

EMULADDER



Daniel Frye, Gabriel Wilmoth, Mckenzie Moize, Rinty Chowdhury,
Robert Meyer



Table of Contents

Project Definition	3
Functional Requirements	3
Usability Requirements	3
System Requirements	4
Security Requirements	4
Project Specifications	4
Focus Domain	4
Libraries / Frameworks	4
Platform	5
Genre	5
System - Design Perspective	6
Top-Down Diagram	6
Class Diagram	7
Use-Case Diagram	8
Sequence Diagram	9
Data Flow Diagram	10
E-R Model	11
Wireframe Model	12-16
System - Analysis Perspective	17
Data Dictionary	17-19
Process Models	20
Subsystem Algorithm Analysis	20-21
Project Scrum Report	21
Product / Sprint Backlog	21
Burndown Chart	22

Subsystems	22
Account Management Subsystem - Rinty Chowdhury	22-25
Contest Management Subsystem - Mckenzie Moize	26-27
Scoring Subsystem - Daniel Frye	27-29
Database Subsystem - Robert Meyer	29-30
Data Collection Subsystem - Gabriel Wilmoth	31-32
Complete System	33
Final Product	33
User manual	33
Source code	33
Team Member descriptions	33-35

Project Definition

The application, emuLadder, is a Fantasy eSports game similar to ESPN's Fantasy Football, Hockey, etc. Users draft real-life eSports players from games such as League of Legends and compete against other users with their drafted players. Users earn points based on their drafted players stats per game. Fantasy games are very popular in sports such as football, soccer, hockey and many more, however, there is not a strong presence of Fantasy eSports. eSports is an ever-growing scene, and emuLadder seeks to provide a Fantasy experience to a community that not only enjoys eSports but also to those that enjoy video games in general. To achieve this, emuLadder looks to provide a competitive environment that keeps the user involved in both their Fantasy game and the eSports games their players are competing in.

Functional Requirements

- A. The application should be able to track eSports data in real-time.
- B. The user should be able to join game rooms by invitations.
- C. The user should be able to create an account using his/her information.
- D. Users should be able to send invitation codes to projected friends. The application should be able to see the user's contact information and social media in order to create new friends.
- E. The application should be able to track player progress and update it in real-time.
- F. The application will be able to handle pseudo-money transactions between users. The user can use that virtual money to add or exchange a good player to his/her team.
- G. Users should be able to host a game room.

Usability Requirements

- A. Game status will be displayed to keep the user updated about the game.
- B. The application will match with the real world so that the user can connect with it easily. It will make things easy for the user to play. All the game symbols and characters will be similar to real-world icons.
- C. Learning and performing the game should be easy for the new user. The game buttons and icons should be self-explanatory. The game will provide help to the new user when extra help is needed.
- D. Users will have some freedom to change the game settings so that the system will be more flexible and user-friendly but it will be limited. Users will be able to undo the job if they don't want to continue with it.

- E. The application will have maintenance options to prevent the system from error. It will also help the user to recognize, diagnose, and recover from bugs.
- F. The application will maintain the basic game standard and feature format.
- G. Users will be able to save and load their data.
- H. The user can customize and personalize the game based on their choice to make the game more fun and engaging.

System Requirements

- A. The system will be web-based, thus the application will be available on any machine with a web browser. Supporting the latest versions of Chrome, Firefox, Safari, Safari for iOS and Internet Explorer 9-11.
- B. Basic internet connectivity.

Security Requirements

- A. The system must maintain the confidentiality of all data that is classified as confidential.
 - a. Passwords
 - b. Emails
 - c. Credit Card Information
- B. The systems libraries/plugins must follow and ensure the confidentiality of login information as well as payment information.
 - a. Sign-up / Sign-in plugin
 - b. 3rd party payment plugin
- C. Two-factor authentication to meet standard security requirements.

Project Specifications

Focus Domain

- A. Our domain is online betting/drafting applications. Our focus is specifically to make an application for users interested in eSports just like the already booming Fantasy Sports applications. A specific target crowd being the exponentially growing crowd that watches eSports.

Libraries / Frameworks

- A. Java backend for connecting to database and auxiliary services.
 - a. SpringBoot Framework
 - b. IntelliJ Development Environment
- B. JavaScript auxiliary service for web scraping and API calls

- a. Node.js
 - b. NodeEclipse Development Environment
- C. MySQL for database services
 - a. AWS Relational Database Management
 - b. MySQL Workbench Development Environment
- D. JavaScript for functionality of web pages
 - a. Angular 8 / Typescript
 - b. Visual Studio Code Development Environment
- E. HTML5 / CSS3 for front end visualization and web design
 - a. Visual Studio Code Development Environment
- F. Amazon Web Services for website hosting
 - a. AWS Console Development Environment

Platform

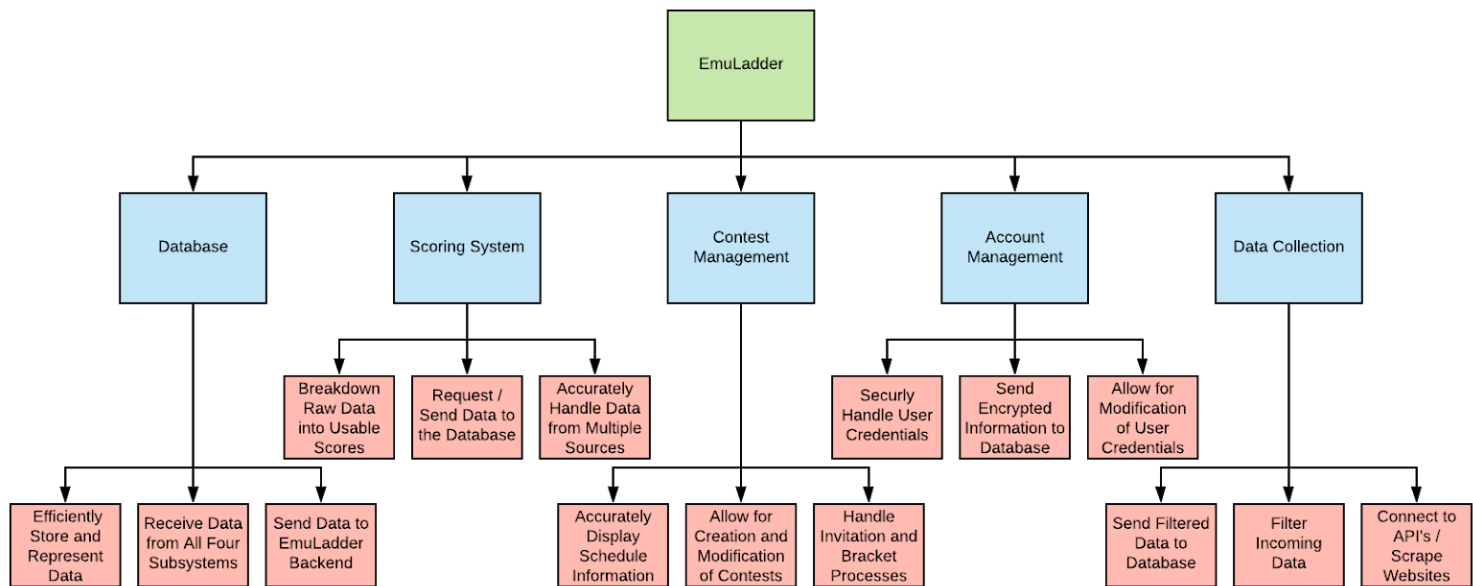
A web-based application for laptop, desktop, and mobile machines.

Genre

- A. Web application within the genre of Strategic Simulation/Sports Management

System - Design Perspective

Top-Down Diagram

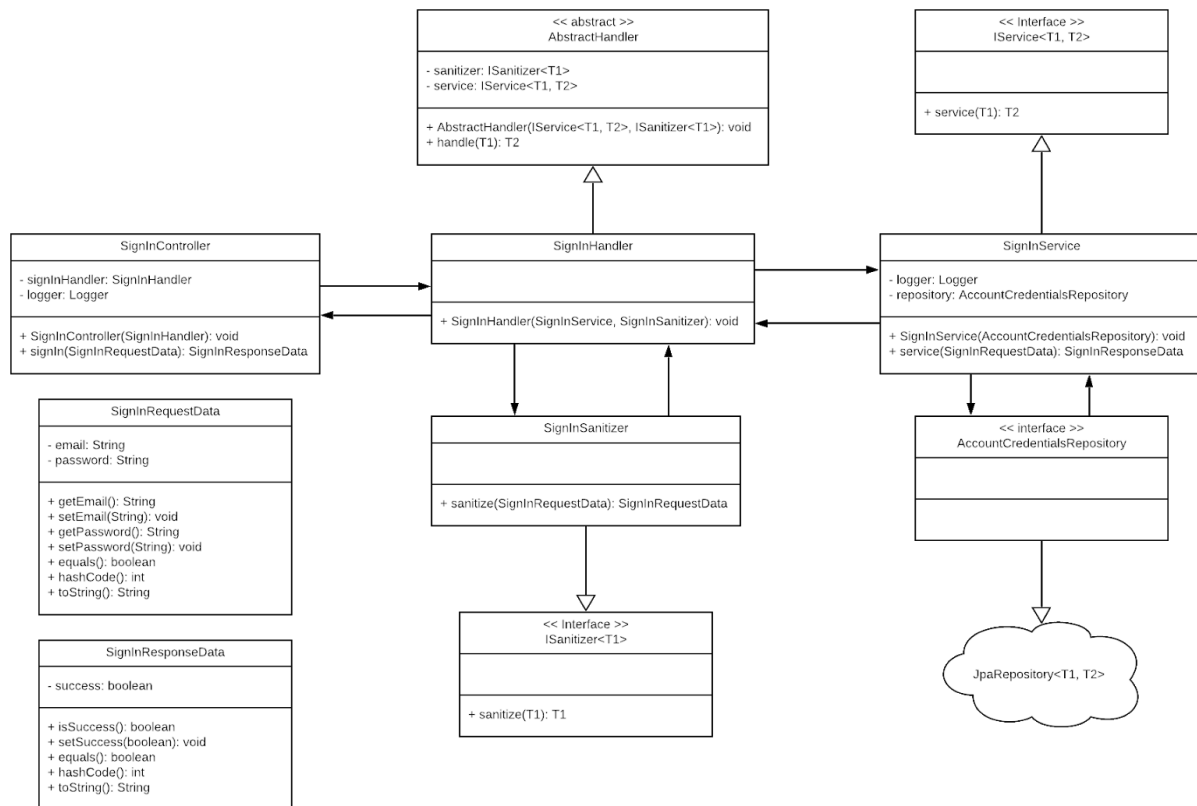


Top-Down Diagram:

The top-down diagram above, breaks down the project system into its lower manageable levels and their connections. In our subsystems: Database, Scoring System, Contest Management, Account Management, and Data Collection are divided into separate sub-processes that take place within each subsystem. Such as:

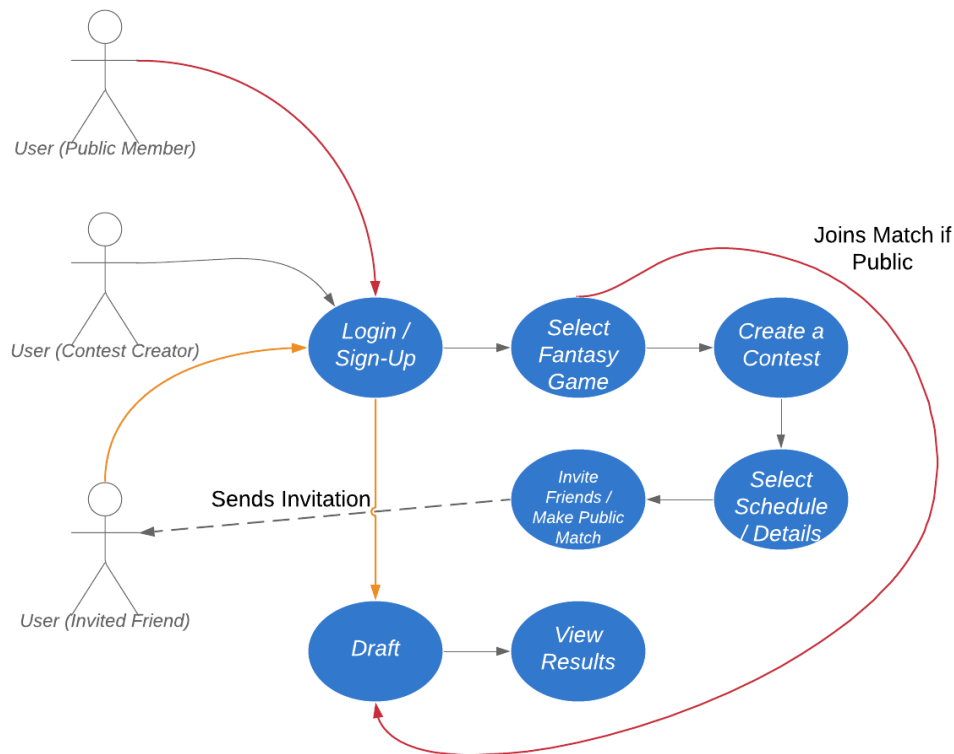
1. Database: Store and Represent Data, Receive Data from subsystems and Send Data to backend of the system
2. Scoring System: Process raw data ,Request and send Data to Database and Handles Data from multiple sources
3. Contest Management: Display Schedule information, Allow creation of and updates of Contests and Invitation and Bracket processes
4. Account Management: Security and user credential handling, Sends Encrypted information to Database, Allows Modification of user credentials.
5. Data Collection: Sends filtered data to Database, Filter incoming data and Connect with API(s)

Class Diagram



Class Diagram: Our backend service will utilize a MVC (Model View Controller) design pattern implemented using the SpringBoot framework in Java. This design pattern has the service exposed by a controller class (SignInController), which takes in as a parameter a request data object based on the mapped context path. The controller then passes the request data to the handler. The handler is responsible for passing the request data through a sanitizer to scrub for sanitize the input, then passing the request data along to the service. The service then performs the business logic on the request data and sends a new response data object back up through the controller and back to the initial caller. Above, I have shown how this process will work for the sign in process, but this will be used for each individual action such as editing an account, password reset, creating contests, getting contest ranking, and many, many more.

Use-Case Diagram



Use-Case Diagram:

In the above Use-Case Diagram we walk through the case in which a user Signs-up or logs into a system. Then said user goes through the actions of selecting a game and creating a contest for said game. The user is then able to select a scheduled event and invite friends to the event. After the user has invited friends they will then be able to draft a team. The User (Invited Friend) goes through a similar process as in signing up for the website and then goes directly to drafting a team in the contest.

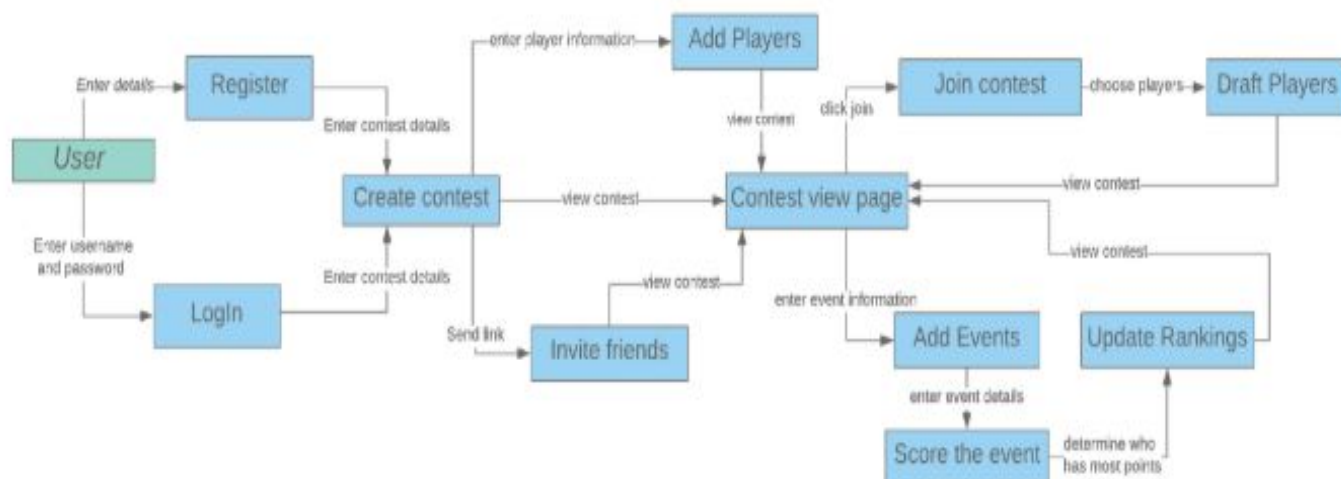
Sequence Diagram



Sequence Diagram:

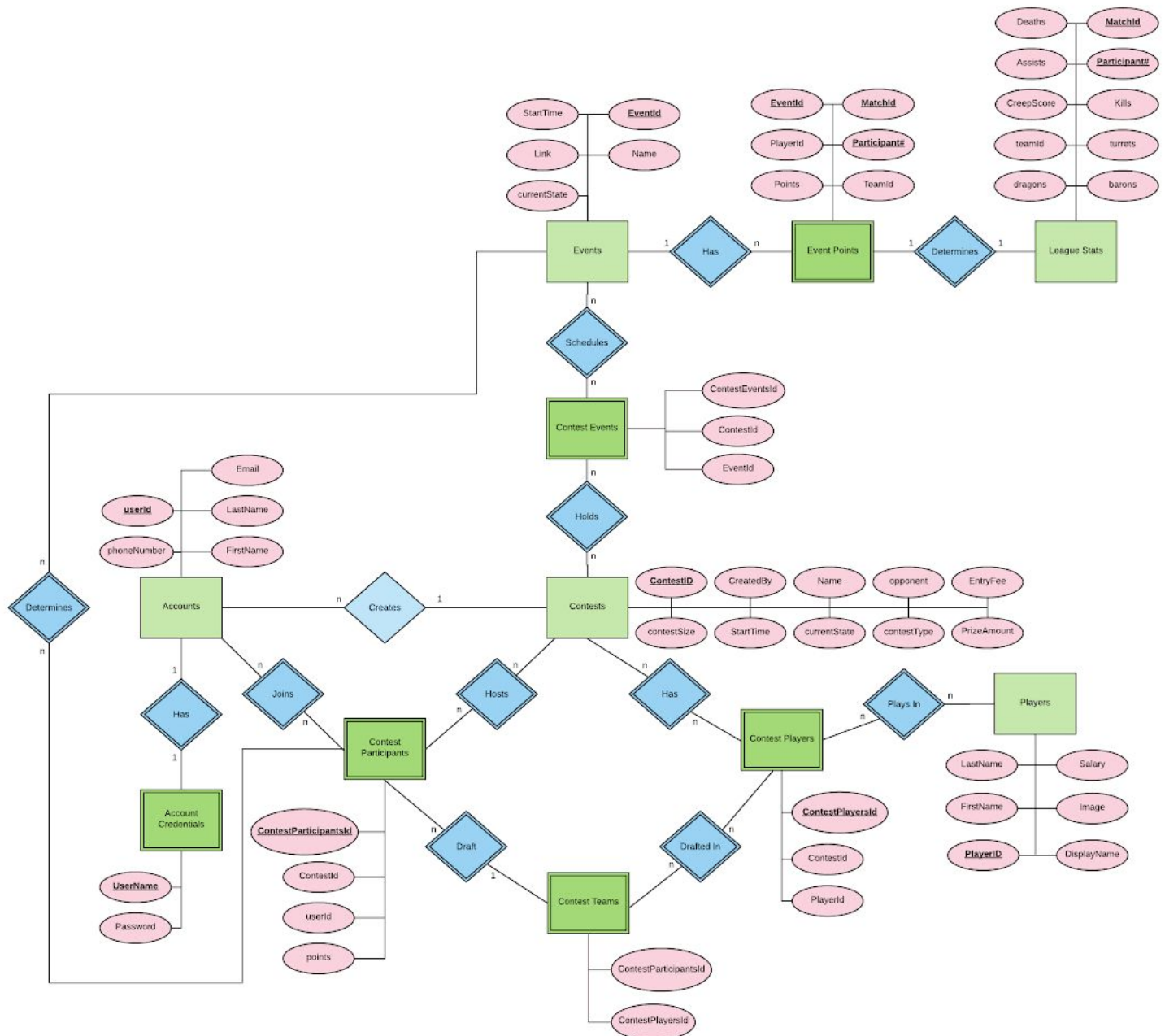
The above Sequence Diagram follows the interaction of a user creating a contest and all the behind the scene calls that will be made. It starts out with the user clicking “create a contest” on the UI and then said click calls a series of events which track all the way to the database which inevitably return the schedule for said event. After the schedule has been returned to the UI the user proceeds with a details page where they set the details for the contest and then to an invitation page where they can invite friends. Once the switch to the invite page occurs there is a call to the database again in which we load all of the users friends for an invite and that is returned to the UI. The user then selects whom they want to invite and click “finish” which will set out an invitation email as well as persist the contest information to the database.

Data Flow Diagram



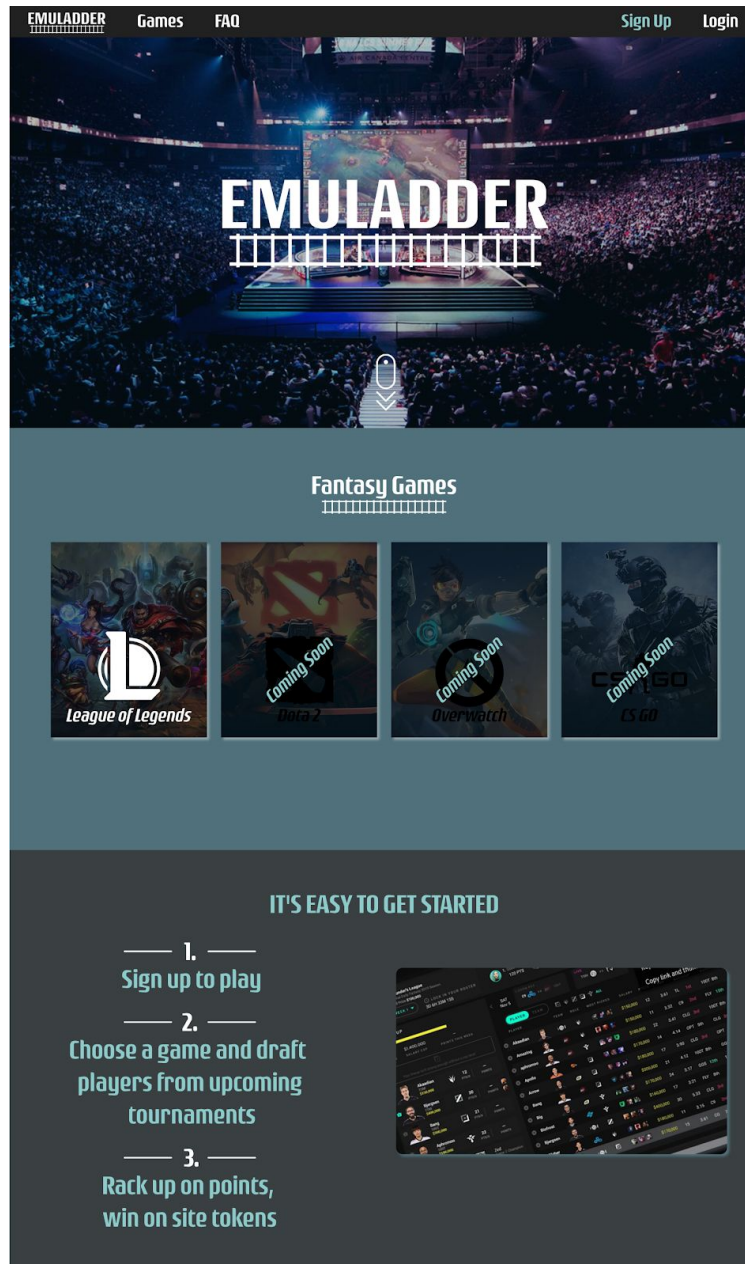
Data Flow Diagram: The data flow diagram demonstrates how the data is being used by our service. For instance, when a user enters their username and password, it is being used for them to login. The diagram details the type of data that is being received and where that data is being sent. It uses contest information to create contests and event information to create events. It uses data from the player list to draft players. It takes the event information to get the current score of the event then it updates the ranking of the player. In the contest view page, it displays all the updated information about the contest. It also sends links to invite friends.

E-R Model

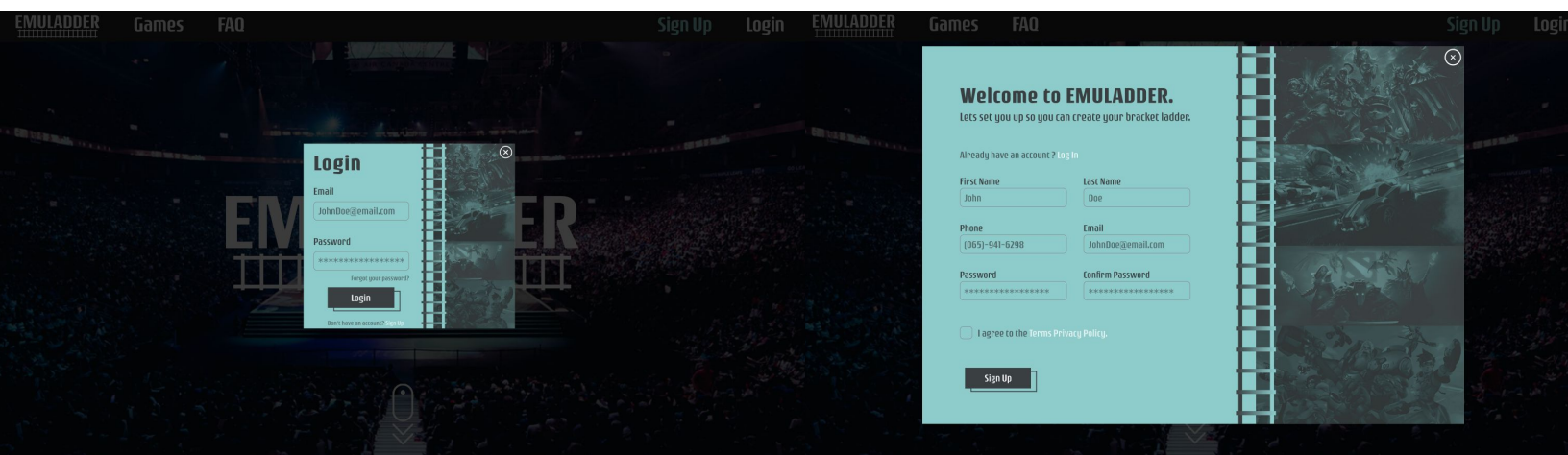


E-R Model: The E-R model defines the entities within the database, as well as the relationship between them. The weak entities and relationships are specified with double lined borders. The multiplicities of the relationship are defined on the lines connecting the entities to their relationships.

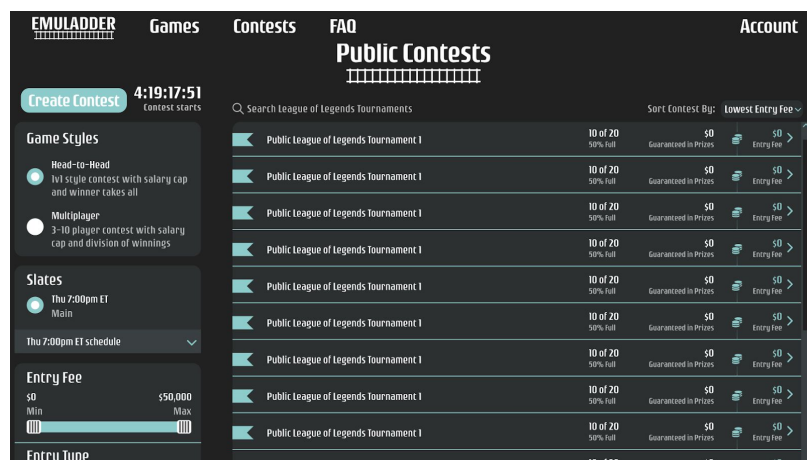
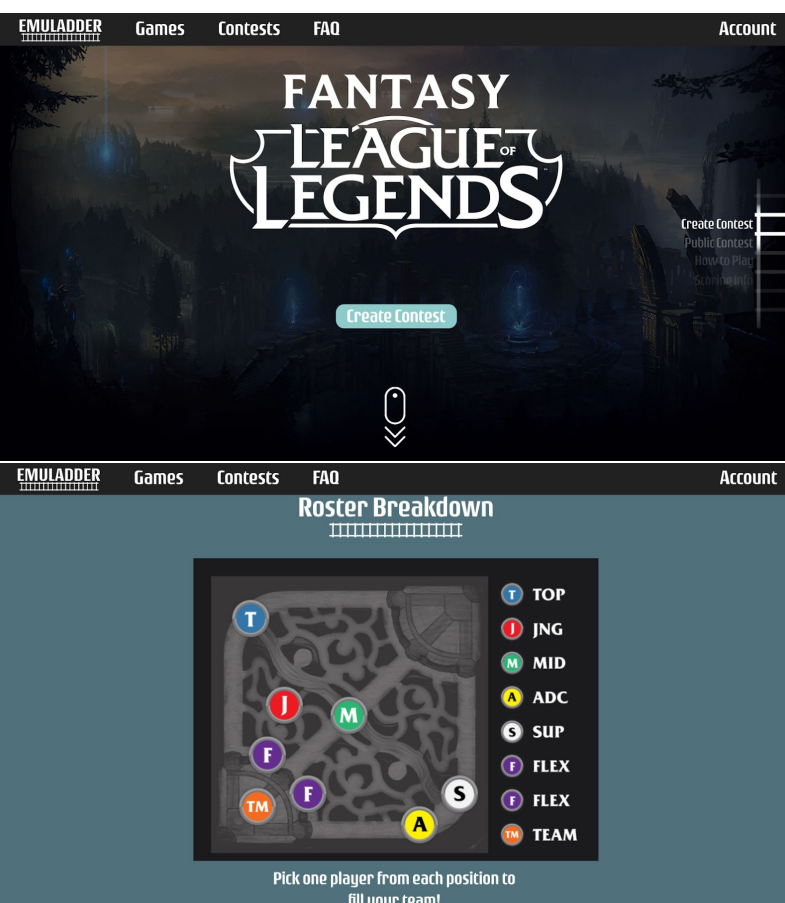
Wireframe Model



Above is the landing page of the web application which gives a general overview of what it takes to get started and pathing to the rest of the application.



Above the have the Login and Sign Up Screens the user can access by clicking either Login or Sign Up on the home screen of the application.



The above is the main contest landing page for League of Legends which allows the user to see public contests and information about how to play.

LOL Classic



In salary cap contests, participants will create a lineup by selecting players listed in the Player Pool. Each player listed has an assigned salary and a valid lineup must not exceed the salary cap of \$50,000.

Contest results will be determined by the total points accumulated by each individual lineup entry (scoring rules summarized below).

Participation in each contest must be made only as specified in the Terms of Use. Failure to comply with these Terms of Use will result in disqualification and, if applicable, prize forfeiture.

Scoring

The player you draft as your captain will earn 1.5x the standard fantasy point value for each statistic.

Players		Teams	
Statistic	Fantasy Points *Players drafted as Captain earn 1.5x fantasy point values	Statistic	Fantasy Points *Players drafted as Captain earn 1.5x fantasy point values
Kills	+3 Pts	Turrets	+1 Pt
Assists	+2 Pts	Dragons	+2 Pts
Deaths	-1 Pt	Barons	+3 Pts
Creep Score	+0.02 Pts	First Blood	+2 Pts

On the main contest page we link to an in depth breakdown of the scoring and rules for said contest if the user wants to know absolutely everything behind the scene. This document fills up a few pages and above is just the heading screen capture.

EMULADDER

Games

Contests

FAQ

Account

FANTASY LEAGUE LEGENDS

Choose a Tournament

Calendar icon

Saturday
Jan 25th
5:00pm
LCS

Calendar icon

Saturday
Jan 25th
5:00pm
LCS

Calendar icon

Saturday
Jan 25th
5:00pm
LCS

Calendar icon

Saturday
Jan 25th
5:00pm
LCS

Calendar icon

Saturday
Jan 25th
5:00pm
LCS

Games

FLY v IMT
1/25 5:00pm

C9 v TSM
1/25 5:50pm

DIG v CLG
1/25 6:40pm

GG v 100
1/25 7:30pm

Contest Type

Head-to-Head

Multiplayer

Opponent

Public

Private

Entry Fee

Free ▼

Name your Contest

Public League of Legends Tournament 1

Select Team

Above is the first step in contest creation after the “Create Contest” button is clicked. The user is presented with a schedule of events and controls for how the contest should be managed.

EMULADDER Games Contests FAQ Account

FANTASY LEAGUE LEGENDS **Public League of Legends Tournament 1** 4:19:17:51
Contest starts Thu 7:00pm ET

Head-to-Head Contest Type 0/2 Entries \$0 Entry Fee \$0 Prizes One entry per person Rules & Scoring

All Games FLY v IMT 1/25 5:00pm C9 v TSM 1/25 5:50pm DIG v CLG 1/25 6:40pm GG v 100 1/25 7:30pm

Available Players

All	TOP	JNG	ADC	SUP	TEAM	All Teams Plays for	Find a Player
Name	Game	FPPG	Rank	Salary			
Bjergsen SOREN BJERG	Mid TSM v C9 1/25 5:50pm	23.56 FPPG	1st Rank	\$14,000 Salary			
Bjergsen SOREN BJERG	Mid TSM v C9 1/25 5:50pm	23.56 FPPG	1st Rank	\$14,000 Salary			
Bjergsen SOREN BJERG	Mid TSM v C9 1/25 5:50pm	23.56 FPPG	1st Rank	\$14,000 Salary			
Bjergsen SOREN BJERG	Mid TSM v C9 1/25 5:50pm	23.56 FPPG	1st Rank	\$14,000 Salary			
Bjergsen SOREN BJERG	Mid TSM v C9 1/25 5:50pm	23.56 FPPG	1st Rank	\$14,000 Salary			
Bjergsen SOREN BJERG	Mid TSM v C9 1/25 5:50pm	23.56 FPPG	1st Rank	\$14,000 Salary			

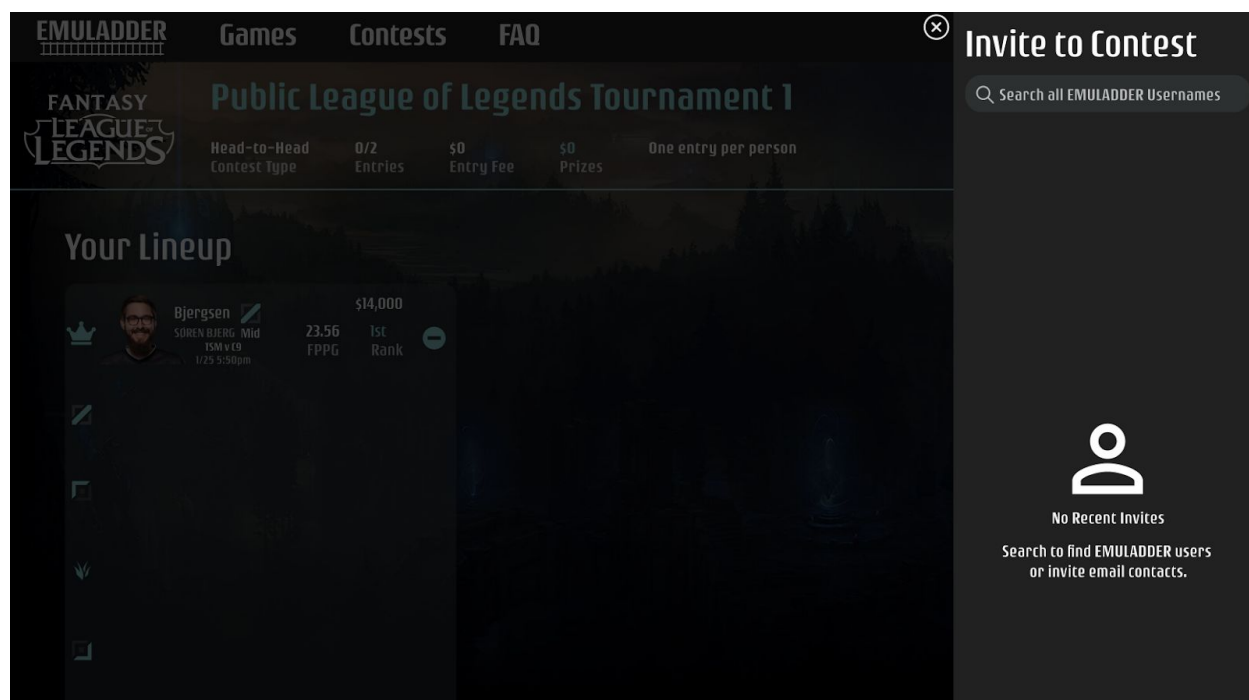
Your Lineup \$36,000 Salary Remaining | \$14,000 AVG/Player

Bjergsen
SOREN BJERG Mid
TSM v C9
1/25 5:50pm

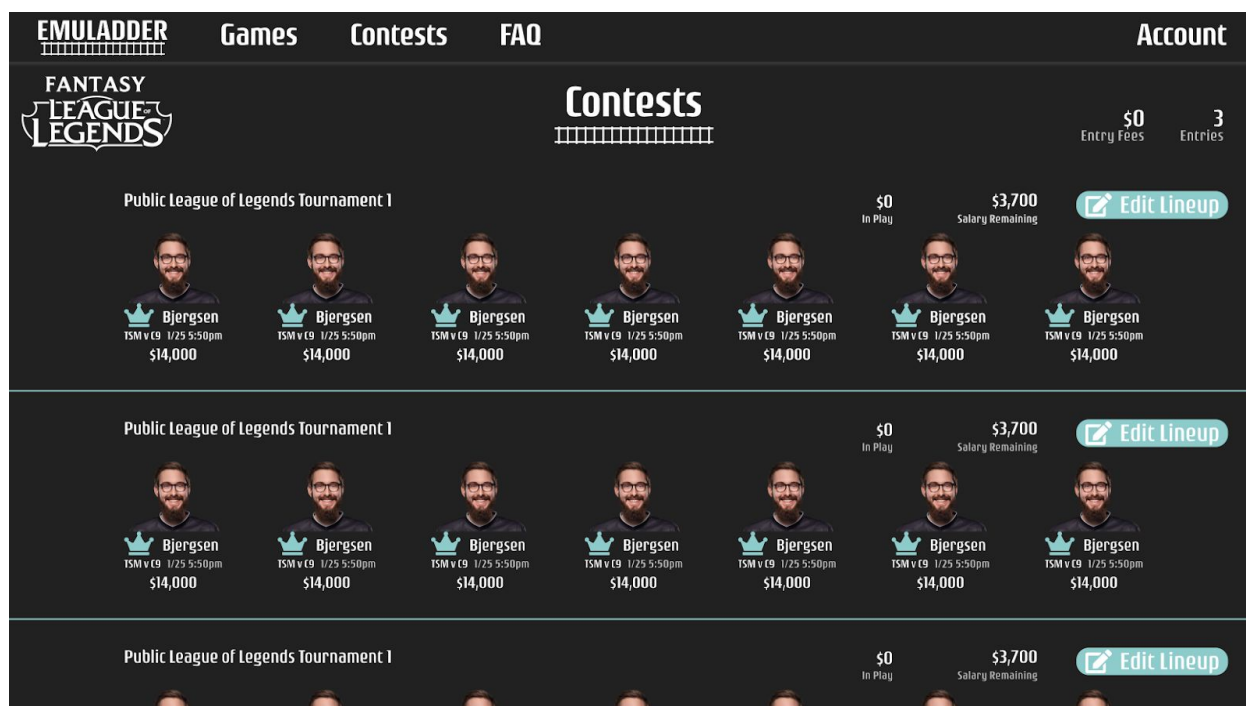
23.56 FPPG 1st Rank

Enter Lineup

Once the user clicks “Select Team” they are sent to the drafting phase of the application in which the user picks their team via a plethora of sorting / filter options. The goal here is to draft the best team while staying under the salary cap.



Once the user enters their lineup they can then proceed to the invite page in which they can then add their friends to the contest.



The contest page can be accessed from anywhere within the application as long as the user is logged in and will display the current contests the user has going on.

System - Analysis Perspective

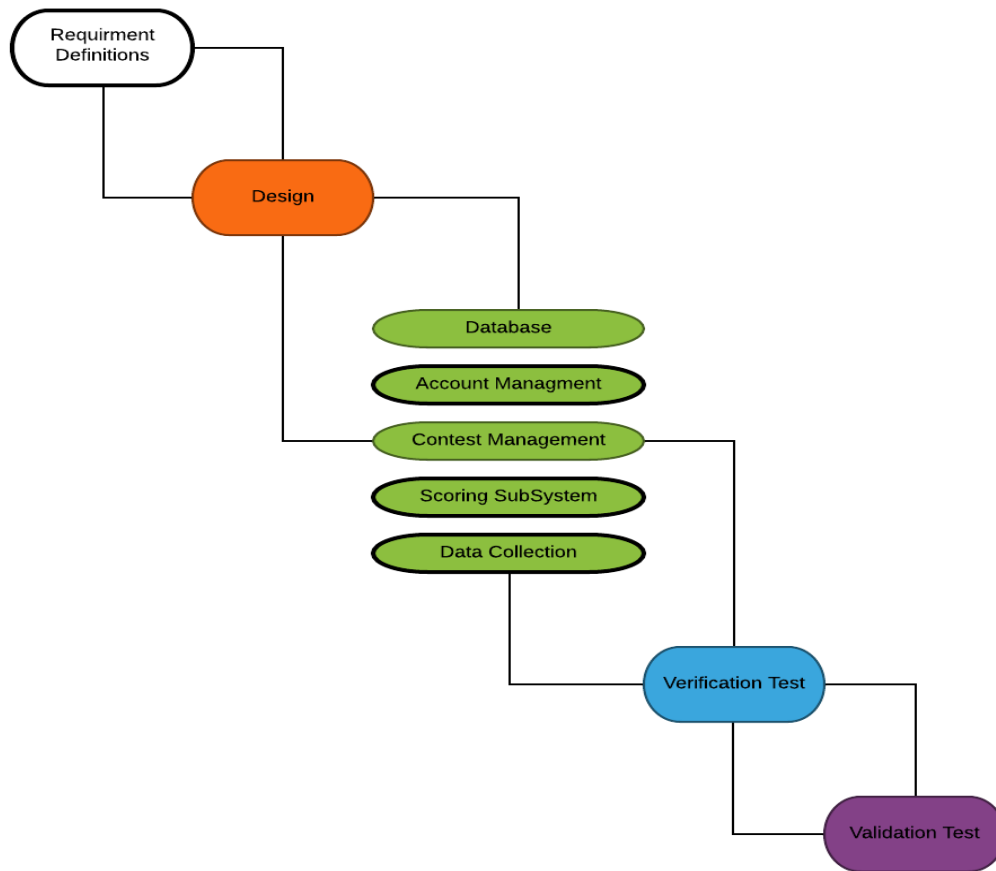
Data Dictionary

Table	Column	Type	Description	References	PK
Account	FirstName	Text	the first name of the user		
Account	LastName	Text	the last name of the user		
Account	UserName	Text	the unique ID of the user		Y
Account	Email	Text	the email used for communication with the user		
Account Credentials	User Name	Text	the unique ID of the user	Account (UserName)	Y
Account Credentials	Password	Text	the password used for user sign in		
Player	PlayerId	Integer	the unique ID of the player		Y
Player	FirstName	Text	the first name of the player		
Player	LastName	Text	the last name of the player		
Player	DisplayName	Text	the "gamertag" or nickname the player goes by		
Player	Image	Text	the name of the player profile image		
Player	Salary	Integer	the amount the player costs to be drafted		
Contest	ContestId	Integer	the unique ID of the contest		Y
Contest	CreatedBy	Text	the user name of the user who created the contest	Account (UserName)	
Contest	StartTime	Date / Time	the day and time the contest starts		
Contest	Name	Text	The name of the contest		

Contest	State	Integer	whether the contest has not started (1), in progress (2), or finished (3)		
Contest	ContestSize	Integer	The amount of players allowed to join the contest		
Contest	ContestType	Integer	If the Contest is created as Head-To-Head(1) or Multiplayer(0)		
Contest	Opponent	Integer	If the Contest is created as Public(1) or Private(0)		
Contest	EntryFee	Double	The set entreeFee for the contest		
Contest	PrizeAmount	Double	The prize winnings for the contest		
Contest Participants	Contest ParticipantId	Integer	the unique ID for the combination of contest and user		Y
Contest Participants	ContestId	Integer	the unique ID of the contest	Contest (ContestId)	
Contest Participants	UserName	Text	the unique ID of the user	Account (UserName)	
Contest Players	Contest PlayerId	Integer	the unique ID for the combination of contest and player		Y
Contest Players	ContestId	Integer	the unique ID of the contest	Contest (ContestId)	
Contest Players	PlayerId	Integer	the unique ID of the player	Player (PlayerId)	
Contest Rankings	Contest ParticipantId	Integer	the unique ID for the combination of contest and user	ContestParticipants (ContestParticipantId)	Y
Contest Rankings	Points	Integer	the number of points that determines the ranking of the users		
Contest Teams	Contest ParticipantId	Integer	the unique ID for the combination of contest and user	ContestParticipants (ContestParticipantId)	

Contest Teams	Contest PlayerId	Integer	the unique ID for the combination of contest and player	ContestPlayers (ContestPlayerId)	
Contest Events	ContestId	Integer	the unique ID of the contest	Contest (ContestId)	
Contest Events	EventId	Integer	the unique ID of the event	Events (EventId)	
Event	EventId	Integer	the unique ID of the event		Y
Event	Name	Text	the name of the event		
Event	StartTime	Date / Time	the day and time the event starts		
Event	Link	Text	the url where the event can be viewed		
Event	State	Integer	whether the event has not started (1), in progress (2), or finished (3)		
Event Points	EventId	Integer	the unique ID of the event	Events (EventId)	
Event Points	PlayerId	Integer	the unique ID of the player	Player (PlayerId)	
Event Points	Points	Integer	the number of points the player achieved during the event		
League Stats	PlayerId	Integer	the unique ID of the player	Player (PlayerId)	
League Stats	Kills	Integer	the number of kills the player achieved		
League Stats	Deaths	Integer	the number of deaths the player achieved		
League Stats	Assists	Integer	the number of assists the player achieved		
League Stats	CreepScore	Integer	the creepscore the player achieved		

Process Model:



Process model:

The above model describes the current plan of how the system stages will be developed. In this model, we are showing that the subsystems will be developed in tandem.

System and Subsystem Algorithm Analysis

Subsystems

Scoring $O(n)$: Operations include adding all points up together for scores and adding points to ranking. Both operations are $O(n)$, making the time complexity of the scoring subsystem $O(n)$.

Contest Management $O(n)$: Operations at $O(1)$ time complexity include creating contests, modifying contests and inviting friends to contests. Operations at $O(n)$ time complexity include adding players to a contest and drafting players. The combination of the operations gives the contest management subsystem a time complexity of $O(n)$.

Account Management $O(1)$: Operations include creating an account, logging into an account, editing an account, signing up for contests and account recovery. All operations in this subsystem are $O(1)$ time complexity, giving the subsystem as a whole a time complexity of $O(1)$.

Data Collection $O(n)$: Operations include retrieving JSON from API, which runs at $O(1)$ time complexity, and converting JSON to database object notation, which runs at $O(n)$ time complexity. This gives the data collection subsystem a time complexity of $O(n)$.

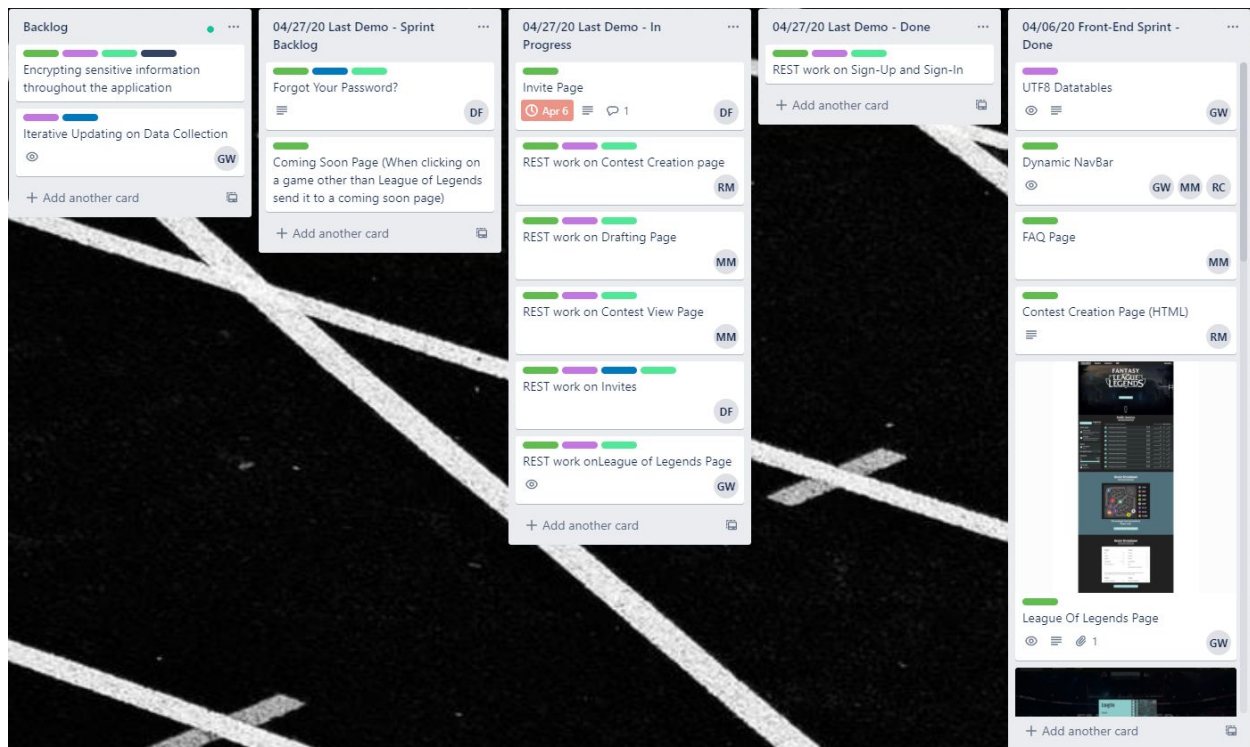
Database $O(n)$: Operations at $O(1)$ time complexity include creating, inserting, deleting, and getting information from the database. Operations at $O(n)$ include updating the table for all users. The combination of these operations gives the database subsystem a time complexity of $O(n)$.

System

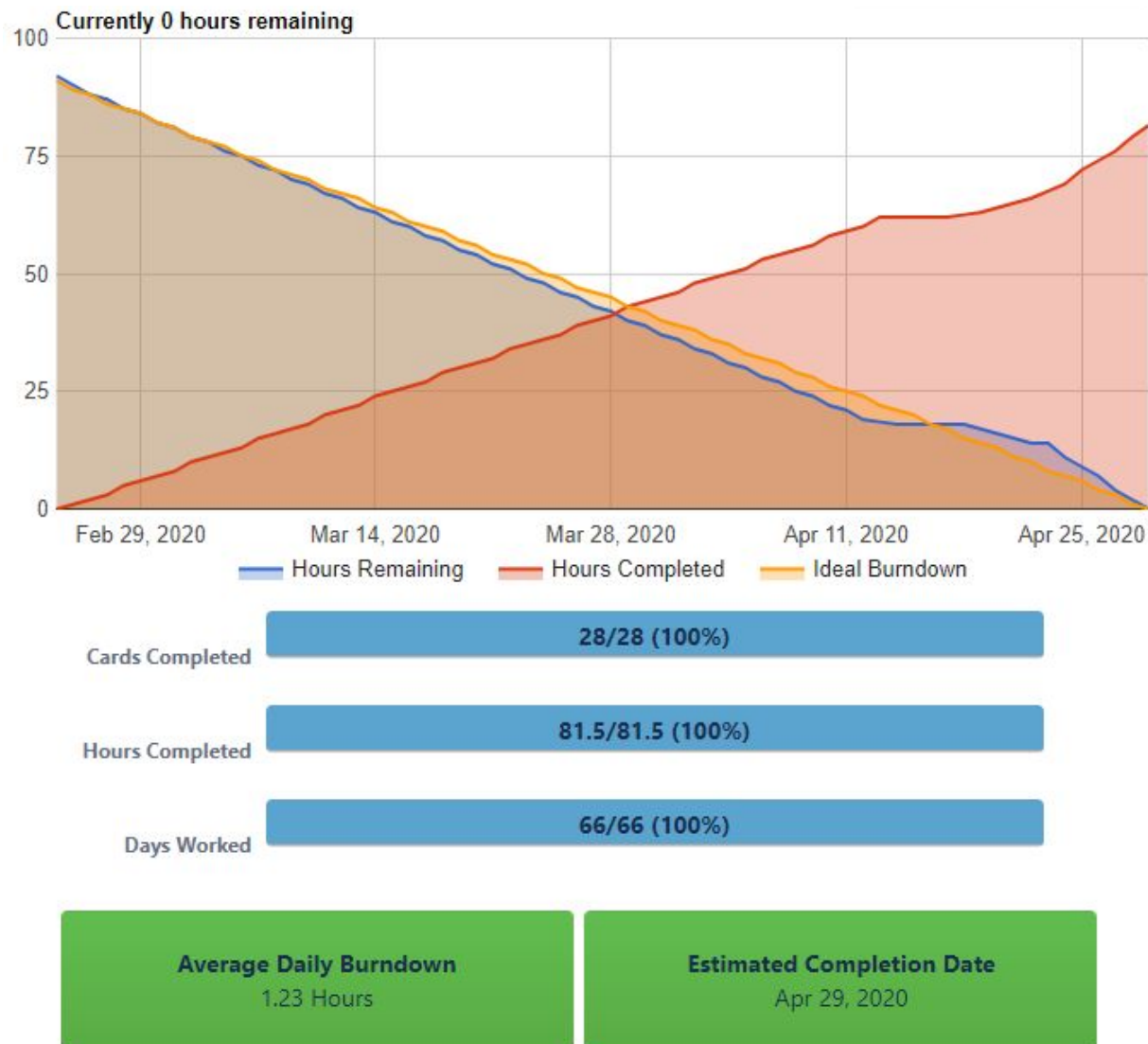
emuLadder $O(n)$: Based on the time complexity of all subsystems for the emuLadder system and the combination of these subsystems, the time complexity for emuLadder will be $O(n)$.

Project Scrum Report

Product / Sprint Backlog:



Burndown Chart:



Subsystems

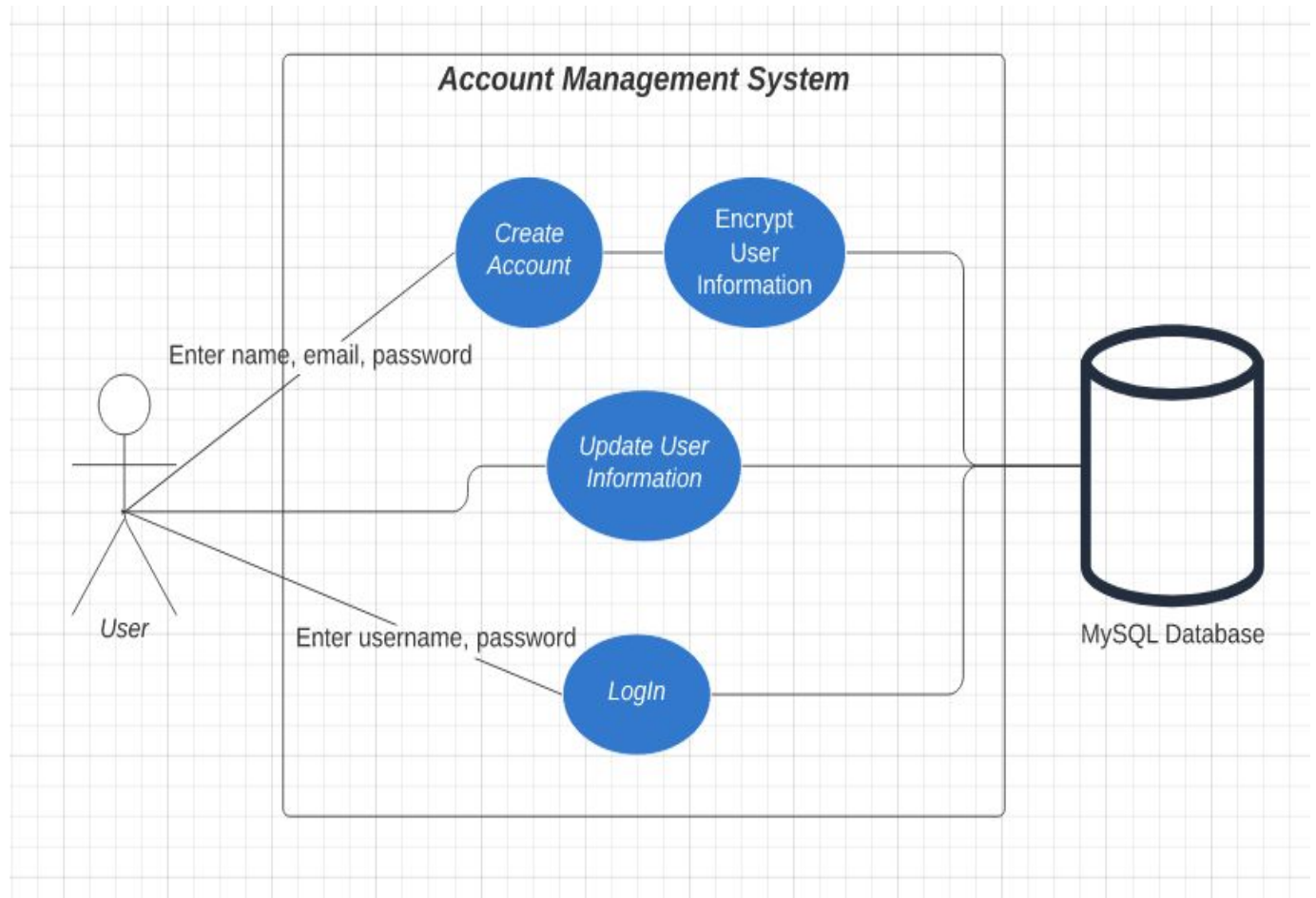
Account Management Subsystem - Rinty Chowdhury:

Description:

Account management subsystem has three parts. They are security, send user information to the database, and allow modification of credentials. Security handles user credentials securely. It makes sure all the user information is encrypted. So that if the system gets hacked, no one gets the user information. Then the second part comes. It sends those encrypted user information to the database. The database stores all the information so that the system can utilize that information. The third part is modification of the user credentials. Some features like forgot password and username, change

password and username will need a way to modify user credentials once the account is already created. Account management system will let the user make the modification and then update the information in the database.

Use case diagram of the subsystem:



Data dictionary:

Account management subsystem connected with two tables in the data database. First one is an Account table which includes columns named FirstName, LastName, UserName, Email. Second one is an Account Credentials table which includes columns named UserName and Password. [Link](#) to predefined Data Dictionary within this document.

Scrum Backlog:

[Link](#) to predefined Scrum Backlog within this document.

Coding:

- HTML5 and CSS3 for web design and visualization.
- Visual Studio Code development environment.

User manual:

- **SignUp**
 - If the user already has an account then they can click on the 'Log In' button and it will take them to the login page.
 - Users will enter their first name on the 'First Name' field and last name on the 'Last Name' field.
 - Users will enter their phone number on the 'Phone' field. The format is shown in the text box. Ex: (000)-000-0000.
 - Users will enter their email address on the 'Email' field which they have access on.
 - Users will create and enter their password on the 'Password' field using standard password requirements.
 - Users will re-enter their password on the 'Confirm Password' field.
 - Users will click on 'I agree to the Terms Privacy Policy' checkbox.
 - Users will click on the 'Sign Up' button and it will take them to EmuLadder homepage.

Welcome to EMULADDER.
Let's set you up so you can create your bracket ladder.

Already have an account? [Log In](#)

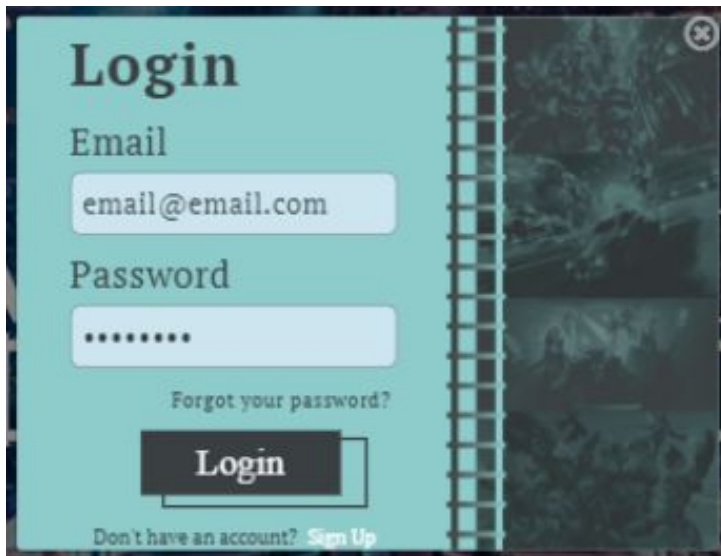
First Name <input type="text" value="John"/>	Last Name <input type="text" value="Doe"/>
Phone <input type="text" value="(065)-941-6298"/>	Email <input type="text" value="JohnDoe@email.com"/>
Password <input type="password" value="*****"/>	Confirm Password <input type="password" value="*****"/>

☐ I agree to the [Terms Privacy Policy](#).

Sign Up

- Login

- Users will enter their email address on the 'Email' field. The same email they used when they signed up.
- Users will enter their password on the 'Password' field. The same password they created when signed up.
- Users will click on the 'Login' button and it will take them to the EmuLadder homepage.
- If the users forget their password then they can click on the 'Forgot your password?' button and it will send them an email to reset their password. The email will be sent to the email address that they used when signed up.
- If the users don't have an account then they can click on the 'Sign Up' button and it will take them to the signup page.



Testing:

- End to end testing to check if SingUp and LogIn functions work properly. Also tested to check if the data gets encrypted and stored into the database properly.

Contest Management Subsystem - Mckenzie Moize:

Description:

The Contest Management subsystem allows users to create, join, and track their existing contests. This subsystem provides users with a page to create a contest through the selection of multiple options such as contest type, entree fee, prize amount, etc. This subsystem also provides a means of allowing users to join existing contests, as well as inviting others to join your contests via email.

Initial Design and Model:

All initial designs and models can be found [here](#).

Data Dictionary:

The data dictionary can be found [here](#). The key items in the dictionary are the Contests, ContestPlayers, ContestParticipants, and ContestEvents tables.

Refinement:

- N/A

Scrum Backlog:

The backlog can be found [here](#). The key backlog items included contest creation page (UI), contest creation (backend), contest creation (database), drafting page (UI), drafting (backend), and drafting (database).

Coding:

- Object oriented programming
- TypeScript / JavaScript (for frontend services)
- Angular 8 / CSS / HTML (for frontend UI work)
- Java (for backend services)
- Spring Framework (for configuring REST client)
- JPA Framework (for connection to database)
- MySQL (for database queries)

User Training:

- Creating a contest
 - User will select a date for their contest via the choices at the top of the screen
 - User will select an entry fee to determine the price to enter the contest

- User will select which type of contest they want: Head-to-Head (where they battle one other person in a private, 1v1 match), Public (where they create a contest anyone can join with a limit of 10 people), and Private (where they create a private contest that can only be joined via a link with a limit of 5 people).
- User will enter a name for the contest
- User will click select team to finalize the creation of the contest
- Drafting a team
 - All of the details of the contest will be listed at the top of the page
 - The user can see the amount of salary that they have remaining to draft with above the right column
 - The user will then draft players for each position using the amount of money they are given by clicking the (+) button
 - Once done, they will click the save button
- Viewing your contests
 - The user will be able to view a list of contests they have signed up
 - From the list, they can click “edit lineup” to change the players they have drafted
 - From the list, they can click “invite” to invite players to join the contest

Testing:

- Unit testing (for backend services)
- Integration testing (when combining backend, frontend, and UI to work together)
- Regression testing (when promoting new features)
- End-to-end testing (to ensure subsystem as a whole functions correctly)

Scoring Subsystem - Daniel Frye:

Description:

The Scoring Subsystem uses game rules to allocate points, add points, and ranks based on total points earned in the current contest. Currently the contest scoring is based off of the League of Legends esports score card.

Players		Teams	
Kill	+3	Turret	+1
Assist	+2	Dragon	+2
Death	-1	Baron	+3
Creep Kill	+0.02	First Blood	+2
10+ K / A Bonus	+2	Win	+2
		Win under 30 minutes	+2

Some contests may include 'best-of' series matchups. Players and teams will accumulate bonus points for winning a 'best-of' series early

Players		Teams	
Game not played	+20	Game not played	+15

This chart gives the breakdown of how scoring is based on.

Subsystem design:

can be seen in the [E-R Model](#) on page 10.

Data Dictionary:

- Event Points EventId: the unique ID of the event Events (EventId)
- Event Points PlayerId: the unique ID of the player Player (PlayerId)
- Event Points Points: the number of points the player achieved during the event
- League Stats PlayerId: the unique ID of the player Player (PlayerId)
- League Stats Kills: the number of kills the player achieved
- League Stats Deaths: the number of deaths the player achieved
- League Stats Assists: the number of assists the player achieved
- League Stats CreepScore: the creepscore the player achieved
- Contest Rankings Contest ParticipantId: the unique ID for the combination of contest and user ContestParticipants (ContestParticipantId)

- Contest Rankings Points: the number of points that determines the ranking of the users

Scrum Backlog:

[Product / Sprint Backlog:](#)

Coding:

- HTML, CSS, TypeScript

Database Subsystem - Robert Meyer:

Description:

The Database Subsystem stores all relevant data that is used in the system as a whole. Using a REST service, new data can be created and stored in the database, or existing data can be retrieved from the database. This data includes data from the Data Collection Subsystem, Account Information, Contest Information, etc.

Subsystem Design:

- The ER Model can be found in the link: [ER Model](#)
- The ER Model is also found on page 10 of this document: [E-R Model](#)

Data Dictionary:

- AccountCredentials: Stores users' sign-in information.
- Accounts: Stores users' personal information.
- ContestEvents: Links contests to multiple events.
- ContestParticipants: Links contests to multiple users.
- ContestPlayers: Links contests to the eSports players that were drafted into a contest.
- Contests: Stores data about each contest.
- EventPoints: Links an eSports player's points earned to the Event they participated in.
- Events: Stores data about each event.
- LeagueStats: Stores statistics for each eSports player from an event, such as kills, etc.
- Players: Stores all necessary information about an eSports Player.

[Link](#) to predefined Data Dictionary within this document.

Scrum Backlog:

[Link](#) to predefined Scrum Backlog within this document.

Coding:

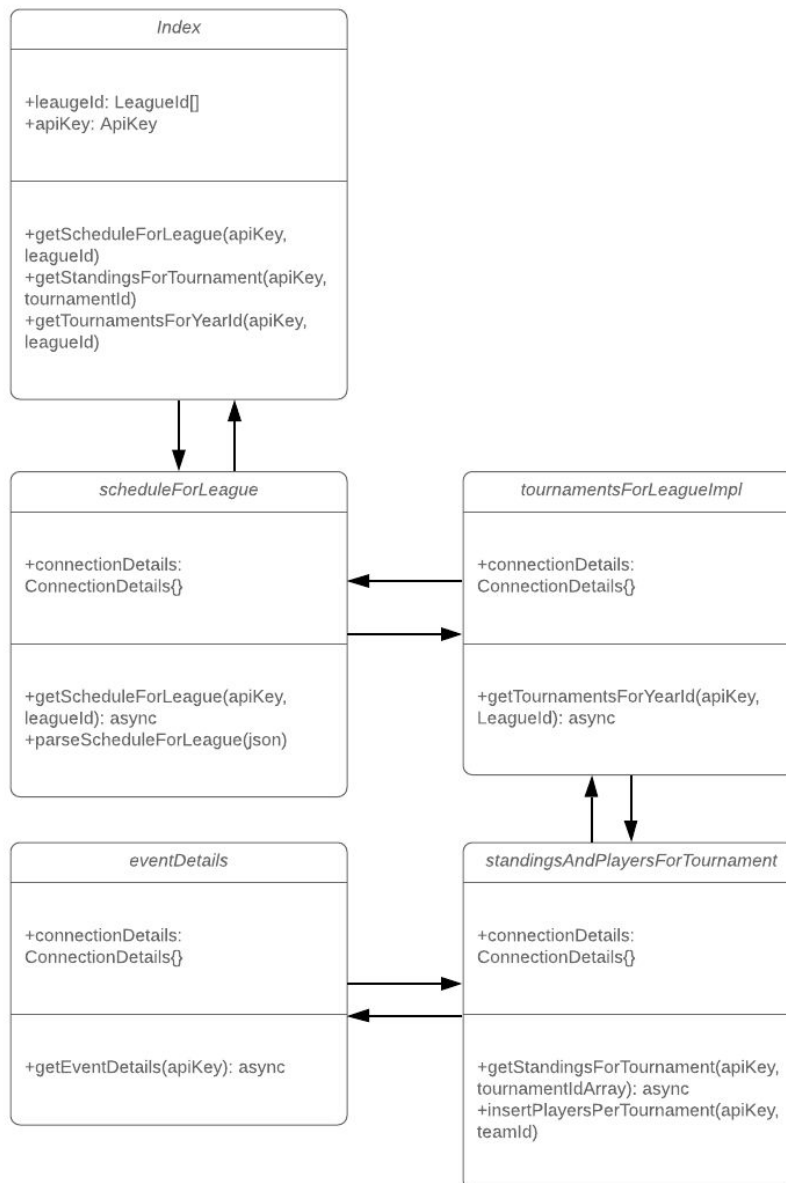
- MySQL
- Java

Data Collection Subsystem - Gabriel Wilmoth:

Description:

The Data Collection Subsystem encompasses a REST service which pulls down data from the RIOT Games League of Legends eSports API. This service provides all relevant data that is needed to fulfill requests from the whole EmuLadder ecosystem.

Subsystem Design:



Data Dictionary:

[Link](#) to predefined Data Dictionary within this document.

Scrum Backlog:

[Link](#) to predefined Scrum Backlog within this document.

Coding:

- Object Oriented
- Node.js

User / Admin Training:

User Training is N/A as this is a hidden data collection service.

Admin Walkthrough:

- The Data Collection service is built upon the unofficial League of Legends API which is documented and maintained [here](#).
- Required Installs
 - Node Version ^ 6.4.1
 - Dotenv ^8.2.0
 - Express ^4.17.1
 - Json-bigint ^0.3.0
 - Mysql ^2.18.1
 - Nedb ^1.8.0
 - Node-fetch ^2.6.0
- The collection service should be run periodically to update all data served to EMULADDER. To run the service type node .index.js into the terminal and wait for execution to finish, resulting in up to date data in the system.

Testing:

- Unit Testing (to ensure data is correct at every level of the collection system as this feeds data throughout our whole ecosystem)
- End-to-end testing (to ensure subsystem as a whole functions correctly)

Complete System

Final product

The production version of EMULADDER has formed into a clean and minimalist version of a Fantasy eSports site. We are looking to hit a target audience of not only people watching / playing eSports but anyone interested in sports at all. We've built a product that is easy to navigate through and has user friendliness at its core. With the ample amount of planning and visualizations of the site we've put in place, we've been able to consistently hit target deadlines and finalize everything we set forward to implement. We set out to offer one Fantasy game being League of Legends and have delivered a fully built out system to allow users to create contests, invite players, draft in-game players, and most importantly have fun. If our proof of concept proves to be successful then we look to expand our already inplace infrastructure to encompass more eSports.

User manual

- [SignUp](#)
- [Login](#)
- [Creating a contest](#)
- [Drafting a team](#)
- [Viewing your contests](#)
- [Admin Walkthrough, Data Collection Service](#)

Source code

- [EMULADDER GitHub](#)
- [Data Collection GitHub](#)

Team member descriptions

- Daniel Frye
 - Built up the Email services using javamail api with google mail as the host using java and typescript. Which is being used by invite and forgot password.
 - Built the invitation sidebar using html css and typescript
 - Worked on forgot password and reset password design

- Gabriel Wilmoth
 - Built out the entire Data Collection System which feeds all eSports related data into our database.
 - Worked on and designed any and all logo's or artwork for the site and Wireframes.
 - Implemented the Animations on the landing page as well as fixed the implementation of the Fantasy Games section of the landing page.
 - Fully Implemented the frontend work on both Sign-Up and Login.
 - Fully implemented frontend and backend work on the Fantasy League of Legends page.
 - Fully implemented Roster and Scoring Breakdown Page.
 - Fully reworked the FAQ Page.
 - Implemented the Coming Soon Page.
 - Built out the ReadMe of both the EMULADDER and Data Collection Repositories on GitHub.
 - Redefined several datatables to match incoming data and better suit the needs for our system.
- Mckenzie Moize
 - Set up backend SpringBoot REST service by creating interfaces and abstract classes for controller, handler, sanitizer, service, etc.
 - Created database and tables on AWS using RDS and MySQL
 - Created JPA repository interfaces for each database table
 - Implemented backend services for account management, including signing up, signing in, forgot password operations, and resetting password
 - Created service classes to make REST calls between front-end TypeScript and backend Java
 - Implemented backend and frontend services for drafting players for contests, as well as created UI for the page.
 - Implemented backend and frontend services for viewing all contests currently registered by player, as well as created UI for the page.
 - Created terms of service, privacy policy, and FAQ informational web pages.
- Rinty Chowdhury
 - Built the navigation bar which includes the emuLadder home button, games button which includes all the available games and

link to go to those games, FAQ button, contests button which shows all the available contests, login button, and signup button.

- Built the game card component which includes game buttons to take the user to the homepage of the game.
- Built the get started component which includes the steps or directions on how to start the emuLadder game. Also includes a pictorial view of the emuLadder game.
- Robert Meyer
 - Fully implemented the frontend work for the create-contest page.
 - Built an events service in TypeScript to display events data from a backend endpoint.
 - Built out backend for events so that the create-contest page can display events data, such as when a contest starts and when each game starts for a particular contest.
 - Assisted in implementation of frontend and backend services to create a contest.