ECSE 429 Part C Report

Antony Zhao (260944810), Ziyue Wang (260951986), Peter Yu (261005015)

Introduction

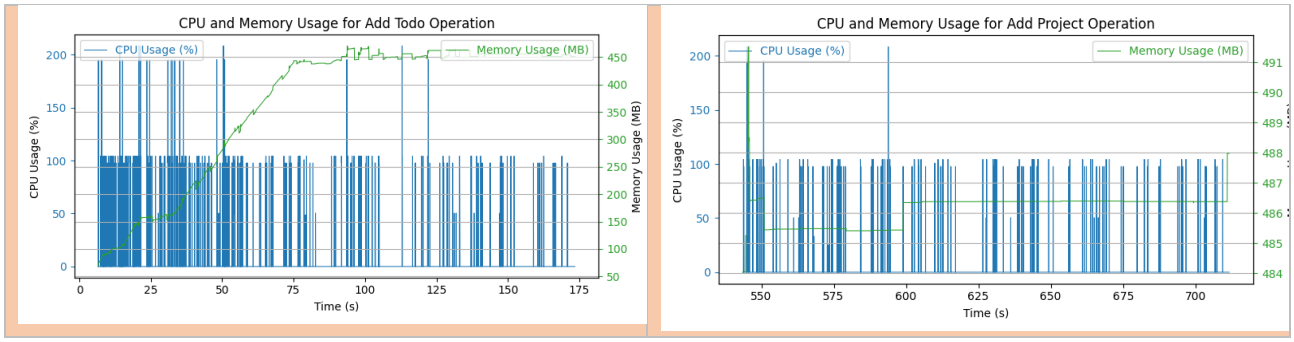
In Part C of our project, we present a comprehensive evaluation of the non-functional aspects of a REST API, focusing on Performance Testing and Static Analysis. The performance testing phase involves creating a program to dynamically interact with the API, using modified unit tests to measure operation times for getting, creating, updating, and deleting objects under varying loads, alongside monitoring resource usage using appropriate operating system tools. In the static analysis phase, the REST API's source code is scrutinized using the SonarQube Community Edition, aiming to identify and recommend improvements for issues related to code complexity, technical debt, and potential risks. This dual approach ensures a thorough understanding of the API's performance characteristics and code health, vital for its reliability and future scalability.

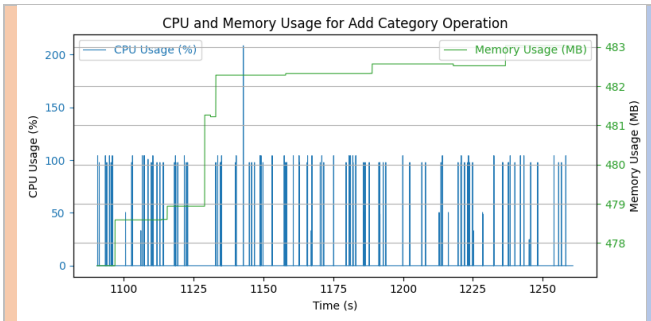
Deliverables

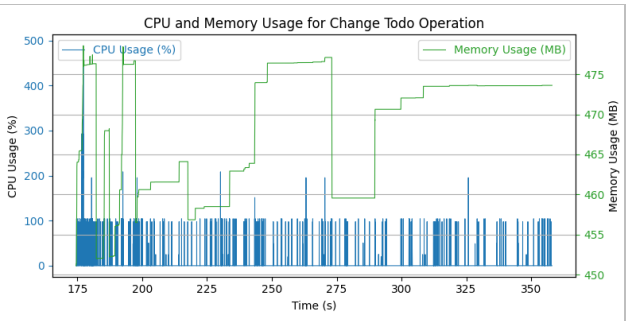
Performance: This section details the re-implementation of the fundamental operations (GET, POST, PUT, DELETE) across the three main sub-nodes (todos, projects, categories). For each sub-node, we have incorporated looping functions capable of executing these basic operations numerous times. This design facilitates the measurement of the relationship between the frequency of operation execution and the time required to complete each operation, enabling comprehensive analysis and reporting. To enhance the clarity and effectiveness of our results and visualizations, we have implemented a primary trigger to initiate all the loops simultaneously. Additionally, we have integrated functionalities for generating graphs and have established designated folders for systematically storing the outcomes in both .csv format and as images.

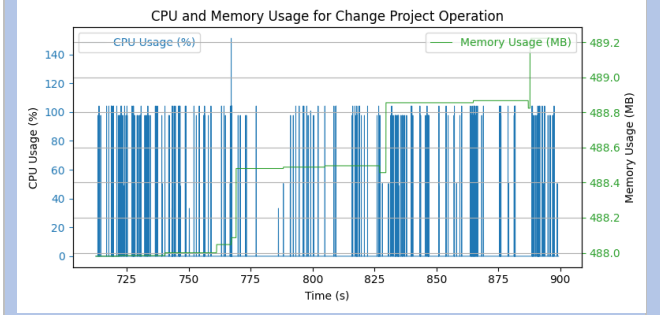
Videos: The two video each recorded the running of tests for performance (*dynamic performance run.mp4*) and the SonarQube analysis generated for source code (*sonarqube demo.mkv*).

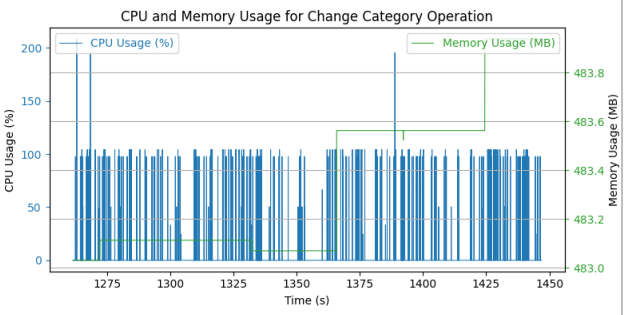
Graphs: Memory/CPU Usage vs. number of objects (again time):

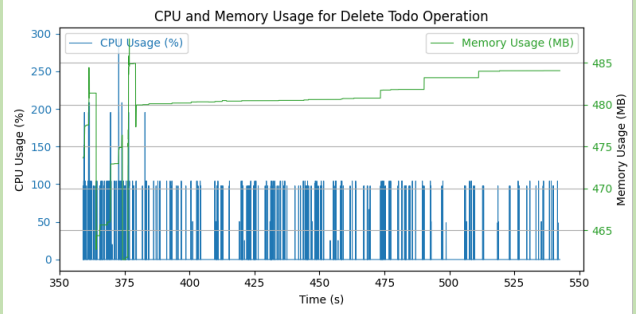


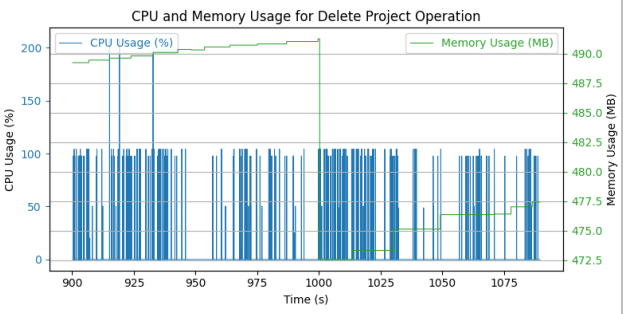


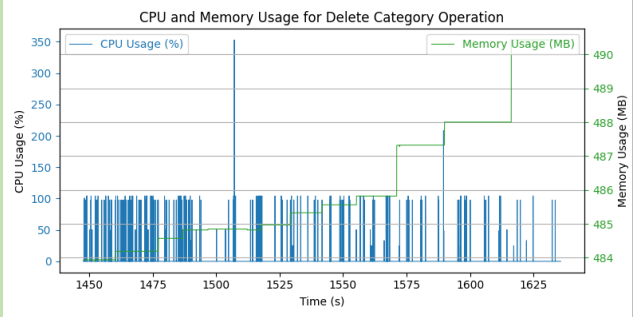












performance/output/csv/add_project_system_logs.csv

	Time	CPU Usage (%)	Memory Usage (MB)	Free Memory (MB)
1	505.8458281	14.4	21195714560	21524570112
2	505.8717291	6.2	21195644928	21524647936
3	505.8902655	0.0	21195452416	21524844544
4	505.9077492	0.0	21195440128	21524844544
5	505.9243267	12.5	21195440128	21524844544
6	505.9405982	0.0	21195440128	21524844544
7	505.9560892	12.5	21195440128	21524844544
8	505.9727193	6.2	21195440128	21524844544
9	505.9879107	100.0	21195440128	21524844544
10	506.0071034	6.2	21195440128	21524844544

performance/output/csv/delete_project_system_logs.csv

	Time	CPU Usage (%)	Memory Usage (MB)	Free Memory (MB)
1	839.6179314	11.3	21328809984	21391474688
2	839.6722955	70.8	21328416768	21391867904
3	839.6908545	12.5	21328769024	21391515648
4	839.706553	12.5	21328801792	21391482880
5	839.7220401	0.0	21328801792	21391482880
6	839.7409703	6.2	21328801792	21391482880
7	839.7632865	0.0	21328801792	21391482880
8	839.7807888	0.0	21326749696	21393534976
9	839.7970238	0.0	21326749696	21393534976
10	839.8132344	0.0	21326749696	21393534976

Recommendations for code enhancements:

Finding 1: In the context of 'change' operations within categories and projects, there was an observed increase in memory usage over time, which contradicts the expected behavior for such operations.

Finding 1: For “change” operation under category and project, the usage of memory increased with time went by, which is not logically make sense for a “change”. The usage of memory usage should fluctuate around 2Mb for standard 'change' processes. This anomaly may be attributed to the mechanism around. The reason is because of repetitions of updating (changing) where, ideally, old information should be deleted and replaced with new data. However, in this instance, such a replacement does not seem to occur, leading to a progressive increase in memory usage.

Finding 2: A similar pattern was observed in the 'delete' operations within the 'to-do' and 'category' sections. This pattern deviates from the conventional expectations associated with deletion processes, as it paradoxically results in an increase in memory usage rather than a decrease. This could be attributed to the unique mechanism of the Java Virtual Machine (JVM), where 'deletion' might entail marking information with a 'deleted' tag, rather than completely removing it from memory.
Finding 2: For “delete” operation under todo and category, the usage of memory increased with time went by, which is not logically make sense for a “change”. The reason is because with repetitions of updating (change), the old information should be deleted and replaced by newer information, which would not result the increasing usage of memory.

According to the previous findings and checking in source code (ex. Challengers.java, ChallengerApiRequestHook.java, ChallengeDefinitions.java), we have a few recommendations for code enhancements:

1. Refactoring for Clarity and Maintainability:
    - Break down large classes or methods into smaller, more manageable units.
    - Use meaningful names for classes, methods, and variables.
2. Error Handling and Logging:
    - Implement robust error handling for HTTP requests and responses.
    - Enhance logging for better insight into application behavior.
3. Performance Optimization:
    - Optimize data processing in frequently called methods.
    - Use efficient data structures and algorithms.
4. Consistency and Code Standards:
    - Ensure consistency in coding style and adhere to Java best practices.
    - Maintain uniform codebase with standard conventions.
5. Concurrency and Thread Safety:
    - Ensure proper handling of concurrent operations and shared resources.
    - Use thread-safe patterns and data structures.
6. Design Patterns and Best Practices:
    - Apply appropriate design patterns for common problems.
    - Follow best practices in object-oriented design.

Code Repository

The following is the folder structure of our repository:

```
ECSE 429 Project Code Repo
├── README.md
├── Todo-Manager-swagger.json
├── features (PART B Code)
├── requirements.txt
├── runTodoManagerRestAPI-1.5.5.jar
├── starter.bat
├── test.bat
├── test_random.bat
├── test_server_down.bat
├── tests (PART A Code)
├── dynamic_performance_run.py
└── performance
    ├── __init__.py
    ├── output
    │   ├── csv
    │   │   ├── add_category_system_logs.csv
    │   │   ├── add_project_system_logs.csv
    │   │   ├── add_todo_system_logs.csv
    │   │   ├── change_category_system_logs.csv
    │   │   ├── change_project_system_logs.csv
    │   │   ├── change_todo_system_logs.csv
    │   │   ├── delete_category_system_logs.csv
    │   │   ├── delete_project_system_logs.csv
    │   │   └── delete_todo_system_logs.csv
    │   └── graph
    │       ├── add_category.png
    │       ├── add_category_smoothed.png
    │       ├── add_project.png
    │       ├── add_project_smoothed.png
    │       ├── add_todo.png
    │       ├── add_todo_smoothed.png
    │       ├── change_category.png
    │       ├── change_category_smoothed.png
    │       ├── change_project.png
    │       ├── change_project_smoothed.png
    │       ├── change_todo.png
    │       ├── change_todo_smoothed.png
    │       ├── delete_category.png
    │       ├── delete_category_smoothed.png
    │       ├── delete_project.png
    │       ├── delete_project_smoothed.png
    │       ├── delete_todo.png
    │       └── delete_todo_smoothed.png
    ├── test_categories.py
    ├── test_projects.py
    └── test_todos.py
```

- ***Todo-Manager-swagger.json***: This file is a JSON file that contains the Swagger documentation for the Todo Manager REST API downloaded from http://localhost:4567/docs/swagger. This file describes the endpoints, request/response structures, and other important information about the API. Postman can import this file for the manual testing session.

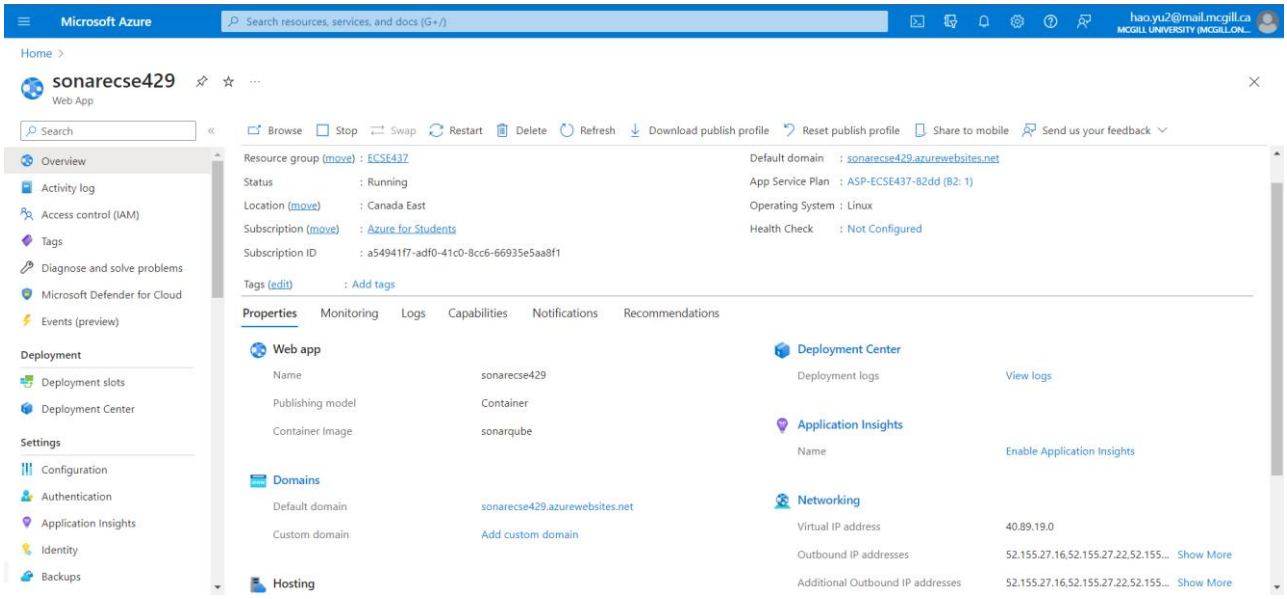
- ***requirements.txt***: This text file lists all the Python dependencies the Todo Manager Rest API requires. Package managers (pip) use it to install all the libraries and versions required to run the API. (pip install -r requirements.py)
- ***features*** and ***tests***: These two folders contain the python scripts for Part A and Part B.
- ***runTodoManagerRestAPI-1.5.5.jar***: This file is a Java Archive (JAR) file that contains the executable code for running the Todo Manager Rest API. It is a packaged file with all the dependencies and libraries required to run the API. The version number (1.5.5) indicates the specific version of the API.
- ***starter.bat***: This batch script file can be executed on a Windows machine to start running the Todo Manager Rest API.
- ***test.bat, test_random.bat*** **and** ***test_server_down.bat:*** These three scripts run tests with behave library for cucumber test cases. The *test.bat* is a normal test script. The *test_server_down.bat* simulates the server is down and then executes the tests which the expected failed all. And the *test_random.bat* runs all tests scenarios in random order.

- ***dynamic_performance_run.py:*** This script is used to run performance tests on a task management system. It includes functions to test various operations such as adding, updating, deleting operations in three different endpoints, Todos, Categories and Projects. The main function starts by creating a directory for storing the output files. It then starts the server for the task management system and waits for the server to be ready. Once the server is ready, it runs the performance tests for each operation, using a separate thread for logging system metrics. After each test is completed, it stops the logging thread and generates graphs for CPU and memory usage during the test.
- ***performance/test_*.py:*** These files contain three test cases for the different endpoints of todos/projects/categories with executing 10,000 times.
- ***performance/output:*** This folder are the place to store the generated logger file for CPU usage and memory usage and visualized (smoothed) graphs.

SonarQube Static Analysis

SonarQube Setup:

We deploy the official docker image of SonarQube, “sonarqube”, in Azure Cloud for collaboration between our group members. The URL is: https://sonarecse429.azurewebsites.net. After initialization of default password and creation the “main” branch, then run Maven command: *./sonar.bat* Check Sonar Report on the URL via User: admin and Password: admin123. More information can be found in the URL and video.



Bugs:

1. challenger/.../challenge/challengesrouting/ChallengerTrackingRoutes.java

```
L31 if(xChallengerGuid != null && xChallengerGuid.trim()!=""){
L63 if(xChallengerGuid == null || xChallengerGuid.trim()==""){
```

Issue: Reliability

Priority: Medium

Technival debt: 1h

Suggestion: Strings and Boxed types should be compared using "equals()".

2. ercoremodel/.../definitions/field/definition/Field.java

```
L376 return examples.get(new Random().nextInt(examples.size()));
```

Issue: Reliability

Priority: High

Technival debt: 49min

Suggestion: Save and re-use Random

3. ercoremodel/.../core/domain/instances/InstanceFields.java

```
L189 final InstanceFields fieldInstance = objectValue.asObject();
```

Issue: Reliability

Priority: Medium

Technival debt: 1h7min

Suggestion: Check for null and throw exception

4. thingifier/.../thingifier/api/restapihandlers/RelationshipCreation.java

```
L117 connectThis.getRelationships().connect(relationshipToUse.getName(),
relatedItem);
L131 relationshipToUse.getName()));
```

Issue: Reliability

Priority: Medium
Technival debt: 14min
Suggestion: Check for null and throw exception

5. thingifier/.../thingifier/application/MainImplementation.java

```
L362 String.format("Will configure app as default profile");
```

Issue: Reliability
Priority: Medium
Technival debt: 5h34min
Suggestion: Use the return value of String.format

6. thingifier/.../sparkhttpmessageHooks/ClearDataPreSparkRequestHook.java

```
L15 this.maxgap = minutes*60*1000;
```

Issue: Reliability
Priority: Medium
Technival debt: 50min
Suggestion: Cast to long

7. thingifier/.../thingifier/htmlgui/RestApiDocumentationGenerator.java

```
L341 Random random = new Random();
```

Issue: Reliability
Priority: High
Technival debt: 2h48min
Suggestion: Save and re-use Random

Clean Code Attribute

Consistency: 21, Intentionality: 808, Adaptability: 137

- Lack of use of collection method isEmpty() when determing if a collection is empty.
- Should replace System.out with a logger
- Blocks of commented out code remains
- Unused importation should be removed
- Duplicating literal should be replaced with defined constant variable
- Inconsistency exists for expression matching
- Cognitive comlexity can be reduced for certain methods