This 4905 project is a website that will track a video game character’s appeal among the user base and relay that information to the client. In this case the video game characters are exclusively from the “Fire Emblem” series of games, but the basic skeleton of this project can be reapplied to other video game series as well. The final project is going to consist of 3 components:

* An API in the form of a JSON module storing all information pertinent to users and votes.
* A Javascript file that uses Express to interact with the API.
* A frontend framework(ReactJS) to send requests to the backend and display requested information to the user.

For the purposes of this project, just about any frontend JS framework capable of passing variables between pages and components can work. ReactJS has been chosen because my pre-existing familiarity with it will increase the chances that I’m able to complete the project within the allotted time frame.

The backend needs to be just robust enough to hold information for a long period of time, update that information based on actions taken by users, and have those updates be consistent between multiple visits to the site. It does not need to handle massive volumes of data, and will likely not need to host large userbases(if it gets deployed publicly, the most I can expect is dozens of concurrent users). Neither will it need to do particularly complex calculations. For that reason, I do not deem SQL databases, or dedicated backend languages such as Java and Python to be necessary for this project. For that reason, I have deemed a Node/ExpressJS backend to be the simplest way to handle API interaction.

***Basic User Functions(MVP)***

The user will be able to do all of the following:

* Create an “account” of some kind. There will be no need to gather sensitive information such as email address or phone number. The purpose of this function is to allow users to track and reverse their own activity.
* Navigate to a page dedicated to the character of their choice. They will be able to do the following on that page:
  + Cast a vote for the character. That vote is going to be in the form of a gauge from 1 to 5, with higher numbers representing more favorable opinions.
  + If they have already cast a vote for that character, change or retract their vote.
  + View the average rating that the character has received.
  + View a chart(likely a histogram) of all votes the character has received.
* View all characters from a specific game ranked by their average rating(for the MVP, there will only be one game).

***API***

The API will be maintained in a JSON module. It contains the following info:

* **Character:**
  + Each entry represents one character.
  + For the Minimal Viable Product, all characters in this table will come from a single game.
  + Attributes:
    - CharacterID- The primary key. A numeric representation of each character in the table.
    - Character- The name of the character.
    - GameNum- A number representing the game the character comes from. Since all games supported by this website will come from the same video game franchise, we can afford to use integers to represent their release order, since most users and by extension any potential admins will be able to identify the games from their GameNUm.
    - VoteAvg- The average of all Votes cast for the character. This average is calculated and updated every time a vote is made. This attribute is recorded in real time in order to shorten the time it takes to return the average upon request.
* **User:**
  + Each entry represents one user of the website.
  + This information is tracked in order to allow users to see their own votes and potentially change them.
  + Attributes:
    - UserID- The primary key. A username that’s unique to each user of the site.
    - Password- The user’s password.
    - Login- A boolean flag to determine whether the user is logged in or not. Votes must be attached to a user and so the system needs a way of knowing whether the user is logged in or not. A value of 1 means the user is logged in and a value of 0 means they are not.
* **Vote:**
  + Each entry represents one vote that gets cast.
  + One user can cast many votes, but each vote can only be cast by one user.
  + One character can receive many votes, but each vote can only be for one character.
  + Attributes
    - VoteID- The primary key. A unique integer identifier for each vote cast.
    - Character- Foreign Key to Character.CharacterID. The character that the vote is cast for.
    - User- Foreign Key to User.UserID. The user that cast the vote.
    - VoteNum- The value of the vote itself.

***MERN Functions***

The Java file(s) will have the following functions:

* Add User
  + Takes in a username and password.
  + Checks the “User” table for the username argument.
  + If the username is already taken, returns a specific integer signifying that.
  + If the password hasn’t met criteria, returns a specific integer signifying that.
  + If both previous checks are successful, adds new record to the “User” table and initializes “User.Login” to 1.
  + Returns 1 if the end of the function is reached.
  + Returns -1 if any extraneous errors are thrown.
* Login
  + Takes in a username and password.
  + Checks the “User” table for both arguments.
  + If there is no entry in the table matching the arguments, returns a specific integer to signify that.
  + If the user is already logged in, returns a specific integer to signify that.
  + Returns -1 if any extraneous errors are thrown.
  + If the user is logged out and login is successful, sets “User.Login” to 1 and returns 1.
* Check for Vote
  + Takes in a user and a character as arguments.
  + Checks the “Vote” table for both argument.
  + If there is an entry in the table matching the arguments, return a key:value pair containing the VoteID and the value of the vote. Otherwise return {0:0}.
  + Return {-1,-1} for any extraneous errors.
* Add Vote
  + Takes in a user, a character, and a vote value as arguments.
  + Inserts arguments into the “Vote” table.
  + Calculates the average of all votes for that character after adding the vote and updates the relevant value in “Character.VoteAvg”.
  + Returns 1 upon success and -1 upon failure.
* Modify Vote
  + Takes in a user, a character, and a vote value as arguments.
  + Updates records in the “Vote” table according to arguments.
  + Calculates the average of all votes for that character after adding the vote and updates the relevant value in “Character.VoteAvg”.
  + Returns 1 upon success and -1 upon failure.
* Delete Vote
  + Takes in a user, a character, and a vote value as arguments.
  + Removes records in the “Vote” table according to arguments.
  + Calculates the average of all votes for that character after adding the vote and updates the relevant value in “Character.VoteAvg”.
  + Returns 1 upon success and -1 upon failure.
* Get Average
  + Takes in a character as an argument.
  + Returns the average value of the votes that character has received according to the “Vote” table.
  + Returns 0 if there are no votes for that character.
  + Returns -1 upon failure.
* Get Votes
  + Takes in a character as an argument.
  + Compiles an array representing the amount of each kind of vote the character has received according to the “Vote” table.
  + Gets the average of that character’s votes using the Get Average function.
  + Returns a key:value pair of the array of votes and the average of votes.
  + Returns {[0]:0} if there are no votes for that character.
  + Returns {[0]:-1} upon failure.

***Scheduled Targets/Goals***

* February- Most frontend work, API construction.
* March- Get/Put/Post requests, remaining frontend work.
* April- Testing, Final Debugging, Styling
* May- Presentation Prep