MoodJourney: Final Report

Team Members: Michael Tin, Augustine Ayoub, James Krejci, Isaias Bernal, Eddie Nguyen, Falak Tulsi (“MAJIEF”)

MoodJourney is a modern, digital and user-friendly journaling application that promotes self-reflection through features such as sentiment analysis and mood trend visualization. This report presents the complete, working implementation of the application.

GitHub Repository	Documentation

I. Installation and Setup

A. Minimum System Requirements

- Intel Core i5 Processor or AMD Ryzen 5 Processor
- 8GB RAM
- 256GB SSD (NVMe or SATA)
- Windows 10 21H2 (64-bit), macOS 10.15 Catalina, or a modern Linux distribution (i.e. Ubuntu 20.04)
- Internet Connection

B. Prerequisites

- Git - Installation
- Python - Installlation
- Visual Studio Code - Installlation
- Node.js - Installation
    - To verify installation, open a Command Prompt/Terminal and run `node -v` and then `npm -v`.
- Rust - Installlation
    - To verify installation, open a Command Prompt/Terminal and run `rustc –version` and then `cargo –version`.

On Windows, you can download Git, Python, and Visual Studio Code through Ninite to simplify the installation process.

C. Platform-Specific Dependencies

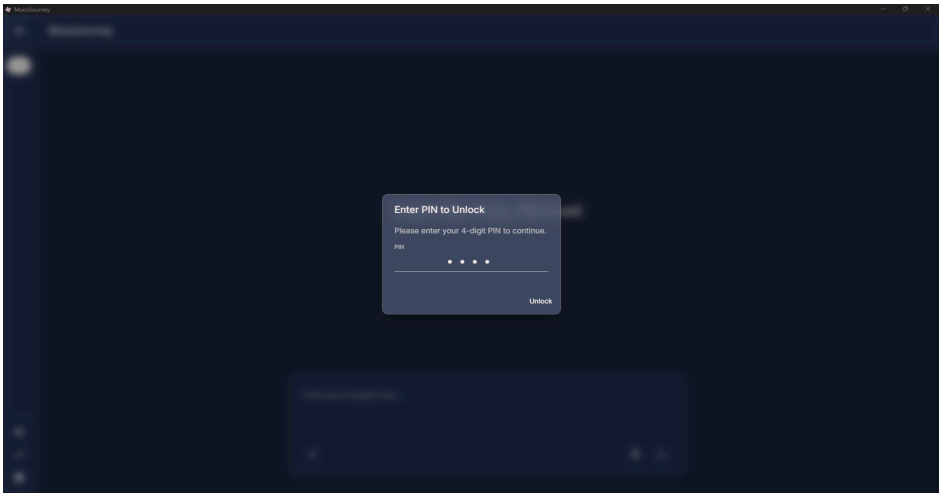
- **Windows**
  - Visual Studio 2022 - Installation
    - **When prompted by the installer, select the "Desktop development with C++" workload.**
  - LLVM 18.1.8 - Installation
    - **When prompted by the installer, ensure to select "Add LLVM to the system PATH for all users".**
    - To verify installation, open a Command Prompt and run `clang -v`.
  - CMake - Installation
    - **When prompted by the installer, ensure to select "Add CMake to the system PATH for all users".**
    - To verify installation, open a Command Prompt and run `cmake --version`.
- **macOS**
  - Xcode Command Line Tools
    - Open a Terminal and run `xcode-select --install`.
  - Homebrew - Installation
  - LLVM and CMake
    - Open a Terminal and run `brew install llvm cmake`.
- **Linux**
  - Essential Build Tools
    - Open a Terminal and run `sudo apt-get update` then `sudo apt-get install -y build-essential`.
  - LLVM
    - Open a Terminal and run `sudo apt-get install -y llvm`.
  - CMake
    - Open a Terminal and run `sudo apt-get install -y cmake`.
  - GLib Development Libraries
    - Open a Terminal and run `sudo apt-get install -y libglib2.0-dev`.
  - GTK 3.0 Development Libraries
    - Open a Terminal and run `sudo apt-get install -y libgtk-3-dev`.
  - WebKitGTK Development Libraries
    - Open a Terminal and run `sudo apt-get install -y libwebkit2gtk-4.1-dev`.
  - libsoup Development Libraries
    - Open a Terminal and run `sudo apt-get install -y libsoup-3.0-dev`.

D. Installation

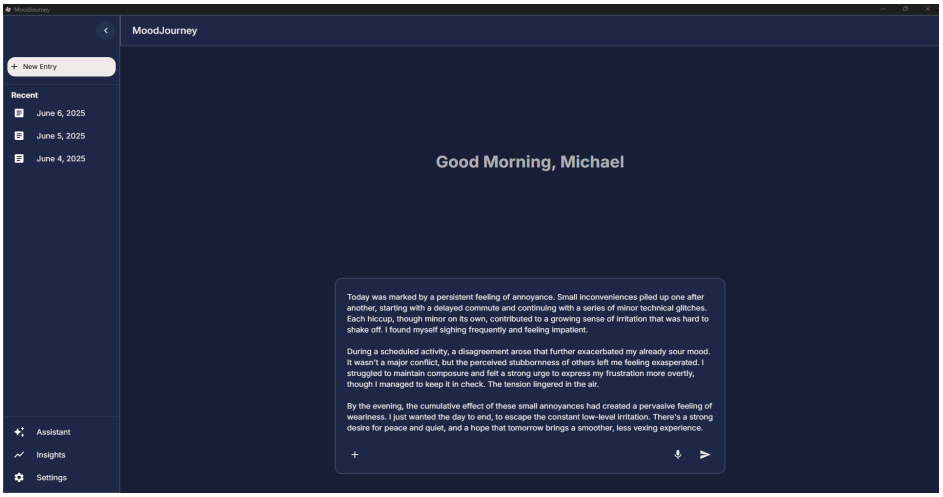
- **IMPORTANT: If you are on Windows, please use a Developer Command Prompt for VS 2022 for the installation.**
- Download and install the necessary Python packages to run the application. **Ensure that these three commands are run from the home directory in your Command Prompt/Terminal.**
    - `pip install requests`
    - `pip install transformers`
    - `pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cpu`
- Clone the repository and navigate to the project folder directory.
    - `git clone https://github.com/MAJIEF-CS180/MoodJourney.git`
    - `cd MoodJourney`
- Download the WhisperAI dictation model and the DistilRoBERTa sentiment analysis model from HuggingFace.
    - `cd scripts`
    - `python download_model_dictation.py`
    - `python download_model_emotion.py`
    - `cd ..`
- Navigate to the `src-tauri\src` directory. You will find a file named `config_template.rs`. Make a copy of this file and name the new file `config.rs`.
- You will need to generate a Gemini 2.0 Flash API key in order to run the Assistant feature. This API key can be generated from the Google AI Studio website. You will need a Google Account.
- In `config.rs`, replace `YOUR_GEMINI_API_KEY_GOES_HERE` with the API key that you generated.
- Navigate to the project folder directory again. Download and install the Tauri CLI. Verify the installation.
    - `npm install -g @tauri-apps/cli`
    - `tauri --version`
- Download and install the Node.js dependencies.
    - `npm install`
- Build and run the application.
    - `npm run tauri dev`

II. Implemented Features

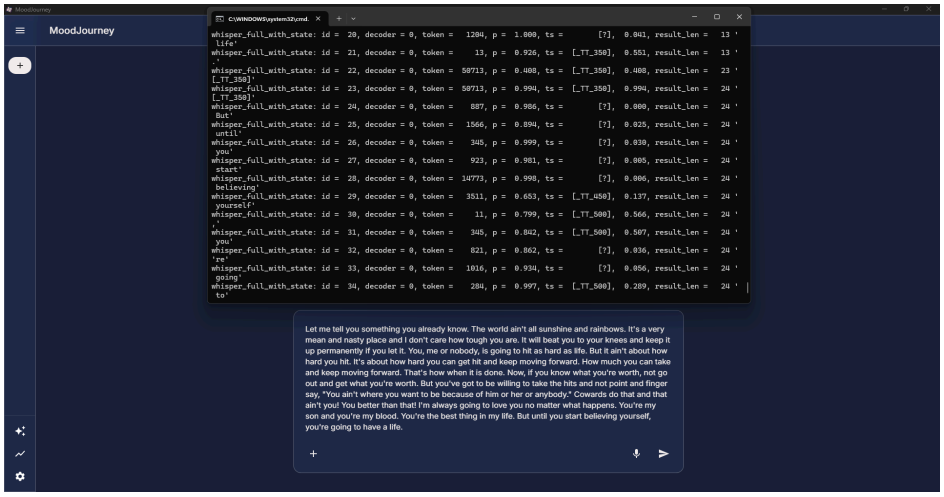
All major functionality and features listed in the original Project Proposal were successfully implemented. These include the following:



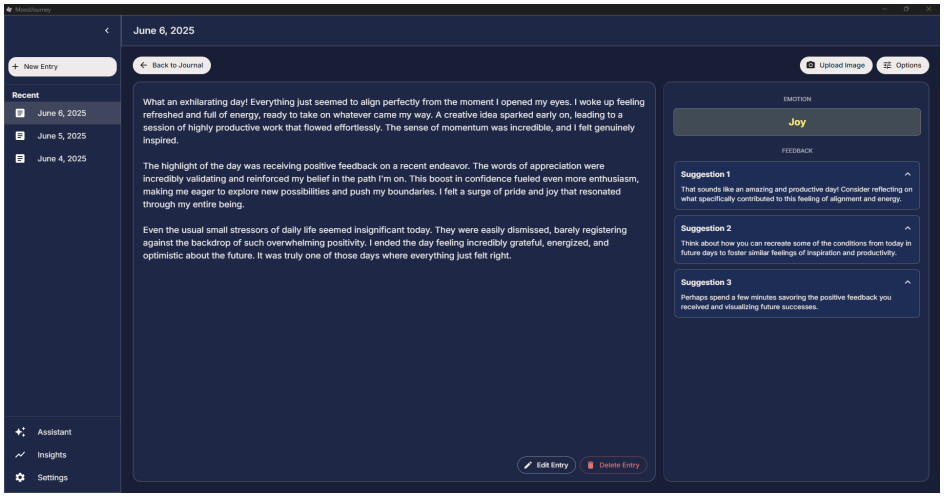
Authentication: The user is able to provide and use an optional PIN that unlocks access to the journal, similar to a keylock on a physical notebook.



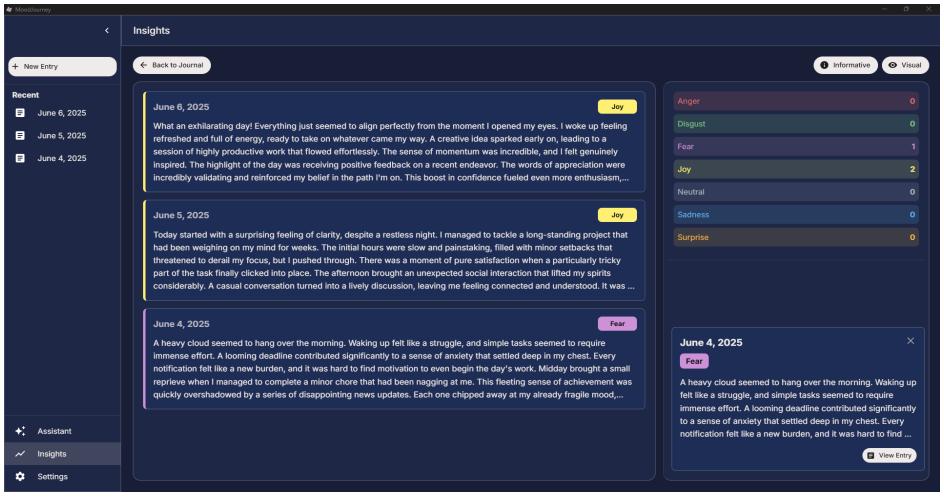
Date Tracking/Text Entry: The user's journal entries are automatically dated, allowing for easy organization and sorting. The user is able to type their journal entries in a simple and user friendly interface.

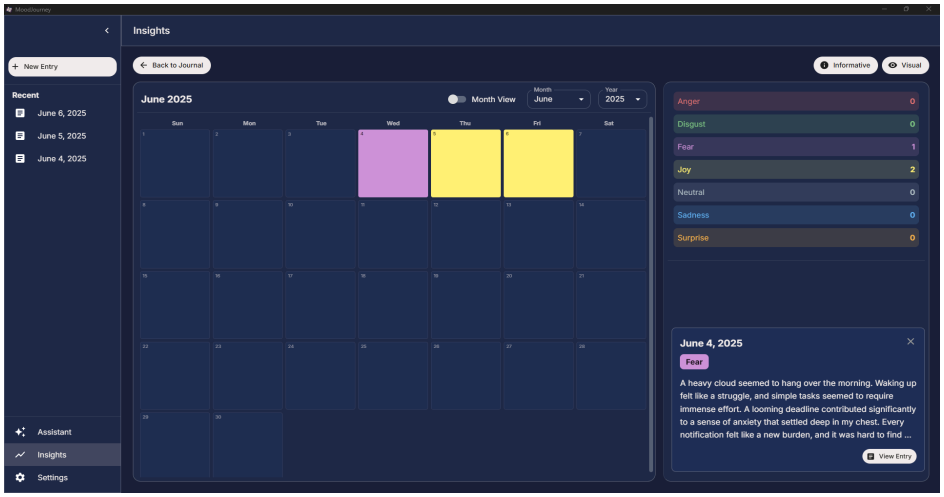


Voice-to-Text Transcription: We use WhisperAI to power a voice-to-text transcription feature. The user is able to dictate audio files as an alternative, convenient method.



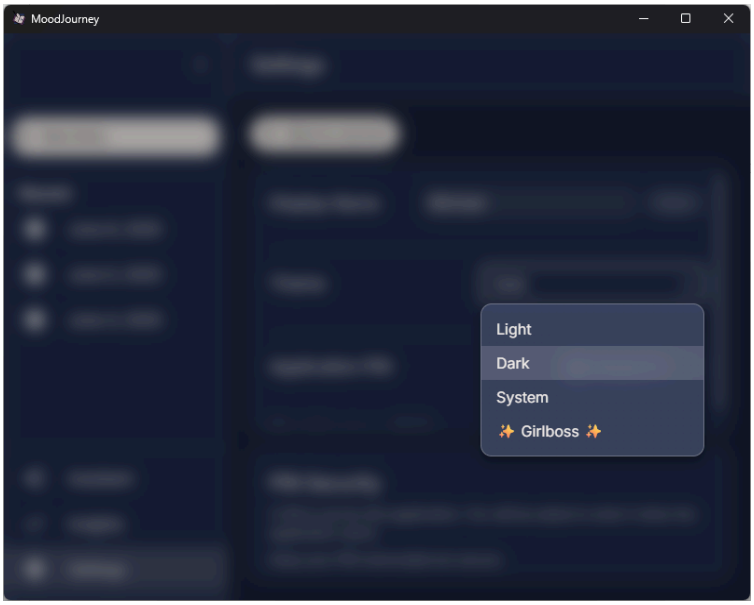
Mood Analysis/AI Suggestions: We use sentiment analysis to determine the overall mood of each Journal Entry. The user is able to view these results. The user is able to receive personalized insights and suggestions based on the identified mood trends.



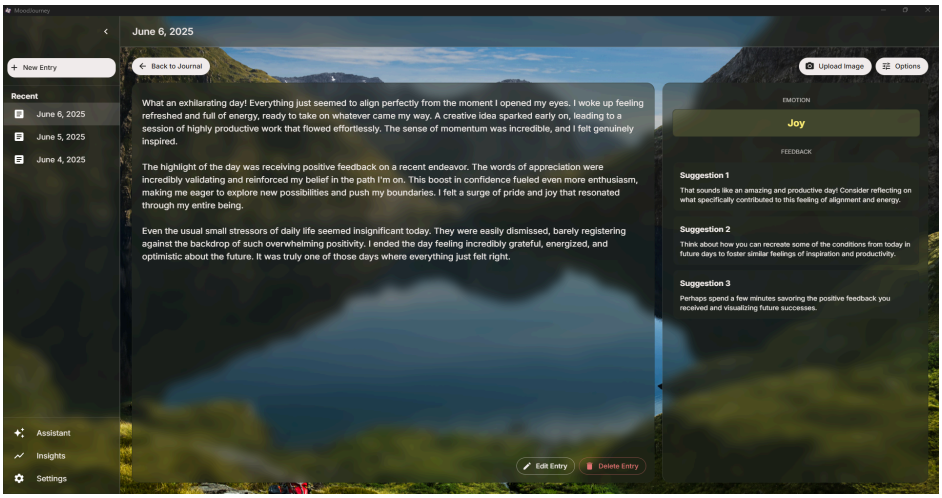


Mood Trends: The application identifies and visualizes patterns (“mood trends”) over time, based on the sentiment analysis results.

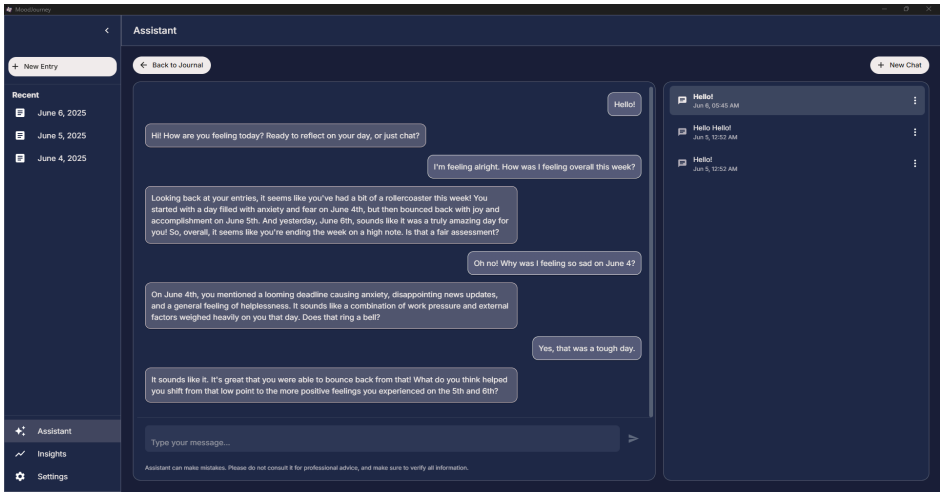
After evaluating the timeline, we decided to add extra features to make the application more well-rounded and comprehensive overall. These include the following:



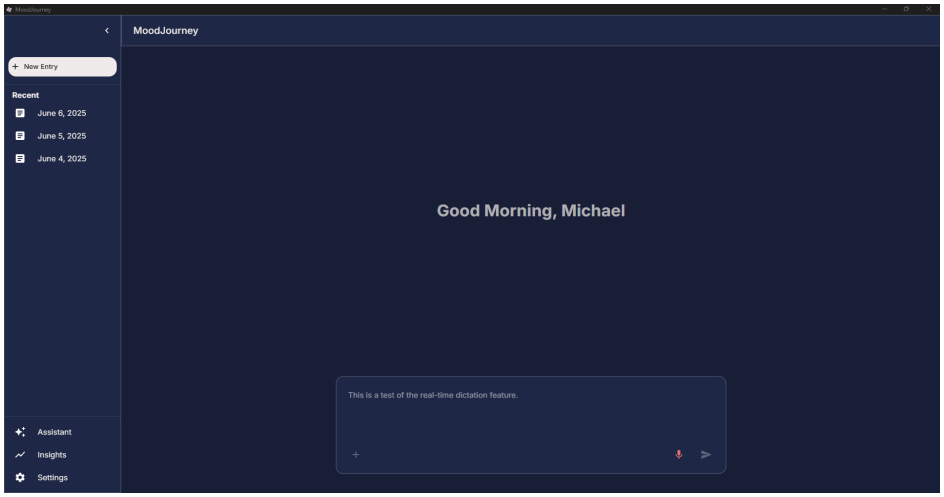
Theme Switching: The user can choose between “Light”, “Dark”, and “Girlboss” themes in order to personalize the application’s appearance. There is also a “System” theme that follows the setting of the user’s operating system.



Upload Image/Monet Theme: The user can upload an image to be associated with a Journal Entry. On Journal Entries that have images associated with them, the user can set the image to be the background of the Journal Entry. They can also activate a “Monet Theme” option that pulls a color from the background image and applies it to the page.



Assistant: The user can initiate conversations with a built-in Assistant. The Assistant intelligently pulls context from the Journal to deliver personalized responses to the user.



Voice-to-Text Transcription (Real-Time): We use the integrated Web Speech API to power a real-time dictation feature. This serves as an alternative, more intuitive version of the original WhisperAI voice-to-text feature.

III. Improvement

A. Individual Reports

1. Michael Tin
    a. Designed and implemented main features of the frontend, including sidebar (journal entries list), layout (main page/journal entry), and theming (light mode, dark mode, girlboss mode).
    b. Fully implemented voice-to-text transcription using WhisperAI and Web Speech API (frontend and backend).
    c. Fully implemented AI Suggestions for Journal Entries and Assistant feature using Gemini 2.0 Flash API (frontend and backend).
    d. Gathered team members' feedback in order to make incremental adjustments to the frontend (layout, design, etc) over the course of the project.
2. Augustine Ayoub
    a. Designed and maintained a prototype frontend throughout development in order to efficiently conduct backend testing.
    b. Fully implemented "Save Entry" feature for Journal Entries.
    c. Designed and fully integrated sentiment analysis into the application using the DistilRoBERTa-base model.
    d. Consulted with team members to help refactor backend code as necessary. when project requirements evolved. For instance, password.rs was completely refactored so that Dependency Injection could be utilized within the unit tests.
3. James Krejci
    a. Designed a robust, comprehensive color palette for the application (primary/secondary colors, light/dark mode variants).
    b. Designed the initial prototype logo for the application using Figma.
    c. Designed and implemented the frontend for the "Set Image as Background" and "Monet Theme" features.
    d. Fix various bugs in the frontend code; consult with the backend team on best ways to design/implement certain features.
4. Isaias Bernal
    a. Designed initial prototype layout for frontend (New Entry, Journal Entry, Settings, etc) as well as navigation between pages using Figma.
    b. Constructed a finalized version of the app logo (in all formats - 32×32, 128×128, etc) using the color palette of the application.
    c. Fully implemented frontend for sentiment analysis in Journal Entry (app briefly flashes green/red based on detected positive/negative emotion).
    d. Designed and implemented the Insights page (mood analysis) of the application (frontend and backend).
        i. Designed an "Informative Mode" and "Visual Mode".

5. Eddie Nguyen
    a. Fully implemented “Edit Entry” feature that allows users to edit a particular Journal Entry.
    b. Constructed the Save/Edit logic, checking whether there is already an entry for the current date before letting the user submit another Journal Entry. Fully implemented the “Delete Entry” feature that deletes a particular entry from the database.
    c. Designed a prototype pop-up dialog for the PIN Authentication feature.
    d. Consulted with team members to fix various bugs in the backend code over the course of the development process.
6. Falak Tulsi
    a. Established a working Tauri project with Rust backend, Cargo dependency management, and React/JSX frontend.
    b. Fully implemented CRUD functions (add, get, update, delete entries by date) for SQLite and oversaw database management throughout the development process.
    c. Fully designed and implemented a comprehensive suite of unit tests with code coverage across entire backend functionality.
    d. Developed a command-line interface for direct interaction with the backend.
    e. Completed the backend implementation of the PIN authentication feature.

B. Differences from Initial Expectations

- Our initial project strategy involved evenly dividing the group into a frontend team and a backend team. We developed a “sequential workflow model” where
    - The backend team would develop a core feature.
    - The frontend team would then follow up with a one-week delay in order to implement the corresponding UI/UX for that feature.
    - This model would repeat until development of the application was complete.
- However, the actual application of this model was difficult, due to its strict nature. One issue in particular was that the frontend team would need to engage with and modify backend components to resolve issues during the debugging process.
    - The PIN Authentication feature required additional functions in order to ensure proper interfacing with the frontend code.
- Additionally, in the later stages of development, we needed to do a lot of code refactoring in order to maintain consistency across the codebase.
    - The database code was modified to ensure that the database was created in %appdata%\roaming\com.moodjourney.app instead of the program directory.
- As we progressed through the development process, we made modifications to our initial approach to ensure that it would be more adaptive to our team’s needs.
- We also communicated more often to ensure that all members were up-to-date on the current progress and issues with development.

C. Time Allocation Analysis

- Most features were successfully developed, integrated, and tested within a single sprint. There was efficient collaboration between all members.
- However, there was one feature that we felt consumed too much development time and resources for little to no results. We initially attempted to implement an on-device LLM to power the AI Suggestions feature (the Suggestions that are generated for each Journal Entry). We wanted to use this approach because our dictation and sentiment analysis models ran completely on device (WhisperAI and DistilRoBERTa).
- This approach proved to be extremely challenging. There were inherent limitations to the rust-bert library, meaning that we had to use older models (i.e. GPT-2). These older models were much worse compared to newer AI models, as they were not trained to be robust and conversational.
- As a result, the original AI Suggestions generated were extremely flawed and full of random hallucinations. Running the on-device model also took up significant system resources, which hindered the development progress overall.
- Transitioning to using the Gemini 2.0 Flash API to power the AI Suggestions feature made development way easier and much more straightforward.
- We were even able to build in another feature (the Assistant feature) with the time saved from using this approach.

IV. Testing

A. Testing Methodology

- We developed a complete, thorough testing approach in order to verify the soundness and reliability of our application.
- Our testing methodology consisted of three parts: Unit Testing, Integration Testing, and System Testing.
- Testing was conducted throughout the entire development process.

B. Unit Testing

- We developed a robust and comprehensive test suite that features extensive code coverage across our backend functions.
- The test suite covers the following modules:
    - db.rs
    - emotion.rs
    - password.rs
    - suggestion.rs
- In the db.rs and password.rs unit tests, we utilized Dependency Injection to isolate the test code from external dependencies. This allowed for controlled, fast, and reliable testing in those components.

Test Name	Test Functionality
test db::tests::test_create_and_get_chat_session_from_db	Verifies new chat session can be created and retrieved
test db::tests::test_add_and_get_entries	Confirms entries can be added and retrieved
test db::tests::test_chat_session_title_truncation_in_db	Ensures long chat titles are trimmed before being saved
test db::tests::test_chat_session_title_update_on_first_user_message_in_db	Confirms that chat title is updated after first message by user
test db::tests::test_get_update_delete_entry_by_date	Confirms fetching an entry by its date, modifying it, and deleting it
test db::tests::test_save_and_get_chat_messages_from_db	Verifies that chat messages are saved and able to be fetched

test db::tests::test_delete_chat_session_and_cascade_in_db	Ensures that when session is deleted, everything corresponding to session is deleted as well
test password::tests::test_delete_pin	Tests that stored pin can be deleted
test password::tests::test_empty_password_clears_pin	Confirms that setting empty password deletes stored pin
test password::tests::test_is_locked_behavior_with_no_pin	Confirms unlocked status when no pin exists
test password::tests::test_manual_locking	Validates user can manually lock the app
test password::tests::test_set_and_check_pin	Tests setting a pin and verifying it
test suggestion::tests::test_generate_chat_response_api_key_not_configured	Ensures system handles missing API key errors
test suggestion::tests::test_generate_chat_response_empty_contents	Tests behavior when empty input is entered
test suggestion::tests::test_generate_suggestion_api_key_not_configured	Tests writing prompts for the user to self reflect on
test suggestion::tests::test_generate_suggestion_empty_prompt	Ensures empty prompt is handled cleanly and does not generate invalid responses
test emotion::tests::test_emotion_classification_basic_joy	Tests that positive input is labeled as joy
test emotion::tests::test_emotion_classification_sadness_example	Tests that sad input is labeled as sad
test emotion::tests::test_emotion_classification_anger_example	Tests that angry input is labeled as anger

test emotion::tests::test_emotion_classification_empty_input	Tests that empty input is labeled as neutral

C. Integration Testing

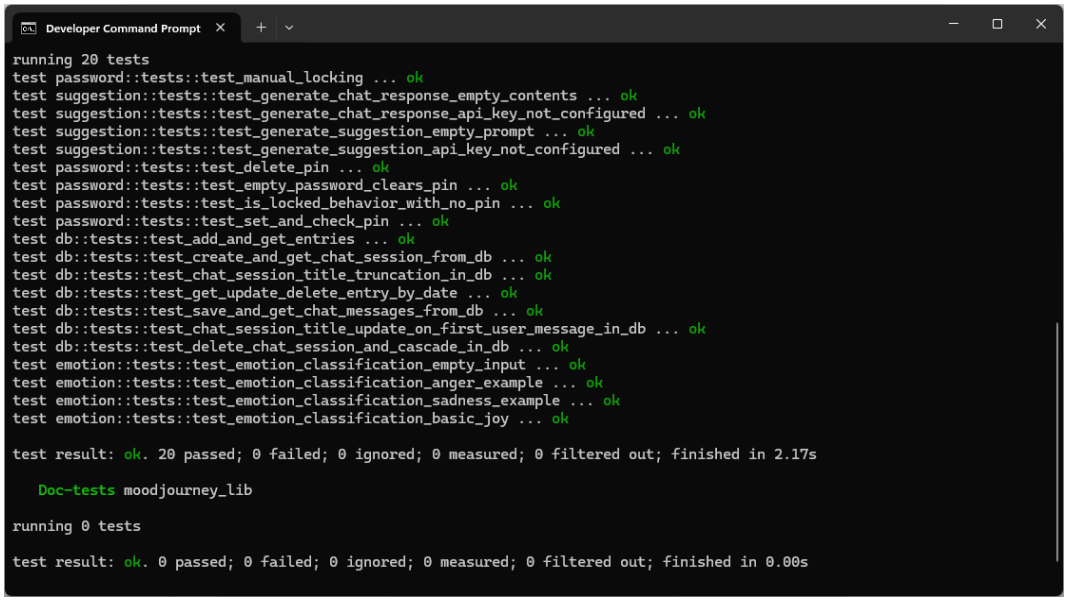
- We deployed basic integration testing in order to verify the functionality between our frontend and backend.
- Over the course of the development process, we tested the following:
    - Saving a new Journal Entry in the frontend and verifying that the Journal Entry has been saved to the database.
    - Retrieving all Journal Entries in the frontend and verifying that this corresponds to the list of Journal Entries in entries.db (via DB Browser for SQLite).
    - Editing an existing Journal Entry in the frontend and verifying that the Journal Entry has been updated in the database.
    - Deleting an existing Journal Entry in the frontend and verifying that the corresponding Journal Entry has been deleted from the database.
    - Verifying that the emotion generated from the sentiment analysis feature is correctly associated with the corresponding Journal Entry in the frontend.
    - Uploading an image in the frontend and verifying that it has been stored with the corresponding Journal Entry in the database.
    - Starting a new Assistant chat in the frontend and verifying that the chat has been stored in the database.
    - Deleting an existing Assistant chat in the frontend and verifying that the chat has been deleted from the database.
    - Enabling the PIN Authentication feature in the frontend and verifying that the backend did create the password.json file.
    - Changing the PIN in the frontend and verifying that the backend updated the password.json file.
    - Changing the PIN in the frontend and verifying that the old PIN can no longer unlock the journal.
    - Deleting the password.json file and verifying that the journal is no longer locked via a PIN.
        - Not ideal for security, but this particular interaction is more of a “proof of concept” than an actual full implementation.

D. System Testing

- Towards the end of development, we successfully utilized system testing in order to assess overall system-level functionality.
- We asked other students to be “beta testers” for the application and perform typical journaling tasks, including (but not limited to):
    - Creating, editing, and deleting Journal Entries
    - Dictating a pre-recorded audio file
    - Dictating speech in real-time
    - Uploading an image to be associated with a Journal Entry
        - Uploading an image to the same Journal Entry again; verifying that the old image was deleted
    - Toggling the “Monet Theme” option on a Journal Entry that has an associated image
    - Switching between Light Theme, Dark Theme, and Girlboss Theme
    - Enabling PIN Authentication, Changing the PIN, Disabling PIN Authentication
- This basic system testing helped us identify miscellaneous issues within the application that were not covered by the other testing methods.

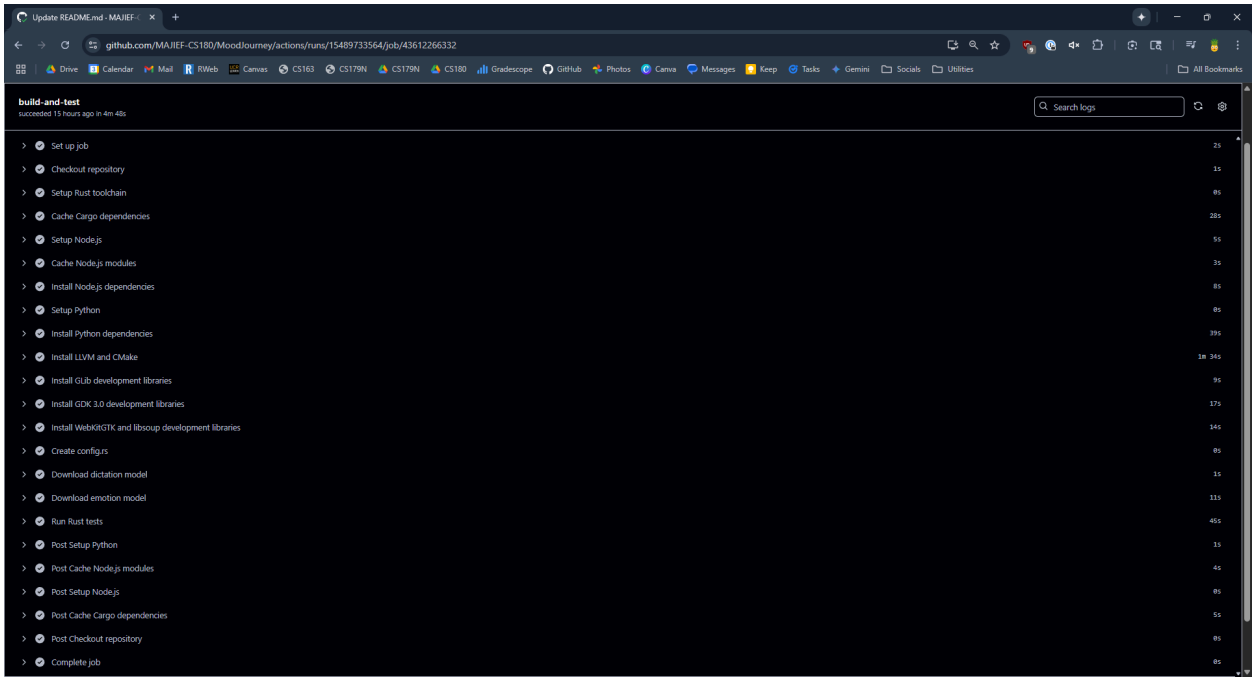
E. Running Tests

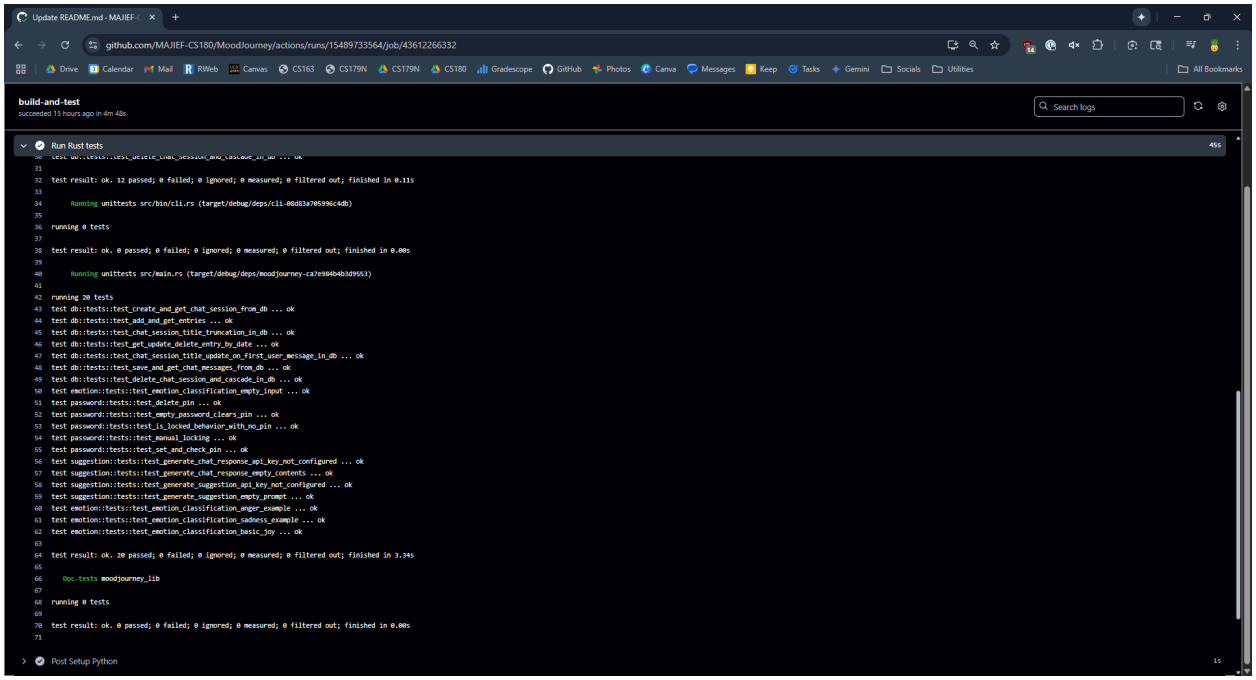
- **IMPORTANT: If you are on Windows, please use a Developer Command Prompt for VS 2022 for running the tests.**
- To run the test suite, open a Command Prompt/Terminal from the project folder directory. Then, navigate to the `src-tauri` folder.
    - `cd src-tauri`
- Run the following command.
    - `cargo test`



F. Automated Testing (GitHub Actions)

- To ensure that our testing methodology is fully comprehensive, we implemented a Continuous Integration (CI) workflow using GitHub Actions. This script automatically builds the project and runs the test suite.
- On successful completion (no errors), all tests pass within 3-4 minutes. This automation helps us identify and debug issues quickly.
- The main.yml file defines the workflow, which is triggered automatically every time code is pushed to the main branch of the repository.
- The workflow consists of a single job which executes on an Ubuntu virtual machine. It is very similar to the commands needed to setup and install the application in a real Linux environment.
- A few optimizations were included. In particular, Rust dependencies and Node.js modules are cached in order to speed up subsequent runs of the workflow.




Update README.md - MAJIEF-
github.com/MAJIEF-CS180/MoodJourney/actions/runs/15489733564/job/43612266332
build-and-test
succeeded 15 hours ago in 4m 48s
Search logs
Run Rust tests
46s
test result: ok. 12 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.11s
Running unittests src/bin/cli.rs (target/debug/deps/cli-08d83a70598c4db)
running 0 tests
test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
Running unittests src/main.rs (target/debug/deps/moodjourney-ca7e984b4b3d9553)
running 20 tests
test db::tests::test_create_and_get_chat_session_from_db ... ok
test db::tests::test_add_and_get_entries ... ok
test db::tests::test_chat_session_title_truncation_in_db ... ok
test db::tests::test_get_update_delete_entry_by_date ... ok
test db::tests::test_chat_session_title_update_on_first_user_message_in_db ... ok
test db::tests::test_save_and_get_chat_messages_from_db ... ok
test db::tests::test_delete_chat_session_and_cascade_in_db ... ok
test emotion::tests::test_emotion_classification_empty_input ... ok
test password::tests::test_delete_pin ... ok
test password::tests::test_empty_password_clears_pin ... ok
test password::tests::test_is_locked_behavior_with_no_pin ... ok
test password::tests::test_manual_locking ... ok
test password::tests::test_set_and_check_pin ... ok
test suggestion::tests::test_generate_chat_response_api_key_not_configured ... ok
test suggestion::tests::test_generate_chat_response_empty_contents ... ok
test suggestion::tests::test_generate_suggestion_api_key_not_configured ... ok
test suggestion::tests::test_generate_suggestion_empty_prompt ... ok
test emotion::tests::test_emotion_classification_anger_example ... ok
test emotion::tests::test_emotion_classification_sadness_example ... ok
test emotion::tests::test_emotion_classification_basic_joy ... ok
test result: ok. 20 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 3.34s
Doc-tests moodjourney_lib
running 0 tests
test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
Post Setup Python
1s