Author: William Wang
Student ID: 261052682

ECSE 429 Project Part C: Report

2024 December 3rd

Deliverables Summary

Unit Test Suite

The unit test suite is divided into two seperate folders:

- experiment
- unit_tests

The **experiment** folder contains a unit test script that tests the time taken by the Rest API to complete an add, update or delete request as the number of todos present in the list increases. The time was measured for the following number of todos present in the list:

- 1 todo
- 10 todos
- 100 todos
- 1000 todos

An attempt was also made to test the API with 10 000 todos but the application could not handle the requests and would throw an error.

The **unit_tests** folder contains the unit tests that were used during PartA of the project. However, each test was adapted to also measure the time taken to complete an add, update or delete request whenever it occured.

Static Analysis Methodology

The open-source static analysis tool Sonar Qube was chosen for the project. The static analysis involved analyzing the Rest-API for the following:

- Code complexity
- Code statement counts

- Technical risks
- Technical debt
- Code smells

In order to run Sonar Qube on the Thingifier, an additional file called sonar-project.properties was added to the folder containing the source code of the API. the new file contains information regarding Sonar's project key configuration. Then, once Sonar Qube is running and a token , the following command is executed in the main directory of the API's source code:

sonar-scanner.bat -D"sonar.projectKey=ECSE429" -D"sonar.sources=." -D"sonar.host.url=http://localhost:9000" -D"sonar.token=[generated_token_value]" -D"sonar.java.binaries=runTodoManagerRestAPI-1.5.5.jar"

where the "generated_token_value" is the token generated by Sonar Qube.

Performance Test Suite Video

A demo video showcasing the performance tests being run alongisde Perfmon can be found in the PartC/Video_demo folder. The video demonstrates the Rest API being started and shows its initial state. Then, the video shows the Perfmon configuration and the User defined Data Collector Set being started. Once Perfmon is running, the video shows the performance unit tests being run. The output of the tests is also shown, where it include useful and informative logs about each test and the time taken to complete a request to add, update or delete a todo instance in the list. The video then shows the todo list to demonstrate that once the unit tests are complete, the todo list is restored to its initial state. Finally, the video shows the user stopping Perfmon and the results obtained in the generated .csv file.

Written Report

This written report provides a description of all the deliverables for the project and describes the findings of unit test suite execution.

Source Code Repository

The source code repository for this project is uploaded as a GitHub repository, where the deliverables for this part of the project can be found in the **PartC** folder. The repository is made accessible with the following link:

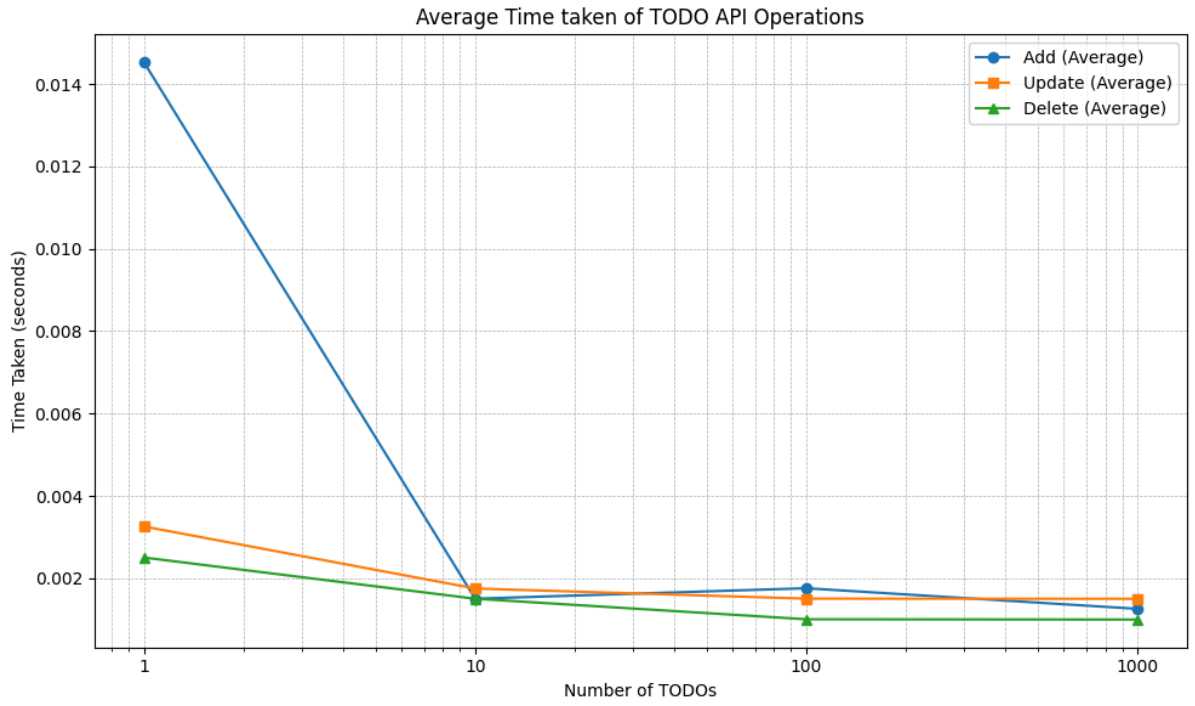
https://github.com/williamwang1382/ECSE429_Project/tree/main

The Source Code Repository contains not only the source code for all the tests cripts used during the performance testing, but it also contains the project report and all the relevant performance plots. As explained above, the video demo of a user running the tests can also be found in the repository.

Performance Testing Findings

Experiment Findings

From the experiment, the following results were obtained:



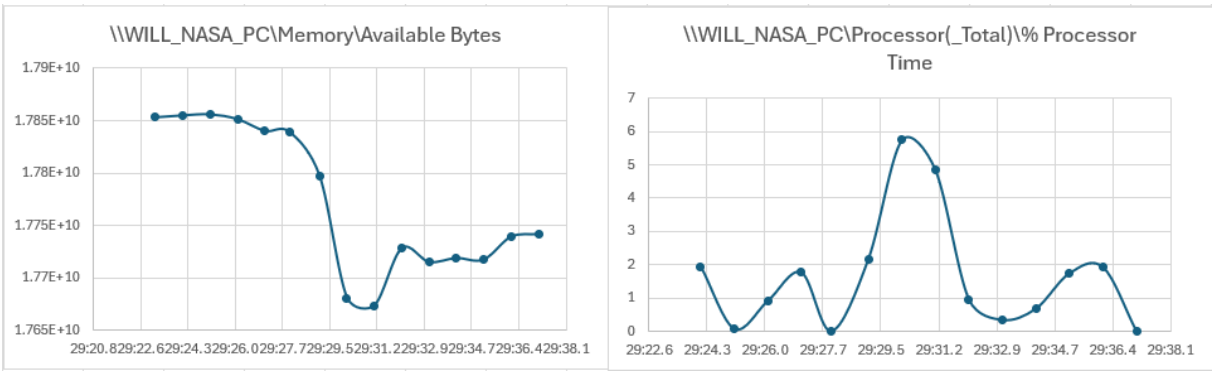
1. Time to add: as the number of todos increases, the time to add a new todo becomes faster past the first todo. The results showed that the time taken to complete an add takes longer when the todo is the first one to be added to the list. Any subsequent todo added seems to be added at a constant time, regardless of how many todos there already are in the list
2. Time to update: as the number of todos increase, the time taken to update a todo remains constant. While some level of variation was observed across the different number of nodes present in the list, through multiple runs, the time difference was never consistent, meaning that the difference is most likely noise. Thus, the results indicate that the time taken to update a todo instance is unaffected by the number of todos present in the list.

3. Time to delete: as the number of todos increases, the time to delete an existing todo becomes faster. Through multiple runs, a steady decrease in the time taken to delete a todo was observed as the number of todos in the list increased, where the time to delete a todo when there is only a singular new todo took the longest and the time to delete a todo after adding 1000 todos was always the fastest.

Based on the results obtained, one possibility that could explain the time decrease for the time to add/update as the number of todos increases is that the Rest API could have some caching mechanism. A Caching mechanism would also explain why only the first todo takes longer to perform an add or an update and afterwards the rest of the todos take constant time.

Unit Test Findings

After running the unit tests alongside Perfmon, the following results were obtained:

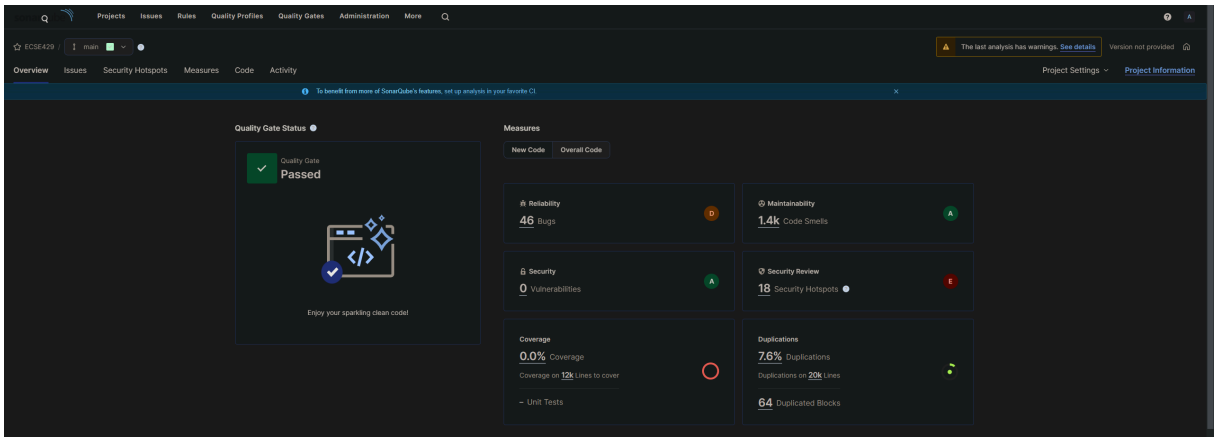


The results indicate that as the number of todos added to the list increases, the amount of memory and processing power required from the local machine increases. That is the case since in the Memory plot, the amount of available memory only stops decreasing once the tests are complete, indicating that with a larger amount of todos, the amount of memory available would keep decreasing. Similarly, the same trend can be observed in the processor's percentage usage, where the increase in processing power required only stops increasing once the test is finished.

The results outline outline a performance risk where with large number of todos, it is possible that the user's local machine can no longer handle the load caused by the request volume and would therefore potentially affect a user's experience while using the Rest API.

Static Analysis Findings

After running Sonar Qube on the Rest API's source code, the result yielded the following statistics:



In the 46 reliability bugs found, 7 bugs were found to be of critical severity, 32 were major and were minor. The most common bugs in the critical and major categories that were found were incorrect string comparison operations, incorrect HTML syntax and unstable code caused by jump statements in "finally" blocks. For incorrect string comparison operations, a straightforward suggestion would be to use the function .equals() instead of "!=". Another suggestion that can be made to resolve HTML syntax bugs would be to revise the HTML code and ensure that the tags follow correct syntax. Removing jump statements in "finally" blocks would also significantly improve code stability.

With Since 1354 code smells found, 1 code smell was found to have Blocker severity, 337 were critical, 454 were major, 261 were minor and 299 were of info severity. Sonar Qube found close to 1400 code smells, it would also be important for the developers to assess the gravity of the code smells found and ensure that the design choices behind the code smells do not lead to technical debt in the future. The priority would also be to primarily focus on the code smells that have high severity levels, as they can not only lead to technical debt, but they can also lead to bugs and worse performance.

Because Sonar Qube also found that 7.6% of the total number of lines of code were duplicated, reviewing and minimizing the amount of duplicated lines of code can improve the project's overall maintainability.

Finally, the implementation of a stronger cryptographic algorithm would be beneficial to the user's safety.