## Project 1 Details

### 🔴 Mandatory To Pass:

#### MVP - Minimum Viable Product

#### Your application must meet these requirements:

1. Built with HTML, CSS, JavaScript and jQuery
   1. HTML: yep
   2. CSS: yep
   3. JavaScript: buttons and links
   4. jQuery:
2. Hosted on github pages:
   1. Absolutely!
3. Frequent Commits to github
   1. Hahahaha I love to cover my butt!!!
4. A README.mdfile with explanations of the technologies used, the approach taken, a link to your live site, installation instructions, unsolved problems, etc.
   1. Tech used:
   2. Link to live sites:
   3. Future updates:
5. Use AJAX to make a request to an external data source like OMDBapi and insert some of the data retrieved into the DOM
   1. ?

**Suggested Steps to Get Started**

The following approach has been proven to help students write complex front-end web apps, such as games.

If you're concerned that using the following approach will result in you and your fellow students having code that is structured similarly - don't be! **What matters is what prospective employers think when they look at your projects's code structure in GitHub!**

1. **Analyze the app's functionality**
   * The app's features, from the user's point of view, should be described using *User Stories*. User stories follow this template: **As a [role], I want [feature] because [reason]**. Example user story: *As a player, I want to see a list of high-scores so that I know what I have to score to make the list*.
   * As a [visitor to site] I want [to see pictures of pups], because [they make my day].
2. **Think about the overall design (look & feel) of the app**
   * Take the users (audience) of the app into consideration.
   * Wants a quick way to see pictures of pups or specific breed pictures
   * Should the app have a clean/minimalist UI (current trend), or should it be themed to match the app's purpose?
   * Simple and all on one page
3. **Wireframe the UI**
   * Wireframes provide a blueprint for the HTML & CSS.
   * Wireframes also help reveal an application's data (state) and functionality.
   * MADE
4. **Pseudocode**
   * Some of the app's features may need to be pseudocoded, that is, outlining the app's logic in a plain, informal way.
   * Pseudocode the app's **overall** functionality first.
   * More detailed pseudocode for a given feature may be required later.
   * ?
5. **Analyze the application's state (data)**
   * What does the application need to "remember" throughout its execution?
   * Doesn’t need to remember?
   * Use the wireframe(s), user stories and pseudocode to help determine what state needs to be tracked.
   * ?
   * Note that in an application with a database, we would analyze the database design during this step.
6. **Set up the project**
   * Create project directory OUTSIDE of any existing git repo (nested repos cause problems).
   * ?
   * Create the starting project files. Here's a possible structure:
     + **index.html**
     + **css/main.css**
     + **js/main.js**
   * Create the HTML boilerplate within **index.html**.
   * Link **main.css** in the <head>.
   * Loading **main.js** just above the closing </body>tag ensures that the DOM is ready before the script runs. Alternatively, if you want to put the script tag in the <head>, add a deferattribute as follows:  
     <script defer src="js/main.js">  
     Be sure to load libraries such as jQuery before the app's scripts.
7. **Create a local repo**
   * $ git init
   * Create a remote repo in your GitHub account then follow the instructions that GitHub provides to add a remote to link your local repo to GitHub.
   * It is recommended that the name of the repo and the project directory match.
8. **Organize the app's JS into sections**
   * Adding comments such as the following will help you organize your app's code:  
     /\*----- constants -----\*/  
     /\*----- app's state (variables) -----\*/  
     /\*----- cached element references -----\*/  
     /\*----- event listeners -----\*/  
     /\*----- functions -----\*/
9. **Code away!**
   * Iterating between adding HTML, CSS & JS is one approach.
   * Start with some markup for the basic layout of the UI.
   * Declare, but don't initialize, the application-wide variables (state). The initialization of the variables to their "start-up" state should be done within an initialize, or similarly named function, i.e., init, reset, etc.
   * Write that initializefunction.
   * Invoke initialize()to "kick off" the app.
   * Next stub up a renderfunction. Be sure to call renderafter state has been updated in event handlers, the initializefunction, etc.
   * Register event listeners - browser apps are typically *event-driven*.
   * If you have user stories, code them in a logical order.
10. **More recommendations for interactive browser app's, such as games**
    * Create a main renderfunction that is responsible for rendering the state of the app to the DOM.
    * If the renderfunction becomes lengthy, add additional rendering oriented functions, for example:
11. function render() {
12. renderHands();
13. renderControls();
14. if (winner) {
15. renderWinnerMessage();
16. } else {
17. renderTurnMessage();
18. }

}

* + Avoid accessing the DOM from outside render-oriented functions. However, "eye candy" animations, a ticking time display, etc. are exceptions to this tip.
  + **Data (state) is the single source of truth of the app** - when implementing an app's logic, the DOM is secondary to data manipulation. **Get used to thinking about how to your app's data changes vs. the display.**
  + As the user interacts with the application (or other events such as timers trigger), code the app such that it:
    - Updates state, then...
    - Calls render()

1. **Make frequent git commits of working code**
   * At a minimum, commit each "milestone" or feature implementation.
2. **Experiment and refactor code as necessary**
3. **Have fun!**