Kenytt Avery, Computer Science Department, California State University, Fullerton

CPSC 449 - Web Back-End Engineering

Project 1, Fall 2022 - due October 22

Last updated Tuesday October 4, 1:20 PDT

Wordle is a popular online word game built by a single developer as a side project and recently sold to the New York Times for an undisclosed sum. If you have not played Wordle, first take a break and try it, then come back and finish reading this document.

In this project you will work with a team to build a back-end API for a game similar to Wordle.

The original game is entirely a client-side application, with both current and future answers stored in the JavaScript code, but the New York Times has begun to add server-side features, including syncing progress across devices.

Additional features that might be added with access to a back-end include:

- Making it harder to cheat
- Offering more than one game per day
- Offering different games to different users

Learning Goals

The following are the learning goals for this project:

1. Determining the back-end functionality required for a web application based on its front-end interface.
2. Designing a back-end API for a web application given a description of its functionality.
3. Implementing back-end APIs in Python with the Quart web framework.
4. Designing and implementing relational database schemas for back-end APIs.
5. Extracting, transforming, and loading database tables from data sources in other formats.

Project Teams

This project must be completed in a team of three or four students. The instructor will assign teams for this project in Canvas.

See the following sections of the Canvas documentation for instructions on group submission:

- How do I submit an assignment on behalf of a group?

Platform

Per the Syllabus, this project is written for Tuffix 2020. Instructions are provided only for that platform. While it may be possible for you to run portions of this project on other platforms, debugging or troubleshooting any issues you may encounter while doing so are entirely your responsibility.

Note: this is a back-end project only. While your browser may be able to display JSON objects returned in response to an HTTP GET request, you do not need to implement a front-end or other user interface for testing API methods. Use an HTTP client program such as HTTPie or curl, or the automatic documentation created by Quart-Schema.

Libraries, Tools, and Code

This project must be implemented in Python using the Quart framework and ancillary tools such as Foreman and the sqlite3 command line tool. Database code must use the Databases library with SQLite. Your queries may use either raw SQL or SQLAlchemy core; you may not use the SQLAlchemy ORM.

You may also use Quart-Schema to validate input and automatically generate documentation.

Code from the Python documentation, the library documentation, examples provided by the instructor, and *A Whirlwind Tour of Python* may be reused. All other code must be your own original work or the original work of other members of your team.

API

Create a RESTful service that exposes the following resources and operations:

Users

Each user should have a *username* and a *password*. This is only a game, and passwords are only used to keep track of the current game and store a user's statistics, so they may be stored in cleartext. If this makes you nervous, you may use Simon Willison's Password hashing in Python with pbkdf2 for a reasonable implementation using only the Python Standard Library.

The API should allow:

- Registering a new user
- Checking a user's password

The password-checking endpoint should require HTTP Basic authentication by checking the `request.auth` object's `type`, `username`, and `password` properties. If the supplied username and password are valid, return HTTP 200 and the following JSON object:

```
{ "authenticated": true }
```

If the username and password are not valid, return HTTP 401 and a `WWW-Authenticate` response header. See the `https://httpbin.org/basic-auth/{user}/{passwd}` endpoint for a working example..

Note: The httpbin.org endpoint specifies the required username and password as path variables in the URL for the resource. Your service should require the username and password of a registered user instead.

Games

As described above, we want to offer users the ability to play more than one game per day, and for different users to have different words in each game. Therefore a game must keep track of the *user*, the secret *word* for the game, and the number of *guesses* the user has made.

The API should allow:

- Starting a new game for a user with a randomly-chosen word
- Guessing a five-letter word
- Listing the games in progress for a user
- Retrieving the state of a game in progress

The secret word should not be sent to the client, only an identifier for the game. A user may start more than one game without finishing.

Games are finished either when the user has guessed the secret word, or when the user has made 6 incorrect guesses.

An incorrect guess should return the following:

- Whether the guess was a valid word.
- If the guess was valid, whether the guess was correct (i.e. whether the guess was the secret word).
- The number of guesses remaining. (Only valid guesses should decrement this number.)
- Vaid but incorrect guesses should also return:
    - The letters that are in the secret word and in the correct spot
    - The letters that are in the secret word but in the wrong spot

When supplied with an identifier for a game that is in progress, the user should receive the same values as an incorrect guess, except that the number of guesses should not be incremented. If the identifier corresponds to a game that is finished, return only the number of guesses.

Service Implementation

Use Quart to define endpoints and representations for the resources above. Each endpoint should make appropriate use of HTTP methods, status codes, and headers.

Your APIs should follow the principles of RESTful design as described in class and in the assigned reading, with the exception that all input and output representations should be in JSON format with the `Content-Type:` header field set to `application/json`.

Statelessness

Your service should be stateless. In particular, each request to a game resource will need to identify the user, and guesses and requests for game state will also require a game identifier..

Databases

Informally, your database schemas should be in approximately third normal form. If you are not familiar with database normalization, see Thomas H. Grayson's lecture note Review: Database Design Rules of Thumb from Course 11.521 at MIT.

Database Population

Populate the answers database directly from the Wordle script. You can download a copy of the JavaScript code and save only the correct answers and valid guesses with the following commands:

```
wget
https://www.nytimes.com/games-assets/v2/wordle.9137f06eca6ff33e8d5a306b
da84e37b69a8f227.js

sed -e 's/^.*,ft=//' -e 's/,bt=.*$//' -e 1q
wordle.9137f06eca6ff33e8d5a306bda84e37b69a8f227.js > correct.json

sed -e 's/^.*,bt=//' -e 's/;.*$//' -e 1q
wordle.9137f06eca6ff33e8d5a306bda84e37b69a8f227.js > valid.json
```

You can load the resulting files into Python using the `json` module from the Python Standard Library.

Submission

Part 1 - Submit the code

Your submission should consist of a tarball (`.tar.gz`, `.tgz`, `.tar.Z`, `.tar.bz2`, or `.tar.xz`) file containing the following items:

1. A README file identifying the members of the team and describing how to initialize the database and start the service.
2. The Python source code the service.
3. A `Procfile` definition for the service.
4. Initialization and population scripts for the database.
5. Any other necessary configuration files.

Note: Do **not** include compiled `.pyc` files, the contents of `__pycache__` directories, or other binary files, including SQLite database files. If you use Git, this includes the contents of the `.git/` directory. See Git Archive: How to export a git project for details.

If you use Quart-Schema to create a `/docs` endpoint, you do not need to write separate documentation for the APIs, but be sure to name functions and parameters so that they are represented appropriately in that interface..

Submit your tarball through Canvas by the due date. Only one submission is required for a team.

The Canvas submission deadline includes a grace period. Canvas will mark submissions after the first submission deadline as late, but your grade will not be penalized. If you miss the second deadline, you will not be able to submit and will not receive credit for the project.

Reminder: do not attempt to submit projects via email. Projects must be submitted via Canvas, and instructors cannot submit projects on students' behalf. If you miss a submission deadline, contact the instructor as soon as possible. If the late work is accepted, the assignment will be re-opened for submission via Canvas.

Part 2 - Team member evaluation

A Canvas Quiz due the week after the project will ask you to provide candid feedback on the relative contributions of each member of the team.

Grading

The project will be evaluated on the following five-point scale, inspired by the general rubric used by Professor Michael Ekstrand at Boise State University:

Exemplary (5 points)

Project is a success. All requirements met. Quality of the work is high.

Basically Correct (4 points)

Project is an overall success, but some requirements are not met completely, or the quality of the work is inconsistent.

Solid Start (3 points)

The project is mostly finished, but some requirements are missing or the quality of the work does not yet meet professional standards.

Serious Issues (2 points)

The project has fundamental issues in its implementation or quality.

Did Something (1 point)

The project was started but has not completed enough to fairly assess its quality, or is on entirely the wrong track.

Did Nothing (0 points)

Project was not submitted, contained work belonging to someone other than the members of the team, or was of such low quality that there is nothing to assess.