Writing 3 - Team 14

Karim Alami, Philip DiGiorgio, Brandon Harvey, Samuel Fritz

Product Specifications

User Stories

As an Ultimate Frisbee Fan, I would like to predict possible outcomes of matches between different AUDL teams so I can learn more about the sport and which teams are more likely to win the league.

As an Ultimate Frisbee Fan, I would like to customize match settings such as temperature, windspeed, precipitation, and humidity parameters, in order to see how different teams would perform against each other under different weather conditions.

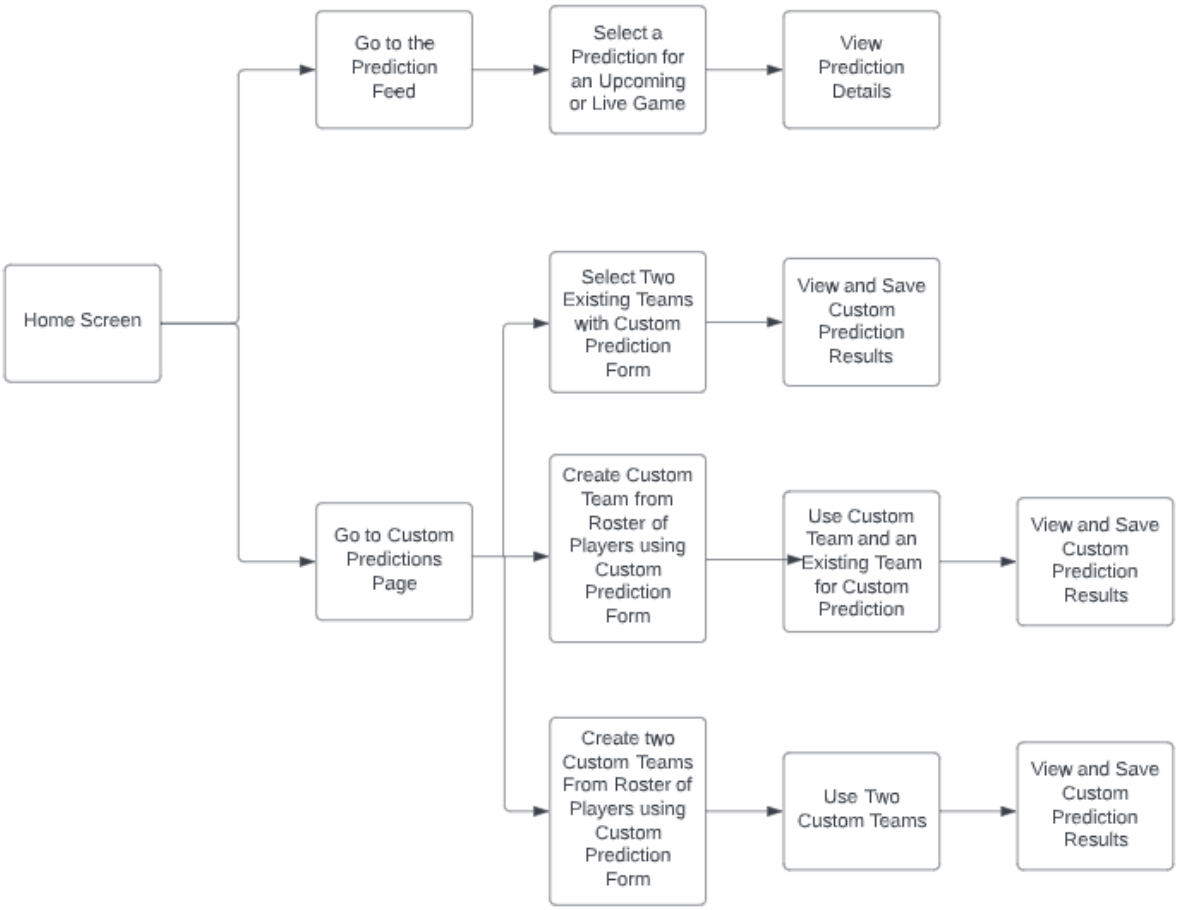
As an Ultimate Frisbee Fan, I would like to view a live prediction feed for current and upcoming games so I can get an idea of who is most likely to win in the given real-time conditions.

As an Ultimate Frisbee Fan, I would like to view in depth details about how each prediction is being calculated, so I can understand the factors that determine the prediction.

As an Ultimate Frisbee Fan, I would like to create my own custom Ultimate Frisbee Predictions using real teams or made-up teams, so I can see how different players and teams perform against each other.

As an Ultimate Frisbee Fan, I would like to create my own custom teams from a selection of all past and current players when creating a custom prediction, so I can create impossible matchup scenarios.

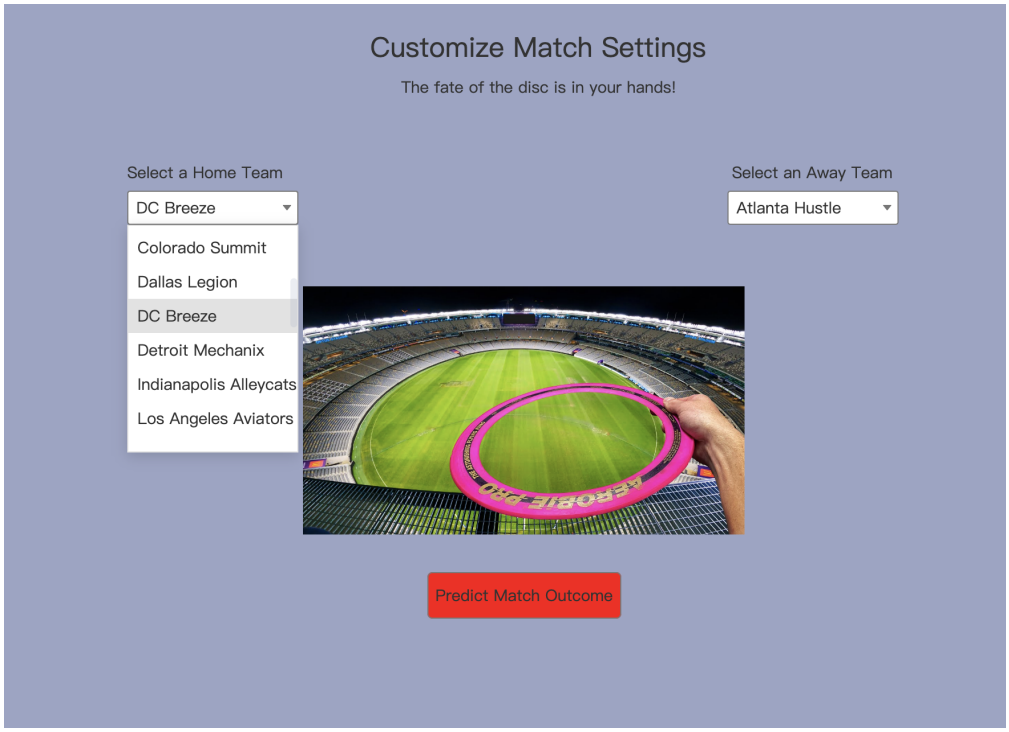
Flow Diagram



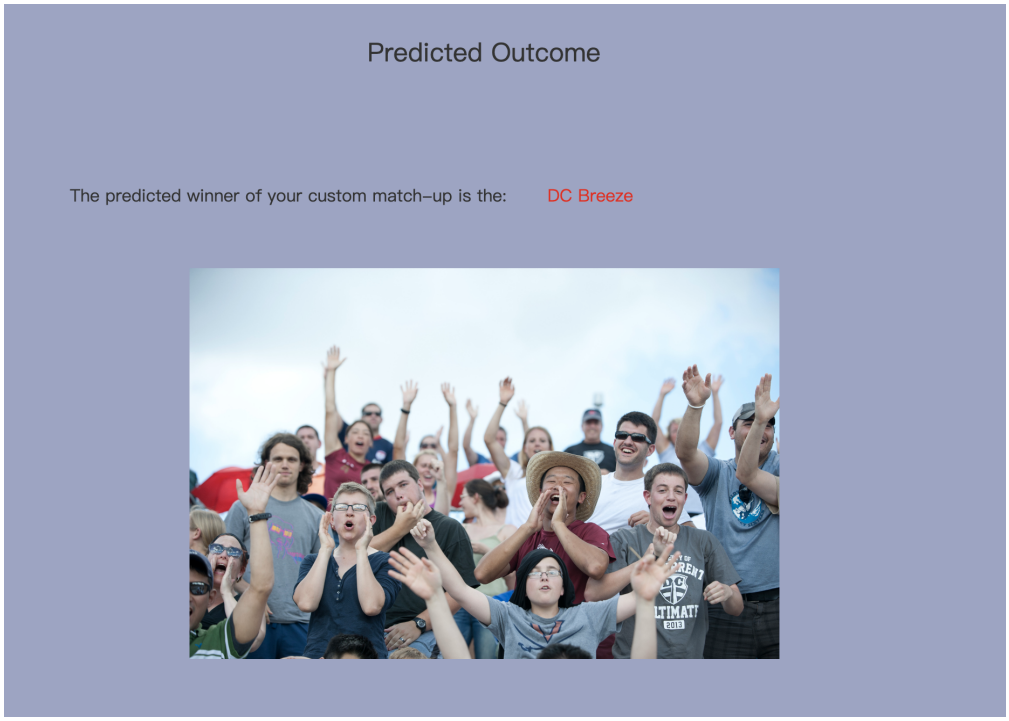
Mockups and Wireframes

Landing page:

Input form (User creating a prediction):

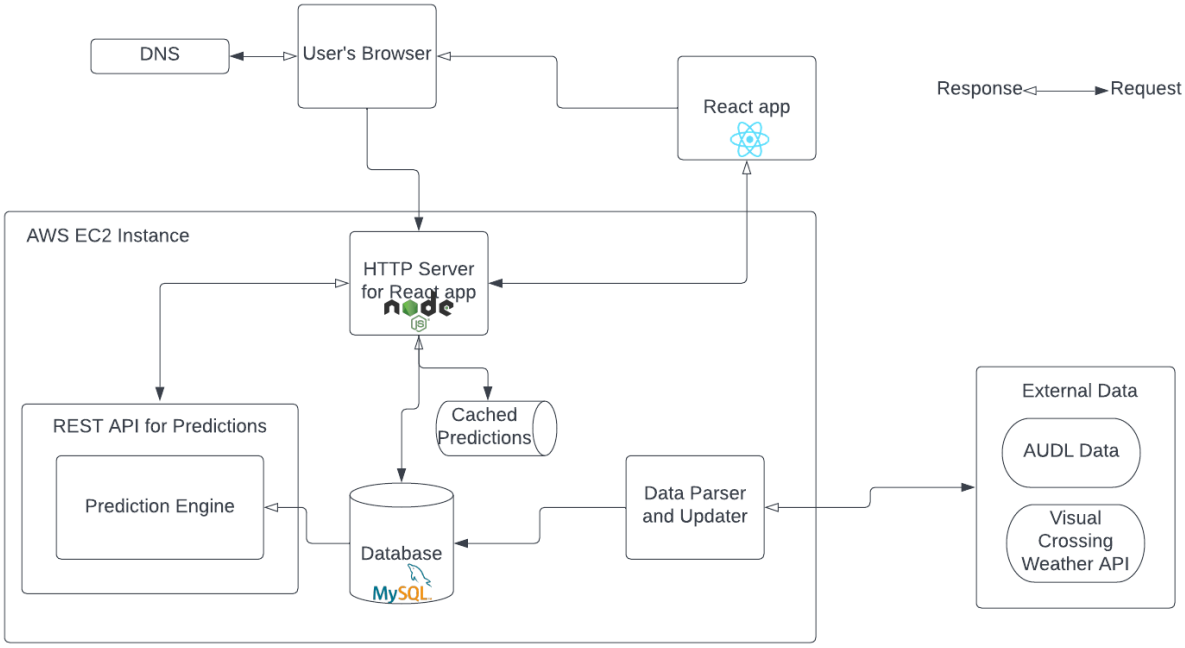


Prediction Output (Prediction generated based on user input):



Technical Specifications

System Diagrams



When a user first tries to enter the site, a DNS lookup is performed, after which a simple GET request is made to the HTTP server that serves the React application. The application is sent to the client's browser, after which point it is used to request data from the server. When a prediction is requested, the cached predictions are checked to see if the prediction was already performed recently. If so, the previous result is returned. If not, the backend requests the prediction API for the new matchup. This result is then returned to the front end and displayed to the user, getting cached along the way.

The data parser retrieves new data from the external data sources on a daily basis and updates the database accordingly. This data is used directly in the prediction engine and in the application to display data.

External Frameworks and APIs

Our project uses a few pieces of external technology. These are all used to build up either the core of the application or to simply the development of another piece. Each is listed below:

AWS EC2:
Our project will be deployed onto an AWS EC2 instance. This enables us to have the project available for public use in the future, and it gives us a centralized database that we can use during development. Specifically, we will use this instance to have a MySQL database that each

developer can connect to and get the most up to date data during development. When it comes to deploy our application, this will serve as the central point of availability, with the specific deployment broken up into different containers.

Docker:
To aid in the deployment of the many different pieces of the application, we will use Docker to have images for each of the different components of the application. The key benefit provided to us by using Docker is the containerization of each component. This will allow us to ensure that tests in development do not cause the entire server to crash and to scale deployments based on demand.

Node.js + Sequelize:
Node.js is our back end for our application. This will serve as the connective piece between the database and public endpoints. It will expose routes that enable the front end to retrieve the necessary data for display on the interface

Sequelize is the Object Relational Mapping (ORM) that we are using to handle the interfacing between Javascript objects and MySQL tables. We are using this so that we can easily interact with the data in the tables in an easy to use object format and simplify the development time of database communication.

React:
React will be used to render the front end for our application. It will allow for dynamic inputs and updates in an easy to use manner, allowing the application display to be easily updated.

Visual Crossing API:

Goal:

- Get temperature, windspeed, precipitation, and humidity data from all past AUDL games as well as forecast data for upcoming games.

Description:

This API is used in the weather folder in the code. There is a separate script used to collect and store data from all past AUDL games. The parameters location and startTime for this script are gathered from game data files stored on a local system while the parameter endTime is computed automatically. This API will also be used to gather and update forecast data for upcoming games every 24 hours (To be implemented in the future).

Endpoints Used:

- GET /VisualCrossingWebServices/rest/services/weatherdata/history
- GET /VisualCrossingWebServices/rest/services/weatherdata/forecast

Scikit-learn:

This will be used in order to implement a Support Vector Machine with reliability in Python 3 allowing for easier implementation and testing of various features available based on previous game data to give a reliable outcome as to which team is predicted to win.

All of these external frameworks and APIs are necessary in order to facilitate the development of the application. Of note, only the AWS EC2 instance incurs a cost for development. However, this cost is very low (estimated around $250 with AWS Pricing Calculator).

Algorithms

Support Vector Machine (SVM) ML Algorithm (Using Sci-kit Python Library):
The main algorithm used for our prediction engine is an SVM model structured using the scikit-learn python library that takes in data from our database in the form of a 2-D array containing a list of samples with a certain number of features along with a 1-D array of targets which signify the team that each of the sames in the 2-D array refers to. After the model is fitted, it uses the built-in predict() function to output which team will be predicted to win based on the parameterized features compared to the SVM.

This algorithm is the key part of the application, as it is how the predictions are made for matchups. Success for the algorithm can be measured by ensuring that a prediction can be made for two inputted teams and a selection of weather factors, ultimately giving a projected winner.