# NBAOlogy - Final Report

## Project Overview

1. Briefly describe what the project accomplished.
2. Discuss the usefulness of your project, i.e. what real problem you solved.

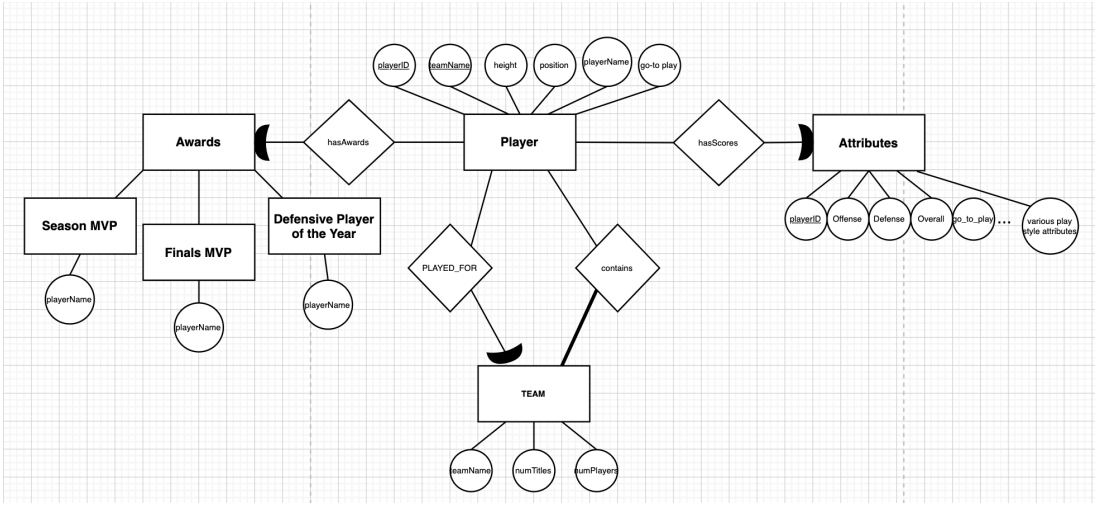
As NBA fans, our group wanted to utilize a database that allowed new and longtime NBA fans to learn more about players in the league, both past and present. Currently, there is no simple, user-friendly way of finding advanced statistics and interesting information regarding a fan's favorite players. NBAOlogy solves this issue. Our website allows users to search a database full of past and present players, create their own player and specify attributes, create a team and determine the strength, find detailed information about a player's playstyle breakdown, and more. There are even multiple versions of players, so users can compare a player during their prime on a certain team to years when they weren't as strong. Our website allows any fan, from casual to hardcore, to learn everything they want regarding past and present NBA players.

## Data and Schema

1. Discuss the data in your database
2. Include your ER Diagram and Schema
3. Briefly discuss from where you collected data and how you did it (if crawling is automated, explain how and what tools were used)

Our data consists of information on over 7000 players that are currently playing or have played in the NBA. Attributes include name, ID, height, numerous offensive and defensive metrics, and intangible metrics including stamina, strength, and hustle. Our MongoDB database consists of information regarding players' awards won, championships won, All-Star appearances etc. Our SQL database with player information and attributes is collected from the NBA2K video game database(https://2kmtcentral.com/). Our MongoDB database with past awards and accomplishments is collected from Basketball Reference (https://www.basketball-reference.com/).

**ER Diagram:**




**Schema:**

Table Set Up:

- The main table is derived from CSV data called nba_full_table
- The players, teams, and attributes tables are derived from nba_full_table and have calculated columns added

attributes_table:

INSERT INTO attributes_table (id, overall, play1, shot_close, shot_mid, shot_3pt, free_throw, offensive_consistency, driving_layup, driving_dunk, post_moves, speed, vertical, strength, stamina, hustle, ball_handle, passing_accuracy, interior_defense, perimeter_defense, steal, block, defensive_consistency, offensive_rebound, defensive_rebound, offensive_overall, defensive_overall)
SELECT
id, overall, play1, shot_close, shot_mid, shot_3pt, free_throw, offensive_consistency, driving_layup, driving_dunk, post_moves, speed, vertical, strength, stamina, hustle, ball_handle, passing_accuracy, interior_defense, perimeter_defense, steal, block, defensive_consistency, offensive_rebound, defensive_rebound,
(shot_close + shot_mid + free_throw + offensive_consistency + driving_layup + driving_dunk + post_moves + ball_handle + passing_accuracy + offensive_rebound) / 10 as offensive_overall,
(interior_defense + perimeter_defense + steal + block + defensive_consistency+ defensive_rebound) / 6 as defensive_overall
FROM nba_full_table

players_table:

INSERT INTO players_table (id, name, team, position, height)
SELECT id, name, team, position, height
FROM nba_full_table

Information:

- ER diagram utilizes concepts of a relational mySQL database with NoSQL/MongoDB
- the 4 tables/entities are player, awards, teams, calculated scores
- calculated scores table will be a mongoDB collection calculated using data from the NBA2k data set that will give offensive and defensive scores based on our design team's metrics
- First query  utilize graph relations to filter all players matched by similar team, position, and go-to play entities
- second query compare offensive and defensive scores to player being searched for
    - this reduces the run time as comparisons to players of different positions and play styles will be eliminated
- third query compare awards won

1. discuss how you used a NoSQL database in your project
2. discuss your design decisions related to storing your app data in relational vs. non-relational databases.

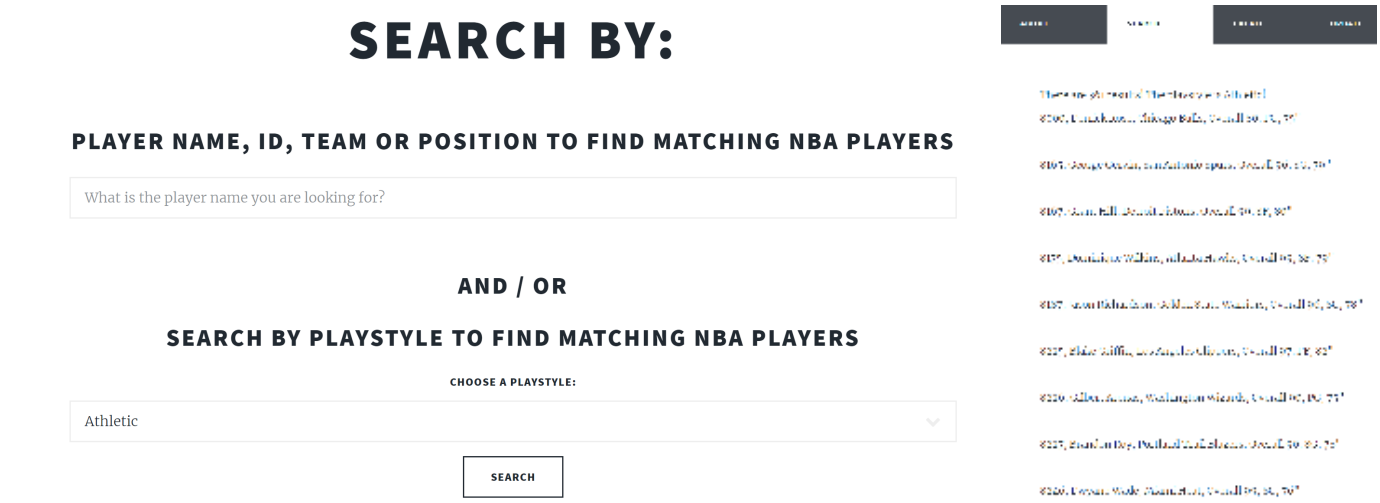
## Functionality

1. Clearly list the functionality of your application (feature specs)
2. Explain one basic function

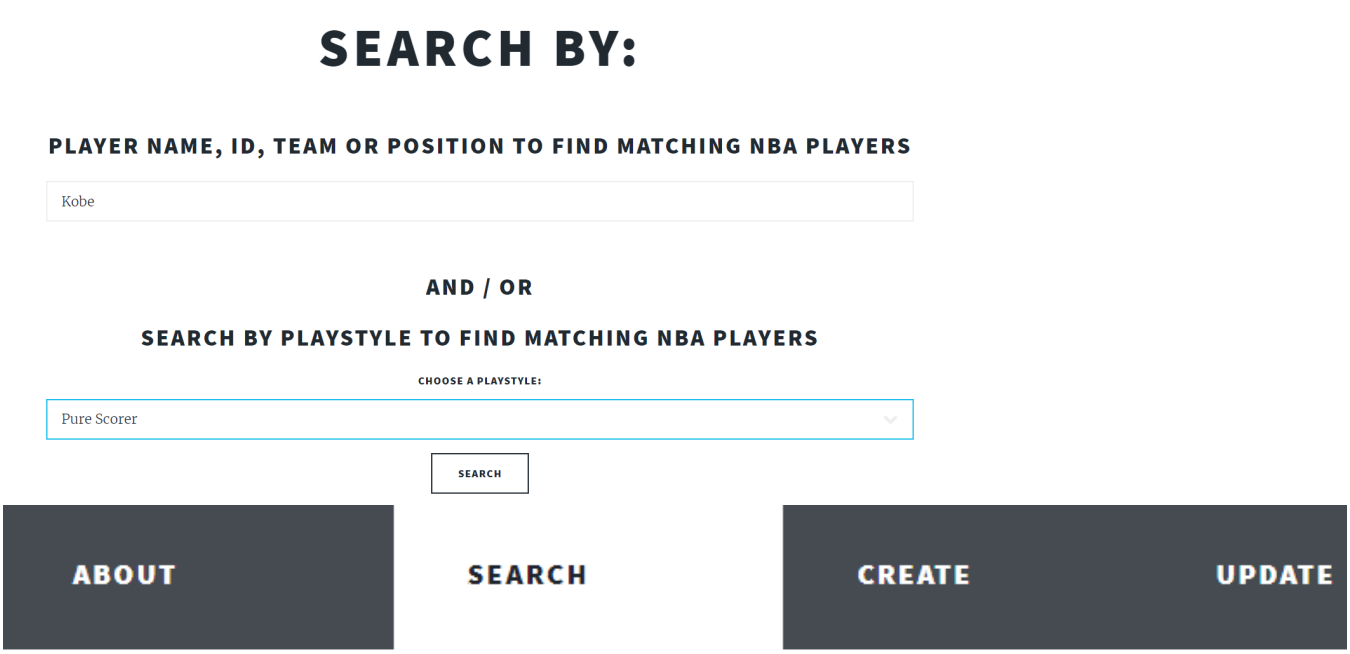
A basic function in our application is searching for players in our database. When we search a player, we can either search by entering a string of characters that a players name in our database matches, or we can search by play-style.

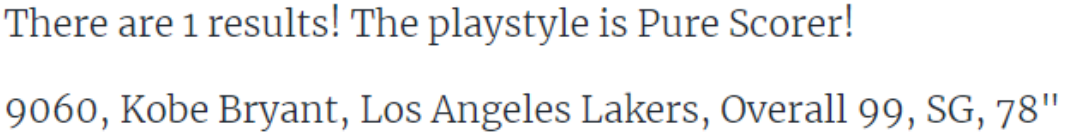
If we search by string, all players whose names contain the fed-in characters will be returned. For example, if I were to search for all players in our database with "Abdu" in their name, I would get 8 results of players with "abdu" located somewhere in their name, along with their PlayerID, Team, Overall rating, Position, and Height. Here is a screengrab of when I search "abdu" using our application:


If we search by playstyle, we are given a drop-down menu of 7 playstyles a user can filter player results by. These play-styles were determined through using algorithms to group players into playstyle groups based on their stats, devised from research of how NBA2K filters their players. These are screengrabs of how our play-style search feature works on our front end:




Finally, a user is able to search by both a players name AND their playstyle. If there is no such player with a given play-style, the search result returns NULL. In any other case, a player would be returned. Here is an example using Kobe Bryant given his playstyle of "Pure Scorer:




**MYSQL**

```php
<?php
if (isset($_POST['submit-rank'])) {
    $search = mysqli_real_escape_string($con, $_POST['search']);
    $sql = "SELECT DISTINCT players_table.id
    FROM players_table NATURAL JOIN attributes_table
    WHERE players_table.name LIKE '%$search%'
    AND attributes_table.overall = (
        SELECT MAX(attributes_table.overall) AS maxOverall
        FROM players_table NATURAL JOIN attributes_table
        WHERE players_table.name LIKE '%$search%')";

    $result = mysqli_query($con, $sql);
    $idSelected = mysqli_fetch_assoc($result)['id'];

    $namesql = "SELECT name
    FROM players_table
    WHERE id = $idSelected
    ";

    $nameResult = mysqli_query($con, $namesql);
    $nameSelected = mysqli_fetch_assoc($nameResult)['name'];

    echo $nameSelected . " playstyle breakdown:" . "</br>";

    $attsql = "SELECT play1, post_moves, passing_accuracy, ball_handle, shot_3pt, shot_mid, defensive_consistency,
    interior_defense, perimeter_defense, speed, vertical, stamina, driving_dunk, offensive_rebound,
    defensive_rebound, shot_close, free_throw, driving_layup, offensive_consistency, block, steal
    FROM attributes_table
    WHERE id = $idSelected";

    $attresult = mysqli_query($con, $attsql);
    $row = mysqli_fetch_assoc($attresult);
```

The above code snippet shows the MYSQL code for advanced functionality 1. We first use the search bar to locate the player id for the associated name. Since there can be multiple players of the same name entered more than once in the database, we use the player with the highest overall for the playstyle breakdown. We then use the player id to pull all attributes from the attribute table. The query is run with the php mysql connection to our hosted database and attribute data is stored in variables to be used in calculations to rank a searched players playstyle breakdown.

**MONGODB**

1.
```php
$client = new MongoDB\Client(
  'mongodb+srv://dilanp2:Redracer3304!@nbaology.6qhnv.mongodb.net/NBAology?retryWrites=true&w=majority');

$db = $client->test;


$seasonMVP = $db ->seasonMVP;
$finalsMVP = $db ->finalsMVP;
$defensivePlayerOfTheYear = $db ->defensivePlayerOfTheYear;


$pgNumMVP = $seasonMVP->count(array('name'=> $namePG));
$pgNumFinalsMVP = $finalsMVP->count(array('name'=> $namePG));
$pgNumDPOY = $defensivePlayerOfTheYear->count(array('name'=> $namePG));

$sgNumMVP = $seasonMVP->count(array('name'=> $nameSG));
$sgNumFinalsMVP = $finalsMVP->count(array('name'=> $nameSG));
$sgNumDPOY = $defensivePlayerOfTheYear->count(array('name'=> $nameSG));

$sfNumMVP = $seasonMVP->count(array('name'=> $nameSF));
$sfNumFinalsMVP = $finalsMVP->count(array('name'=> $nameSF));
$sfNumDPOY = $defensivePlayerOfTheYear->count(array('name'=> $nameSF));

$pfNumMVP = $seasonMVP->count(array('name'=> $namePF));
$pfNumFinalsMVP = $finalsMVP->count(array('name'=> $namePF));
$pfNumDPOY = $defensivePlayerOfTheYear->count(array('name'=> $namePF));

$cNumMVP = $seasonMVP->count(array('name'=> $nameC));
$cNumFinalsMVP = $finalsMVP->count(array('name'=> $nameC));
$cNumDPOY = $defensivePlayerOfTheYear->count(array('name'=> $nameC));
```

The above code shows the connection to the MnogoDB database of awards. There are 3 collections: season MVP, finals MVP, and Defensive Player of the year. The collections only hold the names of players who have one these awards. In the above code, a connection is established first. Then queries are made to get the count of each award one by each of the 5 players in the team analysis. These counts are then factored into a weighted average with player offensive and defensive attribute data.

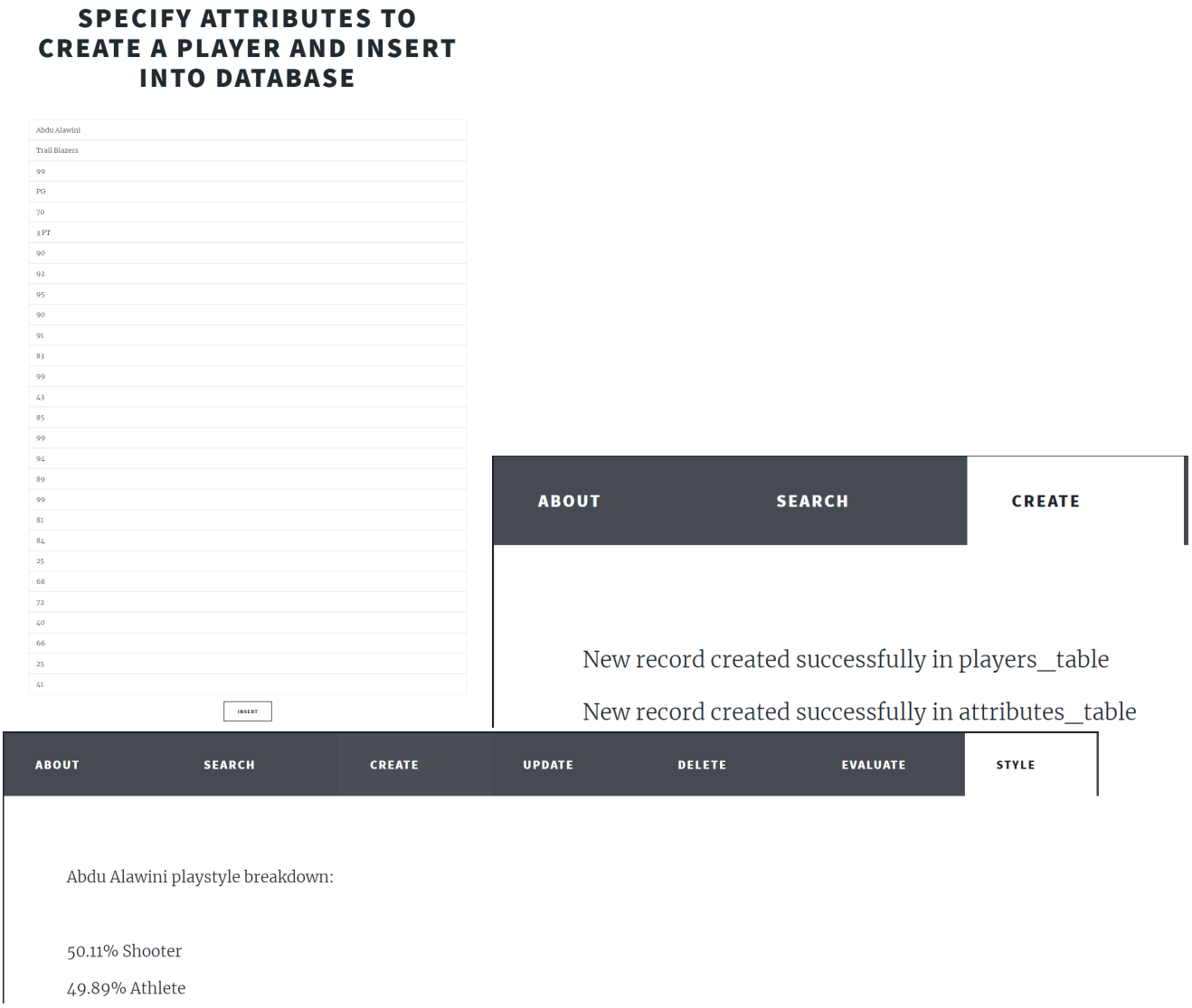
## Advanced Functionality

**Explain your advanced function 1 (AF1) and why it's considered as advanced. Being able to do it is very important both in the report and final presentation.**

Our advanced function 1 is a "Create-a-Player" functionality. Create-a-Player essentially allows a user of our application to insert a player with unique statistics (that the user himself decides) in order to compare that player against others in our database. The player is assigned a specific playstyle (based on our 7 groups of playstyles) and is returned with a breakdown of all of their playstyles that is calculated from their user-fed-in statistics. This can be used in multiple real-life situations, such as updating our database with each new NBA draft class, which would require us to insert all the rookies entering the league. Based on specific rookies statistics, a user can see the exact breakdowns of each playstyle the rookies have - which would be useful to a scout, NBA coach, or a fan that is interested in learning more about each rookies style of play. Our playstyle breakdown works for newly created players in our database as well as already existing players.

What makes this functionality advanced is how our inserted-players information is returned in our database when they are searched for. Through the creation of play-style grouping algorithms that were devised from doing research on what NBA2K uses as criteria for each individual play-style as well as several test-queries to see which players would fit our playstyle algorithms accurately, we were able to create a grouping of 7 unique play styles. What this lets us do is when any player in our database is searched for (including newly created players), their breakdown of playstyles is displayed. A user can create any combination of a players statistics, and our algorithm will return their breakdown depending on which playstyle groups the player fit into.

Here is an example using a created player with a random combination of stats:




As you can see, based on Professor Alawini's fed in statistics, we were able to return a breakdown of his primary playstyles thanks to our devised algorithm.

## Challenges and Work

**Describe one technical challenge that the team encountered.**

During setup for the MongoDB Atlas cloud hosted database, we spent tens of hours trying to download drivers and extensions to allow connection to our NoSQL database. We originally used CPANEL but connecting to MongoDB on this application was nearly impossible, so we moved our application onto a localhost using XAMPP. This continued to give us an issue with downloading the MongoDB driver into our remote hosted folder. We worked with Professor Alawini, TAs, and other students to establish this connection to the databse, but we were unfortunately unsuccessful. Instead of pulling data in our web application, we used hard coded values just to simulate the functionality. We were luckily able to connect to MongoDB Atlas via Shell and demo the queries we would have integrated into the web application. This is shown below.

```
switched to db NBAology
MongoDB Enterprise atlas-km0729-shard-0:PRIMARY> db.seasonMVP.find({name: "Michael Jordan"}).count()

5
MongoDB Enterprise atlas-km0729-shard-0:PRIMARY> db.finalsMVP.find({name: "Michael Jordan"}).count()

6
MongoDB Enterprise atlas-km0729-shard-0:PRIMARY> db.defensivePlayerOfTheYear.find({name: "Michael Jordan"}).count()
1
MongoDB Enterprise atlas-km0729-shard-0:PRIMARY>
```

**State if everything went according to the initial development plan and proposed specifications, if not - why?!**

In our initial development plan, we decided on having 3 main tables - a players, teams, and an awards table - all of which were successfully implemented. Other than our pre-planned schema, not much else went according to what we thought would happen when we first created an outline for our project. At first, we were unsure of how to make a functionality in our application that was actually considered to be "advanced". We ended up deciding on two main features involving interpretation of a players/teams statistics in order to evaluate their overall score. We were successful in making advanced function 1 work, however when implementing advanced function 2, we ran into a problem of not being able to connect our web hosting service to our noSQL database, which contained all our information regarding accolades for players and teams. After hours of watching YouTube videos and assessing stack overflow examples, we found a unique solution to having our Advanced Function 2 work. This period of time in which we worked with new concepts that were foreign to us - such as connecting our databases to our front end and filtering players by their statistics/teams/accolades - really challenged our minds as a group and forced us to work collectively to come up with solutions to our significant issues with our application. Even through our times of struggle, we were able to successfully make NBAology into a web-app that is able to assess the advantages, overall ratings, and playstyles of selected players in our database. We learned numerous skills pertaining to database management, front-end, and back-end development, but were most impacted by overcoming our greatest issues as a team thanks to working on our web-application, NBAOlogy.

**Division of Labor**

Full-stack(front-end/Back-end): Luc Dowell

Implemented UI for our application, provided an interface for database connections.

Back-end/DB: Dilan Patel

Set up both MySQL and noSQL databases.

Back-end: Abhimanyu Kapoor

Created algorithms to filter players and make advanced function 1 and 2.

Back-end: Mukund Madhusudan

Created advanced queries and worked on advanced function 1 and 2.