Vision:

For MS1, our idea was to create an online text editor which would allow multiple clients to work on the same doc at the same time, and it would support many of the common text editing features such as text alignment or a multitude of fonts. The vision is still largely the same as we desire that this text editor is able to save files to a server such that the file is never stored locally. It should, of course, also be capable of the text modifications that are standard in Microsoft Word and Google Docs. We do not want this program to run simply out of the terminal as we want to implement GUI to customize the interface according to our wishes.

Summary of Progress:

Our submission for MS2 has provided functionality for collaborative text editing. Precisely, we have been able to implement the
server-client infrastructure such that the user is essentially building a long string or collection of strings which are sent to
the server via a button. The server then saves those strings on a file that is only stored on the server. We have also
successfully implemented the synchronization logic required to handle multi-user cursor movement, ensuring that one user's edits
correctly shift the cursors of other active users. Currently, deletions are supported, though we are refining the interaction
with complex text selections(like using shift to select multiple chars etc.).

Activity Breakdown:

Jacob Kupperman

- worked on the text features
- provided functionality for bolding and italicizing
- implement text deletion
- synthesize the code with compatibility with Bogue GUI features

Stanley Amkhanitsky:

- Defined shared message types in events.mli/events.ml (user_action and backend_update) to standardize communication between frontend, backend, and network clients.
- Implemented a Lwt-based backend server in backend.ml that accepts multiple clients, parses user actions, and serializes document updates using an Lwt_mutex.
- Added an authoritative document_state and per-client cursor tracking, including logic to update all clients' cursors correctly on insertions and deletions.
- Designed a simple line-based wire protocol (KEY, BACKSPACE, SET, SAVE, FULL, DOC) to encode/decode actions and document updates over sockets.
- Implemented broadcast logic so that any change from one client is propagated to all connected clients, starting our path to basic real-time multi-user editing

Muhammad Ali:

- Implemented dedicated lib/backend/document.ml to isolate the core document logic from the Lwt networking layer

- backend logic for text deletion now supports deletion of characters from anywhere
- Implemented the apply_local_change algorithm, which handles character insertion and deletion while mathematically calculating
  cursor shifts for all concurrent users.
- Restructured the backend to manage user locations via specialized Document.cursor objects, removing the dependency between
  text positioning logic and the Lwt network handles.

<u>Ratchaphon Lertdamrongwong:</u>
- Implemented dedicated lib/frontend/frontend.ml to deliver GUI interface for user, a simple window with text box, buttons. User
  can now type in the GUT textbox, send the text which utilize the backend functions to sync acrossed all clients and relay the
  changes back for the frontend to display the change to all client in real-time, currently italicizing and bolidng is not yet
  integrated into GUT.
- Implemented Unix-socket clinet
- Added mutex for thread-safe state managment for shared resources
- Added command line terminal entry point to start a server and client, currently the ip is hard-coded to `127.0.0.1:9001` for
  easy testing.
- Added graceful shutdown with `crt-c` for server in the cmd line terminal

<u>Productivity Analysis:</u>
We accomplished most of what we had planned to for PM1. However, we made some goal changes for the text features. Instead of focusing on color and indenting. We decided to look at more fundamental features like bolding and italicizing and deleting. This will allow users to manipulate the text at the most basic level setting us up to implement more complicated features. Currently, the GUI interface is implemented as a pop-up window. Unfortunately, we have not progressed enough to implement real-time editing, so users must manually upload the text one string at a time instead of in real-time, one character at a time.