Andrew Brown (ajb5384)

Bibartan Jha (bj8355)

Morgan Murrell (mmm6855)

Alina Nguyen (amn2763)

Team E9: Technical Report

Team Information

- Project Name: NBA & WNBA Database.
- Github repo link: https://github.com/UT-SWLab/TeamE9
- Team Canvas Group: E9

Our team members' information is listed in the table below:

Name	UT EID	Email	Github Username
Andrew Brown	ajb5384	ajbrown2017@utexas.edu	ajbrown99
Bibartan Jha	bj8355	jha.b1bartan@utexas.edu	bibartanjha
Morgan Murrell	mmm6855	murrellmorgan@utexas.edu	m0rganm
Alina Nguyen	amn2763	alinanguyen@utexas.edu	alinanguyen00

Team Members' Information

Motivation

The purpose of this site is to serve as a tool for NBA & WNBA fans who want to keep up with the statistics of their favorite players and teams, as well as the latest news around both leagues. We wanted to provide a website that allows the user to view all the information they want to see, without having to go to several different websites. Through the information and interactive features on our website, we not only hope to introduce users to basketball but also keep current fans up-to-date about the NBA & WNBA.

One purpose of this website is to serve as a repository of information for users to learn more about the NBA and WNBA. Different pages on this site have been created to allow users to look through biographical information for different players, latest statistics for various teams, details regarding several NBA-related articles that have been written in the past few months, and information about current NBA coaches. Furthermore, this website gives users the opportunity to learn more about NBA history: not only can users learn about which teams have won the NBA championship each year from 1947 to 2019, but they can also find out about the record-holding franchise leaders for each team in the league.

Another purpose of this website is to provide many interactive features to users so that users have fun learning about basketball. Users can create their own basketball team on our fantasy basketball simulator and see which roster will give them the best stats. Furthermore, fans can even add their favorite teams and players to their favorites for easier access to information. There is also a feature where fans can easily compare the statistics and information about different players and teams side by side on the Compare page. If fans wish to preserve this information, they can create an account on our website.

User Stories: Phase One

For Phase 1, one of our user stories was, "As a user, I want to be greeted by a front page that shows the latest and/or random basketball scores". This means that when the user goes to the home/front page, they are able to see scores from the most recent or popular basketball game. The user can assume that we are pulling scores from all of the NBA games in our MongoDB database and then randomly choosing 5 games to display. The estimated time we predicted for this user story was 4 hours, and the actual amount of time it took is about 5 hours.

Our next user story is "As a user, I want to navigate easily through the website using a universal navigation bar, so I can find the information I want to view." This will allow the different pages to be linked and for easier access for the user as they can travel any of the main pages no matter where they are currently on the website. For this use case, the user can assume that we created the navigation bar to where it will list out all of the unique pages for the website. We estimated this user story to take 1 hour and the actual time it took was 1 hour.

Our third user story was "As a user, I want to be able to change my account settings and overall website settings." This means that the user can alter elements specific to their account, such as their password or their favorite players list. The user can assume that we use Pyrebase4 to allow them to login, change account settings, and change overall website settings. We estimated this user story to take 1 hour and the actual time it took was 1 hour.

Our fourth user story was "As a user, I want to have an archive of all of the previous basketball champions." This means that the user will be able to see a list and timeline of all the winners of different championships and awards. The user can assume that we retrieved the information all from our MongoDB database. We estimated that this user story would take 4 hours and it actually took about 8 hours to complete.

For our last user story, we had "As a user, I want to be able to view information on basketball coaches." This means that the user can view a page that will list out all of the information of the coaches of different basketball teams. For this user story, the user can also assume that we retrieved the information about the current basketball coaches from our MongoDB database. We estimated this user story to take 2 hours and the amount of time it actually took was about 4 hours.

User Stories: Phase Two

For phase 2, one of our user stories was “As a user, I want to be able to filter players by team.” This means that, on the players model page, a user will be able to choose a team and only view the players on that selected team. The user can assume that the API provides accurate information regarding which team each player is on. We expected that this would only take 1 hour, but the amount of time it ended up taking is 2 hours.

Our next user story is “As a user, I want to view lots of multimedia on each instance page, including social media links and options to add the instance to one of my favorites.” The user can assume that, in each instance page, they will be able to access various pictures and links that will give them more information about that instance. We expected that this would only take 2 hours, but this ended up taking around 6 hours because we had to collect information for over 180 different instances and update our files and MongoDB accordingly.

Our next user story is “Developers found additional sources to collect multimedia from (other APIs, other websites, etc.).” The users can assume that the developers did not just collect information for each instance from the nba-api, but also from other APIs as well for purposes such as player images, latest news articles, etc. We expected that this would only take 2 hours, but it ended up taking around 5 hours to complete.

Our next user story is “As a user, I want to have multiple pages for each model, with only 9-10 instances on each page so that I don’t have to keep scrolling (ease of use).” This means that when a user clicks on a model page such as players, they will not just view all 100 player instances on one page but instead be able to navigate through different pages that contain 9 players each. This will allow model pages to load more quickly and players will need to wait so long for model pages to load. The user can assume that developers put in pagination for the model pages for this purpose. We expected that this would take 1 hour, but it ended up taking 3 hours.

Our next user story is “As a user, I want to be able to choose two teams and compare stats side-by-side.” The user can assume that the developers retrieved stats from all the teams from the appropriate API. We expected that this would take 2 hours, and it actually did end up taking us 2 hours. Our final user story for Phase 2 is “As a user, I want to be able to sort players based on parameters such as name (alphabetically), year started (earliest to latest), and more.” The user can assume that developers retrieved information from different players in regards to each sorting parameters - such as name, start and end year, team, etc. We expected that this would take 1 hour, but it ended up taking around 3 hours to complete.

User Stories: Phase Three

For phase 3, one of our user stories is “As a user, I want each page to be user-friendly”. This means that on each page the user will not have to scroll that much or click on too many links to see all the features on that page. We expected that this would only take 2 hours, but the amount of time it ended up taking is 4 hours in order to optimize all the pages.

Our next user story is “As a user, I want to be able to access the players and team instance pages directly from the comparisons page.” This means that if the user wants to view more information about a player or team, they can immediately access this information while they are looking at their comparisons chart on the comparisons page. We expected this to take about an hour and the amount of time it actually took was less than an hour.

Our next user story is “As a user, I want to be able to search for players, teams, and news by different categories. I also want to be able to filter my search based on different basketball categories.” This means that for our model pages, the user should have the ability to search for instances based on specific basketball characteristics. We implemented a Google-like search and various filters/sorts to assist the user. We expected this to take around 4 hours and the actual amount of time to implement also took 4 hours. The biggest challenge was implementing stacked filters for our model pages. The rest of the search implementation was very straightforward.

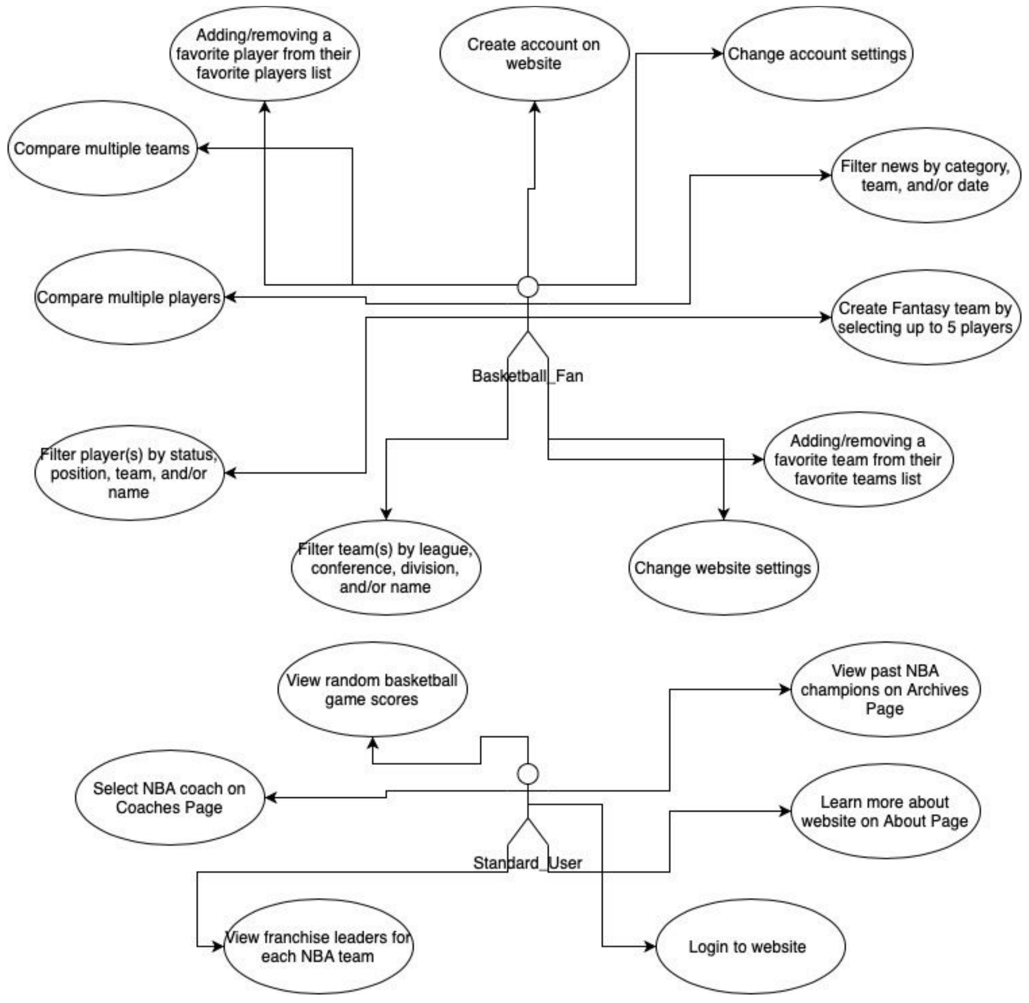
Our next user story is “As a user, I want to see lots of information about each player and team, such as social media links, multiple pictures, and different tables containing information about that player or team.” This means that there will be different types of multimedia on each instance page, such as links to other websites, likes to other models on this website, social media links, images, and tables. We expected that this would take 3 hours, but it ended up taking 6 hours because we had to collect information and images for all 180+ model instances.

Our next user story is “As a user, I want to be able to view a visually-appealing website”. The user can assume that each page will be enjoyable to view, and that each page will have consistency within the stylization (e.g., Player, Team, and News Model pages will look similar).” We expected this process of adding CSS to each page would take 6 hours, but in reality, it required us 10 hours to complete. However, we are still working on this user story, so the total amount of time is expected to increase.

Design

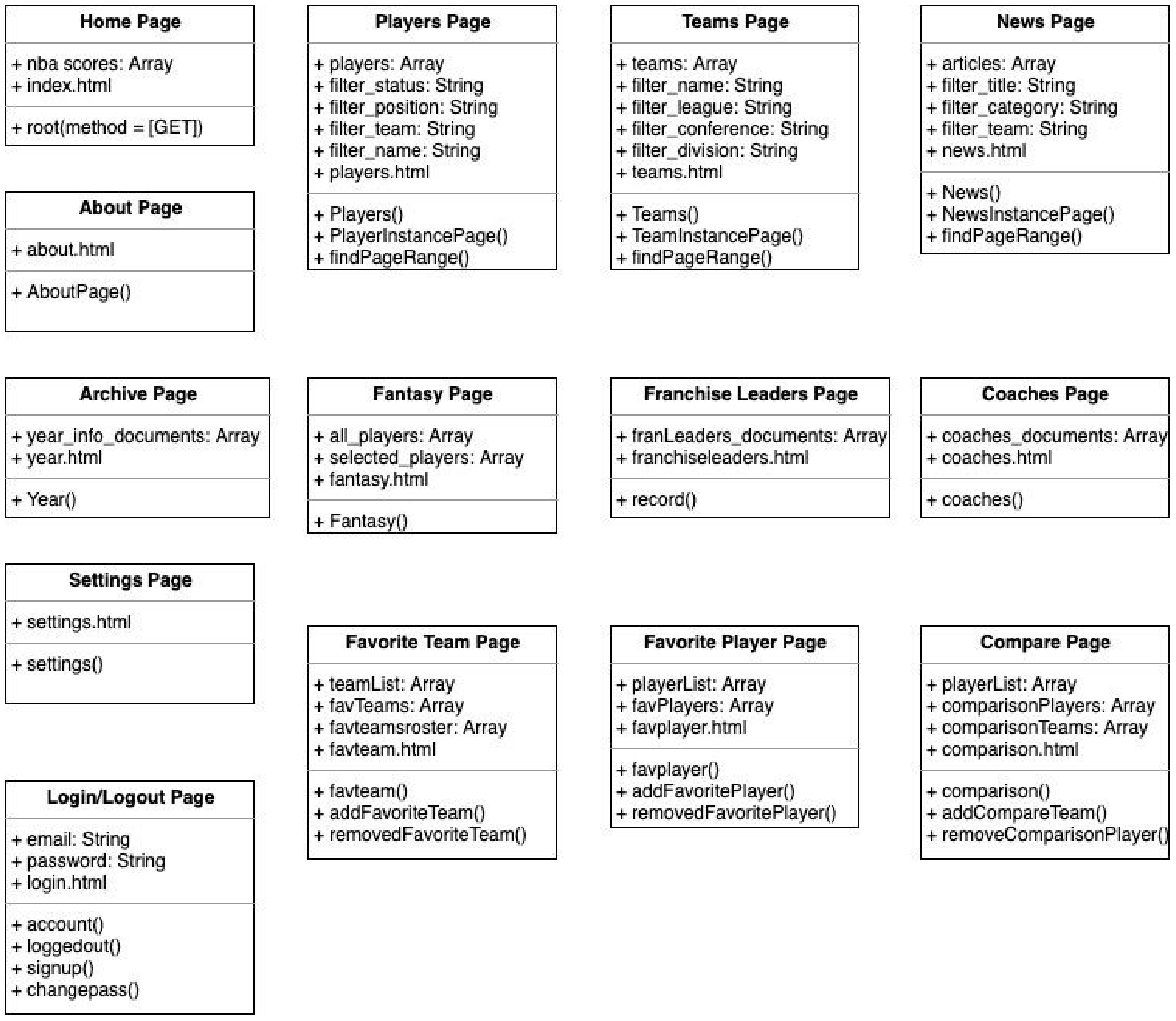
We decided to create a use case diagram because we believe that this type of UML diagram is most applicable for our website. Our actors were the users of our website. More specifically, we defined our actors as either standard users or avid basketball fans. We assumed standard users are people that generally have a moderate interest in basketball. Therefore, most of the use cases associated with standard users revolved around the basic functionalities or our website. Those use cases are “login to website”, “view About Page”, “view random basketball scores on Home Page”, “view past NBA champions on Archives page”, “view NBA coaches on Coaches Page”, and “view franchise leaders for each NBA team.”

For basketball fans, their use cases mainly consisted of the more complex functionalities of our website. Their uses cases are “compare multiple teams”, “compare multiple players”, “adding/removing a favorite player from their favorite players list”, “adding/removing a favorite team from their favorite players list”, “create an account on website”, “change account settings”, “filter basketball news by team or category”, “create a fantasy team”, “change website settings”, filter teams by league, conference, division, or name”, and “filter players by status, position, team, or name.” The use case diagram is shown below.

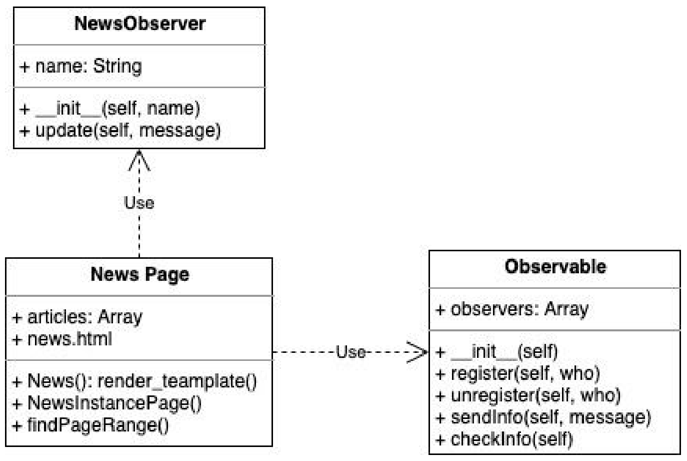

Adding/removing a favorite player from their favorite players list
Create account on website
Change account settings
Compare multiple teams
Filter news by category, team, and/or date
Compare multiple players
Create Fantasy team by selecting up to 5 players
Basketball_Fan
Filter player(s) by status, position, team, and/or name
Adding/removing a favorite team from their favorite teams list
Filter team(s) by league, conference, division, and/or name
Change website settings
View random basketball game scores
View past NBA champions on Archives Page
Select NBA coach on Coaches Page
Learn more about website on About Page
Standard_User
View franchise leaders for each NBA team
Login to website

Use Case Diagram for NBA & WNBA Database

We also decided to create a class diagram because we believe that this UML diagram would also provide an accurate representation for our website. We actually did not create any classes in our backend for our website. In our class diagram, we treated each page on our website as its own class. The main data elements for each page are the fields for each class. The functions that each page performs are the methods for each class. The classes are “Home Page”, “Players Page”, “Teams Page”, “News Page”, “About Page”, “Fantasy Page”, “Franchise Leaders Page”, “Coaches Page”, “Archive Page”, “Favorite Team Page”, “Favorite Player Page”, “Comparison Page”, “Settings Page”, and “Login/Logout Page”. Since users access each page separately from the navigation bar, then we did not add any associations nor generalizations between multiple pages. We consider each page as independent with its own fields and methods. The class diagram is shown below.



Class Diagram for NBA & WNBA Database



Class Diagram (with Observer Pattern)

Models, Multimedia, and Sources

For our project, our 3 models were Players, Teams, and News.

Our Players page contains 100 different selected players from the NBA. On the Players page, we made a grid with a box for each instance. Each player's box contains a picture of that player, along with some key information and a link to that player's instance page.

Our Team page contains 42 different instances of basketball teams (30 NBA teams and 12 WNBA teams). Just like our players page, here we made a grid with a team in each box. Each team's box contains a picture of that team's logo, along with some key information and a link to that team's instance page.

Our News page contains 30 different instances of recent news articles involving the NBA. Just like our players and teams page page, here we made a grid with information about a different news article in each box. Each article's box contains a picture of each article's author, along with some key information and a link to that article's instance page.

For our multimedia, we gathered different images of the players and the team logos from Wikipedia and Google images and displayed them wherever their corresponding team or player is labelled. For the general Players page, each player listed has their image with a few general info about them. For the general Teams page, each team listed has their team logo and some general info about the team. This also applies for any players or teams added to the user's Favorite Players page and Favorite Teams page. We also utilized these images for the Players and Teams instance pages. When a user views an instance page, they are able to see multiple images either of the player or of the basketball team. On our News page, we have different articles listed and also include a picture of the news source, such as the author or the organization. In addition, we included the picture of the coach on our Coaches page and images for the Franchise Leaders page.

Our team collected data and included source links for different pages of our website. For our News page, we included the links to the original sources of the news for each news article that we listed. For our player and teams instance pages, we included the links to their social media accounts such as Facebook, Instagram, and Twitter, also we added links to their Wikipedia pages. If a player/team doesn't have a certain social media account such as Instagram, then the link for the Instagram just directs back to the same instance page. For all instance pages, we also added links to other instance pages that each instance corresponds to (i.e. a player's instance page will link to the instance page for the player's team). These links are also visible when you add a player or team to your Favorite Players or Favorite Teams page.

Tools, Software, Frameworks

We used several tools and libraries to help us be able to build this website and deploy. These tools include: Django, Heroku, Github, Pyrebase4, and PyMongo.

We used Django for pagination purposes. In order to make it easier to implement pagination for our model pages, we used the Paginator class in the Django framework in order to select how many instances we will put in each page.

In order to deploy our website, we used Heroku. We found that Heroku was much easier to work with and was able to upload our website with no problems. As for sharing code amongst each other, we used GitHub and made sure to push code often to keep the website up-to-date as much as possible on each team member's end. We also stayed on track with our user stories by using the Projects tab and taking note of every story that we were to complete.

One of our user stories involved the average user being able to make an account and add various players/teams to their favorites page for easier access. We completed this by using Firebase's Authentication Tool through a Pyrebase4 library, since Firebase did not offer a Python option on their own site. To clarify on why we wanted to use Python with Firebase, as opposed to using it with Javascript: we wanted to make sure that when a specific button is clicked, a function that inputs the user's information and chosen team/player into MongoDB is called. A MongoDB collection cannot be accessed using Javascript, so we found that coding this entire process in Python would be the most simple way. Lastly, in order to access MongoDB, we would need to create a connection to the website using PyMongo.

Sources and Data

NBA API endpoints github: https://github.com/swar/nba_api

We used NBA_API to gather information about each player, team, and coach. This included season statistics, attended schools, player information, and more. This information was used for our instance pages. Even though this API's title just says 'NBA', this also contains all the information for the WNBA teams.

Sports Data API endpoints documentation:
https://sportsdata.io/developers/api-documentation/nba

We used the Sports Data API in order to gather information about the latest news articles regarding the NBA. This was used for the News Articles model page.

Sports Radar API endpoints documentation: https://developer.sportradar.com/docs/read/Home

We used Sports Radar API to gather information about games throughout the NBA regular season. This was used for the random games that appear on our homepage.

User Account database and Heroku App deployment:
https://www.youtube.com/watch?v=Z1RJmh_OqeA&ab_channel=freeCodeCamp.org

We used the Flask tutorial in order to set up Flask for our website and to deploy to Heroku. Flask was very useful for creating our website and all of our pages. This tutorial was also helpful in figuring out any of our Heroku issues when we deployed at the end of each phase.

Django for Pagination: https://samulinatri.com/blog/django-pagination-tutorial/#full-tutorial

We used the Paginator class in the Django framework for our instance pages to select how many instances we will put in each page.

Pyrebase4: https://github.com/nhorvath/Pyrebase4

We used Pyrebase to create a connection via Python to the Firebase Authentication tool. This was used to create user accounts on the website.

MongoDB: https://www.mongodb.com/blog/post/getting-started-with-python-and-mongodb

We used MongoDB to have all of the data used for our website easily-accessible and well-organized. This was used on basically everything part of the website (homepage games, user account preferences, information on each instance page, etc.)

Testing

Backend: For the backend testing, we used the unittest module. Most of the unit testing was testing the functionality of our MongoDB database. To test our MongoDB database, we would create mockup data and perform CRUD (create, read, update, delete) operations for each database collection to see if the database will work as intended (i.e. does the database store the mockup data in a specific collection). In order to avoid actually writing test data to our MongoDB database, we would store all of the elements in a MongoDB collection into an array and then perform the testing. We would check our results using the ASSERT(...) functions to check if the testing performed the way we intended. In Phase III, we added unit tests for the Accounts collection. More importantly, due to our MongoDB changes from Phase II to Phase III, we had to remodify our original unit tests. In Phase III, we modified some of the fields in most of the MongoDB collections. These changes caused our original unit tests to now fail. Therefore, we had to go back to our old unit tests and make sure they are now compatible with the new MongoDB changes.

GUI: For the GUI testing, we used the Selenium Webdriver tool. The primary goal for our GUI testing was to check the navigation bar, scrolling ability, pagination, log-in/out capabilities, searching function on the model pages, randomness of the games listed on the front page, addition/removal of a favorite player, and various buttons on the website (e.g, Coaches page, Fantasy page, etc.). For the navigation bar, the Selenium Webdriver clicked through each navigation link and compared the current URL to the expected URL. For the scrolling ability, the webdriver scrolled slowly through the team/players page. For pagination, the webdriver clicked through a few pages on the team/players page and compared the current URL to the expected URL for each new page. For the log-in/out capabilities, the webdriver used 'seleniumtest@gmail.com' and 'password' to log into an existing account and then log out. The current URL was used to check if it performed the correct actions. The searching function on the model pages was tested by searching for various players, using the appropriate filter, and checking if the expected results were shown. For the addition and removal of a favorite player, the Webdriver logged in using the account mentioned before, added a player to their favorites, and removed via the Favorite Players page. If the test was able to successfully remove the player, that would show that the feature is working properly. The front page showcases random NBA games from the 2019 season. This was tested by using the webdriver to compare the first game listed before and after refreshing the page. The results for this test were printed to 'selenium_test.txt' for easy-viewing.

More GUI features were added to the website during phase 3. These features include a viewable list of teams to choose from when visiting the Coaches and Franchise Leaders page and a selection of years/decades to choose from when viewing the Archive page. Both the Coaches and Franchise Leaders page have similar layouts where the user can choose a team to view related information and go back using the "See all" button. This feature was tested using Webdriver to go through each team and view the related content. Lastly, the Archive page was tested using Webdriver to click through each decade button and confirm that the correct set of years would be visible.

Frontend: For frontend testing, we used Selenium instead of Mocha. We were having issues with Mocha, so we decided to switch to Selenium Webdriver to test our frontend. We believe that using Selenium instead of Mocha would not make much of a difference in terms of the quality of our frontend testing. The main goal for the frontend tests was to have our webdriver test the CSS attributes of various elements on our website. These elements include: the log-in button, homepage table, players model item, players model table, players instance page, and log-in page inputs. This test also compared the layouts amongst each model page to test for style consistency. To test these elements, we located the elements on our deployed website and compared the received CSS attributes to our expected values. The results for this test were printed to 'selenium_test.txt' for easy-viewing.

Reflection

Phase I

What did your team learn?

During this phase, we learned more about using Google Cloud Service, gathering data from APIs, and storing into and reading from MongoDB. We were able to get a better understanding on how to create a functioning website with Flask and Python, and connect it to GCP to be deployed as a web app. For the APIs, we learned more about the NBA API specifically, and were able to write some test code that could read information using the functions they provided. As for MongoDB, all four team members were new to using it, so we had to learn more about being able to take information and store it in a collection. After learning more about it, we were able to take information from the NBA API and store it into our database for the website. We also learned how to use the Project feature in github in order to organize all of our user story ideas, page ideas, and the API's that we are using.

Five things you struggled with and need to improve?

We struggled with figuring out how to store information in MongoDB and read from collections. We also struggled with using Google Cloud Service and transferring code between GitHub and Cloud. We need to improve our knowledge on Bootstrap CSS so we can apply more CSS styling to our pages. We also need to improve our knowledge on Javascript to make the pages more interactive. Lastly, we need to improve on adding user accounts (by learning more about Firebase's Authentication feature) and more collections in MongoDB to keep track of our News model.

Five things that just worked well?

A lot of things went well for us during Phase I. Team cohesiveness was definitely a big strength. We were able to divide some of the work by assigning different user stories to each member while also working together on the About Page. In the process, we were very transparent with what each group member had/had not accomplished. We also had a lot of success creating all of the HTML files for all of our pages. We were able to use Flask effectively to allow the user to navigate between different pages and send requests. We were also able to use Jinja effectively to have HTML pages extend other HTML pages. This really came in handy when we wanted to make sure that all pages had a navigation bar at the top with links to all the other pages. In addition, we have already started incorporating bootstrap for various website features, such as the navigation bar, and created base stylesheets for our different pages.

Phase II

What did your team learn?

During this phase, we learned how to add paginations to all of our model pages. Before pagination, we had all of our instances on a single model page. Now that we have pagination, our model pages are much more user-friendly and more interactive. We also learned to add features that allowed users to sort and filter instances based on different parameters. Since our website is basketball-related, some of our parameters are current status, position, conference, team, name, and location. This is another way to make our website more user-friendly and more interactive. In this phase, we also learned how to connect our MongoDB through PyMongo without having any issues deploying to Heroku. In Phase I, we had already figured out how to take information from our API's and store it into our MongoDB database for the website. However, in order for MongoDB to work when we deployed to Heroku, we had to allow network access for all IP addresses for our database. While this might be an information security vulnerability, not allowing network access to all IP addresses was the biggest issue for us when we deployed to Heroku. We also learned how to connect to Firebase using Pyrebase. Using Pyrebase was extremely helpful for implementing login/logout features and creating accounts on our website. We also learned how to create frontend and backend tests for our website. We learned how to use different testing tools (such as unittest, Mocha, and Selenium) to test the functionality of our database and our website applications.

Five things you struggled with and need to improve?

A number of our issues during this phase revolved around users interacting with our website if they were logged in to their account that they had created on our website. We struggled with adding favorite players and teams to each user's specific document in our MongoDB database. We also struggled with adding a Favorites button to each instance page that would add information to our MongoDB database when clicked. Our initial idea was to have the page stay the same after the button was pressed. This led to bugs and issues on our website. We were able to resolve the issue by having the user be sent to a confirmation page once he/she pressed the button. We also struggled with creating user authentication with Firebase and JavaScript. To resolve this issue, we switched to Pyrebase. We struggled with syncing pagination with Flask. We resolved this framework by switching to Django framework rather than continuing to use Flask. Our last big issue was creating frontend tests with Mocha. We were not able to check CSS attributes with Mocha so our workaround was to use Selenium for our frontend and GUI tests for our website.

Five things that just worked well?

We had quite a few things work well for us during Phase 2. We were able to add social media links for our teams and players in our MongoDB database very easily. We added more multi-media to our instance pages without much difficulty. We now have a login/logout button based on whether a user is logged in to our website or not. We also can add users to our MongoDB database if someone creates a new account along with updating their specific document when they add a team/player to their favorites list. Implementing MongoDB has been our biggest success in this phase. We were able to add/modify/delete elements in our MongoDB database very easily.

Phase III

What did your team learn?

During Phase III, the biggest feature we learned was adding pagination to our model pages. This new feature was very user friendly and helped divide our instances to span several pages instead of having all of them on one single page. We also learned how to view a specific range of pages for pagination as well. Along with pagination, we learned several new style features for all of our pages. We learned how to modify box-shadow for cards and how to stylize different text boxes and navbars. We came up with different user-friendly layout ideas to have less scrolling and clicking for the user. These new ideas made us reconsider how we wanted some of our pages to look. As a result, we decided to redesign our pages with these new ideas in order for our pages to look more dynamic and visually appealing. We learned how to use JavaScript to add features that scroll to certain parts of the page as another way to lessen the amount of scrolling for the user. Finally, we learned how to create Google-like searches and filters for our model pages. We decided to go a step further and create stacked filters to provide a more accurate categorization of our instances. All of these features that we learned in Phase III were used to provide a more comfortable experience for anyone that uses our website.

Five things you struggled with and need to improve?

There were a few issues that we faced with our website during this phase. One of these issues was being able to remove players and teams from the favorites pages. Another issue was formatting the rosters and championships on the favorites teams page. In MongoDB, the rosters and championships are each listed as one long string for each team, and we still need to format them so that each player on the roster and each championship is on a separate row in the table on our page. Another issue we faced was with coming up with creative CSS ideas to make each page more unique. Back in Phase 2, most of our pages just had tables filled with information. It was difficult to figure out different creative ways to format the pages so that each page looks unique and appealing. Another struggle that we faced was with syncing the pagination on our model pages with the searching and filtering options. For our pagination, we found that when the user clicks on a different page the filters and searches are no longer being placed. So, we still need to improve our pagination so that the filters and searches stay in place regardless of which page the user navigates to. Our final struggle was with finding and collecting additional data and images for each instance page. Back in Phase 2, our instance pages each just had one image and one table filled with info. For this phase, we had to display more information (such as social media links) on each instance page. Since we have over 180 instances amongst our 3 models, it was definitely a struggle to collect information from the APIs and store that information into MongoDB from each and every instance. For this issue, we also struggled with figuring out what exact additional information we want to add for each instance.

Five things that just worked well?

In Phase III, we were successful in coming up with new info, features, and links to add to our instances pages. For example, on our teams instance pages, we added different links to news about that team and also the instance pages of some of the players on that team. In addition, we were able to optimize our page layouts for the coaches, archives, and franchise leaders pages to make it more user friendly and visually appealing, such as adding the team icons and buttons to help direct the user and adding pagination or scrolling shortcuts. Overall we were successful in adding CSS to each page for style and design consistency, such as creating a uniform theme with a certain color layout for each page. We also made all the grids of players, teams, and news into a card layout for a cleaner look and to make all the grids on different pages consistent. Lastly, we were able to make the comparisons page more dynamic by adding and removing players and teams and also including the links to redirect to their instances pages.

Phase IV

What did your team learn?

During Phase IV, one of the biggest things we learned was adding the observer pattern design. We implemented the observer pattern for our News model page, in which the observable is our

Sports Radar Rest API and our Observer is our News collection on our MongoDB. Through implementing the observer design pattern, we are able to update our MongoDB News collection whenever new articles are placed in the Sports Radar Rest API. We also learned how to implement information hiding for our searching, sorting, and filtering features for our model pages. We accomplished this by creating classes for each model page that stores all of the information needed for searching, sorting, and filtering. The class also executes the logic needed for these features. One thing we also learned in this phase is how to refactor our code in order to improve the internal structure and make our code easier to read. We did this by combining different routes, combining HTML code, using for loops to improve how we formatted tables, using dictionaries to send our variables when we rendered HTML files from our python code, and moving pieces of code (such as our filter classes) to separate Python files.

Five things you struggled with and need to improve?

During Phase IV, we implemented the observer pattern method which would have our website listen for any new data about sports news that may have been added to the SportsRadar API. However, sometimes the data is unattainable because you need a premium account in order to do an unlimited amount of API calls, which gave us some challenges in testing out our observer pattern and for updating our website. Also, in the beginning of Phase IV we struggled coming up with ways to apply the Observer Pattern since there aren't many new basketball players and teams that are often added to the API. However, we were able to apply the pattern on our News model since News updates occur more frequently. We also struggled with coming up with ideas on how to implement information hiding, since we didn't have any classes in our main.py. However, we were able to create new classes for each model page that stores and implements all of the search, sort, and filter functionalities. In addition, we ran into some issues with our observer pattern causing some errors on our News model page due to certain news articles that had teams or authors/sources that aren't in our database. This led to those news instances not having an image linked with it or having errors since our website didn't know the teams and players involved. Lastly we also ran into problems with our pagination as we stacked certain filters and words in our searching and sorting. The issue was that a user's filters and sorting selections wouldn't continue to apply as they went to the next page of instances. However we were able to collaborate and revise our code to resolve these issues.

Five things that just worked well?

During Phase IV, we added information hiding to our code. The implementation of information hiding was very straightforward. Likewise, implementing the base code for the observer pattern, as well understanding the code flow, was also quite straightforward. As mentioned previously, the observer pattern was used to notify the News page (observer) when the SportsRadar API updated with recent news. It was easy to understand how the Observable would check for new "News", how it would notify the Observers, and how the Observer would update the MongoDB collection to reflect the new “News” onto the website. Furthermore, one comment that we got on our Phase III report was that we were not stacking the different filtering, stacking, and sorting functionalities on our model pages. In Phase IV, we were able to successfully stack these 3 functionalities by using one common HTML form for all 3 (rather than 3 separate HTML forms). Another thing that worked well in this phase is refactoring. We were able to refactor our code by combining similar htmls into one and combining similar routes in our main.py, thus reducing lines of code and the amount of html pages. Through refactoring, we were able to restructure our code by making it easier to read while keeping the functionality the same.