EE 461L Team Project Report: ChampionsDB 2019-2020

Team A6: Ben Buhse, Pearse Flood, Conor Flood, Juan Paez

TEAM INFORMATION

GitHub Repository Link: https://github.com/UT-SWLab/TeamA6

Web Application Link: http://champsdb.herokuapp.com/

Member Name	EID	GitHub	Email
Juan Paez	jfp778	juanpaez22	juanpaez@utexas.edu
Ben Buhse	bwb887	bwbuhse	bwbuhse@utexas.edu
Conor Flood	cf26784	cnflood	conorflood@utexas.edu
Pearse Flood	pkf256	pkflood	pearseflood@utexas.edu

MOTIVATION AND USERS

Soccer is the most popular sport in the world. Even so, its popularity continues to grow with momentum, particularly as it begins to seep into the cultures of the US and Canada, which have traditionally rejected the sport. A natural starting place to learn more about the soccer world is becoming invested in the UEFA Champions League (UCL), the world's most prestigious and competitive club soccer competition. Our web application, ChampionsDB, aims to provide a centralized resource for soccer novices and lifelong fans alike to learn about the UCL.

Right now, those looking to learn more about the UCL do not have a centralized resource to do so-- they may look at match scores on ESPN, view individual player information on Wikipedia, and learn about teams through their webpages. Through several REST API data sources, ChampionsDB puts all of this information in one place and presents it in a clean, integrated fashion. We provide matches, players, and teams from the 2019-2020 Champions League season as three models, each populated with many instances, in order to provide our users with a centralized location for all of this information.

The information we provide can be helpful to anyone interested in the subject, but is particularly aimed at soccer novices. These users will be able to use the information on our site to familiarize themselves with soccer at the most competitive level. By visualizing lots of information about players, teams, and matches in a single web platform, our users will be able to become well-versed members of the soccer world in no time.

REQUIREMENTS

The team developed a set of requirement specifications based on user desires in order to determine the necessary functionality of ChampionsDB. Prior to each phase, we developed a set of user stories that would guide the development of that phase-- these are presented below with estimated and actual completion times. A use-case diagram is also presented, demonstrating the birds-eye view of our system interactions with users, although it does not encompass as much detail as the more fine-grained user stories.

User Stories

Phase I User Stories

1. As a general user, I should be able to navigate smoothly between the index page, the model pages, and the instance pages so that I can easily digest the presented information and find what I am looking for.
    a. Time estimate: 6 hours
    b. Actual time: 5 hours
2. As a general visitor, I should be able to view an about page with information about the site including its purpose, intended users, an explanation of the result produced by the data, the group name and its members, information about each member, links to each data sources, information about how each tool was used, and a link to the GitHub repo.
    a. Time estimate: 2 hours
    b. Actual time: 3 hours
3. As a general user, I want a coherent home page that clearly presents and summarizes the information available to me at ChampionsDB so I can easily understand my options.
    a. Time estimate: 3 hours
    b. Actual time: 3 hours

4. As a user wanting to learn more about the UCL, I want to be able to view a grid of all its teams and their respective shields to easily visualize the makeup of the league.
    a. Time estimate: 2 hours
    b. Actual time: 4 hours
5. As a user wanting to learn more about a specific team, I should be able to view more detailed information like their popular players and recent match results in an instance page for that team.
    a. Time estimate: 2 hours
    b. Actual time: 4 hours
6. As a soccer fan, I should be able to view a grid of match scores so I can see a variety of results in the same place.
    a. Time estimate: 3 hours
    b. Actual time: 2 hours
7. As a user curious about previous matches, I should be able to access pages specific to individual matches containing statistics and information about teams and players involved so I can learn more details about individual matches.
    a. Time estimate: 2 hours
    b. Actual time: 2 hours
8. As a soccer novice, I should be able to view the basic profile information (name, position, nationality) of popular Champions League players in a grid so I can learn more about the most important names in the soccer world.
    a. Time estimate: 3 hours
    b. Actual time: 3 hours
9. As a more involved soccer fan wanting to learn more about a specific player, I should be able to view a detailed player page containing their strengths and stats so that I can understand that player's style and trends better.
    a. Time estimate: 3 hours
    b. Actual time: 5 hours

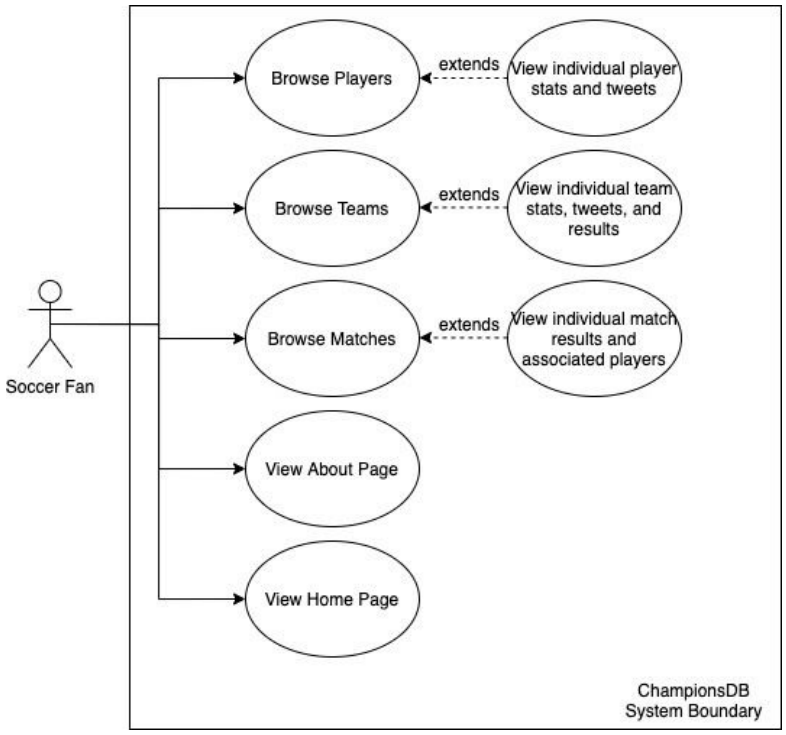
Phase II User Stories

1. As a novice learning about the Champions League, I would like to be able to browse many (hundreds) of instances of players, teams, and matches in order to expose myself to large amounts of information and become well-versed in the soccer world.
    a. Estimated time: 6 hours
    b. Actual time: 12 hours
2. As an analytical fan, I would like to view player statistics from the full season (goals scored, assists, pass accuracy, etc.) so I can compare players and evaluate their strengths.
    a. Time estimate: 6 hours
    b. Actual time: 8 hours
3. As a user, I should be able to view a robust and complete website that does not crash inadvertently
    a. Time estimate: 9 hours
    b. Actual time: 7 hours
4. As a user, I should be able to access all model pages through a pagination bar so that I can more easily navigate the website and its content.
    a. Time estimate: 8 hours
    b. Actual time: 10 hours
5. As a user, I should be able to view clear, non-distorted pictures regardless of the window size or device being used so that I can always have an accurate representation of the site's content.
    a. Time estimate: 2 hours
    b. Actual time: 1 hour
6. As a user, I should be able to easily read the text on all pages of the website. Currently, some pages are a little difficult because of clashing colors.
    a. Time estimate: 2 hours
    b. Actual time: 2 hours

Phase III User Stories

Use Case Diagram

Figure 1. Use Case Diagram



DESIGN AND MODELS

We present the team's software design and the data models used together because the decisions for each were tightly coupled and helped inform each other. At a high level, we chose to represent information in our database with three high-level model entities: Players, Teams, and Matches. These entities were represented in a Python class on our Flask application, and instances of these classes are injected into their corresponding model and instance HTML pages.

A Player object represents a player on a team. The object contains an id, a corresponding team id, a position, player images scraped from Google, season statistics (goals, passes, accuracy, minutes played, ratings), pundit ratings, and a set of Tweets about them. In total, we represent 102 players from the 2019-2020 Champions League, including all goal-scorers and captains, to prioritize players people will be more interested in. Information to populate Player objects was derived from information scraped from Football-API, Twitter API, and FutDB API.

A Team object represents a team that participated in the Champions League 2019-2020 season from the qualification stage to the final. Team objects contain an id, a city and country, a

founding date, the team logo, information about the team's stadium, and Tweets about the team. In all, we represent 41 teams. Football-API and Twitter API were used to populate information for Team objects.

A Match object represents a fixture between two teams during the Champions League 2019-2020 season. It contains the id corresponding to both participating teams and other general fixture information (score, referee, stadium, etc.). The multimedia used to represent Match objects is derived from the two teams that participated in the match. The Football-API was used to gather the data for our Match objects.

These three classes serve as the three models in our web application, each with its own model page and corresponding instance pages. In addition to these three classes, we have one more class that does not serve as a model, but rather serves to contribute information to the existing model: Events. An Events object represents information from a single player in a single match, including the goals they scored, their shooting accuracy, their passing accuracy, the rating they received, and more. Information from Events, as well as relationships between models, are injected dynamically when requests are made, rather than being stored statically in the database.

To illustrate the relationship between these classes, we present a class diagram below:
// TODO: insert class diagram here

TESTING

We tested our program using a combination of the Python unittest framework and Selenium GUI tests. The unittest cases are focused on testing the entirety of `app.py` where all of our Python code is, whereas the Selenium tests are intended to test the Jinja2 templates themselves.

Unittest was used to create a number of unit tests to ensure that the various routes and collection Classes inside of `app.py` work correctly. Tests were created for each route with various valid and invalid routes for each test. This allows us to run our test suite and ensure that our website is loading successfully. Our unittest suite achieves a 99% coverage of our python code and with the

only missing line being the final line of the program, only used for running the website locally. Of course, this does not mean that our website is working properly but it is a great indicator that the various pages can load correctly.

Selenium tests were used to verify the GUI's functionality. The Selenium Chrome Extension was used to generate a .side file (located in our GitHub repository) that outlines all tests that were performed. Test cases were created to verify navbar functionality, ensure pagination worked as designed, and validate linkages between instance pages. A comprehensive test case was also included to evaluate additional extraneous features such as homepage carousel buttons or model page links within each grid component.

SOFTWARE TOOLS AND FRAMEWORKS

The backbone of our application is a MongoDB database server hosted on Google Cloud Platform. This server was populated with data from the three API's referenced in our model section with a set of Python scripts that performed API requests, data aggregations, and database insertions at different levels.

Furthermore, the Flask framework was used to develop our backend web application. Our Flask application specifies definitions for our model objects and specifications for how to respond to different kinds of requests. The application is deployed to the Heroku cloud, where it services requests and interacts with our database.

For our frontend, we employ Jinja2 as a template engine to inject Python objects into the user interface. We also use the Bootstrap CSS framework to polish the appearance of our pages and enable responsiveness and the DataTables library to make

For unit testing our flask code, we used python's unittest module. This module is widely used for testing python code and is well documented on the python website making it a good choice for our website. For testing of our GUI, we developed test cases using Selenium. This allowed us to verify the functionality of all website components.

REFLECTION

Phase 1 Reflection:

After assessing what went well for our team and taking note of the things we struggled with, there are many key takeaways that can help streamline and improve the development process for later phases of the project. Beginning with things our team did well, all members were incredibly receptive to each other's needs which prevented people from falling behind and ensured that everyone stayed on track. Updating and closing issues on Git as they came up also ensured that all members were on the same page in terms of the project's completion status. Additionally, our meetings were organized, focused, and productive, which greatly contributed to our team's ability to finish all parts of Phase 1 by the deadline. Our system for committing changes and updates also proved to be very effective. By issuing pull requests for any file changes and requiring at least one other team member to review them, we were able to avoid merge conflicts and ensure that our master branch was always working. To streamline project development, we also went ahead and implemented a database using mongoDB even though doing so was not required.

Although many things went well for our team, there is always room for improvement. For most tasks, our actual completion times differed, sometimes significantly, from our initial estimates. Now that we have become more in-tune with each other's workflows and productivity levels, we can better estimate completion times in later phases of the project. Additionally, we could have improved how tasks were delegated. Although our team dynamic was good and all members contributed, some team members ended up taking on more responsibility than others. Because of this, task delegation is something we can refine in later phases to make the spread of work more equitable across all team members. Another thing that would have been beneficial was to get more headway on the project during earlier stages of the phase. Majority of our work happened in the week and a half leading up until the phase deadline. Making more progress earlier on would have allowed us to further refine our website instead of saving such changes for later phases. Although not an issue to date, we also should have allocated more time for possible setbacks. If there had been a huge unforeseen bug in our code, we likely would have been pressed for time to address it. Finally, some team members ran into errors due to not routinely pulling the most recent changes before beginning work. Being more cognizant of this would

have helped streamline the development process and avoid unnecessary problems. Despite these minor missteps, our team was largely successful in achieving our goals and collaborating to fulfill all the requirements of Phase 1.

To summarize:

Five things that went well:

1. Team members were very receptive to each other's needs
2. Organized, focused, and productive meetings/standups
3. Updated and closed issues on Git as they came up so that team members were on the same page
4. System in place for issuing and reviewing pull requests so that master branch was always working and bug-free
5. Implemented database using mongoDB to streamline project development even though doing so was not a requirement

Five things we struggled with:

1. Accurately estimating time to complete tasks
2. Room for improvement in how tasks were delegated
3. Would have been beneficial to get more headway during the early stages of the phase.
4. Should have allocated more time for unforeseen setbacks
5. Making sure to routinely pull changes before beginning work

Phase 2 Reflection

Overall, phase 2 was a better experience than phase 1 due to the takeaways and lessons we learned and the improved team chemistry. We started early, identifying key bottlenecks and dependencies to ensure no one was blocked. We also did a good job distributing the work, ensuring everyone could make forward progress without too many dependencies and a fair distribution of work. Members continued to be responsive, and GitHub was used extensively for efficient collaboration.

However, we still encountered some pitfalls that we hope to improve upon in the next phase. We had even more trouble estimating time to completion for the stories and tasks in this phase. This

is not entirely unexpected due to the fact that we are a new team, and therefore not great at estimating our velocity yet, and also purely because we are learning new technologies at the moment. It can be difficult to estimate the completion times for portions of this project since all of us are learning as we go and occasionally certain stories are harder to complete than expected and others are easier.Furthermore, we were less consistent with meeting times, and last minute changes were not uncommon. This is likely due to the increased workload of the semester in all classes, which can unexpectedly shift schedules and priorities.

We summarize our takeaways from phase one by identifying key successes and shortcomings:

Five things that went well:

1. The team was proactive and started early to ensure we had enough time to complete the phase.
2. The team identified bottlenecks and dependencies to improve the efficiency of delegation.
3. The work was distributed evenly, and everyone put in their fair share.
4. Members continued to be responsive and communicate conflicts.
5. GitHub was used often and consistently to keep track of progress and tasks.

Five things we struggled with:

1. Time estimation was inaccurate and led to more pressure at the end of the phase.
2. Meeting times were less consistent and often shifted around.
3. Last minute changes happened occasionally.
4. We struggled familiarizing ourselves with new frameworks and concepts that we have not seen before.
5. Could have tried to work more out of our comfort zones and try new approaches.

REFERENCES

- Cloud Atlas for MongoDB: https://www.youtube.com/watch?v=rE_bJl2GAY8&ab_channel=TechWithTim
- W3Schools Bootstrap tutorial: https://www.w3schools.com/bootstrap/default.asp
- Flask MongoEngine: http://docs.mongoengine.org/projects/flask-mongoengine/en/latest/
- Stack Overflow: https://stackoverflow.com/

    - https://stackoverflow.com/questions/3206344/passing-html-to-template-using-flask-jinja2
    - https://stackoverflow.com/questions/8189702/mongodb-using-an-or-clause-in-mongoengine
    - https://stackoverflow.com/questions/19607851/use-bootstrap-grid-with-variable-number-of-cells
    - Many, many more
- Making API requests: https://www.dataquest.io/blog/python-api-tutorial/
- Scraping image URL's: https://towardsdatascience.com/image-scraping-with-python-a96feda8af2d
- Football-API documentation: https://www.api-football.com/
- Twitter API documentation: https://developer.twitter.com/en/products/twitter-api
- FutDB API documentation: https://futdb.app/
- Pagination: https://gist.github.com/mozillazg/69fb40067ae6d80386e10e105e6803c9
- Python unittest documentation for writing our python tests: https://docs.python.org/3/library/unittest.html?highlight=unittest#module-unittest