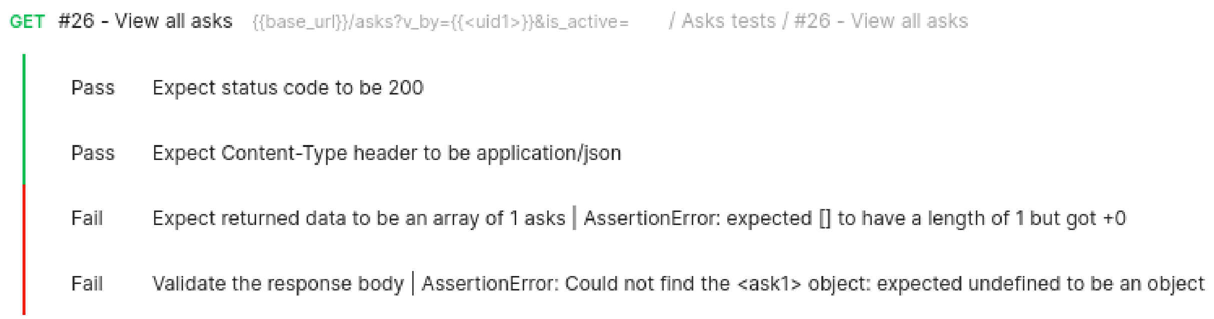
Memo

To: Professor Virgil Bistriceanu
From: Jacklyn McAninch
Date: Sep 16, 2022
Subject: CS 445 Final Project Review

I completed this project to the best of my ability, and with the highest quality of work I could produce. Unfortunately, it is still not perfect, but it contains nearly all of the desired functionality. Upon my final submission, the generated .war file passed 223 of the Postman tests, and failed 19. However, 4 of these seem to be due to incorrect test cases, as shown in this example:



GET #26 - View all asks {{base_url}}/asks?v_by={{<uid1>}}&is_active= / Asks tests / #26 - View all asks

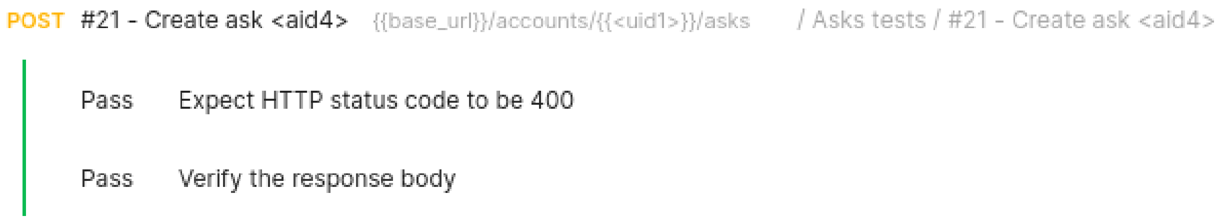
Pass Expect status code to be 200

Pass Expect Content-Type header to be application/json

Fail Expect returned data to be an array of 1 asks | AssertionError: expected [] to have a length of 1 but got +0

Fail Validate the response body | AssertionError: Could not find the <ask1> object: expected undefined to be an object

I believe this to be incorrect as it is queried by <uid1>, who is not a CSR actor and also has not created any valid asks. This particular account has only made one query previously in the Postman tests, which is shown here:



POST #21 - Create ask <aid4> {{base_url}}/accounts/{{<uid1>}}/asks / Asks tests / #21 - Create ask <aid4>

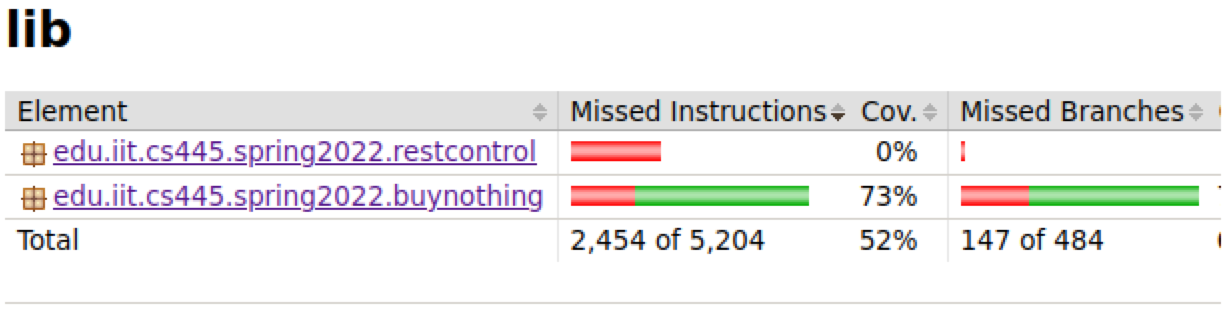
Pass Expect HTTP status code to be 400

Pass Verify the response body

Since this request should not alter the current data, the GET request from #26 should return an empty list, because no valid asks were created under this account with <uid1>. A similar situation occurs for the GET request on test #42 for gives.

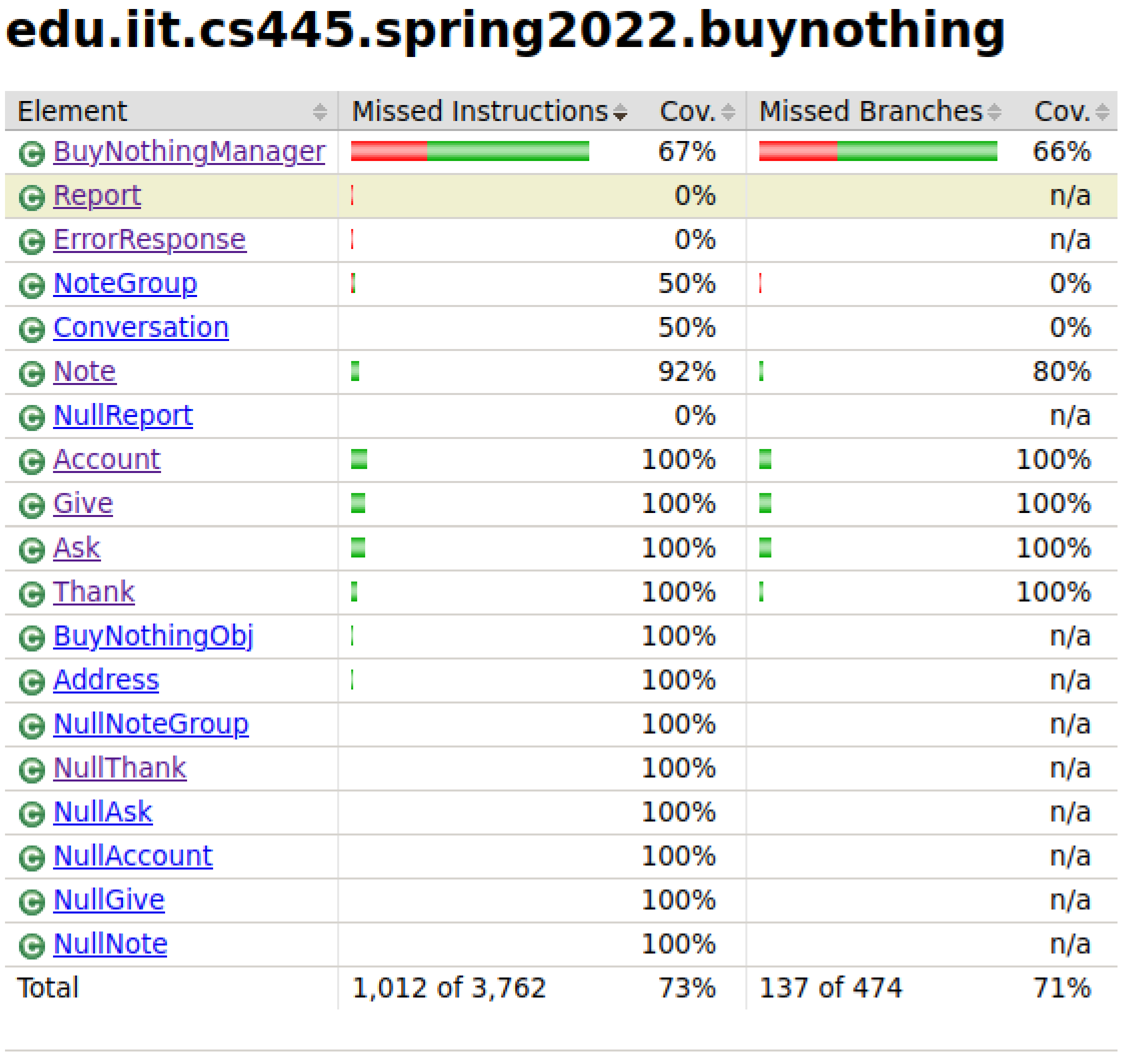
Regarding the rest of the failures, this was due to my inability to structure the manager logic to efficiently organize notes and reports in the desired format. The hardest to implement was the filtering of data according to the query parameters. In the future, I think this could be solved by adding a mapping structure to associate accounts with their respective conversations, rather than having all the data being stored in deeply nested classes.

JUnit testing shows a similar trend. Tests for accounts, asks, gives, and thanks are all decently extensive, with only a few edge cases being skipped because they were tested in Postman. As per the Q&A section for this project, I kept my REST_controller file very light, limiting its functionality to solely handling URL parameters and translating returned objects into JSON objects for the response. Therefore, there are no JUnit tests for it, but as seen with the Postman tests, it performs the correct behavior. Ignoring this section on the JacocoTestReport, I received 73% unit test coverage, as shown below. The most problematic area was the BuyNothingManager, of which the top half of all function are also displayed below:



lib

Element	Missed Instructions	Cov.	Missed Branches
edu.iit.cs445.spring2022.restcontrol	🟥	0%	🟥
edu.iit.cs445.spring2022.buynothing	🟥🟩	73%	🟥🟩
Total	2,454 of 5,204	52%	147 of 484



edu.iit.cs445.spring2022.buynothing

Element	Missed Instructions	Cov.	Missed Branches	Cov.
BuyNothingManager	🟥🟩	67%	🟥🟩	66%
Report	🟥	0%		n/a
ErrorResponse	🟥	0%		n/a
NoteGroup	🟥	50%	🟥	0%
Conversation		50%		0%
Note	🟩	92%	🟩	80%
NullReport		0%		n/a
Account	🟩	100%	🟩	100%
Give	🟩	100%	🟩	100%
Ask	🟩	100%	🟩	100%
Thank	🟩	100%	🟩	100%
BuyNothingObj	🟩	100%		n/a
Address	🟩	100%		n/a
NullNoteGroup		100%		n/a
NullThank		100%		n/a
NullAsk		100%		n/a
NullAccount		100%		n/a
NullGive		100%		n/a
NullNote		100%		n/a
Total	1,012 of 3,762	73%	137 of 474	71%

BuyNothingManager

Element	Missed Instructions	Cov.	Missed Branches	Cov.
generateReport(String, String, String, String, String)	🟥	0%	🟥	0%
searchNotes(String, String, String)	🟥	0%	🟥	0%
createNote(Note)	🟥🟩	57%	🟥🟩	33%
deleteNote(String, String)	🟥	0%	🟥	0%
createAsk(String, Ask)	🟥🟩	51%	🟥🟩	50%
createGive(String, Give)	🟥🟩	51%	🟥🟩	50%
createThank(String, Thank)	🟥🟩	50%	🟥🟩	50%
deleteNoteByToID(String)	🟥🟩	38%	🟥🟩	37%
deactivateAsk(String, String)	🟥🟩	44%	🟥🟩	50%
deactivateGive(String, String)	🟥🟩	44%	🟥🟩	50%
viewMyThanks(String, String)	🟥🟩	57%	🟥🟩	42%
createDefaultReports()	🟥	0%		n/a
findReportByID(String)	🟥	0%	🟥	0%
deleteNoteByAccountID(String)	🟥🟩	41%	🟥🟩	25%
checkMissingNoteInfo(Note)	🟥🟩	39%	🟥🟩	50%
deleteAskByAccountID(String)	🟥🟩	36%	🟥🟩	33%
deleteGiveByAccountID(String)	🟥🟩	36%	🟥🟩	33%
viewNote(String)	🟥	0%	🟥	0%
createAccount(Account)	🟥🟩	55%	🟥🟩	50%
updateAccount(String, Account)	🟥🟩	74%	🟥🟩	75%
findOriginalSourceUID(Note)	🟥	0%	🟥	0%
createReport(Report)	🟥	0%	🟥	0%
updateNote(String, Note)	🟥🟩	75%	🟥🟩	50%
viewNotes(String, String, String, String)	🟥	0%	🟥	0%
searchAsks(String, String, String)	🟥🟩	89%	🟥🟩	81%
searchAccounts(String, String, String)	🟥🟩	92%	🟥🟩	87%
viewAsks(String, String)	🟥🟩	91%	🟥🟩	92%
viewGives(String, String)	🟥🟩	91%	🟥🟩	92%
viewAllReports()	🟥	0%	🟥	0%
updateAsk(String, Ask)	🟥🟩	91%	🟥🟩	66%
deleteGive(String, String)	🟥🟩	86%	🟥🟩	75%
deleteAsk(String, String)	🟥🟩	85%	🟥🟩	75%
checkMissingGiveInfo(Give)	🟥🟩	84%	🟥🟩	87%
checkNoteToType(Note)	🟥🟩	82%	🟥🟩	75%
checkMissingThankInfo(Thank)	🟥🟩	80%	🟥🟩	83%
searchGives(String, String, String)	🟥🟩	94%	🟥🟩	83%
searchThanks(String, String, String)	🟥🟩	94%	🟥🟩	75%
viewMyAsks(String, String)	🟥🟩	93%	🟥🟩	92%
viewMyGives(String, String)	🟥🟩	93%	🟥🟩	92%
findAskByID(String)	🟥🟩	85%	🟥🟩	83%
findGiveByID(String)	🟥🟩	85%	🟥🟩	83%
findThankByID(String)	🟥🟩	85%	🟥🟩	83%
findNoteByID(String)	🟥🟩	85%	🟥🟩	83%

Once again, due to lack of functionality for notes and reports, I was limited in which tests I could perform successfully, which accounts for the majority of missing coverage. The rest are fairly trivial missed branches which were covered by Postman tests.

See the charts below for the total lines of code in source (left) and complexity (right) according to Jacoco test report, ignoring comments, constructors, and whitespace:

edu.iit.cs445.spring2022.buynothing:

Account.java	53	29
Address.java	11	6
Ask.java	47	29
BuyNothingManager.java	590	268
BuyNothingObj.java	13	6
Conversation.java	10	6
ErrorResponse.java	12	6
Give.java	47	29
Note.java	38	22
NoteGroup.java	19	10
NullAccount.java	2	2
NullAsk.java	2	2
NullGive.java	2	2
NullNote.java	2	2
NullNoteGroup.java	2	2
NullReport.java	2	2
NullThank.java	2	2
Report.java	13	7
Thank.java	28	17

edu.iit.cs445.spring2022.restcontrol

BuyNothingApp.java	1	1
REST_controller.java	217	41

edu.iit.cs445.spring2022.test:

BuyNothingTest.java	668

I personally faced many challenges on this project, as it was the first time I had worked with vanilla Java for deploying a web server, nor had I ever worked with HTTP requests. Conceptually these were very simple, but I struggled with nuances like getting the correct path set up, basically fully blind. I ended up solving it by working directly from the Rest Lamp example project and making many small changes to finally get the final result. Similarly, as I was new to Postman, I didn't immediately notice small errors in my environment that had previously inhibited me from running the tests at all for a long time. Once I got comfortable with all of these new platforms, my biggest struggle was trying to debug my code, based solely upon the response JSONs and comparing their results to what the provided tests expected. For example, it took me a while to figure out my classes needed to match the naming conventions in the expected result exactly; i.e. making sure both my version and expected version both used underscores rather than camel case.

The REST concepts themselves were fairly manageable, I think they provided the project with some structure, without which the goal would have been entirely impossible. It also allowed the code to follow a readable structure and I think the final product is quite organized and understandable. At the end of the day, I spent countless hours on this project, namely just debugging these small nuances trying to pass all Postman tests. I think a rough estimate of the total time I spent on it would be about 100 hours, and I think I could have gotten it fully functional had I spent another 10.

All in all, though I was very frustrated with this project, especially for how long it took me, I did learn a lot. I have found many places to investigate on Postman which can help me debug web servers in the future. I employed REST principles and became familiar with the ins and outs of the Jakarta web server package. I definitely wrote the largest program I have ever attempted, and solidified my Java and OOP skills as a result. Though my test report may look to be a bit lacking, I still consider this project to be a success, regardless of the grade I receive. Thank you for allowing me the extra time to work on it and get it to the state it is in today.