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

CPSC 449 - Web Back-End Engineering

Project 4, Spring 2022 - due May 6

Last updated Thursday April 14, 12:55 am PDT

In this project you will use a NoSQL database to build a stateful back-end microservice and to improve the performance of an existing service.

The following are the learning goals for this project:

1. Gaining further experience with implementing back-end APIs in Python and FastAPI.
2. Learning to design a data model for a key-value store.
3. Exploring the features of the Redis NoSQL database.
4. Introducing polyglot persistence to a service by splitting data between data stores.
5. Scheduling recurring tasks using the cron scheduler.

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, April 19.

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

The requirements for this project are the same as for Project 3, with the addition of the Redis NoSQL database and client libraries.

Other than the sources specified in the previous project and the StackOverflow article referenced below, all other code must be your own original work or the original work of other members of your team.

Implementing a Service to Track State

While you can successfully combine the services from previous projects to check that users are making valid guesses, check those guesses to see if they are correct, and record wins and losses, so far there is no ability to limit users to the six tries required by the game, or to restore a game in progress.

Create a new microservice for tracking the state of a game. Your service should expose the following operations:

- Starting a new game. The client should supply a user ID and game ID when a game starts. If the user has already played the game, they should receive an error.
- Updating the state of a game. When a user makes a new guess for a game, record the guess and update the number of guesses remaining. If a user tries to guess more than six times, they should receive an error.

  Note: you do not need to check whether the guess is valid, if the guess is correct, or report on the placement of the letters in the answer. This functionality was completed in Project 2.
- Restoring the state of a game. Upon request, the user should be able to retrieve an object containing the current state of a game, including the words guessed so far and the number of guesses remaining.

As with Project 2, your service should be implemented using RESTful principles.

Storing data in a NoSQL database

Use Redis as the data store for your service. In addition to functioning as a key-value store, Redis provides other data types such as lists, sets, and maps.

Installing Redis

To install and start Redis on Tuffix, use the following commands:

```
$ sudo apt update
$ sudo apt install --yes redis
```

Then verify that Redis is up and running:

```
 $ redis-cli ping

PONG
```

To get started with Redis, see How to Use Redis with Python.

Installing Python libraries for Redis

You will need a Redis client library to access Redis from Python. While you are welcome to use aioredis for performance if you are comfortable with the `async` and `await` in Python, the standard redis-py library is more than sufficient, and later versions of redis-py than the one included with Tuffix 2020 actually merge the two. To install both redis-py and the high-speed Hiredis parser on Tuffix, use the following command:

```
$ sudo apt install --yes python3-hiredis
```

Data Model

Use appropriate Redis data types, including strings and lists to track the state of games.

Materializing Data from Views

Even before you sharded the win/loss database in Project 3, you may have noticed that there was an appreciable pause when querying the `wins` and streaks `views` to determine the leaderboards for wins and streaks. We can speed up this process using Redis.

Create a standalone command-line program (i.e. not a web API) in Python to connect to each of the win/loss shards and pull the top 10 entries from each view. Then connect to Redis and store each entry in a sorted set.

Note: this means that you will be storing 30 entries from each view (10 from each shard).

Scheduling updates with cron

The UNIX `cron` service allows you to schedule recurring jobs. (This facility is similar to the Windows Task Scheduler, but has been around 20 years longer.) Use this to run your Python script every 10 minutes, making sure that the statistics stored in Redis are never more than 10 minutes behind.

For a tutorial introduction to `cron`, see CronHowTo from the Ubuntu Community Help Wiki. To make sure that only one copy of your program is running, even if it hangs, see Prevent Running Of Duplicate Cron Jobs. If you have trouble figuring out crontab syntax, check out the Crontab Generator.

Pulling Leaderboards from Redis

Now that the statistics are being updated regularly in Redis, modify the service from Project 3 to pull the data for the `wins` and `streaks` leaderboards from Redis rather than from the shareded relational database. Not only should the code be simpler, but you should find that while the data may be a bit stale, the performance has improved dramatically. This technique of pulling data from multiple sources in a single service is referred to as polyglot persistence.

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 the game state service.
3. The modified Python source code for the statistics service.
4. `Procfile` definitions for the services.
5. Your Python script for pulling statistics from shards and updating the Redis leaderboard.
6. The `crontab` file you used to schedule leaderboard updates.
7. Initialization and population scripts for Redis, if necessary.

   Note: since Redis is a schemaless database, you may not need this.
8. Any other necessary configuration files or scripts.

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 PDT 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
- Data model design
- View materialization and scheduling
- 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.)