CPSC 437

Cameron Yick

Charles Wong

Yehia Saleh

Yijiao He

Final Project Report:

College Athletics Data Explorer

Project Summary:

This project was created to tackle the messiness and disorganization inherent in college athletic data. To this end, we have scraped data from multiple disparate sources, streamlined them into consistent models, and built a web-application through which users can execute queries with flexible parameters to access clear, aggregated results across schools and sports, in a way previously unavailable to end users.

Project Repository: <https://github.com/hydrosquall/yale_athletics_data>

# Problem Description

## Goal

This project, an insight into the athletics datasets of multiple colleges, was developed in response to the disparate and disorganized nature of college athletic data available online. It is motivated by the need for a higher-level overview of athletic data, from which one can identify trends in the sports recruitment process. These include examining which states and countries college athletes hail from, what fields they choose to major in, similarities in their build, how long they play before they stop participating in sports, just to name a few.

We set out to create a solution that would enable users to obtain an understanding of this information in a hassle-free and clear manner.

## Existing Solutions

Today, information about college sports data is stored in the form of individual sports websites, which contain raw information and statistics about the players and rosters, often in a disorganized, inconsistent format.

Individual pieces of information are often stored on singular pages, and this isolation makes it exceedingly difficult to create effective comparisons that include more than one sport or school. As such, very few aggregated analyses have been attempted on these datasets, a problem this project aims to tackle.

## Our Solution

The project comprises a web-application which allows users to query for athletic data through specific and flexible parameters, including college, major, season and sport. This is enabled due to the prior streamlining of data we have performed. The streamlining process involved 3 stages:

1. Scraping the data from the disparate web pages
2. Cleaning the data and putting it into a human-readable format
3. Making the data consistent and source-agnostic, enabling smooth delivery into the database

This process results in a normalized relational database through which users are able to execute queries of varied complexity in a smooth manner.

While this application covers data specifically from Yale and Harvard College, It is our hope that this project will serve as an underlying foundation upon which further analytical tools can be built.

## Technical Challenges

Our hypothesis was that if the player bios were being programmatically generated,

Data Scraping Challenges

* Throttling the scraper so that we didn’t overload the Yale or Harvard websites
* Parallelizing the scrape to ensure that the entire scrape of 17000+ pages can be conducted within an hour or two.
* Dealing with variations in the HTML structure of players, as some of the older listings structure their player information differently than the more recent listings.

Data Cleaning Challenges

* **Athlete Gender:**
  + It would seem intuitive that a sport roster only contains athletes from one gender. However, we found that some sports mixed genders (Track and Field), and some sports feature one or two people from the other gender (Male Crew includes one female coxswain). Because resolving these issues would require having domain expertise, we ended up dropping the gender column.
* **Athlete Dimensions** (Height/Weight)
  + Height/Weight are only provided for athletes in pure male sports
* **String Encoding**
  + Because player biographies contained special international characters and HTML formatting tags, we had to strip those tags before they could be loaded into the database. There were various formatting issues dealing with conversion from ASCII to UTF-8 Unicode that had to be normalized.
* **Aligning Fields / Clustering Labels**
  + Each school labeled sports slightly differently (Track == Track & Field), so we had to build a data dictionary that mapped sport names, major names, and other dimensions to a common field before being loaded into the database.
* **What constitutes a single athlete?**
  + We are not provided with a unique “player ID”, so we assume that the combination of player + hometown is a unique identifier. However, if multiple athletes share a name and come from the same hometown, it is difficult to disambiguate their records.
* **Misspellings in all fields**
  + People misspelled their major, hometown, high school and other fields, which tells us that their database didn’t have any spell checking when they filled out their form.
* **Combining Biographies for each season into a Single Biography**
  + Each athlete gets a new biography for every season in which they compete. However, while sometimes the biography is purely additive, occasionally they remove or update old information (such as major or previous accomplishments). We deal with this by picking only the latest bio,

# Approach

## Workload Distribution

**Cameron**

* Scraper scripts for the Yale and Harvard Athletics Websites, parallelized with shell scripts
* Extraction scripts to convert the pages to csvs, handle data transformation and cleaning steps with Python / Jupyter Notebooks
* Designed tables structure / ERD diagram for dataset
* Enriched athletes dataset by writing Python script to query Mapquest

**Charles**

* Developed Python Notebooks to further clean and transform data, including cleaning Yale student bios (often stored in an inconsistent manner across sports)
* Handling normalization of data across sources from both Harvard and Yale - different amounts and types of information were available from each college, a complication which needed to be handled.
* Developed visualizations from user-generated query results by using D3, a JavaScript based data visualization framework. Includes bubble chart to represent choices of majors, and a choropleth to represent the geographic distribution of athletes across the US.

**Yehia**

* Created the Rails application and led backend development
* Imported data and created several models (tables) that correspond to the ER diagram
* Integrated the database in the web application including integrity constraints, associations, and through tables for many-to-many relationships
* Designed controllers logic to display query results in the landing page
* Connected the query page to our database
* Used embedded ruby in html pages to access data dynamically in a more organized and readable way

**Yijiao**

* Designed and implemented the front-end query and listing pages to match functionalities required by the application using HTML
* Implement autocomplete and selection for search fields using jquery
* Implement search and sort functions within the results returned from the query using jquery and listjs
* Manually test for inconsistencies in data and bugs in application, check for match between ERD and schema
* Organize the structure and writing of the final report

## Technical approach

Why we chose the tools we chose to use for each part of product

* **Jupyter Notebooks for ETL**
  + Because of the amorphousness of the data across the Yale and Harvard websites, the iterative Python environment allowed us to refine the scrapers as new data challenges emerged.
  + We utilized the Pandas library to support SQL-like operations on the data
  + Geocoder API to resolve the geographic entities to achieve a common naming structure.
* **Rails for Backend**
  + Utilized sqlite3 gem to design and implement the database to be used by Active Record.
  + Rails provides Active Record, an ORM which connects the rich objects of an application to tables in a relational database management system. It allows us to execute SQL like statements on the database, facilitating the creation and use of business objects whose data requires persistent storage to a database.
  + Rails allows us to specify integrity constraints at application (model) level.
  + Rails offers several debugging tools which allowed to project to progress smoothly
* **JavaScript D3 for Data Visualization**
  + D3 allows for quick prototyping of visualizations, and plays well with the Rails backend using JSON.
  + D3 has a wide range of capabilities for producing different SVG-based visualizations, which was useful early on when it was not clear what kind of visual data representations we should use.
  + Being built in JavaScript, D3 allowed us to integrate our visualization modules seamlessly with the other interactive front-end components being built into the web-app

## Architecture

### Information Architecture

*Graph 1: Entity-Relationship Diagram of the database system*

The schema is separated into 5 different tables, with types and constraints as shown in the code snippet below (code is in Ruby):

//Maps to high\_school in ERD

create\_table "high\_schools", force: :cascade do |t|

t.datetime "created\_at", null: false

t.datetime "updated\_at", null: false

t.string "high school"

t.string "high\_school\_id"

End

//Maps to Roster in ERD

create\_table "rosters", force: :cascade do |t|

t.string "sport"

t.string "season"

t.datetime "created\_at", null: false

t.datetime "updated\_at", null: false

t.string "players"

t.string "college"

end

//Maps to played in ERD

create\_table "rosters\_students\_throughs", force: :cascade do |t|

t.datetime "created\_at", null: false

t.datetime "updated\_at", null: false

t.string "student\_id"

t.string "roster\_id"

end

//Maps to Player in ERD

create\_table "students", force: :cascade do |t|

t.string "high school"

t.string "hometown"

t.string "ht."

t.string "name"

t.string "no."

t.string "position"

t.string "wt."

t.datetime "created\_at", null: false

t.datetime "updated\_at", null: false

t.string "bio"

t.string "misc"

t.string "college"

t.string "student\_id"

t.integer "startseason"

t.integer "endseason"

t.string "major"

end

//Maps to college in ERD

create\_table "universities", force: :cascade do |t|

t.string "name"

t.datetime "created\_at", null: false

t.datetime "updated\_at", null: false

end

# Specifications

## Function Specifications

* Querying from data with multiple parameters and displaying them in a table format
* Viewing individual bios for each player
* Sorting any queried table by any of the available parameters, in lexicographical order
* Viewing visualizations across different slices of data

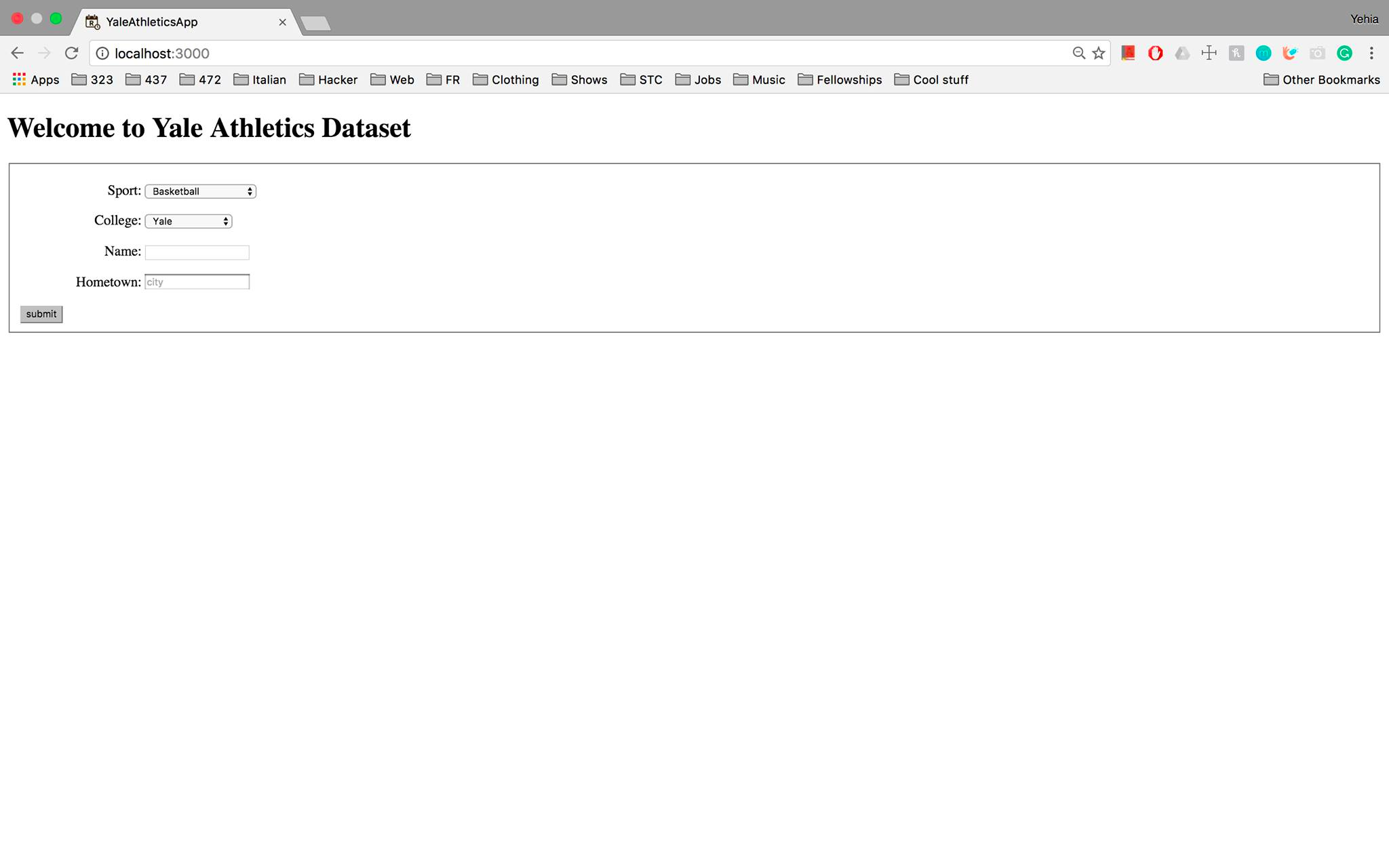
## Technical Specification

* **Jupyter Notebooks**- Scrapes and parses the data into a structured format, data are normalized into BCNF.
* **Ruby on Rails Backend**- Generates Index + listings pages, provides an API endpoint that other analytics tools could eventually run queries against.
* **Sqlite3 Database-** Stores the data and supports SQL operations.
* **D3 / Javascript-** Allows you to manipulate, visualize, and filter views of the data listings

# Use Cases

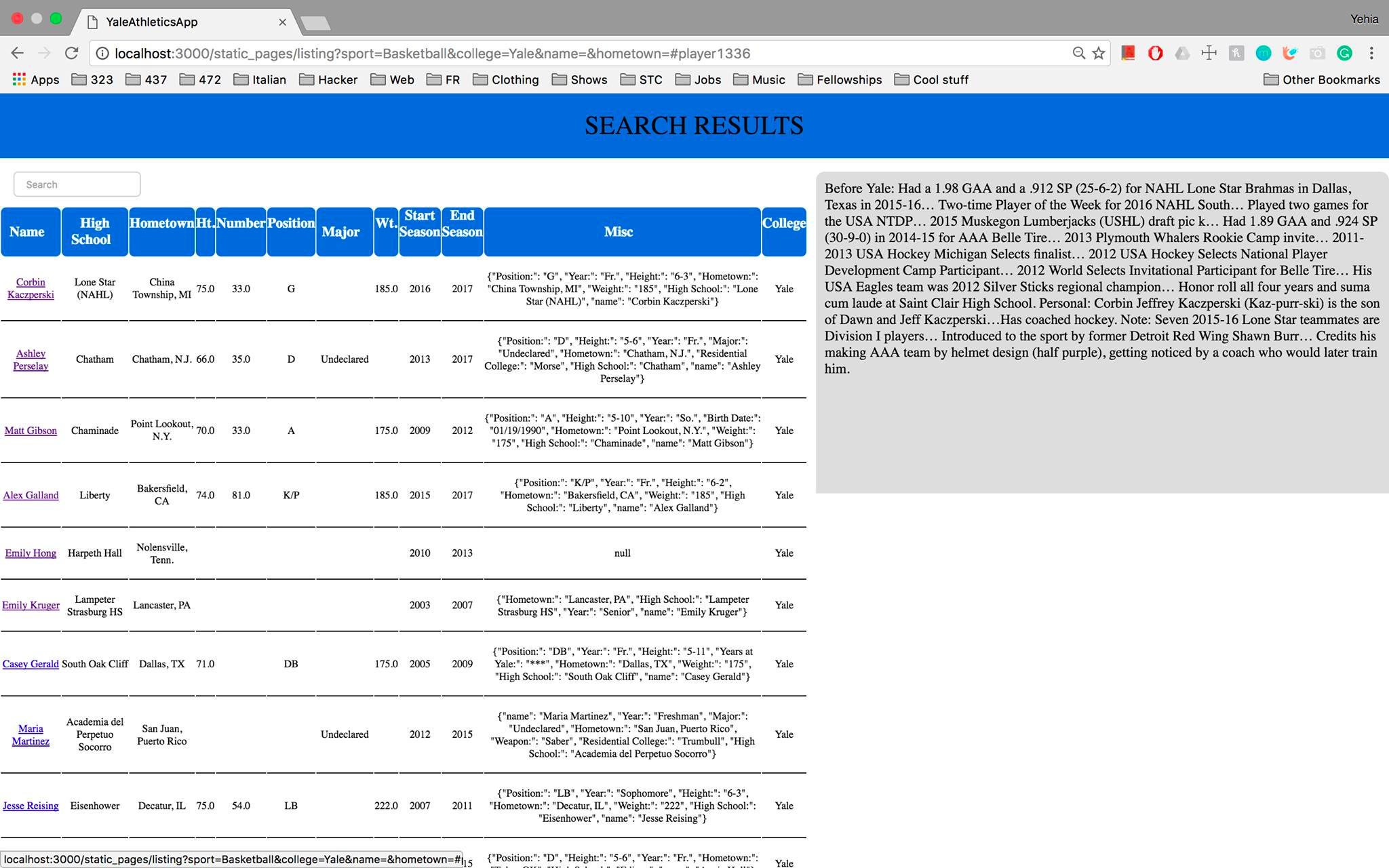
## Search example

The following is an example of how a user can use the application to find the names and information of all the Basketball Athletes from Harvard University, and then proceed to sort the results by height of the athletes.



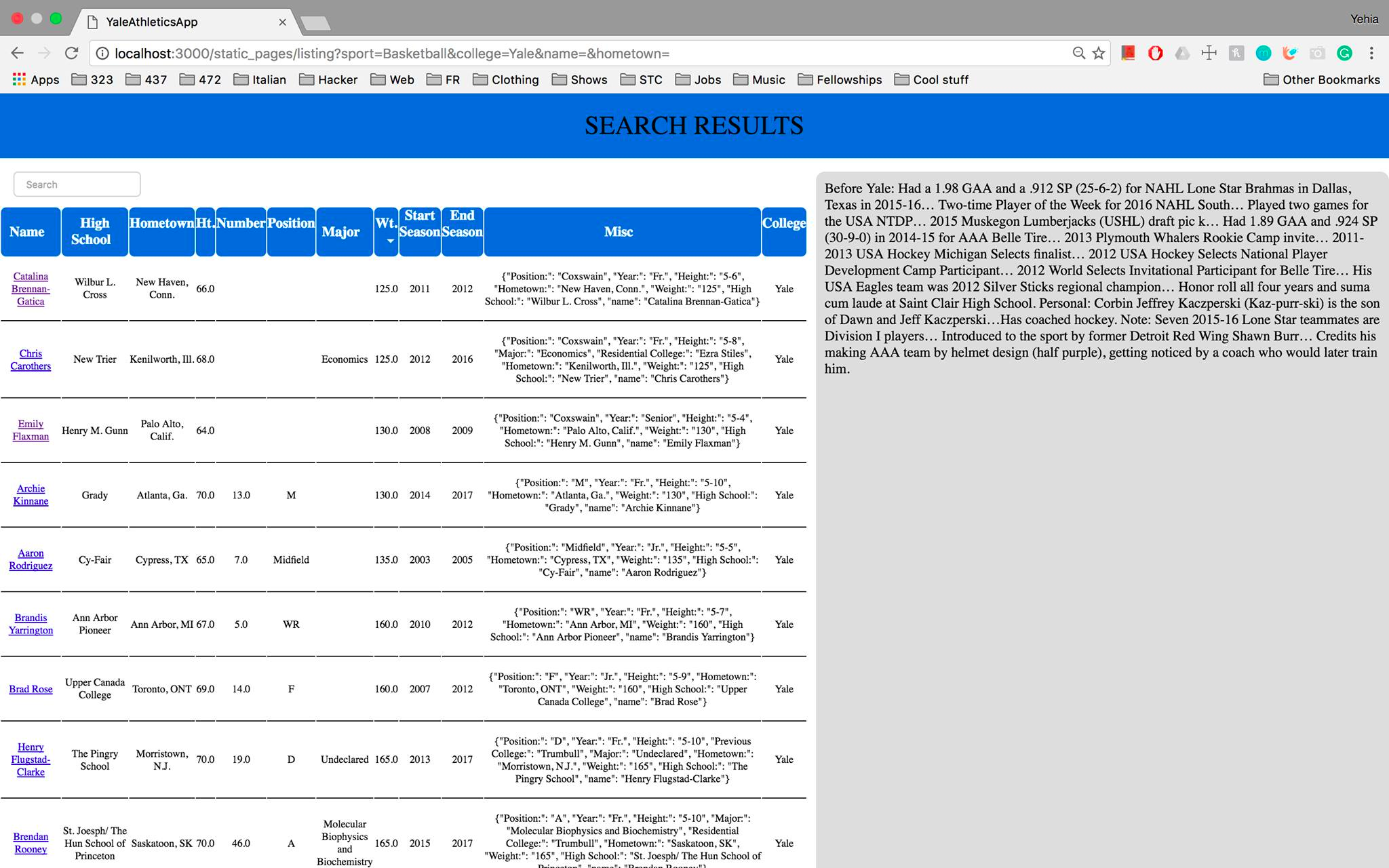
*Graph 2: homepage/query page of application*

On the homepage, there are input fields and selection boxes for the user to indicate which parameters to put into the search. Upon clicking submit, the user will be brought to the following listing results page.



*Graph 3: results page for Basketball players at Harvard University.*

On the results page, the list of qualifying records appears on the left, with information such as name, high school, hometown, etc. Upon clicking the hyperlink on each student’s name, the student’s bio will show up in the grey area on the right.



*Graph 4: returned results of Yale Basketball players ordered by weight*

An additional feature of our interface is that it allows the sorting of results as well as a search within the results. The image above shows the results of Yale basketball players after being sorted by weight.

## Limitations / Next Steps

* We want to set up a scheduled script that will update all the tables automatically once per year, after each of these websites are updated.
* Our search function is currently limited to only search based on 4 parameters: sport, college, player name, and hometown. We plan to allow for more search parameters to exists for the user to be able to narrow down the results as desirable.
* We want to connect the map representation to the query results to be able to dynamically display the map based on user queries
* We want to add more types of interesting summary visualizations so that people can get more actionable insights from viewing the data.

## Potential Audiences and Uses

This web app will be of interest to anyone who wants to learn about the state of athletics within colleges today. This group of audiences includes the athletes themselves, potential students and parents who want to compare sports across colleges for college application purposes, research on student who participate in sports, etc. Currently, there are many uses of this application, the following lists a few of the most likely uses:

1. Player statistic lookup
2. Team roster lookup
3. Team comparison between colleges
4. Geography data of players
5. Find correlation between attributes
   1. State vs sport
   2. State vs major
   3. Sport vs major
   4. Weight/Height vs statistics

With further development of the current API, we will be able to track the data over time, and develop more insights into data such as recruitment trends, performance trends etc. The GUI that we provide will be able to graphically represent these trends for better interpretation of this data. However, because the core of our application cleans and normalizes the myriad different forms of data provided by the universities, this provides a platform for many more things that can be done with it. The user can use their own interface to interact with the data in whichever way they want to achieve different goals.