<u>HACKATHON</u>

Jonathan Segura

Description

In my project, I designed and executed a series of rigorous and exhaustive stress tests to ensure the robustness and resilience of my system. I implemented three types of stress tests: **Smoke Test, Average Load Test, and Spike Test.**

The **Smoke Test** allowed me to verify the basic functionality and ensure that my system could handle the minimum expected load.

The **Average Load Test** was designed to simulate an average workload, providing me with valuable data on how my system behaves under normal conditions.

Finally, the **Spike Test** allowed me to assess my system's ability to handle sudden spikes in workload. This type of test was particularly useful for identifying potential points of failure and ensuring that my system could adequately scale to handle unexpected increases in demand.

These tests were carried out over a predefined period, with a workload that progressively increased as the test advanced. This approach allowed me to observe how my system behaves under a variety of conditions and workloads, which helped me optimize it and prepare for any eventuality.

Requirements

- **Visual Studio:** A Microsoft integrated development environment (IDE) that provides a suite of tools for software development, from code editing to debugging and version control.

- **Visual Studio Code:** A lightweight yet powerful code editor that can also function as an IDE. It offers a wide range of extensions to suit different development workflows.

- **Postgres**: An open-source relational database management system known for its scalability and compliance with SQL standards.

- **Rested:** A web service that allows testing and validating API endpoints, facilitating the debugging and development of web applications.

- **Gehtsoft:** A library that provides a series of useful functionalities for software development. (Please provide more details on how this library is used in your project).

- **GitLab:** A Git-based version control platform that also offers CI/CD functionalities, project management, issue tracking, and much more. Essential for teamwork and efficient source code management.

Installation Guide

PostgreSQL

PostgreSQL is essential for storing the data that my program will generate and read. To install PostgreSQL on Windows, I can navigate to the official website via this link: https://www.postgresql.org/download/. During the installation process, it prompts me to configure the password for the superuser account, which defaults to the username ‘postgres’.

Dotnet

To get started, I need to install the .NET SDK from the official link: https://dotnet.microsoft.com/download. After downloading the installer, I'll run it and follow the prompts provided by the installation wizard. To verify the installation was successful, I can open the command prompt and execute the command ‘dotnet --version’ to check the installed version.

Rested

The Rested extension, which is available for installation in the Chrome browser, has been instrumental in the development of this program for testing endpoints and executing HTTP requests. To install it, I simply visit the ‘chrome web store’, search for the extension by name ‘Rested’, or directly access it via this link. Once installed, I can access the Rested app from the browser's extensions menu to begin utilizing it. Inside, I input an address/endpoint and select the HTTP method I wish to perform, then click the ‘send’ button to receive the API response, which is subsequently displayed at the bottom.

Structure

The structure of my project follows the Model-View-Controller (MVC) pattern and consists of several classes that play specific roles in the program's implementation. Here's a more detailed description of each of the classes:

EndPoint:

I use this class to store the routes used to create HTTP requests to the API and access them easily.

TestType:

The TestType class helps me determine the type of test to be performed in each case. For instance, if it's a GET request type SmokeTest, it implies that the test will be executed to verify the connection and basic functionality of the GET request to the API.

RequestSender:

The RequestSender class is responsible for making the HTTP requests across the network. Its main function is to send requests to the endpoints defined in the EndPoint class and manage the responses from the API.

TestExecutor:

The TestExecutor class organizes and coordinates the execution of the tests. It manages which tests are to be performed, using the information provided by the EndPoint and TestType classes, and delegates the execution to the RequestSender class.

Metrics:

I use the Metrics class to store information collected during the execution of the tests. Specifically, it's used to save the data that will be displayed in the PDF generated by the program.

PDFMaker:

This class is responsible for generating and completing a PDF file with the data collected during the test execution. It uses the information stored in the Metrics class to fill the PDF with the results of the tests, creating a structured and easy-to-understand report.

Development

In the project, I adopted a collaborative approach using a version control system, which allowed me to work together on the same project without any code conflicts. This methodology helped me to efficiently divide and assign tasks, ensuring the cohesion and consistency of the code.

A crucial aspect of my strategy was the centralization of the endpoints in a single place. This decision provided accessibility and ease of modification, essential factors for the maintenance and future scalability of the project.

Initially, each type of test was conceptualized as an independent class. However, during development, I identified that, despite their differences, the tests shared a similar structure. This insight led me to unify and optimize my code by implementing a single class capable of determining and executing the appropriate test based on specific parameters.

To facilitate the identification of the type of test and the corresponding HTTP request, I used an enum class. This approach allowed me to maintain a coherent and flexible code, dynamically adapting to the needs of the project.

Furthermore, I implemented the “executeStressTest” method to simulate stress situations on the API. This method executes requests in parallel, replicating a scenario where the API is under the pressure of multiple simultaneous requests. This simulation was crucial for testing the robustness and responsiveness of the API under high demand conditions, thus ensuring its stability and reliability in a real production environment.

The integration of “executeStressTest” proved to be a vital component in my development process, providing me with a deep understanding of how our API would handle high-load situations and allowing me to make relevant adjustments to optimize its performance.

Challenges

During the development of the project, I faced two fundamental challenges.

Firstly, the code architecture turned out to be complex, which made it difficult to understand and modify efficiently. This led me to spend more time than anticipated on modifying the main structure of the code, which consisted of three classes, one for each type of test, to unify them into a single class to optimize and facilitate the understanding of the code. The process of integrating diverse functionalities and ensuring seamless interaction between them required a meticulous approach to refactoring and optimization, which was both time-consuming and challenging.

Secondly, the logic behind the creation and deletion operations lacked essential information, such as the unique identifier (ID), necessary for selecting and deleting specific users. This omission complicated the task of removing users created during the development process. Including the ID in these operations would have significantly simplified the process, enhancing efficiency and precision in data handling. The absence of a straightforward method to track and manage entities throughout their lifecycle necessitated a reevaluation of my approach to managing state and persistence within the application.

Furthermore, as I delved deeper into these issues, I realized that improving documentation and comments within the code could further alleviate the challenges associated with code complexity. By adopting a more descriptive approach to naming conventions and structuring my code, I aimed to make it more accessible to future revisions or other developers who might work on the project. This reflection led me to implement a more robust documentation strategy, focusing on clarity and maintainability.

In summary, the complexity of the code architecture and the lack of essential data were key challenges that impacted the development efficiency. Addressing these issues taught me valuable lessons in software architecture, data management, and the importance of clear documentation, which will undoubtedly influence my approach to future projects.