Project Document - Group 4: GIFeels

Roadmap of the Report

This report is structured to provide a comprehensive overview of GIFeels. It is divided into the following sections:

1. Introduction
2. Background
3. Specifications and Design
4. Implementation and Execution
5. Testing and Evaluation
6. Conclusion

Introduction

Aims and Objectives of the Project

The primary aim of the mood tracker app, GIFeels, is to provide a simple, visual and non-draining way to track a user's mood on a daily basis.

It does this by asking them to select a GIF that best reflects their current mood. It then presents them with a choice of an inspirational quote or joke to reframe their mindset, and invites them to optionally journal how they are feeling that day. Their choices are saved to a database and they are able to view a summary of each month's mood and review their entries for each date.

It is a fun, visual way for the user to practise mindfulness and decompress, especially in a fast-paced society.

The key objectives are:

- Providing a simple and user-friendly interface for mood logging.
- Offering users motivational content to improve their emotional state.
- Allowing users an optional choice to maintain a personal journal for daily reflections.
- Provide users with an overview page to track their moods over time.
- Ensuring data privacy and security for all user inputs.

Background

GIFeels is designed to help users manage their emotional wellbeing.
Rules and requirements:

- Guest users can access the homepage, select a mood and access the joke/quote but cannot save an entry. This allows us to demo the features to attract users.

- Users must be logged in or register an account to save an entry and access the overview page
- To register a new account, the user must submit:
    - A first name of at least 1 character / max 25
    - A last name of at least 1 character / max 25
    - A unique username of at least 4 characters / max 25
    - A unique email in the correct format (including @) of at least 6 characters, max 35
    - A password of at least 4 characters / max 25
    - Repeated password
    - Agree for GIFeels to store their personal data
- Users can only submit one emotion choice and selected joke or quote a day. This is because the overview page will only let them view one entry per day.
    - They must select and save a joke or a quote, and cannot save both (although they can view both)
- Users submit their emotion choice, joke/quote separately to their journal entry - this is to reflect that journaling can be emotionally taxing so users may not wish to do it everyday.
    - Users can log out and log back in between submitting their emotion and journal.
    - Journal entries are up to 500 characters in length
- Users must submit their emotion choice/joke/quote before submitting a journal entry - i.e. you cannot submit a journal entry without having already saved your emotion for that day.
    - This is because the main purpose of the website is to track emotions.
- Once logged in, the user session will last for 15 minutes before expiring. This is to ensure the giphy API is only called once per session (otherwise it would refresh every time the homepage is refreshed) to prevent the rate limit being reached.

Specifications and Design

Functional vs Non-functional Requirements

Functional Requirements:

- User registration and login functionality
- Mood selection and logging
- Display of motivational quotes or jokes
- Logging of selected quotes or jokes
- Optional journal entry submission
- Viewing past entries in a calendar format
- Viewing a bar chart of mood entries that updates for the month reflected
- Navigation bar with a dynamic login/logout option that updates based on whether the user is logged in or out

Non-functional Requirements:

- **Security:** Data security and user privacy; session management and timeout handling; password security using salting
- **Usability:** Responsive and user-friendly interface

Design and Architecture

The application is built using Flask, with a modular structure that separates different functionalities. The key components include:

- **User Authentication:** Managed through registration using flask-wtf for validation, login using bcrypt for password hashing and security, and Flask sessions
- **Mood Logging:** Allows users to select their mood and receive corresponding content
- **Content Fetching:** Integrates with external APIs (GiphyAPI, QuoteAPI and JokeAPI) to fetch gifs associated with a specified mood and motivational content
- **Journal Management:** Enables users to write and save daily journal entries
- **Calendar Overview:** Displays a calendar where users can select a date to view their entries for that date and a graph displaying monthly mood summaries that dynamically updates with the calendar month displayed

User flow and workflow

The bullet points below show the logic flow for the app:

If the user is logged in:

- On logging in, the users name, user ID and the current date are saved in sessions
- On the homepage, the user is presented with various GIFs reflecting 6 moods.
- The user clicks on a GIF based on their mood.
- The user has the option to choose either a joke or an inspirational quote.
    - They receive the joke or quote through the APIs.
- The user may save the joke or quote in the database. They must be presented with the button to ‘go to journal’ to go to the journal page.
- The user is directed to a journal page where they can input their thoughts and reflections. They must have the option to save it.
    - These thoughts and reflections are saved in the database.
- The final page is an overview with a graph and calendar.
    - The user can change the month on the calendar.
        - The graph will dynamically update with their moods for that month.
    - The user can use the calendar to click on a date.
        - Clicking on a date will redirect to a page displaying what their chosen mood, gif, joke/quote, and if submitted, journal for that day.
        - If they do not have an entry saved for that date, it will redirect back to the overview page and display a Flash message saying as such.

If the user is not logged in:

- The user can still access the homepage, click on a gif, view a joke and/or a quote, and view the journal page
- If they try to save their emotion/joke/quote and/or journal entry, they will be redirected to the login page
- If they try and access the overview page, they will be redirected to the login page
- The login page has a link to the “Register New User” page

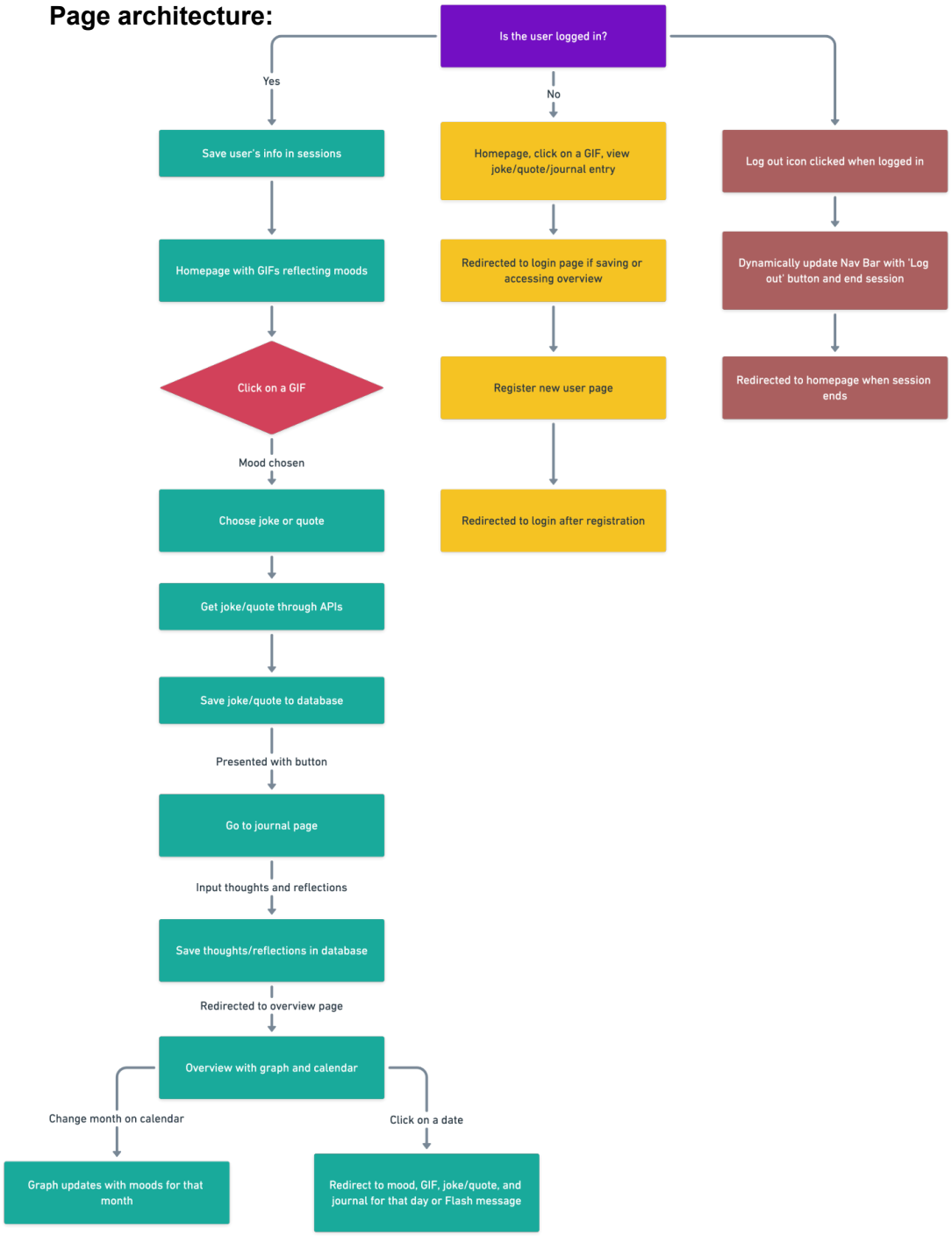
If the user registers a new account:

- They will be redirected to the login page and invited to login

If the user logs out by clicking on the icon on the nav bar:

- When logged in, the Nav Bar dynamically updates with a ‘Log out’ button
- If selected, their session will be ended and they will be redirected to the homepage

Page architecture:



Data authentication

The app must be able to collect user data, their email and password. That data must be stored securely in a database. After the user has signed up, they will be asked to login and the inputted credentials must successfully be validated on the backend to check if that user exists. If they do exist, the user will be logged in successfully and their session begins.

Data validity

1. User Authentication and Authorisation

- **Registration Data:** Validating that the data entered during user registration is accurate and meets predefined criteria. This includes verifying that the email format is correct, the email is unique, passwords match and are of a minimum and maximum length, and that usernames are unique

- **Login Data:** Ensuring that the credentials provided during login match the stored credentials in the database. This includes validating the username and password combination.

2. Input Validation

- **Mood Selection:** Ensuring that the mood selected by the user is from a predefined list of valid moods. This prevents invalid or unexpected mood entries.

3. Session Management

- **Session Integrity:** Ensuring that sessions are managed securely and expire appropriately after a period of inactivity to prevent unauthorised access.

- **Session Data:** Ensuring that session data such as user ID, selected mood, and date are correctly stored and retrieved during the session lifecycle.

4. Data Consistency

- **Database Integrity:** Ensuring that data relationships (e.g., user IDs associated with mood entries and journal entries) are maintained accurately. This involves using primary and foreign keys appropriately.

5. API Data Validation

- **External API Responses:** Validating the data received from external APIs (e.g., QuoteAPI, JokeAPI) to ensure it meets the expected format and content criteria before displaying it to the user or storing it in the database. If the API data is not correct, supplying default options to prevent the app from crashing.

6. Error Handling and Feedback

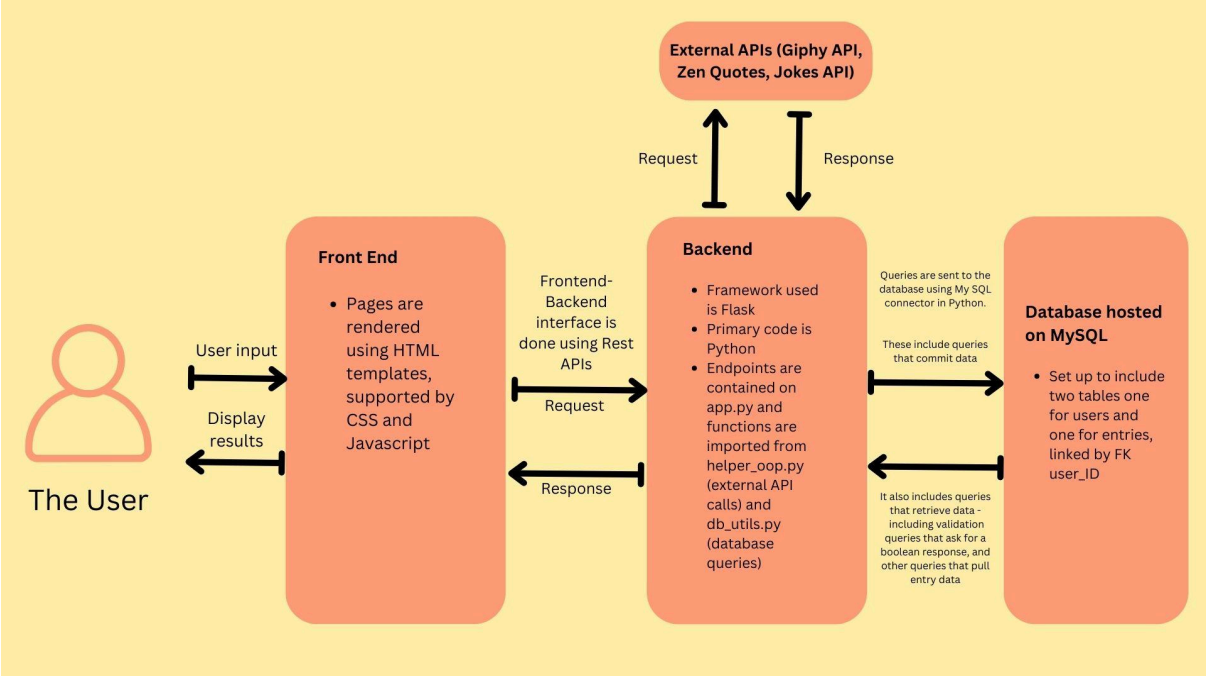
- **User Feedback:** Providing clear and actionable feedback to users when they enter invalid data. This includes using flash messages to inform users of errors such as invalid login attempts, duplicate registrations, or empty journal submissions.

- **System Errors:** Ensuring that system errors are handled gracefully and do not result in data corruption or loss. This includes catching exceptions and logging errors for further investigation.

7. Security Measures

- **Data Encryption:** Ensuring that sensitive data such as passwords are encrypted both in transit and at rest. This includes using bcrypt for password hashing. Using a secret and unique for using Flask sessions.

System architecture:



Implementation and Execution

Development Approach and Team Member Roles

The development approach was iterative, following Agile principles. Frontend and Backend roles were split to reflect team member's strengths and interests.

Key roles are laid out in a table format below:

BACKEND	
Developer	**Role / Tasks**
Rachel	- Set-up and integrate links between the front-end and back-end, using form submissions and AJAX calls. - Backend coding and validation of new registration and log-in pages. - Use of session, flash messages, and user credentials. - Class based coding for the API calls. - Testing for Flask endpoints. - Supported on database queries. - Exception handling. - Helped organise the team into sprints.
Fabiola	- Set up and integrate frontend code with jinja templates in flask. - Creating queries to save and retrieve data from the database. - Class based coding on the database. - Testing database functions. - Use of session. - Use of bcrypt for salting and hashing for user passwords. - Code to create and populate database with python
Hannah	- API call for joke and quote - Exception handling - Database design and set-up - Use of bcrypt for salting and hashing for user passwords. - Setting up defaults for API calls - ReadMe file - Creating mock data
FRONTEND	
Alyssa	- Sign-up page - Log-in page - Homepage for choosing GIFs - Overview page
Laura	- Using Figma to design specific pages - Choice page - Quote page - Joke page - Journal entry page - Navigation and Footer; logo.

Agile Development

The team adopted Agile practices by implementing the following:

- Agile methodology including three short sprints
- Deadlines are set to complete meaningful and chunks of usable code

- Meetings are set weekly after every sprint for review and action points to be made

Agile development was selected to ensure a minimal viable product was created in the first sprint and allow the team to revise stretch targets at each sprint retrospective. It also ensured robust testing took place throughout.

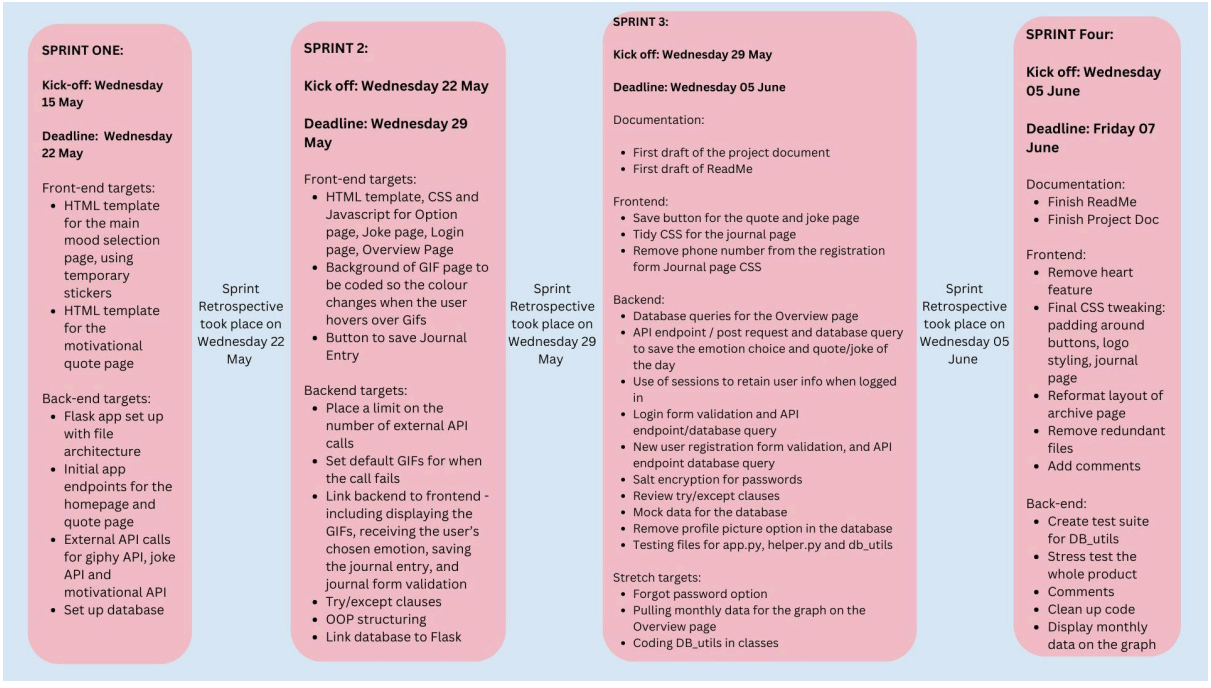
The group chat was also utilised for consistent updates and progress for the developers' respective tasks, and this also encouraged code reviews in between.

Implementation Process

The implementation process included:

- **Setting Up Flask Application:** Configured Flask environment and created initial project structure
- **User Authentication:** Implemented registration, login, and session management
- **Mood Logging:** Developed functionality for mood selection and content fetching
- **Journal Management:** Added features for writing and saving journal entries
- **Calendar Overview:** Implemented a calendar view to display past entries
- **Styling and UI Enhancements:** Applied CSS for a responsive and user-friendly interface

Overview of Sprints:



Code reviews:

Code reviews were implemented in the following ways:

- Weekly calls were held on Sundays where the team took part in pair coding, and shared and reviewed code before merging
- The team reviewed branches for specific features and often worked on the same branch before merging
- Pull requests with other team members added as reviewers were used when the above did not take place
- Weekly sprint retrospectives were held on Wednesday where the team would review and discuss code
- Unscheduled huddles between team members took part on an ad hoc basis.

Tools and Libraries

Tools:			
● Javascript	Used to make html templates interactive	● MySQL	Used to host database
● Python	Used to code the backend of the Flask web app GifFeels	● Ajax	Used to dynamically update the overview page without reloading
● Jinja2	Used to modify HTML templates (inheritance, handling data passed from the backend)	● APIs	External APIs - Giphy API, Zen Quotes, I Can Hadz Dad Jokes - used to get content

Libraries:

● Bootstrap	Used to find HTML/CSS/Javascript templates	● Requests	HTTP client library used to make external API calls
● Unittest	Used to write testing for database functions and external API calls	● Random	Used to make random selections of API response data so the Gifs, Quotes and Jokes are varied
● Flask	Used to provide web framework for the application	● bcrypt	Provides password hashing functions

• Sql connector	Used to allow MySQL queries to be made	• ChartJS	Javascript library used to generate bar graph on Overview page
• Flask-testing	Custom library for testing flask endpoints	• Flask-wtf	Library that allows the integration of WTF forms into Flask
• abc	Library to provide infrastructure to define an abstract base class. Used in the API call class.	• datetime	Used to create and interpret user input as dates.

Implementation Achievements:

- Work of Frontend team allowed for a visually engaging and user friendly platform
- Salting for passwords provided enhanced security
- Use of AJAX calls allowed for Overview page graph to be dynamically updated without page reload

Implementation Challenges:

Some challenges faced during implementation included:

- Integrating external APIs and handling potential errors.
- Designing a responsive and intuitive user interface.
- Front-end and back-end were initially developed separately, which meant that both needed adjustments when it came time to link them together - causing some work to be duplicated.

Implementation Adjustments:

- Some stretch targets dropped due to lack of time - i.e. forgot password feature
- Overview page was originally intended to display entries on selected days on the page itself without need for reload - this was changed to redirect to a separate page
- Team looked into doing the Flask app in OOP style using method views - but this proved too ambitious for the time provided

Testing and Evaluation

Testing Strategy

The testing strategy included:

- **Unit Testing:** Tested individual functions and components. For the API endpoints, we utilised the Flask-testing library as opposed to the unittest library as it had custom made test assertions.

Functional and User Testing

- **Functional Testing:** Verified that all features worked as expected, including mood logging, content fetching, journal entries, and calendar overview. This was done through extensive manual testing from all team members - including attempts to break the system (for example, attempting to bypass the designed flow and submit a journal entry before a user entry).

- **User Testing**: Conducted amongst ourselves and invited friends/family to ensure the interface was intuitive and easy to navigate. This was executed via brainstorming and consulting with each other to determine what the best layout would be for each page, and then feeding back after implementation for any adjustments.

System Limitations

Some limitations identified during testing included:

- Potentially a limited scalability due to the use of mySQL as the database
- Dependence on external APIs for content, which may affect availability if the APIs are down. Defaults were implemented to account for this.
- Session timeout might inconvenience users who take longer to write journal entries
- Basic email validation checks for format but not whether or not it is active.
- Passwords are not required to be strong - this would have security implications.
- While HTTP templates have allowed for a user friendly system, there is a lack of dynamically updating content using Javascript

Future expansion

In the future, the app could be expanded:

- Made mobile friendly or as a mobile app
- Obtaining a production key for the use of gifs
- More dynamically updating content
- Allowing multiple entries per day
- Giving suggestions to the user based on their mood track data -> if they have been feeling down, suggesting places they could seek help.
- Using AI tools to associate journal entries with their moods, and make suggestions about what is linked to positive moods and what to negative moods.

Conclusion

GIFeels successfully provides a platform for users to log their moods, receive motivational content, and maintain a daily journal. Despite some challenges and limitations, the project met its primary objectives and offers a valuable tool for emotional wellbeing and mindfulness. Future enhancements could include integrating more robust database solutions, expanding content options, and improving scalability to accommodate a larger user base. It could also be more personal by adding certain features such as an integrated playlist or colour customisation.