Homework 3: Django Wordish Game

Due date: September 19, 2022 at 11:59pm

In this homework, you will extend your first two homework assignments to implement the Wordish Game by sending web requests to a Django application running server-side. The game should appear to work in (approximately) the same manner as it did your second homework. This assignment introduces Django, a popular web framework written in Python, which you will use for this homework and the next series of homework assignments.

For this assignment, you will use the front end you created in the first assignment (though you can make improvements) and add functionality by writing the server-side routes that render and perform the necessary game logic. (You must implement the game logic in Python on the server, not in JavaScript in the browser.)

The learning goals for this assignment are to:

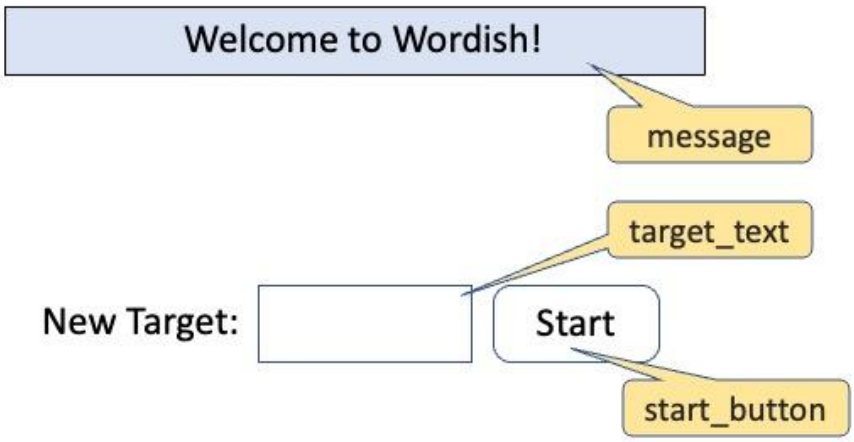
- Learn how requests are routed and processed in a typical MVC web application.
- Demonstrate an understanding of typical data interactions between a web client and a web server, including the difference between HTTP GET and POST parameters and the use of HTML forms as input to a stateless web application.
- Demonstrate thorough, manual validation of HTTP request parameters by a web application.
- Gain hands-on experience with Django, a production quality web framework.

Specification

This section describes the behavior of the game that you will implement. The game itself should behave the same as the JavaScript Wordle game you implemented in Homework #2, with the difference that users now have to input the target word on a separate page instead prompting for it on the game page as we did in Homework #2. We'll call this page the "start" page.

After specifying the target word and clicking "Start", the "game page" is displayed and the user may start guessing. Details of the start page are described in the Requirements section below.

If your server actions detect inconsistencies in the data sent with a request that are so severe that the game cannot continue, you must display an appropriate error message on the start page and permit the user to specify a new target word and start a new game.



Requirements

Your submission must follow these requirements:

- A GET request for the empty URL (i.e. http://localhost:8000/ if the server is run on port 8000) must route to an action function (in `views.py`) which will return the start page.
- The start page is sketched above. It must contain:
    - A `<div>` element with `id="message"` which is used to specify a welcome message or an error message.
    - A `<form>` containing an `<input>` with `id="target_text"` and a `<button>` with `id="start_button"` which will be used to set the target word and start the game.
    - Optionally, a way to start the game with a random word, perhaps by letting the user click the start button with no target word specified, or perhaps by adding an additional button. (We won't test this functionality.)
- Submitting the form on the start page (with correct data for #target_text) will route to an action function which will return the game page.
- Submitting the form on the game page (with correct data) will route to an action function which will compute the new game state.
    - The game page will be returned showing the updated game state.
    - If the user has won or lost the game, the game page will display a message indicating whether the user has won or lost. (Optionally, you may disable the button. Optionally, you may add an href to the game page which links back to the start page so the user can start a new game. But AutoGrader will not check these things.)

- Your submission must follow all of the requirements specified in the first homework assignment.
- Your application must run on the web server. The web browser must simply submit requests to the web server (after each button click) and display the response; the client may not use JavaScript or perform any processing of the user inputs.
- You are not to store anything in the database. The server should be *stateless*: the server may not store any data between web requests. To meet this requirement your server will need to send extra parameters (in addition to the displayed game status) to the client with each response. The client will need to resubmit this extra data with its next request.
- Your application may not crash as a result of any input sent to the server-side or because of any actions the user performs. Note that the user can send any data (malformed or otherwise) to the server, and you must anticipate and validate these requests.
- Messages about invalid target words must be displayed in `#message` on the start page.
- Messages about invalid guess words must be displayed in `#status` on the game page.
- Messages about invalid game state (due to hacked fields) must be displayed on the start page.
- You may not use any external libraries (e.g. jQuery, Bootstrap, etc.).
- You may not use any Python packages other than the standard Django packages.
- Your application should run with Django 4. See the Django Quick Start guide posted on Canvas for info on how to use pip to install the correct version.
- Cite all resources used and any additional notes you would like to convey to us in the `hw3/README.md` file using the format described in the HW#1 specification.

Configuring the .gitignore

You should update your `.gitignore` file to exclude additional files your system may create in your repo, but that should not be committed. Specifically, any virtual environments (e.g., `my_env` or `venv` folders), compiled Python bytecode files (`*.pyc`), database files (`db.sqlite3`), editor swap/backup files (e.g., `*.swp` or *~), and IDE config files should be excluded.

We have created a `.gitignore` file at the top of your repository. Take a look at it. Notice it specifies the exclusion of several of the files listed above. If you have additional files that should not be committed to your repo you should add these files (or folders) to your `.gitignore` file.

So, if you create a virtual environment into your repo, be sure that the name of the virtual environment directory (e.g., `my_env`) is in your `.gitignore`.

Turning in your work

Your submission should be turned in via Git and should consist of a Django project/application. Please put your code in the "hw3" directory. Use the following commands to create the files:

```
cd hw3
django-admin startproject webapps .      <= Notice the "."
python3 manage.py startapp wordish
```

The "." at the end of the "startproject" command will put the project in the current directory.

Here is an example directory structure (some directories/files omitted) of a submission.

```
[YOUR-ANDREW-ID]/hw3/
    |-- webapps/
        |-- settings.py
        |-- urls.py
        |-- [etc.]
    |-- wordish/
        |-- migrations/
        |-- static/
        |-- templates/
        |-- __init__.py
        |-- admin.py
        |-- apps.py
        |-- models.py
        |-- tests.py
        |-- views.py
        |-- [etc.]
    |-- manage.py
    |-- README.md
```

Grading

Same as before: You must run the grading script on https://grader.cmu-webapps.org to get credit for this assignment. You may correct, resubmit, and regrade your homework, up to the deadline, to get all the points. You may submit after the deadline, as per the homework late policy in the syllabus.