Ben Wasserman

CPSC 408

2280906

5/17/19

Naïve Sports Betting Application

**Overview**

The application that I chose to create is a sports betting application. At this point in the development process, I limited the development to one sport: basketball. The application allows users to register, login, query for specific games, make picks, edit picks, see account information and delete their account. The README.md has details on where are all the project requirements are implemented.

**Related Applications**

There are many related applications that exist, but the market for niche sports betting applications is endless. Oddsshark, Sportsline, CBS and ESPN all maintain services and software for NBA betting. That said, there are many variations for every sport, and people create their own logic to fit their specific pool. The motivation for this project came from the way that my dad runs a golf betting pool. He has all 100 people email him his picks for the week and he uses an excel spreadsheet to implement the logic. This is slow and inefficient. I created this proof of concept NBA betting application to show that an application can be created to make sports betting easier and more efficient for people that want to bet with their friends and family.

**Framework and Dependencies**

The web application was developed using ASP.NET Core MVC. The application also leverages some third party packages. A lightweight and fast object relational mapper, Dapper, is utilized to help with querying the database. Dapper is known to be very efficient, and the programmer still has the freedom to write the queries completely as SQL. On the front end, jQuery and AJAX are used for dynamic page loading. For example, on the /home/games page the user can select different time periods where they would like to see the games. This page is dynamically uploaded using an AJAX call to a REST API that was created and queries the database. Similarly, when a user edits a pick the drop down menu automatically populates in the table in place of the previous pick. This is achieved with an AJAX call. Most of the data is loaded using AJAX calls to the REST API endpoints that I created.

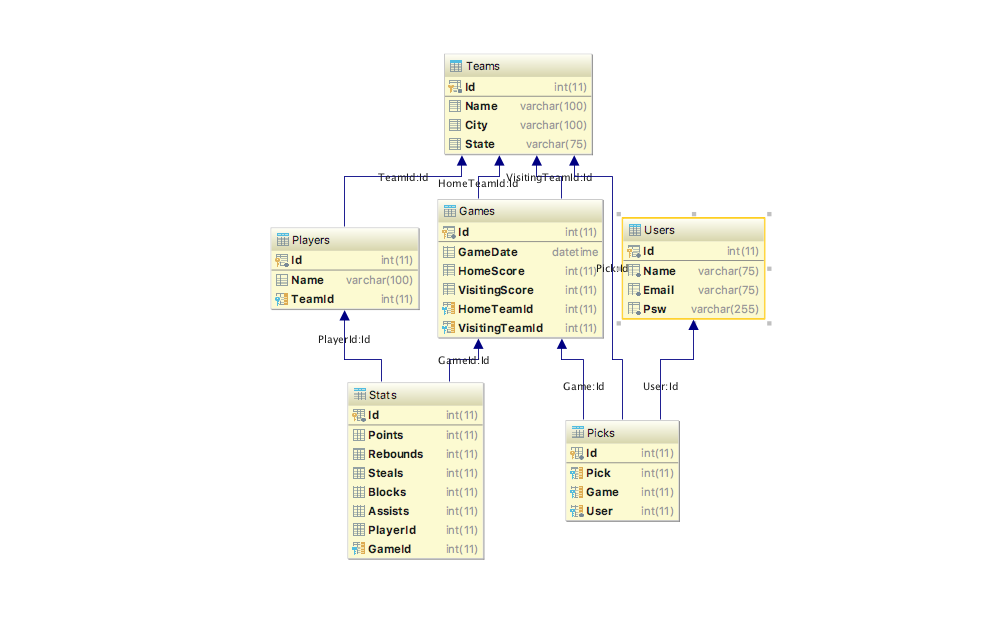
**Front End Implementation**

On the front end, bootstrap is used for most of the css and styling. This package can be found in ~wwwroot/lib/bootstrap. There is also some custom css that is implemented to make the page look a little better. This can be found in ~wwwroot/css/site.css. The html scripts are located in ~Views/Home. Models are passed into the Views and then properties of each model are accessed within the Views to display and query data.

**Back End Implementation**

On the back end, most of the queries are implemented with Dapper in the directory ~Controllers/ApiControllers/. The REST APIs are built out for Users, Games, and Picks. The user can query the GameApiController to get picks for a specific time period (upcoming games, previous games, or today) and the user can specify the games they would like to see by team. The most complex queries involve getting a user’s picks and getting the games for specific time periods for a specific team. To retrieve a user’s picks four inner joins are used to populate the Pick entity that gets passed to the Picks.cshtml page. The query joins the Games table on the Teams table based on the HomeTeamId and VisitingTeamId attributes (both foreign keys). It also joins the Picks and Games tables so that the id of the game, the game date and the picks can be obtained. The query to get the games for a specific time and for a specific team uses two joins to get the Teams and the Game information as well as a nested query to get only the games for the team that is selected from the drop down menu. Transactions are utilized in the bulk pick feature of the application. Users can select multiple picks from the Games page and then submit all the picks at once. These queries are executed as a transaction so that all the picks are either committed or rolled back. There is also error checking with the roll back. Stored procedures are used for inserting a user when someone registers and for login authentication. Passwords in the database are recorded using a one way hash and a salt for security.

**Schema Diagram**

****

The schema diagram above shows all the tables that are implemented and their attributes. There are foreign key constraints that link each table to ensure referential integrity is implemented. The foreign key for the User attribute within the Picks table implements a cascading deletion so that when a User is deleted, all their picks are deleted as well. All the other foreign keys use strict constraints.

**Future Work**

There is a lot more work to be done in order to make this application shippable. I plan to add a UserTeams table that will allow users to create their own pool and invite friends and family. I also plan to add the following features which will make the application more usable, scalable and relevant within the sports betting market:

* *Connect to API for live scoring.* Live scoring is crucial to the NBA betting application. RapidApi hosts an NBA API that has endpoints for all the data that I would need: teams, games, scores, players, statistics, etc. This API isn’t used in this version of the application because the free version only allows a limited number of hits per day.
* *Implement peer to peer payment.* Peer to peer payment is the next thing that I plan to implement so that users can settle their bets within the application itself. There are a variety of packages and plugins that can help with this on the implementation and security end.
* *Allow users to customize their pool with their own betting logic.* The idea for this application came from the notion that users would be able to customize their pool with their own logic. This is the stretch goal for the project. Team administrators would be able to go into the settings page and specify their own rules that would allow the application to be customizable for any individual sports betting application.