UPlanner, the Event Digest:

Data Extraction and Machine Learning to Simplify Locating Campus Opportunities

Introduction

As an undergraduate student of a large university, there is no shortage of emails to read, events to attend, people to meet, organizations to join, or competitive opportunities to pursue. The inundation of this information often comes at a price; students, faculty, or staff may face exposure to an excess of information that simply is irrelevant to them. Furthermore, organizers of events, clubs, and scholarships may fail to properly connect information of their event to the proper audience at hand.

To address this problem of this excess of information sources of campus opportunities, the web application UPlanner was created. UPlanner examines various sources of information, including calendars, emails, and web pages, to identify, extract, and categorize opportunities on campus, storing this information in a database management system (DBMS). Users can then access this platform to view events from the database catered to their preferences. UPlanner uses web scraping with machine learning, automated job scheduling, relational database management systems, and a user approval process and front-end web application to simplify finding campus opportunities.

UPlanner was first conceived for the OHI/O 2016 hackathon[1] at The Ohio State University. The 24-hour project served as a prototype of a product that would allow users to find personalized campus opportunities in a single place, rather than burdening students with finding resources themselves from a myriad of sources. As the Secretary of the STEM Exploration & Engagement Scholars, a professional and academic development cohort, I was manually examining several calendars, emails, and websites a week in order to produce the weekly calendar[2] and newsletter. To ease the process of writing the weekly newsletter, I wrote a web application[3] in JavaScript to "scrape" the events from the calendar's API service. However, the tedious process of finding events, scholarships, and other opportunities to add to the calendar led me to recognize the problem of this overabundance of university resources. Consequently, I chose to address this problem at the OHI/O hackathon.

[1] HackOHI/O 2016 - http://hack.osu.edu/2016/
[2] STEM EE TeamUp Calendar - http://go.osu.edu/stemeecal
[3] DCD Generator - https://github.com/wustep/stemee

The marketing and business side of the application is addressed in [Stephen Pioro's paper]. This paper seeks to outline the development and design process of the project, describing the technologies used and the design choices that led to the current build of the application.

Methods

One of the initial considerations of rapid prototyping is selecting the technology stack. I chose Node.js[4], React[5], Express[6], and MySQL[7]. Node.js was chosen due to its increasing popularity and success in building highly scalable web applications[8]. React, Facebook's component-based JavaScript Library, was chosen due to its speed in creating browser-rendered user interfaces. Express is the Node.js go-to tool for website routing and building APIs, and MySQL was chosen due to familiarity and its reliability over many decades in the DBMS sphere.

With this tech stack in mind, I chose to develop the platform in the following manner:

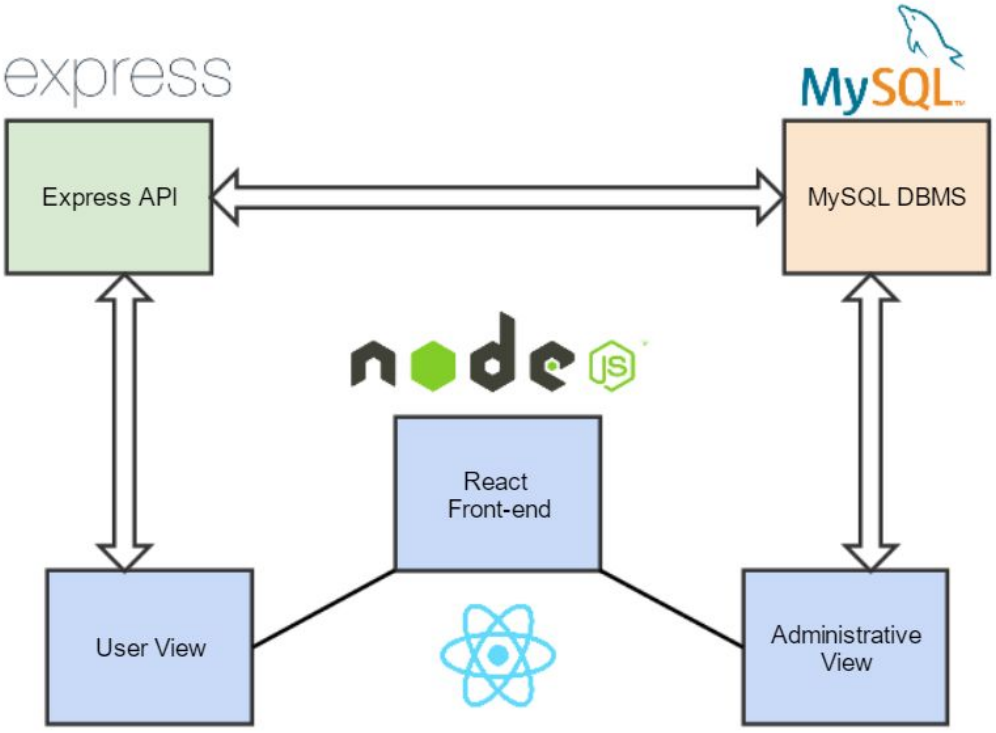


Figure 1. Platform diagram, describing tech stack and relationship between views.

[4] Node.js - https://nodejs.org/en/
[5] React - https://facebook.github.io/react/
[6] Express - http://expressjs.com/
[7] MySQL - https://www.mysql.com/
[8] Tilkov, Vinoski: Node.js - http://steve.vinoski.net/pdf/IC-Node.js.pdf

Users would only see the React Front-end User View, which pulls data from the Express API, which pulls data from the MySQL Database. The Administrative View is used to populate and manage the database with events scraped from external resources, as pictured right. In the hackathon, the sole resource scraped was the Teamup calendar service.

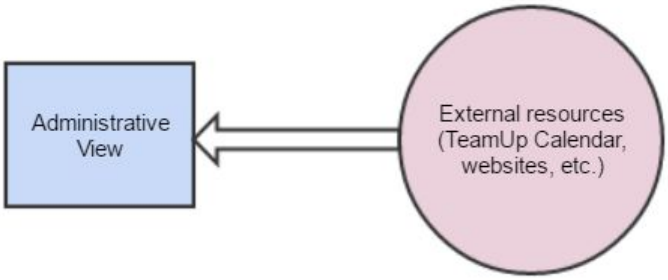


Figure 2. Scraper diagram.

The MySQL database was developed during the hackathon to look like the diagram in *Figure 3*. Since rapid prototyping was a priority, constraints and relationships were not completely defined. Security issues, like password encryption, were also not initially addressed in the model. Each event is given tags (like “Mechanical Engineering” or “Music” or “Art”) and these tags are related to each other (“Computer Science” is closely related to “Electrical Engineering”) to personalize events to users.

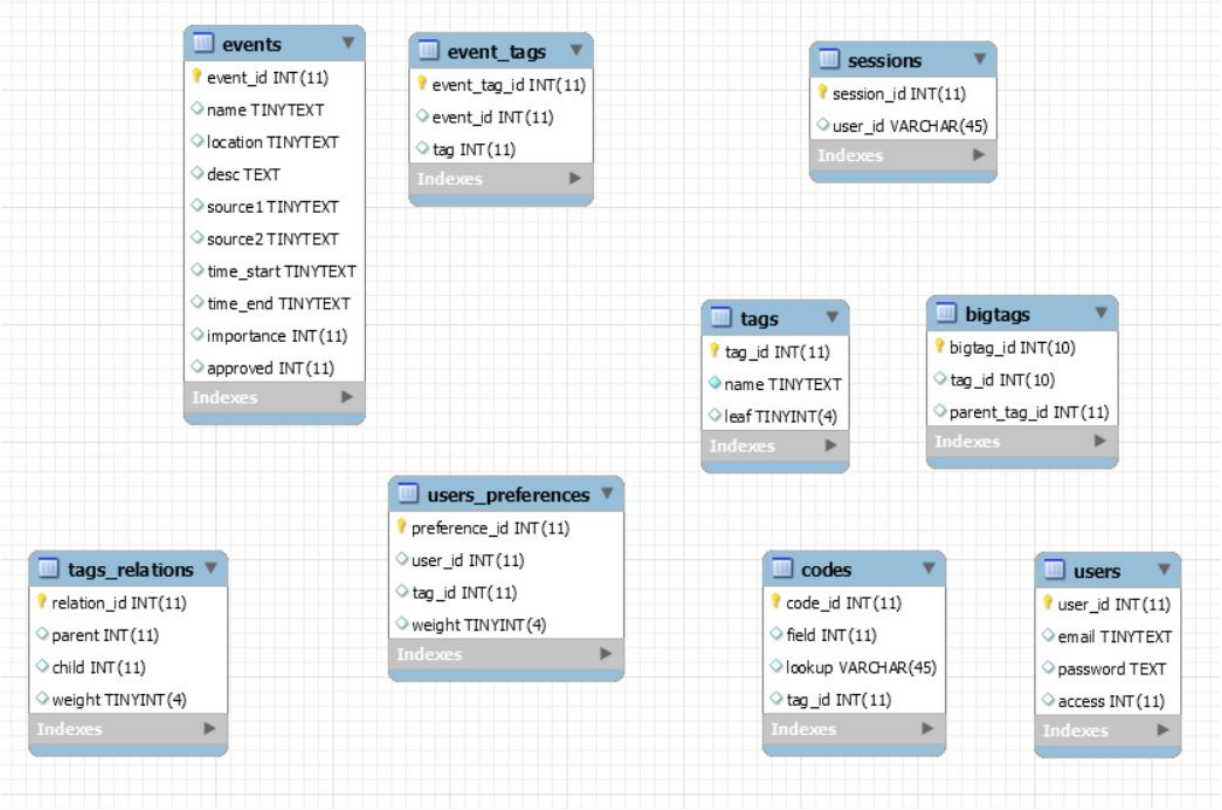
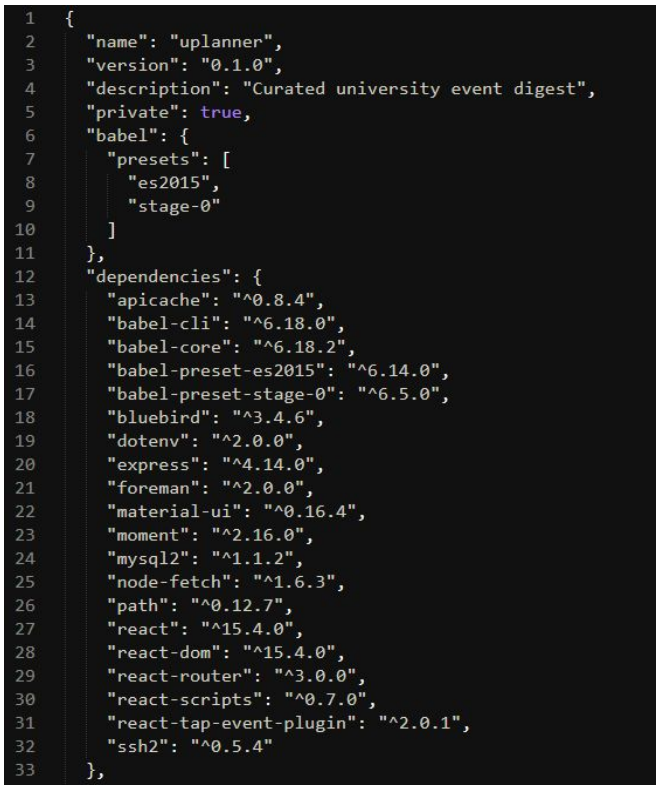


Figure 3. MySQL Workbench EER Diagram, hackathon build.

Additionally, an SSH tunnel was used to connect to the database. This was initially created through software like PuTTY during the hackathon, but it was later incorporated into the API itself (using npm module ssh2[9]). This was a restriction required by the remote host (Namecheap), but SSH gives the added benefit of more security.

Initially, the server and client instances were hosted locally, but post-hackathon they were moved to Heroku[10]. The database remains hosted on a remote Namecheap MySQL server. With all of these details in mind, the team used Slack, GitHub, and Google Drive to improve collaboration and code revision. I worked on the Administrative View, the database structure, and the Express API. Two other teammates—Jacob Shoaf and Ishan Taparia—focused on the React User View. Stephen Pioro worked on wireframing, future expansion ideas, and the hackathon pitch.

Several dependencies were used, most notably React-scripts, Babel and Material-UI. React-scripts came from create-react-app[11] which served as a boilerplate for the hackathon project. Eventually, foreman[12] was introduced to run separately the Express API and the React Front-end. Babel[13] served as a ES2015 transpiler that enhanced the syntax for JavaScript. Material-UI[14] provided React components that enhanced the design of the product, following Google's Material Design guidelines.



```
{
  "name": "uplanner",
  "version": "0.1.0",
  "description": "Curated university event digest",
  "private": true,
  "babel": {
    "presets": [
      "es2015",
      "stage-0"
    ]
  },
  "dependencies": {
    "apicache": "^0.8.4",
    "babel-cli": "^6.18.0",
    "babel-core": "^6.18.2",
    "babel-preset-es2015": "^6.14.0",
    "babel-preset-stage-0": "^6.5.0",
    "bluebird": "^3.4.6",
    "dotenv": "^2.0.0",
    "express": "^4.14.0",
    "foreman": "^2.0.0",
    "material-ui": "^0.16.4",
    "moment": "^2.16.0",
    "mysql2": "^1.1.2",
    "node-fetch": "^1.6.3",
    "path": "^0.12.7",
    "react": "^15.4.0",
    "react-dom": "^15.4.0",
    "react-router": "^3.0.0",
    "react-scripts": "^0.7.0",
    "react-tap-event-plugin": "^2.0.1",
    "ssh2": "^0.5.4"
  },
```

Figure 4: Package.json of the project, listing the dependencies used, March 2017 build

[9] Node SSH2 - https://www.npmjs.com/package/ssh2
[10] Cloud Application Platform | Heroku - https://www.heroku.com/
[11] Facebookincubator - Create-React-App - https://github.com/facebookincubator/create-react-app
[12] Node Foreman - https://www.npmjs.com/package/foreman
[13] Babel - https://babeljs.io/
[14] MaterialUI - http://www.material-ui.com/

Results

By the end of the HackOHI/O hackathon on Nov 19-20, 2016, the prototype included several working API routes, a Material user-interface, and the MySQL database (in Methods) populated from the scraper's events from Teamup. Since the hackathon, various improvements have been made to the project and its algorithms to automate and improve the digest's personalization. The next major step is implementing a Naive Bayes Classification model to determine the tags of each event based on its content and source.

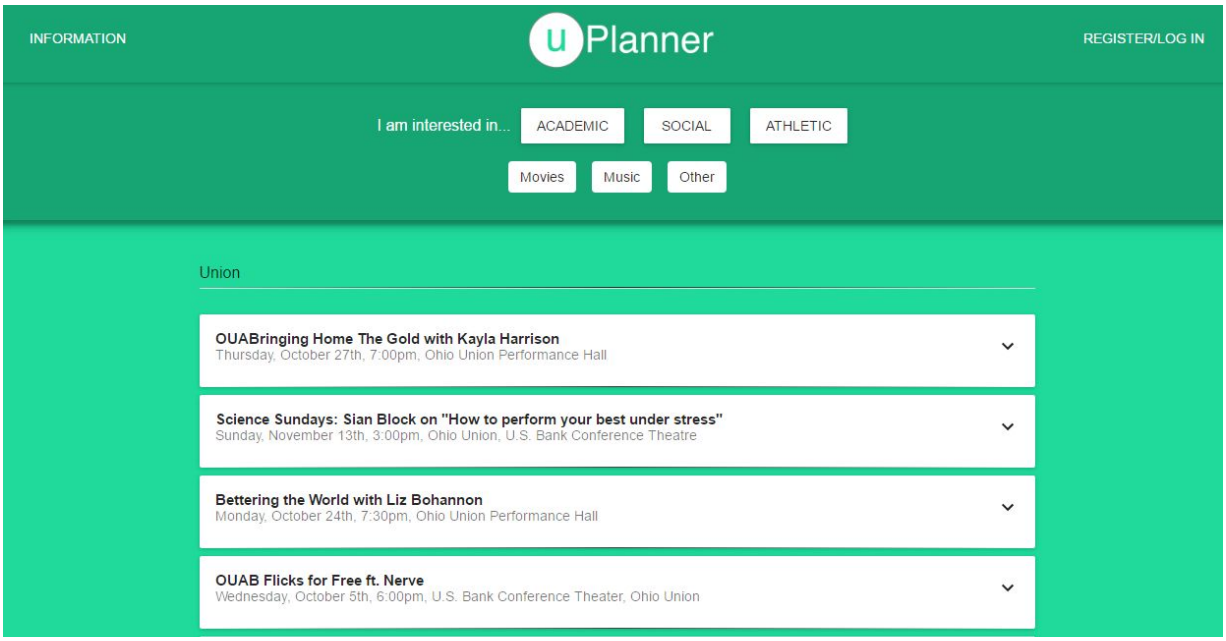


Figure 5. React front-end web interface, hackathon build.

Major post-hackathon improvements include mobile-responsiveness, style changes, working guest subtag navigation, API expansion and caching, improved search, and a Heroku-deployable setup. The guest tag navigation follows the model in the next page to determine whether an event should appear. There is much more to be done in the users backend, web scraping, and classification.

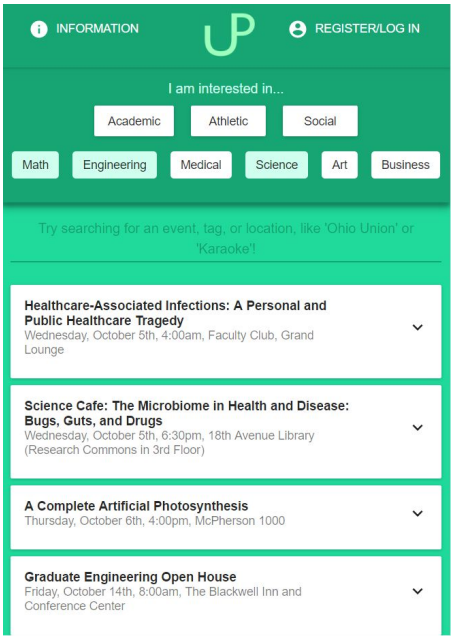


Figure 6. Front-end web interface, March 2017 build, in mobile view demonstrating tag filtering.

The algorithm to display events was developed further, shown below. In the web app, the guest side of the chart is fully functional but the user side hasn't been developed yet. This chart is essentially translated to a series of SQL queries to filter events accordingly.

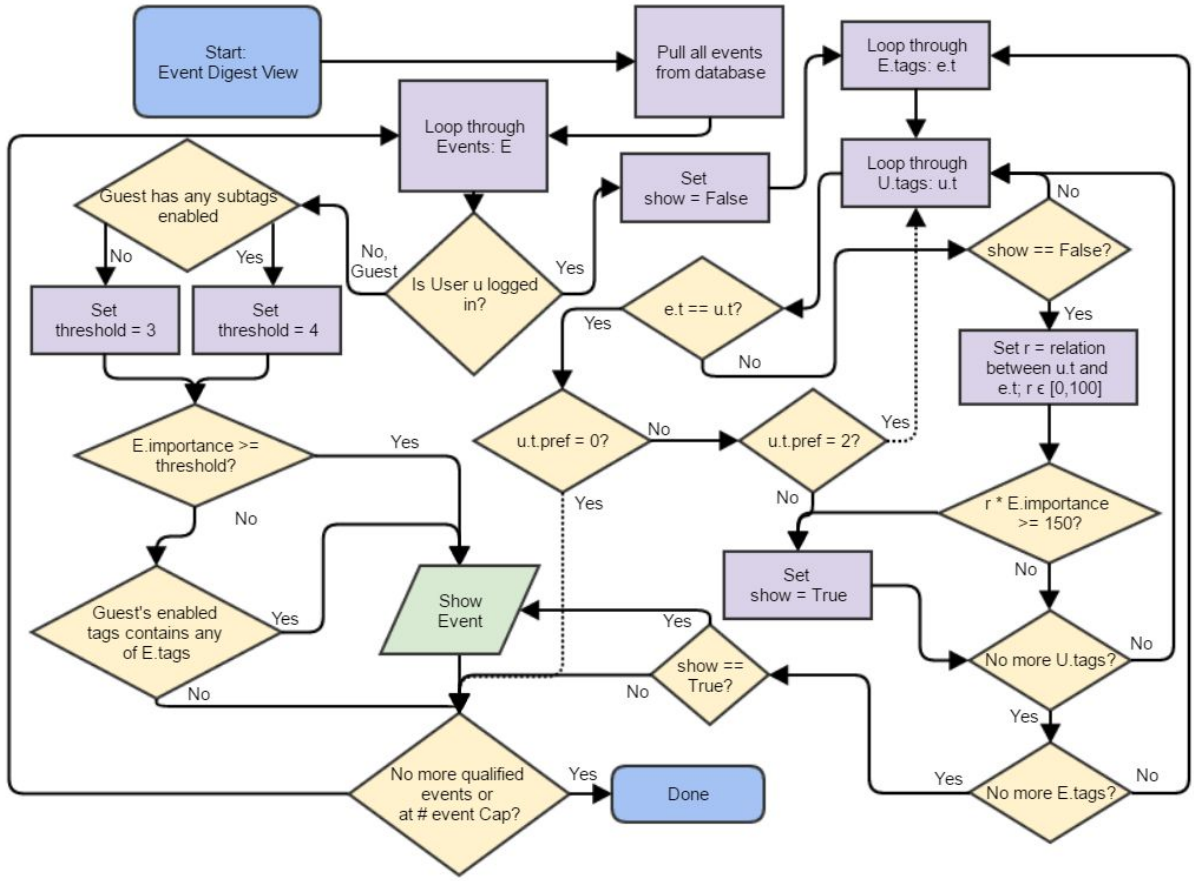


Figure 7. Event display algorithm

Parameters:

E.tags - List of tags that event E belongs to

- e.g. {11,12,100} for {'Computer Science', 'Electrical Engineering', 'Undergraduates')

E.importance - Relevance of event, 5 being the most relevant / broad, 1 being the least

- 1 might be a complex lecture, only applicable to specific majors, where 5 might be a big social event, like a free concert for all students

U.tags - List of tag preferences from the registered user

- e.g. {[11,2], [12,2] [14,0], [18,2], [100,1], [101,0]}
- The values are tag and preference; 0 = dislike, 1 = default, 2 = like, referenced by "u.t.pref" in chart

r - relationship between two tags

- These are defined in a table based on a tree-like structure, ranging from [0,100]
- e.g. Electrical Engineering and Computer Science might have r=75, but Computer Science and Physics might have r=25.

To automate curation of events, Naive Bayes Classifiers may be used, using the source and description to determine the tags' of events. This is the next major step of the project and is currently being developed.

Classification model:

Each event E has source s (where E was found, e.g. could refer to STEM Scholars Teamup Calendar, ACM-W Mailing List), Σd_i and Σt_i where t_i is the i[th] word of the title and d_i is the i[th] word of the description. To estimate $P(C \mid E)$, the probability that E belongs to the category C in question, we apply Naive Bayes[15].

Assume there are two classes, C and ¬C (mutually exclusive and collectively exhaustive). Note also let $P(s \mid E)$ mean the probability that E belongs to source s. We want to see if $P(C \mid E) > P(\neg C \mid E)$ to decide whether or not E belongs in C.

By the Naive Bayes model, let $P(E \mid C) = P(s \mid C) * \prod_i P(d_i \mid C) * \prod_i P(t_i \mid C)$

By definition, $P(C \mid E) = \frac{P(E \cap C)}{P(C)} = \frac{P(E)}{P(D)} P(s \mid C) * \prod_i P(d_i \mid C) * \prod_i P(t_i \mid C)$.

Therefore, $P(\neg C \mid E) = \frac{P(E)}{P(D)} P(\neg s \mid C) * \prod_i P(\neg d_i \mid C) * \prod_i P(\neg t_i \mid C)$

Thus, $\frac{P(C \mid E)}{P(\neg C \mid E)} = \frac{P(C)P(s \mid C)}{P(\neg C)P(\neg s \mid C)} \prod_i \frac{P(d_i \mid C)}{P(\neg d_i \mid C)} \prod_i \frac{P(t_i \mid C)}{P(\neg t \mid C)}$

Expressing the ratio in terms of series of likelihood ratios, we get

$ln(\frac{P(C|D)}{P(\neg C|D)}) = ln(\frac{P(C)P(s \mid C)}{P(\neg C)P(\neg s \mid C)}) + \sum_i \frac{P(d_i \mid C)}{P(\neg d_i \mid C)} + \sum_i \frac{P(t_i \mid C)}{P(\neg t_i \mid C)}$. If this ratio > 0, we classify E as C.

In practice, the Bayes model will be likely implemented by another node module (such as yosriady/node-bayes[16] or ttezel/bayes[17] on GitHub), but this is the basis for the classification. Additionally, some hard-coded rules may exist for classifying. For example, if an event comes from the ACM-W (Women in Computing) mailing list, we can likely predetermine that the event be classified as "Computer Science" without fail.

After the completion of the classifier to automate tagging events, the administrative view and web scraper will be further developed. The goal of the project is to automate as much as possible of the web scraping (to be likely weekly) and classification. Some administrative

[15] Naive Bayes Classifier - Wikipedia - https://www.wikiwand.com/en/Naive_Bayes_classifier
[16] yosraidy/node-bayes - https://github.com/yosriady/node-bayes
[17] ttezel/bayes - https://github.com/yosriady/node-bayes

oversight needs to be added to make edits and examine where the classification model or scraper may need to be changed.
As it stands, the scraper only scrapes Teamup calendar and is largely manual. With more work on these aspects, the application can become far more useful and self-sustaining. After these features are developed, the application will be launched.

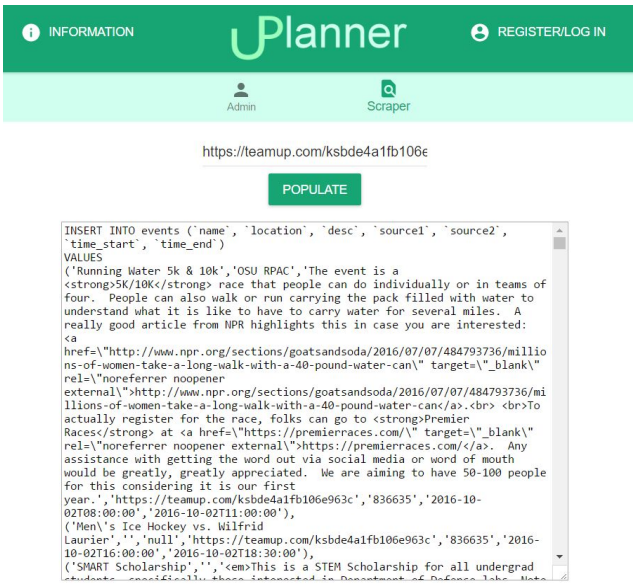


Figure 8. Web scraper interface, March 2017 build, which would generate a query to insert all events scraped from the resource.

Conclusion

The UPlanner prototype demonstrates the prospects of scraping different campus resources to be viewed in a single web app. The next steps of the project are expanding the development team, developing the machine learning model, and obtaining faculty support. From there, the application will be tested further for its utility at The Ohio State University by enlisting beta testers and feedback. The goal is for the product to be useful without significant administrator or moderator support, so developing a strong scraper and machine learning model is key in accurate predictions. Regardless of the success of the product, I have gained valuable experience working with a NodeJS tech stack, web scraping, and machine learning. By the end of 2017, I hope to have a beta test with at least 50 students, staff, and/or faculty. If the product becomes more successful, I would expand its scraping tech to as many university resources as possible, allowing for more types of people to benefit from the product.