Stefano DelPiccolo
Miguel Vazquez
Gabe Raymondi
CSCI 4448

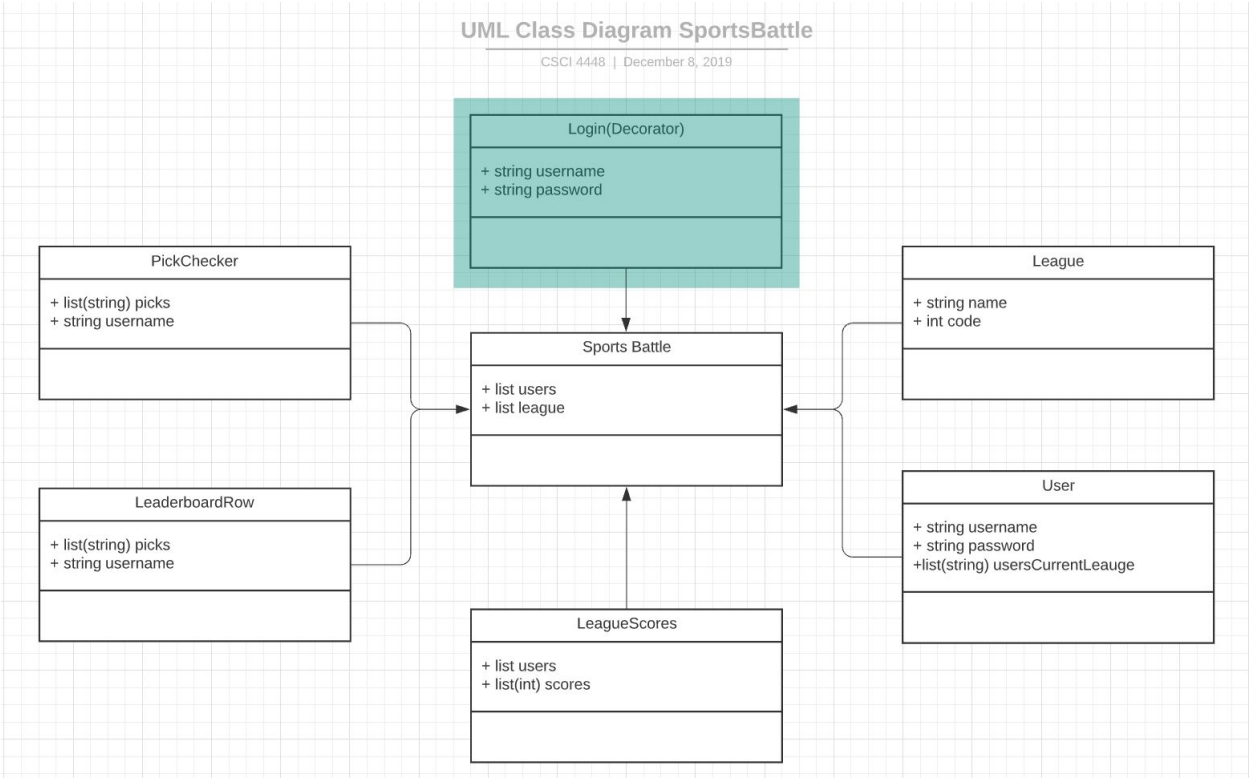
Final Report: Sports Battle

As a team, we were able to design and build a functioning website that creates users in order to compete against other users at picking scores for NFL games. Every week each user will pick new teams based on matchups and build their score from their correct matchups at the end of the week, and if they have a higher score than their opponent, they win! We were able to implement most features described in your project 5 uml class diagram, however due to our large scope and short time we had to leave certain features out as well we describe later in the report.

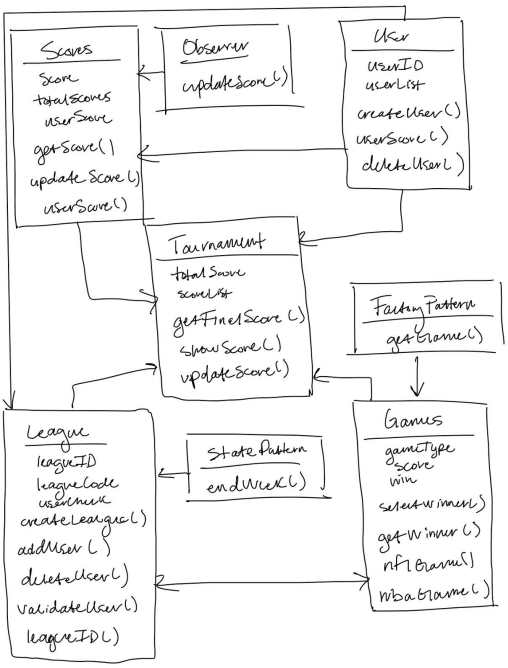
We programmed our app using python and hosted our app using a microframework called Flask. Our frontend consists of HTML and CSS that displays the interface of our app.

UML Class Diagram

Project 6



Project 5



The overall theme of our project stayed the same, we are still creating leagues and users for users to compete against each other. The biggest change comes from the change in scope of our project. Because we didn't have as much time as we wanted, we removed some features shown in our previous uml class diagram. We weren't able to add certain features like removing users, however we are able to validate users by creating logins. We also weren't able to add two sports like we wanted to (NBA and NFL) but were able to successfully create a site for one of them.

Third-Party Code

Throughout our project design process and implementation we knew we wanted to create a website that could host and record local sport battles, we just didn't know exactly how we were going to do that, so we did some research. A lot of the third-party code comes from our sportsreference API which is where we get all of the data from sports games as well as our framework for hosting our website online, Flask. Our original work is seen in the implementation of these tools because we designed the website ourselves. From our user interface designed frontend to our collection of data in the backend, our website is one of a kind. Although our website is unique, we did hit some bumps along the road that required more research and debugging and we have below a list of the resources that we used to make our final project deliverable a success.

1. Sportsreference API
    a. Sportsreference is a free python API that pulls stats from www.sports-reference.com and allows them to be easily used in python-based applications. Sportsreference exposes a plethora of sports information from major sports leagues in North America. However in our case we will only be focusing on the National Football League.
2. Flask Web App with Python
    a. Using the microframework known as Flask, we can create url routes, flask pages, and pass variables.
3. Passing data to html page using flask
    a. Helped us connect a part of our backend to the frontend html page.
4. Jinja2
    a. Jinja2 is a modern day templating language for Python developers. It is used to create HTML, XML or other markup formats that are returned to the user via an HTTP request.
5. To prevent cached responses from the browser
6. Flask tutorial
    a. Helped us learn how to generate more elaborate web pages that have a complex structure and many dynamic components.

- Statement on the OOAD process for your overall Semester Project
    - List three key design process elements or issues (positive or negative) that your team experienced in analysis and design of the OO semester project

Conclusion

Here are three key design process elements that our team experienced in analysis and design of our semester project.

1. Project Scope
    a. In the beginning of our design process, we had a lot of great ideas on what we wanted to develop. When it came down to it, our project was going to be a website that created users to add to leagues to compete in different sports leagues. Now that being said, we had so many features we wanted to implement but unfortunately couldn't because our scope was to large and we only had a limited amount of time. This delayed our progress a couple times because we kept having to shorten the scope of features we wanted to implement. We believe that even though we had great ideas, the possibility of those getting done in time weren't as realistic and we didn't include the time spent on learning the microframework that we were using. This was a negative issue that happened throughout our project

timeline, however we learned a value lessing on doing a better job managing our project scope within our timeframe.

2. Design Perspectives
    a. As we mentioned before we all had lots of ideas on the features we wanted to implement into our app and we each came up with a couple ideas on how we were going to design it. We looked at what we wanted to create as a whole and split it into different pieces, also known as functional decomposition which we learned in class this semester. Decompose the project into small pieces and then build up from there which object-oriented programming follows with classes and objects. This year we learned that design is a process of synthesis, a process of putting things together, a process of combination. This helped us create a well design that ultimately made a fully functional system. In software, well designed systems respond to well to change. In the end we were able to experience the goal of design. Design pieces, in our case, classes and objects, within the context in which they must live in order to create robust and flexible systems.

3. Refactoring
    a. During our project progress we realized that our code wasn't neat at all and didn't make a lot of sense when we went back to look at it, so we decided to refactor our code. Refactoring is a controlled technique for improving the design of an existing code base. We made a few changes to our code including adding classes to help our system flow better. The cumulative effect of each of these transformations is quite significant. By doing them in small steps you reduce the risk of introducing errors. You also avoid having the system broken while you are carrying out the restructuring - which allows you to gradually refactor a system over an extended period of time. Overall refactoring improved the design of our software, made the software easier to understand, and made it easier to add remaining features. The only problem with refactoring was the slowing down of new features. Although, the whole purpose of refactoring is to make us program faster, it did cost us time and debugging to get our system running again.