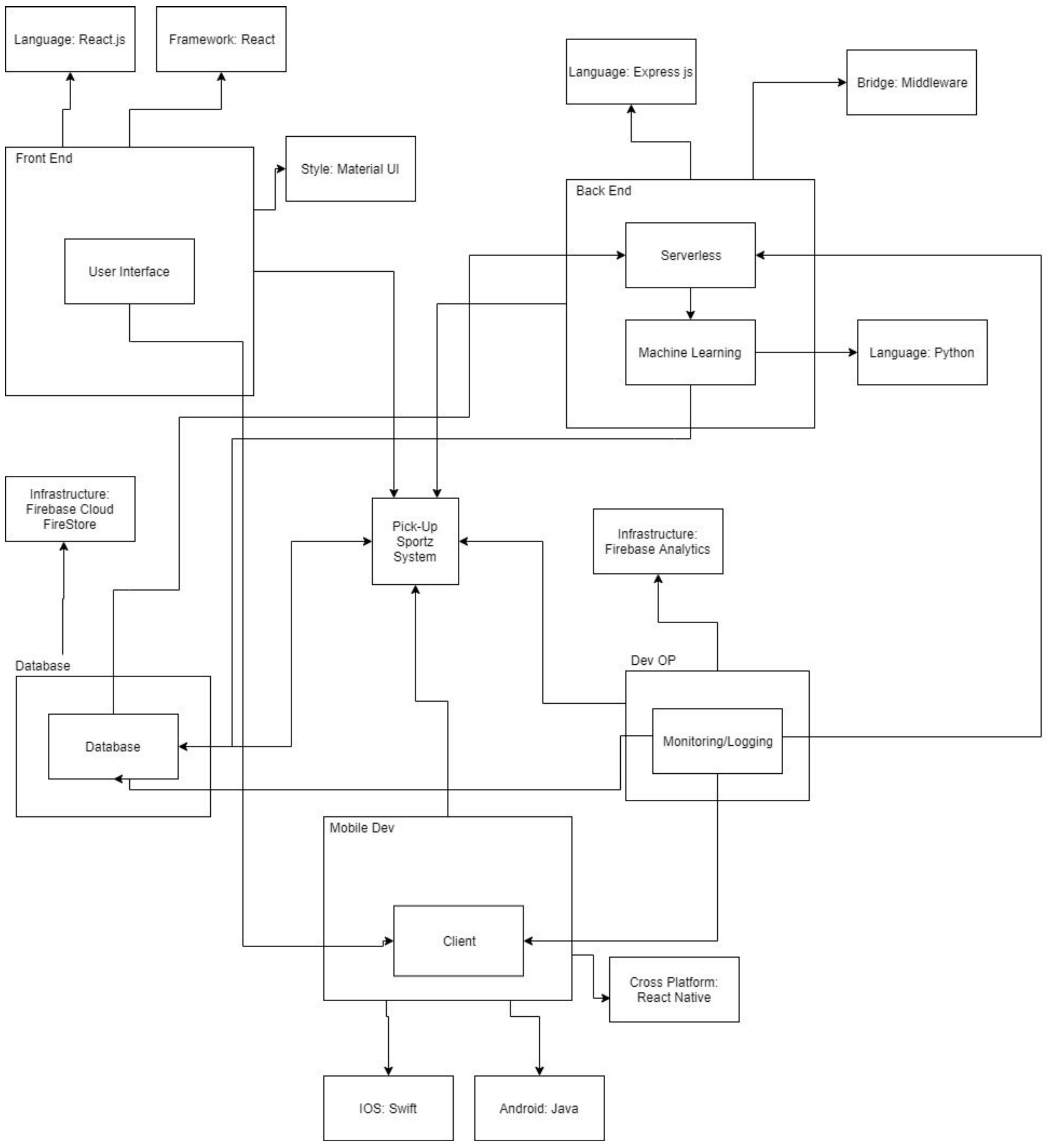
Pick-Up Sportz
Architecture and Design Document

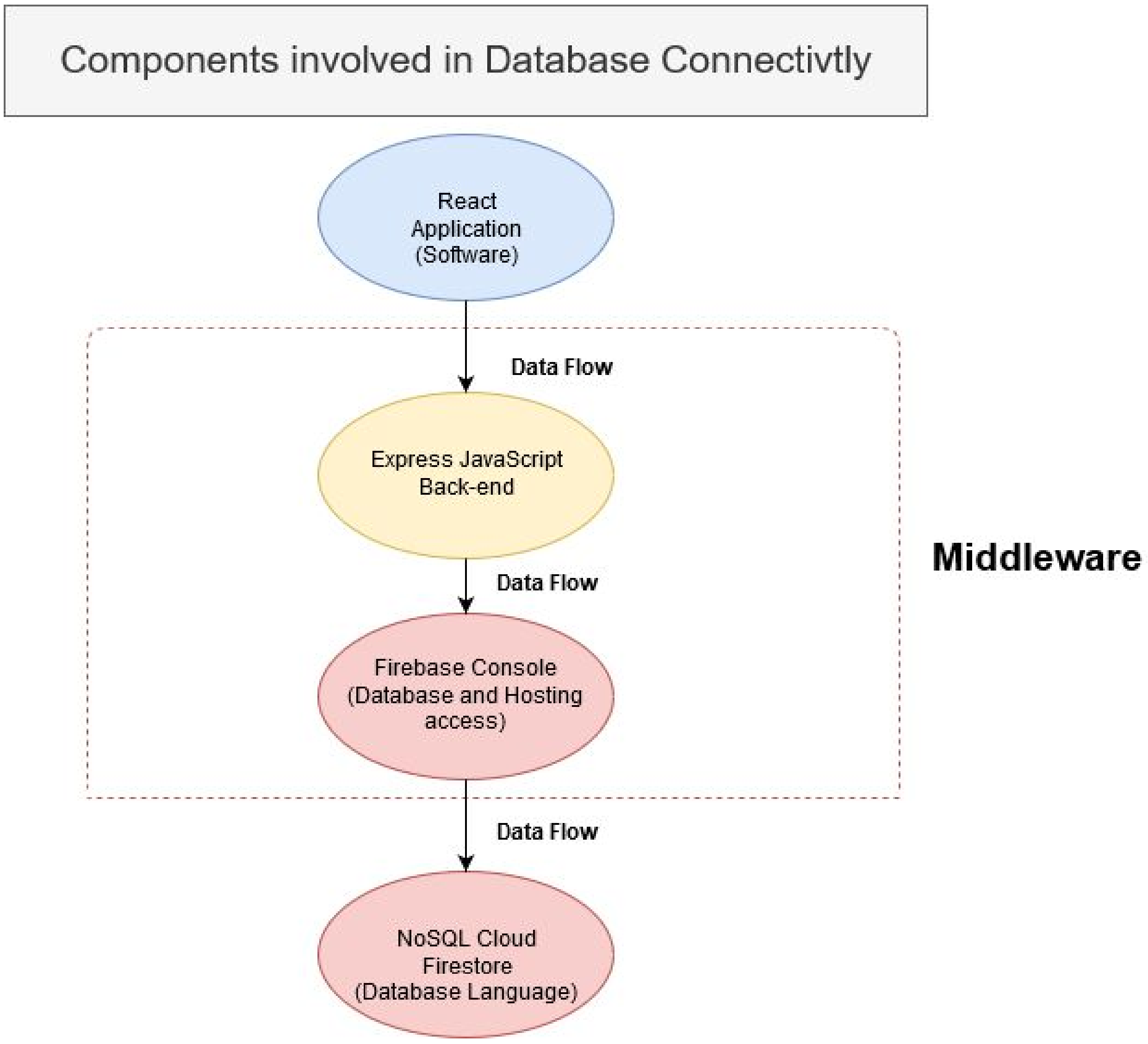
By
John Him
Jamil Khan
Brandon Le
Benjamin Seo
Chaz Del Prato
Christine Duong

Table of Content

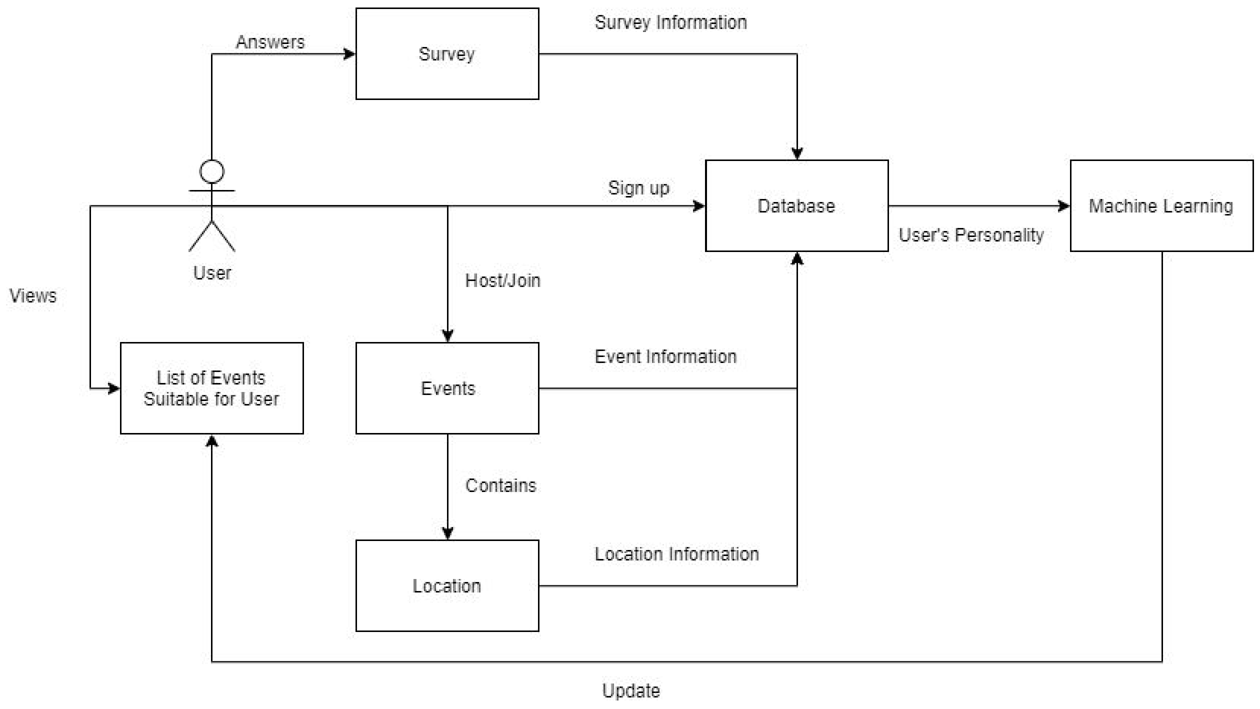
Diagrams

System Diagram(s) and Component Diagram(s)




Components involved in Database Connectivtly
React
Application
(Software)
Data Flow
Express JavaScript
Back-end
Middleware
Data Flow
Firebase Console
(Database and Hosting
access)
Data Flow
NoSQL Cloud
Firestore
(Database Language)

Information Flow Diagram



Quality and Quantity Standard

To set up the front end of our web application, we will be using React.js through the React Native framework. It will be set up with the Material UI style. By using React Native, there is less burden on us to handle the UI interactions and makes threading a lot easier.

For mobile development, using React Native will allow us to develop the application for either the Android, iOS platform, or even both to allow the app to function across both platforms. If we choose to build the app solely on iOS then we will use Swift, and Java if we choose to make the app exclusively for Android.

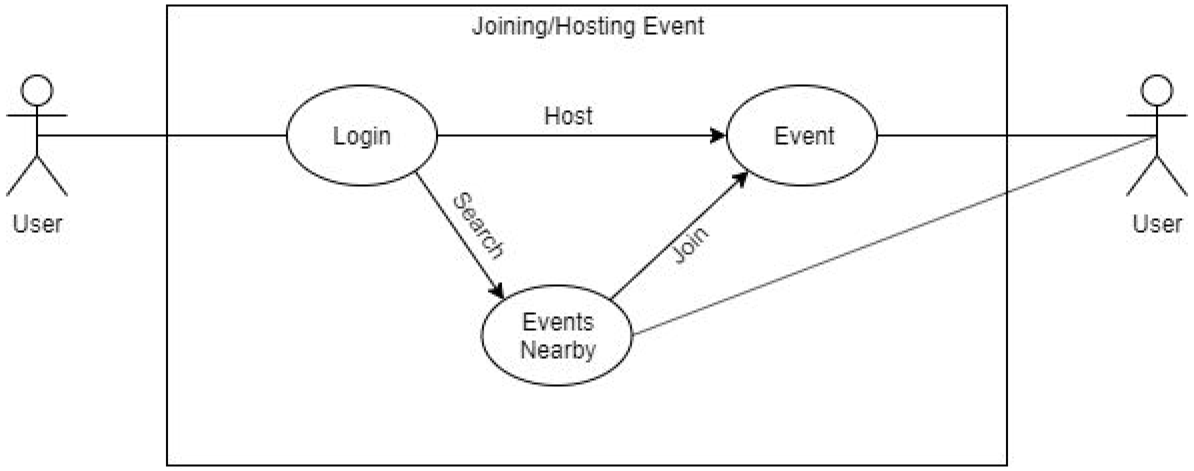
To set up the back end of our application, the language we are planning to use is Express.js. This will consist of a serverless component and our machine learning component written in Python that will communicate with our client, database, and our DevOps.To bridge our serverless component to our database we would use a middleware framework.

To set up the database component, we will use NoSQL Cloud Firestore as a primary foundation for our relational database. The database will obtain data from numerous components such as, client, server, monitoring/logging, and machine learning components. From this, it will work hand-in-hand with our back end.

To set up the DevOp component, it will consist of an infrastructure using Firebase with a firebase analytics. This portion will consist of our monitoring/logging components that will communicate with our client, server, and database.

Use-Case Diagrams

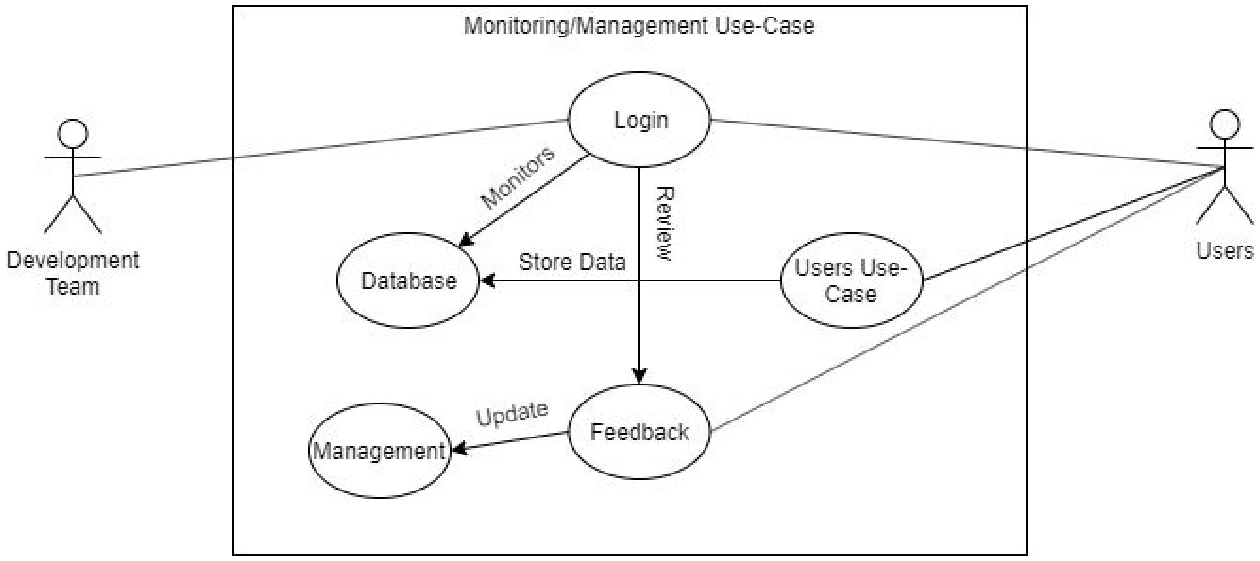
User Host/Join Use Case:



- Use-case field
    - This use case shows the action of the user when joining/hosting an event.
- Use-case name
    - Host/Join an event
- Subject area
    - Users Use Case
- Actors
    - User (left) is the actor that is hosting/joining a game
    - User (right) is the actor that is other user(s) that hosted or joined a game
- Use-case overview
    - A user (left) first logins then either search for an event nearby to join or host their own event.
    - A user (right) has either joined the event that the user (left) has hosted or joined the same event the user (left) has joined.
- Preconditions
    - User (left) must have signed up and answered the survey.
- Termination outcome
    - The use case might end when the event the user (left) joined is submitted to the system, i.e. the game has started.
    - The use case might end when the event the user (left) joined is cancelled.
- Condition affecting termination outcome

    - Cancelled event.
    - Event has passed the deadline.
    - Event has started
- Use-case description
    - Cancelled event.
        - The system will notify the users that the event has been cancelled and will close down the event.
    - Event has passed the deadline.
        - The system will notify the users that the event has passed the deadline, erasing the event from the database.
    - Event has started
        - The system will notify the users that the event has started and will save the event into the users schedule/database, while also erasing the event from the event nearby list.
- Use-case associations
    - Sign up Use Case
    - Check Schedule Use Case
    - Filter/Map Use Case
    - Look Up Other Users Profile Use Case
- Traceability to a list of other related documents, models, and products that are associated with this use case
    - Map API
    - Users Profile/History
    - Nearby Events
- Input summary
    - Event search bar/filter
    - Event descriptions/information
    - Login information
- Output summary
    - List of events nearby
    - Schedule
- Usability index (out of 10)
    - Satisfaction (7)
    - Importance (10)
    - Frequency (10)
- Use case notes
    - System must keep track of information of events, such as type of event, location, either hosted or joined, etc. so that the machine learning component can learn through the user’s personality which can result in better events suited for them.

Monitoring/Management Use Case:

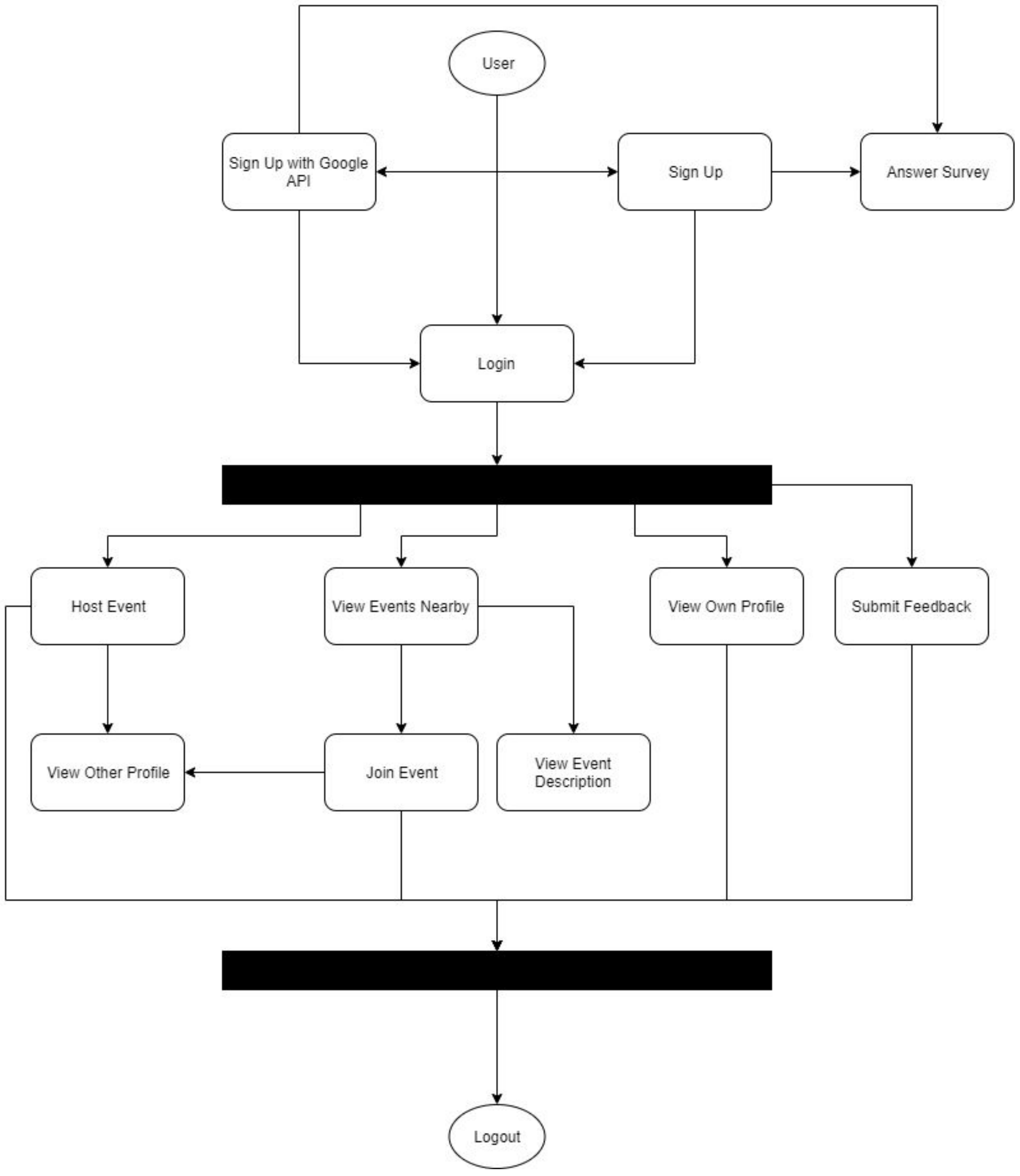


- Use Case Field
    - This use case shows what and how the development team monitors/manage in the system
- Use Case Name
    - Manage/Monitor the System
- Subject Area
    - Developers Use-Case
- Business Event
    - Login
- Actors
    - Development Team
    - Users
- Use-Case overview
    - The development team monitors the users activity; such as events joined/hosted, change in personal information, etc. while also managing feedback from users when they use the product.
- Preconditions
    - When users change their information or someway alter the database through using the application. In addition, when users submit a feedback/comment that has to do with some functionalities of the product.
- Termination Outcome
    - There is no termination outcome for monitoring and managing the application. There needs to be constant managing and monitoring.
- Condition Affecting Termination Outcome
    - No conditions
- Use Case Associations
    - User Use Case
- Input summary

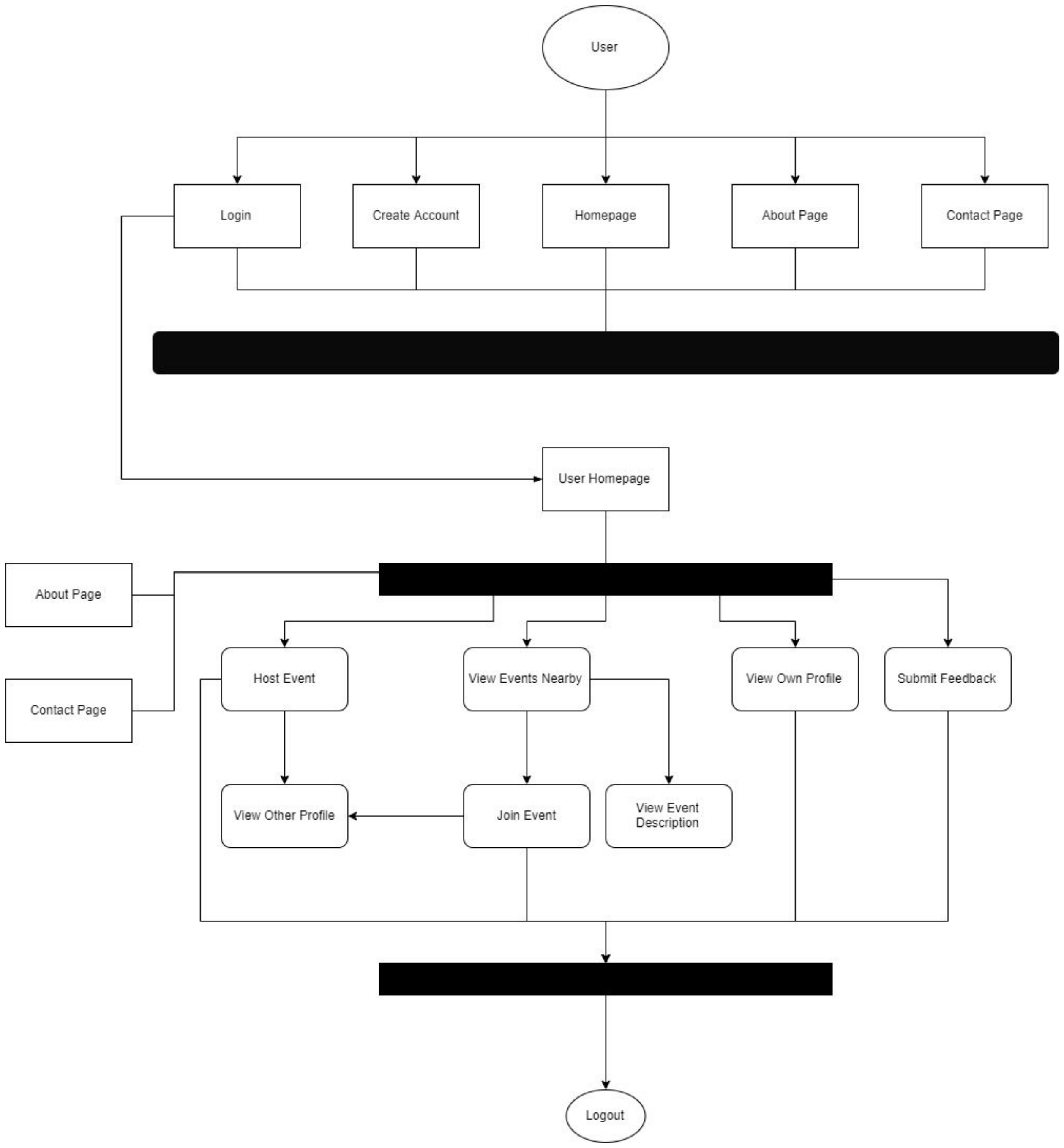
    - Login information
    - Notification for change in system
    - Update Notification
    - Feedback reply
- Output summary
    - System Notification
    - Feedback reply
- Usability Index (out of 10)
    - Satisfaction (6)
    - Importance (10)
    - Frequency (10)
- Use case notes
    - This use case will allow developers to implement new functionalities according to user feedback. In addition, it will allow developers to secure the confidential information/data leaks.

Activity Diagram

Mobile

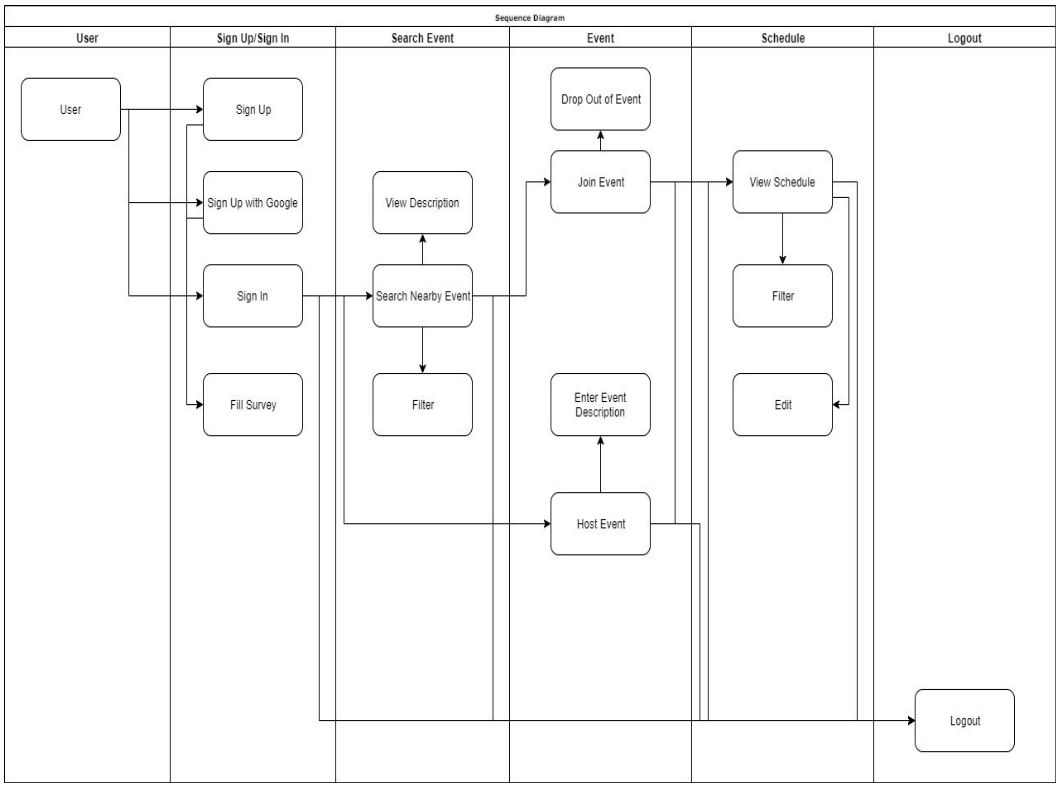


Web

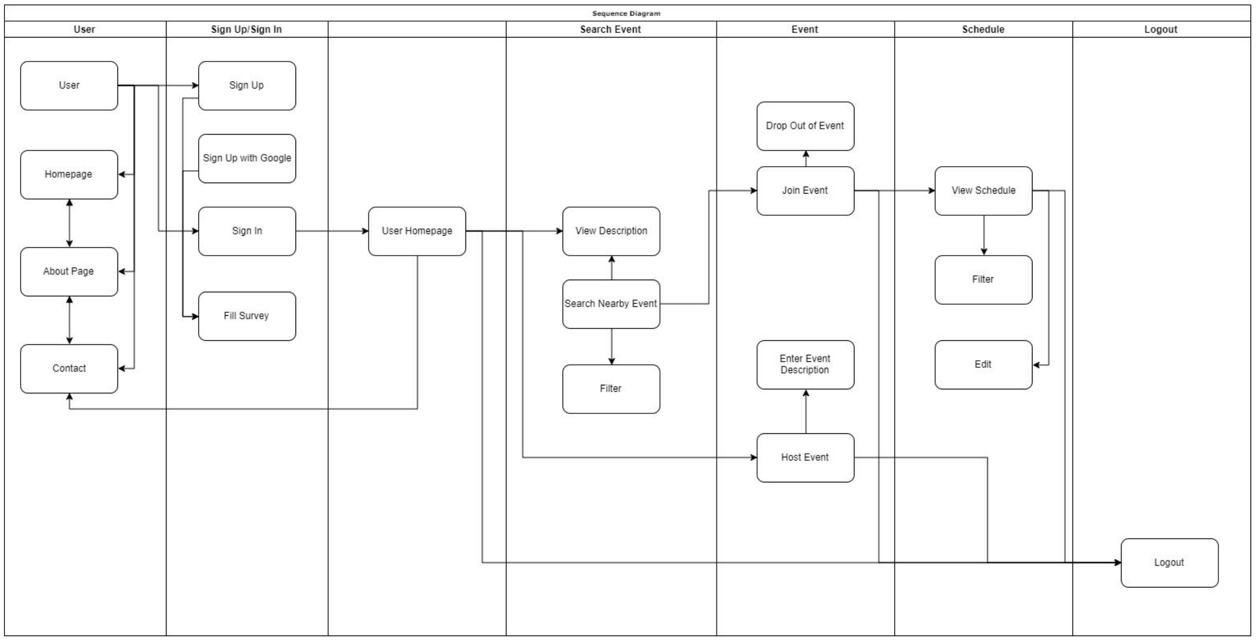

User
Login
Create Account
Homepage
About Page
Contact Page
User Homepage
About Page
Contact Page
Host Event
View Events Nearby
View Own Profile
Submit Feedback
View Other Profile
Join Event
View Event Description
Logout

Sequence Diagram

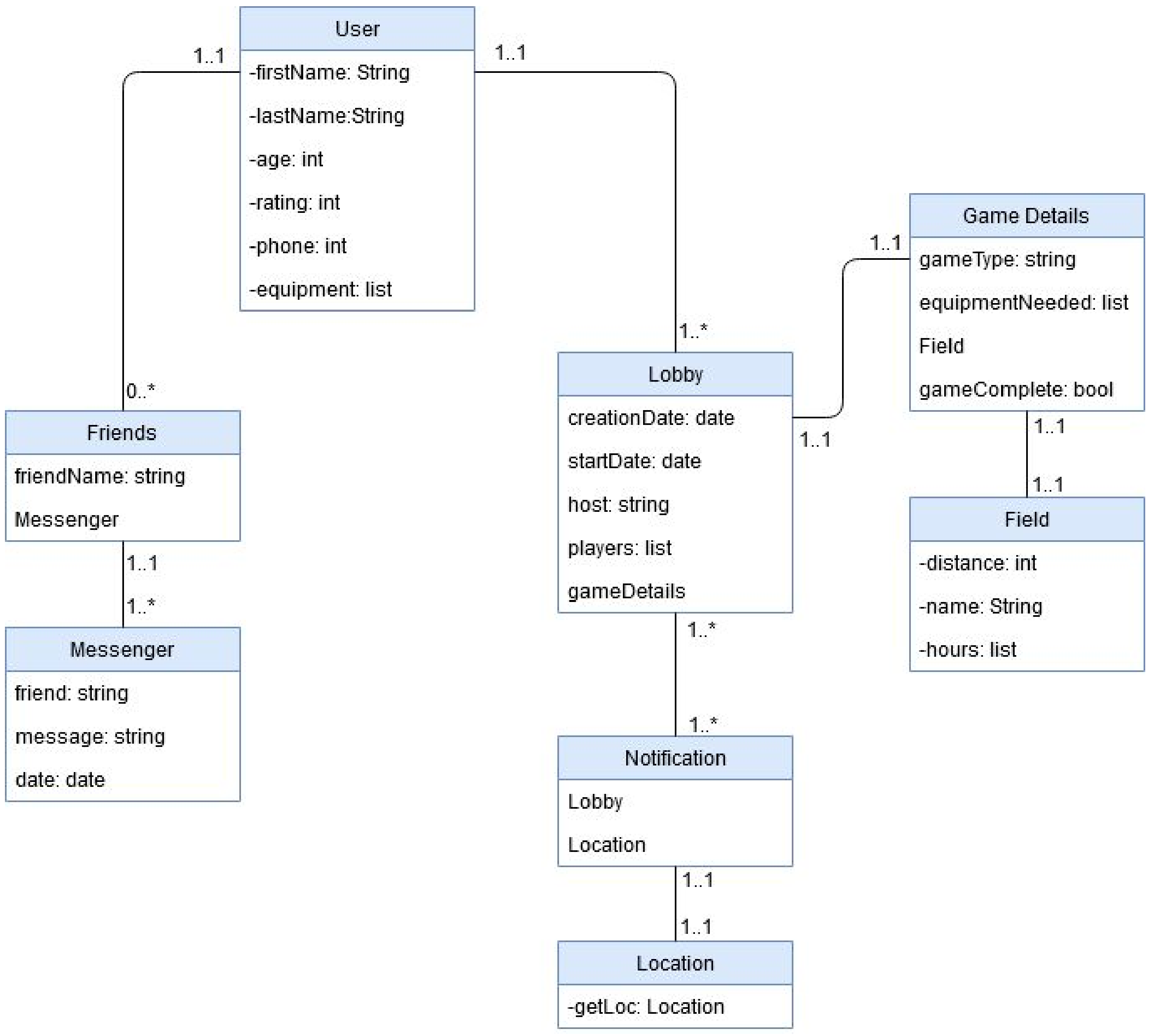
Mobile



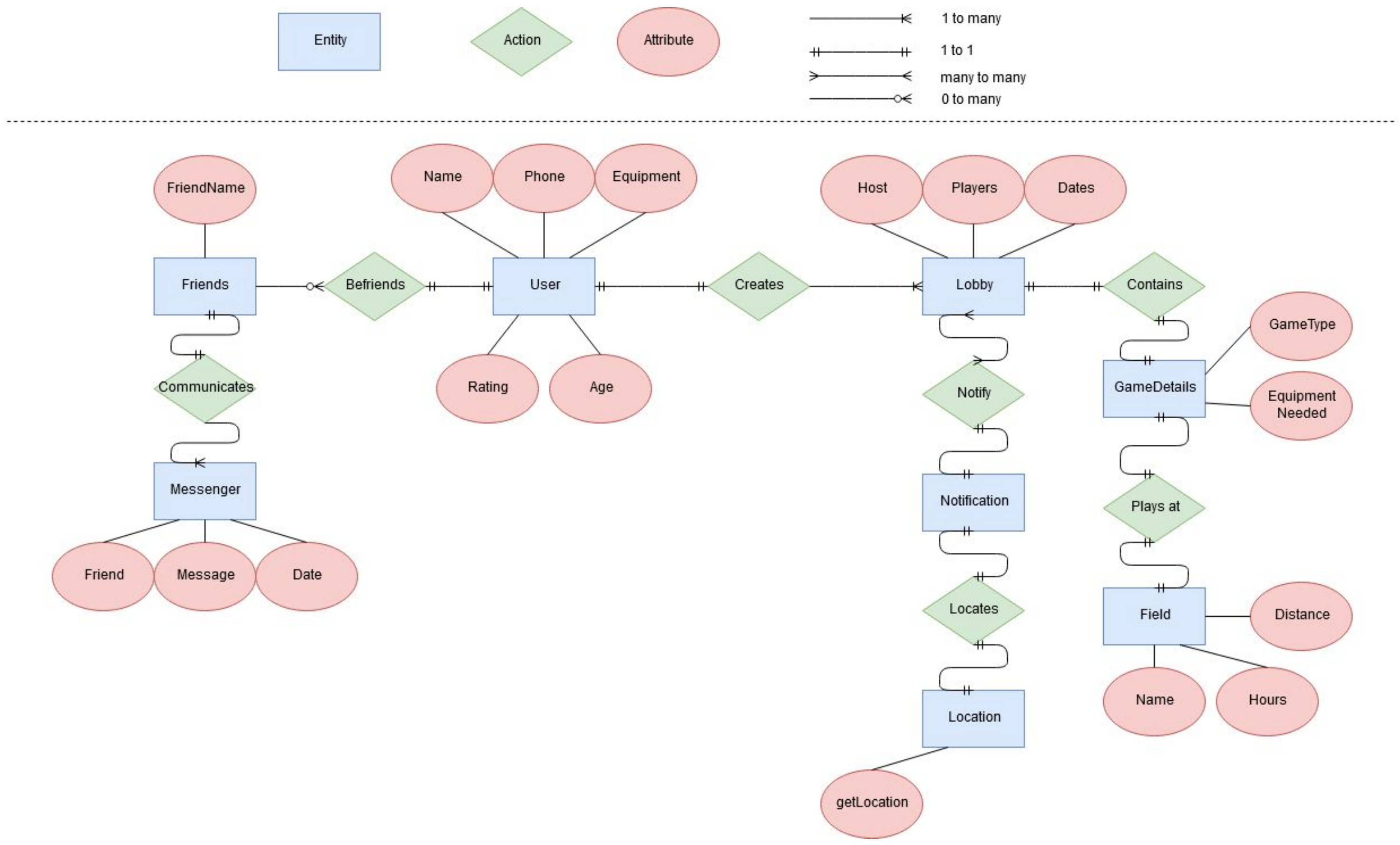
Web


Sequence Diagram
User
Sign Up/Sign In
Search Event
Event
Schedule
Logout
User
Homepage
About Page
Contact
Sign Up
Sign Up with Google
Sign In
Fill Survey
User Homepage
View Description
Search Nearby Event
Filter
Drop Out of Event
Join Event
Enter Event Description
Host Event
View Schedule
Filter
Edit
Logout

Class Diagram



Relational Diagram



Analysis

Tradeoff Analysis

Architecture Patterns Employed	Pros	Cons
Layered Pattern	Lower layer can be used by different higher layers. Layers make standardization easier as we can clearly define levels. Changes can be made within the layer without affecting other layers	Not universally applicable. Certain layers may have to be skipped in certain situations.
Client-Server Pattern	Good to model a set of services where clients can request them.	Requests are typically handled in separate threads on the server. Inter-process communication causes overhead as different clients have different representations.
Master-Slave Pattern	Accuracy - The execution of a service is delegated to different slaves, with different implementations.	The slaves are isolated; there is no shared state. The latency in the master-slave communication can be an issue, for instance real-time systems. This pattern can only be applied to a problem that can be decomposed.

Component Choices	Pros	Cons
Front End		
Languages		
HTML	Publish Online Docs, Retrieve online info, designing forms, include apps in their documents. Easy to learn.	Open to author interpretation. Technical progress is slow. It is a declarative language, meaning that it has limited functional prowess

	Well-interpreted by browsers. HTML parsers are forgiving. Light-weight. Has a vast array of structural and aesthetic elements that can infer meaning and presentation.	compared to functional languages.
CSS	Design colors, fonts and layouts. Adapt display across platforms. Easier site maintenance. Tailored pages. Help make spontaneous and consistent changes. Improves page loading speed. Device Compatibility. Ability to reposition. Makes the search engine better crawl your web pages.	Cross-browser issues. Confusion due to its many layers. Vulnerable
Javascript	Speed, simplicity, popularity, interoperability, server load, and gives the ability to create rich interfaces.	Client-side security. Different browsers may use javascript.
Framework		
AngularJS	Two-way data binding, DOM manipulation, improved server performance, faster application prototyping, responsive web, MVVM architecture, plain html templates, fast development	JavaScript support is mandatory, inexperience with MVC, difficult to debug the scopes. Difficult to learn.
ReactJS	Updates process is optimised and accelerated. JSX makes components/blocks code readable. It displays how components are plugged or combined with. React’s data binding establishes conditions for creation dynamic applications. Up to date. Facebook team supports the library. Advice or code samples can be given by Facebook community.	Learning curve. Being not full-featured framework it is required in-depth knowledge for integration user interface free library into MVC framework. View-orientedness is one of the cons of ReactJS. It should be found 'Model' and 'Controller' to resolve 'View' problem. Not using isomorphic approach to exploit application leads to search engines indexing problems.
JQuery	jQuery is flexible and fast for	jQuery is easy to install and learn,

	web development. It comes with an MIT license and is Open Source. It has an excellent support community. It has Plugins. Bugs are resolved quickly. Excellent integration with AJAX.	initially. But it’s not that easy if we compare it with CSS. If jQuery is improperly implemented as a Framework, the development environment can get out of control.
Style		
Bootstrap	You are the Boss. You Pick the Focus. You Maintain Responsibility.	Personal Risk. Lack of Networking. Slow Growth.
Material UI	Included icon font. Customizable. Responsive. Large selection of components. Perfect for React projects. Adheres really well to the material design standard.	Large / heavy. Active development means API changes are frequent. Refuses to use the flexbox model.
Back End		
Languages		
Express.js	flexibility, simplicity, extensibility and performance. Fast app development. I/O request handling. Open-source community. Easy integration of third-party services and middleware. Easy to learn.	Event-driven nature (callbacks). Code organization.
Python	Very good with dealing with AI and ML. Preferred languages for creating AI or ML based web & mobile apps. A multipurpose language which can be used by software product development. Easy to use.	Performance issue. Absence from mobile and browser computing. A lot of design restrictions.
Ruby	Oo language , Excellent documentation, Rapid prototyping/ Development, RoR=MVC framework, Easy language to start with , Cool community (extremely	Not that hot anymore , Ruby generally except for web apps is not too notch (mainly comparing with python) Generally regarded as a slow language.

	friendly)	
Database		
Infrastructure		
MySql	.MySQL products remain solid. There is more MySQL investment and innovation than ever before. MySQL is designed with a focus on the Web, Cloud and Big Data. There are more MySQL projects than before.	MySQL is an open source (kind of). MySQL is not as mature as other relational database management systems. MySQL is Oracle-owned instead of community driven. Big names are jumping ship
Firebase	simple serialization of app state. 3-way data binding via Angularfire. minimal setup. no server infrastructure needed to power apps with data.	not widely used or battle tested for Enterprises. very limited querying and indexing. no aggregation. no map reduce. can't query or list users or stored files.
DevOP		
Infrastructure		
AWS	Easy to Use. No Capacity Limits. Provides Speed and Agility. Secure and Reliable.	Limitations OF Amazon EC2. Security Limitations. Technical Support Fee. AWS pricing list. General cloud Computing Issues.
React	Biggest community and code owned and maintained by Facebook. Big ecosystem. Flexibility. Scalable. Can be used for native and web apps.	Requires to learn JSX and is more programmer-oriented. It's kind of verbose. Writing components isn't as straightforward as pure HTML & JS. Being too flexible in structure can be problematic. In React, everything is just JavaScript.
Vue	Backed by Laravel and Alibaba. Big players are starting to adopt the framework. Simplicity of the syntax and short learning curve for newcomers. Uncomplicated structure. Features a lot of concepts from Angular 1 and React.	Owned by one person, maintained by a small team. Being too flexible in structure can be problematic. Legacy code from the current Angular 1.5 will need some conversion.

Mobile Dev		
Android		
Java	Used for building enterprise scale web application. Android mobile app developers also rely on this language. Most stable language.	Paid for commercial use. Poor performance. Far from a native look and feel on the desktop. Verbose and complex code
SDK	Easier integration. Faster time to market. Stronger security. Cost savings. Reliability.	Unfamiliar
IOS		
Objective C	Objective-C is a mature language with a huge community, lots of experienced developers, best practices, and coding styles. There are plenty of third-party libs, that are well tested, used in many huge projects. (note: some of them may not be updated to the latest iOS releases). There are tons of legacy code that needs to be supported. (This one especially for Crusty). Compatibility with other languages. You can use C or C++ with no problem inside your Objective-C files. If you need to use C++ in your Swift code you should first wrap it in Objective-C code first and then import that in Swift.	Sooner or later Objective-C won't be able to support some new features, so there will be Swift-only marked methods and frameworks. It is an outdated language with lack of many valuable features available in other programming languages.(Just look at Swift enums and cry).
Swift	Modern, more expressive, feature-rich programming language. Very active community. Ported to other platforms. Rapidly applying new features and proposals.	lack of 3-d party solutions due to fast evolution of language. There are lots of projects on Github that still on Swift 1.2. Immature community and a lot of misleading advice.
Cross Platform		
React Native	Faster to Build. One	Less Smooth Navigation. Lack of

	Framework, Multiple Platforms. Hot Reloading. Smaller Teams. Fast Applications. Simplified UI.	Some Custom Modules. Native Developers Still Needed.
PWA	Progressive. Responsive. App-like. Updated. Secure. Searchable. Reactivable. Installable.	iOS support from version 11.3 onwards. greater use of the device battery. not all devices support the full range of PWA features. it is not possible to establish a strong re-engagement for iOS users. support for offline execution is however limited.

Language Choices	Pros	Cons
Java	Used for building enterprise scale web application. Android mobile app developers also rely on this language. Most stable language.	Paid for commercial use. Poor performance. Far from a native look and feel on the desktop. Verbose and complex code
Python	Very good with dealing with AI and ML. Preferred languages for creating AI or ML based web & mobile apps. A multipurpose language which can be used by software product development. Easy to use.	Performance issue. Absence from mobile and browser computing. A lot of design restrictions.
Java Script	Fast, simple, popular, interoperability, serverload, rich interfaces, extended functionalities, versatile.	Client-side security. Different browser support.
React.js	Components can be reused. It has a virtual DOM. Popular.	Fast pace in development. Lack of conventions. Difficult to learn
Express.js	Flexible, simple, extensible, and good performance. Fast app development. Good I/O request handling. Open source community. Easy integration of 3rd party	Performance bottlenecks with heavy computation tasks. A lot of callback issues.

	services and middleware	

Framework Choices	Pros	Cons
AngularJS	A complete framework. 2D Data flow. A complete solution. Improved server performance. Faster application prototyping. Responsive Web. Plain HTML templates	Difficult to learn overall. JavaScript is mandatory. Inexperience with MVC. Possible time consumption
ReactJS	Easy to learn and use. Creating Dynamic Web applications is easier. Reusable components. Performance enhancement.	Only a javascript library. Difficult to learn if working with Redux. 1D data flow. High pace of development. Poor Documentation. JSX as a barrier
Django	Easy to learn. Clarity and Readable. Versatile. Fast to write. No Design Holes. Fast. Fully loaded. Secure. Scalable.	Uses routing pattern to specify its URL. Too monolithic. Everything is based on Django ORM. Components get deployed together. Need to know the whole system to make it work.

Database Choices	Pros	Cons
SQL/RDBMS/Relational	Relational Databases are well-documented and mature technologies, and RDBMSs are sold and maintained by a number of established corporations. SQL standards are well-defined and commonly accepted. A large pool of qualified developers have experience with SQL and RDBMS. All RDBMS are ACID-compliant - they satisfy the requirements of Atomicity, Consistency, Isolation, and Durability.	RDBMS do not work well, or at all, with unstructured or semi-structured data due to schema and type constraints. This makes them ill-suited for large analytics or IoT event loads. The tables in the relational database will not necessarily map one-to-one with an object or class representing the same data. When migrating one RDBMS to another, schemas and types must generally be identical between source and

		destination tables for migration to work (schema constraints). For many of the same reasons, extremely complex datasets or those containing variable-length records are generally difficult to handle with RDBMS schema.
NoSQL/Non-Relational Databases	Schema-free data models are more flexible and easier to administer. NoSQL databases are generally more horizontally scalable and fault-tolerant. Data can easily be distributed across different nodes. To improve availability and/or partition tolerance, you can choose that data on some nodes to be eventually consistent.	NoSQL databases are generally less widely adopted and mature than RDBMS solutions, so specific expertise is often required. There is a range of formats and constraints specific to each database type.

Server vs. Serverless Choices	Pros	Cons
Server	Have access to a lot of public APIs. Can handle complex and long-running codes faster and easier. It is scalable.	Cost more. Difficult to set up different environments.
Serverless	Cost Less. Easier to set up different environments. It is scalable	Can only access private APIs. Complex and long-running functions are not good.

Machine Learning Write Up

Application Use Case
Whenever a user wants a sport to be recommended to them based on their height and weight.

Benefits

- Stakeholders
    - will not waste time to explore what other sports best fits them. This will result in better experience and better networking.
- Users
    - they can benefit by playing sports that are recommended to them based on other users with similar body attributes.
- Business
    - it can benefit through new users okaying new sports which can affect the ad revenue specifically targeted to newcomers.

ML Model vs. Our Objectives
Our machine learning model that we plan to use is supervised learning. It consists of a target/outcome variable which is to be predicted from a given set of predictors. Using these set of variables, we generate a function that maps inputs to desired outputs. The training process continues until the model achieves a desired level of accuracy on the training data. Since our objective for the application of machine learning in our product is to recommend user's sports that are best suited for our users, this model works hand-to-hand with our objectives.

Architectural Choices
- Web Server (Layered Pattern Architecture)
    - We chose to use a layered pattern architecture since it works well with our database, which is Firebase's Cloud Firestore, and messaging system, such as user-to-user communication and system messages. It also works well with web applications.

Architectural Choices Considered
- Honestly, we had a difficult time finding a machine learning architecture that best fit our model. What stuck to us was the decision tree, but have considered clustering. However, since we are using a recommendation system, we thought decision tree would be easier.

Machine Learning Required Sections

Business needs

- **What is the business purpose of your ML model?**

  The business purpose of our ML model is to recommend different sports to our user based on their height and weight given from questions asked when creating an account/profile. It takes in the weight and height of the user and outputs a recommended sport to the user.

- **What features/labels does the model training need?**

  features: {weight, height}
  label: {sport}

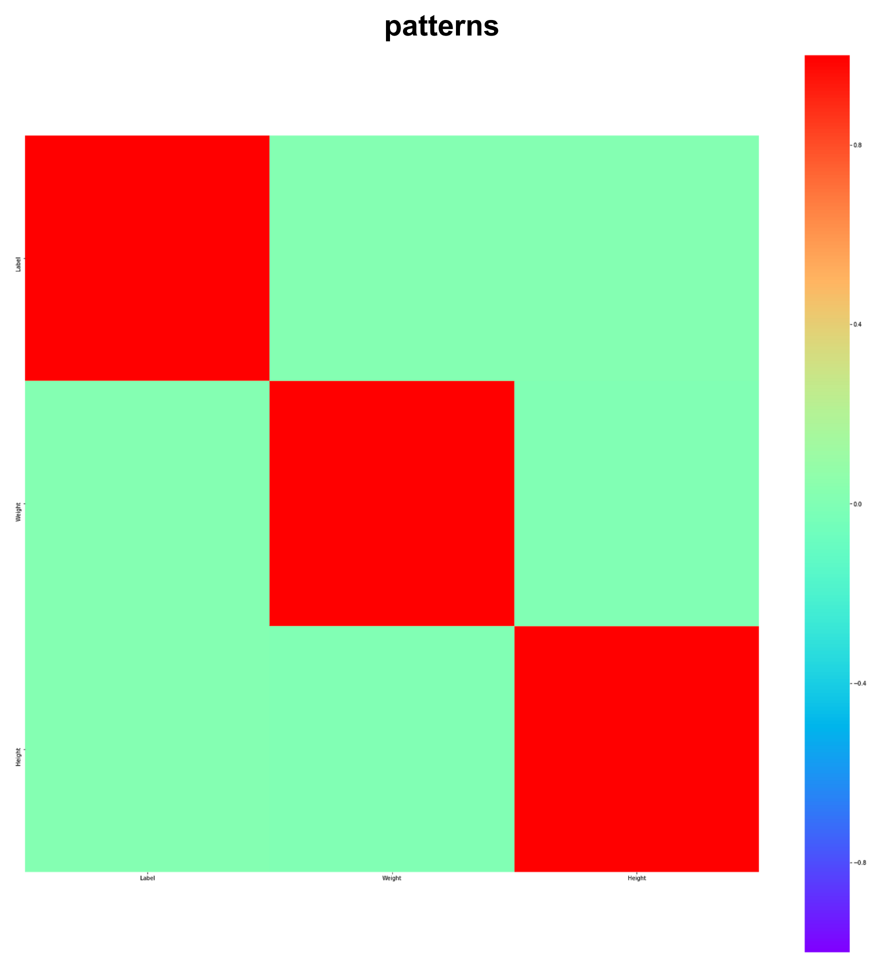
Sources of Internet data, URL's

- Due to analyzing Internet data through kaggle, we found that the body and weight of the professional sports player coincides with other sports. Because of this, we plan on not using realistic internet data, however, we create our own dataset that represents the realistic internet data. We did this by randomly generating the values for weight and height within the range of realistic weight and height values for all sports. For example, height would be randomly generated between 5ft to 7ft, whereas, weight would be randomly generated between 120 lbs and 220 lbs. Due to the painstaking task of gathering realistic internet data of 10+ sports and merging the into one, it was easier to just randomly generate realistic data through the use of Maven with Java Faker and SDGen.

Approach to training data

- **What are the features/labels of the Internet data?**
  - Since we are creating our own data (as explained above) the features would be the user's weight and height. In addition, the labels would be the sports that they play. When going through the decision tree model, a sport will be recommended and placed into that label.
- **What are the features of your business data?**
  - Same reason as above, the features of our business data are the user's weight and height.
- **How are the two sets related?**
  - Due to not using internet data but using our own created business data, there are no relations to the two sets. However, if we did use internet data from painstakingly searching 10+ sports and merging them into one, we would say that the two sets would be related to the height and weight of the user. Our ML model is very simple in that it recommends a sport through the user's weight and height.

- **(After mapping Internet date to your needed data) Data visualization and analysis of your features/labels that must include a Pearson heatmap of feature/label**

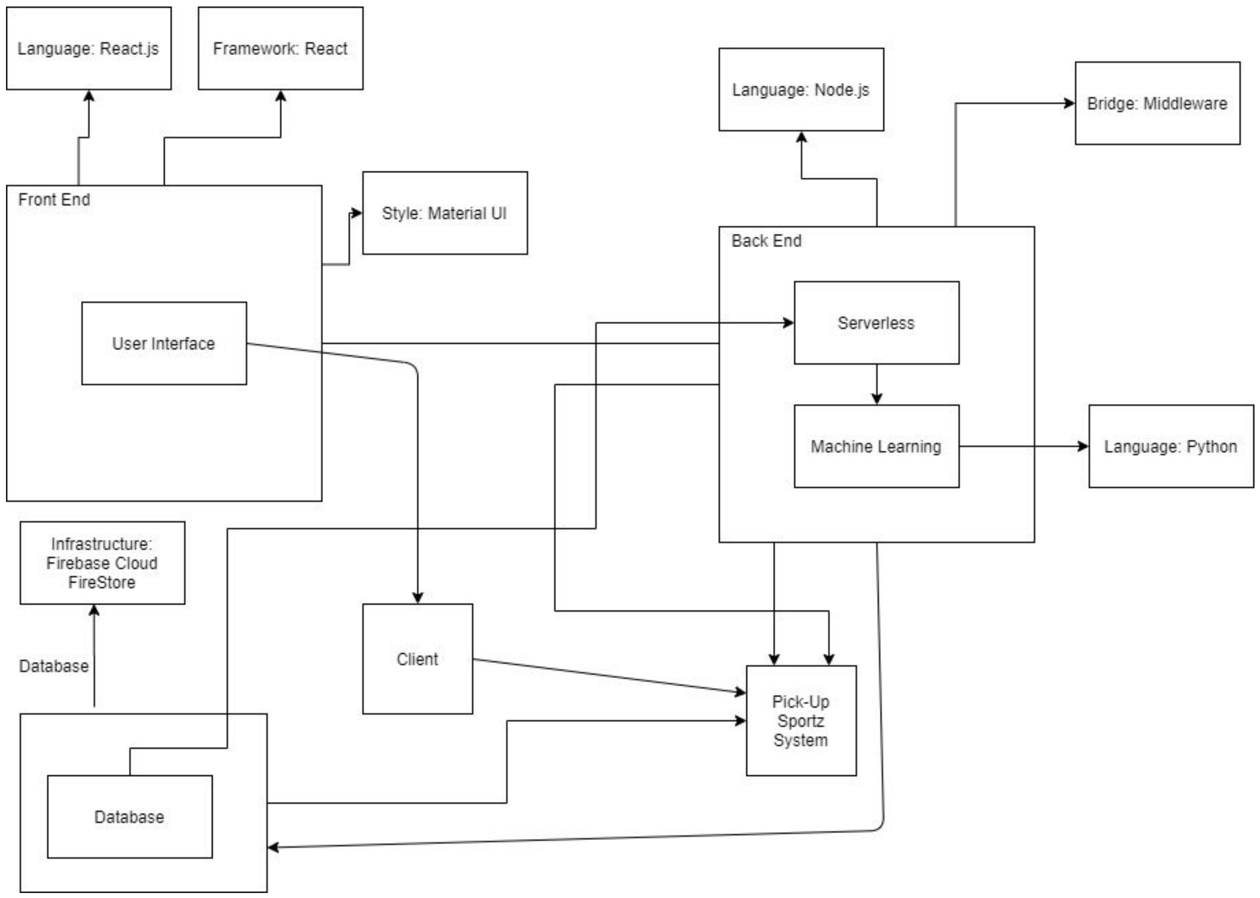


- Since the correlations range from +1 to -1, where nearing 0 means that they have least correlation and nearing +1 or -1 means that they are more highly correlated, we can see from the heatmap, The teal squares (0.0) are the least correlated and the red squares (0.8) are the most correlated. Therefore, for the top left corner of the heatmap, the label is correlated to the label. The center square of the heatmap correlates the weight with weights. The bottom right corner of the heatmap correlates the height to the height. We believe that, by just looking at the heatmap, we do not believe it will help us for our prediction. Since our model is training our datasets based on the features (weight and height) and the label (Label) where the weight and height coincides with many different sports, the performance is not going to be great.

- **Preprocessed Data**
  - There is no preprocessed data. As explained above, we generated our own data with realistic values after analysing realistic data from kaggle.

Approach to machine learning as a web service

- **Architectural diagram as where your machine learning model fits into your overall architecture**



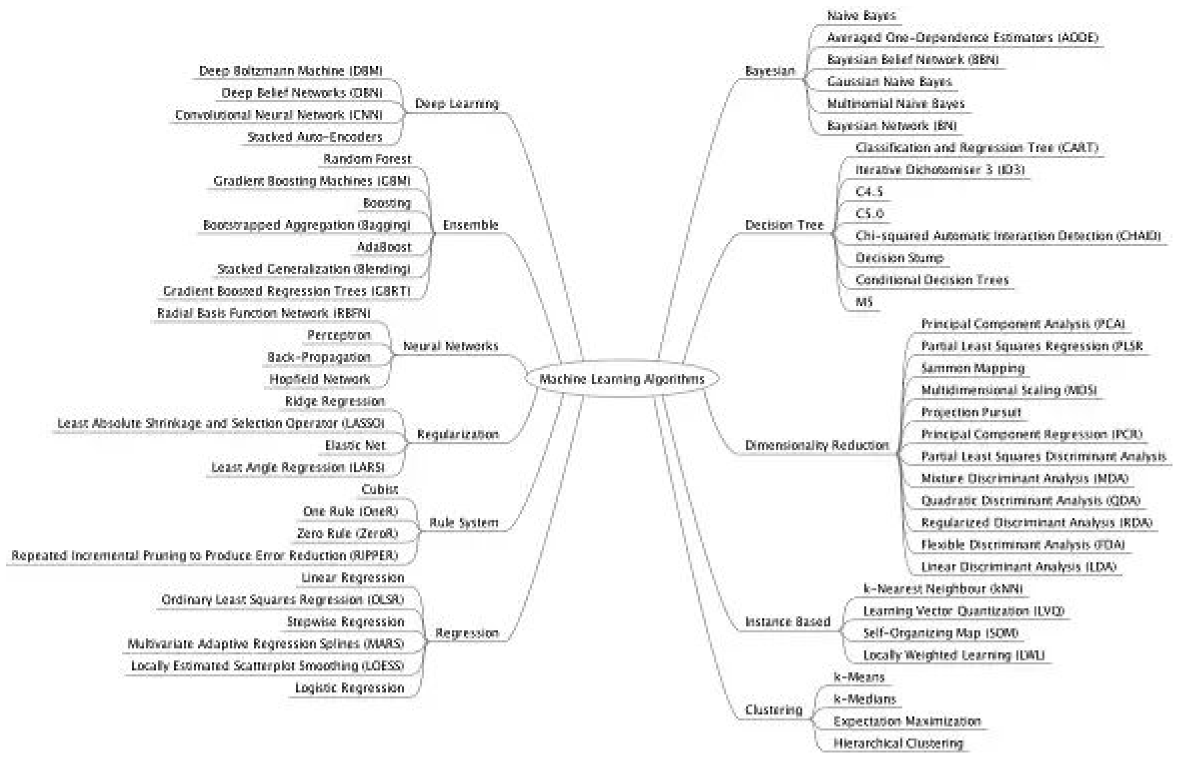
    - The machine learning model fits in the backend that communicates with our serverless components and our database.
- **API framework used, if any**
    - No API Framework used. We are importing libraries like pandas, sklearn, and tensor flow to code the model in javascript. We are planning to write the model in the same workspace as where we developed our web application.
- **Your servers if you have multiple servers**
    - No servers.
- **Containers if you have multiple containers.**
    - No containers.

Tradeoff analysis on model selection

Machine Learning Models	**Pros**	**Cons**
Neural Network	Flexible. Can Be Used for	Black boxes, meaning we

	both regression and classification problems. Any data which can be made numeric can be used in the model since the neural network is a mathematical model with approximation functions. Once trained, predictions are fast. Works best with more data points.	cannot know how much each independent variable is influencing the dependent variables. Computation is very expensive and time consuming to train with traditional CPUs. Depends heavily on training data. This can lead to overfitting and generalization.
Decision Tree	Supervised learning algorithm that works for both categorical and continuous dependent variables.Requires less effort for data preparation during pre-processing. Not require normalization and scaling of data. Missing values in the data does not affect the process of building decision trees to any considerable extent.	Small changes in the data can cause large changes in the structure of the decision tree causing instability. Calculations can go far more complex compared to other algorithms. Involves higher time to train the model. Relatively expensive.
Regression	Play with data without worrying about the intricate details of the model.	Oversimplifies many real world problems. More often than not, covariates and response variables do not exhibit a linear relationship.
Instance Based	Suitable when the examples are not all available from the beginning but are collected online. New examples observed require only an update to the database. Does not suffer from data interference, meaning that acquiring examples about an operating regime does not degrade modeling performance about others.	The definition of a distance metric to evaluate the relevance of the examples neighboring the query point. The selection of the structure of the local approximator. The choice of the number of examples to be used for each identification of a local model and their relative weight. The selection of the features to be considered. Large amount of memory to store the data. Each request for information involves starting the identification of a local model

		from scratch.
Dimensionality Reduction	Reflects our intuitions about the data. Allows estimating probabilities in high-dimensional data (no need to assume independence, etc.) Dramatic reduction in size of data (faster processing as long as reduction is fast). Smaller storage.	Too expensive for many applications (Twitter, web, etc.) Disastrous for tasks with fine-grained clases. Understand assumptions behind the method (linearity etc.)
Deep Learning	Features are automatically deduced and optimally tuned for desired outcome. Features are not required to be extracted ahead of time. This avoids time consuming machine learning techniques. Robustness to natural variations in the data is automatically learned. The same neural network based approach can be applied to many different applications and data types.	Requires a very large amount of data in order to perform better than other techniques. It is extremely expensive to train data to complex data models. There is no standard theory to guide you in selecting right deep learning tools as it requires knowledge of topology, training method and other parameters.


Machine Learning Algorithms
Deep Learning
Deep Boltzmann Machine (DBM)
Deep Belief Networks (DBN)
Convolutional Neural Network (CNN)
Stacked Auto-Encoders
Ensemble
Random Forest
Gradient Boosting Machines (GBM)
Boosting
Bootstrapped Aggregation (Bagging)
AdaBoost
Stacked Generalization (Blending)
Gradient Boosted Regression Trees (GBRT)
Neural Networks
Radial Basis Function Network (RBFN)
Perceptron
Back-Propagation
Hopfield Network
Regularization
Ridge Regression
Least Absolute Shrinkage and Selection Operator (LASSO)
Elastic Net
Least Angle Regression (LARS)
Rule System
Cubist
One Rule (OneR)
Zero Rule (ZeroR)
Repeated Incremental Pruning to Produce Error Reduction (RIPPER)
Regression
Linear Regression
Ordinary Least Squares Regression (OLSR)
Stepwise Regression
Multivariate Adaptive Regression Splines (MARS)
Locally Estimated Scatterplot Smoothing (LOESS)
Logistic Regression
Bayesian
Naive Bayes
Averaged One-Dependence Estimators (AODE)
Bayesian Belief Network (BBN)
Gaussian Naive Bayes
Multinomial Naive Bayes
Bayesian Network (BN)
Decision Tree
Classification and Regression Tree (CART)
Iterative Dichotomiser 3 (ID3)
C4.5
C5.0
Chi-squared Automatic Interaction Detection (CHAID)
Decision Stump
Conditional Decision Trees
M5
Dimensionality Reduction
Principal Component Analysis (PCA)
Partial Least Squares Regression (PLSR)
Sammon Mapping
Multidimensional Scaling (MDS)
Projection Pursuit
Principal Component Regression (PCR)
Partial Least Squares Discriminant Analysis
Mixture Discriminant Analysis (MDA)
Quadratic Discriminant Analysis (QDA)
Regularized Discriminant Analysis (RDA)
Flexible Discriminant Analysis (FDA)
Linear Discriminant Analysis (LDA)
Instance Based
k-Nearest Neighbour (kNN)
Learning Vector Quantization (LVQ)
Self-Organizing Map (SOM)
Locally Weighted Learning (LWL)
Clustering
k-Means
k-Medians
Expectation Maximization
Hierarchical Clustering

Description of your model

Our model will be based on the Decision Tree Machine Learning model. More specifically, the "Classification and Regression Tree".

Train model and Hyperparameter Optimization

We first split the training and testing data with the function train_test_split(X, y, test_size=0.1, random_state = 100), where X is the features (weight and height), y is the label (Label), test_size is the size of testing data we want which is 10% of the total dataset (90% of the dataset is training data), and random_state to randomly select the 10% testing set.

After getting our training and testing data, we then used the DecisionTreeClassifier function to train the data and predict the test data. We used the "gini" index, random_state = 100, max_depth = 20, and the min_samples_leaf = 1.

Does your model overfit or underfit?

We believe that our model underfits due the data coinciding with other sports, meaning that the weight and height of players for a particular sport is similar to the weight and height of players of other sports.

Accuracy Predicted and Expectations?

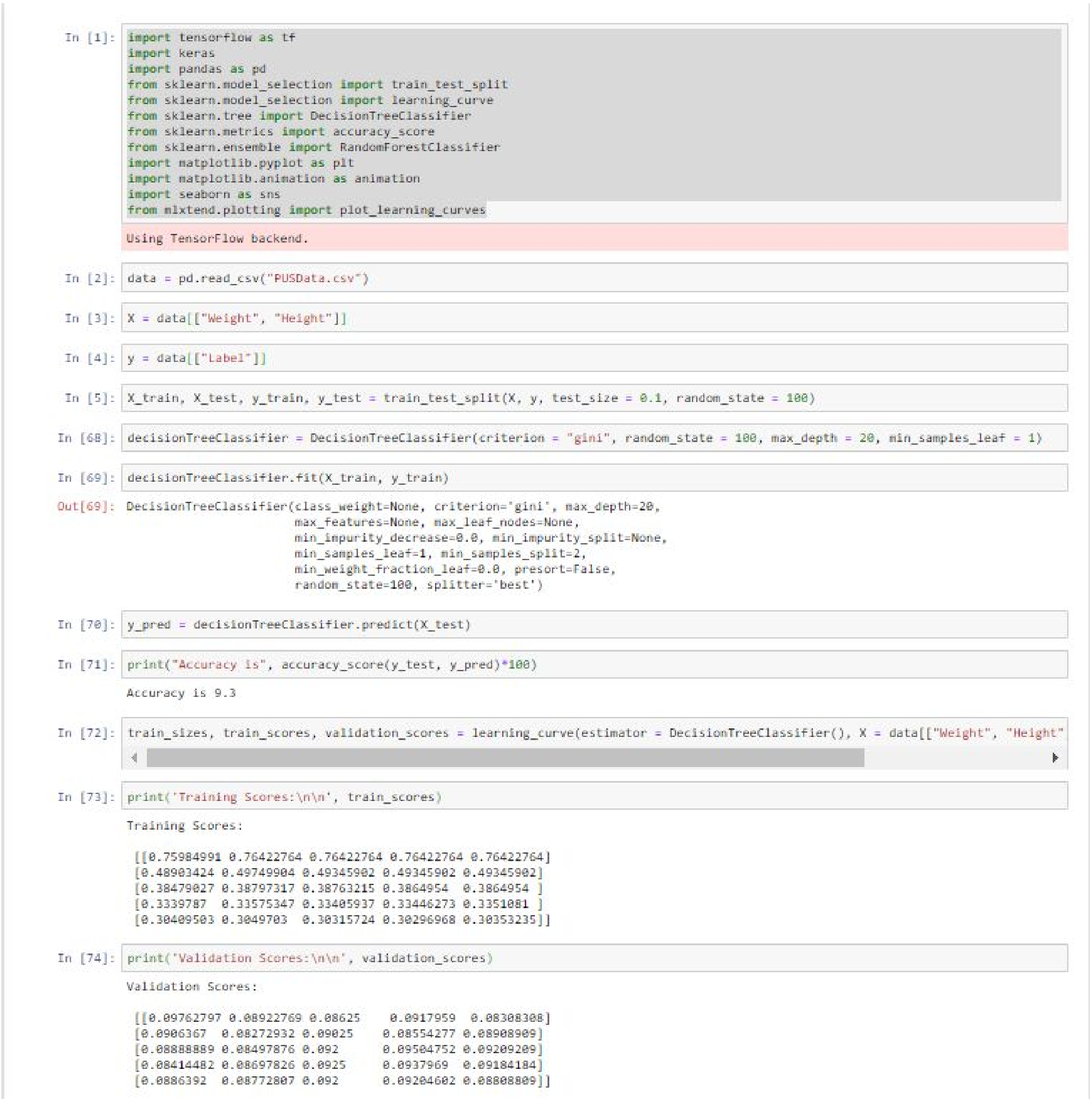
When using the decision tree classifier, the testing set accuracy is 9.3% and the training set accuracy is 27.22%. When using the random forest classifier, the testing set accuracy is 7.8% and the training set accuracy is 28.08%. In addition, for both classifiers, when increasing the depth, the accuracy does not improve and caps at a certain accuracy. For example, for decision trees, increasing the depth decreases the accuracy for the testing set to 8.75%. In addition, the decision tree classifier has a higher accuracy for both testing and training set compared to using the random forest classifier, thus, we are going to use the decision tree classifier as our machine learning model.

Deployment Considered?

We plan to deploy our ML model as a JavaScript language. We have the python version tested, but need to translate this into javascript so that it can run in our web application. There exists several libraries for react that can help us do this. The main essentials are npm pandas and npm decision tree. There are other optional libraries that can help with data manipulations such as test_train_split and read_csv. There are other alternative deployments, but we have not read much into it. We are still figuring out what is the best way to deploy that will not hinder the performance of our application.

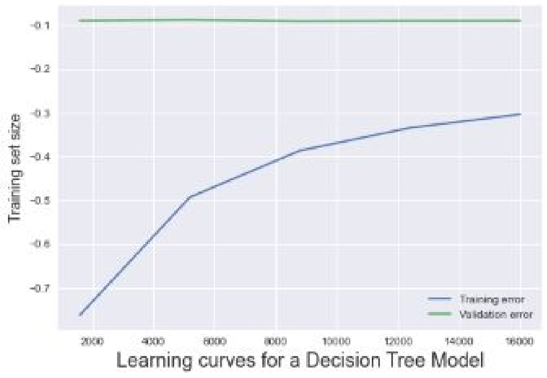
Making Predictions and Evaluating Results

For training and testing the dataset using the Decision Tree Classifier:



```
In [1]: import tensorflow as tf
        import keras
        import pandas as pd
        from sklearn.model_selection import train_test_split
        from sklearn.model_selection import learning_curve
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.metrics import accuracy_score
        from sklearn.ensemble import RandomForestClassifier
        import matplotlib.pyplot as plt
        import matplotlib.animation as animation
        import seaborn as sns
        from mlxtend.plotting import plot_learning_curves

        Using TensorFlow backend.

In [2]: data = pd.read_csv("PUSData.csv")

In [3]: X = data[["Weight", "Height"]]

In [4]: y = data[["Label"]]

In [5]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 100)

In [68]: decisionTreeClassifier = DecisionTreeClassifier(criterion = "gini", random_state = 100, max_depth = 20, min_samples_leaf = 1)

In [69]: decisionTreeClassifier.fit(X_train, y_train)

Out[69]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=20,
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, presort=False,
                     random_state=100, splitter='best')

In [70]: y_pred = decisionTreeClassifier.predict(X_test)

In [71]: print("Accuracy is", accuracy_score(y_test, y_pred)*100)

         Accuracy is 9.3

In [72]: train_sizes, train_scores, validation_scores = learning_curve(estimator = DecisionTreeClassifier(), X = data[["Weight", "Height"

In [73]: print('Training Scores:\n\n', train_scores)

         Training Scores:

          [[0.75984991 0.76422764 0.76422764 0.76422764 0.76422764]
          [0.48903424 0.49749904 0.49345902 0.49345902 0.49345902]
          [0.38479027 0.38797317 0.38763215 0.3864954  0.3864954 ]
          [0.3339787  0.33575347 0.33405937 0.33446273 0.3351081 ]
          [0.30409503 0.3049703  0.30315724 0.30296968 0.30353235]]

In [74]: print('Validation Scores:\n\n', validation_scores)

         Validation Scores:

          [[0.09762797 0.08922769 0.08625    0.0917959  0.08308308]
          [0.0906367  0.08272932 0.09025    0.08554277 0.08908909]
          [0.08888889 0.08497876 0.092      0.09504752 0.09209209]
          [0.08414482 0.08697826 0.0925     0.0937969  0.09184184]
          [0.0886392  0.08772807 0.092      0.09204602 0.08808809]]
```

```
In [75]: train_scores_mean = -train_scores.mean(axis = 1)
         validation_scores_mean = -validation_scores.mean(axis = 1)

In [76]: print('Mean Training Scores \n\n', pd.Series(train_scores_mean, index = train_sizes))

         Mean Training Scores

          1599    -0.763352
         5198    -0.493382
         8797    -0.386677
         12396   -0.334672
         15995   -0.303745
         dtype: float64

In [77]: print('Mean Validation Scores \n\n', pd.Series(validation_scores_mean, index = train_sizes))

         Mean Validation Scores

          1599    -0.089597
         5198    -0.087650
         8797    -0.090601
         12396   -0.089852
         15995   -0.089700
         dtype: float64

In [78]: plt.style.use('seaborn')
         plt.plot(train_sizes, train_scores_mean, label = 'Training error')
         plt.plot(train_sizes, validation_scores_mean, label = 'Validation error')
         plt.ylabel('Training set size', fontsize = 14)
         plt.xlabel('Learning curves for a Decision Tree Model', fontsize = 18)
         plt.legend()

Out[78]: <matplotlib.legend.Legend at 0x19b02fabda0>
```



```
In [79]: print("Test Accuracy:", decisionTreeClassifier.score(X_test, y_test))

         Test Accuracy: 0.093

In [80]: print("Train Accuracy:", decisionTreeClassifier.score(X_train, y_train))

         Train Accuracy: 0.2722777777777778
```

To make a prediction:

```
In [8]: userData = [[178 ,70]]

In [9]: y_pred = decisionTreeClassifier.predict(userData)

In [10]: y_pred

Out[10]: array(['BEACH_VOLLEYBALL'], dtype=object)
```

For a user with height = 70 inches and 178 pounds, we will use the classifier to predict the label for this particular user's and its metrics. In this case, it predicted Beach Volleyball for this user.

There are several issues with this prediction, such as, the metric for this user can correlate to other sports besides beach volleyball, however, but with the current dataset we have, this prediction is the best case for what we have for right now.

What are your performance metrics for your machine learning capability?

With about 20,000 datas in our dataset, the performance of our machine learning capability did not take long (took about 10 seconds). We think that it has to do with splitting the dataset (90% training data and 10% testing data)

Does the machine learning capability meet your business needs in terms of metrics? Why or why not?

We believe it fits our business needs in terms of metric since we have a web application and the time it takes for the model to recommend a sport is short.