Group 1: Harrison & Henry (Table 2) Score: 4.9/5

Functionality (2/2): The chat application met all core requirements. During demo day, we were shown proof of the app operating on both localhost and over the Internet. It handled sending, receiving, and deletion of messages without issue. The GUI, built with Tkinter, follows a Gmail-style layout with recipient and text area fields, along with an inbox that can be refreshed manually, but also works for live message delivery. Messages are visually distinguished by color based on whether they were sent or received. The Settings tab allows the user to adjust the displayed message count. A nitpick is that we observed a minor delay when sending requests through the localhost server (most likely tied to how often the client GUI refreshes for new messages?). This is a very minor concern however and doesn't impact functionality per se. Having to manually refresh the inbox, while a fine design choice, isn't the best for a chat app?

Documentation (0.9/1): The documentation is comprehensive and clear. Endpoints are explicitly defined, and the SQLite database schema is well described. A configuration file is used effectively, allowing parameters such as message limits or specific protocol magic numbers to be easily adjusted. I liked their approach, as opposed to just hardcoding the protocol's parameters in the implementation itself. The code was well documented, with docstrings and in-line comments throughout. We were not told during demo day what the space savings were when comparing their JSON protocol to their custom one (nor did we see it in the documentation), so we docked points for that.

Tests (1/1): We'd call the group's tests as more than adequate. We were shown a quite robust suite of unit tests, many of which utilizing Pytest fixtures. The tests seemed to thoroughly validate the messaging, client/server interactions, and database invariants (such as preventing duplicate accounts or conversations). They systematically test for edge cases too: for example, deleting the final message in a conversation should result in the removal of the conversation itself. No stress-testing the application under heavy-load was showcased, though the code itself seemed to be designed with scalability in mind. Since the rubric doesn't mention that as a requirement, we treated it as a nice to have, and didn't deduct any points.

Code Clarity (1/1): The codebase is well structured and highly readable. They use an OOP approach, and demonstrate deep knowledge of Python (I saw multiple instances of decorators like @classmethod). Asynchronous I/O is managed via a threadpool on the backend server, while synchronization is ensured through a queue.

Group 2: Jayson & Edward (Table 2) Score: 4.5/5 (due to minor nitpicks, see below)

Functionality (1.8/2): The application demonstrates reliable client/server communication on both localhost and over a network. We were shown both a GUI and CLI clients during Demo Day. Live message delivery worked correctly, with a clear distinction maintained between read and unread messages. The functionalities specified by the problem statement were all implemented: password hashing, account creation, account search, and deletion all perform as specified. The Gmail-style interface features a text field for specifying the number of unread messages, and the CLI client supports this via an "unreadcount" command. A small nitpick from us: during demo day no handling of edge cases was shown during the Demo part of the showcase, but they did seem to have code handling it when we looked at it during code review. The GUI was also slightly unintuitive and it wasn't clear to me how to list accounts other than when trying to send a message, so we deducted another 0.1 points for that.

Documentation (0.9/1): Documentation is thorough and precise. A detailed wire protocol was shown to us in a Google Doc, outlining all the op_codes used by the protocol, required payload fields, and their descriptions. The code structure was well documented, with docstrings throughout. Same as the previous group, they don't specify what the savings ratio is between the JSON protocol and their custom wire protocol (at least not during demo day).

Tests (0.8/1): A comprehensive test suite resides in its own /tests directory. The tests we saw covered a wide range of functionalities—from client/server actions to the messaging system and account management. While not as thourough as the first group's we reviewed, the tests were definitely 'adequate' (see the grading rubric's wording), which is why we only deduct a small fractional amount of points here. Also, no stress testing for performance was showcased, similar to the first group. If we didn't do fractional grading, we'd consider this group to have adequate tests and assign full scores for this category.

Code Clarity (1/1): The code is exceptionally clean and well organized. All class methods are clearly defined and well-explained (we saw many docstrings throughout the codebase). They show a lot of foresight with a easily replacable protocol, and supporting version numbers in their wire protocol. They also heavily relied on OOP (e.g., they implement a ChatClient class and its methods like connect(), listen(), and handle_server_response()), and all the classes and their methods are nicely explained from within the implementation.