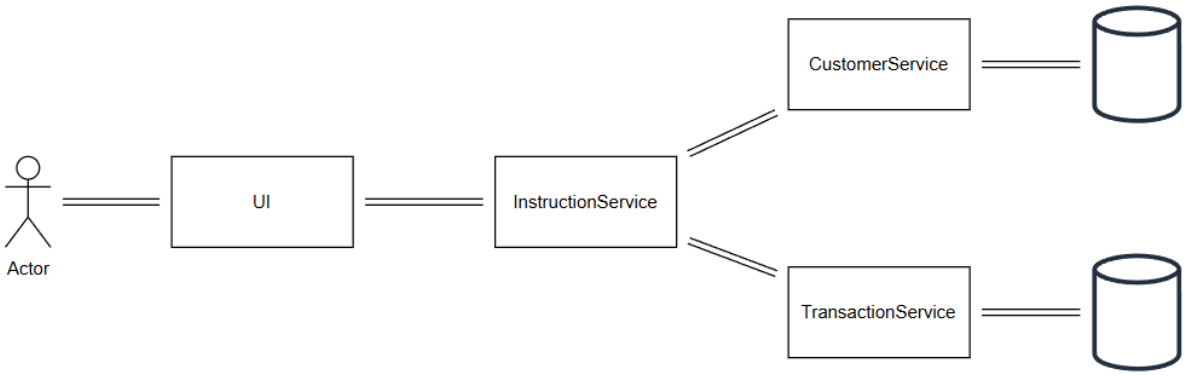
Discussion

Implementation Approach

As this is a short assignment I have chosen to use a monolith. However, in a more real-world scenario I would divide the code into services as follows:

1. UI Service
2. Customer Service
3. Transaction Service
4. Workflow Service (tie customer and transactions workflow together)



Segmenting the services into a microservices architecture would mean easily changing out a service or underlying DB easily without changing, building, and deploying the whole system. I have taken advantage of the DAO design pattern for this reason. The DAO implementation means the facade for the service remains untouched while the DB implementation can be freely changed.

Testing Approach

I believe testing the real application is the best approach rather than mocking/stubbing. Mocking and stubbing can lead to issues as mocks and real code need to be maintained together. Also, testing the real application like SpringBootTest gives better feedback as it is

the actual underlying implementation being tested. It does tend to be longer when running multiple tests as springboot needs to load for each but this I feel is worth it in the long run.

Lower-level code can be tested simply with Junit of course and for simple things mocking makes sense. A complete e2e test is worth doing though as when makes sense.

I have added tests for the CustomerController. Given a real-world example all controllers and DB interactions would be tested. Not just happy paths but negative scenarios and edge cases.

Regarding the UI there are many ways to test it from manual testing to libraries such as Cypress. The UI often gets overlooked unfortunately. This is almost always the window into which a user sees the application and as such needs to be rock solid.

Automation & Monitoring Possibilities

CI/CD is very important for timely feedback and building/deployment of applications. This application could be built, tested, and deployed from a CI/CD system such as Jenkins easily.

Monitoring can be done via plugins like Splunk and Grafana, or any number of other tools. I have added logging to the application that can be easily found in these systems e.g. [CUST] & [TXN].

Docker and K8s can be leveraged for rapid deployment and lifecycle management locally and in a CI/CD environment. They are widely used and have extensive documentation and online support available.

The monitoring and automation mentioned previously are very good for locally hosted/deployed application. Cloud services such as AWS have mechanisms in place specifically for monitoring, security, and lifecycle management that are readily available and useable with minimum setup.

Leveraging Artificial Intelligence (AI) & Machine Learning (ML)

AI and ML could be used to enhance the system and user experience. For example, AI could help predict areas of fraud in the system and mitigate user issues, loss of revenue for the customer and company, and enhance trust.

ML could be used to analyze patterns in user spending and suggest ways to improve fiscal health, cut down on expenses, or suggest to setup regular payments such as Direct Debits (DD).