**M226a Miniproject: TBZ – Ebanking**

thingy

The planing process

The programming process

The testing & conclusion

The bug fixing process

The diagrams

The planing:

The idea of coding an e-banking application was the first thing that came to my mind; It’s a simple concept, which can be expanded almost endlessly. The perfect base for a project!

I started by creating a simple use case diagram for the whole application with basic interactions for a user and also an admin. This diagram was finished pretty quickly, as there was not much to it, yet.

The second step to create was a class diagram, which was more basic than a reaction between nitrous-trihydrogen and water.

The diagram consisted of the main class and 6 more classes including a blueprint for the Account, which is the class with the least changes compared to final version of the project.

The third step was the simple sketch. I copied the class diagram, removed all the attributes and added some connections.

The fourth step was the sequence diagram. And I can tell you, I don’t like them. I tried to visualize the actions I had in my mind in the most simple way possible.

The conclusion of the whole planing process: It’s helpful in the beginning to know where to start, but after the creation of the first few classes, no one is looking at the diagrams again, as everything already is different.

The programming:

As already mentioned in the planing section: The diagrams were a help in the beginning, but were already forgotten by the time this project was converted into a maven project.

Mostly, the programming was pretty fun and I remember putting too much work in it.

I started by creating the planned classes and quickly noticed, that the project wouldn’t be enough complex for a good grade with only these classes implemented. I then did a little brainstorming with me, myself & I and came to the conclusion of transforming all simple classes and methods to more complex ones, which are performing much better and are more transparent than the old code.

After the first week I started watching tutorials and explanations about streams and functional programming. So, naturally, I implemented my newly learned skills into the project and I don’t think my project just looks and works better, but I as a software engineer grew massively.

As some more time passed, I started noticing that I would also have to persist the Accounts. I then started working on a simple, dynamic and fast method, which solves this problem.

After I finished most of the actions a user can use, I moved to the admin panel. There I started with a simple method to view the stats of the e-banking app, another one, which allows the admin to approve registered accounts and after that I coded a method used by both kinds of accounts to delete their account.

This was the “mostly-bug-free-zone” of programming.

I then proceeded to implement a GUI in the form of JavaFX, which was one of the biggest mistakes in this project. I have spent well over 6 hours trying to get everything to work on my Computer at my workplace. Nothing worked, because Maven was configured to read Class files of version 52.0 and jdk-17 uses class file of version 61.0.

I feel like, if I wouldn’t have had this problem, I would’ve implemented more of the in the beginning planned functionalities. But this big problem aside; I got everything to work on my Laptop and started creating simple front end scenes in the Scenebuilder. After that, I created controller for controller used to communicate between the back- and front end.

This was pretty easy to code, as I already separated the I/O from the back end. I only had to change the println statements to .setText in certain cases and I was good to go.

In this phase of programming I almost had a NullPointerException in having exceptions!

The phase of the encryption and decryption part was not quite like that. Funny people would probably say, that instead of a NullPointerException, the bugs almost created a StackOverflowError.

As this was a completely new library I wanted to implement I had to learn it’s functionalities. But this was everything but easy, as there were almost no good examples or tips on how to implement anything from the Google Tink library.

The testing & conclusion:

In between the two of those phases I created some unit tests, which worked fine.

I started by randomly choosing methods I wanted to test and writing simple unit tests with Mockito and JUnit5. That’s pretty much everything.

Overall the project was extremely fun and I think I learned a lot of important information for my future.

The bug fixing:

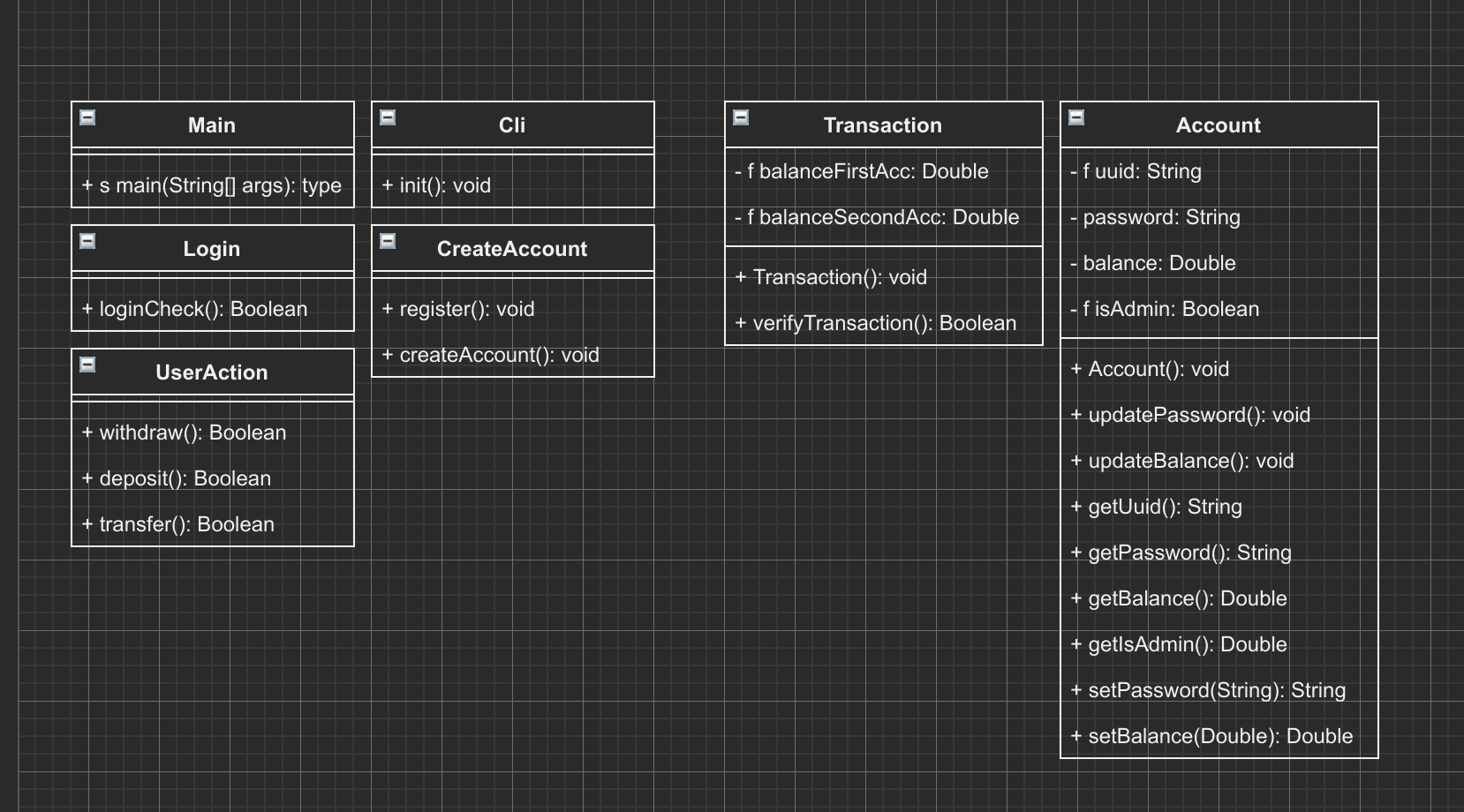
The only reason this is part of the programming has it’s own section is the fact that it was kind of funny.

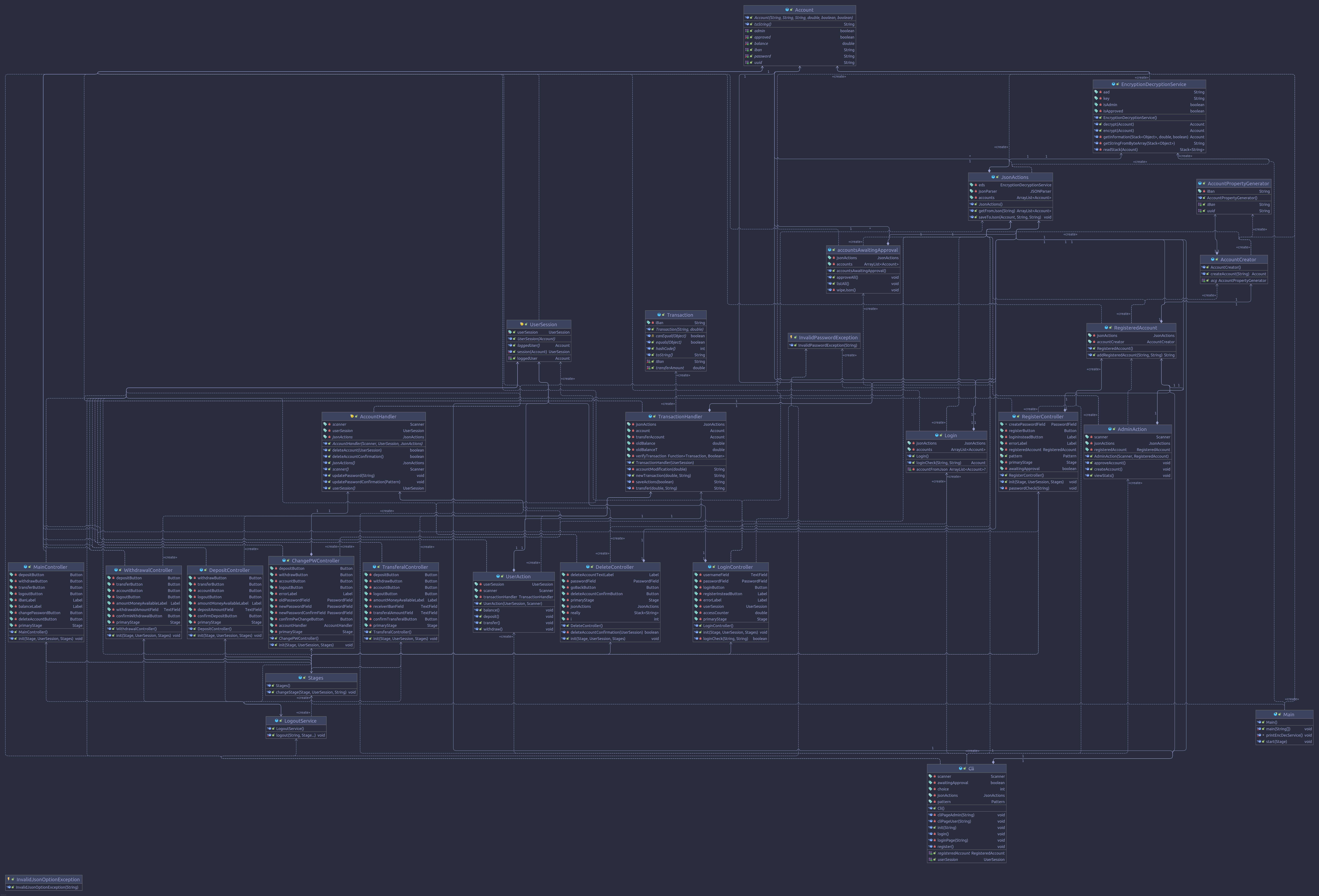
The most bugs I had, where bugs caused by stuff like a missing “new” before the initialization of a class or… as mentioned before, the whole Google Tink encryption & decryption stuff. And no. Stacktraces of Tink do not include any useful information or make any sense.

The diagrams:

Class diagram:

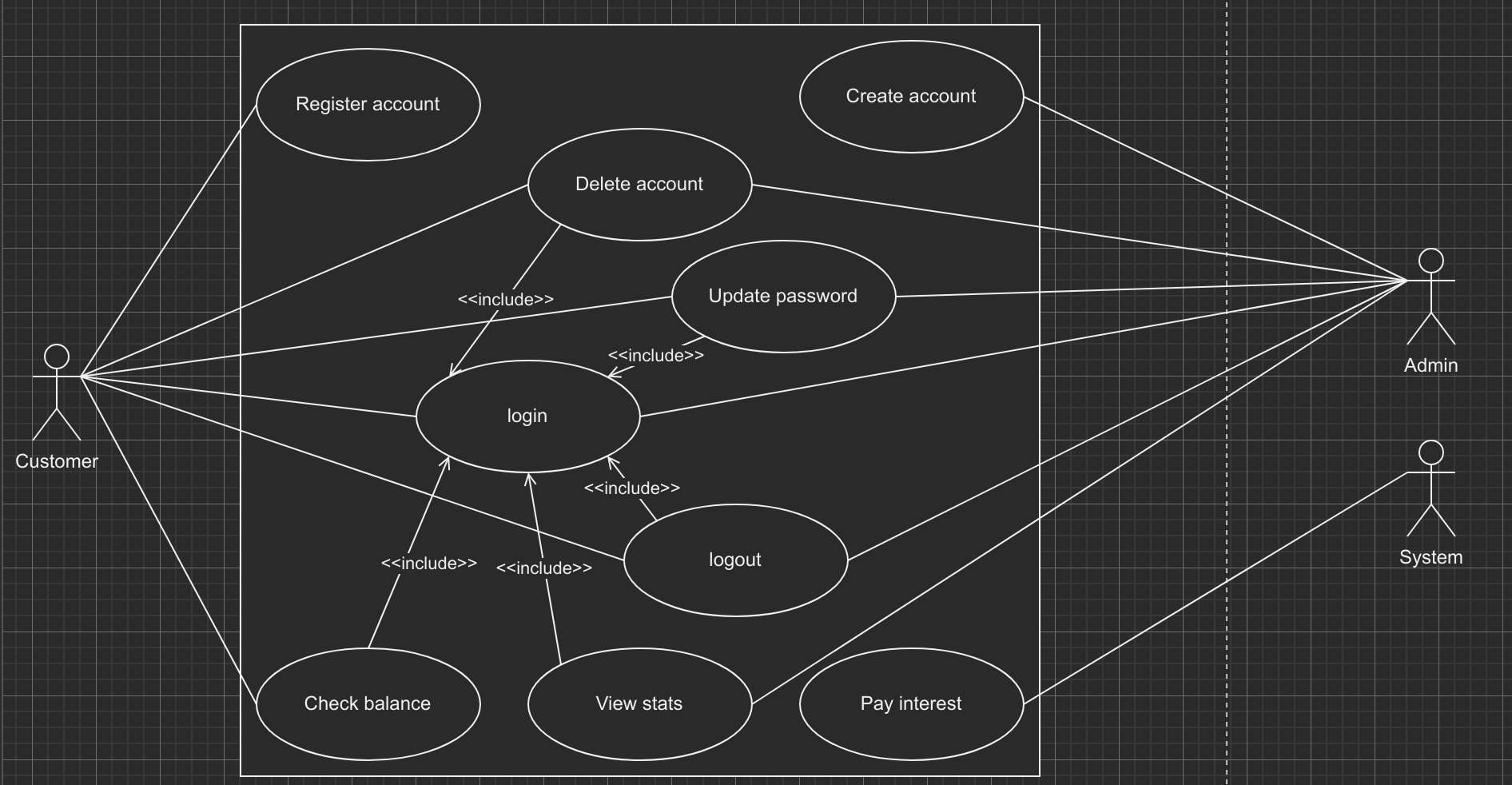
First:



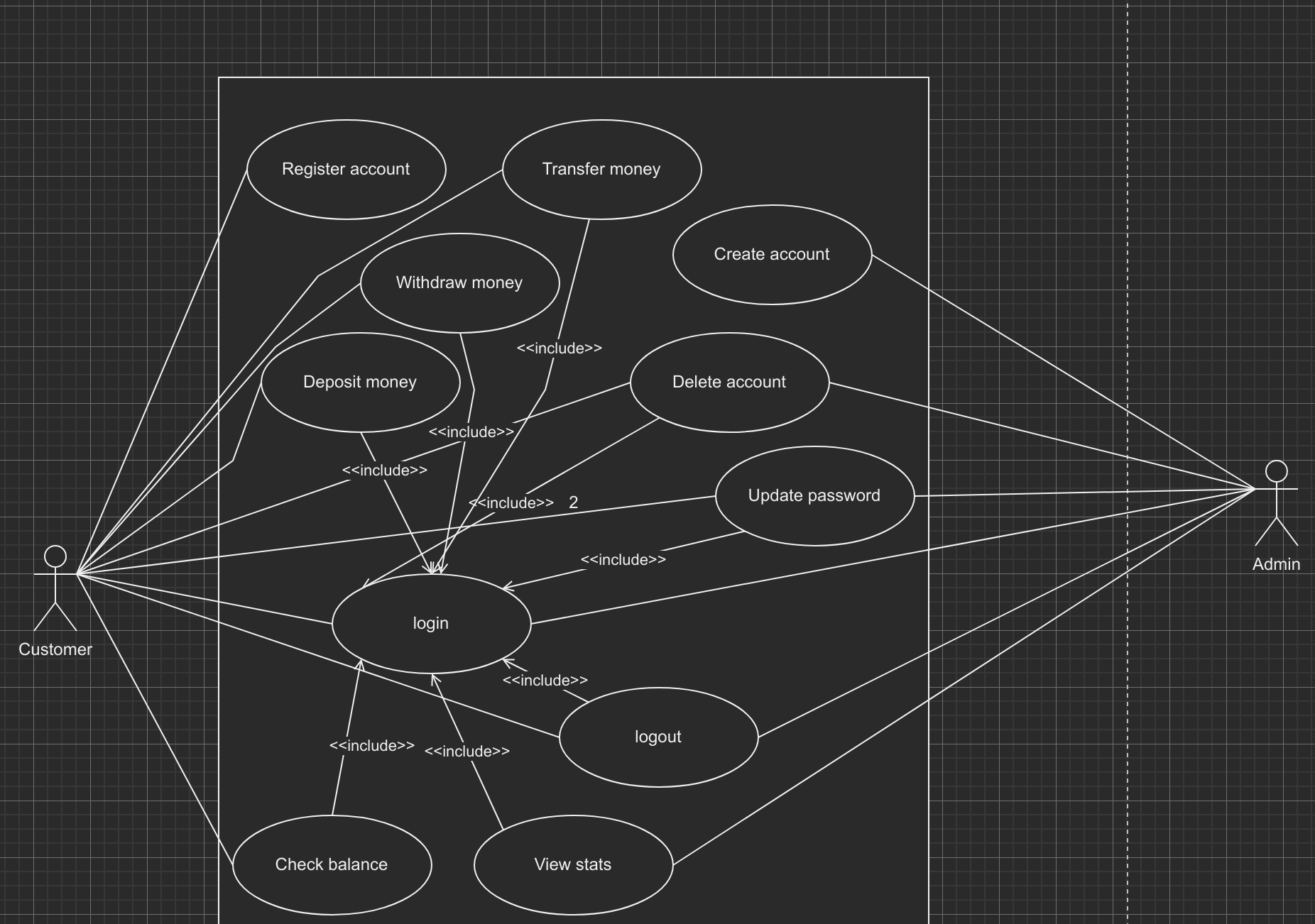
Final: (higher resolution in repo)

Usecase:

First:

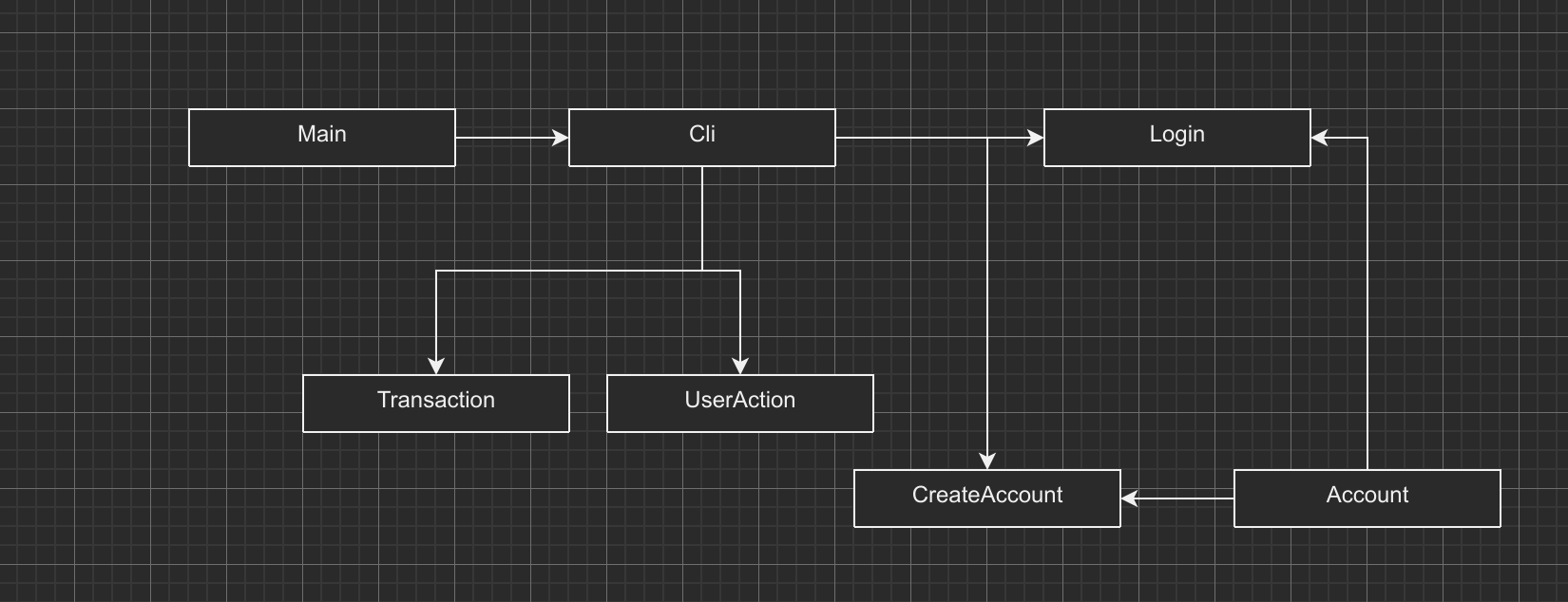


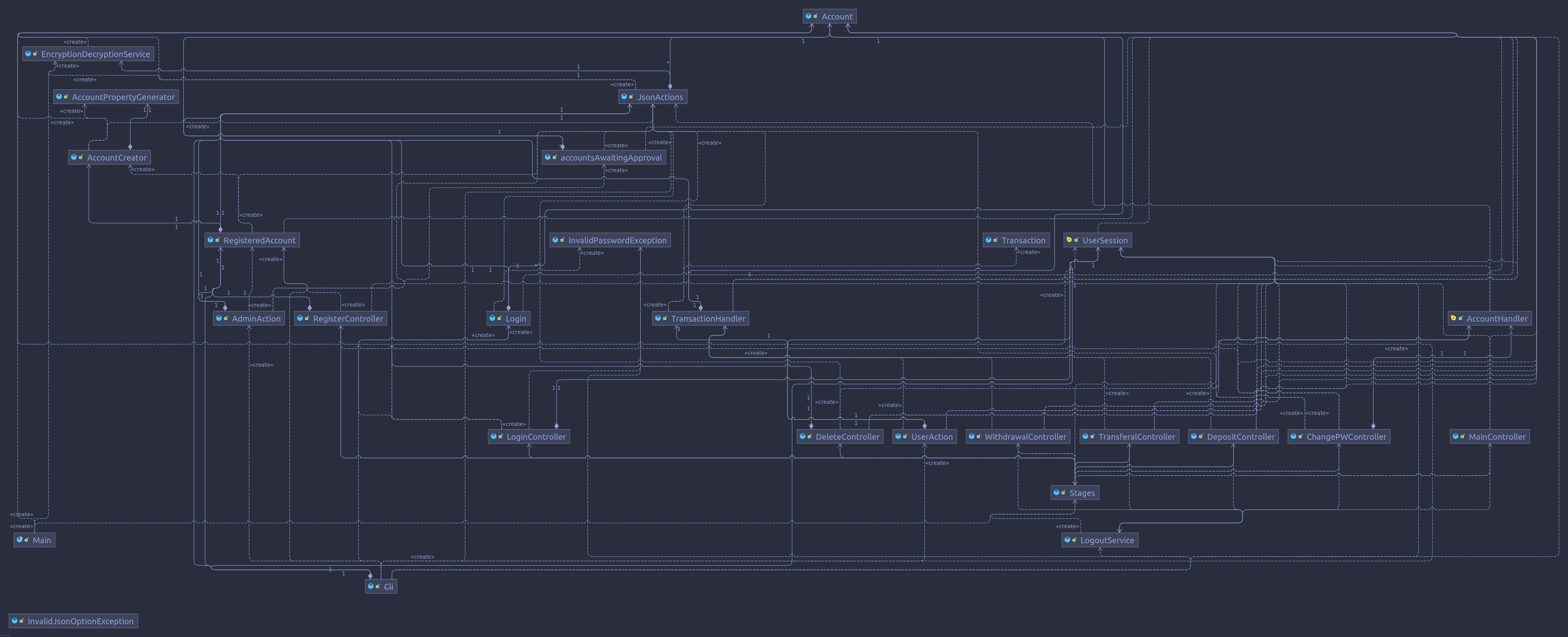
Final:



Sketch:

Rough:



Detailed:

Sequence:

Only version: (higher resolution in repo)

