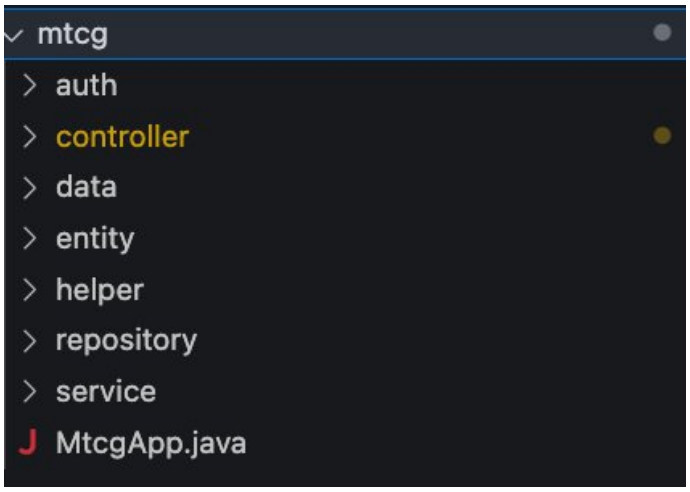
Protocol MCTG

Adrian Sanchez Chiner

- **Design:**

It was the first time for me making a new project with the object oriented design pattern that java enforces and I have noticed how as the project progressed it became increasingly difficult to maintain an overview over my files.

I ended up having such an structure in my folder MCTG.



The separation from entity, controller, service and repository helped me a lot to maintain a structure consistent but I struggled when I had to implement new ideas like the Helper Class or the Auth Class. When I look at it with perspective I would much rather unify both or make Auth a part of Helper... I ended up not doing it because of the single responsability pattern I tried to implement and if I am being honest with myself Helper ended up being the go-to Class for functions that were needed in more than one please but I am not quite happy with the naming scheme, because as the project grows the complexity of the class would clearly increase. I take this as a lesson that with a bit of planing from my part about what I will need in the future I could have make a better pattern here.

Apart from that... Unit test are so important.. Something that ended up needing reword several times during the project was the request mapper function. As new call request with different bodies were implement that instead of having a simple JSON had a simple string or an array of Strings enforce adaption in this method. The problem is that it is a rather complex method that in case of breaking can destroy the whole project, so simple unit test for the current cases implemented would have helped a lot in the future to now that nothing was broken adding a new case. So I think another lesson here is that test-driven development can really in the end help save time in the future, because as time goes by it becomes increasingly difficult to understand past code that we ourselves have written.

Another big part of how I have implemented my project, even though it is not part of the java code is in my opinion my start.sh script. With this file I have preconfigured the whole stack used for the project. This scripts takes care of backup the database and reimporting from the file on start, it starts the postgres container, compiles the project using pure java and maven (not dependent of a IDE) and recompiles when one of the files in the project is altered. This configuration allowed me to use my IDE of choice (Visual Studio Code) and helped with working from different laptops because of the seamless export and preparation of the files.

Another very useful tool was the test.sh script that I altered from moodle. The original version focused on testing the routes thoroughly but as in my project buying a new packages is buying a random one from the available ones it incurred in incompatible set ups for configuring the deck or making a battle. After some adaptations in the script that included the automatic drop of the tables at the beginning it became my most useful tool for testing apart from unit test itself.

- **Unique feature:**

As unique feature from my project I have incline myself for doing a history route. In this history route a user can see what happened in the game, such as buys from him or other users of new packages, trades between users and also battles from other users. This is all public information to help the users understand the game and in my opinion it only makes sense to make it publicly available! In the future it would be needed to create custom call for filtering the information, because this history table is expected to grow a lot in the future and would probably be best to implement a custom query for the first 15 elements that incrementally delivers more as the user wants to see more. That would be a great way to handle this in case of wanting to go in production.

- **Unit Test Design:**

As I was explaining before, the idea of testing that mapper would have been perfect at the beginning so when I needed to choose what components of my project were worthy of unit testing I inclined for my Helper class.. As I said, as the project grows also did that class and as there were a lot of code being used in different parts of the project I decided that a bug there would create the most problems to fix. That is the reason behind expanding more the uni testing in that class rather than in more of the services.

- **Time spent:**

I have the incredible amount of 79 commits in my repository... I guess time flies when we are having fun! I have used a simple system to know approximately how much time I have spend in the project. Whenever I made new commit if I worked more than half an hour on it I added it in my excel file. From the 79 commits 32 of them were small things that required less than this half an hour or were directly small fixes. From the other 47 most of them were 1 hour time span, because I usually commit whenever I arrive to a save point, in this case 29 were around 1 hour. And the rest were mixed between 2 to 3 hours of pure work. All together makes 82 as can be seen in the small excel extract down here.

	Commits	Stunden	
	79		
15 minutes *	32	8	h
1 hour *	29	29	h
2.5 hours *	18	45	h
Total	79	82	

- **Link to git:**

Link to Github from MCTG from Adrian.