CPSC 449 - Web Back-End Engineering

Project 1, Spring 2022 - due March 11

Last updated Thursday, February 17, 2:45 am PST

In this project you will analyze a problem domain, design and document a RESTful API for a web application in that domain, and consider how the application can be decomposed into microservices.

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. Creating detailed documentation for an API design.
4. Analyzing an application domain to determine bounded contexts.
5. Using bounded contexts to decompose a monolithic application into microservices.

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, February 28.

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?

Application Domain

By now you have probably heard of Wordle, 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. You may even be a regular player. If not, your first homework assignment is to play a game or two.

As it turns out, Wordle is entirely a client-side application. Both current and future answers are stored in the JavaScript code.

We would like to build a game similar to Wordle, but making appropriate use of server-side APIs. This would make it easier to implement a number of features, including:

- Making it harder to cheat
- Offering more than one game per day
- Offering different games to different users
- Allowing users to sync progress across devices

Designing an API

Given the application domain described above, identify the operations that a server-side version of Wordle would need to implement in order to provide the game functionality of the game described above (e.g, checking guesses, determining how close guesses are to the answer, tracking statistics, and sharing streaks). Pay particular attention to preventing users from cheating.

Read RESTful web API design from the Microsoft Azure Architecture Center, then design appropriate RESTful resources for each operation.

Documenting the API

Document the HTTP methods, URL paths, JSON data formats, and HTTP status codes in enough detail that a front-end developer could use your API to build the rest of the game. Include which resources are public and which resources and methods require authentication. You may wish to consult Tom Johnson's Documenting APIs: A guide for technical writers and engineers, particularly Section III: Documenting API endpoints.

For some examples of effective API documentation, check out the documentation for Stripe, Medium, and Mailchimp. You may wish to start with a template like Documenting your REST API and refer to An example of an API contract between the server and front-end devices.

Decomposing into microservices

Finally, it's time to start thinking about how your monolithic API could be decomposed into microservices. Begin by reading the Microsoft Learn module Decompose a monolithic application into a microservices architecture and the AWS Prescriptive Guidance on Decomposing monoliths into microservices.

Propose a decomposition of the API you designed into microservices. You may find that some contracts need to change. While there are hard-and-fast rules for system design, it shouldn't be difficult to identify at least five potential bounded contexts.

Document your proposed microservices. You may wish to include one or more diagrams showing how the services relate to each other.

Submission

Submit your design documentation as a `.pdf` file through Turnitin on Canvas before 9:45 pm PST on the due date. The first page of your document should identify the members of your team, the semester, and the project number. 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)

Designs are reasonable and clearly presented; explanatory text clearly and concisely explains design choices with appropriate context and analysis; organization makes it easy to review.

Basically Correct (4 points)

Designs are reasonable, but the presentation is not easy to follow, portions are not clear or lack context, or the work has minor mistakes.

Solid Start (3 points)

The approach is appropriate, but the work has mistakes in analysis, design, or presentation that undermine the usefulness of the design

Serious Issues (2 points)

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

Did Something (1 point)

The project 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 and service design
- Written description and documentation
- 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.)