
4. Liam worked on getting the output styled appropriately with a nice earth-tone theme. It looks pretty sweet!!!
5. Pat integrated the Scraper unit testing suite using Python Unittest. The advantage of this is a pretty well-defined process of testing that makes a lot of sense. We essentially tested for expected output of the scraping function on input, and that all the expected values were present.
6. The UI isn't as testable as we would like, but we are able to limit the possible inputs of users with WTForms validation techniques. This limits the possibilities for bad inputs.
7. Pat successfully deployed the application to Heroku, so our site is now live and able to be used by anyone with a desktop computer.

Final Comments/ What We Would Implement Next

1. The biggest issue with our platform is its speed. Since we are running a curl/analyze process on each search result, this takes quite a long time, and varies depending on the internet connection of the client. Obviously, this doesn't really scale well to addition of sites. We came up with a few potential solutions to help mitigate the problem:
    a. Look into some sort of **cloud-based solution** that runs processes on multiple machines at the same time while collecting the results from each appropriately.
    b. Utilize a **database solution**: save results of searches to a common db, and search the db for matching results before moving into a curl. This actually helps the product scale really well with a higher number of users, since more possible values will be added to the db. The downside is that if information changes on the sites, the database will not reflect that information. This could potentially be solved by a database migration on a monthly basis in order to ensure the most accurate results.
2. Our Application is currently deployed to Heroku, but has some issues that would need to be solved in order to scale a little nicer.
    a. Burger Bar Doesn't Work - This is an oversight of working on the web side since day one and not prioritizing mobile use
    b. Timeout on "Is Landscrape Complete?" due to server side issues. This could be solved with a watchdog timer of some sort within the application or on the Heroku side.
    c. Again, speed is reduced due to a non-local hosting environment.
3. Profile Support - For this version of the app, we wanted a platform that was as simple and fast to use as possible for any user. In the future, it might be interesting to implement some sort of profile feature in order to produce better/faster results for users. In scaling, we could also add additional features that would improve the user experience on the app.
4. UI/UX can always be improved.