Demo Day 1 Code Review: Aayush Karan & Michael Hla

ARTEMAS RADIK[1] AND SWATI GOEL[2]

[1] *Harvard College, A.B./S.M. Computer Science & Statistcs, artemas@college.harvard.edu*
[2] *Harvard College, A.B. Computer Science & Physics, sgoel@college.harvard.edu*

Compiled February 26, 2023

1. INTRODUCTION

This code review was conducted on commit `573fc69` of the repository submitted to us by Aayush and Michael, available publicly on GitHub. Aayush and Michael are reachable via email at `karan@college.harvard.edu` and `michaelhla@college.harvard.edu`, respectively. The code review aims to answer the following questions with regards to the submission:

1. Does it function as specified?
2. Is the code rational, correct, and clear?
3. Is the code and installation documentation adequate?

2. FUNCTIONALITY

We tested both the gRPC and non-GRPC versions of the program for the following functionaity.

1. Creating an account with a unique username.
2. List all accounts or a subset of the accounts by text wildcard.
3. Send a message to a recipient. If the recipient is logged in, deliver immediately; otherwise queue the message and deliver on demand. If the message is sent to someone who isn't a user, return an error message.
4. Deliver undelivered messages to a particular user.
5. Delete an account. You will need to specify the semantics of what happens if you attempt to delete an account that contains undelivered message.

Testing

We began by following the provided installation instructions in the engineering notebook, starting the non-gRPC server via `python3 socket_server.py 10.250.113.127 5000`. Our first test involved creating an account with the client as seen below.

Listing 1. Account Creation Test

```
python3 socket_client.py 10.250.113.127 5000
Welcome to Messenger! Please login or create an account:
Create Account
a
Account created. Welcome a!
```

We wanted to ensure that unique usernames were enforced, so we spawned another client and tried to create another account with the existing username.

Listing 2. Unique Username Test

```
python3 socket_client.py 10.250.113.127 5000
Welcome to Messenger! Please login or create an account:
Create Account
a
The account a already exists. Please try again.
```

We were satisfied with these two tests as evidence of account functionaity, so we began testing the username listing feature. As a prerequisite, we created additional accounts with usernames `blueberry`, `cranberry` and `elderberry`. Per the engineering notebook, the username listing feature follows Python RegEx. Thus we began by listing usernames following the `.*` expression, which we expected to return all usernames. We then listed usernamed following the `^.*berry.*$` expression, expecting the last three usernames. We were highly impressed with the RegEx matching functionaity.

Listing 3. Account List and Filter Test

```
python3 socket_client.py 10.250.113.127 5000
Welcome to Messenger! Please login or create an account:
List Accounts
.*
Users matching .*:
a blueberry cranberry elderberry
List Accounts
^.*berry.*$
Users matching ^.*berry.*$:
blueberry cranberry elderberry
```

The next test involved sending a message from the logged in `blueberry` client to the logged in `cranberry` client.

Listing 4. Send Live Message Test

```
python3 socket_client.py 10.250.113.127 5000
Welcome to Messenger! Please login or create an account:
Create Account
blueberry
Account created. Welcome blueberry!
Send
To:
cranberry
Message:
hello there, cranberry.

Message successfully sent.
```

The `cranberry` client successfully received the message.

Listing 5. Receive Live Message Test

```
python3 socket_client.py 10.250.113.127 5000
Welcome to Messenger! Please login or create an account:
Create Account
cranberry
Account created. Welcome cranberry!
<blueberry>: hello there, cranberry.
```

Next we logged out `blueberry` and sent an outbound message from `cranberry`. Upon re-logging in `blueberry` we requested our undelivered messages via `Open Undelivered Messages` as described in the engineering notebook.

Listing 6. Receive Undelivered Message Test

```
python3 socket_client.py 10.250.113.127 5000
Welcome to Messenger! Please login or create an account:
Login
blueberry
Welcome back blueberry!
Open Undelivered Messages
<cranberry>: Hi, blueberry!
```

The blueberry client successfully received the undelivered message. Sending a message to a non-existent user also correctly returns an error as seen below.

Listing 7. Send Message to Nonexistent User Test

```
python3 socket_client.py 10.250.113.127 5000
Welcome to Messenger! Please login or create an account:
Login
blueberry
Welcome back blueberry!
Send
To:
gooseberry
Message:
do you exist?
Sorry, message recipient not found. Please try again.
```

The last remaining functionaity we tested for was deletion, as seen below on the `blueberry` account.

Listing 8. Account Deletion Test

```
python3 socket_client.py 10.250.113.127 5000
Welcome to Messenger! Please login or create an account:
Login
blueberry
Welcome back blueberry!
Delete Account
Account blueberry successfully deleted.
Welcome to Messenger! Please login or create an account:
```

Thankfully, the gRPC submission command structure follows extremely closely to the non-gRPC one. The only distinction is that no prompt is required to automatically deliver queued messages on the gRPC implementation. We ran similarly extensive testing on the gRPC implementation and verify that all functionaity is present, but have omitted those tests from this report as they repeat much of what is already present.

3. CODE RATIONALITY, CORRECTNESS, AND CLARITY

Non-gRPC Implementation

Design is well-thought out and code is reasonably structured and organized. The code goes a long way to acheive correctness, implementing threading locks, timeouts, and several error-catching if statements. For instance, the server implements an authentication scheme through the use of a `logged_in` variable. Certain client-facing features are available only once this variable has been toggled, while other client-facing features are available to only unauthenticated users.

Furthermore, significant error-checking also occurs client-side. For instance, even though the server wouldn't allow a logged in user to log in as somebody else, the client program also checks for this state before asking the server to perform a log in. This could offer performance benefits at scale by shifting much of the processing from the server to the clients.

Many edge cases are also accounted for. For instance, the server properly handles situations where a client

1. Terminates process via CTRL-C.
2. Trys to login in two locations simultaneously.
3. Sends a malformed input.

gRPC Implementation

On rationality: Very well thought out design. A lot of effort went into making the code safe, for example limiting message size:

Listing 9. Code Safety Example

```
# check length requirements
if len(request.message) > 250 or len(request.receiver) > 50:
  res.status = -1
  return res
```

Python dictionaries are concurrency safe. But the project goes above and beyond in handling concurrent requests. If there is extra credit, would flag this project as a contender. One note: the design could better leverage the inherent structure of gRPC. The strength of gRPC is the ability to call functions/specify operations on a remote server. Fully leveraging this functionality would simplify the code, by removing some of the work being done on the server. For example, this would make multithreading on the server unnecessary.

On correctness: Create Account and Login functionality are tied together (ie creating an account automatically triggers login). Our implementation was different, but I think theirs is a better design choice. Attempting to list 0 users from the List Account function sometimes fails, but this is fine, because providing more information than asked for isn't problematic. Everything else works as specified.

On clarity: The command structure was intuitive and code was well spaced. What we saw in the code paired well with what we read in the documentation.

4. DOCUMENTATION

The documentation of the project is excellent, both in the code and the installation documentation. The code is well-documented throughout, with helpful comments that explain what is happening below in both the gRPC and non-GRPC implementations. These comments adequately annotate error handling, multithreading, and function capabilities.

The installation documentation was precise and provided us with direct commands to run the provided programs, and even accounted for edge cases such as users on M1-based Mac systems. We were impressed by the extensiveness of the installation and usage documentation, and were happy to have this when testing both implementations for meeting the required specifications.

5. GRADING

Given the excellent work and ability to adequately satisfy the project requirements across the three evaluation criteria, we are happy to award this project a **25/25 score**.