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

CPSC 449 - Web Back-End Engineering

Project 2, Spring 2022 - due April 8

Last updated Tuesday March 8, 1:45 am PST

In this project you will build two RESTful back-end microservices.

The following are the learning goals for this project:

1. Designing back-end APIs for a web application given a description of their functionality.
2. Implementing back-end APIs in Python with the FastAPI web framework.
3. Designing and implementing relational database schemas for back-end APIs.
4. Extracting, transforming, and loading databases given data sources in other formats.
5. Creating Procfile descriptions for web service processes.

Project Teams

This project must be completed in a team of three or four students. All students on the team must be enrolled in the same section of the course. You are free to choose the initial members of your team, but the instructor may adjust membership as necessary to accommodate everyone.

Teams should be formed by the beginning of class on Monday, March 14.

In order to submit a project as a team, you must join a group for that project in Canvas:

- Several groups have been pre-created by the instructor, and your team must join one of these for your submission. Projects cannot be submitted using groups that you create yourself.
- The first team member to join will automatically be assigned as the group leader.
- Teams are specific to the project, so you must join a new group for each project, even if you worked with the same team on a previous project.
- If there are no empty groups available, email the instructor immediately to request a new group to be created.

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

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

Platforms

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.

Libraries, Tools, and Code

This project must be implemented in Python using the FastAPI framework and ancillary tools such as Foreman and the sqlite3 command line tool. Database code must use the `sqlite3` module from the Python Standard Library; you may not use an ORM library such as SQLAlchemy or Peewee.

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.

Application Domain

Over the next several projects we will build back-end services for a server-side game like Wordle, similar to the one that you designed in Project 1.

Services

Create two RESTful microservices, one for the word list of valid guesses and one for checking guesses against answers.

The word validation service should expose the following operations:

- Checking if a guess is a valid five-letter word
- Adding and removing possible guesses

The answer checking service should expose the following operations:

- Checking a valid guess against the answer for the current day.

  If the guess is incorrect, the response should identify the letters that are:

  - in the word and in the correct spot,
  - in the word but in the wrong spot, and
  - not in the word in any spot

- Changing the answers for future games.

Statelessness

Each service is independent: clients will need to validate guesses against one service before checking the answer against the other service.

Both services should be stateless. In particular, the answer checking service should not store the current day's answer on the server side, and should not track the number of guesses made by a client.

API Implementation

Use FastAPI to define RESTful resources for valid guesses and for answers. Your resources 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`.

Databases

Each service should have its own independent database. Services should be accessed only through their HTTP interfaces, and neither service should use the other service's database.

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 Relational Database Design: Rules of Thumb from the MIT OpenCourseWare for MIT Course Number 11.208. Note in particular that data items should be atomic: this means, for example, that lists of words should be stored as separate rows, not as a single column.

Database Population

Populate the word list database with five-letter words from `/usr/share/dict/words`, omitting capitalized words and words containing punctuation and non-ASCII characters.

If you are comfortable with UNIX command-line tools, you may wish to do this with a shell script and the sqlite-utils command-line tool. If not, consider the `fileinput` and `string` modules from the Python Standard Library.

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

```
$ curl --silent https://www.nytimes.com/games/wordle/main.bfba912f.js |
sed -e 's/^.*var Ma=//' -e 's/,Oa=.*$//' -e 1q > answers.json
```

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

Managing processes

Define your services in a single `Procfile` and use `foreman` to start them both. Set the port for each service to use Foreman's `$PORT` environment variable so that they will not conflict.

Submission

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 databases and start the services.
2. The Python source code for each microservice.
3. `Procfile` definitions for the service.
4. Initialization and population scripts for each database.
5. Any other necessary configuration files.

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.

You do not need to write separate documentation for the APIs, but be sure to name functions and parameters appropriately and test your services using the automatic documentation.

Submit your tarball through Canvas before 9:45 pm PST on the due date. Only one submission is required for a team.

The Canvas submission deadline includes a grace period of an hour. 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.

Grading

The project itself 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)

Code is correct and internal documentation is clearly written; organization makes it easy to review.

Basically Correct (4 points)

Results are reasonable, but the code is not easy to follow, does not contain internal documentation, or has minor mistakes.

Solid Start (3 points)

The approach is appropriate, but the work has mistakes in code or design that undermine the functionality of the result.

Serious Issues (2 points)

The work contains fundamental conceptual problems in procedure, design, or code such that it will not lead to a working system.

Did Something (1 point)

The solution began an attempt, but is either insufficiently complete to 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 submission was of such low quality that there is nothing to assess.

The individual contributions of team members will be confidentially evaluated by the rest of the team along the following axes:

- API implementation
- Database design and implementation
- Discussion, research, and problem solving
- Status updates and regular contact with the rest of the team
- Willingness to help other members of team, or take on unpopular jobs

Each team member will assign scores on a scale ranging from +2 (contributed the most) to -2 (contributed the least), subject to the constraint that the sum of contributions to each factor across all team members must be 0. (For example, if you were to evaluate one team member's contribution as +2, you would also need to evaluate another team member's contribution as -2, or evaluate two other team members as -1.)