CSC301 A1

Abdelrahman Alhendi

January 30th, 2024

1 Introduction

This document will be the writeup.pdf portion of my assignment. I will explain my classes and functions here, along with what sources I used to assist me with the code. I will explain which parts can easily scale up to A2 and which will need much more work.

Firstly, I understand that in the instructions we were told to add the documentation of our methods in the actual code and use a specific command to turn it into a certain file type. Unfortunately, I did not have enough time to research this and learn how to do it, so I am placing the documentation directly in my code and in this written document. My apologies. I hope to fix this problem for A2 and do what was properly instructed.

2 WorkloadParser

The WorkloadParser was relatively simple to make. It simply opens up the file named after the argument for runme.sh -w. The WorkloadParser will open this file and read each line, parsing the lines and finding out what commands need to be done. It will place it in a dictionary and call the make_post_request method or the make_get_request method. Each of these methods was taken from the code posted on the Quercus by Professor Andi. I had to change the code slightly to account for the use of JSONs and not dictionaries. I used CHATGPT to help me research how to turn the dictionary into a JSON and transfer it over. I also used CHATGPT to figure out how to take in the argument which will be the .txt file with the commands in the runme.sh file. All of the rest of the WorkloadParser was done by me without any research. I believe that nothing in the WorkloadParser needs to be scaled up for A2 except for the config.json parsing done near the beginning. Since in A2, the config.json we will get will be significantly larger, thus the current code will not be able to properly handle it.

3 OrderService

The OrderService is the main portion of the API since unfortunately due to time constraints I was unable to implement the ISCS. Again, my apologies. The server setup in the main was gotten from the code given on Quercus by Professor Andi. I then learned how to parse the given file from args to a JSON using CHATGPT. The main uses this JSON to get the port number from config.json, and passes the JSON to every handler in OrderService so they can individually get the necessary IP and Port numbers. The handling of POST and GET requests was learned from the given code on Quercus by Professor Andi. Here I turn the given data into a JSON. I learned how to do this using CHATGPT but I found this specific JSON library, org.json, myself on github. https://github.com/stleary/JSON-java/blob/master/README.me. This is the link to the library. Then I have an abundance of if statements and try-catches to protect all my handlers from garbage data. I then use the passed-in JSON from the main to find the needed IP and Port numbers and use these to make the needed URI for POST or GET requests. I learned how to make these POST and GET requests using the following website. https://www.twilio.com/blog/5-ways-to-make-http-requests-in-java. I then got help from Piazza to figure out a small error with this code, specifically how to put the data into the request. I then use try-catches and if statements to properly handle all errors and status code returns. The GET requests are slightly different. Instead of inputting the necessary data into the request, you must simply add the necessary id of what is needed to be found into the URI directly. From here, I extract it in the proper GET handlers. I learned how to extract this URI thanks to the code posted on Quercus by Professor Andi, where he showed off how to get the URI used for the request.

All of the handlers in the OrderService follow this same general order. The only slight exception is the OrderHandler which makes two GET requests, then checks the quantity to ensure it is valid, and then makes a POST update request to lower the quantity of the Product.

The Shutdown command makes two POST requests to the UserService and the ProductService, forcing them to shut down and then shutting down the OrderService itself afterward. The command prompts opened up won't directly close but the servers will and no possible commands can be made after the shutdown is called.

Unfortunately, I did not have the time to implement the restart function. Again, my apologies. I hope to implement this fully in A2.

In terms of scalability, I believe that all of this code should be able to scale up to A2 relatively simply. The only concern I have is for the sheer amount of commands being made constantly. I am sure some form of multi-threading is needed to ensure it all works appropriately. I believe that the current Order-

Service would not be able to aptly handle such a large amount of commands. Similar to the WorkloadParser, the config.json parsing done throughout the OrderService must also be scaled up for the same reasons.

4 UserService and ProductService

The UserService and the ProductService work nearly identically, with slight differences everywhere due to them both taking in data that holds different attributes. Similarly, to the OrderService, they create their ports in main using the config.json. They both have a GetHandler and a PostHandler.

The GetHandler parses the URI from the request to find the needed id and uses that to find the appropriate User or Product from their respective databases. From here, it returns the gotten User or Product through the sendResponse method by making it the response.

The PostHandler turns the data into a JSON, has many checks for garbage data, and then uses the JSON to find what command must be executed. The "create" command first makes an instance of a UserService or ProductService with the data from the JSON, and then uses the insert(UserService) or insert(ProductService) method to insert the user or product into their respective databases. If the user or product already exists in the database, it will instead send back the appropriate response. The "update" command first uses the getUser or getProduct method to find the appropriate user or product using the given id. From there, it checks what fields need to be updated and then uses the updateUser or updateProduct methods to update them in their respective database. Finally, the "delete" command gets the appropriate user or product from the database, again using the getUser or getProduct methods, checks if the fields correspond so that it is a valid "delete" call, and if it is, uses the deleteUser or deleteProduct methods to delete the user or product from their respective databases. All of these methods and if statements check for any errors and send back the appropriate status codes and responses.

All of the methods used for the databases were made thanks to the research on a selection of websites. The following are the links for these websites.
https://www.sqlitetutorial.net/sqlite-java/create-database/
https://www.sqlitetutorial.net/sqlite-java/create-table/
https://www.sqlitetutorial.net/sqlite-java/select/
https://www.sqlitetutorial.net/sqlite-java/insert/
https://www.sqlitetutorial.net/sqlite-java/update/
https://www.sqlitetutorial.net/sqlite-java/delete/
https://www.sqlitetutorial.net/download-install-sqlite/
The last link is instructions on how to install SQLite. I then downloaded SQLite from the following page:
https://www.sqlite.org/download.html

I kept getting an error when trying to connect to the database. I was eventually able to learn this line Class.forName(”org.sqlite.JDBC”); from the following website that solved the problem: https://stackoverflow.com/questions/16725377/unable-to-connect-to-database-no-suitable-driver-found.

I also had to download the SQLite driver from the following GitHub page: https://github.com/xerial/sqlite-jdbc#download

I also kept getting a ”logging error”, after consulting CHATGPT and reading the website of the error, I learned that I need another .jar file to fix the problem: https://www.slf4j.org/codes.html#StaticLoggerBinder https://repo1.maven.org/maven2/org/slf4j/slf4j-simple/1.7.36/

5 runme.sh

The runme.sh took an extensive amount of research to figure out. To learn how to make the proper if statements in bash, I consulted my old assignments from CSC209 and CHATGPT. I then used the following website to learn how to compile and run code: https://stackoverflow.com/questions/8949413/how-to-run-java-program-in-terminal-with-external-library-jar
This website, along with CHATGPT and the –help information for the java/java commands, taught me what compiled code is, how to compile it, where I should put the compiled code, and how to properly compile my .jar files. I decided to place these .jar files in their own folder, lib, in the root of the CSC301A1 folder. My apologies since the given instructions gave us a specific folder structure, and this lib folder was not part of it. I will further consult the Professor to know if I should change this for A2. After much trial and error, and thanks to all of the resources online that I listed, I was able to finally figure out the runme.sh, and how to properly use it.

The runme.sh has flags to compile all of the code, and start all individual services. Call runme.sh -c to first compile all of the code. Then call runme.sh -u, runme.sh -o, and runme.sh -p in any order to start up all the services. Finally, start the WorkloadParser with runme.sh -w. Ensure the config.json file in the root has the necessary information on Ports and IPs in JSON format. Again, due to the reason stated earlier, the runme -i does not run the ISCS since it is not implemented for this assignment. Again, my apologies.

I additionally added a runme.sh -j which uses wget to download the needed SQL driver. I learned about wget thanks to a Piazza post, and then I inquired chatgpt about it. I also used the manual page of wget to further learn about the flags I would need.

Scaling up the runme.sh should not take long since the same services will be present for A2. The only thing that may need some work is the runme.sh -i once the ISCS is implemented for A2. But it should be very similar to the other if statements regardless.

The major shortcuts made were in the lack of a restart command, the ISCS, and some differences as to what was told in the instructions, such as the project structure. The project structure was done in this specific way, with a lib folder containing the .jar files, simply because it seemed to be more convenient and organized. The lack of certain features was due to me running out of time. I started this project immediately in the first week, but inexperience in research led to lackluster research that resulted in code that had to be redone when Professor Andi released information on how to properly make API requests and handles. Over this project, I have gained many skills, especially in research, but in my development of these skills, I lost a lot of time. I chose to ignore the restart and ISCS implementations since they seemed to have the least amount of impact on the test cases. I chose to prioritize the bulk of the function of the different commands, along with the many edge case situations, and proper runme.sh usage. I hope to implement all of these missing features in A2, where I will hopefully have the skill to properly do the necessary research without losing much time. I also consulted CHATGPT a lot in my research. This was something that I was not proud of since you cannot directly cite the sources CHATGPT learned from. I hope to use CHATGPT significantly less in A2. This, I hope, is realistic considering how much I learned on how to research.