Task Allocation

1. User Authentication & Authorisation
TODD
—> ideally most being done by 22nd of Nov.

Responsibilities:
- Implement authentication (login/logout) for **Admin**, **MHWP**, and **Patient**.
- Ensure session management and validation of username/password pairs.
- Handle edge cases like failed login attempts.

- Focus on the user authentication and session management. Ensure smooth login/logout experience for different user types.
- Consider adding password recovery or reset options as an enhancement.

2. Admin Features & System Management
JAMES (paired with Peace if needed)
—> ideally most being done by 22nd of Nov.

Responsibilities:
- **User Management**: Admin can allocate patients to MHWPs, edit, delete, and disable user records.
- **Reporting**: Admin can display summaries of all related data, including patient allocations and weekly bookings.
- **Advanced Features**: Integrate additional admin functionality, like auditing changes to user data (who changed what and when).

- Handle the core admin functions and ensure that users can be allocated, edited, and disabled properly.
- Implement data summary and reporting features for the admin.

3. New Features, Innovation & Design BRUCE

Responsibilities:
- **Health Progress Tracking (Patient)**: Allow patients to track and visualise progress (e.g., therapy sessions, mood trends over time).
- **Patient Feedback System**: Add a feature that allows patients to leave feedback after their sessions with MHWPs.
- **Advanced Dashboard (MHWP)**: Create an advanced dashboard for MHWPs, including additional metrics like therapy progress, feedback scores, etc.
- **Meditation & Relaxation**: Enhance the meditation section with new sources, categories, or features (e.g., personalized recommendations).
- **Other Novel Features**: Research and propose other novel features that could be beneficial for the platform (e.g., AI-driven mood predictions, real-time chat with MHWP).
- **Overall Design of the Application**: Taking a step back and seeing what can be improved on the overall design and experience of the application.

- Focus on building new features and enhancements that improve the functionality of the platform, especially in areas that were not originally mentioned in the requirements.
- Consider adding personalized recommendations or advanced progress tracking for patients and MHWPs.

---

4. Patient Self-Management & Health Tracking

OLA & FRAN

—> ideally most being done by 22nd of Nov.

Responsibilities:

- **Edit Personal Information**: Allow patients to update name, email, and emergency contact.
- **Mood Tracker**: Implement a color-coded mood tracker, where patients can describe their mood and add comments.
- **Journal Entries**: Enable patients to write and save journal entries with timestamps.
- **Meditation & Relaxation**: Provide access to meditation exercises with external sources (URLs or embedded).
- **Advanced Features**: Allow patients to track their overall health progress (e.g., weight, sleep patterns, etc.).

- Build and manage self-management tools like mood tracking, journaling, and meditation resources.
- Consider adding a way for patients to set reminders for journaling or mood updates as an enhancement.

---

5. Appointment Management & Patient-MHWP Interaction

PEACE (paired with James if needed)

—> ideally most being done by 22nd of Nov.

Responsibilities:

- **Booking and Canceling Appointments**: Implement booking and canceling of appointments with a calendar interface.
- **Confirmation**: MHWP must confirm or cancel appointments, and both parties (patient and MHWP) will receive email notifications.
- **Appointment Notifications**: Implement email notifications for appointment confirmations, cancellations, and reminders.

- Build and manage the patient-MHWP interaction for appointment scheduling and confirmations.
- Integrate email functionality for appointment notifications and reminders.

6. Data Management & Persistence (Data Structures & Overall Architecture) TIM

- Example Responsibilities:
- **Data Persistence**: Ensure that all changes to data (e.g., user information, appointments, mood tracker) are saved and persist across sessions.
- **Data Integrity**: Ensure realistic data entry (e.g., valid dates, mental conditions).
- **Data Structures**: Choose and implement appropriate data structures for persistent storage (e.g., classes, dictionaries, lists) and manage file-based or database storage.
- **System Architecture**: Organize the overall structure of the application. Create modules, organize the codebase into manageable components, and ensure clear separation of concerns.
- **Advanced Data Management**: Explore additional storage options (e.g., SQLite for persistence, or a more sophisticated database if allowed).
- Assigned to: Member 6
- Focus on managing the data structures and ensuring that all system data (user records, appointments, mood data) is stored efficiently and persists across sessions.
- Handle any advanced data structures or database needs for persistence.
- Ensure that the overall architecture is clean, maintainable, and modular.

7. Video Presentation, Documentation, & Testing FRAN (and someone who feels like doing a bit more - testing and debugging; for now OLA)

Responsibilities:

- **Video Presentation**: Prepare the video demonstrating the application's features and installation process, with a voiceover or text explanations.
- **Documentation**: Create clear documentation for the project (including installation guides, feature explanations, and code comments).
- **Testing**: Write and execute tests for all features, ensuring the system works as expected (unit tests, integration tests).
- **Final Testing & Debugging**: Ensure the entire system is functional and bug-free before submission. Perform a final round of testing on all features.

- Focus on creating the video and documentation for the project.
- Perform final system-wide testing and ensure the project meets all technical and functional requirements.
- Provide a detailed walk-through of features in the video and ensure it meets the submission criteria.

Collaboration Considerations

- **Communication**: Regular meetings and check-ins to track progress and ensure features are aligning.
- **Code Reviews**: Regular peer reviews of code to ensure quality and avoid integration issues later.
- **Documentation**: Keep detailed notes and comments throughout the code to facilitate integration and future debugging.

Workflow and Dependencies

- Generally, everyone can start working on their parts right away, keeping in mind that code updates will be needed along the way.
- Later on, if needed, an overall system manager will be chosen to ensure all parts of code are working together.
- Same with finalising the front-end (depending whether Todd is sick of it or not…)

- Data storage: JSON and Pandas; all members should work on their features and update the code when Tim has done centralised data storage.

(Suggested dependencies)

- Todd (Authentication) and James (Admin Features) can begin first, as the authentication and admin functionalities are foundational.
- Ola (Patient Features) and Peace (Appointment Management) can then start after the user authentication system is in place.
- Bruce (New Features) and Tim (Data Management & Persistence) can work concurrently, but the data management and architecture work needs to be done early to ensure a solid foundation for all other features.
- Fran (Video & Testing) should ensure everything is well-documented and conduct final tests on the whole system.

Tim’s proposed approach to data structures:
- define our entities, relations, and attributes (e.g. user with userId, firstName, lastName, responsibleMHWPId etc)
- set up a database class that uses dataframes internally for saving, editing and providing that data upon request and saves/loads it between sessions
- set up dedicated functions that serve as interfaces between your code and the database class (e.g. storeUser(userArgs**), editUser, etc.) so that no-one has to directly change/edit the database to avoid chaos
- set up *integrity checks upon saving and editing that throw ValueExceptions* when entered data is invalid which _you will handle with try except blocks_ in the parts of code that handle user input