Parlay: A Web Application Designed to Facilitate Competition in the Form of Sports Betting Leagues

A Manuscript

Submitted to

the Department of Computer Science

and the Faculty of the

University of Wisconsin–La Crosse

La Crosse, Wisconsin

by

Drew Lohmeyer

in Partial Fulfillment of the

Requirements for the Degree of

Master of Software Engineering

December, 2021

Parlay: A Web Application Designed to Facilitate Competition in the Form of Sports Betting Leagues

By Drew Lohmeyer

We recommend acceptance of this manuscript in partial fulfillment of this candidate's requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

Abstract

This manuscript outlines the software development processes and principles followed throughout the creation of Parlay, a sports betting web application designed for entertainment and competition. Users join together in leagues and place sports bets using virtual currency. The user with the most virtual currency at the end of a league is declared the winner. The requirements, design, implementation, testing, and deployment of Parlay are all described in depth below in an effort to encapsulate the project's entire development effort.

Acknowledgements

I would like to express my sincere gratitude to my project advisor Dr. Kenny Hunt for his continuous support and guidance. His vast knowledge of the Web Development field was very beneficial to me throughout the development of my project. I would also like to thank the entire Department of Computer Science at the University of Wisconsin–La Crosse for the countless hours of instruction along the way. To my family and fiancé, thank you for all of your encouragement throughout my entire academic journey.

Table of Contents

List of Tables

List of Figures

Glossary

Burndown Chart

A burndown chart tracks the total amount of work remaining in a project versus time. It provides a visualization of the speed of the development process and can be a useful tool when estimating when a project will be completed.

DNS

The Domain Name System (DNS) is responsible for mapping domain names typed in a URL search bar to their corresponding server IP addresses.

JSON Web Token (JWT)

JSON Web Token is a data transmission protocol that can be used as a secure communication strategy between two parties. Tokens are digitally signed to establish a level of trust in the identities of both token senders and receivers.

Moneyline

The moneyline contains the odds, or likelihood, that a specific team will win a game. The moneyline can be used when placing bets to predict which team will win a game.

NPM

Node Package Manager (NPM) is a package manager that can be used across any JavaScript-based framework such as Angular and Node.js. The package manager accesses 3rd party tools from the NPM registry, or library, that can then be used as dependencies for various project needs.

Odds

Odds (American odds) are numeric values that describe how likely an arbitrary outcome is to occur. Negative values indicate that an outcome is more likely than not to occur and positive values indicate the opposite. Also, odds directly correspond to bet payouts. For example, if the odds for an outcome were -200, $200 would need to be risked in order to profit $100. If the odds for an outcome were +200, then risking $100 would profit $200.

Reverse Proxy

A reverse proxy is special proxy server that sits in front of a web server and redirects all network traffic to a separate server before reaching the intended origin.

Spread

The spread for a sporting contest is an estimate of the margin of victory or defeat for the teams competing in the contest. A negative spread value for a team indicates that the team is expected to win the game by that value. A positive spread value for a team indicates that the team is expected to lose the game by that value.

Total

The total for a game is the estimated number of combined points scored by both teams competing in a game. Bets can be placed using the expected total value by predicting if the actual total value will be over or under the expected value.

UTC

Coordinated Universal Time (UTC) is a universal time standard that is used to regulate global time. UTC time is located at the Prime Meridian in Greenwich, London and is not impacted by Daylight Savings Time

1. Introduction

Each year, as the summer months wind down and the leaves begin to turn, there is one additional season in the air besides fall: football season. In 2019, nearly 17 million people attended National Football League (NFL) games across the United States [1] with another 180 million watching games on either cable or broadcast television over the course of the season. The National Collegiate Athletic Association's football league (NCAAF) attracts an immense fan base as well. The top NCAAF sub-division, the FBS, includes 130 universities dispersed across the country that compete in packed stadiums filled with students, alumni, and others cheering on their respective teams. In the current day and age of extreme data collection and advanced analytics, sports fanatics have started to find more ways to entertain themselves besides simply attending or viewing these football contests. One of these entertainment venues is through participation in the global sports betting market which is estimated to grow to a valuation of $155 billion by 2024.

Sports betting (or sports gambling) is the act of placing a sports bets. A bet consists of an amount of currency a bettor is willing to risk, one or more sporting contest predictions, and the amount of currency that the bettor will gain if all predictions are correct as a payout. Payouts are determined by the likelihood that all predictions contained in a bet will be correct. For example, risking $100 on a team that is not expected to win a game (an underdog) will payout a much greater amount if successful as compared to successfully risking that same amount on a team that is heavily favored to win.

According to a recent study by sports betting news outlet PlayUSA, optimistic football fans are expected to risk more than $20 billion in the form of sports bets on the 2021 NFL and NCAAF seasons. It is also estimated that the companies brokering these bets (sportsbooks) will pocket over $1.5 billion or 7.5% of the total money at risk across the betting landscape. [2] So, what keeps people coming back? Bettors crave the thrill and excitement of watching their predictions play out during dramatic sports contests all while having the opportunity to gain financially.[3] Sports gambling also carries aspects that allow a person to feel like they are in control. Unlike blindly pushing a button on a casino slot machine, bettors have the opportunity to analyze past game data and take prediction advice from "the experts" in order to convince themselves they have a statistical advantage over the sportsbook providers.[4] However, despite the growing popularity of sports betting, one thing will remain certain moving forward; the average participant will continue to lose. Sports betting odds and payouts are set using complex algorithms that will always favor the house over time.

"Parlay: the sports betting league" is a web application designed for entertainment purposes that enables users to place bets on upcoming football contests. Rather than competing against sportsbook providers that will always have an unfair advantage, users compete against one another using *virtual currency* in organized groups called leagues. This virtual league-based betting format creates a unique environment in which users still experience the sports gambling highs and lows but with no risk and a much more attainable goal of performing better than your friends who are playing by the same rules.

Users are able to log onto this web application and create leagues. After creating a league, a user is able to invite friends to create a Parlay account and join the league. Parlay breaks leagues down into weeks that begin and end every Tuesday morning. Each week all league participants are required to risk some fixed percentage of their currency by placing bets on upcoming NFL and NCAAF games. Game data, such as betting odds and live scores, is automatically updated periodically, and bets are evaluated based on final game results. As weeks progress and bets are placed, users attempt to rise to the top of their league's standings by winning bets and gaining virtual currency. If at any point throughout the duration of the league a participant's balance hits $0.00, he or she is eliminated from the league. The last user remaining or the user with the most currency at the end of the league is declared the winner. Overall, these leagues create a friendly but engaging sports betting interface targeted for individuals who enjoy watching some of the most talented athletes in the world compete against one another on the football field.

2. Requirements

2.1. Overview

This section first details the development processes used by the author to complete the project and also describes how these processes relate to two software development life cycle models. This discussion is followed by a detailed explanation of the functional and non-functional requirements considered for this project.

2.2. Life Cycle Model

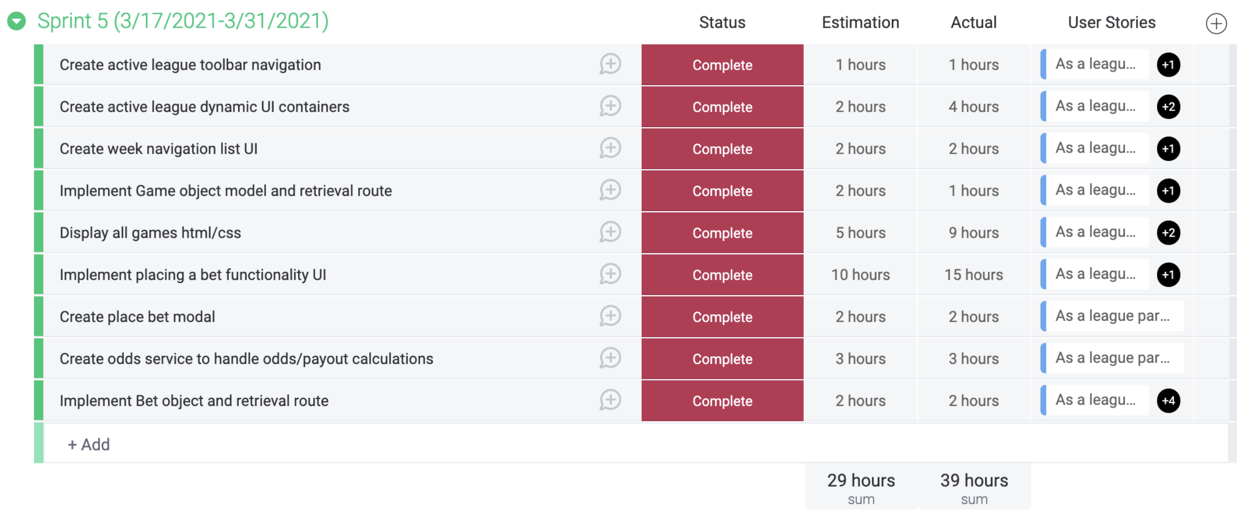
Software development life cycle models lay the foundation for how a project should be developed over a long period of time in order to ensure deadlines are met and an exceptional final product is produced. There are many different models currently in use across the software development industry. Each of these models have various strengths and weaknesses and can be considered more or less advantageous based on the characteristics of a project such as team size, time constraints and development complexity.[5] For this project, a combination of both incremental prototyping and agile methodologies was used. The members involved, or stakeholders, include the author as the sole developer, Dr. Kenny Hunt as the product owner, and the general population as the system end-users.

Incremental prototyping is a life cycle model that focuses on dividing a software project into manageable pieces, prioritizing functionality that is considered the most important for a project. The creation of these pieces is performed in a series of iterations that are driven by the desire to receive strong customer feedback at many stages throughout the development process. With each iteration, a new prototype is constructed and demoed as key features are implemented, tested, and merged. In addition, these manageable pieces can be designed at an earlier stage of the development process and updated with the completion of each prototype.

In this project, Dr. Hunt's advice of fully designing and implementing the front-end user interface (UI) as an initial prototype followed by the back-end API and database afterwards was used. This implementation strategy closely follows the incremental prototyping model. Once all of the functionality was described and arranged in the UI (the most important piece of the web application), the server-side web API and associated functionality was relatively simple to design and implement in later prototypes.

The agile life cycle model is similar to the incremental prototyping model in the sense that both models decompose the workload down into smaller iterations, and design and testing is performed throughout the entirety of the project. The biggest difference between the two is that with agile, functionality is implemented by a complete user story at a time. All necessary user stories are gathered at the beginning of the project and placed into a prioritized product backlog. Once the backlog is created, agile iterations called sprints can begin. Sprints are much more defined and organized compared to incremental prototyping iterations. They are encapsulated in a strict, repeatable time frame, usually one to four weeks. In general, a sprint starts with sprint planning to determine which user stories will be removed from the backlog and implemented to completion followed by the actual development of said user stories. A sprint review then completes each sprint and presents the completed work to the product owner, and a sprint retrospective is used to identify and discuss areas of weakness to improve development processes moving forward.

For the development of the Parlay web application, a modified agile approach was applied in parallel with the aforementioned incremental prototyping model. The project management consisted of a set of user stories and 40-hour time-boxed sprints comprised of the next most important pieces of functionality rather than fully defined user stories. In between each sprint, the author, who was also the sole developer for the project, met with Dr. Hunt to perform the sprint review and retrospective for the completed sprint and also the sprint planning for the next sprint. As the product owner, Dr. Hunt was presented with each sprint's completed work and was given the opportunity to provide constructive feedback to the author. The next sprint was planned by the author with the next most important tasks prior to each meeting and was also discussed with Dr. Hunt. Each task was given an estimation of the total amount of hours required to complete it along with links to which user stories the task was targeting. Any of the tasks not completed within the set sprint schedule were removed, re-evaluated, and usually placed at the beginning of the next sprint. All user stories and tasks were documented and managed via *monday.com*. This site provides its users with flexible options for planning, tracking, and managing work items. An example of a single sprint's documentation for Parlay is shown in Figure 1. The tasks in Sprint 5 took longer than expected to complete, resulting in a few tasks being removed and placed in Sprint 6.



Sprint 5 (3/17/2021-3/31/2021)

	Status	Estimation	Actual	User Stories
Create active league toolbar navigation	Complete	1 hours	1 hours	As a leagu... +1
Create active league dynamic UI containers	Complete	2 hours	4 hours	As a leagu... +2
Create week navigation list UI	Complete	2 hours	2 hours	As a leagu... +1
Implement Game object model and retrieval route	Complete	2 hours	1 hours	As a leagu... +1
Display all games html/css	Complete	5 hours	9 hours	As a leagu... +2
Implement placing a bet functionality UI	Complete	10 hours	15 hours	As a leagu... +1
Create place bet modal	Complete	2 hours	2 hours	As a league par...
Create odds service to handle odds/payout calculations	Complete	3 hours	3 hours	As a league par...
Implement Bet object and retrieval route	Complete	2 hours	2 hours	As a leagu... +4
+ Add				
		29 hours sum	39 hours sum	

Figure 1. Sprint 5 Documentation

This software development strategy used by the author was selected because of its organizational focus and overall flexibility. Early on in development the author was able to remove the uncertainty surrounding complex features of the application while still maintaining an organized and calculated foundation via sprint execution. As many of the requirements of this application were not trivial, the author felt most comfortable doing some initial planning and design up front followed by gradual prototyping of important functionality. In contrast, the waterfall model, for example, requires extensive and complete design documentation before implementation can begin. All of that planning up front assumes the knowledge of every detail of every project component which did not seem feasible for this project. Another appealing feature of the chosen life cycle models was the routine schedule provided by the sprint structure. Sprints come equipped with goals and expectations for development which produces constant motivation for a large project taking place over the course of many months.

In all, the development of Parlay consisted of 485 total hours contained within 11 sprints taking place from January 6, 2021 through September 1, 2021. Each sprint was targeted for 40 hours of estimated work. From January through mid-May, the author was able to contribute roughly 20 hours per week towards the project resulting in two weeks allocated per sprint for the first eight sprints. For the remainder of the project, the author's capacity decreased to roughly 10 hours per week resulting in the final three sprints having a duration of four weeks. Figure 2 displays the burndown chart for the project. The x-axis is broken

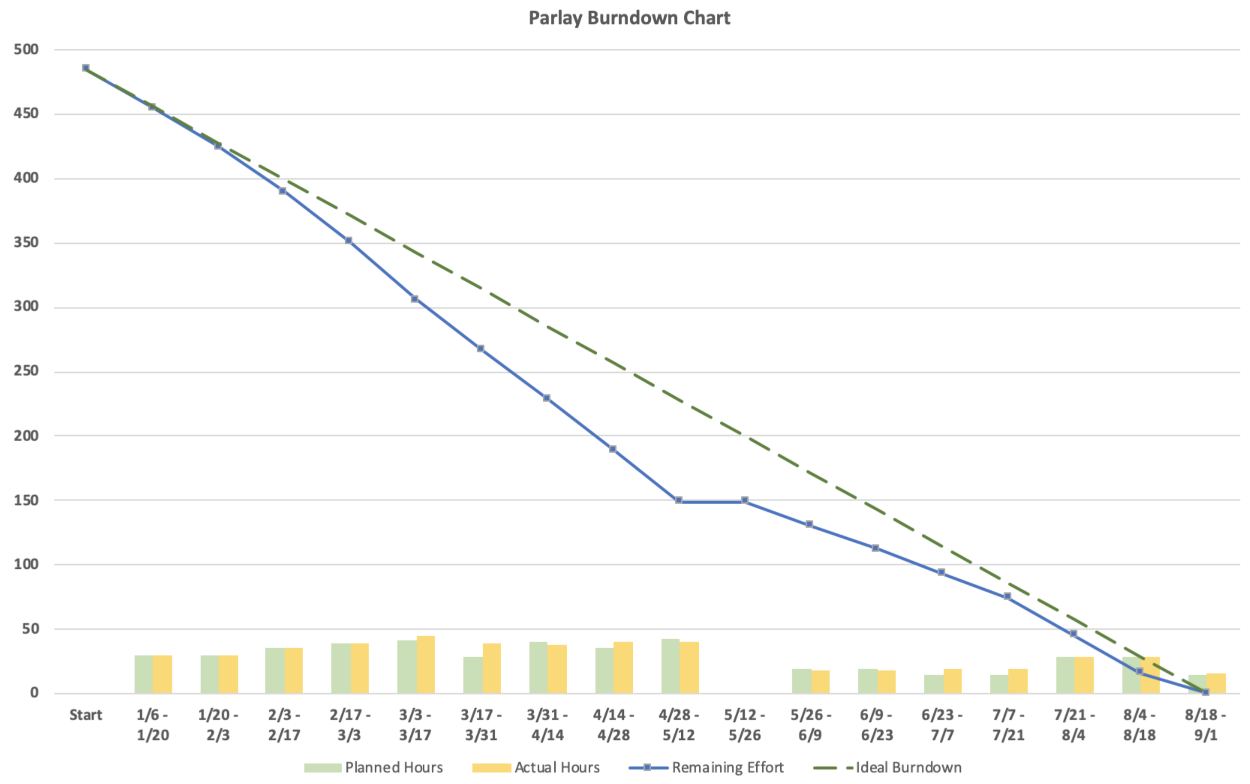


Figure 2. Burndown Chart

down into two week increments and the y-axis represents effort spent in hours. Each data point consists of the number of hours estimated for the two week period and the number of hours actually contributed. The actual number of hours is subtracted from the total remaining effort to create the data points for the blue *Remaining Effort* line. Lastly, the green dashed line represents the ideal burndown rate. An average of 28.5 hours was contributed every two weeks over the course of the entire project. One notable point in the burndown chart includes a 10 hour underestimation for the 3/17–3/31 data point. The development of the Create League UI page took much longer than anticipated due to some unforeseen complexity surrounding league start and end dates. Another notable time frame was 5/12–5/26. Following the spring semester, the author took a short break from development and resumed in early June.

2.3. Functional Requirements

As agile techniques were applied in this project, a complete list of user stories were identified before implementation of the product began. These user stories were the foundation for all development activities and served as the project's functional requirements.

Three user groups are involved in the Parlay web application. The first type of user is a universal user which includes any user that visits the site. The next type of user is a league participant. League participants are universal users that have joined together with other users to form a league. Finally, within each league, one of the league participants is designated as the league manager, the third user group. This hierarchy of users is shown in Figure 3.

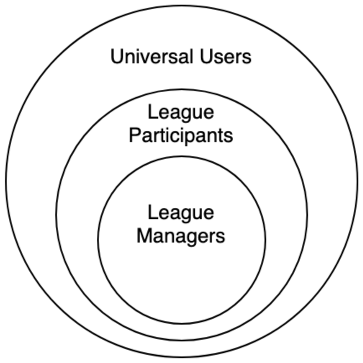


Figure 3. Hierarchy of User Functionality

The user stories for the universal user group describe account management functionality such as registering for an account, signing in and out of the application, resetting a forgotten password, and changing an existing password. The rest of the universal user actions include creating a new league, joining an existing league, viewing a list of all joined leagues, and accessing a user help screen.

League participants have the ability to view their league's settings and the names of the other participants in their league. They can also change their team name or leave a league before it starts. Once a league begins, participants can view any upcoming or past football games that were played after the league's start date. Each game consists of the teams competing, any available scores, and the associated bookmaker odds which map to specific outcomes of the game. League participants are able to use these odds and their available balance to place bets on any game before it starts by making predictions on the final score of the game. As bets are placed each week, users can track the status and results of their bets along with all bets placed by other users in the league both for current and past weeks. An active standings page is also maintained as participants' balances move up and down over the course of the league.

The league manager user group inherits all available functionality of the first two users groups and is also able to perform various league bookkeeping actions. Prior to a league becoming active, a league manager can change league settings, send out email invitations for the league, remove users from the league, change the league's public access code, reassign their league manager privileges, and delete the league all together. Once a league completes, rather than needing to create a new league, a league manager can simply choose to reactivate the completed league which allows the league to start over with all of the same users still in the league.

2.4. Non-Functional Requirements

Non-functional requirements describe attributes of a software system rather than specific functional use cases. These requirements are essential when evaluating the overall performance and usability of a product. The following non-functional requirements were considered for this project.

- The most recent live bookmaker odds must be displayed to all users at all times to prevent unfair advantages when placing bets.
- All user passwords must be hashed before being stored in the database.
- Due to a high reliance on scheduled server events, the system shall be able to identify and recover from a missed or failed scheduled event.
- Appropriate web development security characteristics such as secure user authentication, effective data cleaning and validation, and trusted, encrypted network communication must be applied.

3. Design

3.1. Architecture Overview

The Parlay application follows the typical web-based client, server, database architecture. The technologies used for development were Angular, a front-end JavaScript based framework, Node.js, a back-end JavaScript runtime environment, and MySQL, a relational database management system. The application is hosted by Amazon Web Services (AWS), and it is comprised of both an Elastic Compute Cloud (EC2) server instance and a Relational Database Service (RDS) MySQL instance. Figure 4 provides a high-level overview of the core application components and how they interact with each other. This chart is described in detail below.

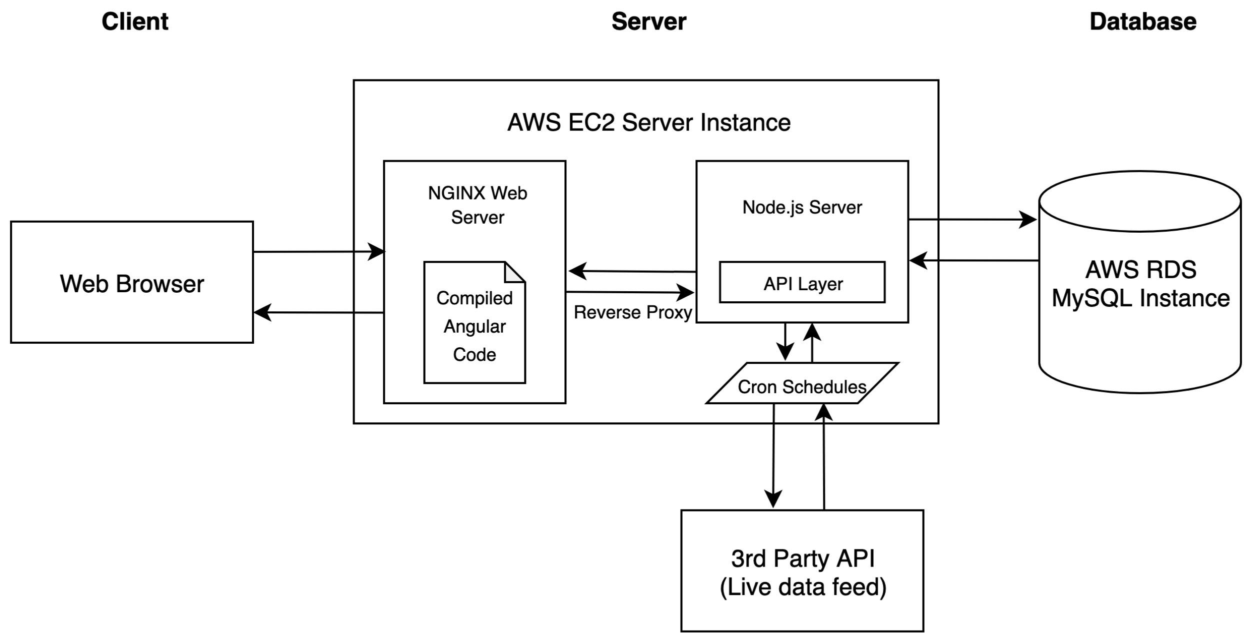


Figure 4. Architecture Overview

All interactions with Parlay begin with a client web browser navigating to the *parlayleagues.com* domain owned by the author. The browser contacts the AWS EC2 server running an NGINX web server listening for web traffic on ports 80 and 443. Any web traffic that targets port 80 is redirected to port 443 to ensure a secure and encrypted connection. The NGINX web server supplies the static, compiled Angular code to the browser to display the client interface. As the application is used, the client makes AJAX requests to the server which are passed along to the internal Node.js server's API layer via a reverse proxy. The API layer issues queries against the AWS RDS MySQL instance in order to handle the requests coming from the client.

Separate from client interactions with the API layer, the Node.js server has additional responsibilities. The Parlay web application is constructed around the availability and access of real-time sports data. In order to keep the interface up-to-date, Parlay needs to refresh its sports data periodically. This is automated through the use of cron schedules which have the ability to trigger events at set times throughout the day. Cron jobs are defined by cron rules and are asynchronously managed internally by the server's Operating System. Parlay subscribes to a 3$^{\text{rd}}$ party API called SportspageFeeds. While SportspageFeeds provides several tiers of access to their API, this project relied on the free tier which allows up to 20 API requests each day. Each API call retrieves the most recent bookmaker odds and other game data which is parsed and stored into the appropriate database tables for future re-use.

3.2. Client Design

This sub-section describes the user interface design for this project.

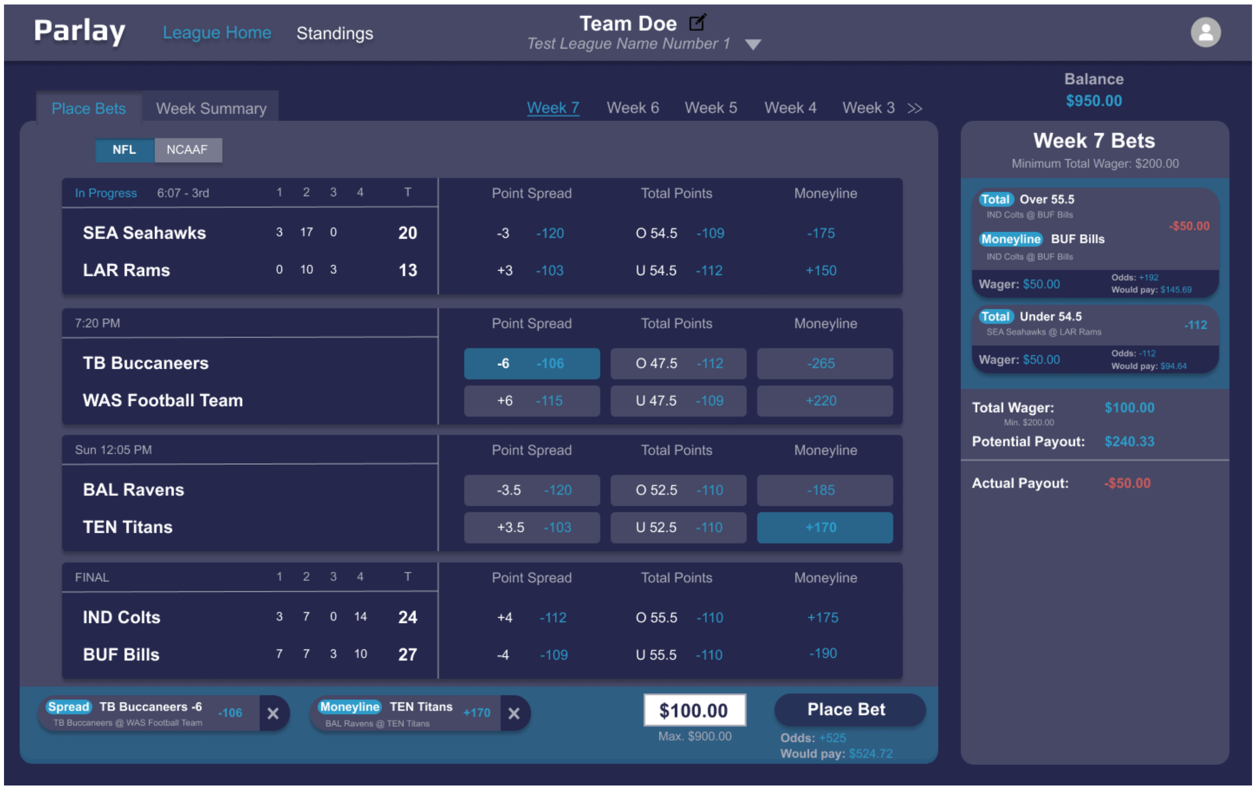
3.2.1. User Interface Design

Before any design documents or feature implementation began, the entire Parlay web application interface was created in a user interface design tool called Figma. Figma allows users to efficiently test out various design ideas through its drag and drop mechanisms. The end result for this project was a static version of every web-page necessary for the fully functional end product. When it came time to implement the user interface pages using the Angular front-end framework, the Figma UI mock-ups were extremely useful and expedited the development effort greatly. Figures 5 and 6 compare the most complex Parlay screen with Figure 5 showing the initial Figma static version and Figure 6 showing the actual version deployed in the completed application.

It is evident that the two screen captures are very similar. In the center *Place Bets* container, the games and bottom pending bet section are nearly identical. One addition in the actual version is the use of team logos. The logos assist users in identifying the teams competing in each game and add an additional sense of separation between each game in the list. The actual version also adds a search input which allows users to easily find a specific team or game. Above the *Place Bets* container, the actual version condensed the week navigation feature and added a *Weeks Remaining* visual cue to inform users how many weeks are left in the league. The top toolbar remained the same in both versions. Moving to the right side of the screen, a *Must Risk* value was added and the bet sidebar's layout was adjusted slightly. Overall, the static mock-up was followed very closely during implementation.

Once finalized, the static Figma mock-ups were annotated with low-level, detailed behavioral descriptions. The purpose of this document was to describe each element on every page and consider all possible user interactions with an element that could result in state changes or potential error cases. This allowed the author to dive deeper into the user interface design and think about the true purpose and need for each UI element in the application. An example of how this process was documented is shown in Figure 7 and Table 1.

Once the user interface design and documentation was complete, the creation of the project's user stories and database design was relatively straightforward. The static UI screens made



In Progress 6:07 - 3rd	1	2	3	4	T	Point Spread	Total Points	Moneyline
SEA Seahawks	3	17	0		20	-3 -120	O 54.5 -109	-175
LAR Rams	0	10	3		13	+3 -103	U 54.5 -112	+150

7:20 PM	Point Spread	Total Points	Moneyline
TB Buccaneers	-6 -106	O 47.5 -112	-265
WAS Football Team	+6 -115	U 47.5 -109	+220

Sun 12:05 PM	Point Spread	Total Points	Moneyline
BAL Ravens	-3.5 -120	O 52.5 -110	-185
TEN Titans	+3.5 -103	U 52.5 -110	+170

FINAL	1	2	3	4	T	Point Spread	Total Points	Moneyline
IND Colts	3	7	0	14	24	+4 -112	O 55.5 -110	+175
BUF Bills	7	7	3	10	27	-4 -109	U 55.5 -110	-190

Spread TB Buccaneers -6 -106
TB Buccaneers @ WAS Football Team

Moneyline TEN Titans +170
BAL Ravens @ TEN Titans

$100.00
Max. $900.00

Odds: +525
Would pay: $524.72

Week 7 Bets
Minimum Total Wager: $200.00

Total Over 55.5
IND Colts @ BUF Bills
Moneyline BUF Bills
IND Colts @ BUF Bills
-$50.00
Wager: $50.00 Odds: +192 Would pay: $145.69

Total Under 54.5
SEA Seahawks @ LAR Rams
-112
Wager: $50.00 Odds: -112 Would pay: $94.64

Total Wager: $100.00
Min. $200.00
Potential Payout: $240.33

Actual Payout: -$50.00

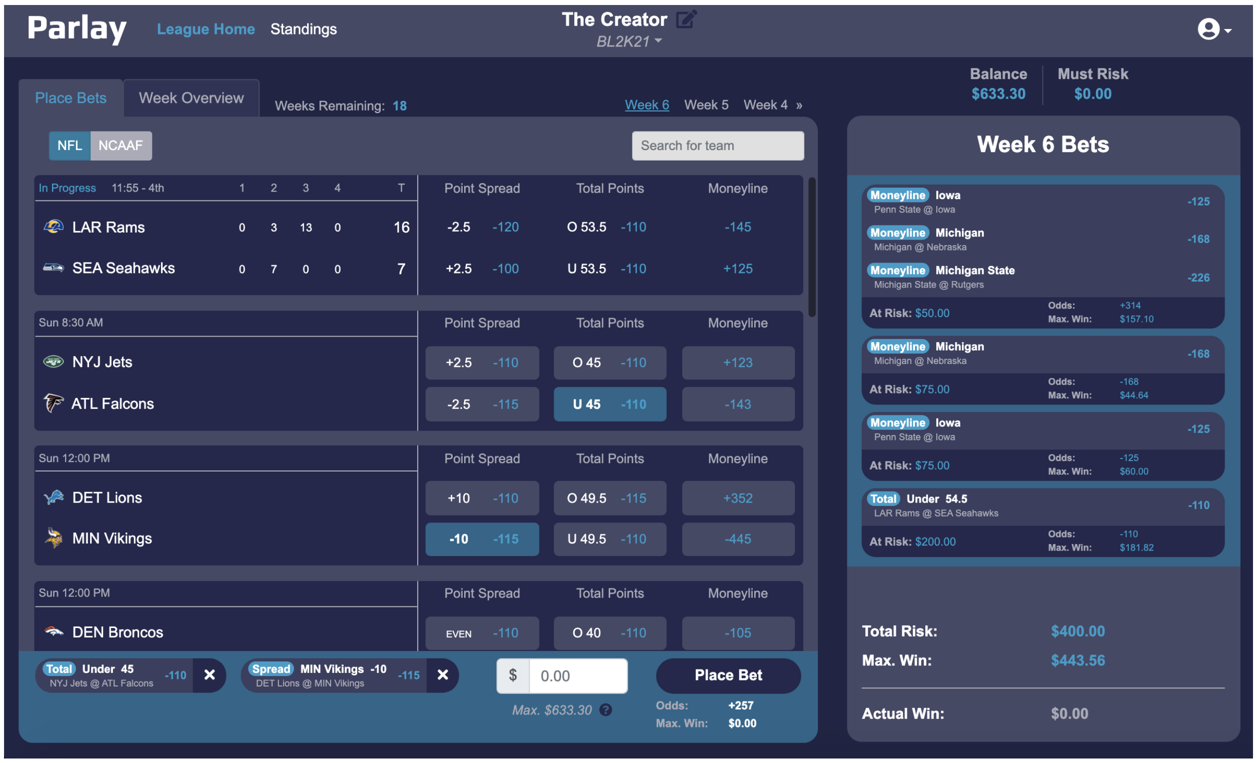
Figure 5. Figma Mock-up of League Home Page

it easy to visualize all of the functionality available to a user and, similarly, all of the tables and attributes necessary for the database.

3.2.2. Web-page Design

The client web-pages were all developed using the JavaScript-based Angular framework and Bootstrap 4. Angular allows for web applications to be decomposed into reusable components in a modular fashion. Each component consists of an HTML file, Typescript controller file, and a CSS style sheet. The HTML file structures all of the elements for the component and binds itself to variables defined in the controller and event listeners. The controller file processes the data needed for the component. These files have the ability to send and receive data and interact with other components. A common scenario within a component begins with a button element placed in the HTML file that listens for click events. When the button is clicked, it invokes a handler located inside of the Typescript controller file. These handler methods are responsible for initiating any dynamic action that takes place on a web-page. Two examples of the dynamic action that these handlers invoke include navigating a user to a different screen or firing an API request to GET or POST data.

While Angular's CSS style sheets allow rules to be applied at the component scope level, the author tried to avoid the use of manual CSS rules as much as possible. Bootstrap is an open-source CSS framework that contains many design templates that allow for consistent styling across an application. One such design pattern provided by Bootstrap that was used



In Progress 11:55 - 4th	1	2	3	4	T	Point Spread		Total Points		Moneyline
LAR Rams	0	3	13	0	16	-2.5	-120	O 53.5	-110	-145
SEA Seahawks	0	7	0	0	7	+2.5	-100	U 53.5	-110	+125

Sun 8:30 AM	Point Spread		Total Points		Moneyline
NYJ Jets	+2.5	-110	O 45	-110	+123
ATL Falcons	-2.5	-115	U 45	-110	-143

Sun 12:00 PM	Point Spread		Total Points		Moneyline
DET Lions	+10	-110	O 49.5	-115	+352
MIN Vikings	-10	-115	U 49.5	-110	-445

Sun 12:00 PM	Point Spread		Total Points		Moneyline
DEN Broncos	EVEN	-110	O 40	-110	-105

Total Under 45 -110
NYJ Jets @ ATL Falcons

Spread MIN Vikings -10 -115
DET Lions @ MIN Vikings

$ 0.00
Max. $633.30

Odds: +257
Max. Win: $0.00

Balance $633.30
Must Risk $0.00

Week 6 Bets

Moneyline Iowa -125
Penn State @ Iowa
Moneyline Michigan -168
Michigan @ Nebraska
Moneyline Michigan State -226
Michigan State @ Rutgers
At Risk: $50.00 Odds: +314 Max. Win: $157.10

Moneyline Michigan -168
Michigan @ Nebraska
At Risk: $75.00 Odds: -168 Max. Win: $44.64

Moneyline Iowa -125
Penn State @ Iowa
At Risk: $75.00 Odds: -125 Max. Win: $60.00

Total Under 54.5 -110
LAR Rams @ SEA Seahawks
At Risk: $200.00 Odds: -110 Max. Win: $181.82

Total Risk: $400.00
Max. Win: $443.56

Actual Win: $0.00

Figure 6. Actual League Home Page

heavily in the Parlay web application was screen size responsive styling. As a screen size grows or shrinks, elements on the page dynamically resize or become visible or hidden. This allows for a web application to deliver the best possible user experience regardless of the screen size visiting the site. Figure 8 shows the same league home page in Figure 6 above at a medium and mobile screen size. Although the mobile screen layout is missing some key visual information, users are still able to place bets and navigate the site while on-the-go.

3.3. Database Design

The first task when considering a database for the Parlay web application was selecting which database management system to use. It was determined from initial project planning that a relational database management system (RDBMS) would be the best fit. The two primary factors that went into this decision were that the data requirements for the project could be structured nicely in a relational scheme with primary and foreign keys and that complex SQL queries could be written easily for any data retrieval need.[6] From there, the author did not have a strong connection with any of the major RDBMS providers, so after a little research, MySQL was chosen because of its industry popularity and cheap access within the AWS RDS console.

Using the completed UI mock-ups and user stories, a high-level entity relationship (ER) diagram was created. The diagram is shown in Figure 9 and is described below.

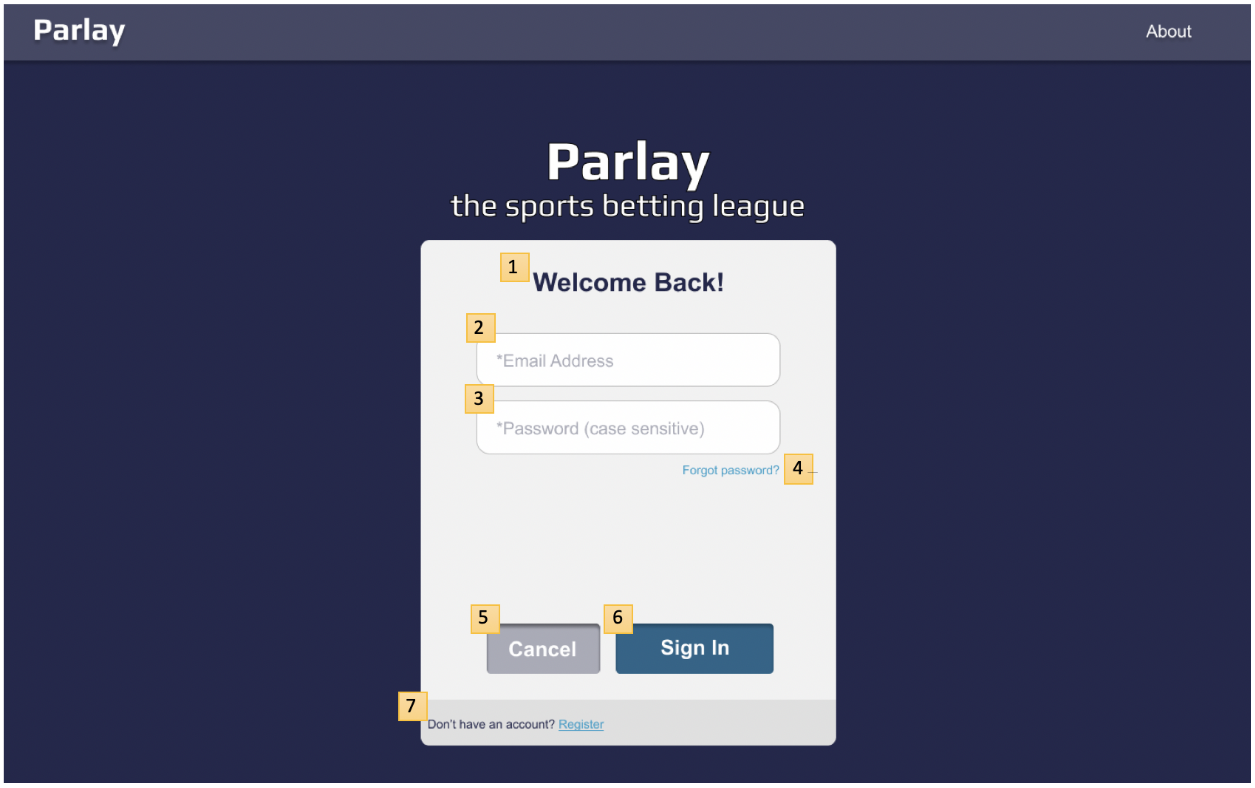
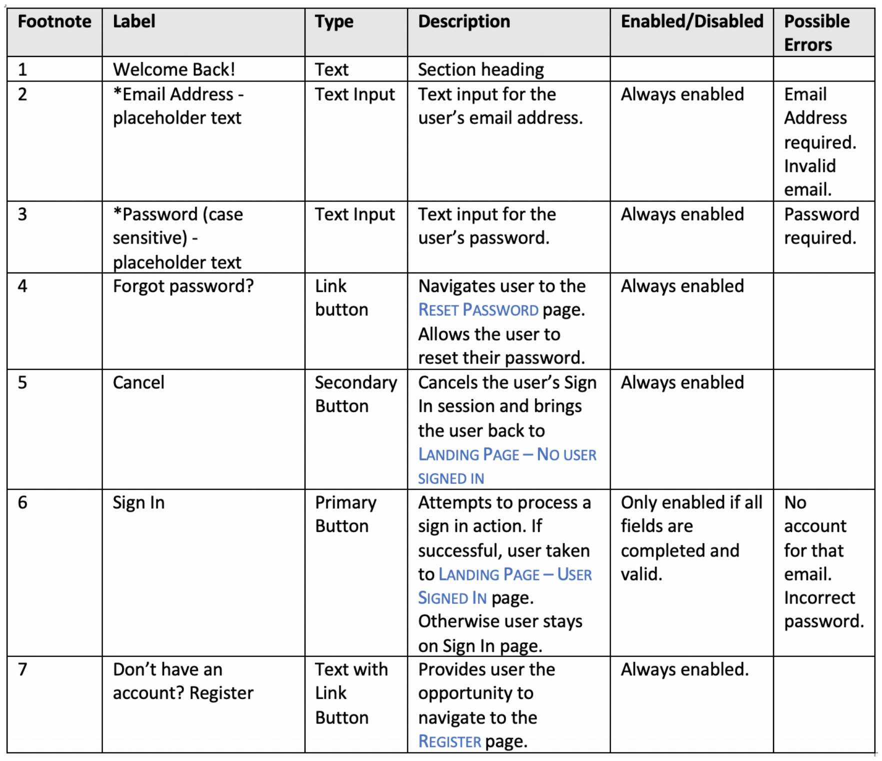
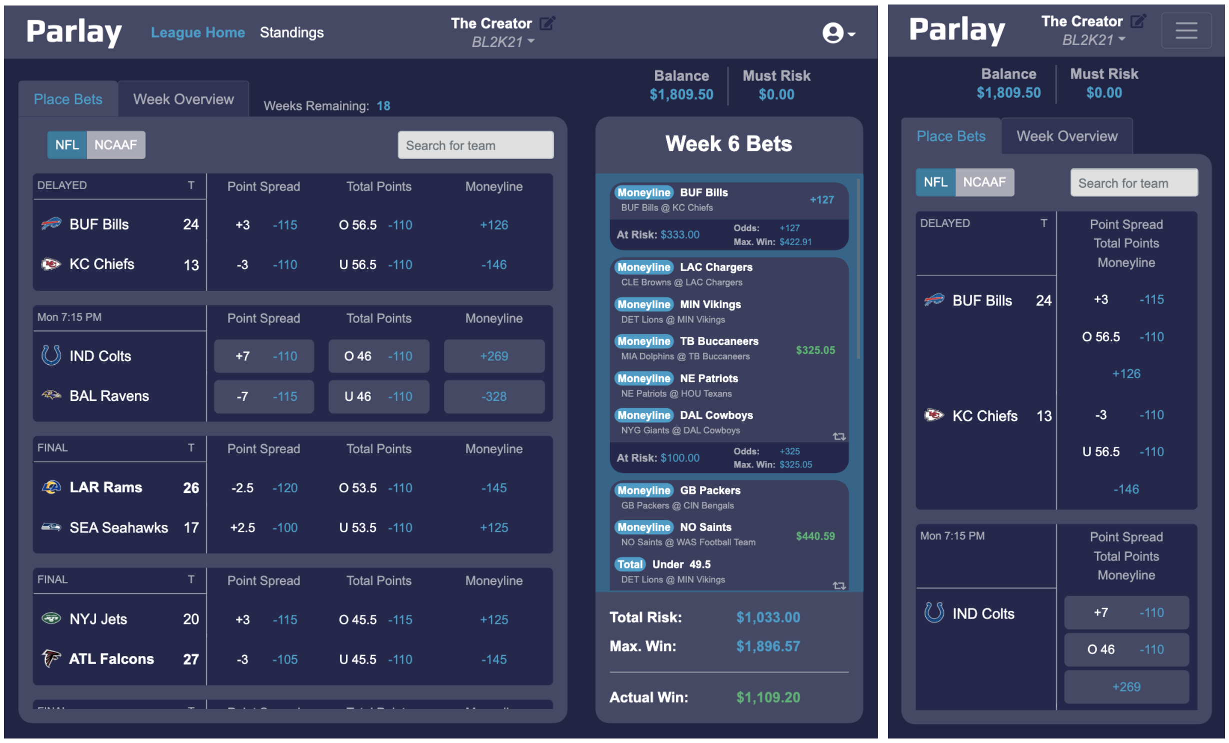

Parlay
About
Parlay
the sports betting league
1
Welcome Back!
2
*Email Address
3
*Password (case sensitive)
Forgot password?
4
5
6
Cancel
Sign In
7
Don't have an account? Register

Figure 7. Static Login Screen Used in UI Design Document



Footnote	Label	Type	Description	Enabled/Disabled	Possible Errors
1	Welcome Back!	Text	Section heading		
2	*Email Address - placeholder text	Text Input	Text input for the user's email address.	Always enabled	Email Address required. Invalid email.
3	*Password (case sensitive) - placeholder text	Text Input	Text input for the user's password.	Always enabled	Password required.
4	Forgot password?	Link button	Navigates user to the Reset Password page. Allows the user to reset their password.	Always enabled	
5	Cancel	Secondary Button	Cancels the user's Sign In session and brings the user back to Landing Page – No User Signed In	Always enabled	
6	Sign In	Primary Button	Attempts to process a sign in action. If successful, user taken to Landing Page – User Signed In page. Otherwise user stays on Sign In page.	Only enabled if all fields are completed and valid.	No account for that email. Incorrect password.
7	Don't have an account? Register	Text with Link Button	Provides user the opportunity to navigate to the Register page.	Always enabled.	

Table 1. Description of Login Screen UI Elements



Weeks Remaining: 18

Balance	Must Risk
$1,809.50	$0.00

DELAYED	T	Point Spread		Total Points		Moneyline
BUF Bills	24	+3	-115	O 56.5	-110	+126
KC Chiefs	13	-3	-110	U 56.5	-110	-146

Mon 7:15 PM		Point Spread		Total Points		Moneyline
IND Colts		+7	-110	O 46	-110	+269
BAL Ravens		-7	-115	U 46	-110	-328

FINAL	T	Point Spread		Total Points		Moneyline
LAR Rams	26	-2.5	-120	O 53.5	-110	-145
SEA Seahawks	17	+2.5	-100	U 53.5	-110	+125

FINAL	T	Point Spread		Total Points		Moneyline
NYJ Jets	20	+3	-115	O 45.5	-115	+125
ATL Falcons	27	-3	-105	U 45.5	-110	-145

Week 6 Bets

Moneyline BUF Bills — BUF Bills @ KC Chiefs — +127
At Risk: $333.00 — Odds: +127 — Max. Win: $422.91

Moneyline LAC Chargers — CLE Browns @ LAC Chargers
Moneyline MIN Vikings — DET Lions @ MIN Vikings
Moneyline TB Buccaneers — MIA Dolphins @ TB Buccaneers
Moneyline NE Patriots — NE Patriots @ HOU Texans
Moneyline DAL Cowboys — NYG Giants @ DAL Cowboys
$325.05
At Risk: $100.00 — Odds: +325 — Max. Win: $325.05

Moneyline GB Packers — GB Packers @ CIN Bengals
Moneyline NO Saints — NO Saints @ WAS Football Team
Total Under 49.5 — DET Lions @ MIN Vikings
$440.59

Total Risk: $1,033.00
Max. Win: $1,896.57
Actual Win: $1,109.20

DELAYED	T	Point Spread / Total Points / Moneyline
BUF Bills	24	+3 -115
		O 56.5 -110
		+126
KC Chiefs	13	-3 -110
		U 56.5 -110
		-146

Mon 7:15 PM		Point Spread / Total Points / Moneyline
IND Colts		+7 -110
		O 46 -110
		+269

Figure 8. Medium and Mobile Sized League Home Page

A single user can become a participant in many leagues. Many participants make up a league; one of which is also the league manager. Before a league starts, invitations can be sent via email that allow anyone to create a Parlay account and join that league. Once a league has started, each participant has the opportunity to place bets. A single bet is made up of many picks that each reference a single game. Every game is played at a single venue, contains many periods, is played by two teams, and is associated with a single sports league such as the NFL. Every team is similarly organized in a sports league and can also be ranked in a weekly poll. A set of polls are all associated with a single season and each season is associated with a sports league. Lastly, all leagues are able to select a set of sports leagues that it has available to place bets on.

Next, the ER-Diagram was broken down into the database tables and attributes needed to fulfill the application's user story requirements. The MySQL create table statements for two of these tables is shown in Figure 10.

Each entry within the Participant table is automatically assigned a unique *participantId* which serves as the primary key. There are two foreign keys within the table that reference the user account that the participant is associated with and the league that the participant is in. Each participant has a customizable *teamName* along with a *balance*, *prevBalance*, and *prevRank*. The *balance* attribute tracks the active amount of currency remaining for a participant. The *prevBalance* attribute holds the amount of currency a participant had at

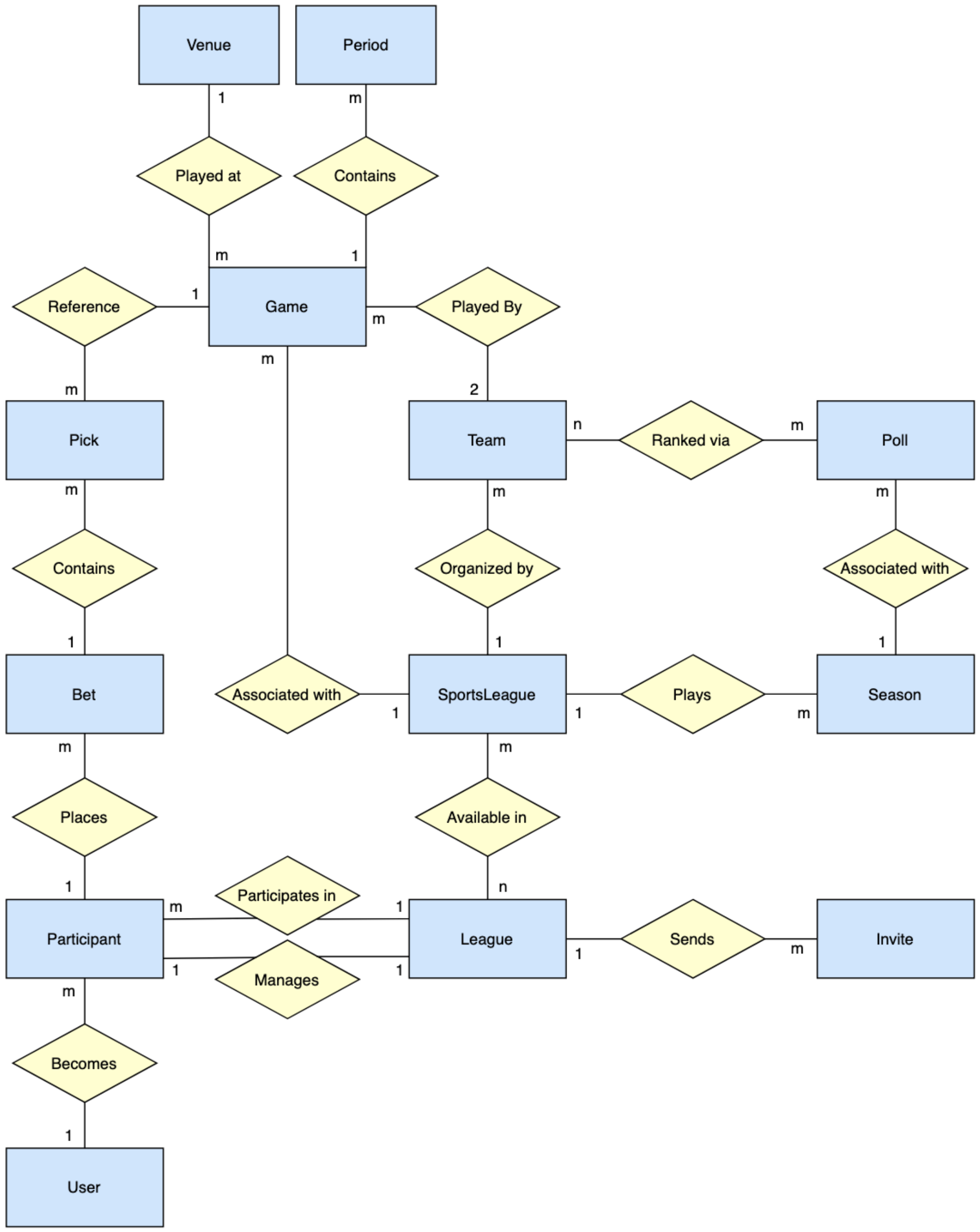
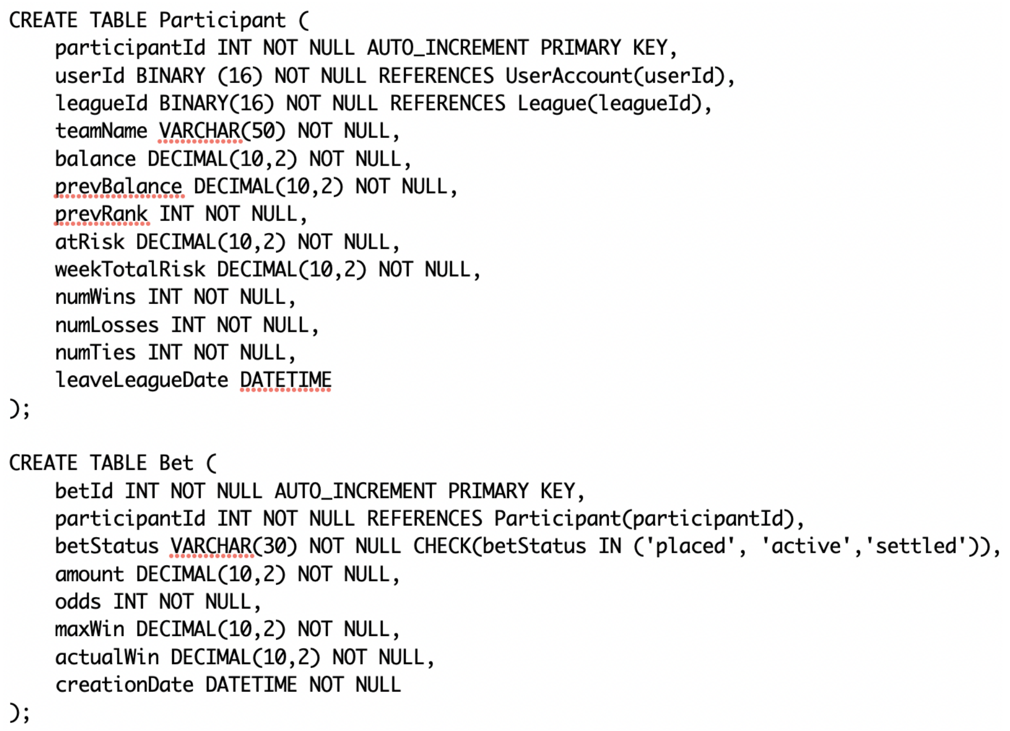


Figure 9. ER-Diagram for Parlay



```
CREATE TABLE Participant (
    participantId INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    userId BINARY (16) NOT NULL REFERENCES UserAccount(userId),
    leagueId BINARY(16) NOT NULL REFERENCES League(leagueId),
    teamName VARCHAR(50) NOT NULL,
    balance DECIMAL(10,2) NOT NULL,
    prevBalance DECIMAL(10,2) NOT NULL,
    prevRank INT NOT NULL,
    atRisk DECIMAL(10,2) NOT NULL,
    weekTotalRisk DECIMAL(10,2) NOT NULL,
    numWins INT NOT NULL,
    numLosses INT NOT NULL,
    numTies INT NOT NULL,
    leaveLeagueDate DATETIME
);

CREATE TABLE Bet (
    betId INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    participantId INT NOT NULL REFERENCES Participant(participantId),
    betStatus VARCHAR(30) NOT NULL CHECK(betStatus IN ('placed', 'active','settled')),
    amount DECIMAL(10,2) NOT NULL,
    odds INT NOT NULL,
    maxWin DECIMAL(10,2) NOT NULL,
    actualWin DECIMAL(10,2) NOT NULL,
    creationDate DATETIME NOT NULL
);
```

Figure 10. Participant and Bet Database Table Design

the end of the previous week. This value is needed in order to determine how much currency a participant is required to risk in the current week. The *prevRank* column is used to show a trend value in the standings page that tracks if a participant has moved up or down in the overall league standings. The *atRisk* field holds the total amount of currency that is currently placed on bets that have not yet completed, and similarly, the *weekTotalRisk* attribute holds the total amount of currency that has been risked on bets throughout the current week. If this value is less than the amount that the participant must risk at the end of the week, the difference is deducted from the participant's balance. The number of wins, losses, and ties is also tracked for each league participant. One of these three values is updated every time a participant's bet is evaluated. Finally, the *leaveLeagueDate* column of type date-time is a UTC formatted date string used to track if and when a participant has left the league. The field is set to *null* if the participant is still a member of the league. All date-time values stored in the database remain consistent as UTC formatted strings.

For the Bet table, it is similar to the Participant table in that it has an auto-incremented integer primary key. There is also one foreign key included in the table, *participantId*, that maps to the participant who placed the bet. A bet has one of three possible statuses, 'placed', 'active', or 'settled'. When a bet is created, it is assigned a *betStatus* of 'placed'. Once one or more of the games involved in the bet have started, the bet gets moved to the 'active' *betStatus*, and it is moved to 'settled' once it is determinate. The *amount* of a bet is how much currency the participant decided to risk on the bet and the *odds* attribute holds the total bet odds, or the likelihood that the bet will succeed. The *amount* and *odds* fields are what determines the participant's *maxWin* value. The *actualWin* column starts at \$0.00

when the bet is placed and will either be changed to negative one times the *amount* of the bet if the bet were to fail or the *maxWin* value if the bet were to succeed. Lastly, the *creationDate* attribute tracks when each bet was placed by a league participant and entered into the database.

For more sensitive database tables such as the User table and the League table, the primary key attributes for these tables were created as type BINARY(16) to hold a universally unique identifier (UUID). These columns were populated using the MySQL UUID() method which generates a unique 128-bit number to represent the UUID. The UUID_TO_BIN() and BIN_TO_UUID() methods were also used when retrieving and storing these id values as strings and binary values. UUIDs were used for the identifiers that were commonly used in the application's URL query parameters to help enhance security. Unlike a typical auto-incremented integer field, it is impossible to predict the other entries surrounding a UUID. If auto-incremented identifiers were used in the URL, malicious users would be able to easily guess surrounding identifiers which brings them one step closer to accessing data that they should not be able to access.

3.4. Server Design

The Node.js server for the Parlay web application was the last portion of the project that was designed and implemented. The Parlay server is broken down into two main components: the API data layer and the cron scheduling automation layer.

3.4.1. API Data Layer

As previously mentioned, the entire user interface designed at the beginning of the project included all of the scenarios where calls to the API layer were needed. So, when it came to designing these API routes, the first step was organizing and naming the routes in a RESTful manner. REST, which stands for representational state transfer, defines an architectural style with constraints and guidelines to follow when creating an HTTP interface for communication between two parties. Therefore, a RESTful API applies the REST standards to client server interactions within a web application. The main reason why a RESTful API design pattern was chosen and followed was because of the strict separation between client and server it provides. One way Parlay could expand in the future is with the development of a mobile application. By creating a flexible and independent RESTful API layer for the current project, a future mobile client will require minimal effort when connecting to and utilizing the same API endpoints.

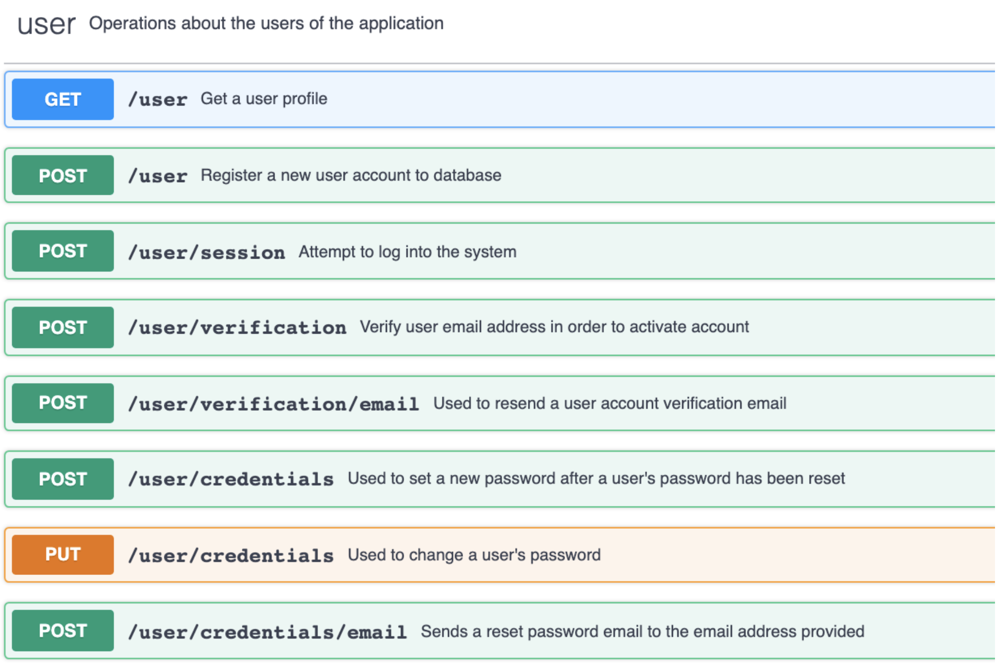
At a high-level, an API endpoint is made of an HTTP verb and resource noun that the operation is to be performed on. The four main HTTP verbs are GET, POST, PUT, and, DELETE. Routes using the GET verb retrieve resources for the client to read. The POST route is used to create new resources, PUT is used to edit or update existing resources, and DELETE is used to remove resources.

All data that is passed between the client and server was transferred as JSON. Each API route defines the request body and parameters it expects from the client and also details the structure of the response it will return. All inputs to the API layer are strictly checked to ensure that all expected parameters exist and are of the proper data type. If any of these validations fail, a 400 client error response is returned. Some routes are also protected with the 'Authorization' header. Upon logging into the application, a client receives a temporary JSON web token from the server. Any request, besides logging out, made by the client while logged in requires that the unique token be provided to avoid receiving a 401 unauthorized error response. This requirement allows the server to verify that the caller has been previously authenticated by the system. Also, by using an encrypted token that is unique for each user, the system mitigates Cross Site Request Forgery (CSRF) vulnerabilities where a malicious user attempts to impersonate another user when making an API call. Even if a user's id was exposed, actions still could not take place on his or her behalf because of the JSON web token authorization check.

The Parlay API specification defines four main resource nouns that all routes are grouped together by: user, league, game, and bet. A detailed API specification was created using *swagger.io* and is discussed below.

There are eight API endpoints defined under the user resource, and they are shown in Figure 11 below. The routes cover functionality for retrieving a user's profile and registering a new user account. Once registered, users may try to create a new session, verify their email for account activation, and resend a verification email if they did not receive the first verification email. Actions can also be taken in order to manage a user's credentials. If users forget their password, they have the ability to reset their login credentials for their account by following a reset password link sent to their email address. Users credentials can also be changed rather than reset using the PUT */user/credentials* endpoint. For each of these user API routes, the user account to be acted upon is identified via the 'Authorization' header token. This token has the unique user identifier encrypted within it and can be securely parsed and verified by the Node.js JSON web token passport module at any time.
League resources are the most dynamic resource used in Parlay and can be manipulated by clients in many ways. These functions are displayed in Figure 12. There are two API endpoints at the root level of the league resource. All leagues can be retrieved for the calling user and leagues can also be created. When creating a league, a start date and end date must be specified. There are limited options when choosing these values and the options are retrieved via two separate routes. Next, all four primary HTTP actions can be taken on a specific league. A league can be retrieved, reactivated (which creates a copy of the league), edited, and deleted. Separate modification routes are also included for generating a new random league access code and changing a league's manager. The last four API endpoints available for a league relate to the members of a league. All members of a league can be retrieved, new members can join a league before it starts, members can change their team name, and members can be removed from a league before it starts by a league manager.

There is only one API endpoint used when working with game resources. Shown in Figure 13, this route gets a list of all games and filters that list based on the sports leagues available for the provided *leagueId* and *weekNum* query parameters. The result is a set of


user Operations about the users of the application

GET `/user` Get a user profile

POST `/user` Register a new user account to database

POST `/user/session` Attempt to log into the system

POST `/user/verification` Verify user email address in order to activate account

POST `/user/verification/email` Used to resend a user account verification email

POST `/user/credentials` Used to set a new password after a user's password has been reset

PUT `/user/credentials` Used to change a user's password

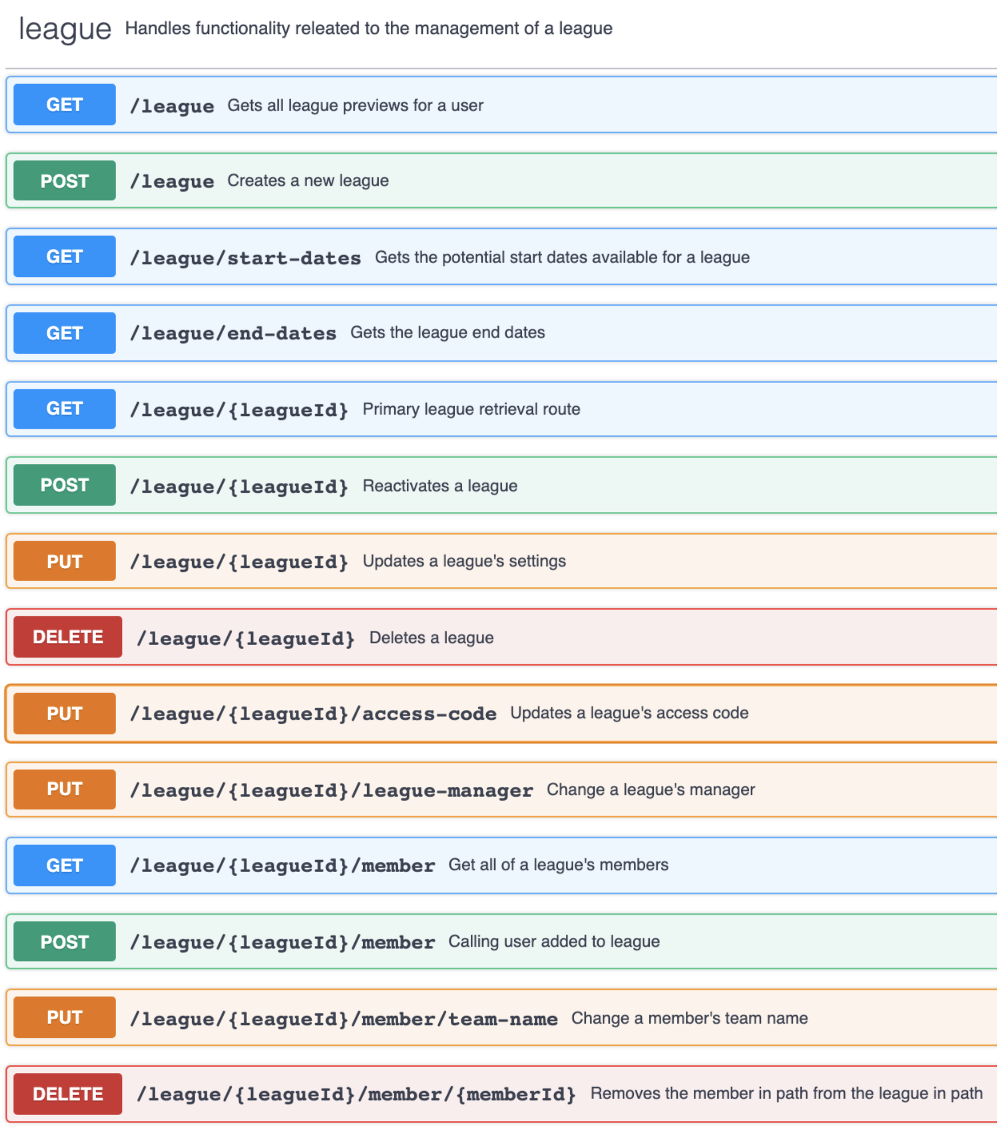
POST `/user/credentials/email` Sends a reset password email to the email address provided

Figure 11. User API Endpoints

games to be displayed for a single week of a league.

The final major resource that makes up the Parlay web application is the bet resource (Figure 14). The primary GET route for the bet resource is structured much like the league retrieval route. A list of all bets is retrieved and filtered on *leagueId* and *weekNum* query parameters passed in from the client. Similarly, a *leagueId* and *weekNum* can also be passed to the bet retrieval route for a single user. This route fetches all bets placed by the calling league participant in a single week. Lastly, bets are created using the POST */bet* endpoint.

Each of these endpoints defined in the API specification also outline precise request and response bodies and the property data types within them. An expanded view of the create bet route is shown in Figure 15. There are no query parameters necessary for this route, but the JSON structured request body is required. The request body holds a *leagueId* of type UUID string that the caller is placing the bet for and the *amount* that the participant is risking as a number. The *expectedOdds* and *expectedMaxWin* number properties are sent to the server to ensure that the client and server are in sync with what those values are expected to be. This protects users from a bet being placed for them that did not match up with what they were being shown in the user interface. For example, if the user interface was showing that a user would win \$100, but when the server made the calculations, it only calculated a win value of \$90, then the bet would not be placed and an error would be shown to the user. The final property that is required for placing a bet is the *picks* array. Each pick contains a *gameId* string and information to specify which exact game prediction that the pick corresponds to. There is no data sent back to the client in the response body for this route. For any HTTP GET route, a response body will be outlined in a similar way to


league Handles functionality releated to the management of a league

GET /league Gets all league previews for a user

POST /league Creates a new league

GET /league/start-dates Gets the potential start dates available for a league

GET /league/end-dates Gets the league end dates

GET /league/{leagueId} Primary league retrieval route

POST /league/{leagueId} Reactivates a league

PUT /league/{leagueId} Updates a league's settings

DELETE /league/{leagueId} Deletes a league

PUT /league/{leagueId}/access-code Updates a league's access code

PUT /league/{leagueId}/league-manager Change a league's manager

GET /league/{leagueId}/member Get all of a league's members

POST /league/{leagueId}/member Calling user added to league

PUT /league/{leagueId}/member/team-name Change a member's team name

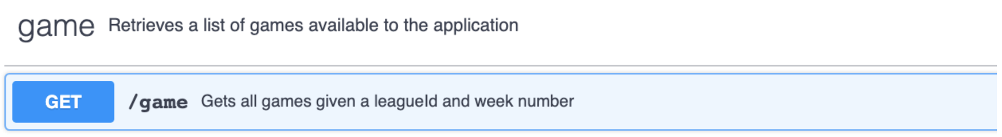
DELETE /league/{leagueId}/member/{memberId} Removes the member in path from the league in path

Figure 12. League API Endpoints

describe the structure of the retrieved data.

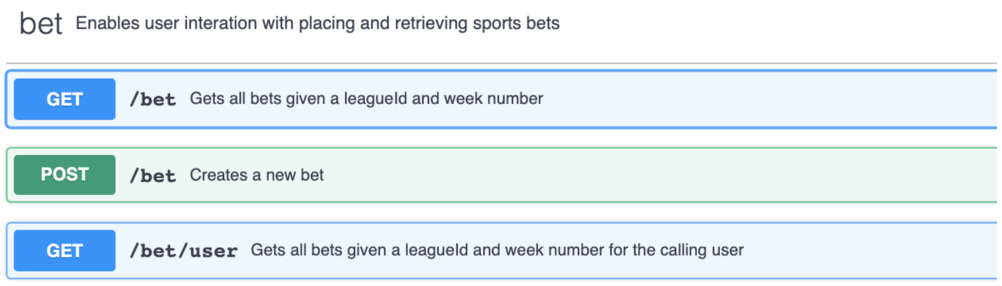
3.4.2. Cron Scheduling Automation Layer

The Parlay web application relies heavily on the use of scheduled server events. These scheduled events are configured through the use of NPM package *node-cron*. Events afre scheduled by defining a set of cron rules. A rule is made up of special characters and numbers to specify the minutes, hours, days of month, months, and days of week that the event should trigger. There are many rules defined for refreshing the game data at fixed times throughout each day and week. As previously mentioned, 20 requests per day can be made to the 3rd party SportspageFeeds API for free. For each weekday, the requests execute periodically at the same times each day. However, on Saturday and Sunday, more requests are made to refresh the NCAAF and NFL games, respectively. In a typical week, the large majority of


game Retrieves a list of games available to the application

GET /game Gets all games given a leagueId and week number

Figure 13. Game API Endpoint


bet Enables user interation with placing and retrieving sports bets

GET /bet Gets all bets given a leagueId and week number

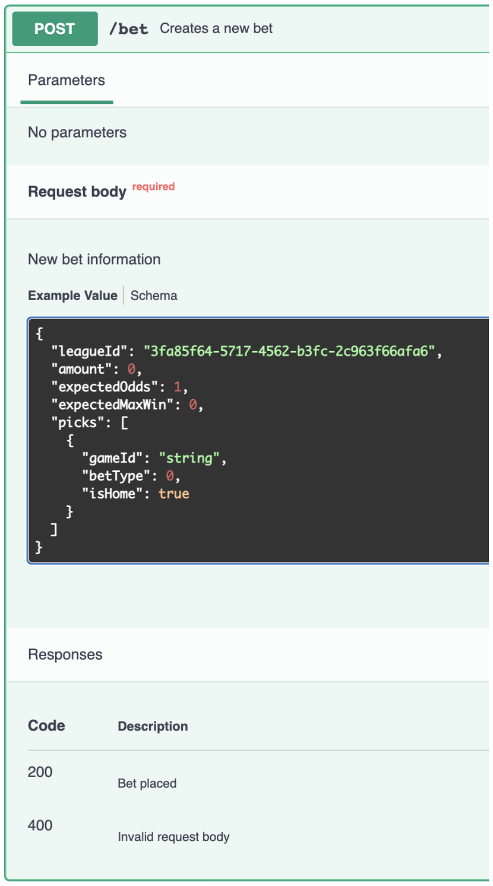
POST /bet Creates a new bet

GET /bet/user Gets all bets given a leagueId and week number for the calling user

Figure 14. Bet API Endpoints

NCAAF games take place on Saturday and the large majority of NFL games take place on Sunday. So, on Sundays, in order to provide the best user experience, 16 requests are made to refresh the NFL games and only four requests are made to refresh the NCAAF games. The requests are also placed at strategic times during the day. Many NFL games start at 12:00 p.m. each Sunday, so requests are made to refresh the bookmaker odds leading up to these games at 11:15 a.m., 11:30 a.m., and 11:45 a.m. This helps to prevent game odds from becoming inaccurate due to unexpected events taking place before kickoff. Two such events could include a key player being ruled out of a game, which hurts his team's chances to win or an inclement weather forecast, which makes it more likely that the total points scored in the game will be lower. Refreshing the game data at times where many games have likely completed is important too. This is because bets placed by users cannot be evaluated until the system is aware of the final scores of the games involved in said bets. Users want to see their bets processed in a timely manner, so they can see how they are stacking up amongst their competitors. Reusing the example of many games starting at 12:00 p.m. on Sundays, as games usually last roughly three hours from start to finish, a game refresh event is scheduled for 3:30 p.m. This event is likely to capture many games that have completed recently which allows for bets placed on these games to be updated relatively quickly.

Besides refreshing the bookmaker odds and game data, daily cron schedules are also used to manage the state of all leagues and perform other bookkeeping operations. At 5:00 a.m. each morning, the server executes a method to check for any new leagues that need to be changed to an 'Active' league status. Every Tuesday at 5:00 a.m., all 'Active' leagues are advanced to the next week. This process includes deducting any required currency that was not risked from user balances, checking if the league should be marked as 'Complete', and updating additional participant data fields in preparation for the new week. Upon completion of this update each day, a log entry is written to the *LeagueUpdateLog* database table. Three calls to this same operation follow the initial league update process to ensure that a log entry was written to the database. If any subsequent call finds that the league update operation was

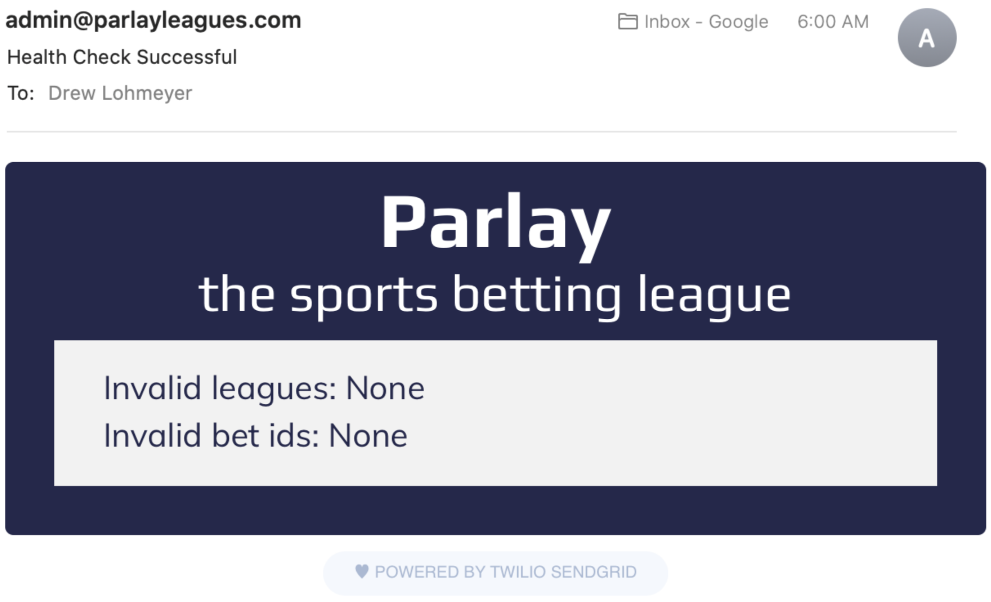


```
{
  "leagueId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "amount": 0,
  "expectedOdds": 1,
  "expectedMaxWin": 0,
  "picks": [
    {
      "gameId": "string",
      "betType": 0,
      "isHome": true
    }
  ]
}
```

Code	Description
200	Bet placed
400	Invalid request body

Figure 15. Create Bet API Endpoint

missed, it will run the operation and write to the log table. These subsequent calls act as a safeguard to recover in the case where a league update scheduled event was missed. Each morning at 6:00 a.m., health checks are run by the server to try and identify any bad data that may exist within the database. A health check report is then emailed to the author which includes identifiers of the out-of-sync data if any are found. An example of this daily summary email is shown in Figure 16.


admin@parlayleagues.com

Health Check Successful

To: Drew Lohmeyer

Inbox - Google 6:00 AM

Parlay

the sports betting league

Invalid leagues: None

Invalid bet ids: None

POWERED BY TWILIO SENDGRID

Figure 16. Daily Server Health Check Report

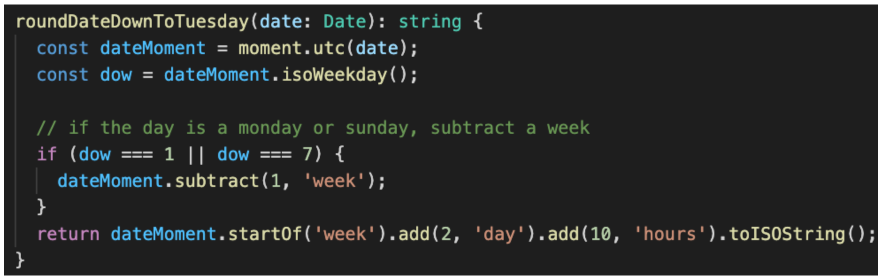
4. Implementation

4.1. Working with Timestamps

A large majority of the data used in the Parlay web application involves timestamps. Leagues have start and end dates and are broken down into weekly time frames. Games have start times and bets have creation dates. Games and bets are only to be shown within the proper week in a league. In order for Parlay to be successful, the management of these various timestamps needs to be organized and consistent. This is achieved with a 3rd party package called *moment.js* which is used in both the client-side Angular code and the server-side Node.js code.

Moment.js is a package dedicated to simplifying timestamp management and display.[7] Time formatted objects or strings can be passed into moment constructors by either using the basic `moment(<time>)` syntax which processes the time input with the machine's current locale timezone or the UTC `moment.utc(<time>)` syntax which processes the time input as a UTC formatted time. The time input is optional for both of these constructors. If no time parameter is provided, the current time and date is used. In the Node.js server code, the UTC constructor is always used. This is because all timestamps are stored as UTC formatted strings in the database. It is important when comparing dates and making key decisions, such as choosing which games should be returned to the client, that all timestamps are in the same timezone or format. When working with the timestamps in the client code, things become a little more challenging. UTC formatted moment objects are still primarily used in the controller files, but the normal moment constructor is required in any of the user-facing code. This allows all of the time-oriented data in the application to be displayed dynamically based on which timezone the client is accessing the web page from.

Once moment objects are created, there are many convenient time-oriented methods available for use. Common moment operations used within Parlay include adding or subtracting units of time to or from a moment object, comparing two moment objects to determine which is first or last, figuring out which day of the week a moment object is on, formatting a moment object into a human-readable string, and converting a moment object into a UTC string. The code snippet shown in Figure 17 shows an example of the moment.js package being used within the Create League component. This method takes in a JavaScript Date object and returns a UTC formatted string representing a date that has been rounded down to the previous Tuesday. This method is used when calculating a league's duration. Leagues can begin on Tuesday, Wednesday, Thursday, or Friday, but all leagues end on a Tuesday. Therefore, when calculating a league's duration it is useful to round the selected start date down to the nearest Tuesday in order to evaluate the correct number of weeks in a league. In this method, the `date` input is first used to instantiate a UTC formatted moment object. If the input is on a Sunday or Monday, one week must be subtracted from the moment in order for the `startOf('week')` operation to place the moment object on the correct Sunday, two days before the desired date. Two days and ten hours are then added to that result which is then converted to a UTC formatted string to be returned.

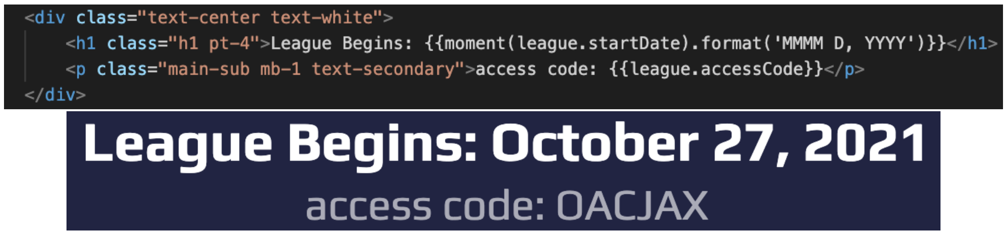


```
roundDateDownToTuesday(date: Date): string {
  const dateMoment = moment.utc(date);
  const dow = dateMoment.isoWeekday();

  // if the day is a monday or sunday, subtract a week
  if (dow === 1 || dow === 7) {
    dateMoment.subtract(1, 'week');
  }
  return dateMoment.startOf('week').add(2, 'day').add(10, 'hours').toISOString();
}
```

Figure 17. Example of Moment.js Usage

Figure 18 shows an example of formatting a league's start date in a component's HTML file and the resulting text shown on the user interface. This text is dynamically generated based on the accessing client's timezone. The `moment.format()` method takes a special string as input that is used to specify how the date should be shown. The 'MMMM' tells moment to include the full month name, the 'D' shows the date number without leading zeros for single digit numbers, the ',' is a literal comma, and the 'YYYY' maps to the full four digit year. Moment.js contains an expansive list of formatting options beyond what is shown here.



```
<div class="text-center text-white">
    <h1 class="h1 pt-4">League Begins: {{moment(league.startDate).format('MMMM D, YYYY')}}</h1>
    <p class="main-sub mb-1 text-secondary">access code: {{league.accessCode}}</p>
</div>
```

League Begins: October 27, 2021

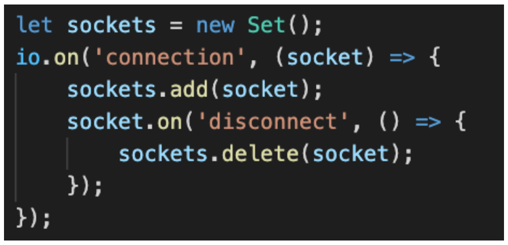
access code: OACJAX

Figure 18. Formatting a Date String Using Moment.js

4.2. Transmitting Real-Time Data

In order to prevent users from being able to gain a competitive advantage, it is important that Parlay's client interface is always displaying the most up-to-date game odds that are available. Users shall be prevented from leaving a browser tab open for a long period of time and holding on to the odds that were originally shown when the page loaded. This would allow users to compare the previous odds with the current odds using a separate client window and place their bets using the client showing the more advantageous odds. This undesirable scenario is prevented in two different ways. First, when a bet is placed in Parlay, the expected odds and payout displayed in the client must match up with the corresponding odds and payout stored in the database in order for the bet to be placed. However, this only partially solves the issue at hand because this solution on its own could deliver a poor user experience. A user could be creating their bet with the odds changing underneath them, unknowingly. Then, when the user tries to place the bet, it would be invalid because the odds have changed, and the only way to fix the issue would be to refresh the browser. This leads to the second piece of the solution: transmission of real-time data between client and server in order to keep the two parties in sync at all times, providing a user-friendly, secure mechanism for placing bets. This real-time data transmission is made possible using a 3rd party dependency called *socket.io*.

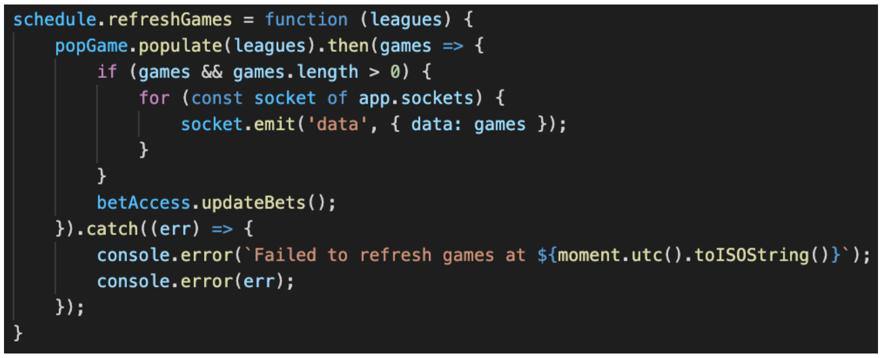
Socket.io is a JavaScript based library based on the WebSocket communication protocol. It allows for bi-directional communication between web clients and servers.[8] Upon server start, Parlay instantiates a socket.io object. Then, using an event-driven approach, the socket.io instance subscribes and listens for new client connections and stores each active connection using a JavaScript `Set`. The subscription code snippet is shown in Figure 19. As each client socket connection is passed into the event callback, it is first added to the active sockets `Set` object. Then, a 'disconnect' event is listened for on each socket. On socket disconnection, the socket is removed from the `Set` of active connections.



```
let sockets = new Set();
io.on('connection', (socket) => {
    sockets.add(socket);
    socket.on('disconnect', () => {
        sockets.delete(socket);
    });
});
```

Figure 19. Subscribing to Client WebSocket Connections

This `Set` of active socket connections is then used by the server to know which connections events need to be emitted to. Every time a call to the 3rd party SportspageFeeds API is made, any game that has been updated is immediately sent to client as a socket event emission. This allows all clients to always stay in sync with the latest data that has been collected by the Parlay server. A screenshot of the *refreshGames* method is shown in Figure 20. After the game data is retrieved, all active socket connections are iterated over and emit a 'data' event containing all of the game objects that have been updated.



```
schedule.refreshGames = function (leagues) {
    popGame.populate(leagues).then(games => {
        if (games && games.length > 0) {
            for (const socket of app.sockets) {
                socket.emit('data', { data: games });
            }
        }
        betAccess.updateBets();
    }).catch((err) => {
        console.error(`Failed to refresh games at ${moment.utc().toISOString()}`);
        console.error(err);
    });
}
```

Figure 20. Emitting Socket Events to Client When Refreshing Game Data

From a client perspective, each client first must register themselves with the server to be added to the server's `Set` of active connections. After registering, a client must listen for 'data' events being emitted from the server. Upon execution of an event, the client can notify any subscribers to the data of the updated games so the user interface can be updated accordingly. This registration and event listening process in the client are captured in Figure

21.

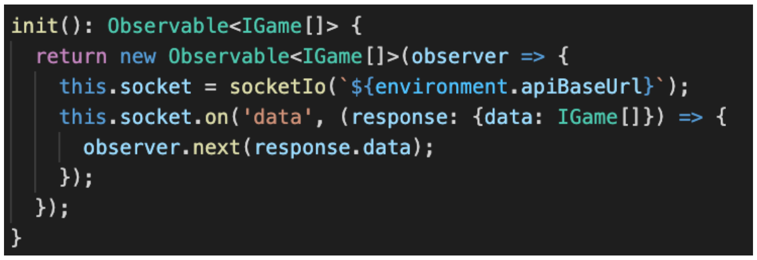

```
init(): Observable<IGame[]> {
  return new Observable<IGame[]>(observer => {
    this.socket = socketIo(`${environment.apiBaseUrl}`);
    this.socket.on('data', (response: {data: IGame[]}) => {
      observer.next(response.data);
    });
  });
}
```

Figure 21. Registering Client With Server Socket

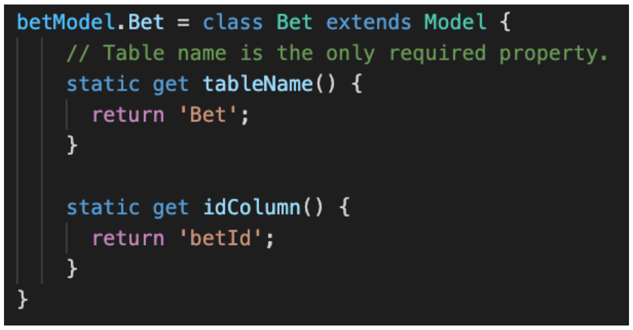
To summarize, a typical scenario begins with the client calling the `init` method in Figure 21 and subscribing to the observable that is returned for any game updates. Inside of the `init` method, a client socket instance registers itself with the server. This action fires a 'connection' event which enters the server event listener in Figure 19. At this point the connection between the client and server is established. While this connection is active, if a scheduled game refresh were to occur, calling the `refreshGames` method in Figure 20, a 'data' event would be sent to the client. This would trigger the 'data' event listener shown in Figure 21, thus receiving the refreshed game data in real-time. In doing so, the client is able to remain in sync with the server and always display the most recent game data without making users refresh their browsers.

4.3. Use of Object-Relational Mapping for Database Access

Object-relational mapping, or ORM, allows for database tables and entries to be accessed and created efficiently from a programming language that is not geared toward databases. For this project, Parlay's Node.js server, written in JavaScript, needed to to work closely with the MySQL database. Natively, there is no simple way to create and test database queries in JavaScript. Also, writing out long, complex SQL queries as string literals is tedious and error-prone. *Objection.js* is a 3rd party ORM package that is used heavily in the Parlay web server when accessing the database.

Objection.js is built upon a SQL query builder called *knex* and is designed and tested for querying MySQL databases from Node.js programs. Objection.js allows developers to define models to represent database tables and, optionally, the relationships between the models to assist with the query implementation.[9] Figure 22 shows an example of defining a database table model using objection.js. The only required property is the *tableName* property which maps to the actual table name used in the database. An *idColumn* property is also defined to inform the ORM tool of the table's identifier column name.

Once a model class is defined, Objection.js provides numerous methods to use when needing to interact with various database tables. The most common methods used in Parlay are *select*, *join*, *where*, *insert*, and *patch*. These methods all map to common SQL commands and

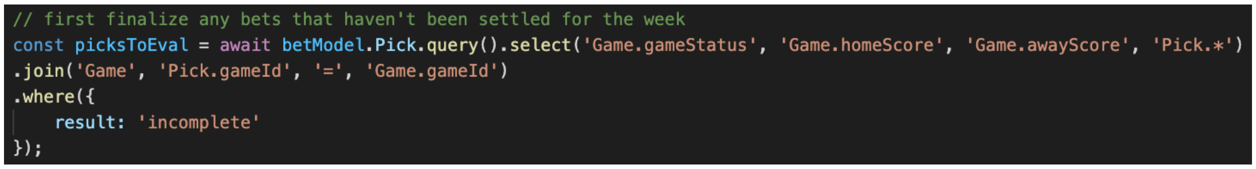


```
betModel.Bet = class Bet extends Model {
    // Table name is the only required property.
    static get tableName() {
      return 'Bet';
    }

    static get idColumn() {
      return 'betId';
    }
}
```

Figure 22. Database Bet Model Defined With Objection.js

are functionally equivalent to one another. The primary difference is that the Objection.js method versions are much easier to use syntactically when coding in JavaScript. Figure 23 shows an example of generating a query involving the Pick and Game tables. The purpose of this query is find all picks and their associated game information that have an 'incomplete' *result*. The query begins on the Pick table which is joined with the Game table on the *gameId* keys. The columns desired from each table are specified as the parameters passed into the *select* method. Lastly, only the table rows containing a *result* field of 'incomplete' are returned based on the *where* method. A resulting array of objects is then returned from the query.



```
// first finalize any bets that haven't been settled for the week
const picksToEval = await betModel.Pick.query().select('Game.gameStatus', 'Game.homeScore', 'Game.awayScore', 'Pick.*')
.join('Game', 'Pick.gameId', '=', 'Game.gameId')
.where({
    result: 'incomplete'
});
```

Figure 23. Example Select Query Using Objection.js

The *insert* and *patch* methods both work very similarly. The *insert* method requires that all columns necessary to create a new table row are passed into the method via an object. The *patch* method takes in one or more columns that need to be changed and is often combined with a *where* method to specify which row(s) the change shall apply to. Figure 24 shows the *insertBet* method used in Parlay to add a new bet entry to the database. All columns except for the *betId* column are provided in the object parameter. The *betId* column is auto-incremented and, therefore, does not need to be included.

4.4. Security

4.4.1. User Authentication and Authorization

One of the biggest areas in the Parlay web application where the application of proper security principles was essential was user authentication and authorization. User authentication is the process of verifying the identity of an individual making a request to the server. User authorization controls what information authenticated users are able to access within the

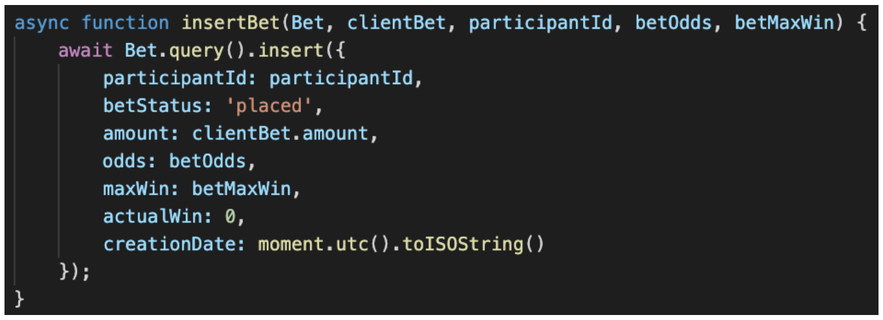

```
async function insertBet(Bet, clientBet, participantId, betOdds, betMaxWin) {
    await Bet.query().insert({
        participantId: participantId,
        betStatus: 'placed',
        amount: clientBet.amount,
        odds: betOdds,
        maxWin: betMaxWin,
        actualWin: 0,
        creationDate: moment.utc().toISOString()
    });
}
```

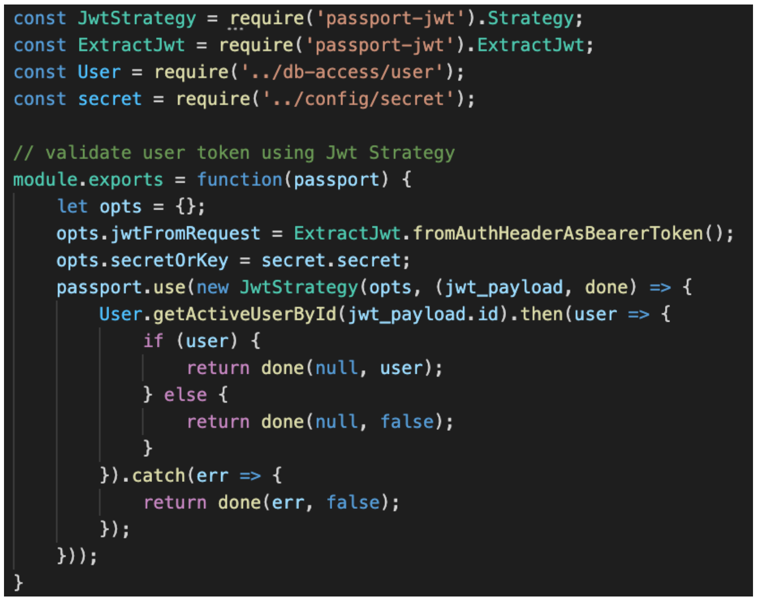
Figure 24. Inserting a New Bet With Objection.js

system. The Parlay web server must be able to permit or restrict access to user information in accordance with its security policy.

When a user attempts to login to the Parlay application, the user's email and password credentials are first authenticated. If the credentials are verified, Parlay creates a JSON web token and sends it back to the client. Tokens are signed with a private server secret that is also necessary to decrypt the token and reveal a user's unique identifier (ID) stored within it. Tokens are also assigned an expiration date of one week after they are issued. This forces users to re-authenticate themselves once a week to prevent leaked tokens from being valid for long periods of time in the hands of malicious users.

Once a token is issued to a client, the associated user gains access to the protected portions of the Parlay server. In order to access any of the application's API routes beyond registration and logging in, a client must provide its authenticated JSON web token in the 'Authorization' header of its requests. This allows the Parlay web server to determine which user is making the request and ensure that they have been authenticated. This authorization process is implemented using a 3rd party package called *passport*. Passport is a flexible middleware that can be used to validate a JSON web token passed along in the request headers and parse the token's payload. The code responsible for performing this operation is shown in Figure 25. The method uses the JSON web token and server secret to return the user ID payload in the callback function. The second parameter to the callback function is a *done* callback function that is used to return any errors and the authorized user information from the database. If the ID within the token payload does not map to a valid user in the database, a 401 Unauthorized error response is returned to the client.

Outside of authorizing client requests as legitimate users, additional authorization checks are also performed to ensure users are only accessing information that they are allowed to access. For example, users can only access information about a league that they are a member of. Thus, any API route that accesses league information first checks to make sure that the calling user is an active member of the league. Similarly, some routes restrict access to only league managers. Before deleting a league, the Parlay server ensures that the calling user is the league manager of the league that has been targeted for deletion.



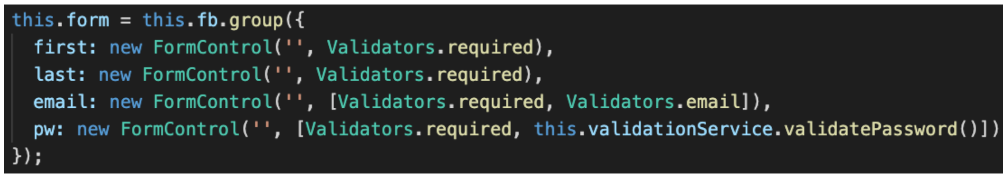
```
const JwtStrategy = require('passport-jwt').Strategy;
const ExtractJwt = require('passport-jwt').ExtractJwt;
const User = require('../db-access/user');
const secret = require('../config/secret');

// validate user token using Jwt Strategy
module.exports = function(passport) {
    let opts = {};
    opts.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken();
    opts.secretOrKey = secret.secret;
    passport.use(new JwtStrategy(opts, (jwt_payload, done) => {
        User.getActiveUserById(jwt_payload.id).then(user => {
            if (user) {
                return done(null, user);
            } else {
                return done(null, false);
            }
        }).catch(err => {
            return done(err, false);
        });
    }));
}
```

Figure 25. JSON Web Token Passport Middleware

4.4.2. Input Validation and SQL Injection Protection

On the client side, all inputs displayed on the user interface were driven by Angular Reactive Forms. Angular Reactive Forms provides the ability to attach built-in or custom validators on any form control.[10] Two commonly used built-in validators used in the Parlay web application include a *required* validator which prevents inputs from being left empty and an *email* validator that ensures a text input is in a valid email address format. Others include validators for min or max length of an input or validating an input against a regular expression. Figure 26 shows how these validators are applied to the form controls on the user registration form. The registration form also defines a custom validator for the password input. Custom validators allow developers to define additional criteria that must be met in order for a form control to be considered valid. In the case of this password input, passwords must be at least six characters, cannot contain a space, only consist of alpha-numeric and a set of special characters, and contain at least one alpha-numeric character and at least one special character. These client-side input validators act as a strong first step in restricting users to only entering safe and valid data.



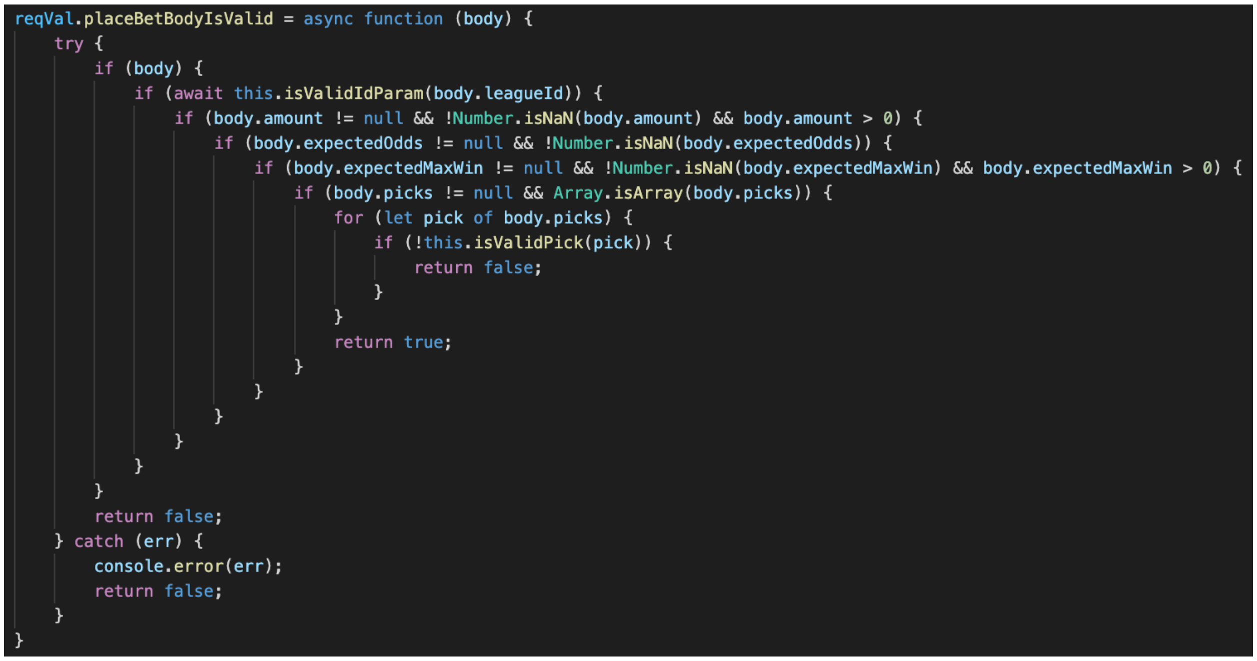
```
this.form = this.fb.group({
  first: new FormControl('', Validators.required),
  last: new FormControl('', Validators.required),
  email: new FormControl('', [Validators.required, Validators.email]),
  pw: new FormControl('', [Validators.required, this.validationService.validatePassword()])
});
```

Figure 26. Applying Validators to Angular Form Controls

It is also important to note that Angular provides additional client-side protections, natively, when attempting to mitigate malicious input-based attacks such as a cross-site scripting

(XSS) attack. A XSS attack injects malicious JavaScript code into an application that is then loaded and run on other client machines that access the application. By default, Angular applies DOM sanitization to any data variable that is injected into the DOM. This provides baseline protection against XSS attacks as malicious script tags are escaped rather than executed.

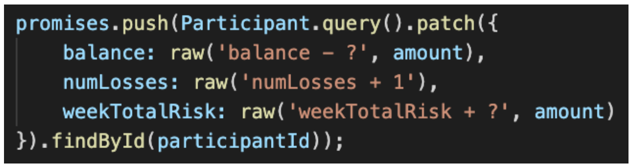
As HTTP requests are made to the server, it is important for the server to do its own validation on the inputs attached to each request as well. Before performing any actions with the data, each API route handler sends all inputs to a *validationService* that ensures all required inputs are provided and are of the correct data type. If any mistakes are found in the inputs, a 400 request error is returned to the client. Figure 27 shows the server input validation performed on the body of a place bet request. The body must contain a valid *leagueId* UUID, three valid number parameters (*amount*, *expectedOdds*, and *expectedMaxWin*), and an array of valid pick objects. If all checks are passed, *true* is returned from the method to indicate the body is valid.



```
reqVal.placeBetBodyIsValid = async function (body) {
    try {
        if (body) {
            if (await this.isValidIdParam(body.leagueId)) {
                if (body.amount != null && !Number.isNaN(body.amount) && body.amount > 0) {
                    if (body.expectedOdds != null && !Number.isNaN(body.expectedOdds)) {
                        if (body.expectedMaxWin != null && !Number.isNaN(body.expectedMaxWin) && body.expectedMaxWin > 0) {
                            if (body.picks != null && Array.isArray(body.picks)) {
                                for (let pick of body.picks) {
                                    if (!this.isValidPick(pick)) {
                                        return false;
                                    }
                                }
                                return true;
                            }
                        }
                    }
                }
            }
        }
        return false;
    } catch (err) {
        console.error(err);
        return false;
    }
}
```

Figure 27. Place Bet Request Body Validation

Lastly, security measures were taken into consideration when generating SQL queries from within the Parlay web server to protect against SQL injection attacks. A SQL injection attack inputs malicious data into the server with the intent of manipulating internal SQL query generation to steal sensitive data from the database or alter the database. Figure 28 shows how these attacks are prevented in Parlay via parameterized queries. The '?' parameters within the raw SQL strings are placeholders for data variables to be inserted into. With this syntax, the overall functionality of the generated SQL queries cannot change because the input variables are treated as encapsulated string literals.



```
promises.push(Participant.query().patch({
    balance: raw('balance - ?', amount),
    numLosses: raw('numLosses + 1'),
    weekTotalRisk: raw('weekTotalRisk + ?', amount)
}).findById(participantId));
```

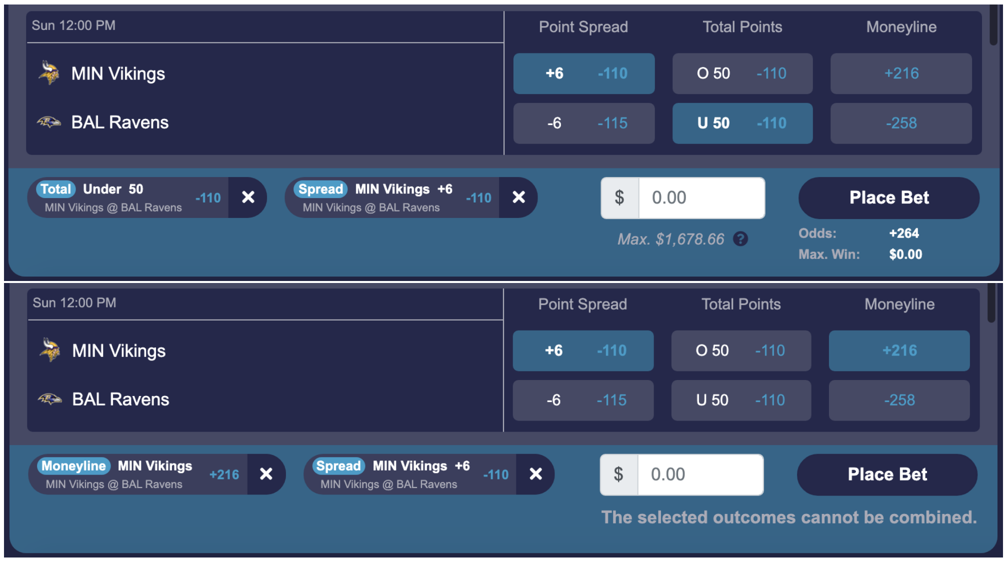
Figure 28. Defending Against SQL Injection Attacks

5. Testing

5.1. Unit Testing

As the code was written for the Parlay web application, a large amount of unit testing was performed. The majority of the unit tests ran against the application components were managed using a white-box testing strategy. White-box testing indicates that the underlying structure of the code is known by the tester. Since the author both wrote the code and performed the testing, this testing strategy was assumed.

One example of a unit test scenario executed during development was related to the user story: “As a league participant, I want to place new bets using the spread, total, and moneyline bookmaker odds for upcoming football games so that I can risk my currency to try and gain more currency.” As a user selects predictions for a game, there are only certain combinations of picks that should be considered valid. For example, a user can combine a *spread* pick with a *total* pick but cannot combine a *spread* pick with a *moneyline* pick in the same game. Figure 29 provides additional context on this matter.



The selected outcomes cannot be combined.

Figure 29. Valid and Invalid Selected Picks

Table 2 breaks down each of the scenarios that were tested to verify that the implementation for this feature was correct. As white-box testing was used, all single selections and all combinations of more than two selections could be skipped because the structure of the code made those cases redundant, trivially.

5.2. Integration Testing

The goal of integration testing is to ensure that various modules of an application work together in harmony. The first major phase of integration testing that took place in Parlay’s development life cycle was upon completion of the entire front-end user interface. The

#	Scenario	Input(s)	ExpectedOutput
1	Away spread and home spread	User clicks both spread buttons	"The selected outcomes cannot be combined."
2	Away spread and over total points	User clicks away spread and over total points	Picks are valid.
3	Away spread and under total points	User clicks away spread and under total points	Picks are valid.
4	Away spread and away moneyline	User clicks away spread and away moneyline	"The selected outcomes cannot be combined."
5	Away spread and home moneyline	User clicks away spread and home moneyline	"The selected outcomes cannot be combined."
6	Home spread and over total points	User clicks home spread and over total points	Picks are valid.
7	Home spread and under total points	User clicks home spread and under total points	Picks are valid.
8	Home spread and away moneyline	User clicks home spread and away moneyline	"The selected outcomes cannot be combined."
9	Home spread and home moneyline	User clicks home spread and home moneyline	"The selected outcomes cannot be combined."
10	Over total points and under total points	User clicks over total points and under total points	"The selected outcomes cannot be combined."
11	Over total points and away moneyline	User clicks over total points and away moneyline	Picks are valid.
12	Over total points and home moneyline	User clicks over total points and home moneyline	Picks are valid.
13	Under total points and away moneyline	User clicks under total points and away moneyline	Picks are valid.
14	Under total points and home moneyline	User clicks under total points and home moneyline	Picks are valid.
15	Away moneyline and home moneyline	User clicks away moneyline and home moneyline	"The selected outcomes cannot be combined."

Table 2. Pick Selection Test Case Breakdown

completion of the front-end Angular portion of the project was a development milestone and provided a timely opportunity to test the interactions and data communication between various components using expansive mock data. Interactions between the components making up a league's life cycle were one area of the application tested thoroughly.

Initially, a mock league was created and placed in an 'Upcoming' state. While in an 'Upcoming' state, a league's start date and access code are displayed along with the league's members and settings. Test cases were executed to confirm that a league manager could generate a new league access code, change league settings, leave the league, delete the league, and remove users from the league. This same page was also tested by viewing an 'Upcoming' league as a normal league member. A league member is only allowed to leave a league in this state. Next, an 'Active' league state was tested. The same application route now displays all of the upcoming game information, user betting information, and performance and ranking information for all other members in the league. League functionality such as placing bets, viewing bets, viewing league standings, and viewing previous week information were all validated. The mock data was progressed to simulate weeks of time advancing which brought the league to its final state, the 'Completed' state. The 'Completed' league state provides a read-only summary of all league activity that took place over the course of the league. The only action allowed is a reactivation of the league which can only be performed by the league manager. Overall, these tests verified that the user interface components working together to support a Parlay league were successfully integrated.

Additional integration testing was also performed during the development of the back-end server. As API endpoints were implemented, the front-end mock data was gradually replaced with actual data being retrieved and processed from the database. The integration of client and server code allowed for entire user stories to be tested and validated.

5.3. Acceptance Testing

Acceptance testing is testing performed by end-users of a product with the objective of validating an application against its requirements.[11] After successful deployment of the Parlay web application, user acceptance testing was conducted in the form of a three week pilot league. The author along with seven others participated in the league. Before the league began, all seven users were informed of the project's requirements, given a high-level functional overview, and were asked to report any questionable behavior that they observed. There were a handful of work items that came out of the inaugural Parlay league based on user feedback and application performance, but, for the most part, the league was a great success.

Nearly every user's initial reaction to the application was the question: "Can I use my phone?". At this point, there was some mobile screen size support, but many areas of the site were not functional. After a second pass, all critical user functionality was made available for use from a mobile device. A couple of defects were identified early on in the pilot league too. The first was a race condition when processing multiple bets for the same user at the same time. All bets are processed asynchronously by the Node.js server. As multiple bets were processing at the same time, a user's current balance was inaccurate in some cases due to one bet updating the balance after a separate bet had already retrieved the initial balance. A code fix was inserted in the bet processing code to ensure that the retrieval and updating of a user's balance is an atomic operation. The second bug was regarding duplicate bets being placed for users. After some investigation, it was found that multiple rapid clicks of the *Place Bet* button was making multiple API POST requests to place the bet. This issue was fixed by disabling the *Place Bet* button immediately when clicked and only re-enabling it when the API call returned.

6. Deployment and Future Enhancements

6.1. Deployment

The first step to the deployment process was setting up an AWS EC2 server instance. AWS offers a wide range of instance types varying in memory and performance. For this project, an Amazon Linux 2 free-tier t2.micro instance was used to host the application.

Once the instance was successfully launched, a web server needed to be installed and configured. Nginx, a popular open-source software used to serve web content in an event-driven approach, was chosen as the web server for Parlay. Nginx was configured to listen for all web traffic communicating with the EC2 server instance on ports 80 and 443. Any incoming traffic via port 80 is redirected to port 443. Port 443 uses a secure network channel that encrypts all data passed between client and server using the Transport Layer Security (TLS) network protocol. In order to take advantage of this additional security layer, a TLS certificate needed to be signed and acquired from a trusted Certificate Authority (CA). Parlay's signed certificate from CA, *Let's Encrypt*, contains its domain name, *parlayleagues.com*, and its public key. The TLS certificate allows any client visiting Parlay's domain to establish a trusted, secure connection with the Parlay server. Once the network traffic is routed to the correct port, it is then routed to Parlay's active Node.js server via a reverse proxy. Parlay's Node.js server runs on the EC2 instance localhost port 3000 and is managed using a 3rd party process manager tool called PM2. PM2 allows Node.js processes to be efficiently run in a production environment with features such as automatic load balancing and log management.

Once the application was deployed and available via the EC2 instance IP address, the *parlayleagues.com* domain name was then purchased and associated with the instance. The domain and a domain email address were purchased from domain name registrar *Namecheap*. In order for the domain to be associated with the AWS EC2 server instance, an elastic IP address needed to be generated. Normally when a server instance stops or restarts, a new IP address is assigned to the instance. This behavior is not acceptable when attempting to map a single IP address to a domain name. An elastic IP address is a reserved public IP address that remains constant throughout any of the aforementioned instance life cycle events. Using *Namecheap*'s advanced DNS tab, an "A Record" containing the instance elastic IP address as a value and a "CNAME" record containing the instance public DNS as a value were associated with the domain. These two DNS records completed the association between the domain name and server. Additional DNS records were also added to the *parlayleagues.com* domain for trusted email generation via *sendgrid.com*, TLS certificate verification, and registration with the Google search engine.

6.2. Future Enhancements

Since deployment on August 1, 2021, Parlay has gained 66 total users and has successfully processed more than 4,000 bets. Parlay has a variety of future enhancements planned to improve usability and progressively expand its user base. Most importantly, a mobile application will be created for both iOS and Android app stores. A large majority of Parlay's active users access the Parlay web application with their mobile device browsers despite the limited functionality this provides. The creation of a mobile app is essential in order to offer Parlay's full suite of functionality on mobile devices and reach a broader set of users. The next most important enhancement is allowing users to interact with one another. Currently, users are fairly isolated from one another. Users can visually see how other users are performing but there are no means of interaction between them. User interaction in Parlay will eventually come in multiple forms including adding friends via friend requests, supporting direct user-to-user league invites, and enabling league chat rooms. A third enhancement is to give league managers more control over their leagues. While a league is active, league managers should be able to change league settings and possibly even adjust member balances. Lastly, a wider set of league offerings needs to be available year-round. Parlay only supports private betting leagues for the NFL and NCAAF sports leagues which only take place from September through February. Methods of expansion include public league availability and additional sports league support for sports such as basketball, baseball, and hockey.

7. Bibliography

[1] "National Football League Total Attendance at Regular Season Games 2008 to 2020." *Statista*, 14 Jan. 2021, `https://www.statista.com/statistics/193420/regular-season-attendance-in-the-nfl-since-2006/`.

[2] "Football Will Draw More than $20 Billion in Bets; $1.5 Billion in Revenue." *Play USA*, 9 Sept. 2021, `https://www.playusa.com/2021-football-betting-projection/`.

[3] Winters, Ken C. and Jeffrey L. Derevensky. "A Review of Sports Wagering: Prevalence, Characteristics of Sports Bettors, and Association with Problem Gambling." *Journal of Gambling Issues* 43 (2019): 102-107.

[4] "The Psychology of Sports Betting." *Kindbridge Health*, 30 Sept. 2021, `https://kindbridge.com/gambling/the-psychology-of-sports-betting/`.

[5] Kumar, Kuldeep and Sandeep Kumar. "A rule-based recommendation system for selection of software development life cycle models." *ACM SIGSOFT Softw. Eng. Notes* 38 (2013): 1-6.

[6] Pattinson, Tamara. "Relational vs Non-Relational Databases." *Pluralsight*, 19 Oct. 2021, `https://www.pluralsight.com/blog/software-development/relational-vs-non-relational-databases`.

[7] "Moment.js Documentation." *Moment.js*, `https://momentjs.com/docs/`.

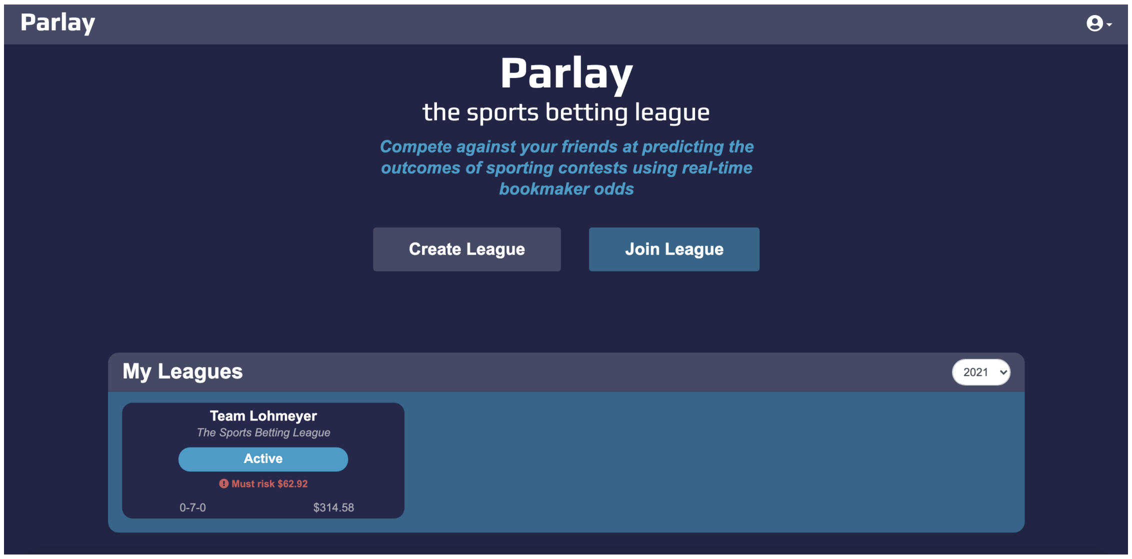
[8] "Socket.io Documentation." *Socket.io*, `https://socket.io/docs/v4/`.

[9] Koskimäki, Sami. "Objection.js: A SQL-friendly ORM for Node.js." *Objection.js*, `https://vincit.github.io/objection.js/`.

[10] "Validating form input." *Angular.io*, `https://angular.io/guide/form-validation`.

[11] Davis, Fred D. and V. Venkatesh. "Toward preprototype user acceptance testing of new information systems: implications for software project management." *IEEE Transactions on Engineering Management* 51 (2004): 31-46.

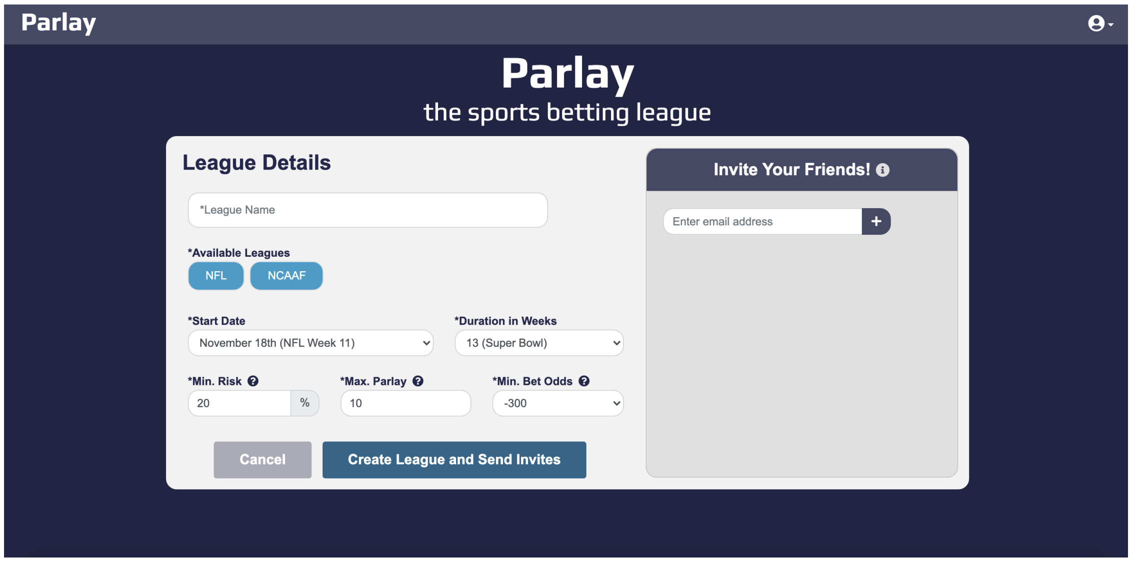
8. Appendices



Compete against your friends at predicting the outcomes of sporting contests using real-time bookmaker odds

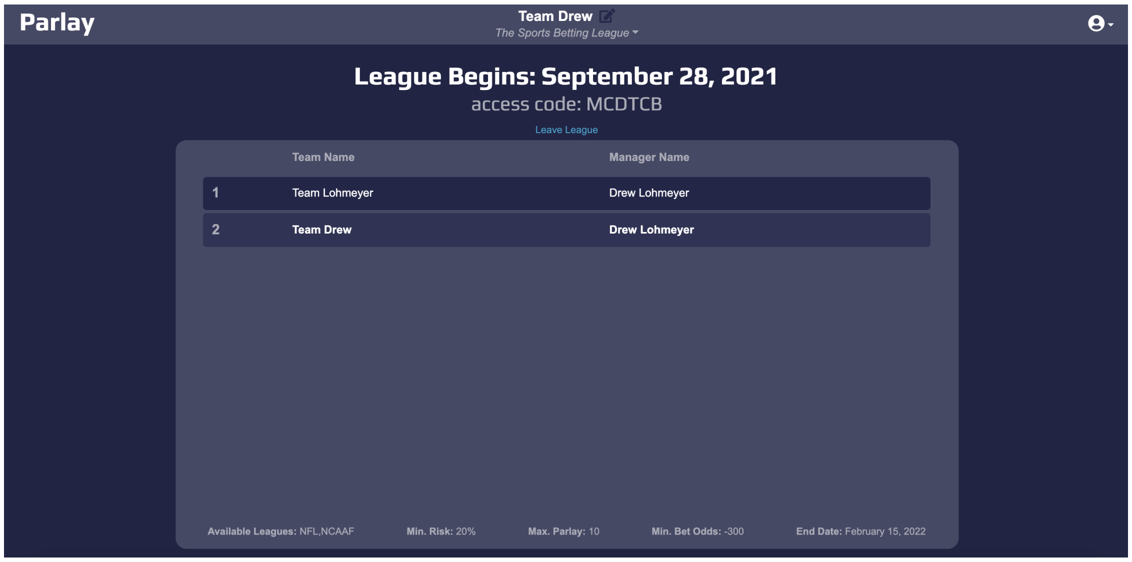
Team Lohmeyer
The Sports Betting League
Must risk $62.92
0-7-0 $314.58

Figure 30. Landing Page



*League Name
*Available Leagues: NFL, NCAAF
*Start Date: November 18th (NFL Week 11)
*Duration in Weeks: 13 (Super Bowl)
*Min. Risk: 20 %
*Max. Parlay: 10
*Min. Bet Odds: -300

Figure 31. Create League Page



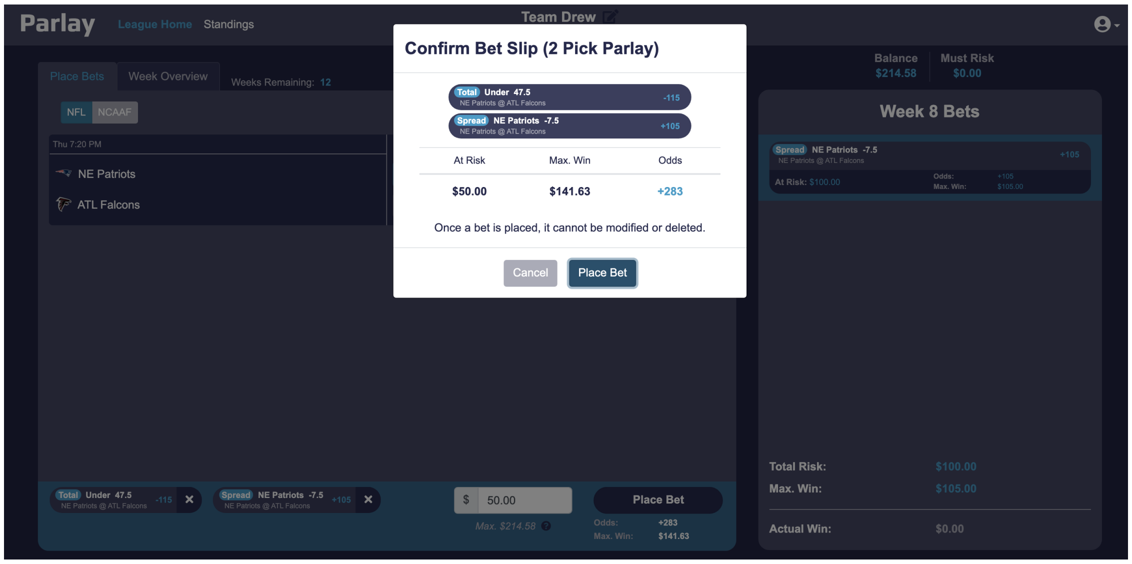
League Begins: September 28, 2021

access code: MCDTCB

	Team Name	Manager Name
1	Team Lohmeyer	Drew Lohmeyer
2	Team Drew	Drew Lohmeyer

Available Leagues: NFL,NCAAF   Min. Risk: 20%   Max. Parlay: 10   Min. Bet Odds: -300   End Date: February 15, 2022

Figure 32. Upcoming League Home Page



Weeks Remaining: 12

Balance $214.58   Must Risk $0.00

Confirm Bet Slip (2 Pick Parlay)

Total Under 47.5 NE Patriots @ ATL Falcons -115

Spread NE Patriots -7.5 NE Patriots @ ATL Falcons +105

At Risk	Max. Win	Odds
$50.00	$141.63	+283

Once a bet is placed, it cannot be modified or deleted.

Week 8 Bets

Spread NE Patriots -7.5 NE Patriots @ ATL Falcons +105

At Risk: $100.00   Odds: +105   Max. Win: $105.00

Total Under 47.5 NE Patriots @ ATL Falcons -115

Spread NE Patriots -7.5 NE Patriots @ ATL Falcons +105

$ 50.00

Max. $214.58

Odds: +283

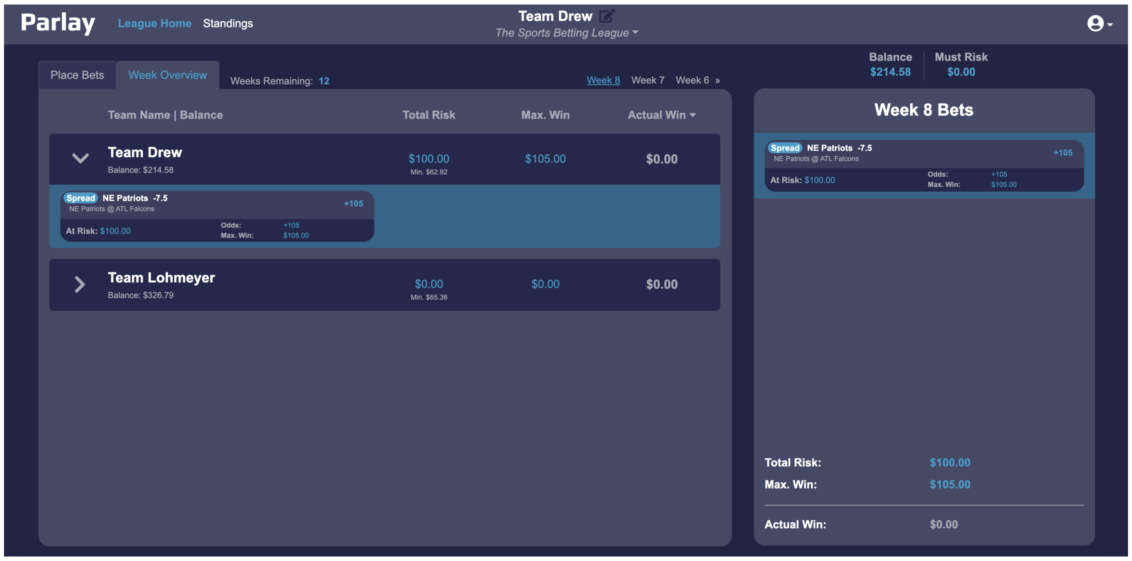
Max. Win: $141.63

Total Risk: $100.00

Max. Win: $105.00

Actual Win: $0.00

Figure 33. Place Bet Modal



Weeks Remaining: 12

Balance $214.58 | Must Risk $0.00

Team Name | Balance	Total Risk	Max. Win	Actual Win
Team Drew Balance: $214.58	$100.00 Min. $62.92	$105.00	$0.00
Team Lohmeyer Balance: $326.79	$0.00 Min. $65.36	$0.00	$0.00

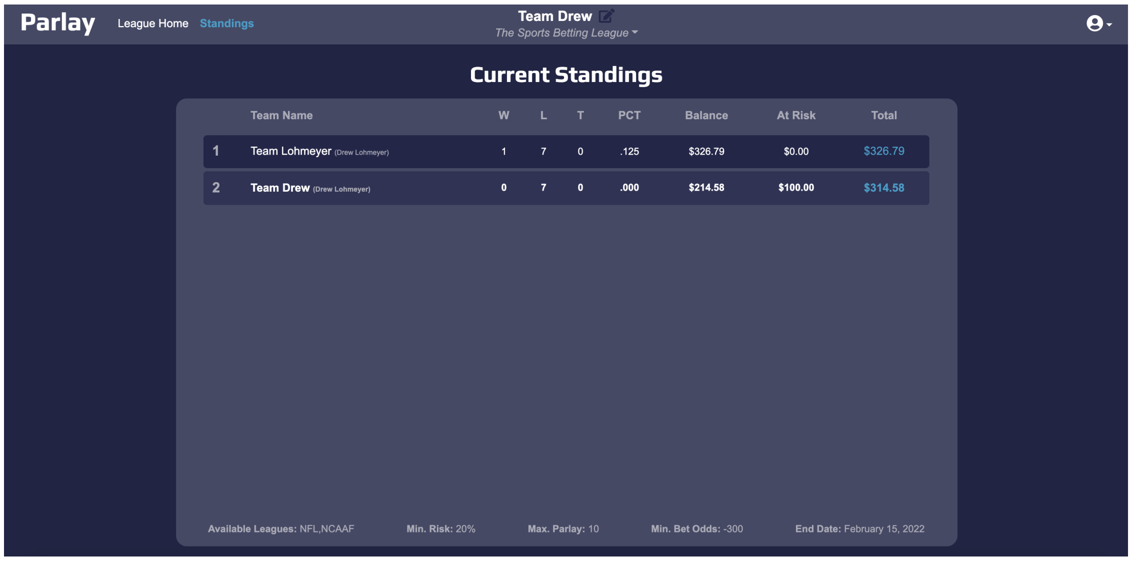
Spread NE Patriots -7.5 +105
NE Patriots @ ATL Falcons
At Risk: $100.00 Odds: +105 Max. Win: $105.00

Week 8 Bets

Spread NE Patriots -7.5 +105
NE Patriots @ ATL Falcons
At Risk: $100.00 Odds: +105 Max. Win: $105.00

Total Risk: $100.00
Max. Win: $105.00
Actual Win: $0.00

Figure 34. Week Overview Tab



Current Standings

	Team Name	W	L	T	PCT	Balance	At Risk	Total
1	Team Lohmeyer (Drew Lohmeyer)	1	7	0	.125	$326.79	$0.00	$326.79
2	Team Drew (Drew Lohmeyer)	0	7	0	.000	$214.58	$100.00	$314.58

Available Leagues: NFL,NCAAF Min. Risk: 20% Max. Parlay: 10 Min. Bet Odds: -300 End Date: February 15, 2022

Figure 35. League Standings Page