Bets-R-Us
Systems Requirement Specifications

Preface

People have been gambling on sporting events for over 3,000 years, since the first ever Olympic Games in Olympia Greece circa 776 BC, to the Roman colosseum games of 70-80 AD, to modern day Football or Baseball. Historically sportsbook gambling was viewed as a vice and not legal in most states, and as a result, it was run by criminal organizations and illegal bookmakers for profit. Over roughly the past 20 years however, the face of sports gambling has been changing in the United States.   The past couple decades have seen more and more states legalizing sportsbook gambling as a way to generate tax revenue and that has resulted in a wave of online sports betting. Some applications already exist for this purpose but our application's goal is to make it easier to understand and introduce it to a more casual gambling audience.

Introduction

The following SRS Document outlines the requirements for a sportsbook gambling application. The application will provide users with a secure and entertaining platform to wager on a variety of sporting events. The application will require a login and password to access the user's account. Users will be able to view upcoming events as well as the live odds associated with them. If a user desires, they may place wagers on games in any of the four major professional sports leagues. (Baseball, Football, Hockey, Basketball)   Upon the completion of the event, the application will verify the final score of the contest and winning wagers will have the appropriate funds deposited into the player's bankroll. The system will have a user-friendly design and be responsive. The system will also provide users with the ability to track their winnings and losses, as well as provide a variety of payment options.

Glossary of Terms

Term	Definition
Bankroll	The total amount of money that a sports bettor has set aside for the purpose of betting.
Cover	A winning outcome on a point-spread bet, in which the underdog team either loses by fewer points than predicted or wins the game outright.
Dog	A team or athlete considered to be the underdog in a given contest, usually due to lower odds of winning.
Draw	A tied or even outcome in a game or contest, resulting in neither team being declared the winner or loser.
Favorite	A team or athlete considered to be the likely winner of a given contest, usually due to higher odds of winning.
Handicap	A point spread or line assigned by oddsmakers to equalize the chances of each team winning a contest.
Hook	A half-point added to a point spread to prevent a tie or push outcome.
Line	The point spread or odds set by oddsmakers for a specific game or contest.
Lock	A highly confident pick to win, often used to refer to a sure bet.
Longshot	A team or athlete considered to be an unlikely winner of a contest, usually due to lower odds of winning.
No-Action	A bet that is voided and results in no payout, often due to a canceled or postponed event.
Odds	The numerical representation of the likelihood of a specific outcome, used to determine payouts for winning bets.
Outright-Bet	a wager on the overall winner of a tournament or competition
Over	A bet on the combined score of two teams to be over a set total in a game.

Parlay	A type of bet in which multiple selections are combined into one wager, with the winnings from each leg being used to fund the next.
Point-Spread	A handicap assigned to a favorite team, reflecting the number of points they are expected to win by.
Push	A tied outcome on a point-spread bet, resulting in no winnings or losses for the bettor.
Stake	The amount of money placed on a specific bet.
Under	A bet on the combined score of two teams to be under a set total in a game.
Underdog	A team or athlete considered to be the unlikely winner of a given contest, usually due to lower odds of winning.
Wager	A bet placed on the outcome of a sporting event.

User Requirements (in order of priority)

UR-1. User registration and login: Users must be able to create an account and log in to the app. Users should also be able to delete their accounts and make changes to their information.

UR-2. Banking requirements. Users must be able to deposit money via credit card or bank transfer to their betting accounts to place bets. Users must also be able to withdraw winnings to their bank account.

UR-3. Live odds and scores: The app must provide real-time odds and scores for all available sports and events that are supported.

UR-4. Betting options: The app will offer a wide range of betting options for the 4 major sports leagues, MLB, NFL, NHL, and NBA currently.

UR-5. User history and tracking: The app should allow users to view their past bets and track their betting history.

UR-6. Easy navigation: The app must be easy to navigate and understand, with clear and concise information

UR-7. Application should be reliable and responsive. It shall handle all transactions in a timely manner with limited down time.

System Requirements

SR-1 User Registration and Login

SR-1.1 Users shall be able to create and account with the following information

SR-1.1.1 Required information
Username
Password
First Name
Last Name
Email Address
Street Address
City
State
Zip Code

SR-1.1.2 Successful processing results in a User object and Bankroll object created successfully with a 0.00 balance in the Bankroll and all fields populated for the User.

SR-1.1.3 If processing fails on exception an error message should appear with a reason for the failure.

SR-1.1.3.1 Processing failure for username already exists. Unique username is required for new accounts. User creation shall fail with error message if account creation is attempted with an existing username

SR-1.1.3.1 Processing failure for account with email already exists. One account per email address is allowed, user creation shall fail with error message if account creation is attempted with an email address already in use.

SR-1.2 Users shall be able to log into and view their account information by providing their username and password.

SR-1.2.1 Upon successful input of user credentials, the screen displays the user's information.
Username
First Name
Last Name
Email Address

Street Address
City, State, Zip Code
Bankroll Balance

SR-1.2.2 Invalid credentials results in an error message instructing the user that no account exists for the credentials provided.

SR-1.3 Users shall be able to edit their information in their account if needed.

SR-1.3.1 Users shall be able to edit first name, or last name in the event of name change.

SR-1.3.1 Users shall be able to edit their mailing address in the event of a move.

SR-1.3.1 Users shall be able to configure a new email address if they choose.

SR-1.4 Users shall be able to delete or deactivate their account by providing a valid username and password.

SR-2 Banking Requirements

SR-2.1 Users shall be able to use bank account information to add funds to their betting account

SR-2.1.1 Required inputs and processing
Name on account
Account Number
Routing Number
Amount of transfer

SR-2.1.2 Add funds to the bettor's account from a bank account. Upon successful input of all previous information the application shall deduct from the bank account the amount of transfer plus processing fees and deposit the amount of transfer into the bettor's account.

SR-2.1.3 Add funds to the bettor's account shall fail with error message if improper or incorrect account number or routing number is provided by the user. If the process fails then the user shall be prompted with an error message to please try again.

SR-2.2 Users shall be able to use credit card information to add funds to their betting account

SR-2.2.1 Required inputs and processing

Name on account
Credit Card Number
3-digit CVV code
Amount of transfer

SR-2.2.2 Add funds to the bettor's account from credit card processing. Upon successful input of all previous information, the application shall charge from credit card the amount of transfer plus processing fees and deposit the amount of transfer Into the bettor's account.

SR-2.2.3 If processing fails on exception an error message should appear with a reason for the failure.

2.2.3.1 Processing failure for invalid or incorrect card number. If an invalid card number is used, the process shall fail, and the user prompted with error message That the card number is invalid.

2.2.3.2 Processing failure for invalid CVV number. If the CVV number is incorrect, the process shall fail and the user will be prompted with the message that the card used is not valid.

SR-2.2.3.3 Processing failure for expired credit card. If the expiration date that is entered by the user is before the actual date, the process shall fail and the user will be prompted with a message that their card has expired.

SR-2.3 Users shall be able to view the balance of their betting account and make withdrawals

SR-2.3.1 Withdraw funds via electronic transfer to a bank account. Upon successful processing, the amount of withdrawal is deducted from the bettors account and transferred to the bank account of the user.

Required inputs
Amount of withdrawal
Bank Account number
Routing Number

SR-2.3.2 Withdraw funds via electronic transfer to bank account failure. If the user enters incomplete or incorrect bank information, the transfer shall fail and the user presented with the error message.

SR-2.3.3 Withdraw funds via check by mail. Upon successful processing, the amount of withdrawal is deducted from the bettor's account and a check generated in the amount specified to be mailed to the address of record on the account.

Required inputs
Amount of withdrawal

SR-2.3.5 (Priority 3)
Users should be able to log into their account and view their deposit and withdrawal history.

SR-3 Live Schedule, Scores and Odds Requirements

SR-3.1 Supported Sports

SR-3.1.1 The system shall support NBA betting.
SR-3.1.2 The system shall support NFL betting.
SR-3.1.3 The system shall support NHL betting.
SR-3.1.4 The system shall support MLB betting.

SR-3.2 Live Data Accuracy and Connection Health

SR-3.2.1 The system shall provide updates to the scheduling, scores, and odds database at a minimum of 6 times per hour.
SR-3.2.2 The system shall retrieve live scheduling, scores, and odds data from a reputable live data source.
SR-3.2.3 The system shall display live data source connection health to the user.
SR-3.2.4 The system shall display live data database connection health to the user.

SR-3.3 Live Data Processing and Error Handling

SR-3.3.1 The system should perform data validation on incoming data from any live data source.
SR-3.3.2 The system shall provide a data standardization mechanism to process incoming data from any live data source.
SR-3.3.3 The system shall store data from any live data source in a scalable, performance-based database.
SR-3.3.4 The system shall only store data from any live data source after data standardization is complete.
SR-3.3.5 The system shall implement error handling to handle data validation errors, system errors, and network errors during processing of data from all live data sources.

SR-3.3.6 Error handling shall include logging errors, providing error messages to users, and automatically retrying failed data processing operations.

SR-3.3.7 The system shall alert the user of a timeout error if data fetching or processing operations exceed 5 seconds.

SR-3.4 Live Scheduling

SR-3.4.1 The system shall display all current and upcoming sports events for supported sports.

SR-3.4.2 The system shall display the start time of an event if the event has not started for supported sports.

SR-3.4.3 The system shall not display the event start time after the event has started.

SR-3.4.4 The system shall indicate when an event is in progress, and when an event has ended.

SR-3.4.5 The system should support sorting of supported sport events by time nearest to farthest.

SR-3.4.6 The system should allow the user to select how many supported sport events are displayed at once.

SR-3.4.7 The system shall allow users to filter current and upcoming supported sport events by sport.

SR-3.4.8 The system shall display sport specific timing information while an event is being played.

SR-3.4.9 The system shall provide and display instant updates of event schedules for supported sports when live data is refreshed.

SR-3.5 Live Scores

SR-3.5.1 The system shall not display an event score if the event has not started yet.

SR-3.5.2 The system shall display the current event score if the event has started.

SR-3.5.3 The system shall display the final score after the event has ended.

SR-3.5.4 The system shall provide and display updates to live scores when prompted for a refresh by the user

SR-3.6 Live Odds

SR-3.6.1 The system shall display current odds for the supported betting types before and during an event.

SR-3.6.2 The system shall not display odds information for events that have ended.

SR-3.6.3 The system shall provide and display instant updates of odds when live data is refreshed.

SR-4 Betting Requirements

SR-4.1 Supported Betting Types

SR-4.1.1 The system shall support spread betting for all supported sports

SR-4.1.2 The system shall support over/under betting for all supported sports.

SR-4.1.3 The system shall support moneyline betting for all supported sports.

SR-4.1.4 The system should support parlay betting with a maximum of 20 legs.

SR-4.2 Betting Actions

SR-4.2.1 The system shall not allow for bets to be placed if live data functionality is in an error state.

SR-4.2.2 The system shall not allow bets to be placed if the database connection is in an error state.

SR-4.2.3 The system shall log all bets to a user's betting history.

SR-4.2.4 The system shall display all active user bets and all the bet information associated with the bets to the user.

SR-4.2.5 The system shall not allow for a bet to be modified after the bet is placed.

SR-4.2.6 The system shall not allow for a bet to be cancelled after the bet is placed.

SR-4.2.7 The system shall check the balance before the bet is placed to ensure the user has enough money in their account to place a bet.

SR-4.2.8 The system shall calculate and display a potential payout for user selected bets before bets are placed.

SR-4.2.9 The system shall payout a winning bet to a user’s account balance if they win.

SR-4.2.10 The system shall deduct the bet amount from the user’s bankroll upon successfully placing a bet.

SR-4.2.11 The system shall deposit back into the user’s bankroll the amount of the original bet in the event of a push.

SR-5 Betting History and Tracking Requirements

SR-5.1 Betting History

SR-5.1.1 The system shall have a database to store and retrieve users' betting history and transaction records, including bet types, odds, amounts, and outcomes.

SR-5.1.2 The system shall allow a user to view their full betting history.

SR-5.1.3 The system should allow users to search their betting history by date, sport, bet type, or outcome.

SR-5.1.4 The system shall store information about all user bets, including bet type, amount, odds, and outcome.

SR-5.1.5 The system shall display betting history in real-time showing the status of bets as they are placed, settled, or canceled.

SR-5.1.6 The system shall provide users with the ability to export their betting history and transaction data in a machine-readable format

SR-5.1.7 The system shall provide users with the ability to report any issues or errors related to their betting history or tracking.

SR-5.2 User Privacy

SR-5.2.1 The system shall keep betting information about a user restricted to only being accessible by authorized individuals.

SR-5.2.2 The system shall store all user data securely and in compliance with relevant data privacy regulations.

SR-6 Navigation and Ease of Use Requirements

SR-6.1 Intuitive Navigation: The system should be easy to navigate, with a clear and straightforward structure that is easy to understand.

SR-6.1.1 The program shall have multiple level GUIs

SR-6.1.2 The app should have an easily accessible main menu that displays all the different types of sport bets available, along with any relevant sub-categories.

SR-6.1.3 The menu should be organized in a logical and straightforward manner, with clear labels and icons that indicate the type of sport bets.

SR-6.1.4 When a user selects one of these sections, they should be able to see a list of available sport bets, along with clear descriptions and images to help them make their choice.

SR-6.1.5 the app should have a search function that allows users to quickly find specific sport bets or categories

SR-6.2 Consistent User Interface: The system should have a consistent user interface that makes it easy for users to understand how to use it.

SR-6.2.1 The app should have a standardized layout for all sport bets pages, with similar elements appearing in the same place on each page.

SR-6.2.2 The app should have consistent behavior across different pages and sport bets, such as using the same gesture or button to initiate a sport bet.

SR-6.3 User Feedback: The system should provide feedback text to users about their selection and the results of those actions.

SR-6.4 Help and Support: The system should provide help and support resources for users, such as a FAQ, tutorial, or customer support hotline under the main menu.

SR-6.5 Customization: The system should allow users to customize their experience, such as changing the appearance or settings, to meet their specific needs.

SR-6.5.1 The system should allow users to add sport bets to their favorites.
SR-6.5.2 The system should allow users changing the appearance or settings

SR-7 Response time and reliability Requirements

SR-7.1 Responsive Design: The system should be designed to be responsive, so that it works well on different devices and screen sizes.
SR-7.1.1 The gambling system should respond to all user inquiries within 5 seconds.
SR-7.1.2 If the response time of the system is greater than 5 seconds, the system must show a processing indicator to the user to indicate that the system is actively working on the response.
SR-7.2 Availability: The app should be always available for use, with minimal downtime for maintenance or updates
SR-7.2.1 The app should be available 24/7
SR-7.2.2 Downtime for maintenance or updates should not exceed 30 a week.
SR-7.3 Load Balancing: The app should be able to handle high traffic volumes and unexpected spikes in user activity, ensuring smooth and responsive performance for all users.
SR-7.3.1 The app should be able to handle 10000 users at the same time.
SR-7.3.2 The app response should not be impact at high traffic (<10000)
SR-7.4 Recovery: The app should have a robust disaster recovery plan in place, to ensure that it can quickly and effectively recover from unexpected failures or outages.
SR-7.4.1 The app should have a robust disaster recovery plan in place that allows for the recovery of all data prior to failure.
SR-7.4.2 The app should recover to full function within 30 minutes from failure or outage.

System Architecture

Bets-R-Us will utilize model-view-controller design pattern. The MVC pattern was chosen because it is easy to organize web application frameworks in this pattern. Furthermore, the pattern makes periodic release easy to plan and implement. The services are organized in a manner in which new services can be added modularly with little impact to the architecture. In the Bets-R-US software, the model is handled in MongoDB, the views are handled by ReactJS and handle the construction of the requests to the backend API. The controller portion consists of 5 controllers that control the flow of information for the User, Bet, Schedule, Banking, and

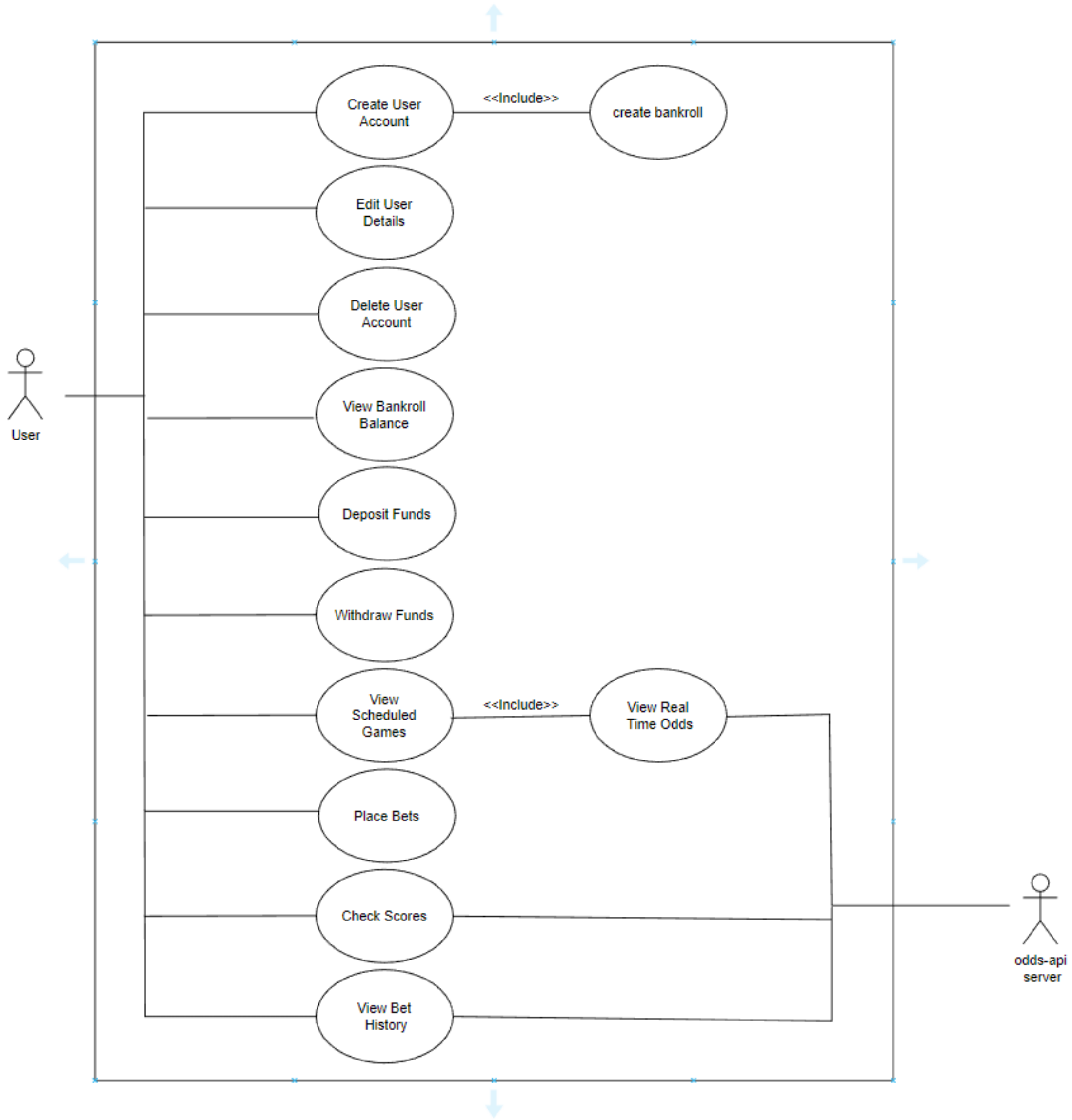
Scores objects. SwaggerAPI will be implemented as a visualization and demo tool until the front-end architecture can be laid out.   The development plan is to integrate a WebUI in React JS once the backend functionality is up and in a reliable state. This will allow greater focus on the backend functionality for the purpose of prototype.

Tech Stack

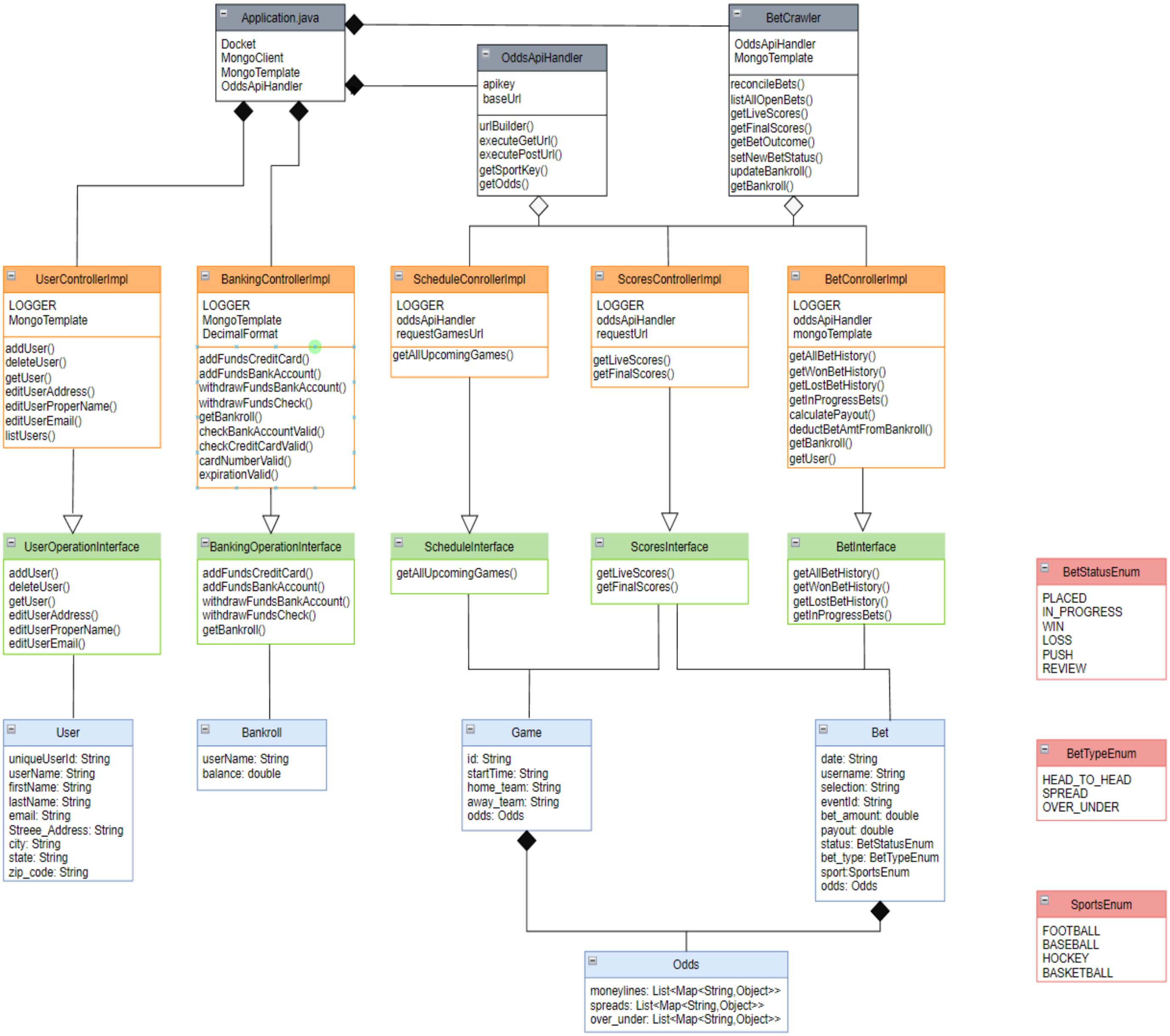
- Springboot for backend REST API
- MongoDB as backend database
- Integrated Swagger API docs for API visualization
- *ReactJS for front end/ UI
- Log4j for system logging
- Jackson-Databind for JSON parsing

System Models

Use Case Diagram



Class Diagram



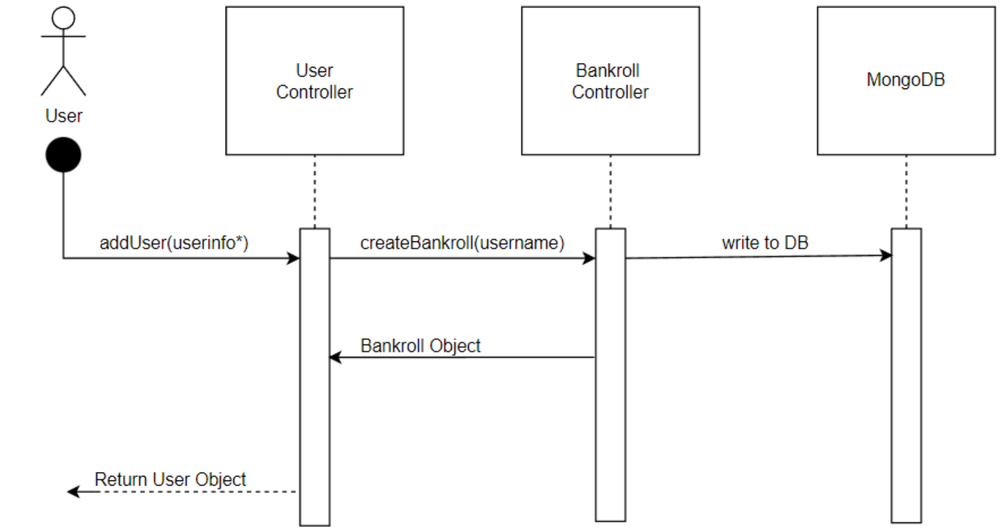
Use Cases

Use Case: Create User Account

Title:	**Adding a User Account**
Description:	**The user creates a new user account to Bets-R-Us, with user control and bankroll controller features, using a MongoDB database.**
Actors:	**The User is the actor that triggers the request to create a new user account.**
Stimulus (Trigger)	**The system receives the request to create a new user account. This is triggered by a user registration form on the site**
Preconditions:	**The user does not already have an account on the site.**
Postconditions:	● **A new user account has been created in the database.** ● **The user can log in to the site using their newly created account.**

Main Success Scenario:	● The system receives a request to create a new user account. ● The system validates the user's details, such as username, password, email address, and other required information. ● The system sets the user's bankroll limit using the bankroll controller feature. ● The system saves the user's details to the database using MongoDB. ● The system confirms that the user account has been created successfully. ● The user is notified that their account has been created and can log in to the site
Extensions:	● If the user already has an account on the site, the system displays an error message and prevents the creation of a new account. ● If the user's details are incomplete or invalid, the system displays an error message and prompts the site administrator to correct them. ● If there are technical issues with the database, the system displays an error message and prompts the site administrator to resolve the issue.
Priority:	Adding a new user account functionality is priority 0. Adding a new user account is basic functionality and is required in the base project.

Sequence Diagram Create User Account



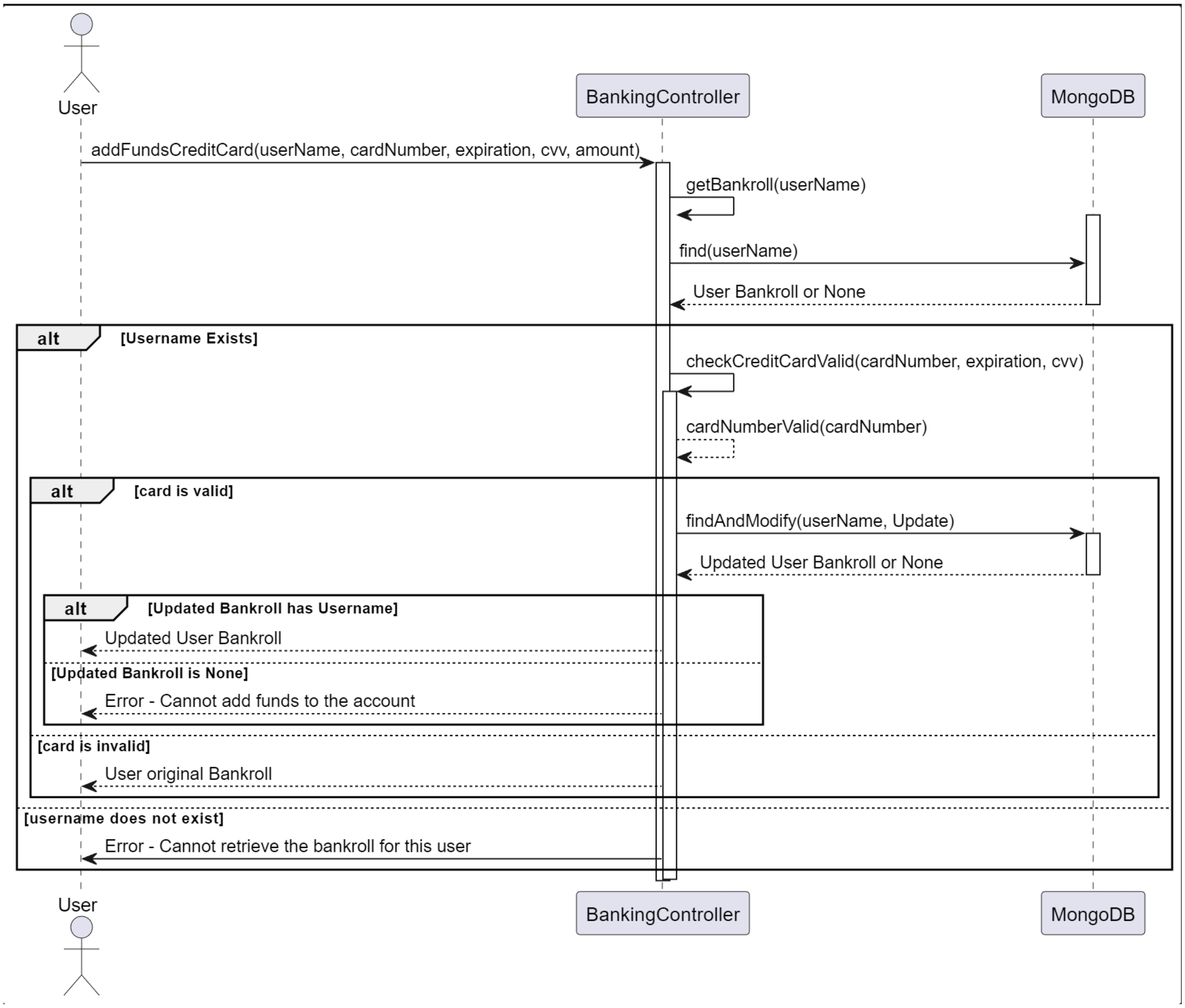
* where userinfo = (String userName, String firstName, String lastName, String email, String streetAddress, String city, String state, String zip)

Use Case: Add Funds to User Account with Credit Card

Title:	**Add Funds to User Account with Credit Card**
Description:	This use case describes the process of adding funds to a user's account using a credit card in a banking system. The system validates the credit card information and updates the user's bankroll in the database.
Actors:	User, Banking Controller, MongoDB

Stimulus (Trigger)	User requests to add funds to their account using a credit card.
Preconditions:	User has an account in the banking system.
Postconditions:	User's bankroll is updated with the added funds, or an error message is returned if the process fails.
Main **Success Scenario:**	● User initiates the process by sending a request to Banking Controller with required credit card information and the amount to be added. ● Banking Controller retrieves the user's bankroll from MongoDB. ● If the username exists, the system checks if the credit card is valid. ● If the credit card is valid, MongoDB updates the user's bankroll with the added amount. ● Banking Controller returns the updated user bankroll to the User.
Extensions:	● If the username does not exist: ○ Banking Controller returns an error message to the User: "Cannot retrieve the bankroll for this user." ● If the credit card is invalid: ○ Banking Controller returns the user's original bankroll to the User. ● If the updated bankroll is not found for the user in MongoDB: ○ Banking Controller returns an error message to the User: "Cannot add funds to the account."
Priority:	This is a priority 0 use case as the system could not function properly without account funding

Sequence Diagram Add Funds to Bankroll with Credit Card

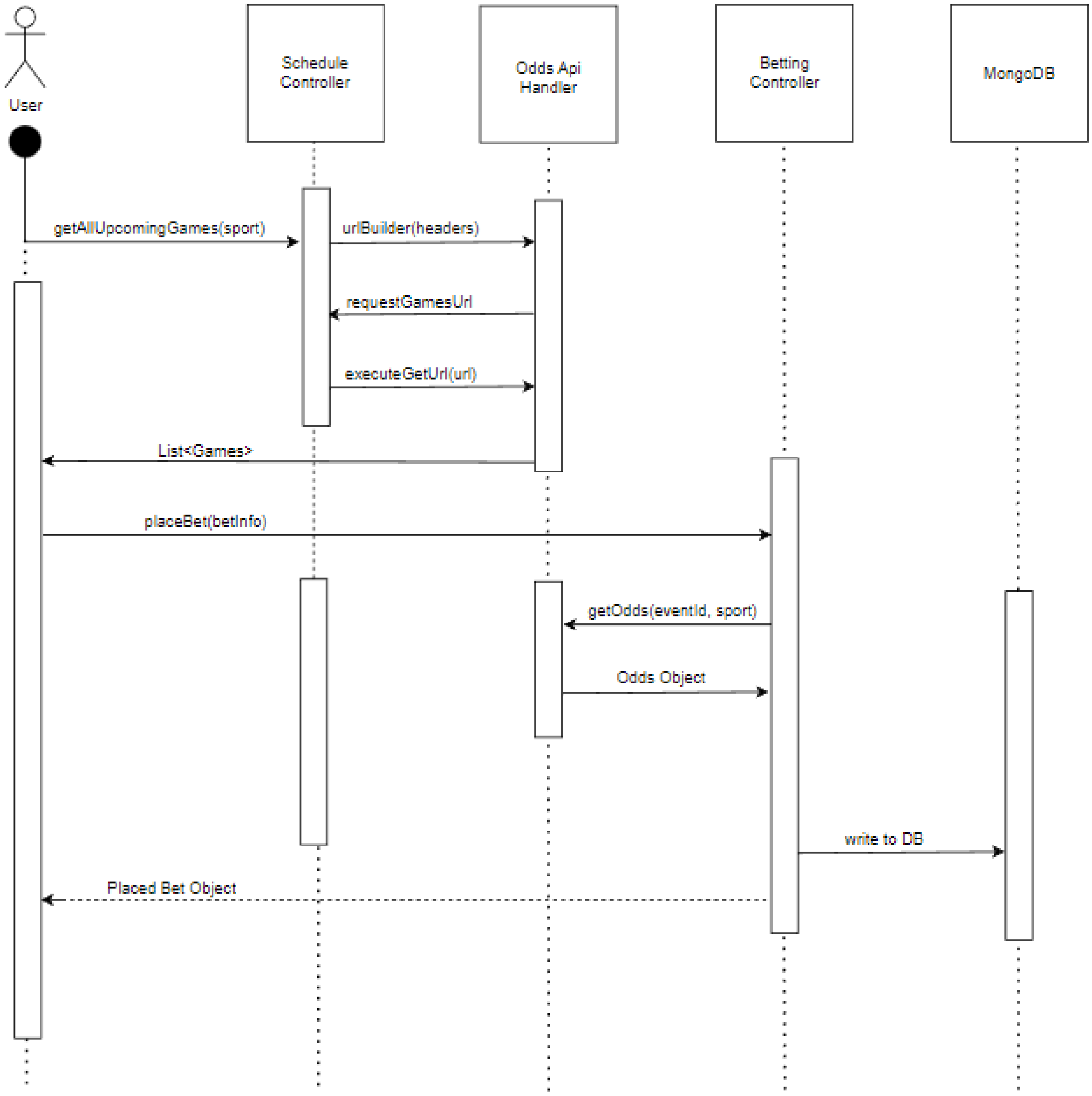

User
BankingController
MongoDB
addFundsCreditCard(userName, cardNumber, expiration, cvv, amount)
getBankroll(userName)
find(userName)
User Bankroll or None
alt
[Username Exists]
checkCreditCardValid(cardNumber, expiration, cvv)
cardNumberValid(cardNumber)
alt
[card is valid]
findAndModify(userName, Update)
Updated User Bankroll or None
alt
[Updated Bankroll has Username]
Updated User Bankroll
[Updated Bankroll is None]
Error - Cannot add funds to the account
[card is invalid]
User original Bankroll
[username does not exist]
Error - Cannot retrieve the bankroll for this user
User
BankingController
MongoDB

Use Case: User Placing a Bet

Title:	User Placing a Bet
Description:	The user checks the schedule of upcoming games with associated odds and picks one to place a bet on.
Actors:	The User is the actor that triggers the events of placing a bet. The OddsAPI is the actor that provides live scoring and odds.
Stimulus (Trigger)	The placing of a bet is triggered by a user from the betting interface, based on results of the schedule interface.
Preconditions:	The system is in a state of crawling open bets and checking scores for winners and losers asynchronously.
Postconditions:	The system is in a state of crawling open bets and checking scores for winners and losers asynchronously. After the bet has been placed however, it is added to the list of in progress bets in the database, the crawler continues to check until the score for the event is final.
Main Success Scenario:	After the bet has been placed, it is stored in the system and periodically checked against the final scores to see if the bet won. If the crawler determines the bet is a loss no further action is taken by the system. If the bet is a win, then the payout is added to the user's bankroll. In the event of a push, the user's original bet amount is refunded.

Extensions:	If a user tries to place a bet for more money than the amount in their bankroll then placing the bet should fail with error message insufficient funds please add funds.
Priority:	The place bet functionality is priority 0. Bet placing is basic functionality and is required in the base project.

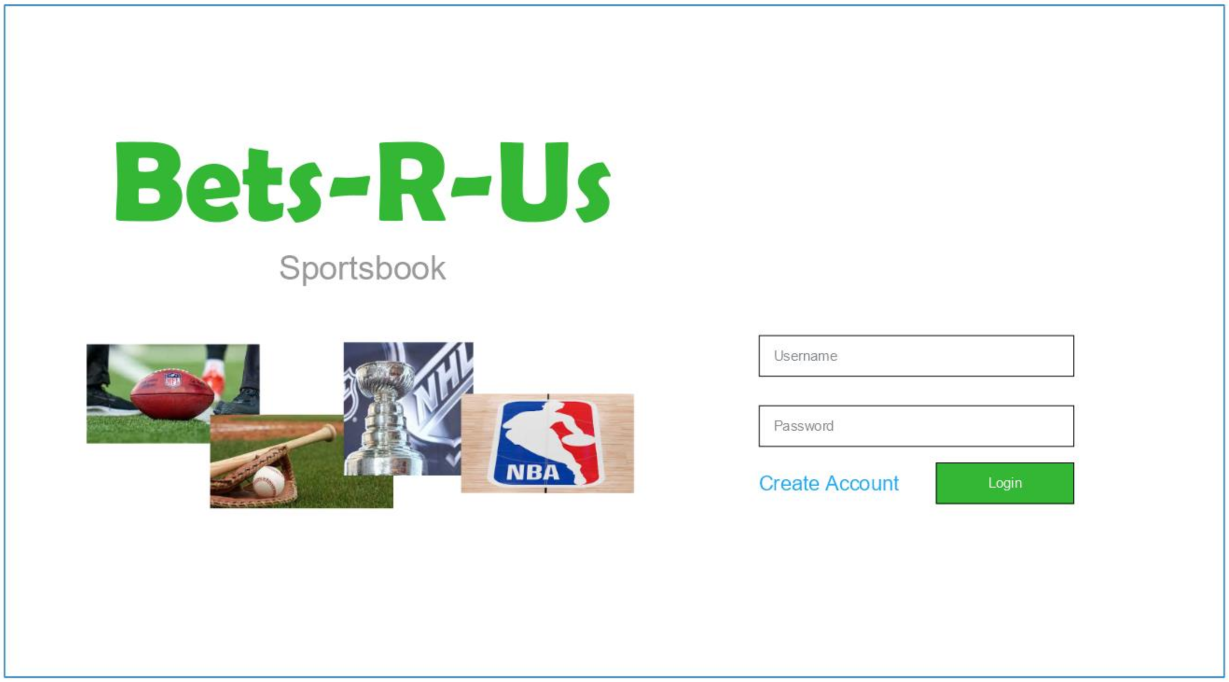
Sequence Diagram User Placing a Bet



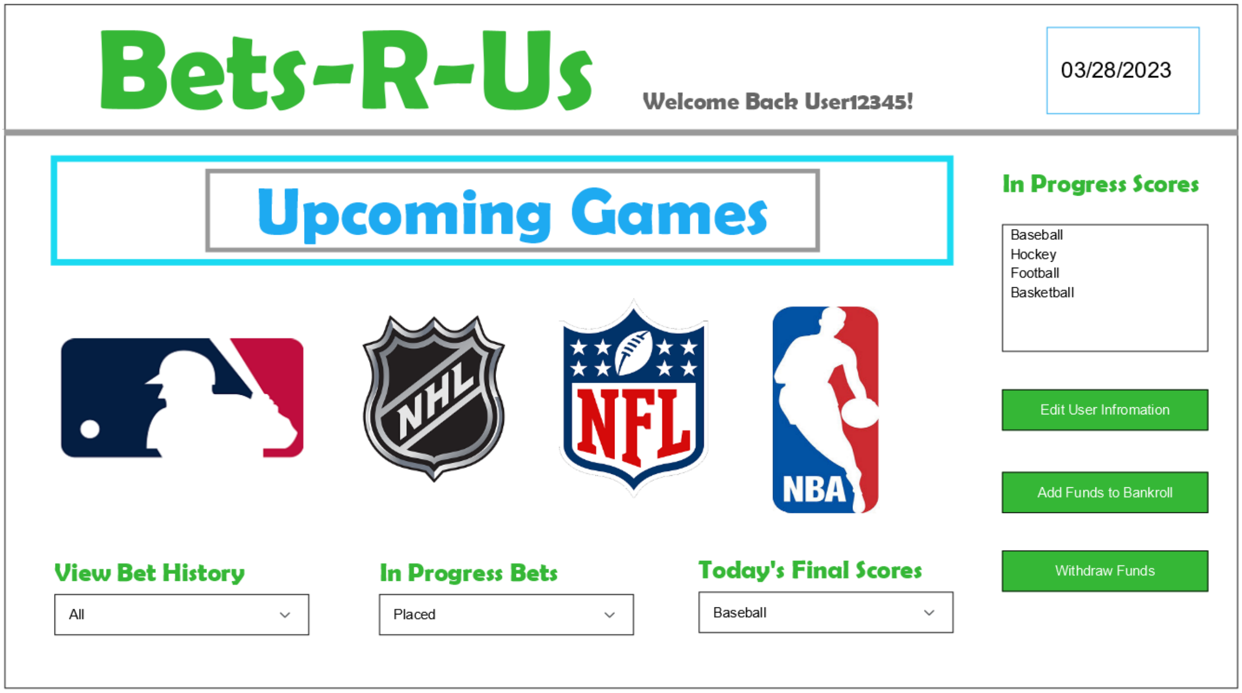
betInfo = String username, String eventId, String selection, double bet_amount, BetTypeEnum bet_type, SportsEnum sport

Wireframes

Landing Page/Login Screen



Home Page



03/28/2023

View Bet History: All

In Progress Bets: Placed

Today's Final Scores: Baseball

System Evolution

The system currently supports the four major professional sports leagues in the United States, in the future it is possible to add new sports to the enumeration of supported sports, for instance NCAA Basketball or Major League Soccer. Also currently supported are the three traditional bet types used in sportsbook betting, moneyline, spread, and over/under. As the program evolves it is possible to add what are known in the sportsbook industry as “player prop bets”. Player prop bets can be made on any athlete to reach certain stat goals. For example, a prop bet may be placed on a running back in the NFL to run for at least 100 yards, or a major league hitter to have at least 2 RBI in a game. These special bets require more detailed odds retrieval from Odds-API and add another layer of complexity in the betting controller. Another evolution that came to mind as the project was moving forward was to implement a Game cache. The Game objects are not persisted in the DB because once the game is played, if no bets were placed on the event, then it results in unnecessary additions to the DB and used disk space. However, it becomes complicated when the user is trying to select a game to place a bet on. Every time the user checks the schedule of upcoming games the OddsApi needs to be hit to get upcoming games and their odds which results in a lot of external API calls and resource overhead. For example if the schedule of games contains 5 games, then in order to get the List<Games> associated with the getAllUpcomingGames call in the Schedule controller, one external API call is made to get the event Id's of the 5 games and then 5 calls need to be made to get the individual odds of each game. The result is 6 total external calls every time the schedule is loaded, which may be several times in an hour leading up to an event. The implementation of a Game cache would allow the results of the get scheduled games call and the odds calls being cached and then subsequent calls could retrieve the information from the cache. Retrieval of the games and odds from a cache is not only faster but also eliminates the need to reach out to the API for every subsequent schedule check. When the events have ended they can be removed from the cache, most likely on a time basis such as every 24 hours, or they can be removed on a FIFO basis.