Project 3 – Guess-A-Word Game

Document Revision 1.0

Expectations

- Instructor Guidance in Class – Moderate
- Independent Effort – **High**
- Originality – Low
- Teamwork – *None*

Overview

In this project students will complete a Guess-A-Word game using a web service API that has been provided to them. Students will be responsible for the creation of the functionality around the provided web service, providing users with the ability to sign up to play the game, log in, and see some stats about how well they have played.

Please note that the "stats" feature described here is a modest introduction to the idea of Gamification.

At the end of this project, the student solutions will use two Web Service endpoints. One endpoint will be the work of the student, hosted on AWS. The other endpoint will be the wordgame web service endpoint provided by the instructor, hosted on misdemo.temple.edu.

Scenario – the Guess-A-Word game

Like other, similar, games the players in the Guess-A-Word game guess one letter at a time. Every time a player makes a guess, it is put into the provided API and the game record is updated. The API responds with a game status that can be presented to the user.

Players are allowed up to 6 wrong guesses. Only wrong guesses count against the number of allowed guesses. If the player completes the word with fewer than 6 wrong guesses, the player wins the game. If the player exhausts all 6 of the allowed wrong guesses without completing the word, then the player loses.

All this fancy logic has been created for you and is implemented in this web service:

https://misdemo.temple.edu/wordgame

Disclaimer

This document, and the project itself, will evolve over time. That is the nature of programming and/or software development projects. Requirements are assessed, prototypes are built, the quality of the prototype is evaluated, and then the whole process repeats again and again until you have something called a "minimum viable product" (MVP). The MVP has "just enough" features to be useful.

The MVP is launched and as demand, time and budget dictate, the whole development process starts over.

This iterative process is called the "Agile Methodology".

CONTINUED…

Instructions

Together as a class

A good amount of planning has already been done for you. Here's a review of what has been given to you. The following items have been/will be discussed in class and they can be found on the MIS Community Site!

- The UI Sketch
- Some "Proof of Concept" code that illustrates how the game is played and how the web service works.
- Lambda code for your Web Service to get you started.
- A SQL script containing the CREATE TABLE statements that your solution needs.

Steps

1. As a class, create a "gold" database on the class database server.

2. As a class, unzip the proof of concept code found in project3-poc-new.zip, rename the folder to "project3wordgame".
    a. Validate that the "proof of concept" code works in Chrome.
    b. Validate that your new project3wordgame folder appears in your S3 Bucket in Visual Studio code.
3. As a class, download the project3shafer-new.zip file and upload it to Lambda.
    a. Change the timeout from 3 seconds to 3 minutes in General Configuration
    b. Put your own "gold" database credentials into index.js
    c. Set up the REST API gateways (for both the "root" endpoint and the proxy endpoint.)
    d. Test your work with Thunder Client
    e. Review how the Lambda code is working. And how it can be tested.

On your Own

4. Reference the UI sketch.
5. It would be smart to first create a non-functional prototype… just so you can be sure that your HTML is good, that you know what HTML ids you are using, and that you know how the solution looks from the user's point of view.
6. Integrate the login and signup features of the API you were given so that users can login and sign up. (You may wish to copy / paste material from your earlier work to speed this along.)

   HINTs: When a user logs in, the usertoken will be provided. You need to use that 36 character usertoken to make a new game. *Put that usertoken in a form so that you can serialize it later.*

   It would be smart to create a loginController as well as a newGameController. When login is successful, call the newGameController. In that controller issue a POST to the wordgame endpoint. You will get a gameid out of that. *Put that gameid in a form so that you can serialize it later.*

   When the newGameController is successful, direct the user to the interface (div-play) that will allow them to play the game.

7. To create the stats feature, you will first need to capture the outcomes of the games. You can't report on data you have not collected. You will need to build the Lambda web service feature that posts new data to the history table. And, you will need to create a form on the client side to hold things like the game status ("Win" or "Lost") that you will need to POST to your history feature.
8. You will also need to create features that can be used to retrieve data from the history table. There is more than one right way to do this.

   I would suggest that you create one feature for every piece of data you need. (That is, a feature for the total number of games played by the user, a feature for percentage wins, a feature for percentage losses, and a feature for the current champion.)

   This strategy will allow you to earn partial credit … just in case you can't get all the features working.

   a. Be sure to consider the scenario when no games have been played. (You want to avoid divide-by-zero errors.)
   b. Be sure to document any features you make so that they appear at the "root" URL for your endpoint.

Turning in your work

As in project 2, a separate document will be provided that will outline how you should turn in your work as well as a checklist of things to check before turning in your work.

In that document, a video will be provided that illustrates what a good/ideal solution looks like. Don't wait for it though. You have enough resources and knowledge to get started.

How this project will be graded

In this project, your "base" project grade will be as follows:

100%	The instructor can sign up to use your word game, play the game successfully, and see the stats related to the game. (Total games played by the user, percentage wins, percentage losses, and current champion.)
95%	The instructor can do all the above, but one of the stats was not implemented.
90%	The instructor can do all the above, but two of the stats were not implemented.
85%	The instructor can do all the above, but three of the stats were not implemented.
80%	No stats were implemented. The instructor can sign up to use your word game, and play the game successfully.
70%	Either signup / login fails, or game play fails.
50%	Solution does not work.

Please be advised that points may be deducted from your "base" project grade for things like bad HTML, critical A11y errors, incomplete / incorrect visual appearance. These items will appear in the "Checklist Items" list that will be provided in the "Turning in your work" document for Project 3.