Link to repo: https://github.com/PerdoGarcia/cs262/tree/main/ps2

Overall thoughts (the rest of the notebook is quite messy):

- Does the use of this tool make the application easier or more difficult?
    - Overall, gRPC makes the application easier, since I no longer have to handle the parsing of the inputs and outputs to each socket and can focus on the logic of the application
    - However, gRPC did make the application harder to program for specifically instantaneous delivery. I was not able to figure out a way to keep track of the clients using gRPC so that I can have the server notify a client in some situations like I could with sockets
    - This meant that we needed to change our instantaneous delivery solution to keep track of "instantaneous messages" using a list for each user instead.
- What does it do to the size of the data passed? (see experiments in the link above)
    - gRPC seems to pass slightly more data than our custom wire protocol but far less data than our json implementation
    - The message takes more time to deliver than the wire protocol all the time, but takes less time to deliver than a large json message and more time to deliver than a small json message
- How does it change the structure of the client? The server?
    - The client:
        - The client did not change much, but we were able to remove a lot of the parsing code needed for json and wire protocol versions
        - We needed a new type of request in order to implement "instantaneous messages"
    - The server:
        - The server needed to be refactored into a class, and because of that I needed to turn my helper functions for processing each different type of request into a class function
        - Also, because gRPC uses threads to service client requests, I needed to add a lock on the "database" parts of the server (the big dictionary storing all the accounts, messageId, etc.) in order to avoid race conditions. This was not needed in the socket version because the selector processed requests one at a time

- How does this change the testing of the application?
    - Unit tests are easier because we can start the server in the unit testing file + use stubs
    - Also less parsing and formatting needed for the tests, which is nice
    - Still there are overall the same test cases and test formats, so it's not that different

2/25/25

- Started writing gRPC unit tests:
    - https://stackoverflow.com/questions/51792592/how-to-write-unit-tests-for-your-grpc-server-in-python
- Unit tests are easier because we can start the server in the unit testing file + use stubs
- Also less parsing needed for the tests, which is nice

2/24/25

- Started implementing gRPC on server-side
- Realized that because a gRPC server uses threads I need a lock on the account dictionary and messageId (which are now stored as variables in a class) to avoid race conditions
    - Using with self.lock: format in order to have a lock guard and not worry about unlocking before return
- For implementing each of the grpc functions, basically needed to take the helper function I wrote for the json/wire protocol versions and write it into a class function
- Completed initial changes to use gRPC and tested that server started. Need a client in order to iron out any bugs
- Notable: do NOT have instantaneous delivery yet
- Fixed a few bugs in .proto where I accidentally forgot/included extra fields.
- Notes: the gRPC code on server side is a lot shorter than the json or wire protocol versions because I don't have to handle the parsing of the request and formatting of the response myself
- Figured out a different way to do instantaneous delivery that involves writing a new method on the server side that allows the client to continuously query for new "instantly delivered" messages
    - In my opinion, this makes the implementation of
- Changed the proto and the server code to implement this new method as needed
- Started documenting the server using docstrings
    - python -m pydoc -w server - just write the html
    - python -m pydoc -b server - generate a server
- Finished documenting the server

2/21/25

- Started by finding some resources for implementation
- gRPC reference:
  - Quickstart: https://grpc.io/docs/languages/python/quickstart/
  - What is GRPC: https://grpc.io/docs/what-is-grpc/introduction/
  - Basics: https://grpc.io/docs/languages/python/basics/
- Protocol buffers: https://protobuf.dev/overview/
- Created .proto files
  - Notably, needed to create some sub classes for representing accounts messages for read message
  - Not sure how to do instantaneous delivery yet - how can one gRPC call yield 2 responses?
    - After reading more, it seems like each "client" application also needs to be a server with gRPC calls that can be made to allow for instantaneous delivery
    - ^ NEED TO IMPLEMENT THAT LATER
- Command to generate proto files: python -m grpc_tools.protoc -I./ --python_out=. --pyi_out=. --grpc_python_out=. message_server.proto