M3 - Requirements and Design

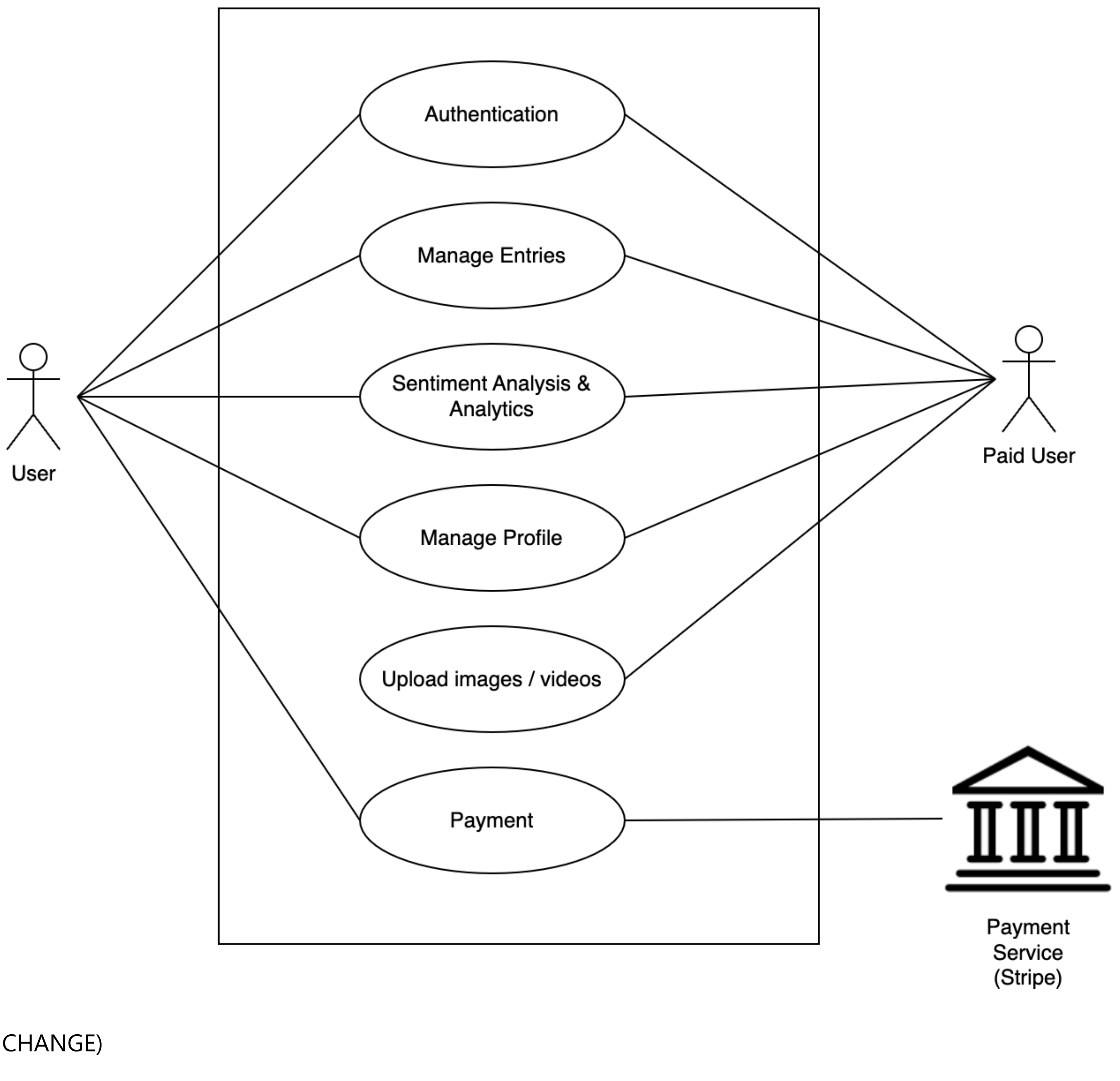
1. Change History

2. Project Description

Journal - Therapy with the Bot is an unique journaling and mental health companion application designed to help users track their moods, engage in self-reflection, and manage stress effectively. Unlike traditional journaling apps, our platform integrates an AI-powered therapy bot that provides sophisticated prompts, emotional support, and personalized feedback. By analyzing user entries and mood trends, the app encourages healthier emotional habits and enhances mental well-being.

3. Requirements Specification

3.1. Use-Case Diagram



(TO CHANGE)

3.2. Actors Description

1. **Users**: The primary actor of the application. "Users" can authenticate, manage entries, perform sentiment analysis, update their profile, and make payments.
2. **Paid Users**: It is a secondary actor and upgraded version of the "User" actor. It inherits all "User" functionalities except the payment use case and additionally allows media uploads in journal entries.

3.3. Functional Requirements

1. **Create, Delete, Edit and Export Journal Entries**

    - **Overview**:
    1. Calender View On the Front End: A Calendar with indicator to mark days that contain journal entries and days without.
    2. Click On Specific Day: If day has been journaled, 3 buttons appear Edit, Export, Delete. If day has not been journaled, Create button appears.
    - **Detailed Flow for Each Independent Scenario**:

    1. **Create**

        - **Description**: Create a Journal Entry
        - **Primary actor(s)**: User, Paid User
        - **Main success scenario**:
        1. The user clicks on the Create button for the specified day.
        2. A chatview box opens with a chatbot welcoming the user by name.
        3. The chatbot ensures a structured and meaningful entry.
        4. The user input responses are saved on the backend.
        - **Failure scenario(s)**:
        - 1a. User clicks the Create button, but nothing happens
            - 1a1. The system displays an error message & prompts the user to try again.
        - 2a. The chatbox fails to load.
            - 1a1. The system displays an error message & prompts the user to try again.
        - 3a. Chatbot fails to guide the user through structured journaling.
            - 3a1. The chatbot displays an error message.
        - 4a. Entry submission fails due to a backend/database error.
            - 4a1. The system notifies that the entry was not saved.
            - 4a2: The system provides an option to retry saving.

    2. **Edit**

        - **Description**: Edit an existing journal entry.
        - **Primary actor(s)**: User, Paid User
        - **Main success scenario**:
        1. The user selects a date that already has a journal entry and click the edit button.
        2. The saved journal entry is loaded.
        3. The user has the ability to modify the text.
        4. The updated entry is saved.
        - **Failure scenario(s)**:
        - 1a. User clicks the Edit button, but nothing happens.

        - 1a1. The system displays an error message & prompts the user to try again.
    - 2a. The entry fails to load.
        - 2a1. The system displays an error message & prompts the user to try again.
    - 4a. The Modified changes fail to save.
        - 4a1. The system prompts the user to retry.

3. **Export**

    - **Description**: Export A Journal Entry as PDF or CSV file.
    - **Primary actor(s)**: User, Paid User
    - **Main success scenario**:
    1. The user selects a date that already has a journal entry and click the export button.
    2. The chatbot asks for the preferred export format PDF or CSV.
    3. The user selects a format.
    4. The user selected file format is generated and the user downloads it.
    - **Failure scenario(s)**:
    - 1a. User clicks the Export button, but nothing happens.
        - 1a1. The system displays an error message & prompts the user to try again.
    - 3a. User selects the format, but the file is not downloaded successfully.
        - 3a1. A system message informs the user about the failed download.
        - 3a2. The system provides an option to retry.

4. **Delete**

    - **Description**: Delete a journal entry.
    - **Primary actor(s)**: User, Paid User
    - **Main success scenario**:
    1. The user selects a date with an existing journal entry.
    2. The Delete button is clicked.
    3. The system asks if a user would like to proceed?
    4. The user confirms, and the entry is permanently removed from the db.
    5. The calendar view updates to remove the journaled indicator.
    - **Failure scenario(s)**:
    - 2a. User clicks the Delete button, but nothing happens.
        - 2a1. The system displays an error message & prompts the user to try again.
    - 4a. User confirms deletion, but the journal entry is not removed from the database.
        - 4a1. The system notifies the user about the issue and suggests retrying.
    - 5a. The calendar view does not update after deletion.
        - 5a1. he system attempts a UI refresh.
        - 5a2. If unsuccessful, the user is asked to reload the page manually.

2. **Sentiment Analysis & Analytics**

    - **Overview**: Journal Entries stored on the db are analyzed the text and a sentiment score is generated.

    - **Detailed Flow for Each Independent Scenario**:

    1. **Compute Sentiment Score (Backend)**

- **Description**: Sentiment Analysis and Generate Score.
- **Primary actor(s)**: ML Model or API (depending on complexity)
- **Main success scenario**:
1. When a user creates or edits a journal entry, the system automatically analyzes the sentiment of the text.
2. The sentiment score is stored in the database alongside the journal entry.

- **Failure scenario(s)**: -1a. User submits an entry, but the sentiment analysis process fails. -1a1. The system logs the issue and retries the analysis.

- 2a. Sentiment score is generated but not stored in the database -2a1. The system attempts to resave the data -2a2. If resaving fails, an error message is displayed.

2. **Display Sentiment Trends in Analytics**

   - **Description**: Display an overview of the user’s mood over a period of time.
   - **Primary actor(s)**: User, Paid User
   - **Main success scenario**:
   1. The user clicks on the Analytics button on top of the calander.
   2. visual graphs of sentiment trends over a period of time is then displayed.
   3. User is able to chose Monthly/weekly, Pie Chart, Frequently used words in entries, etc..

   - **Failure scenario(s)**:

   - 1a. User clicks Analytics, but sentiment data retrieval fails
     - 1a1. The system retries fetching data.
     - 1a2. If unsuccessful, an error message is displayed.

   - 2a. User attempts to view a chart, but does not render
     - 2a1. The system refreshes the UI and attempts to reload the chart.

   - 3a. User does not have enough journal entries for meaningful analysis.
     - 3a1. A message informs the user that more entries are needed.
     - 3a2. The system suggests journaling more frequently.

3. **Activity Suggestions**

   - **Overview**:

     1. The system shows some suggested activities based on detected mood from Mood Tracking functionality.

   - **Detailed Flow for Each Independent Scenario**:

     1. **Activity Suggestions**:
        - **Description**: Based on the user's activity to happiness correlation, the chat bot suggests activity
        - **Primary actor(s)**: User, Paid User
        - **Main success scenario**:

            1. User clicks on the Analytics button on top of the calendar.
            2. System retrieves mood tracking data.
            3. System compose a prompt based on user's mood tracking data, and invoke an LLM
            4. Suggested activities are displayed to the user.
        - **Failure scenario(s)**:
            - 1a. System fails to retrieve mood tracking data.
                - 1a1. An error message is displayed for retrieving mood tracking data
                - 1a2. System prompts user to retry
            - 2a. Failure to invoke LLM
                - 2a1. An error message is logged for invoke LLM failure
                - 2a2. System selects a suggests some general relaxing exercise
            - 3a. Activity description fails to load
                - 3a1. An error message is displayed telling user the error, and potential solutions
                - 2a2. System prompts the user to try again.

4. **Authentication with Google**

    - **Overview**:

        1. Authenticate User: The User authenticates and creates an account

    - **Detailed Flow for Each Independent Scenario**:

        1. **Authenticate User**:
            - **Description**: The actor authenticates using their google account
            - **Primary actor(s)**: User, Paid User
            - **Main success scenario**:
                1. User presses "Sign-in" button
                2. User is prompted to type in the google account information (Email followed by password)
                3. User is authenticated
            - **Failure scenario(s)**:
                - 2a. User types in wrong information
                    - 2a1. The authentication API displays an error message telling the user of the error and potential solution.
                    - 2a2. The authentication pop up prompts the user to try again

5. **Adding and Deleting Media in the Journal Entry**

    - **Overview**:

        1. Adding media to the entry: Users can attach media files to the journal entries
        2. Removing media from the entry: Users can remove attached media files from a journal entry

    - **Detailed Flow for Each Independent Scenario**:

        1. **Adding media to the the entry**:

- **Description**: The user should be able to attach photos or videos to their journal entry from their phone storage or directly by using the devices's camera
- **Primary actor(s)**: Paid User
- **Main success scenario**:
    1. The user selects a journal entry
    2. The user clicks on the "Attachment icon" button
    3. The system prompts user to select media from device or capture a new photo using the device's camera
    4. The user selects one of the following option
    5. The media is uploaded
- **Failure scenario(s)**:
    - 4a. User selects an unsupported file format
        - 3a1. An error message is displayed: "Unsupported file format: Please select a following file format: JPG, PNG, JPEG"
        - 3a2. The user is prompted to try again with a valid file format
    - 4b. The device doesn't have enough storage
        - 3b1. The user clicks on to capture a new media
        - 3b2. An error message is displayed: "The storage is full, please free up space"
    - 5a. The media is not successfully uploaded
        - 5a1. An error is displayed: "The media was not successfully uploaded. Try again"
        - 5a2. The user is prompted to try again

2. **Removing media from the entry**:

- **Description**: The user should be able to remove previously attached media from their journal entries.
- **Primary actor(s)**: Paid User
- **Main success scenario**:
    1. The user selects a journal entry containing attached media
    2. The user selects the media they want remove
    3. System displays a pop-up: "Are you sure you want to remove this media?"
    4. The user confirms their choice
    5. The system removes the media from the journal entry
- **Failure scenario(s)**:
    - 2a. Media is already deleted or missing
        - 5a1. The system displays an error: "File not found"
        - 5a2. The user is prompted to check the presence of media in the journal entry
    - 4a. User mistakenly deleted the media and would like to recover it
        - 4a1. The system provides an Undo option for 20 seconds after it is deleted
        - 4a2. If the user does not undo, the media is removed from the journal
    - 5a. The media is not successfully deleted
        - 5a1. An error is displayed: "The media was not successfully deleted. Try again"

            - 5a2. The user is prompted to try again

6. **Upgrade Account**

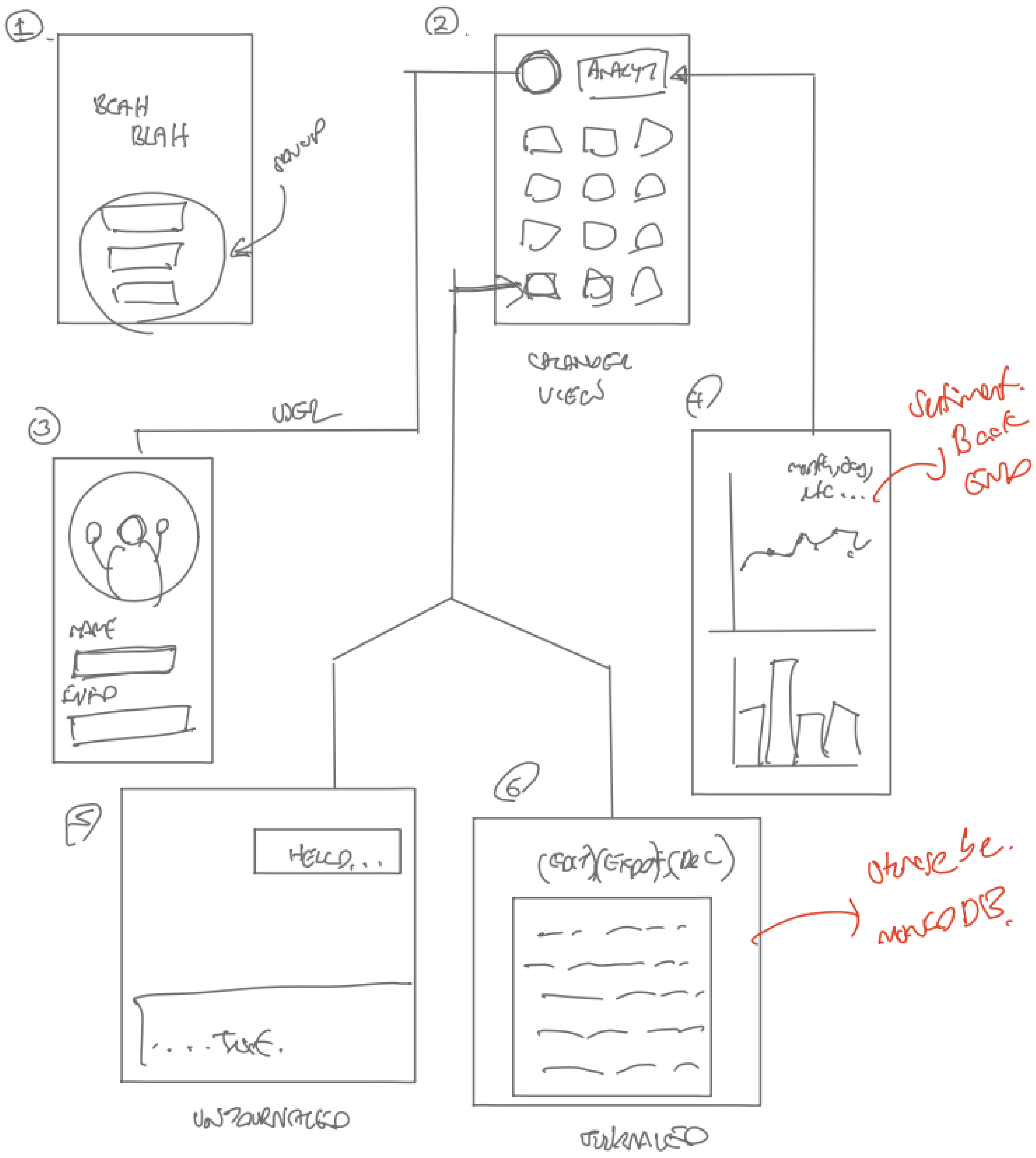
    - **Overview**:

        1. Upgrade account using third-party payment method: The user is able to upgrade their membership to get the perks. They would be a paid user after they upgrade the account.

    - **Detailed Flow for Each Independent Scenario**:

        1. **Upgrade account using third-party payment method**:
            - **Description**: User should be able to use third-party payment option to upgrade privileges of their account to enable adding photos and videos to journal entries.
            - **Primary actor(s)**: User
            - **Main success scenario**:
                1. User goes through third-party payment system
                2. The payment is approved by their financial institution
                3. The user is notified their account has its privileges elevated. It will also mention the perks that come with it and a short set of instructions.
            - **Failure scenario(s)**:
                - 2a. There is an error with the third-party payment and it returns as not completed.
                    - 2a1. The user is notified of the transaction failure and the type high-level error if that is returned.
                - 3a. The account fails to upgrade
                    - 3a1. The system prompts the user to call 123-456-7890

3.4. Screen Mockups



3.5. Non-Functional Requirements

1. **Entry Encryption**
   - **Description**: User entries will be stored in an external database encrypted with AES protocol
   - **Justification**: Encrypting user data as an extra layer of security will make more people willing to choose our application to save all of their sensitive personal information. Protecting this information is also extremely important from a privacy/ethics point of view, along with for the safety of users as well.

2. **Response Times**

- **Description**:
  - Operations for managing journal entries should be completed within 2 seconds when the server is available.
  - Sentiment analysis API should take within 20 seconds.
  - Chat bot responses should be fetched within 5 seconds.
- **Justification**: Fast response times are crucial for a smooth user experience. Waiting a maximum of 5 seconds for a prompt for writing sounds reasonable along with max 20 seconds for analysis to be done after an entry is created as it will already take some time for the user to navigate to the menu if they wanted to see progress tracking of the app. For operations involving journal entries, people will probably want something a bit more responsive since they might want to look through multiple past journal entries.

3. **Usability**

   - **Description**:
     - Operations for users managing journal entries should take less than 3 clicks.
   - **Justification**: Minimizing the number of clicks required to perform operations makes it so that the user can get to adding or viewing journal entries quickly so that people who are more impulsive for entries are more likely to remain consistent with journaling.

4. Designs Specification

4.1. Main Components

1. **User Management**
   - **Purpose**: Handles user authentication (via Google/Facebook), and profile management.
   - **Interfaces**:
     1. Get /api/profile/isPaidUser
        - **Purpose**: Checks if a user is a paid user
        - **Request Parameters**: username: an identifier of the user
        - **Response Body**: Boolean. True if the user is a paid user, otherwise False.
     2. Get /api/profile
        - **Purpose**: Gets the user's profile.
        - **Request Parameters**: username: an identifier of the user
        - **Response Body**: an object containing:
          - paidStatus: Boolean
          - reminderSetting: {Weekday: Time to remind}
     3. PUT /api/profile/reminder
        - **Purpose**: Updates the user's reminder
        - **Request Body**: username, updated reminder
        - **Response Body**: status code 200
2. **Manage Journaling (Journal Entries management)**
   - **Purpose**: Handles the management (create, edit, delete, export) of the Journal. Also handles the addition of media to the journal.
   - **Interfaces**:
     1. POST /api/journal
        - **Purpose**: Create a new journal entry
        - **Request Body**: Date, Username

            - **Response Body**: Message: a message generated by the chatbot.
        2. GET /api/journal
            - **Purpose**: Retrieve a journal entry
            - **Request Parameters**: Date, Username
            - **Response Body**: journal: the chosen journal entry
        3. PUT /api/journal
            - **Purpose**: Edit an existing journal entry
            - **Request Body**: Date, updated content, Username
            - **Response Body**: the updated content
        4. DELETE /api/journal
            - **Purpose**: Delete a journal entry
            - **Request Parameters**: Date, Username
            - **Response Body**: status code: 200
        5. POST /api/journal/media
            - **Purpose**: Adding photos and videos to the existing or new journal entries
            - **Request Body**: Date, media, Username
            - **Response Body**: The image selected by the user.
        6. DELETE /api/journal/media
            - **Purpose**: Removing photos and videos to the existing or new journal entries
            - **Request Parameters**: Media, Username
            - **Response Body**: Status code: 200.
        7. POST /api/journal/file
            - **Purpose**: Export an existing journal entry
            - **Request Body**: Date, Username, format (pdf, csv)
            - **Response Body**: filename, downloadURL.
3. **Analysis & Sentiment Tracking**
    - **Purpose**: Analyze and tracks the user's mood to get a trend over a certain period of time
    - **Interfaces**:
        1. GET /api/analytics/getTrend
            - **Purpose**: Get the user's sentiment trend over a certain period of time
            - **Request Parameters**: Username, (optional) period: ENUM (WEEK, MONTH, YEAR). get the sentiment trend within a period. Default period includes all entries.
            - **Response Body**: return the user's sentiment date within the specified period.
        2. POST /api/analytics/suggestion
            - **Purpose**: Get a suggestion based on the user's trend within a week
            - **Request Body**: username
            - **Response Body**: the chatbot's suggestion.

4.2. Databases

1. **[MongoDB]**
    - **Purpose**: Stores user journal entries along with sentiment scores.
2. **[AWS_S3]**
    - **Purpose**: Stores media files (images, videos, audio) uploaded in journal entries.

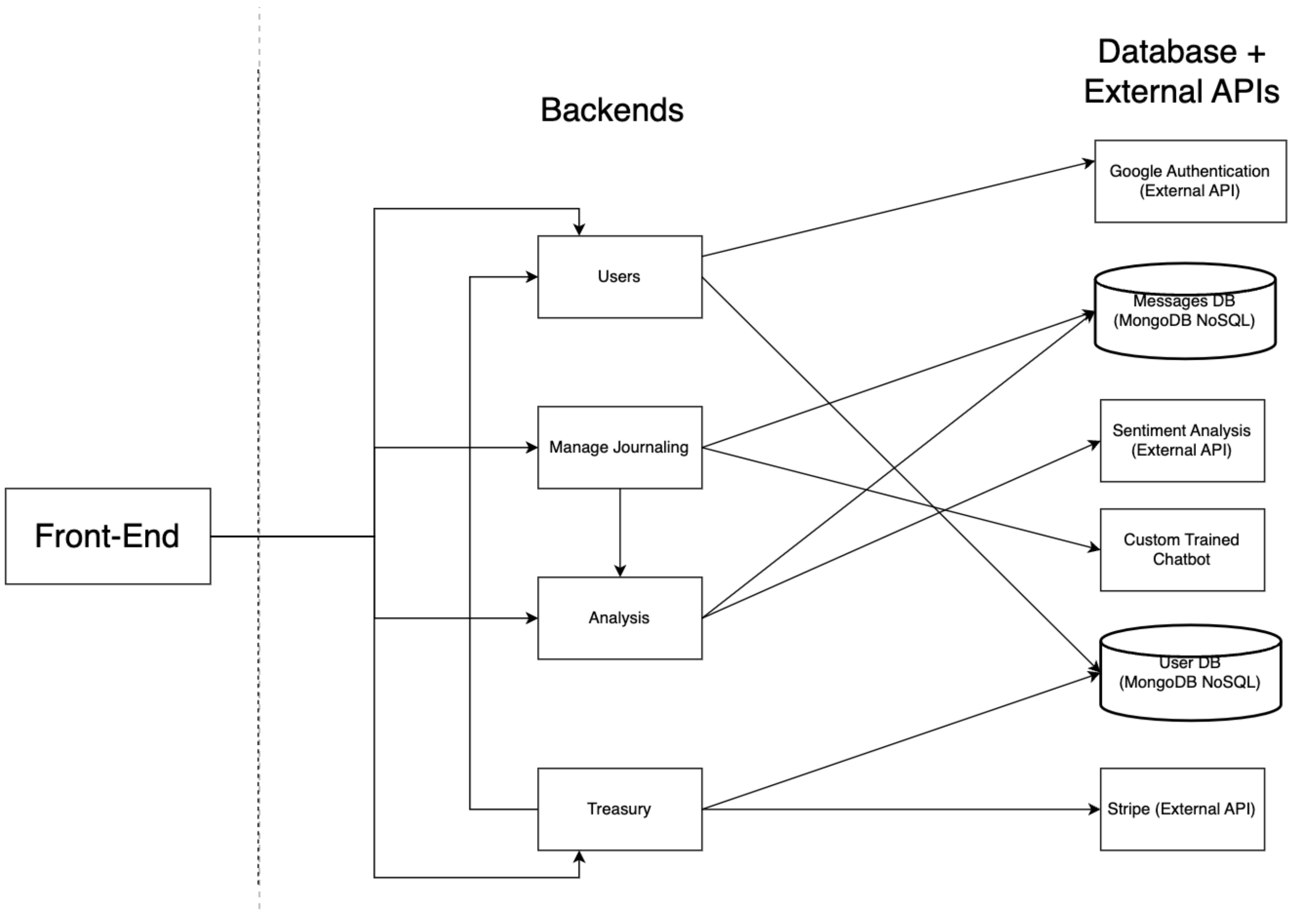
4.3. External Modules

1. **Google Authenticator**
    - **Purpose**: Handles the authentication, and registration of an user.
    - **Rationale**: There was a tutorial in google authentication. Everyone in group is familiar with it
    - **Interfaces**:
        1. GET /api/authenticate
            - **Request Parameters**: Credential (Google email and password)
            - **Response Body**: Login status
2. **Stripe - Third-party payment**
    - **Purpose**: Manage a secure payment transactions using Stripe.
    - **Rationale**: Stripe has developer-friendly APIs and is easy to learn. The alternative was Paypal which is supposedly less flexible for developers.
    - **Interfaces**:
        1. POST /api/payment
            - **Request Body**: Amount, Card
            - **Response Body**: Transaction Status
3. **Sentiment Analysis - Openai API**
    - **Purpose**: Analyzes user's journal entry to determine sentiment.
    - **Rationale**: Most of the group has used the openai API previously. It is very simple to learn as well.
    - **Interfaces**:
        1. POST /api/sentiment
            - **Request Body**: journal entry
            - **Response Body**: Sentiment score

4.4. Frameworks

1. **Docker**
    - **Purpose**: Automate the deployment of our application
    - **Reason**: Enables easy deployment across different environments without dependency issues.
2. **MongoDB**
    - **Purpose**: Store user and journaling data
    - **Reason**: We choose to use MongoDB because it is a NoSQL database, which allows flexible storage of semi-structured journal data. We also worked with it during CPEN322/CPEN320, so all group members are familiar with it.
3. **AWS EC2**
    - **Purpose**: Cloud hosting
    - **Reason**: Enables scalable and cost-effective for hosting backend services. It is also introduced in our tutorial, so all group members are familiar with it.
4. **Kubernetes**
    - **Purpose**: Container orchestration
    - **Reason**: Helps manage and scale containers efficiently across multiple nodes.
5. **RASA**
    - **Purpose**: Chatbot for guided journaling.
    - **Reason**: Enables structured journaling by dynamically adjusting responses based on user input.

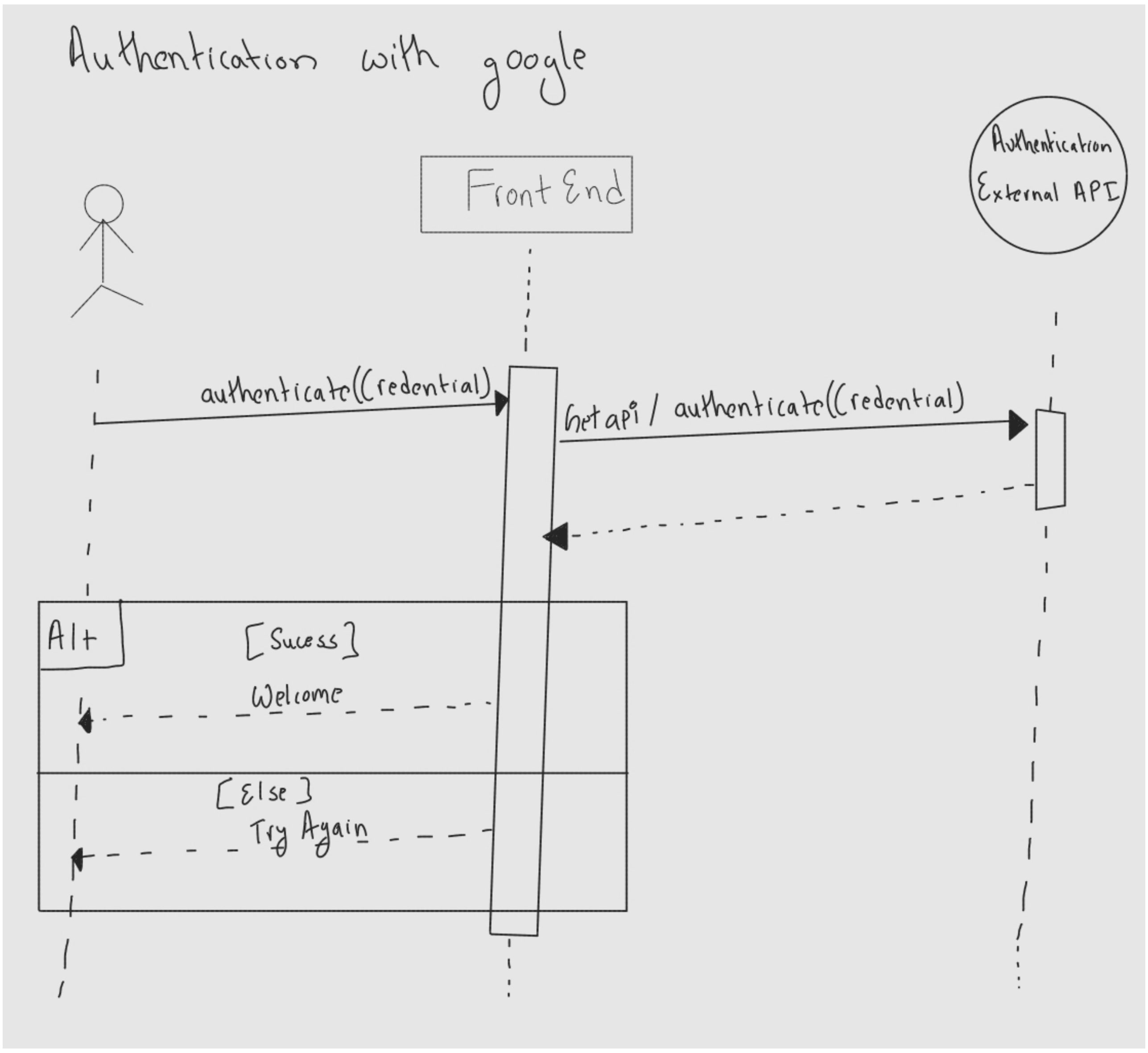
4.5. Dependencies Diagram



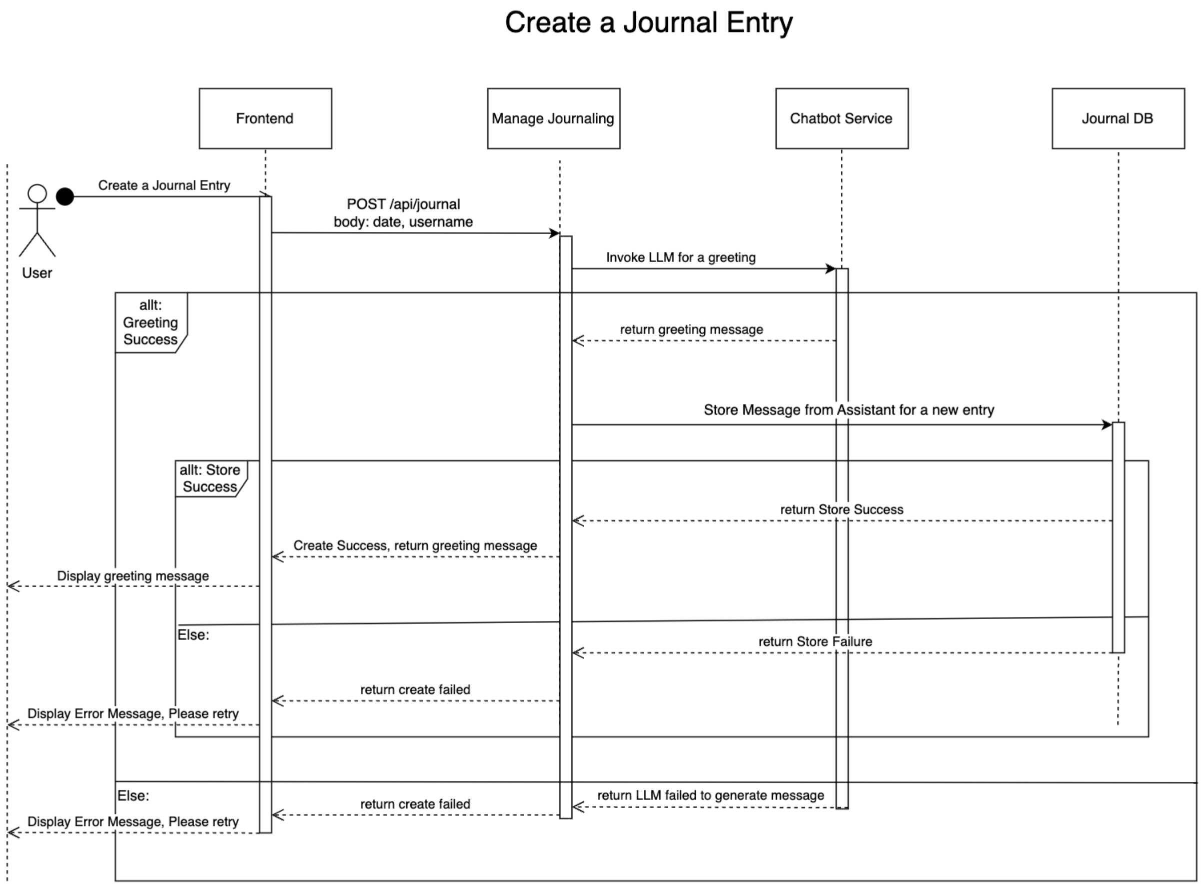
(feel free to change it in google drive)

4.6. Functional Requirements Sequence Diagram

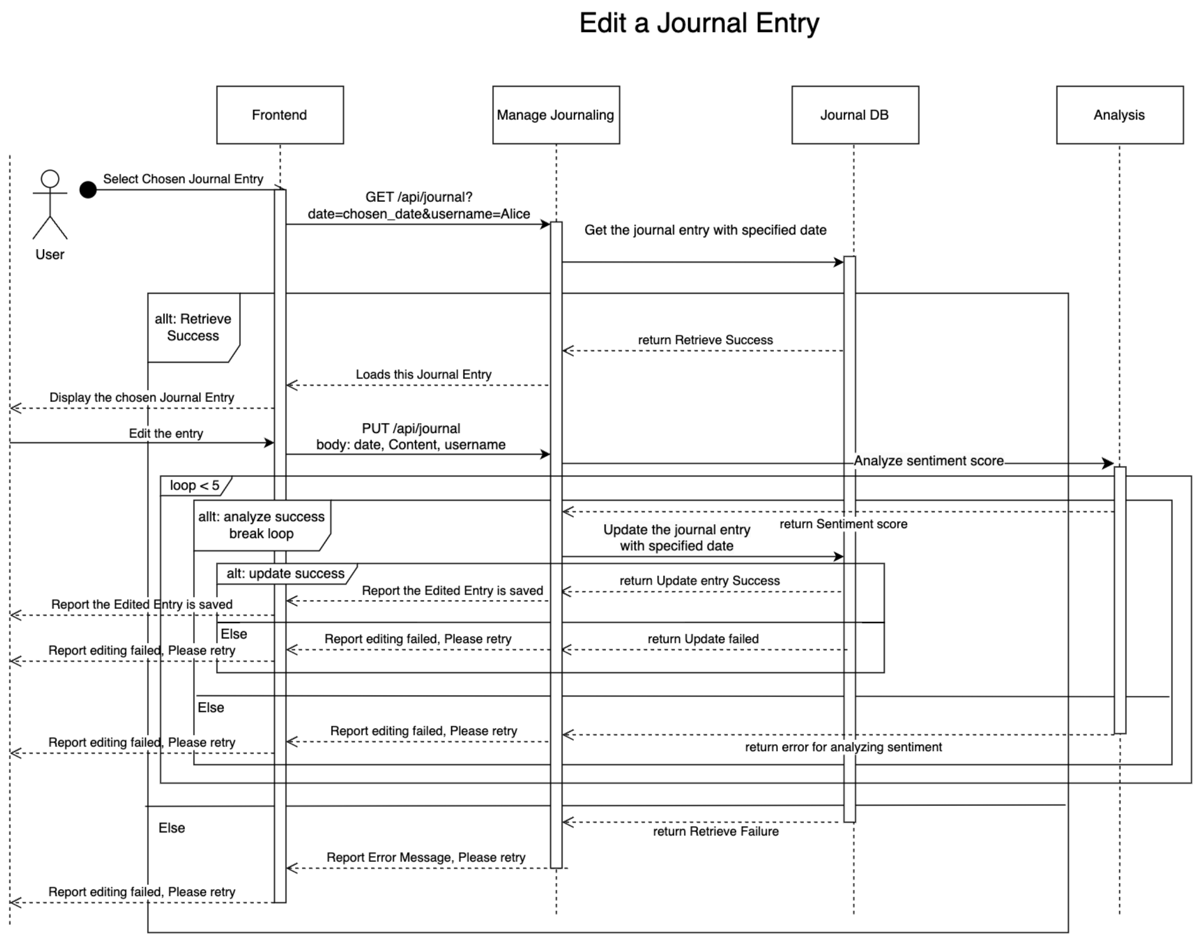
1. **[Authentication]**


Authentication with google
Front End
Authentication External API
authenticate(Credential)
Get api / authenticate(Credential)
Alt
[Sucess]
Welcome
[Else]
Try Again

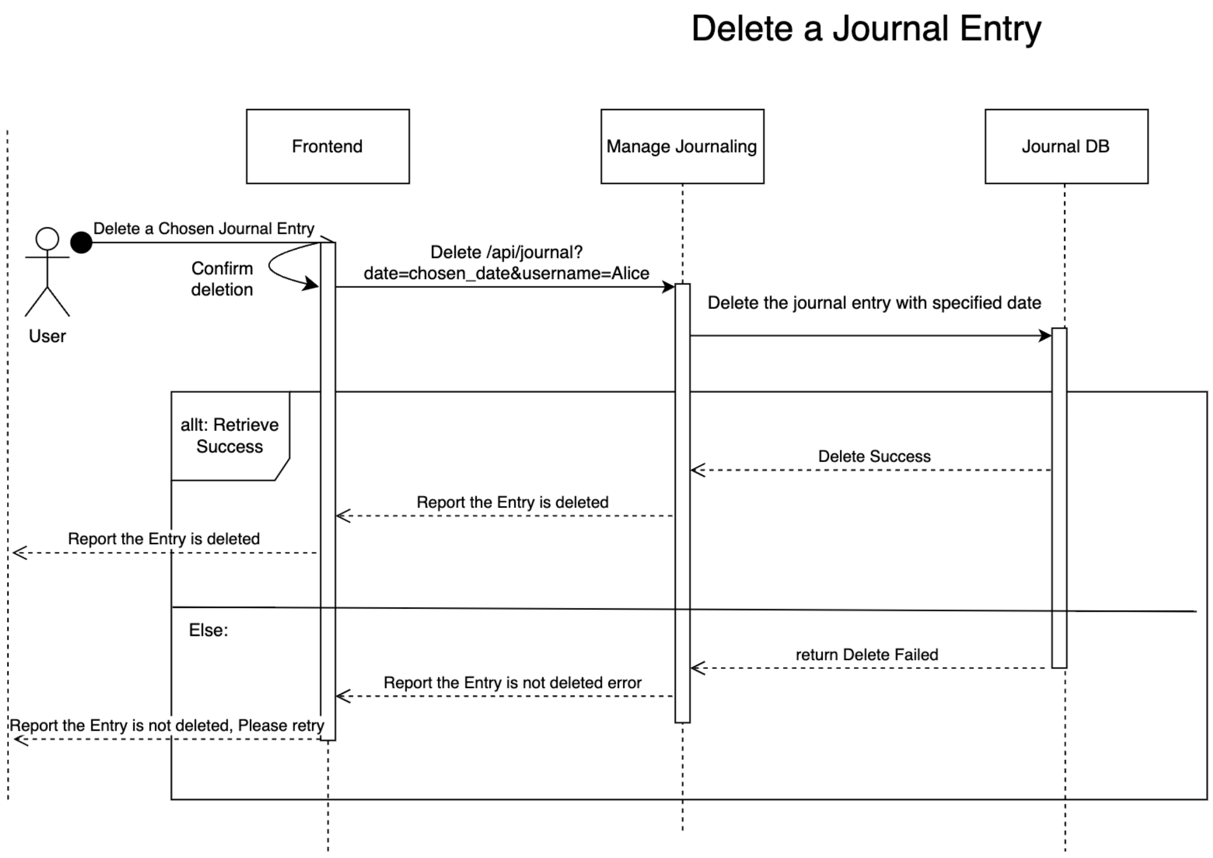
2. **[Create]**



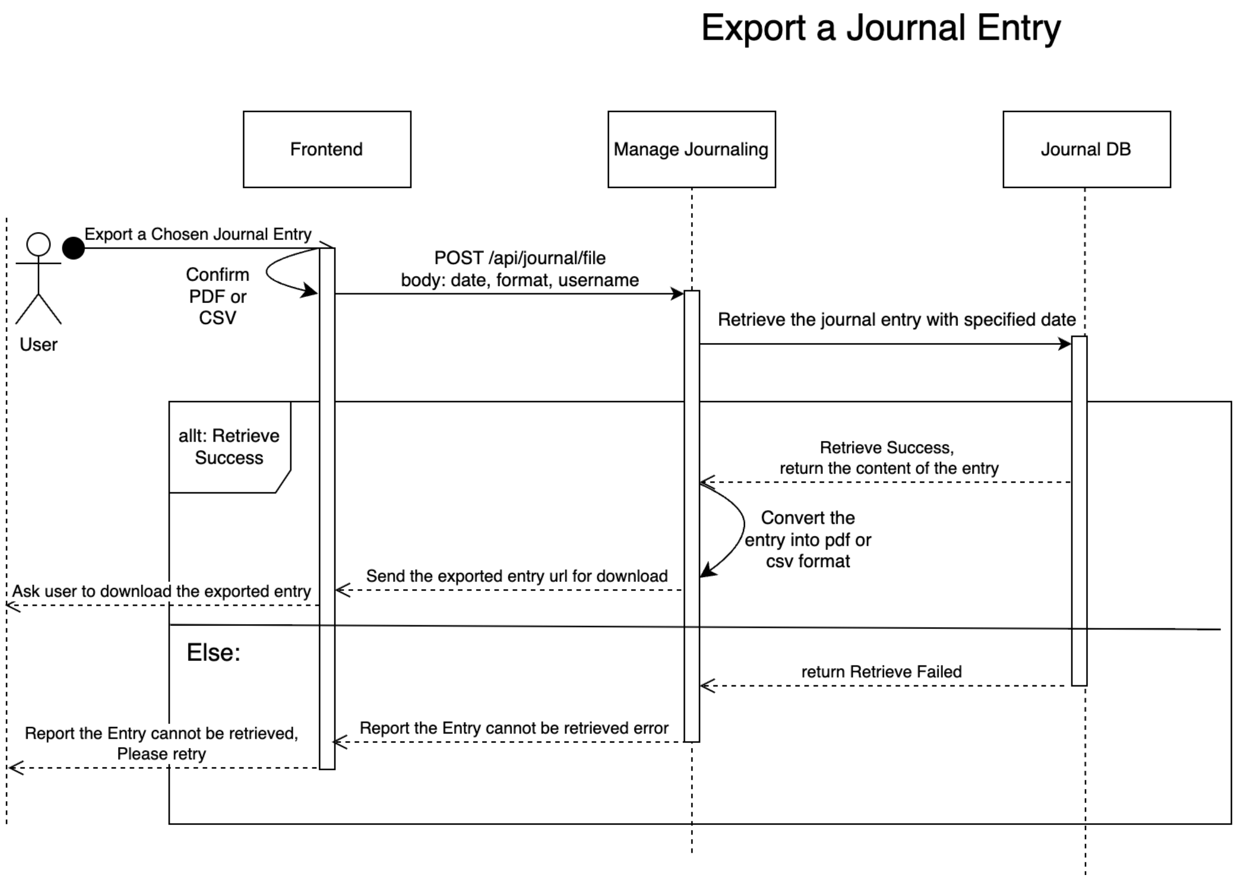
3. **[Edit]**


Edit a Journal Entry
Frontend
Manage Journaling
Journal DB
Analysis
User
Select Chosen Journal Entry
GET /api/journal?date=chosen_date&username=Alice
Get the journal entry with specified date
allt: Retrieve Success
return Retrieve Success
Loads this Journal Entry
Display the chosen Journal Entry
Edit the entry
PUT /api/journal body: date, Content, username
Analyze sentiment score
loop < 5
allt: analyze success break loop
return Sentiment score
Update the journal entry with specified date
alt: update success
return Update entry Success
Report the Edited Entry is saved
Report the Edited Entry is saved
Else
return Update failed
Report editing failed, Please retry
Report editing failed, Please retry
Else
Report editing failed, Please retry
return error for analyzing sentiment
Report editing failed, Please retry
Else
return Retrieve Failure
Report Error Message, Please retry
Report editing failed, Please retry

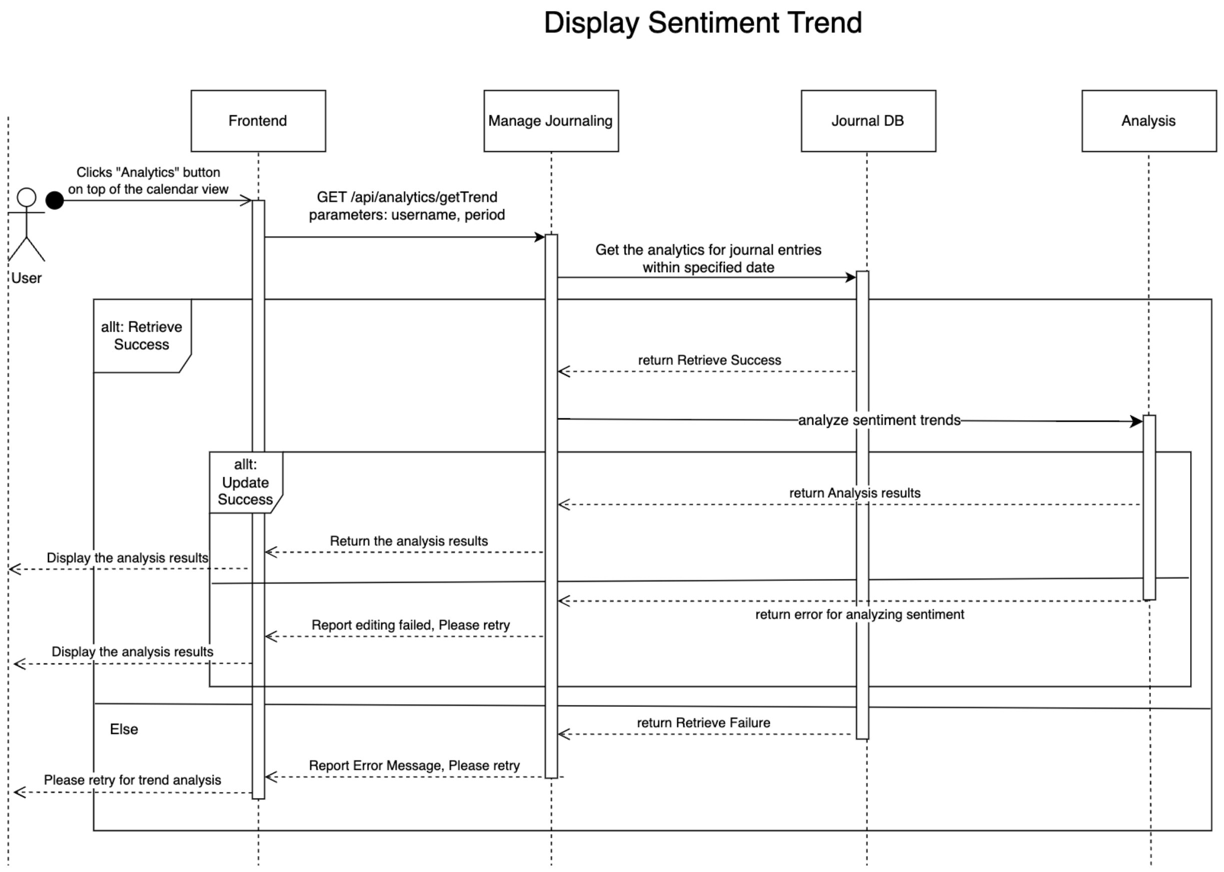
4. **[Delete]**


Delete a Journal Entry
Frontend
Manage Journaling
Journal DB
User
Delete a Chosen Journal Entry
Confirm deletion
Delete /api/journal?date=chosen_date&username=Alice
Delete the journal entry with specified date
allt: Retrieve Success
Delete Success
Report the Entry is deleted
Report the Entry is deleted
Else:
return Delete Failed
Report the Entry is not deleted error
Report the Entry is not deleted, Please retry

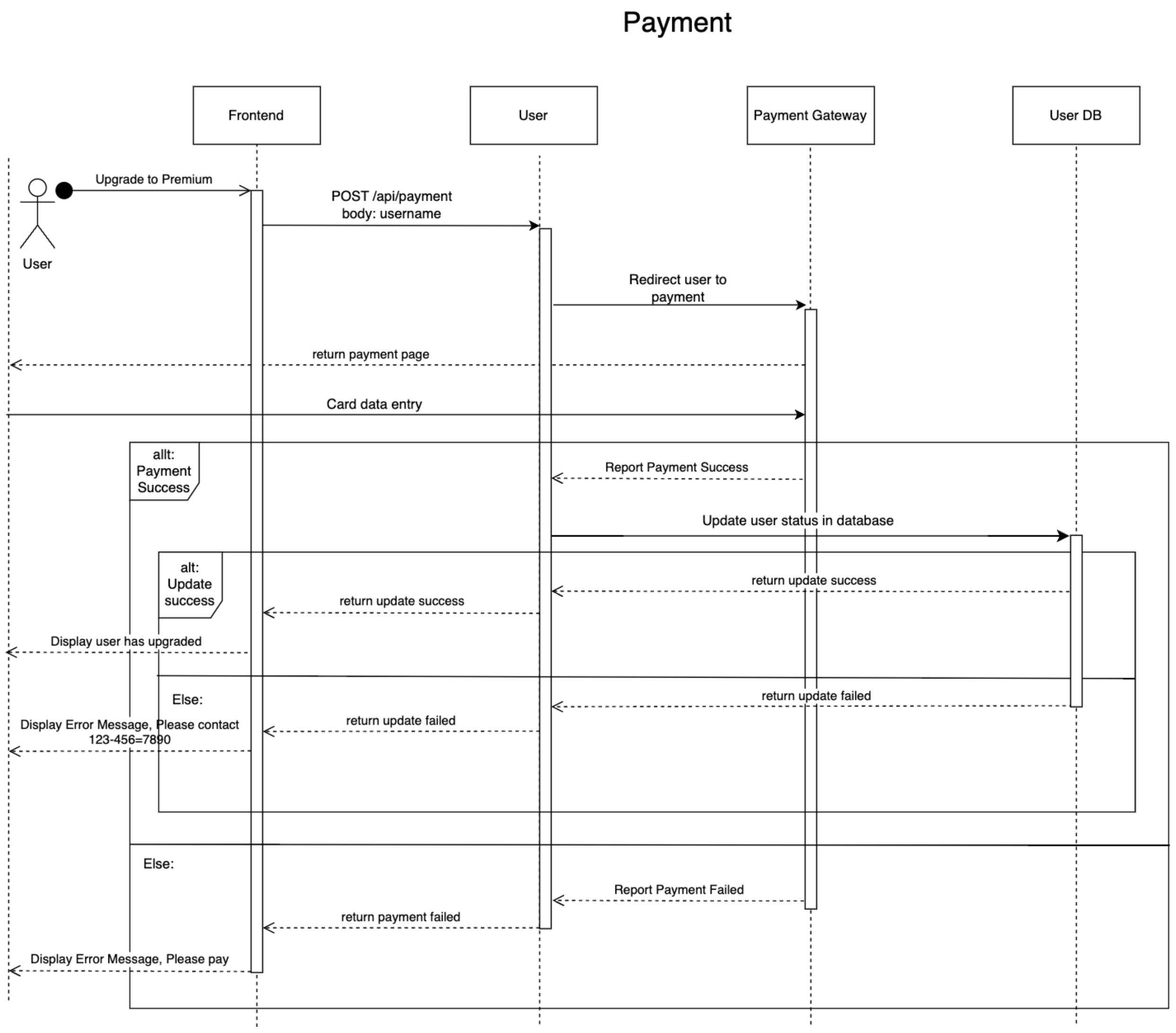
5. **[Export]**


Export a Journal Entry
Frontend
Manage Journaling
Journal DB
User
Export a Chosen Journal Entry
Confirm PDF or CSV
POST /api/journal/file body: date, format, username
Retrieve the journal entry with specified date
allt: Retrieve Success
Retrieve Success, return the content of the entry
Convert the entry into pdf or csv format
Send the exported entry url for download
Ask user to download the exported entry
Else:
return Retrieve Failed
Report the Entry cannot be retrieved error
Report the Entry cannot be retrieved, Please retry

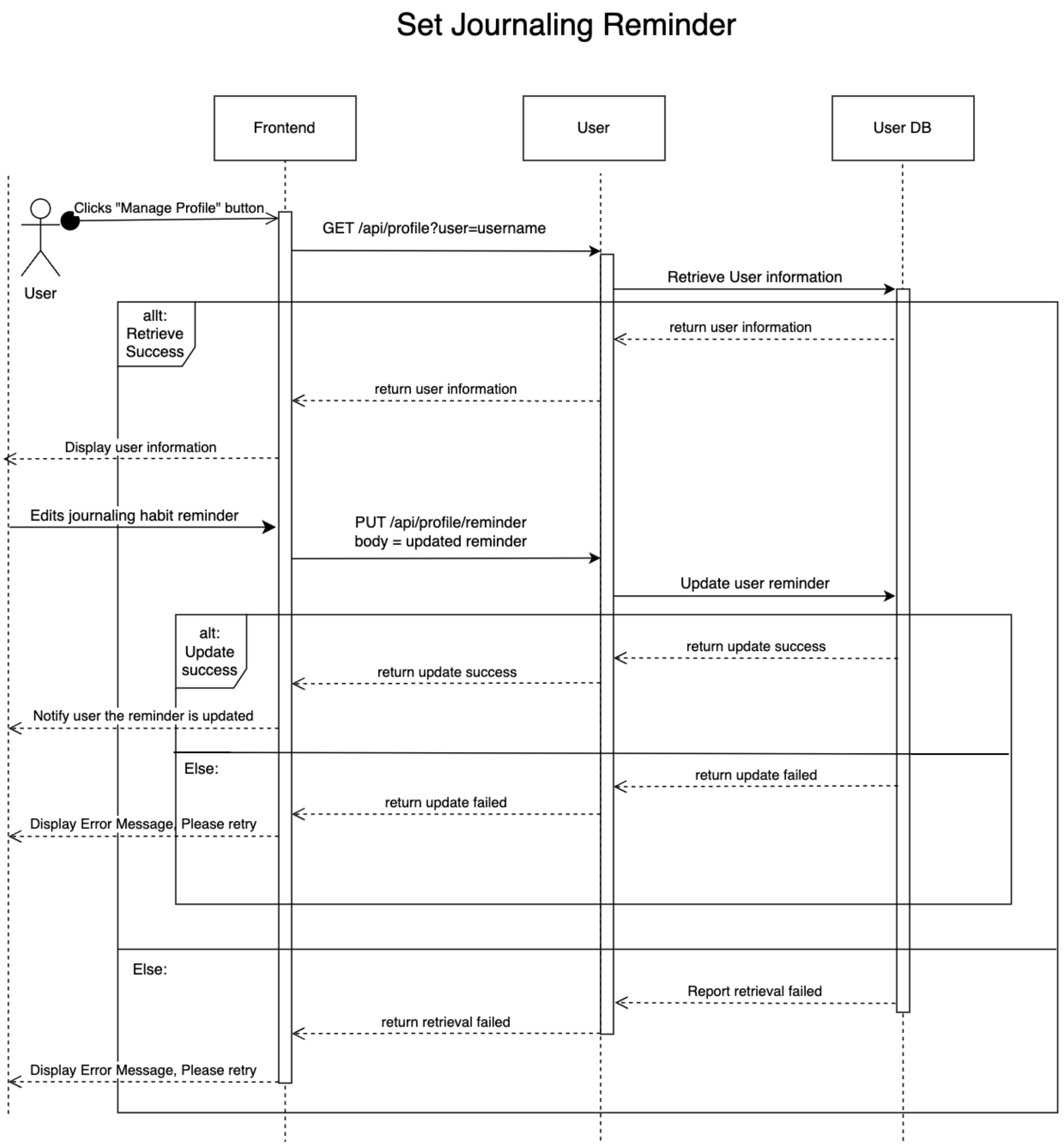
6. **[Analyze Sentiment]**


Display Sentiment Trend
Frontend
Manage Journaling
Journal DB
Analysis
User
Clicks "Analytics" button on top of the calendar view
GET /api/analytics/getTrend parameters: username, period
Get the analytics for journal entries within specified date
allt: Retrieve Success
return Retrieve Success
analyze sentiment trends
allt: Update Success
return Analysis results
Return the analysis results
Display the analysis results
return error for analyzing sentiment
Report editing failed, Please retry
Display the analysis results
Else
return Retrieve Failure
Report Error Message, Please retry
Please retry for trend analysis

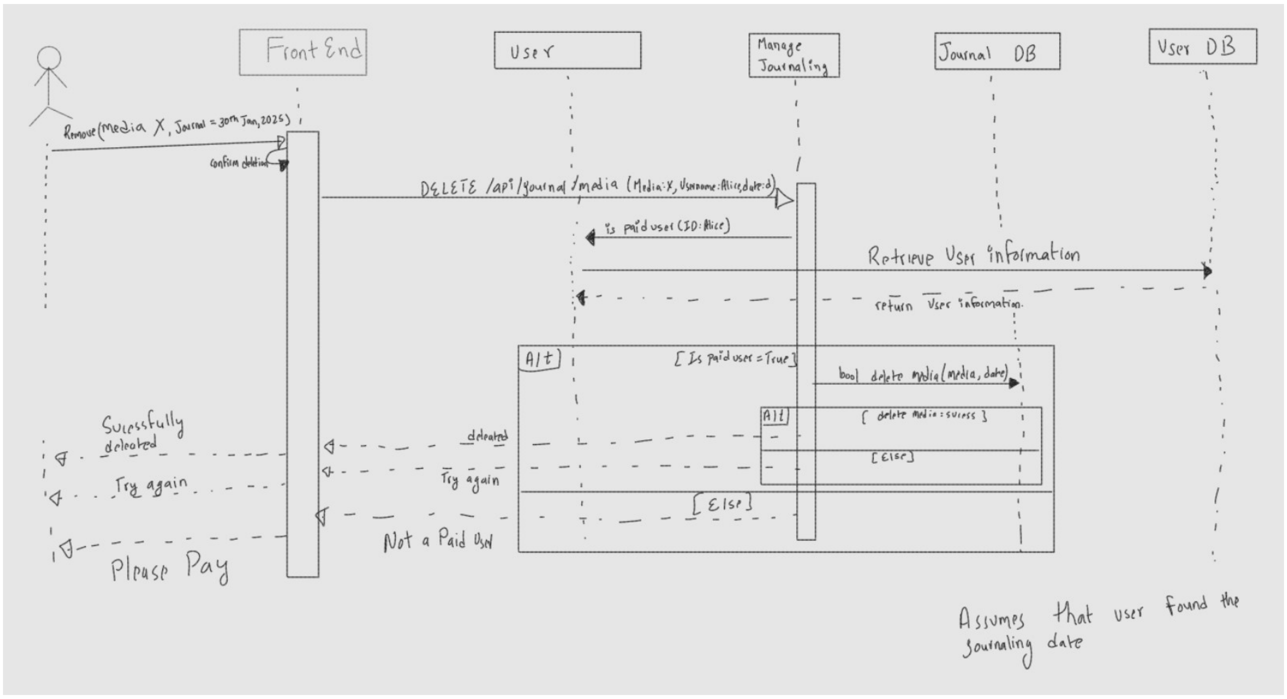
7. **[Payment]**


Payment
Frontend
User
Payment Gateway
User DB
User
Upgrade to Premium
POST /api/payment
body: username
Redirect user to payment
return payment page
Card data entry
allt: Payment Success
Report Payment Success
Update user status in database
alt: Update success
return update success
return update success
Display user has upgraded
Else:
return update failed
return update failed
Display Error Message, Please contact 123-456=7890
Else:
Report Payment Failed
return payment failed
Display Error Message, Please pay

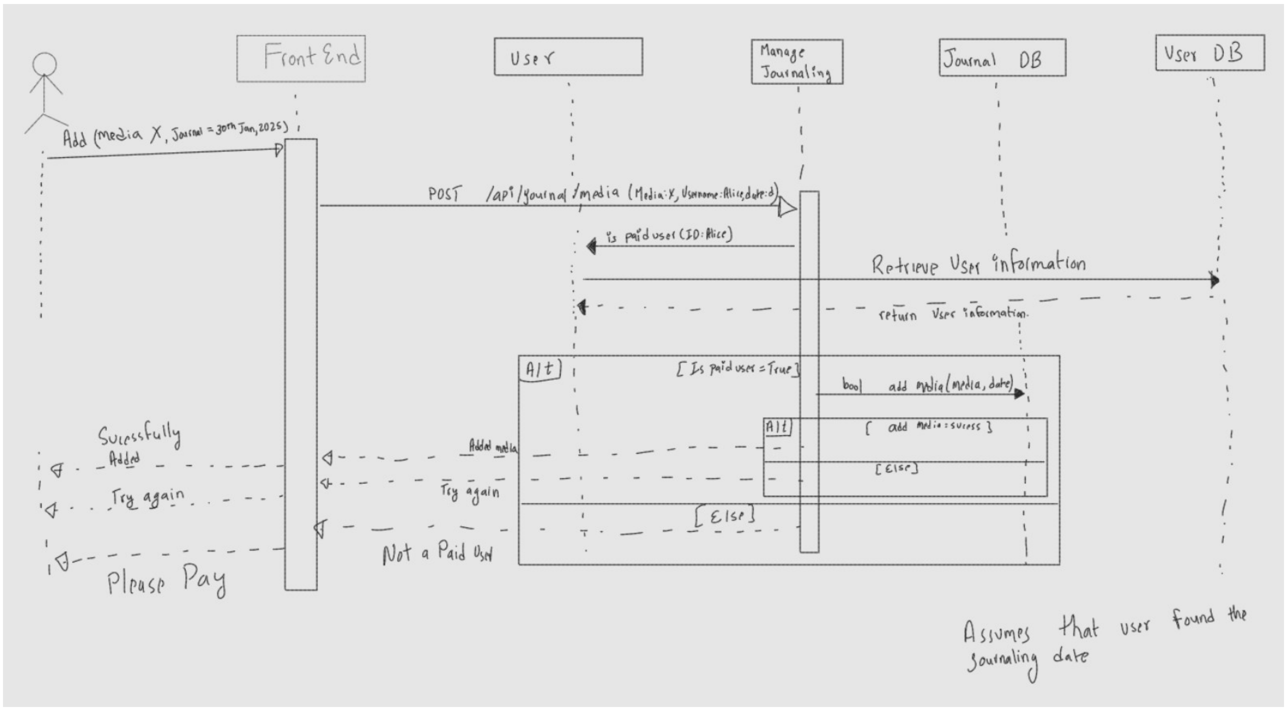
8. **[Set reminder]**



9. **[Remove Media]**



10. **[Add Media]**



4.7. Non-Functional Requirements Design

1. **Encryption of Entries**

   - **Planned Implementation**: We will use AES-256-CBC encryption protocol to ensure all entires in the database are only viewable by the user who wrote them. During registration a "salt" will be created and this will be combined with Password-Based Key Derivation Function 2 and an initialization vector to encrypt and store data. Salt will be stored in the DB with the password-based key being derived each time the user needs to modify/add entries.

2. **Performance**

- **Planned Implementation**: Using https to MongoDB, we can receive and transmit simple text, image and video (or database operations that surround these things), at a rate that will match. For sentiment analysis, because there are multiple sentiments along with multiple variables that need to be tracked, the openai API will be used to compute weights for different emotions to later display in the "sentiment summary" screen. For prompting the user, we will use the SARA framework to customize our own chatbot LLM to analyze and follow a set flow of conversation and to log activities that the user reports throughout the day for statistics tracking. Using this framework will allow us to be more flexible in that it can be run much faster than an external API call.

3. **Usability**

    - **Planned Implementation**: We will make sure the buttons design flow will be within 3 click. The user would click on the entry from calendar activity, which would take the user to the entry view activity. Then the user would do their final click of manage entry to get access to the entry. Please refer to the Screen Mockups for a detailed view.

4.8. Main Project Complexity Design

- **Description**: Based On a custom trained chatbot that is designed to guide users through structured journal entries by dynamically adjusting its questions based on user responses. It ensures that entries are meaningful, complete, and well-organized by leveraging Rasa NLU for intent classification and dialogue management. The chatbot must also integrate with sentiment analysis and the backend database to store journal entries.

- **Why complex?**: Requires custom NLP training to recognize user intent accurately. Must handle custom dialogue management, adapting dynamically to different user responses. Needs context retention, meaning it must remember user input across multiple conversation turns. Requires error handling and fallback mechanisms to manage incomplete or unclear entries. Must integrate with sentiment analysis models to analyze the emotional tone of journal entries. Requires secure and efficient storage integration with the backend for saving journal entries.

- **Design**:

    - **Input**: User’s journal entry responses (text or speech-to-text),
    - **Output**: A structured journal entry stored in the database. A sentiment score attached to the journal entry for analytics.
    - **Main computational logic**: Intent Classification: Determine user response category emotion, intent. Dialogue Management: Adjust responses and next questions dynamically in real time. Context Retention: Maintain conversational context and Memory across user interactions. Sentiment Analysis: Compute sentiment scores after post entry/completion. Backend Storage: Save structured journal entries to the database along with sentiment score.
    - **Pseudo-code**:

```
def startEntry():
    # Initialize Journaling Session
    sessionData = {
        "mood": None,
        "sleep": None,
        "exercise": None,
        "stress": None,
        "goal": None,
        "socialInteraction": None,
        "sentimentScore": None,
    }

    # Step 1: Greet The User
    chatbot.say("Hi! How are you doing today, ready for your daily 
journaling?")
    response = user_input()

    if intent(response) in ["goodbye"]:
        chatbot.say("See you tomorrow! Bye!")
        return

    # Step 2: Ask about Mood
    chatbot.ask("How is your mood today? Yesterday you weren't all that 
feeling great.")
    sessionData["mood"] = user_input()

    # Step 3: Ask about sleep
    chatbot.ask("How much sleep did you get last night?")
    sessionData["sleep"] = user_input()

    # Step 4: Ask about exercise
    chatbot.ask("Did you get in exercise for the day?")
    sessionData["exercise"] = user_input()

    # Step 5: Ask about stress levels
    chatbot.ask("Is your stress level low, medium, or high?")
    sessionData["stress"] = user_input()

    # Step 6: Ask about goals for the day
    chatbot.ask("What do you want to accomplish today?")
    sessionData["goal"] = user_input()

    # Step 7: Ask about social interactions for the day
    chatbot.ask("Did you have any meaningful social interactions with 
anyone for the day?")
    sessionData["goal"] = user_input()

    # Step 7: Provide a summary
    chatbot.say("Here's your daily journaling log:\n" +
                "- Mood: " + sessionData['mood'] + "\n" +
                "- Sleep: " + sessionData['sleep'] + " hours\n" +
                "- Exercise: " + sessionData['exercise'] + " hours\n" +
                "- Stress: " + sessionData['stress'] + "\n" +
                "- Goal: " + sessionData['goal'] + "\n" +
                "- Social Interaction: " + 
sessionData['socialInteraction'])

    #Step 8: Save To DB
    entry = {
        "userName": userName,
        "date": getDate(),
        "mood": sessionData["mood"],
        "sleep": sessionData["sleep"],
        "exercise": sessionData["exercise"],
        "stress": sessionData["stress"],
        "goal": sessionData["goal"],
        "socialInteraction": sessionData["socialInteraction"],
        "sentimentScore": sentimentScore
    }
    saveToDB(entry)
```

5. Contributions

- Nyi Nyi (3-4 hours): Complexity design, database module, functional requirement, frameworks, Screen Mockups
- Christine Jiang (10 hours): Functional requirement, sequence diagrams, component diagram, framework, main components
- Kevin Li (4-5 hours): nonfunctional requirements, nonfunctional design, functional requirement
- Amod Ghimire (5 hours): Functional requirement, sequence diagram, ector description, external modules