Flask Boggle

Download starter code

In this exercise, you plan & help code a Boggle game in Javascript and Python.

The Game

The goal of the game is to get the highest point total. To gain points, players create words from a random assortment of letters in a 5x5 grid. We will be providing the functionality to generate the grid.

Words can be created from adjacent letters – that is, letters which are horizontal or vertical neighbors of each other as well as diagonals. The letters must connect to each other in the proper sequence to spell the word correctly. This means that the next letter in the word can be above, below, left, or right of the previous letter in the word (excluding any letters previously used to construct the word). We will also be providing functionality to determine if a word can be constructed from a boggle board.

Warning: Writing Tests

A main focus of this exercise is on testing Flask. Make sure you **write tests for all views you add to app.py**.

Step One: Planning and Reading Code

Before looking at our code

- Take a look at the first four functions in the ***boggle.py*** file. Don't worry about the ***find*** and ***find_from*** functions. You will be using them, but you don't need to understand in depth what they do!
- Take a look at the ***app.py*** file and you will notice there is no Flask related code. Add some! As usual, this is where your routing logic should go.
- For your CSS and JavaScript, make sure you have a ***static*** folder, and make sure to create a separate file for your tests.

Step Two: Displaying the Board

- The first thing you need to do is display the board in a Jinja template.
  - You will be generating a board on the backend using a function from the ***boggle.py*** file and sending that to your Jinja template.
  - Using Jinja, display the board on the page.
- Since you will also be using this board in other routes, make sure to place it in the session.
- Once you have displayed the board, add a form that allows a user to submit a guess.

Step Three: Checking for a Valid Word

- Now that you have a form built: when the user submits the form, send the guess to your server.
  - The page should not refresh when the user submits the form: this means you'll have to make an HTTP request **without** refreshing the page—you can use AJAX to do that!
  - Make sure you include axios so that you can easily make AJAX requests.
  - Using jQuery, take the form value and using axios, make an AJAX request to send it to the server.
  - On the server, take the form value and check if it is a valid word in the dictionary using the ***words*** variable in your ***app.py***.
  - Next, make sure that the word is valid on the board using the ***check_valid_word*** function from the ***boggle.py*** file.
  - Since you made an AJAX request to your server, you will need to respond with JSON using the ***jsonify*** function from Flask.
  - Send a JSON response which contains either a dictionary of ***{"result": "ok"}***, ***{"result": "not-on-board"}***, or ***{"result": "not-a-word"}***, so the front-end can provide slightly different messages depending if the word is valid or not.
- On the front-end, display the response from the backend to notify the user if the word is valid and exists on the board, if the word is invalid, or if the word does not exist at all.

Step Four: Posting a Score

- Now that you are checking to see if a word is valid, let's start keeping score! The score for a word is equal to its length. If a valid word is guessed, add its score to the total and make sure to display the current score as it changes.
- You can store the score on the front-end for now: it does not need to persist.

Step Five: Adding a timer

Instead of letting the user guess for an infinite amount of time, ensure that a game can only be played for 60 seconds. Once 60 seconds has passed, disable any future guesses.

Step Six: More statistics!

Now that you have a functional game, let's keep track of how many times the user has played as well as their highest score so far. When the game ends, send an AJAX request to the server with the score you have stored on the front-end and increment the number of times you have played on the backend.

Since you will be sending this data as JSON when you make an axios request, you will see this data come in your Flask app inside of ***request.json*** *not* ***request.form***. Use pdb or IPython to set a breakpoint and see what ***request.json*** looks like, it is *not* the same data structure as ***request.form***.

Step Seven: Refactoring

Take a look at your code and see if there are opportunities to refactor!

- If you have not already, can you design your front-end in an Object Oriented way?
- Do your view functions have docstrings that describe what they are doing?
- Are you handling duplicate words? Make sure if you submit the same word, it does not count twice.

Further Study: More challenges

- Better style your application! Make this something you'd be proud to show!
- Before the game is played, display a form that allows the user to create a boggle board of N*N size. This will require you to change the ***boggle.py*** class.
- Try to implement a hint feature! Add a button to ask for a hint, which will then highlight the first couple of characters in a valid word that the user hasn't discovered yet.

Solution

You can see Our solution.