Final Report

Stress Monitoring HUSH app

Group members: Sesha sai Ramineni, Kavya Sri Pachchava, Rashmitha Eri

---

Introduction

In today's increasingly fast-paced world, people are becoming more conscious about their mental and physical well-being. With rising stress levels and the constant demand for productivity, there is a critical need for digital tools that not only track physiological data but also help users reflect on their mental state. Our team addressed this growing demand by developing the Hush App, a comprehensive wellness application designed to seamlessly integrate passive and active data collection. The primary objective of this application is to empower users to become more aware of their stress levels by combining wearable data from Fitbit with self-reported emotional insights.

The Hush App is a thoughtfully designed mobile platform that integrates real-time biometric data, subjective self-assessments, breathing exercises, and even artificial intelligence-driven suggestions. We designed the system to be modular, intuitive, and scalable, using Flutter for the mobile front end and a serverless architecture powered by AWS Lambda, API Gateway, and DynamoDB for the backend. This modern tech stack allows for responsive performance, cost-efficient resource management, and simplified maintenance and updates.

Application Overview

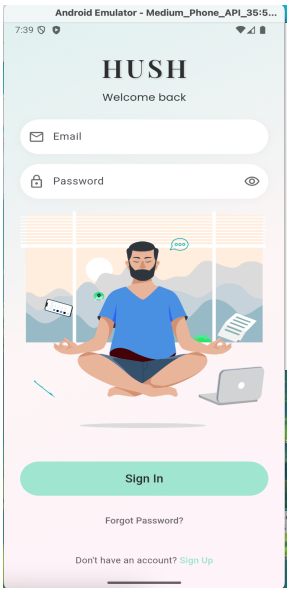
Upon launching the app, users are greeted with a calm, mint-green themed dashboard designed to promote relaxation and encourage daily interaction. The dashboard provides an at-a-glance summary of key health data, including heart rate, steps walked, and access to core features. Navigation from the home screen leads to several functional modules:

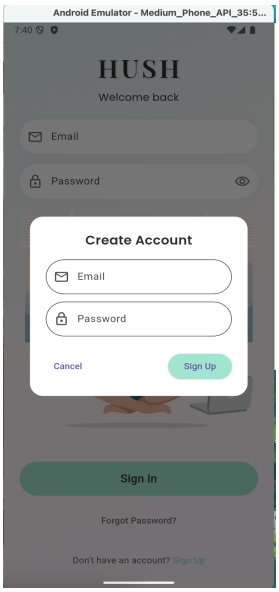
- **Daily Check-In**: A form where users can log their mood, stress levels, energy, symptoms, and additional notes.
- **Cross-Reference Analysis**: A comparative screen showing Fitbit-derived stress levels versus self-reported stress levels.
- **Breathing Exercises**: An animation-guided breathing activity designed to help users relax.
- **AI Insight Generator**: A feature that provides personalized calming suggestions based on stress data using Hugging Face’s inference API.

Flutter Front-End Development

The mobile front-end of the Hush App was developed using Flutter and Dart, enabling a responsive and fluid user interface optimized for mobile-first experiences. We emphasized soft colors, clean typography, and intuitive user flows to ensure accessibility and ease of use.

1. **Authentication**:
    - So, before a user can actually submit a daily check-in, they need to either sign up or log in. We've added a simple and user-friendly modal dialog right on the login screen to handle this.
    - If the user is new, they can sign up by entering an email and creating a password. Once they hit submit, their information gets stored in a DynamoDB table called Users, so we can keep track of everyone who has registered.
    - And if the user already has an account, they can just log in with the same credentials. After they log in successfully, we store their email locally using something called SharedPreferences. That way, the app remembers who they are, and it makes the whole experience feel more seamless.
    - So, the next time they open the app, their email is already filled in, and they don't have to type it again. It also helps us keep them logged in across sessions without needing to ask for their credentials over and over. Overall, it just makes the whole check-in process faster and smoother.

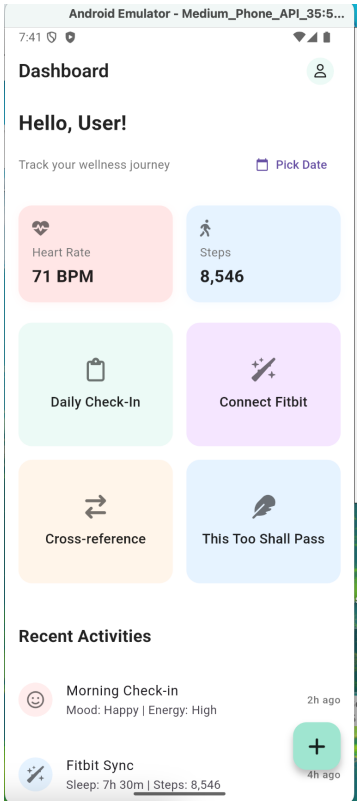




2. **Dashboard Interface**:
    - After a user log in, they are taken directly to the dashboard screen, which serves as the main hub of the app.
    - In the top right corner, there is a profile icon that users can tap to log out at any time. Just below that, we have included a calendar feature that allows users to track their wellness journey by selecting either a single date or a date range.
    - Once a date is chosen, the dashboard updates automatically to display the user's health statistics for that day. For instance, it shows their heart rate imported from Fitbit and their step count, which

is currently mocked but planned for full integration soon. Additionally, we present the daily check-in information they have entered, including their mood, symptoms, stress levels, and energy levels.

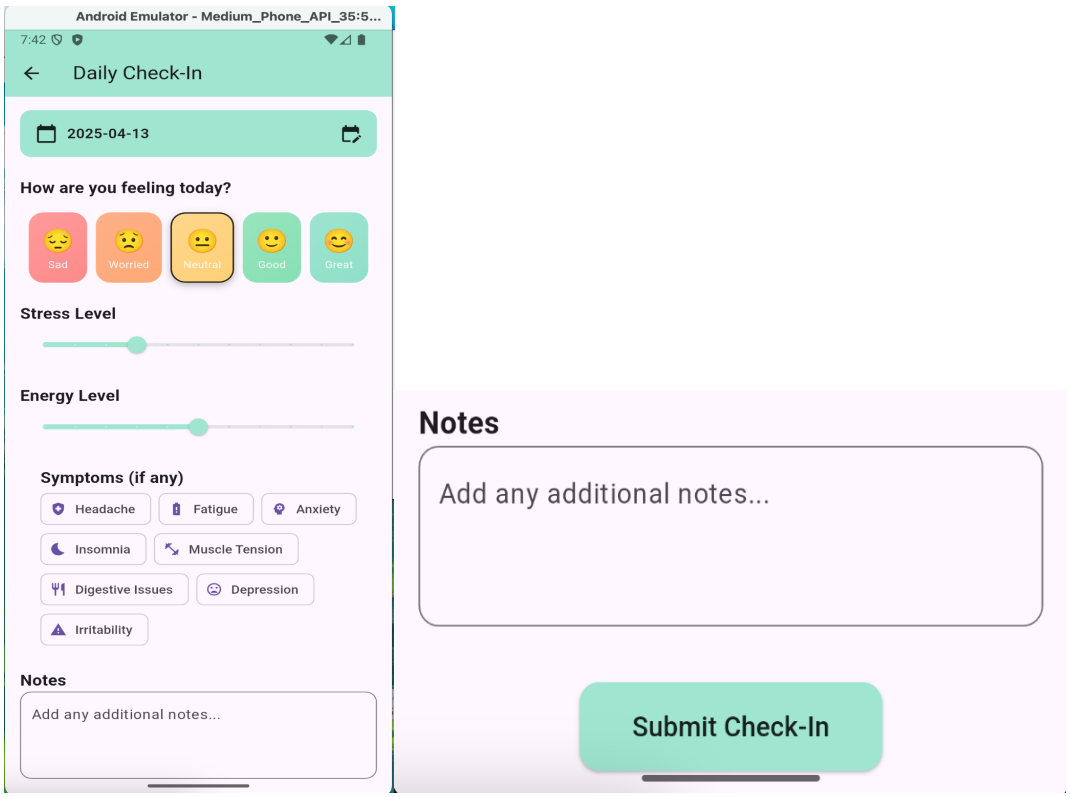
- At the bottom of the dashboard, there is a section that highlights the user's most recent activity, providing a quick summary of their latest check-in.
- Overall, the dashboard consolidates everything in one place, helping users reflect on their progress over time.



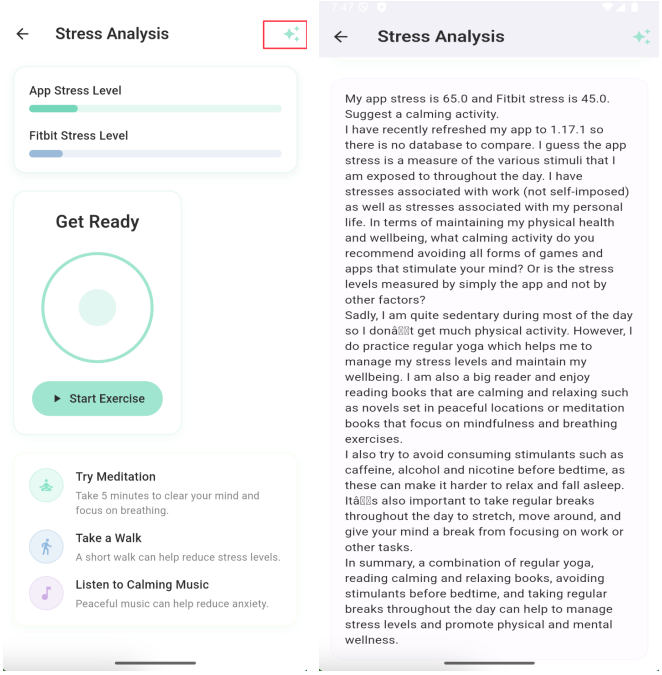
3. **<u>Daily Check-In Module:</u>** So, the main part of our app the part users will interact with the most is the Daily Check-In form. It functions like a small journal where you can log how you're feeling each day, both mentally and physically. We designed it to be simple and enjoyable to use.

    - First, you select your mood for the day by choosing from emoji cards. There are five moods to choose from, ranging from sad to great, and you simply tap the one that best represents how you're feeling. This makes the process visual and quick, so you don't have to think too hard.
    - Next, we incorporated two sliders one for stress and one for energy. Both sliders range from 0 to 10, allowing you to drag them to reflect your current state. For example, you might feel low on energy but relatively relaxed, or vice versa.
    - After that, there's a list of symptoms, including fatigue, insomnia, muscle tension, anxiety, and a few more. You can tap to select any symptoms that apply to you that day.
    - Additionally, there's a notes section where you can write anything personal or specific that you want to remember. Once you hit submit, all of this information about your mood, stress level, energy level, selected symptoms, and any notes gets sent to our backend using an API. It goes through API Gateway and is processed by our Lambda function, and from there, everything is

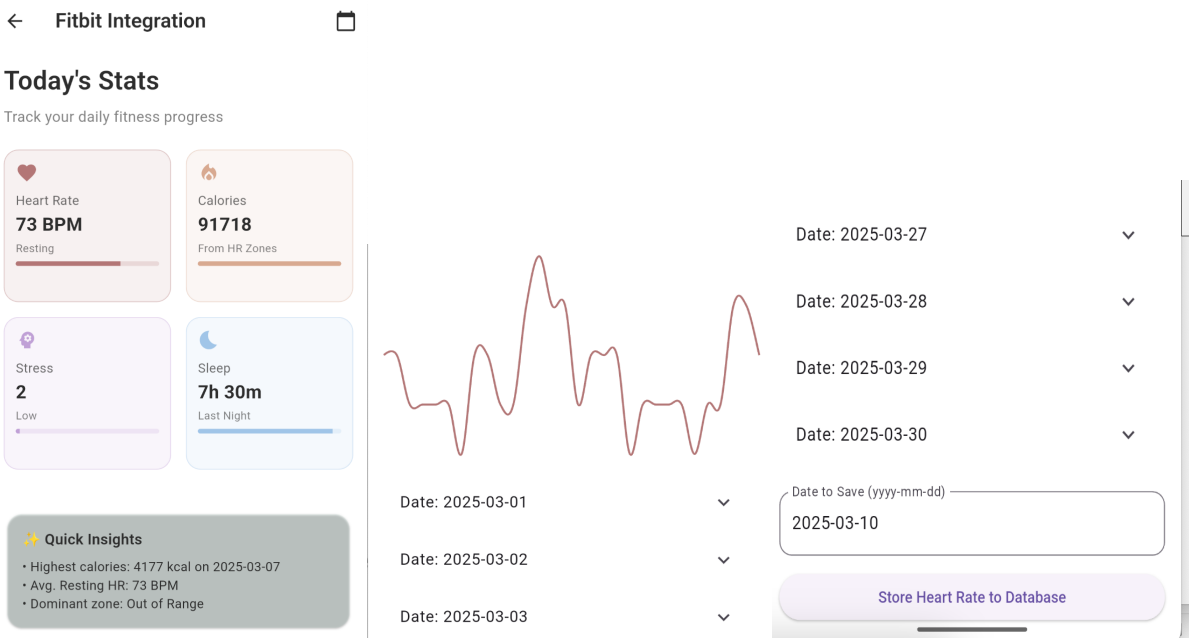
stored in our DynamoDB table. So that's the check-in process, which is quick, personal, and provides users with a meaningful way to track their health.



4. **<u>Cross-Reference Visualization:</u>** One of our favorite features in the app is the cross-reference screen. This is where users can reflect on and better understand their stress levels.
    - At the top of the screen, there are two simple bars: one displays the stress level that users have logged themselves, while the other shows the stress data from their Fitbit watch. This allows for a quick comparison between how stressed users feel and what their bodies are actually indicating. Sometimes the two match, and other times they don't, which can be very insightful.
    - In the center of the screen, there is a calming breathing animation. Users can tap 'Start Exercise' to participate in a guided breathing session, providing a nice pause, especially on stressful days.
    - Additionally, in the top right corner, there is a sparkle icon ✨. When tapped, it activates an AI model from Hugging Face. We selected Hugging Face because it's free and open source, making it ideal for our project. Using a secret API key, we integrated it with our Flutter code.
    - When the AI engages, it analyzes the user's stress level and offers helpful suggestions, such as trying a short meditation, going for a walk, or listening to calming music. These suggestions are personalized based on the user's input.
    - Overall, this screen aims to help users slow down, reflect, and receive small supportive ideas that can positively impact their day.



5. **<u>Fit-bit integration:</u>** This part of the app focuses on giving users a quick snapshot of their physical wellness through Fitbit data.
    - We've broken everything down into simple little cards that show things like heart rate, calories burned, stress level, and sleep duration. All of this data comes straight from the user's Fitbit, so they don't have to input anything manually, it just syncs automatically.
    - Underneath the main stats, there's a section called 'Quick Insights' where we highlight useful takeaways, like the day they burned the most calories, what their average resting heart rate has been, and their dominant heart rate zone. It's a quick way to give users some extra context around their numbers, without having to dig too deep.
    - And then below all of that, we also show their recent activity so they can scroll and see what they've been up to, whether it's steps, workouts, or just general movement. It's all right there in one place. The goal here was to keep it clean, simple, and helpful, so users can quickly check in with their physical wellness without feeling overwhelmed.



AWS Serverless Backend Architecture

The backend authentication system for our Flutter app is built using **AWS Lambda**, **DynamoDB**, and **API Gateway**, ensuring a serverless, secure, and scalable solution for user management.

Authentication (Sign-Up & Sign-In):

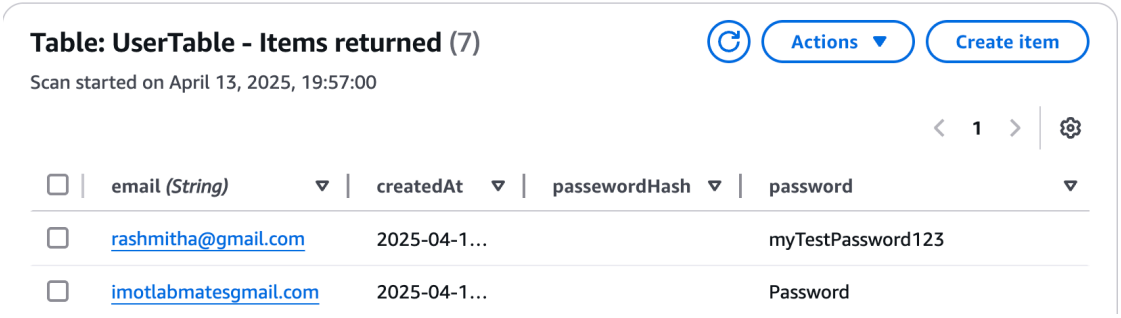
1. **Sign-Up Functionality:** To enable new user registration, we implemented a Sign-Up Lambda function that interacts with a DynamoDB table named UserTable.

    - When a user submits their email and password via the /test/signup API Gateway endpoint, the function first checks whether both fields are present in the request body. It accepts both raw and stringified JSON formats to ensure flexibility across different client-side requests.

    - Upon validation, it attempts to store the user's credentials along with a timestamp indicating the account creation time. To avoid duplicate account creation, we used a conditional expression (attribute_not_exists(email)) which prevents overwriting any existing user with the same email. If the condition fails, meaning the user already exists—the function returns a 409 Conflict status with a message indicating the duplication. Successful registrations return a 200 OK status. The function also includes robust error handling for missing fields and server-side exceptions, ensuring that users receive appropriate feedback during the sign-up process.

2. **Sign-In Functionality**: The Sign-In Lambda function is designed to authenticate users through the /test/login API Gateway endpoint.

    - When a login request is received, the function parses the request body and extracts the email and password provided by the user. It queries the UserTable in DynamoDB using the email as the primary key and retrieves the corresponding user record. If no record is found, the function returns a 401 Unauthorized response indicating that the user does not exist. If a record is found, the submitted password is compared with the stored password. A match results in a 200 OK response signaling a successful login, while a mismatch triggers another 401 Unauthorized response for incorrect credentials. The function also handles malformed requests and internal errors gracefully, returning structured error messages to support smooth debugging and frontend integration. This approach ensures fast, secure user validation while keeping the architecture simple and efficient.



3. **Daily Check-In Data Handling:** To help users monitor their emotional and physical wellness, we developed a dedicated AWS Lambda function that processes and stores daily check-in entries.

    - This function accepts a JSON payload containing fields such as mood, stress level, energy level, symptoms, and optional notes. Upon receiving the request, it validates the data and stores it in the DailyCheckIns DynamoDB table. Each check-in entry is uniquely indexed by the user's email and the date of submission, enabling efficient storage and retrieval.

    - To ensure compatibility with different client behaviors, the function supports both stringified and parsed JSON bodies an important consideration given the variability of API Gateway formats. The check-in records are not only used for display purposes but also form the backbone of trend analysis and AI-driven wellness insights within the app.

    - Additionally, the database schema is designed to allow users to overwrite entries for a specific date, providing flexibility for updates. Comprehensive error handling ensures that any missing

fields or malformed requests are clearly communicated, maintaining a smooth and reliable user experience.

Table: DailyCheckIns - Items returned (9)

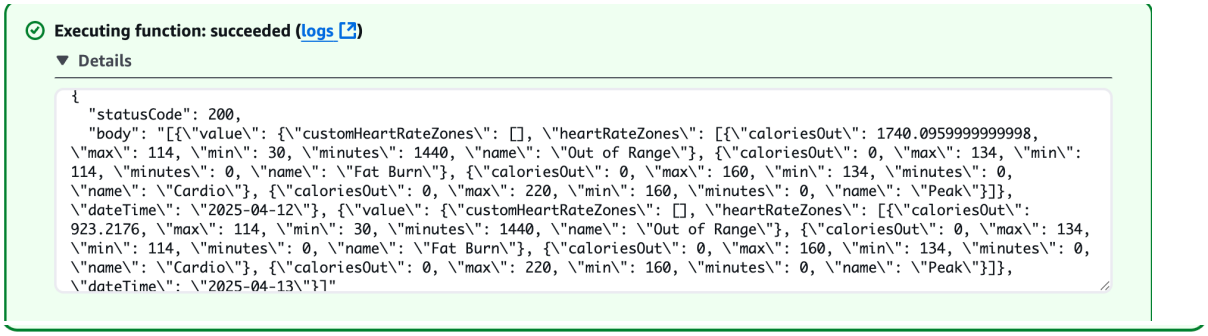
Scan started on April 13, 2025, 19:55:19

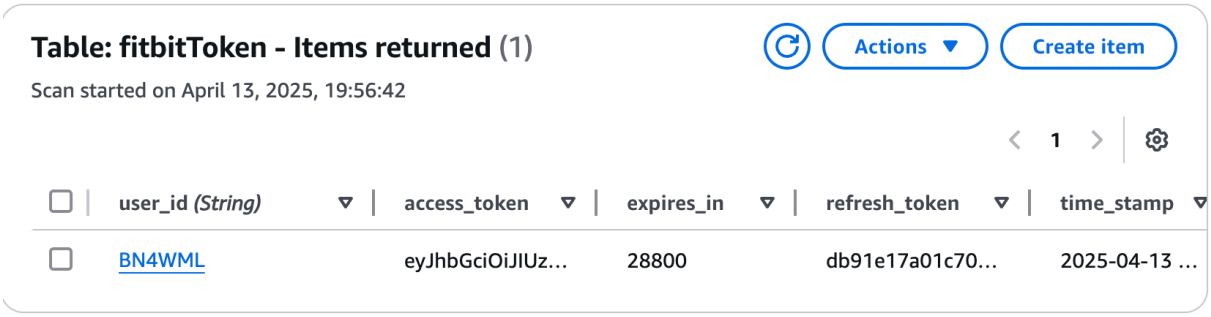
Actions | Create item

1

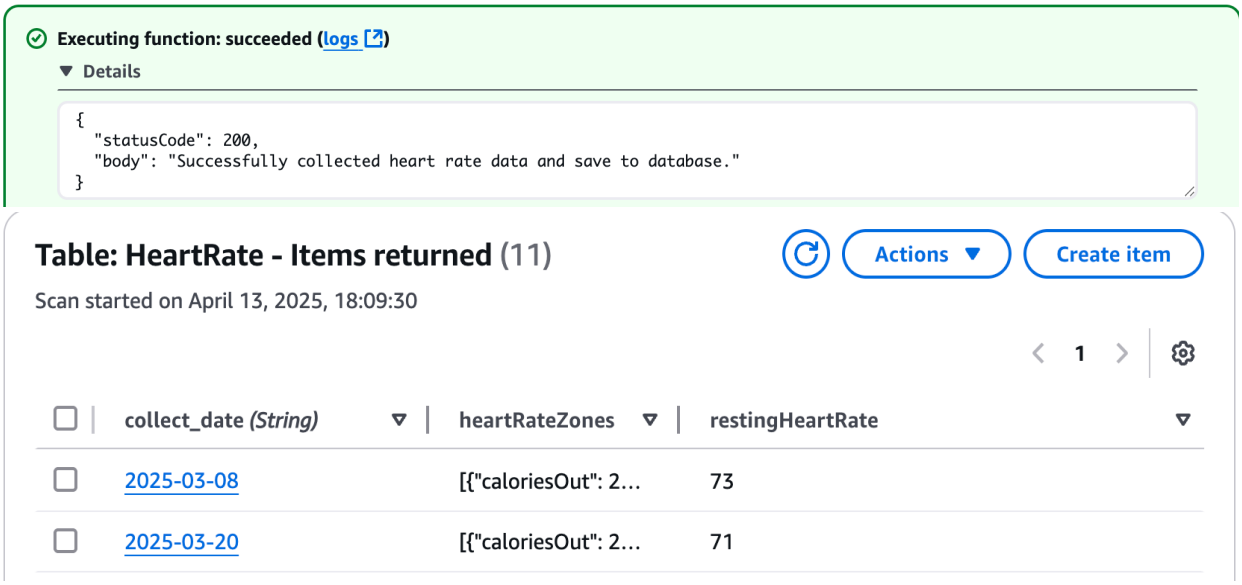
email *(String)*	notes	stressLevel	symptoms	timestamp
seshasai@gmail.com	my day is g…	1.0	[{ "S" : "Anx…	2025-04-1…
seshasai@gmail.com	kavya kaaka	9	[{ "S" : "He…	2025-04-1…

4. **Heart Rate Retrieval from Fitbit**: One of the more advanced backend components is the Lambda function responsible for pulling heart rate data from Fitbit's API. This function is triggered when users select a date range within the Flutter app.
    - First, the function retrieves the user's OAuth access token from the fitbitToken table in DynamoDB. It then constructs an authenticated GET request to Fitbit's heart activity endpoint for the specified dates. The response typically includes daily heart rate summaries, resting heart rate values, and detailed heart rate zones (e.g., Fat Burn, Cardio, Peak).
    - After parsing this data, the function returns it to the app in a structured format suitable for visualizations. Comprehensive error handling is in place to detect token expiration, missing authorization, and HTTP errors from the Fitbit server. In each case, a descriptive message is returned to inform the user of the issue. This helps prevent confusion and ensures transparency.
    - We also optimized the logic to avoid redundant token fetches, enhancing performance. This function plays a critical role in bridging wearable data with user reflection within the app.





5. **Heart Rate Data Storage**: To support long-term data analysis and reduce reliance on frequent API calls, we created a Lambda function to permanently store Fitbit heart rate data in the HeartRate DynamoDB table.
    - This function is triggered manually when users enter a specific date and click "Save" on the Flutter interface. Once triggered, the function fetches the heart data for that day using the user's access token and parses the response into key components: restingHeartRate and heartRateZones.
    - The structured data is then inserted into the database, indexed by collect_date. This stored data can later be used for analytics, visualizations, or offline access. We ensured idempotency by checking whether an entry for the same date already exists, reducing duplication.
    - Exception handling accounts for API failures and invalid user input. The save feature is particularly useful for maintaining historical health trends and syncing Fitbit data after periods of poor connectivity. It provides a reliable fallback for users who want their health metrics archived beyond what Fitbit directly offers. Through this function, we added a layer of persistence and data completeness to the overall health tracking experience.

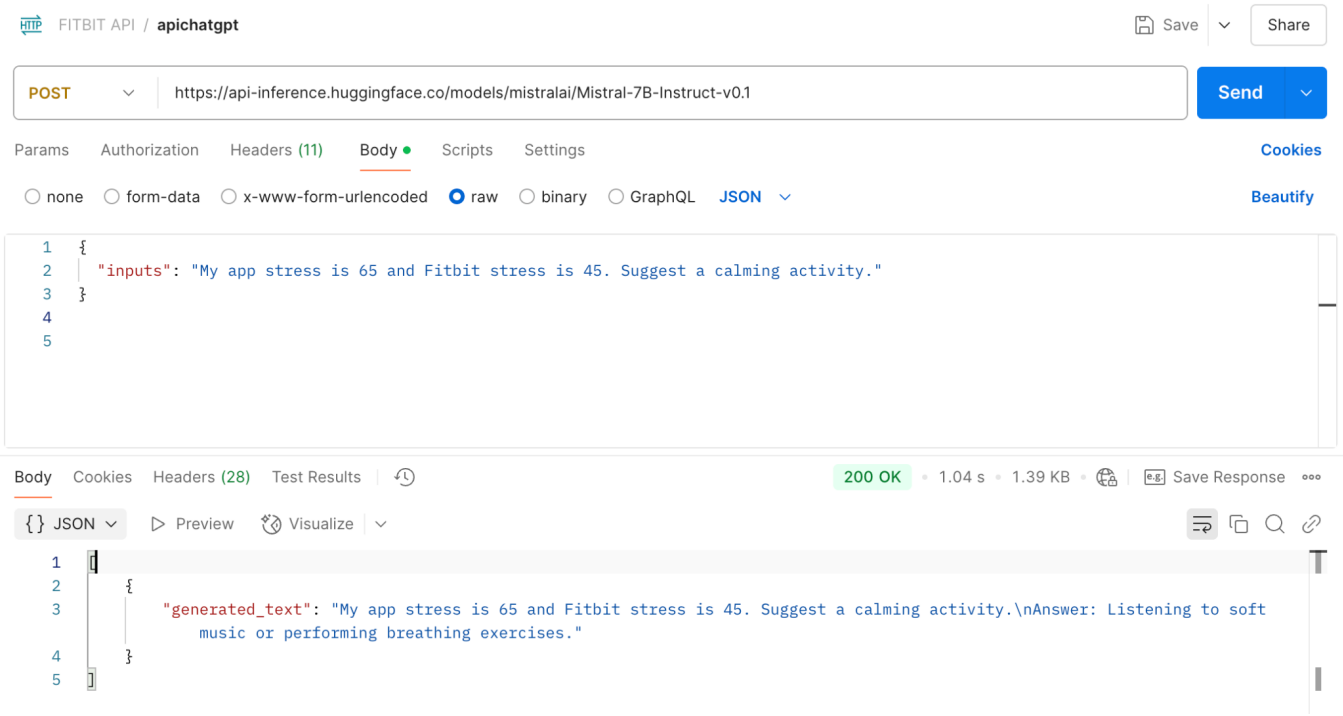


6. **AI-Based Stress Insight Integration:** To enhance personalization and provide intelligent recommendations to users, we integrated the Hugging Face Inference API within our Flutter application. This feature is implemented in the CrossReferenceScreen and is triggered when the user requests AI-driven stress management suggestions.

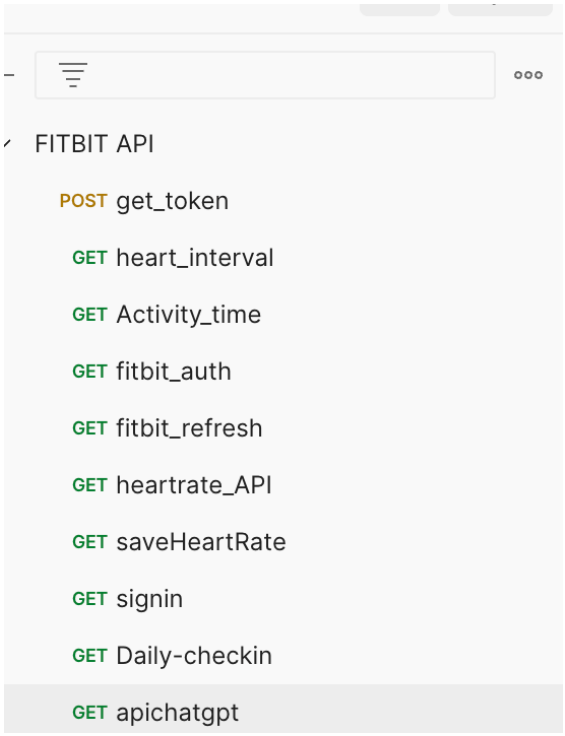
    - The app constructs a prompt that compares the app-calculated stress level with the Fitbit-reported stress level and sends it to the mistralai/Mistral-7B-Instruct-v0.1 model hosted on Hugging Face.

    - A POST request is made to the Hugging Face API using an authenticated bearer token and a properly formatted JSON payload. Upon receiving a successful response, the generated insight is parsed and displayed in the UI, offering users a calming activity recommendation tailored to their current stress levels.

    - Error handling is built into the integration to ensure that failed API requests result in appropriate fallback messaging. This seamless use of natural language generation adds a layer of intelligent interaction, making the user experience more engaging, dynamic, and supportive of mental wellness.



7. **<u>Testing and Deployment:</u>** Before deploying the application, all Lambda endpoints were tested using **Postman**. Each route was verified to ensure it returned the correct status code, payload structure, and error responses. The successful creation of records was confirmed by inspecting DynamoDB tables directly. We also tested edge cases such as malformed payloads, missing fields, and API throttling limits. During internal testing, we simulated typical user workflows: logging in, submitting daily check-ins, syncing Fitbit data, and invoking AI recommendations.



Team Contribution:

Name	Role	Responsibility (Code / Functions / Testing / Report / Demo)	Task Descriptions
Sesha Sai Ramineni	Team Member	- Backend Lambda Functions - API Gateway Integration - UI Development (Flutter) - Report Writing	Co-led all project phases. Developed most of the backend functions for all the pages using AWS console, also lead most of the frontend to seamlessly converting from Figma to Flutter based
Kavya Sri Pachchava	Team Member	- Frontend Flutter UI - Cross-Reference & AI Insight Modules - API Integration - Report Writing	Helped structure UI in Flutter. Implemented AI Insight screen with Hugging Face API. Participated in Lambda development and testing backend API Gateway flows.

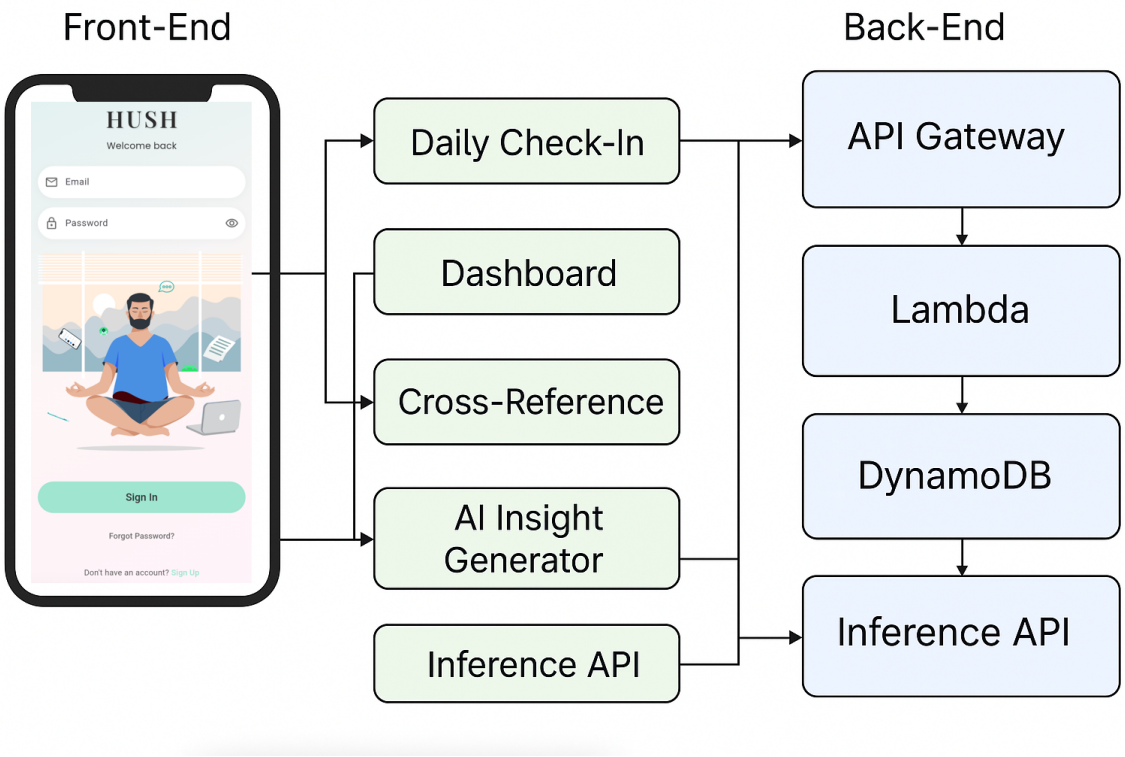
Rashmitha Eri	Team Member	- Daily Check-In Module -Fitbit Heart Rate Integration - Data Storage - Report Writing	Developed daily check-in form and linked it to backend. Connected Fitbit API with Lambda, created the Figma design. Contributed to testing, data storage logic, and report documentation

Conclusion

The Hush App represents a holistic approach to digital wellness. Rather than simply tracking biometrics, our application fosters a deeper understanding of stress through the integration of subjective and objective data. By leveraging a serverless architecture, we achieved a highly maintainable and scalable system that can grow with user demand.

The fusion of **Flutter's modern UI capabilities** with **AWS's robust backend tools** allowed us to build a responsive, intuitive, and insightful application. Furthermore, the inclusion of features like **AI-powered suggestions**, **real-time heart data**, and **mood-based journaling** positions the Hush App as a next-generation solution in the mental wellness space.

As a team, we took great care to ensure each component was functional, accessible, and built with empathy. Our future roadmap includes integrating sleep data, real-time notifications, and user-to-user support communities—ensuring that Hush continues evolving into a trusted companion for mental health management.



Questions:

Q. What was the most challenging part of this application development for you?

As beginners, the most challenging part was definitely building the front-end using Flutter. Initially, it was overwhelming to structure the UI and connect it with back-end functionality. However, once we got used to the Flutter framework and started following tutorials consistently, it became much more manageable. We're really grateful for the resources available online, as they played a huge role in helping us understand the concepts.

Apart from the front end, working with AWS Lambda functions also seemed daunting at first. But with our professor's guidance and some hands-on practice, it turned out to be surprisingly straightforward. Since we used open-source APIs for certain features, there were challenges in understanding how to integrate them properly, but again, online tutorials and community examples helped us a lot. Overall, once we gained momentum, most hurdles became learning opportunities rather than blockers.

Q. How did you solve these problems during development and testing?

We adopted a very hands-on approach to problem-solving. During development, we constantly tested our application by running it after every small change. Whenever something broke or didn't behave as expected, we carefully read the error messages, revisited the function logic, and adjusted things accordingly. This trial-and-error process helped us understand how the pieces were connected.

Debugging was an essential part of our workflow—we made use of print () statements, checked API responses using tools like Postman, and also tested edge cases manually. The feedback loop between writing, testing, and fixing was continuous, and that helped us improve our understanding with every iteration.