Arjun Pothukuchi, Ava Dagostino and Evan Tao
GROUP 7
CIS 550 PROJECT REPORT

Abstract

Fantasy Olympics is a web application in which users are able to create an account and play a simulated Olympic Games with each other. In the game, users select up to 10 real Olympic athletes for some set of events. The athletes are each given a performance score by our algorithm. The total scores for each player are compared, and the player with the highest score wins; scores are added to a global leaderboard for players who agree to doing so. The game makes use of Olympic data for athletes, events and countries; athlete biographies from Wikipedia; and weather data from WeatherUnderground. This game provides a platform for fans of the Olympics to engage with their favorite athletes and participate more in Olympics sporting events through fantasy play. The game also encourages Olympics fans to learn about lesser known athletes and events.

Our app can be found here:
http://node-express-env.tfxkrvfkca.us-west-2.elasticbeanstalk.com/
Github link: https://github.com/arjun19403/cis-550-project
Alternatively, running 'npm install' then 'npm start' in the project directory will start the app locally.

Use case

Fantasy Olympics can be used for fun, relaxation, or even as a tool to learn about the history of the Olympics!
A typical use case follows:

1. Create an account to play, and choose a team name and password or login with the credentials you provided previously
2. Choose any number events using the search bar to get the Athletes that participated in each event
3. Choose up to 10 athletes to be a part of your team and click Play!
4. We will simulate the athlete's performance at the given location on the given day, using the actual weather data recorded at that time and place
5. Check out how you did versus other players in your game, and see if you can get on the Global Leaderboard!

Key and special features

- Create new account or login to old account
    - App verifies name, team name and password information (information stored on DynamoDB)

    - Password is hash-encrypted
- Select random historical date, location, temperature and humidity
    - Temperature and humidity are historically accurate for the particular date and location
- Select and remove Olympic events using drop down menu, and select and remove Olympic athletes which update according to selected events
- Display “Athlete has been taken” error when another player selects an athlete before you do (Player and athlete combinations are stored on DynamoDB)
    - Overcomes concurrency issues
- No longer lists athletes when they have been chosen by other players; no longer lists events when they have no more athletes to be chosen
- Generate and display athlete score using unique scoring algorithm
- Refresh scores after other players have played
- Allow “Play Again” option after all players have played (Player and athlete combinations are removed from DynamoDB)
- Choose to opt-in to global leaderboard
- Save and display top 10 highest scoring players on global leaderboard (Stored on DynamoDB)
- Post results to Twitter

Modules and architecture

For the frontend we used VueJS along with HTML and CSS in order to display the application, as well as the data from the server. Since this is a single-page application, we used the Axios javascript library to issue AJAX requests to interface with the server. For the backend, we used the Express framework on top of NodeJS to handle connections from clients and send queries to the database.
We used Amazon AWS RDS to host our MySQL database and DynamoDB to as our NoSQL database.

Data instance use

We married our application with the data instance by inserting the cleaned data into a database hosted by RDS. There is also a NoSQL data instance using DynamoDB. The database was accessed by the Node server-side application to interact with the gameplay.

The game accesses the data by:

- Listing the set of events

- For each event the user chooses, it lists the athletes who have participated in that event historically
- For each athlete the user chooses, it looks at that athlete's event participation and country to generate a score
- The DynamoDB instance is used to store information about the state of the game, the login information of the players, and the performance of players' drafted athletes

Complementary data, data cleaning and import mechanism

- Data sources used
  - Original Olympic data set
  - Weather data from www.wunderground.com
  - Athlete biographies from www.wikipedia.org
- We imported the data using a set of Python scripts and the BeautifulSoup package
  - The scripts for weather data and Wikipedia bios retrieved data using HTML requests
  - The script for the original Olympic data set connected to the mySQL instance directly; the data was parsed from a *.csv* file, appropriately cleaned and aggregated, and then added to the respective mySQL tables
- Cleaning
  - We restricted the weather data to January and June of 2014 and 2015 in predetermined cities.
  - We aggregated medal data (1 for bronze, 2 for silver, 3 for gold) for a particular athlete and event.
  - Only the relevant columns from our source data were selected.
  - We reformatted athlete names from "LAST, First" to "First_Last" to retrieve Wikipedia entries.
  - We filtered out athletes with unretrievable Wikipedia entries .
  - We added climate values ('hot,' 'cold,' 'dry,' 'wet') for 20 predetermined countries which we would select athletes from.
  - We added IDs to several countries that were missing them.

Algorithms, communication protocols, etc.

- **Scoring algorithm**: each athlete was given a randomized score such that athletes who historically won more medals are more likely to receive a higher score in the game. On top of that, athletes whose home country has similar

weather to the weather in the game have the potential to earn more points, and athletes whose home weather is different than that of the game are less likely to receive more points. The algorithm is quite naive; one future improvement of the game would be to implement a more sophisticated algorithm, that looked at things like actual results (feet/inches in a long meter jump) and things like consistency and change over time. Let m be an athlete's historic medal score. The scoring is as follows:
    - If weathers match: $5 + (rand(\{1, 2, ..., 10m + 5\})) + rand(\{1, 2, ..., 5\})$
    - Else: $5 + (rand(\{1, 2, ..., 10m + 5\})) + rand(\{-2, 1, ..., 2\})$
- **AJAX:** We used JSON to transmit data between the client's browser to the server. Communications from the server would always have a "success" flag that indicates whether an error occurred code execution, such as a database error. If the success flag was false, then an error code was provided such that the error could be displayed to the user.
- **AWS SDK:** Wrappers around the connections to the MySQL and NoSQL database were provided through the JavaScript SDKs provided by Amazon. We had one SQL connection and one DynamoDB Document Client per route that handled the database operations.

Technical Challenges

- Had to find a way to categorize countries by climate; this was overcome by manually inserting 'hot,' 'cold,' 'dry,' 'wet' to top 20 countries with most Olympic athletes
- Webscraped athlete bios yielded some invalid/inconsistent results; this was overcome by replacing these results with 'N/A'
- Ava and Evan started the project with very little web app development experience; this was overcome by learning online and from Arjun
- Tried to implement player profile pictures; this was not overcome because we could not figure out how to post picture files to backend

Optimization techniques employed (indexing, optimization)

Created indices on Athlete country_id and bio, since when the app queries the list of events, it only selects events with at least one athlete such that the athlete has one of 20 possible country_id and bio not equal to 'N/A,' and when the app queries the list of athletes for a particular event, it only selects athletes with one of 20 possible country_id values and bio not equal to 'N/A.'

Arjun Pothukuchi, Ava Dagostino and Evan Tao
GROUP 7
CIS 550 PROJECT REPORT

Before Optimization
Logging in: 1052.598 ms
Getting Weather: 191.048 ms
Getting Events: 146.253 ms
Getting Athletes: 192.236 ms
Calculating Scores: 490.672 ms
Getting Player Scores: 117.572 ms
Getting Top Scores: 521.779 ms

After Optimization
Logging in: 280.357 ms
Getting Weather: 171.048 ms
Getting Events: 101.394 ms
Getting Athletes: 159.283 ms
Calculating Scores: 148.668 ms
Getting Player Scores: 75.544 ms
Getting Top Scores: 83.986 ms

Technical specifications

We hosted our Node app with a free t2.micro EC2 instance while using Elastic Beanstalk to deploy the application. Elastic beanstalk handles routing requests to and from Node, hosting the Node environment on the EC2 instance, and organizing the load balancing. As mentioned above, we are hosting a MySQL instance on RDS and a DynamoDB instance with several tables.

Potential future extensions

- Allow players to choose and save profile pictures on Amazon S3
- Incorporate event scores (times, distances, etc.) into scoring algorithm
- Create message board/forum for players to interact with each other

Division of work among group members

- Evan
    - Features
    - Database setup
    - HTML parsing & cleaning
    - Data upload

Arjun Pothukuchi, Ava Dagostino and Evan Tao
GROUP 7
CIS 550 PROJECT REPORT

- Ava
    - Features
    - Frontend display
    - Scoring algorithm
    - CSV parsing & cleaning
    - Data upload
- Arjun
    - Features
    - HTML/CSV parsing
    - Installed frontend and backend frameworks and setup deployment
    - Organized the structure of the code