University of the Philippines Manila
College of Arts and Sciences
Department of Physical Sciences and Mathematics

ACAPP: A Mental Health Application for the Well-Being of Filipino University Students

A special problem in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Computer Science

Submitted by:

Ma. Gishelle Anne M. Ngo

June 2023

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "ACAPP: A Mental Health Application for the Well-Being of Filipino University Students" prepared and submitted by Ma. Gishelle Anne M. Ngo in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Avegail D. Carpio, M.Sc.
Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Richard Bryann L. Chua, Ph.D (*cand.*)		
2. Perlita E. Gasmen, M.Sc. (*cand.*)		
3. Ma. Sheila A. Magboo, M.Sc.		
4. Vincent Peter C. Magboo, M.D., M.Sc.		
5. Marbert John C. Marasigan, M.Sc. (*cand.*)		
6. Geoffrey A. Solano, Ph.D.		

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Vio Jianu C. Mojica, M.Sc.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences and Mathematics

Marie Josephine De Luna, Ph.D.
Chair
Department of Physical Sciences and Mathematics

Maria Constancia O. Carillo, Ph.D.
Dean
College of Arts and Sciences

Abstract

A recent study found that 75% of university students experienced severe psychological distress, while 25% experienced mild to moderate levels of distress during the COVID-19 pandemic [1]. Factors such as academic, personal, and family-related issues contribute to psychological distress among college students, leading to physical, emotional, and psychological strain that negatively impacts their overall well-being and functioning. ACAPP, a responsive web application, has been developed to address these challenges by fostering self-awareness, cultivating healthy habits, and equipping student users with effective well-being strategies. While online mental health applications already exist, student engagement with such platforms tends to be low. To overcome this issue, ACAPP introduces a menu-based assistant feature that offers an interactive experience, aiming to enhance user engagement.

Keywords: mental health, well-being, Filipino university students, mHealth, mental health apps, menu–based assistant

Contents

List of Figures

List of Tables

I. Introduction

A. Background of the Study

According to World Health Organization [2], 1 in every 8 people in the world live with a mental disorder. However, most people do not have access to effective care. Despite the passing of a mental health law in 2018, mental healthcare in the Philippines remains insufficient, with limited government efforts and accessibility issues, particularly for people with disabilities [3]. Financial constraints and fear of stigmatization and discrimination further discourage individuals from seeking professional help, resulting in many Filipinos relying on themselves and only seeking help when their condition is severe [4]. The COVID-19 pandemic has further negatively impacted the mental health of Filipinos with reports of moderate-to-severe anxiety, moderate-to-severe depression, and psychological impact during the early phase of the pandemic [5]. In a recent study conducted by [1], 75% of university students participants experienced severe psychological distress, while 25% experienced mild to moderate levels of psychological distress amidst pandemic.

Well-being encompasses a positive state that involves quality of life and the ability to contribute to the world with a sense of purpose [6]. Considering students within their environment, their well-being represents whether they can function effectively to act in response to the school demands and whether the school can accommodate to students' needs and expectations optimally, involving a balance between the strengths of the students for effective functioning and the school resources for healthy growth. Hence, student well-being comprises eudaimonic indicators, such as fully functioning and positive development, and hedonic indicators of subjective well-being, such as presence of positive affect, absence of negative affect, and life satisfaction [7]. College students experiencing psychological distress due to academic, personal, and family-related issues may suffer from physical, emotional, and psychological strain, affecting their overall well-being and hinder-

ing their functioning [1]. Therefore, prevention or early intervention measures are necessary to address this population's distress.

Filipinos are increasingly showing interest in telemedicine and telehealth, particularly in virtual mental health programs [8]. This indicates that online mental health services are becoming more sought-after. Several technologies have been created and are presently obtainable in the market to offer convenient mental health services. A study conducted by [9] revealed that smartphone apps are the most commonly used technology to support mental health and well-being in the adult population. Different technologies have distinct purposes, with apps offering guided activities, relaxation techniques, and tracking features. Social media and discussion forums provide a platform for individuals to learn from others' experiences and apply that knowledge to their own situation. Web-based programs and websites are helpful in identifying ways to manage stress and anxiety on a daily basis.

B. Statement of the Problem

Caring for mental health is an important aspect of overall well-being as it affects one's thoughts, emotions, behaviors, and relationships with others [10]. Poor mental health can lead to mental disorders that can negatively impact one's quality of life. On the other hand, taking steps to improve mental health can enhance well-being and prevent the onset of mental disorders.

In the Philippines, the shortage of mental health professionals remains a significant challenge, with just over 500 psychiatrists in practice [11]. Additionally, financial constraints, stigmatization, and discrimination are significant barriers to accessing mental health services in the country. While there is currently one mental health app available to the public specifically made for Filipino adults [12], it may not be sufficient to meet the diverse needs of the population.

Despite the availability of effective online mental health interventions, university students exhibit low engagement and high attrition rates when using men-

tal health apps [13][14], which could limit the efficacy and effectiveness of these interventions. As such, there is a need for a mental health app designed specifically for university students. This app should take into consideration the unique challenges that university students face including academic, personal, and family-related stress. By tailoring mental health interventions to the needs of this population, it may be possible to improve engagement and reduce attrition rates, thereby enhancing the efficacy and effectiveness of online mental health interventions in the Philippines.

C. Objectives of the Study

The main objective of this project is to create a mental health application that is web-based and equipped with tools for fostering self-awareness, cultivating healthy habits, and acquiring well-being strategies. Additionally, the application will incorporate a menu-based assistant to enhance user engagement. The student users would be able to perform the following functions:

1. Register in the system
2. Customize his or her profile
3. Track and record their daily mood and identifying contributing factors
4. Track and record their chosen habits
5. Learn well-being strategies
6. Express their thoughts and feelings in a diary
7. Access to online articles related to mental health
8. Talk to a menu-based assistant for assistance and guidance
9. Receive daily insights of their mood, habit, well-being strategies tried, and online articles read

10. Receive notifications to encourage mood and habit tracking and to alert when daily insights are available

The menu-based assistant included in the application offers the following features:

1. Welcoming the user
2. Ask user's preferred notification time
3. Send reminders to encourage mood and habit tracking
4. Notify the user when daily insights become available
5. Provide guidance and support to users regarding the primary features of the app, including mood and habit tracking, well-being strategy development, diary writing, daily insights, and access to mental health resources

On the other hand, the admin would be able to perform the following functions:

1. Manage contents of the app including the well-being strategies and online articles.

D. Significance of the Project

Mental health applications aim to address issues in mental health in terms of accessibility, financial constraints, stigma, discrimination, and transportation. These apps have been helpful during the pandemic as they allow people to receive assistance without physically visiting clinics or offices, reducing the transmission of the virus.

Prevention, as the saying goes, is better than cure. This mental health application aims to prevent mental health issues from worsening by promoting self-awareness and providing well-being strategies to university students. Through mood monitoring, habit tracking and diary features of the application, users can increase their self-awareness and understanding of themselves.

The implementation of a menu-based assistant in this mental health application is to provide an interactive experience to users. The assistant can offer guidance, assistance, and encouragement, motivating users to use the app.

E. Scope and Limitations

1. This mental health application is intended for Filipino university students only.

2. The virtual assistant in the app is a menu-based assistant only that presents predefined options to the user for selection, rather than accepting raw text input.

3. The menu-based assistant can provide information and guidance for mental health prevention but cannot replace the expertise, personalized care, and professional judgment of qualified mental health professionals. It is essential to consult with a licensed mental health professional for accurate diagnosis, individualized treatment plans, and ongoing support.

F. Assumptions

1. The user understands the English language.

2. The user of the app must be at least 18 and above.

3. The app can benefit anyone who can use it as it serves as an online log-book for mental health hygiene practices and provides helpful resources. However, if a user is in crisis, having suicidal thoughts, experiencing severe mental health problems, psychotic episodes, psychiatric disorders, serious addictions, or engaging in risky behavior, they may need additional support or treatment from a mental health professional [10].

II. Review of Related Literature

According to World Health Organization, well-being is a positive state that involves both the quality of life and the ability to contribute to the world with a sense of meaning and purpose [6]. The resilience, capacity for action, and preparedness to overcome challenges are indicators of a society's well-being. In recent years, there has been a significant increase in the use of technology for promoting well-being among students. With the rising rates of mental health concerns among young adults, these apps have the potential to provide accessible and convenient support for individuals in need. This literature review aims to examine the current research on mental health apps for university students and explore their potential benefits and limitations.

The majority of university students are considered adults and are generally aged 18 and above. According to a survey conducted by [9], smartphone apps are the most commonly used technology to support mental health and well-being in the adult population. Eisenstadt et al. conducted a systematic review of available research on mental health apps that promote emotion regulation, positive mental health, and well-being in the general population aged 18-45 years in 2021[15]. The review analyzed 52 publications and used 39 randomized controlled trials for the meta-analysis. They found that there is empirical support for mental health apps that apply theoretical insights from Cognitive Behavioral Therapy (CBT), a psychotherapy technique that teaches individuals to think, react, and behave differently to reduce anxiety, as well as mindfulness approaches, which involves moment-by-moment awareness of what a person is experiencing, such as paying close attention to breathing, noises, sensations in the body, inner feelings and thoughts, and our reactions to specific situations. Mood monitoring in various forms has been found to be well-supported empirically. It has been found to increase emotional self-awareness, and there is some evidence to suggest that a low level of self-awareness is a risk factor for anxiety, depression, and stress [15].

Serene, a mobile app developed by [16], was designed using therapies like

cognitive-behavioral therapy (CBT) and mindfulness-based cognitive therapy (MBCT). The efficacy of the app was tested on Canadian residents aged 18 and above to measure its impact on stress, depression, anxiety, self-compassion, well-being, and wisdom. The app includes features on dealing with unexpected distress or upsetting emotions, cognitive restructuring, and techniques for mindful journaling and self-compassionate writing. The study found that using Serene significantly reduced depression and increased self-compassion among participants. Through increased self-compassion levels, participants were able to deal with stressors more effectively by using adaptive emotional regulation and wise reasoning. Generally, participants reported that using all aspects of the app (mindfulness practices, cognitive restructuring, and journaling) helped improve their overall subjective well-being.

In another recent study, Chen et al. [17] conducted a systematic review and meta-analysis of mobile mindfulness meditation (MMM) targeted specifically at university students. The objective of the review was to analyze studies that reported the effects of MMM on primary outcomes, such as stress, anxiety, and depression, as well as on secondary outcomes, including mindfulness, well-being, and resilience. The results showed that MMM groups were more effective than control groups in reducing stress [18] and alleviating anxiety, but there was no significant difference in depression, which contrasts with the findings of Al-Refae et al. [16]. The study also found that the MMM groups were more effective than the control groups in enhancing well-being and improving mindfulness, which is consistent with the findings of Clarke and Draper [19], and Al-Refae et al. [16].

Jitanan et al. developed an app called Friend from Heart which incorporates mental and physical health to promote well-being in undergraduate students in a university in Thailand [20]. They found that the majority of students wanted an application that offers mental health information, such as ways to relieve stress and stress management, as well as physical health information, such as exercise, and eating healthy food. By incorporating both mental and physical health infor-

mation, the app can help students adopt a holistic approach to their well-being. To cater to the needs of students who prefer counseling services via messaging, chat, and listening to their problems, the app used a chatbot to provide basic assistance in the form of articles and videos based on the LINE system, a messaging platform.

Jeong et al. investigated the use of a social robot coach for delivering positive psychology interventions to college students living in on-campus dormitories to create an interactive experience and establish rapport with users [21]. The robot aimed to improve psychological wellbeing, mood, and readiness to change behavior of college students through seven sessions of interventions. It was equipped with additional features, such as weather forecast, jokes, music, and interactive games. The study involved 35 students who used the social robot coach in their dormitory for a period ranging from one week to a month. The results showed statistically significant improvements in psychological wellbeing, mood, and readiness to change behavior of the participants.

To provide a culturally appropriate mental health app, Hechanova-Alampay et al. tested a mobile app for mental health created for Filipinos called Lusog-Isip [12]. The participants of the study consisted of 392 Filipino adults (18 years old and above) with 39% students. Lusog-Isip app provides access to evidence-based interventions to improve well-being and coping strategies while providing access to a referral list of mental health services providers. The app was found to increase overall well-being among participants and improve their ability to cope with stress using adaptive coping strategies such as cognitive reappraisal, social support and relaxation/recreation. Results revealed that one of the advantages of the app is in terms of emotional release as the app have journaling and mood tracking feature, which allowed users to freely express themselves. Consistent with the findings of Eisenstadt et al. [15], self-awareness was observed as an important first step in managing mental health.

A study in [22] proposed a different approach by developing a virtual reality

(VR) mobile application called Serenity. The app aimed to help Filipino college students reduce their stress to prevent anxiety, lack of motivation, irritability, and depression. The main concept of the app is to allow the user to portray a traveler roaming around the Philippines with five different sceneries namely, the island of Palawan, Sagada, Siargao, Pagudpud, and Batanes. The app was then tested with nine (9) college student, each has three (3) students with different stress level classification. The developed virtual reality system is found to be effective however, for college students with low stress level only.

Multiple trials have demonstrated the effectiveness of a mental health app platform called IntelliCare in reducing symptoms of depression and anxiety for adults [14]. However, despite its efficacy, better engagement strategies are necessary for university students. According to Torous et al. (2018), the low engagement and adherence rates in most mhealth apps could be attributed to poor usability and a lack of user-friendliness [23]. In 2021, Lattie et al. investigated the uptake and effectiveness of the IntelliCare app platform for promoting mental wellbeing, stress management, and connecting college students with campus resources [14]. Unfortunately, the modified version of the app specifically designed for college students showed low uptake due to insufficient storage and a lack of guidance within the app. Furthermore, students were not fully aware of the app's capabilities, leading to underutilization of its functionalities. Many students who downloaded the app stopped using it early on, with nearly one in four students using the app only once after download. Although mobile-only apps are gaining popularity for mental health interventions, it is essential to understand their effectiveness and appeal among college students in comparison to other digital platforms.

Oti and Pitt conducted a scoping review to summarize the published literature on digital mental health interventions which take a user-centered approach in developing interventions for students in higher education [24]. The review included 23 studies that addressed depression, anxiety, overall wellbeing, and mental health awareness. The authors discovered a need for further investigation into the im-

pact of user-centered design practices on the effectiveness of digital mental health interventions in this demographic. They identified four crucial elements necessary for the success of online mental health interventions among students in higher education. Firstly, the interventions should be convenient, as students prefer interventions that can easily fit into their schedules. Secondly, personalization is crucial, as low engagement rates often result from a lack of personalization [9], and students have a strong preference for personalized interventions. Thirdly, the interventions should ensure anonymity, privacy, and safety, as students are concerned about the confidentiality of their data. Lastly, the use of appropriate language is essential, as students want to avoid patronizing and judgmental language in the online intervention.

To understand more the user needs, experience, satisfaction, and expectations of the publicly available mental health apps, a review analysis was conducted by Alqahtani and Orji wherein they mined reviews of 106 mental health apps from the App Store and Google Play Store [25]. They found out that users put more emphasis on the user interface and user-friendliness of the apps. Furthermore, 67.9 percent of the mental health apps available in the marketplace seem to be commercially motivated and heavily focused on app functionalities, with little or no attention to overall user experience and usability issues which significantly impact the effectiveness and user adoption.

The shortage of mental health professionals has still been an existing problem in developing countries. Philippines, as a developing country, have a little over 500 psychiatrists in practice [11]. Chatbot, a computer program designed to have a conversation with a human being, especially over the internet, can be a potential solution to this problem. In a scoping review done by Abd-alrazaq et al., out of the 41 chatbots reviewed, only 4 were implemented in developing countries [26]. These chatbots are commonly used for the delivery of therapy, training, and screening. A chatbot can also enhance engagement to the app through the natural human conversation and use of interactive features in a study conducted by Hungerbuehler

et al. in 2021 [27]. The chatbot was found to have response rates comparable to those of face-to-face interviews, suggesting that this could be a feasible way to collect data such as workplace mental health assessment. This technology can help reduce the burden of mental health professionals in the Philippines.

Despite the low engagement of university students to mental health apps available, 61.6% of students expressed their willingness to use these apps [14]. In order to fully utilize the features of these online mental health interventions and improve the wellbeing of university students, it is important to implement them in a more engaging and personalized manner.

III. Theoretical Framework

A. Well-Being

Well-being is a positive state experienced by individuals and societies. Similar to health, it is a resource for daily life and is determined by social, economic and environmental conditions. Well-being encompasses quality of life and the ability of people and societies to contribute to the world with a sense of meaning and purpose. Focusing on well-being supports the tracking of the equitable distribution of resources, overall thriving and sustainability. A society's well-being can be determined by the extent to which they are resilient, build capacity for action, and are prepared to transcend challenges [6].

Student well-being generally refers to a state of psychological, intellectual, emotional, physical, social, and spiritual wellness (Adams, Bezner, Drabbs, Zambarano, & Steinhardt, 2000). Taking the person-in-context perspective, student well-being represents whether the student can function effectively to act in response to the demands of the school and whether the school can accommodate to students' needs and expectations optimally, involving a balance between the strengths of the students for effective functioning and the school resources for healthy growth. Hence, student well-being comprises eudaimonic indicators, such as fully functioning and positive development, and hedonic indicators of subjective well-being, such as presence of positive affect, absence of negative affect, and life satisfaction [7].

Cristobal and Bance identified five dimensions that contribute to the well-being of Filipino university students [28]. These dimensions include the physical, psycho-emotional, social, spiritual, and academic dimensions.

In the physical dimension, students experience well-being by maintaining good health habits, such as getting adequate sleep and nutrition, exercising regularly, and having good grooming habits. They also do not have any physical health problems and maintain physical fitness.

In the social dimension, students experience well-being through social connections and support, which fulfill their relational needs. They exhibit empathy towards others and social awareness. They engage in helping others, social involvement, leadership, and service.

In the psycho-emotional dimension, positive and negative emotions play a crucial role in regulating emotions and achieving stability.

In the spiritual dimension, Filipino students value their faith, relationship with God, and commitment to moral values. They experience well-being when they engage in religious practices and personal experiences of spirituality, even during the COVID-19 pandemic.

Finally, in the academic dimension, students experience well-being through productivity, academic resilience, and effective task management in their academic lives.

B. Self-Awareness

Self-awareness refers to the capacity of becoming the object of one's own attention [29]. In this state, one actively identifies, processes, and stores information about the self. One becomes self-aware when one reflects on the experience of perceiving and processing stimuli.

Self-awareness represents a complex multidimensional phenomenon that comprises various self-domains and corollaries. Emotions or traits are private self-aspects that can be distinguished from public self-dimensions – visible characteristics such as one's body, physical appearance, mannerisms, and behaviors [29]).

Eurich categorized self-awareness into two types: internal and external [30]. Internal self-awareness involves recognizing one's values, passions, and aspirations by reflecting on one's thoughts, feelings, and behaviors. External self-awareness, on the other hand, involves comprehending how others perceive us by being open to feedback from others.

Moreover, the authors presented four self-awareness archetypes: Introspectors,

Aware, Seekers, and Pleasers. Introspectors understand themselves but do not challenge their views or seek feedback from others, which can harm relationships and limit success. Aware individuals know themselves and value others' opinions, benefiting from the advantages of self-awareness. Seekers, on the other hand, lack self-understanding and may feel frustrated with their performance and relationships. Pleasers are so preoccupied with others' perceptions that they may disregard their own needs, leading to poor choices that do not contribute to their success and happiness.

C. Interview with Guidance Counselors

After conducting interviews with guidance counselors from the University of the Philippines Manila Guidance and Counseling Program, including Ms. Micah Joy Lacerna, RPm, SHC (GC III), Ms. Beatriz Andrea Morente, RPm, SHC (GSS I), Mrs. Elgie A. Ocampo-Madla, MA Ed (Guidance), Ms. Jesusa Fernandez, MA, RGC (GSS II), and Ms. Maria Beatriz De Leon, RPm, MHFR (GC I), it was recommended that the app should focus on the needs of university students. To achieve this, the counselors recommended including a disclaimer to clarify the app's capabilities, focusing on prevention by disseminating information and providing various resources, recognizing the need for multiple approaches to mental health, promoting mental hygiene among students by incorporating evidence-based well-being strategies and recommended habits.

D. Rasa

It provides flexible conversational AI for building text and voice-based level 3 assistants. It is using a machine learning-based approach, which uses data from real conversations to improve accuracy over time. It is used by developers, conversational teams, and enterprises. Rasa has three major components that create contextual assistants: Rasa NLU, Rasa Core, and Rasa X (Rasa, 2019).

D..1 Rasa NLU

It is like the "ear" of the assistant—it helps the assistant understand what's being said. It takes user input in the form of unstructured human language and extracts structured data in the form of intents and entities [31]. Intents are labels that represent the goal, or meaning, of a user's specific input, while entities are the important keywords that an assistant should take note of. Using these data, the NLU model created can make predictions about the intents and entities in new user messages, including the messages that doesn't match any of the examples that the model has seen before.

The training pipeline of Rasa is composed of the tokenizer, Named Entity Recognition (NER), featurizers and intent classification models.

(1) Tokenizer is the first step in the pipeline because it prepares text data to be used in the subsequent steps. It takes a stream of text and split it into smaller chunks, or tokens, usually individual words. Some of the common tokenizers are WhitespaceTokenizer, Jieba, and SpacyTokenizer.

(2) Named entity recognition components are used to extract entities from user messages. Unlike tokenizers, the outputs are not needed in subsequent pipeline components, instead expressed in the final output of the NLU model. It includes the words in a sentence that are identified as entities, what kind of entities they are, and how confident the model was in making the prediction.

(3) Featurizers accept tokens and encode them as vectors, which are numeric representations of words based on multiple attributes. It should always come before the intent classifier and after the tokenizer.

(4) Intent classification models use the output of the featurizers to make a prediction about which intent matches the user's message. The

EmbeddingIntentClassifier works by feeding user message inputs and intent labels from training data into two separate neural networks which each terminate in an embedding layer. The cosine similarity is then used to calculate the similarities of the embedded message inputs and the embedded intent labels. Its output is

expressed as a list of intent predictions, from the top prediction to the bottom.

D..2 Rasa Core

It is Rasa's dialogue management component. It decides how an assistant should respond based on (1) the state of the conversation and (2) the context. It learns by observing patterns in conversational data between users and an assistant [31].

Policies have a very important role in the dialogue management. They are the components that train the dialogue model. Dialogue policies run in parallel, unlike the NLU training pipeline. All the policies make their predictions about the next best action on each turn in the conversation.

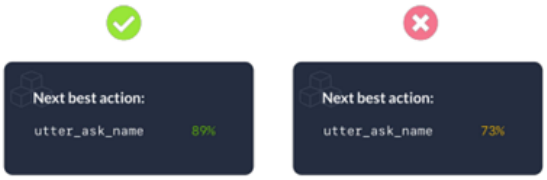

Figure 1: Predictions of next best action

In Rasa, there is a priority system to address cases where two policies predict with equal confidence. It automatically weights policies according to default priority levels, which are tuned to make sure the assistant chooses the most appropriate action when there's a tie. Higher numbers are given higher priority, and machine learning policies are given a priority of 1. Thus, the highest confidence and highest priority policy predicts the assistant's next action.

D..3 Rasa X

It is a toolset for developers to build, improve and deploy contextual assistants with the Rasa framework. It can be used to (1) view and annotate conversations, (2) get feedback from testers, and (3) version and manage models. Rasa X also enables developers to share their assistant with real users and collect the conversations they have with the assistant, allowing developers to improve their assistant without interrupting the assistant running in production [31].

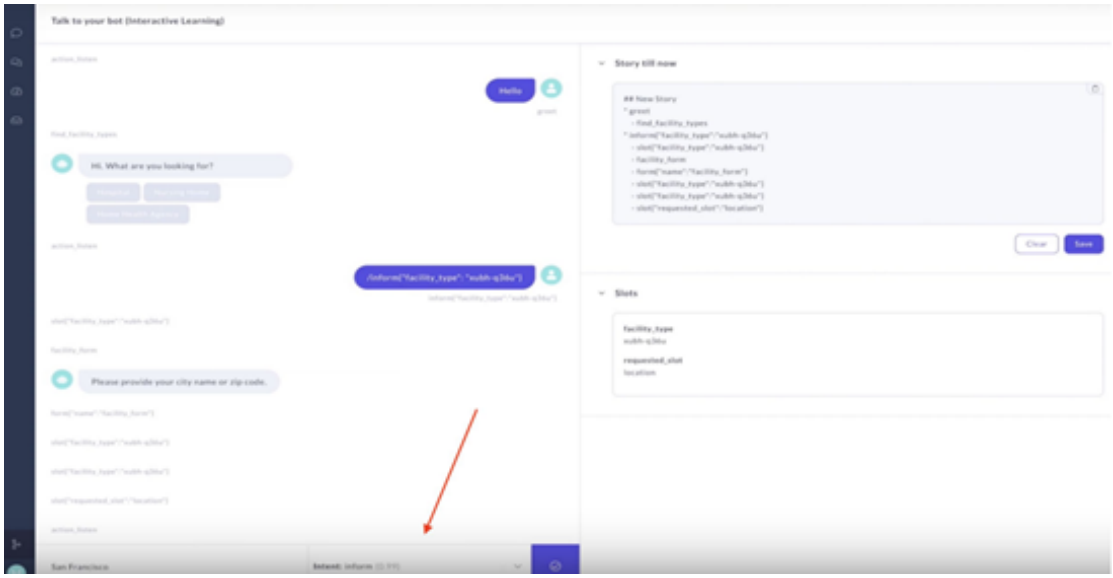

Figure 2: Rasa X Dashboard

IV. Design and Implementation

A. Use Case Diagram

Figure 3: Use Case Diagram

Figure 3 illustrates that the web app caters to two types of users: university students and admin. University students have a range of features available to them, such as the ability to register, customize their profiles, track their moods and habits, learn well-being strategies, write in diary, read online articles, receive alert notifications, receive daily insights, and engage with the assistant. On the

other hand, admins are responsible for managing the contents of online articles and well-being strategies.

B. Activity Diagrams

B..1 Registration of Students

Figure 4: Activity Diagram of Registration of Students

Figure 4 displays the registration process for student users. Upon signing up, the assistant will send a welcome message and present the app's disclaimer. Following this, the assistant will ask the student about their notification preferences.

B..2 Mood Tracker

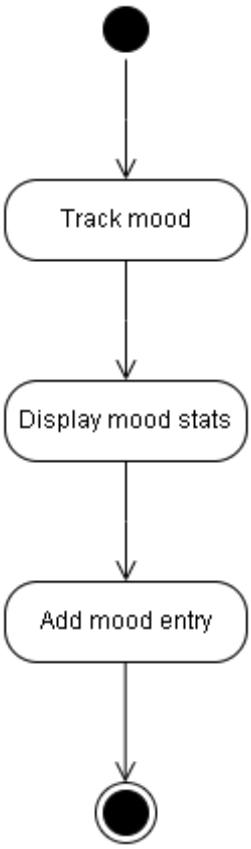

Figure 5: Activity Diagram of Mood Tracker

Within the mood tracker, the student can view their mood statistics. The student can then input their mood entry through the assistant.

B..3 Habit Tracker

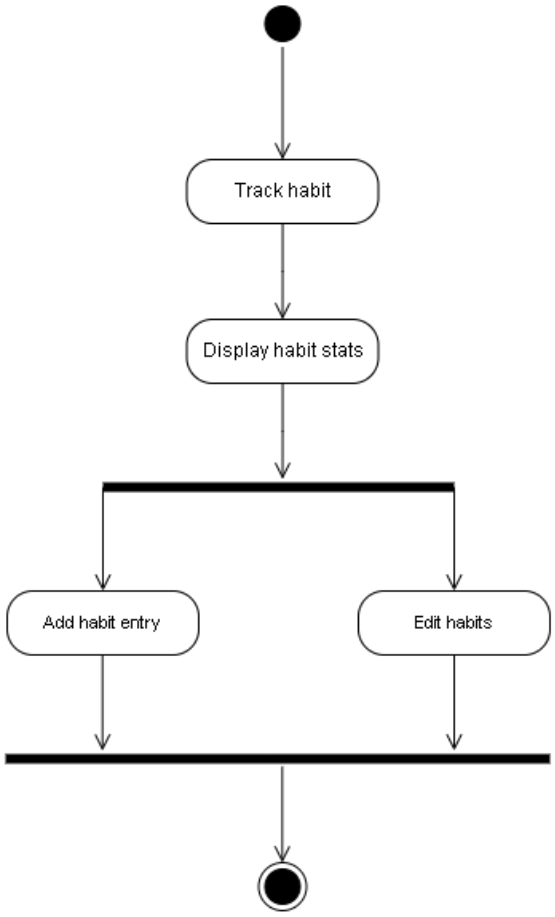

Figure 6: Activity Diagram of Habit Tracker

The app's habit tracker allows the student to first view their habit statistics. Following this, they have the option to either add or edit habits they wish to track. The student can then input their habit entry through the assistant.

B..4 Learn Well-Being Strategies

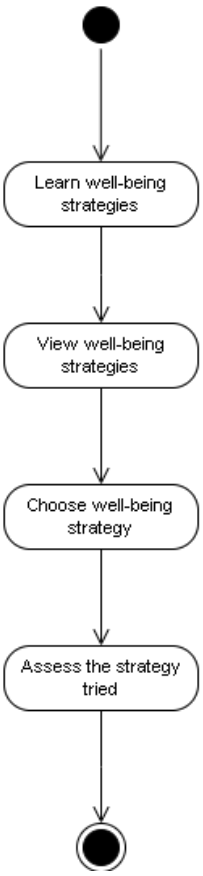

Figure 7: Activity Diagram of Learning Well-Being Strategies

The app provides the student with access to well-being strategies. Students can view well-being strategies and choose a strategy they would like to develop. Once they have tried a well-being strategy, they can assess it.

B..5 Diary

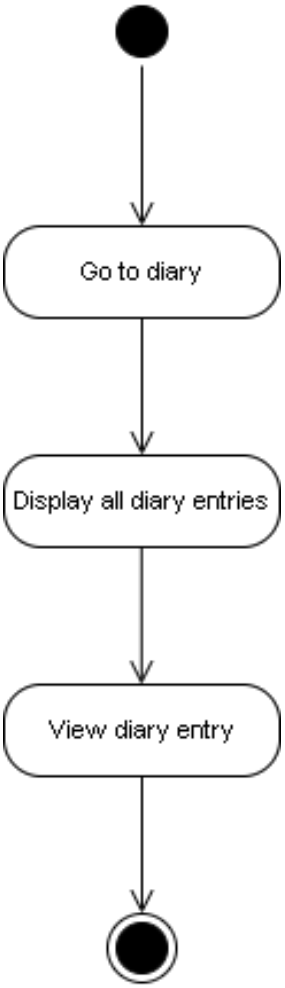

Figure 8: Activity Diagram of Diary

Through the diary feature, the app enables students to reflect on their moods, habits, coping strategies, and articles they have read. First, it will present a collection of diary entries. Then, the user can choose a diary entry to view.

B..6 Access to Online Resources

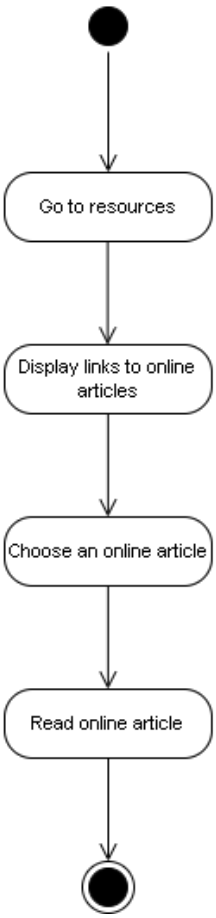

Figure 9: Activity Diagram of Access to Online Resources

The app provides students with access to online articles to learn more about mental health. Students can view the list of online articles and can choose one to read.

B..7 Talk to the Assistant

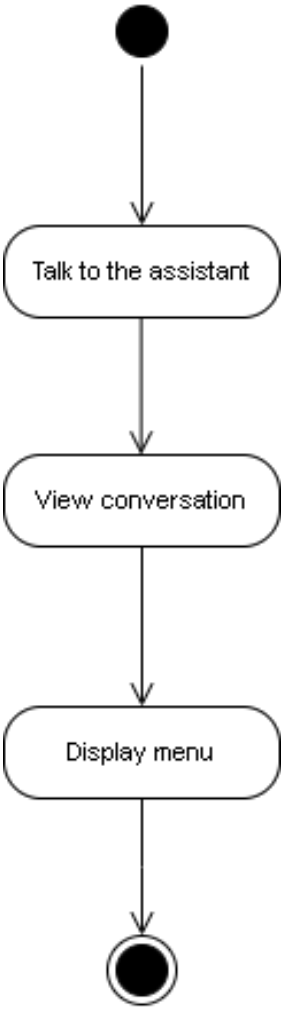

Figure 10: Activity Diagram of Talk to the Assistant

To provide an interactive experience, a menu-based assistant feature was added.

B..8 Account Settings

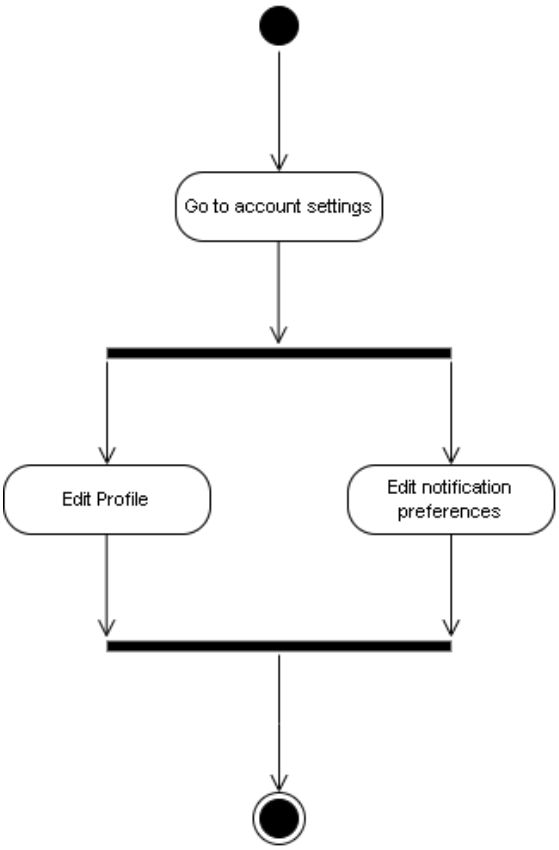

Figure 11: Activity Diagram of Account Settings

The user can edit his or her profile through the account settings. The user can also edit his or her notification preferences.

C. Data Dictionary

C..1 Student User

Attribute	Data Type	Description
user_id	int(10)	unique id of the user
email	varchar(30)	unique email of the user
password	varchar(30)	password of the user
first_name	varchar(30)	first name of the user
last_name	varchar(30)	last name of the user
birthday	varchar(30)	birthday of the user
contact_number	varchar(30)	contact number of the user
tracker_time_prompt	datetime()	time of the tracker prompt
insights_time_prompt	datetime()	time of the insights prompt
profile_image	varchar(100)	password of the user

Table 1: Data Dictionary Table for Student Users

C..2 Mood Entry

Attribute	Data Type	Description
mood_entry_id	int(10)	unique id of the mood entry
user_id	int(10)	unique id of the user
mood	varchar(10)	mood of the user
mood_trigger	varchar(30)	cause of mood of the user
date	datetime()	date of the mood recorded

Table 2: Data Dictionary Table for Mood Entries

C..3 Habit

Attribute	Data Type	Description
habit_id	int(10)	unique id of the habit
habit_name	varchar(30)	name of the habit
users	ManyToManyField	users who want to track the habit

Table 3: Data Dictionary Table for Habit

C..4 Habit Entry

Attribute	Data Type	Description
habit_entry_id	int(10)	unique id of the habit entry
user_id	int(10)	unique id of the user
habit_id	int(10)	unique id of the habit
data	varchar(30)	data of the habit recorded
date	datetime()	date of the habit recorded

Table 4: Data Dictionary Table for Habit Entries

C..5 Strategy

Attribute	Data Type	Description
strategy_id	int(10)	unique id of the strategy
strategy_name	varchar(30)	name of the strategy
description	varchar(300)	description of the strategy
source	varchar(50)	source of the strategy

Table 5: Data Dictionary Table for Strategy

C..6 Strategy Tried

Attribute	Data Type	Description
strategy_tried_id	int(10)	unique id of the strategy tried
user_id	int(10)	unique id of the user
strategy_id	int(10)	unique id of the strategy
is_effective	bool()	true if the strategy is effective, false otherwise
is_liked	bool()	true if the user likes the strategy, false otherwise
date	datetime()	date of the strategy tried

Table 6: Data Dictionary Table for Strategy Tried

C..7 Diary

Attribute	Data Type	Description
diary_id	int(10)	unique id of the diary
user_id	int(10)	unique id of the user
thoughts	varchar(300)	thoughts of the user
date	datetime()	date of the diary entry recorded

Table 7: Data Dictionary Table for Diary

C..8 Article

Attribute	Data Type	Description
article_id	int(10)	unique id of the article
title	varchar(30)	title of the article
summary	varchar(300)	summary of the article
source	varchar(50)	source of the article

Table 8: Data Dictionary Table for Strategy

C..9 Article Read

Attribute	Data Type	Description
article_read_id	int(10)	unique id of the article read
user_id	int(10)	unique id of the user
article_id	int(10)	unique id of the article
date	datetime()	date of the article read

Table 9: Data Dictionary Table for Article Read

D. Entity Relationship Diagram

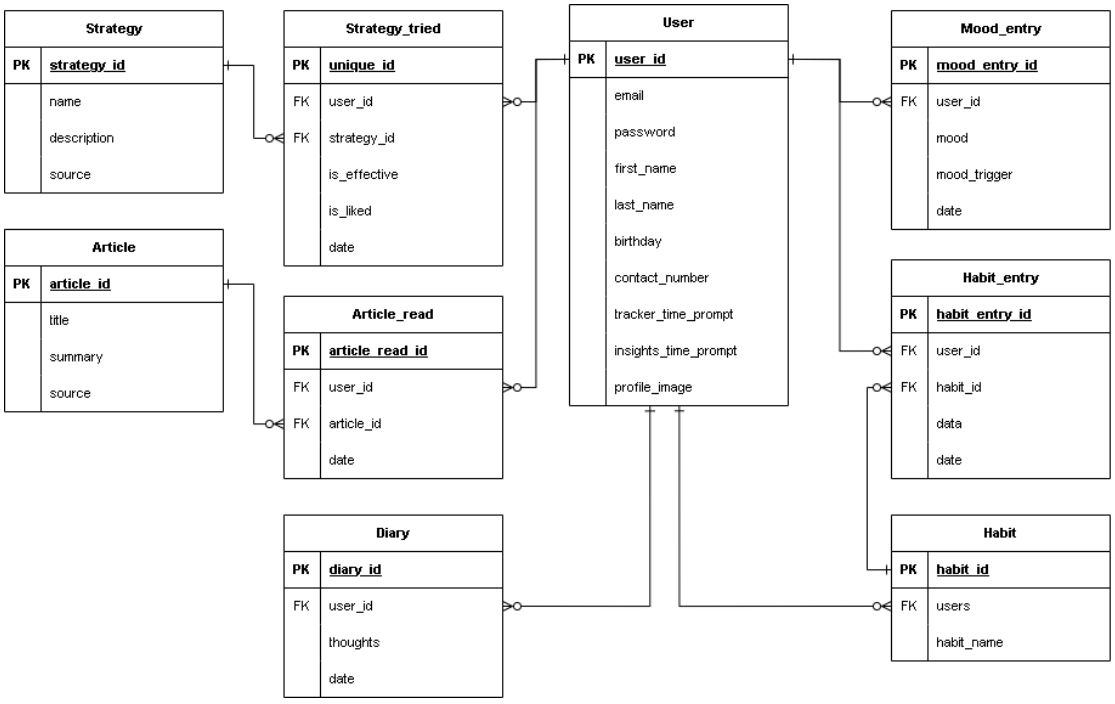

Figure 12: Entity Relationship Diagram

Figure 8 shows the Entity Relationship Diagram for the database design of the system. The system will be database-dependent since it intends to keep record of the students moods, habits, diaries, well-being strategies, and online articles.

E. Contents of the App

The data to be used for well-being strategies, recommended habits and online articles will be manually searched and compiled.

F. System Architecture Diagram

The user interacts with the user interface to access features of the app and send a message to the assistant. The user message is forwarded from the app's backend to Rasa Open Source for processing. Rasa Open Source analyzes the user message and determines the most suitable action or response. In cases where the action involves interactions with external systems or databases, Rasa has an agent that interacts with the action server to execute the required tasks. Upon generating a response, the Rasa agent returns it to the web app. The user interface renders the assistant's response and presents it to the user.

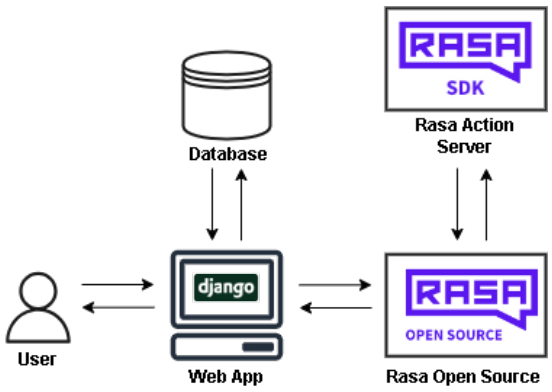

Figure 13: System Architecture Diagram

G. Technical Architecture

The minimum system requirements for the Django app are the following:

1. Django
2. At least 1GB disk space
3. 4 GB RAM or higher

4. Intel Core i3 processor and above

5. Google Chrome, Mozilla Firefox, Microsoft Edge, or any other browser

The Rasa server requires the following:

1. PostgreSQL

2. Python 3.10

3. Intel Core i3 processor and above

4. At least 8 GB RAM

5. At least 100 GB free disk space

V. Results

A. Sign Up Page

To utilize the features of ACAPP, student users are required to create an account. This can be done by visiting the app's sign-up page, where student users will be prompted to provide their email and create a password for their account.

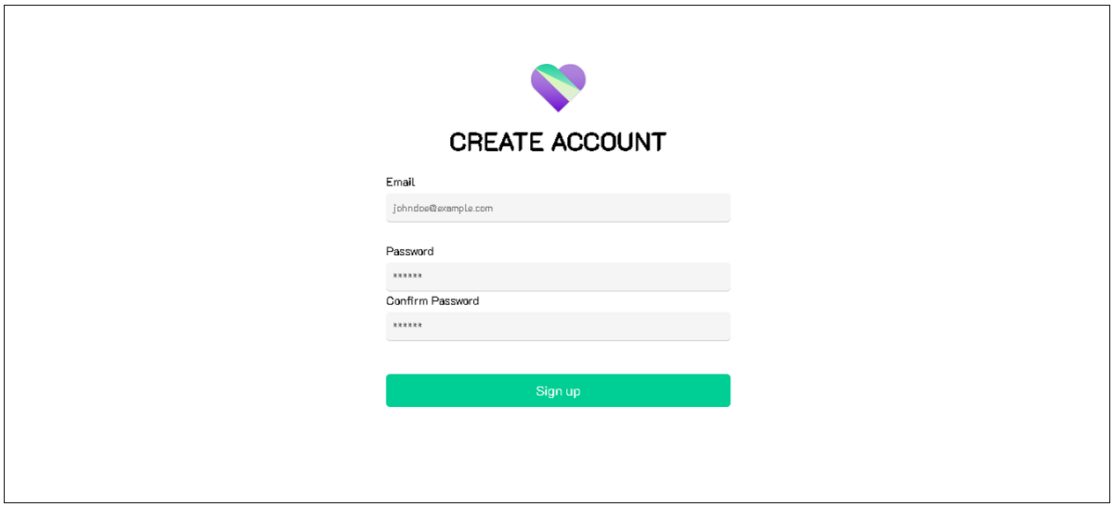

Figure 14: Sign Up Page

B. Chat Page

After successfully creating an account, the menu-based assistant within ACAPP will send a warm welcome message to the user. It will introduce the purpose and functions of the app, ensuring that the user is well-informed. Additionally, it will inquire about the user's notification preferences, allowing them to customize their experience according to their needs and preferences.

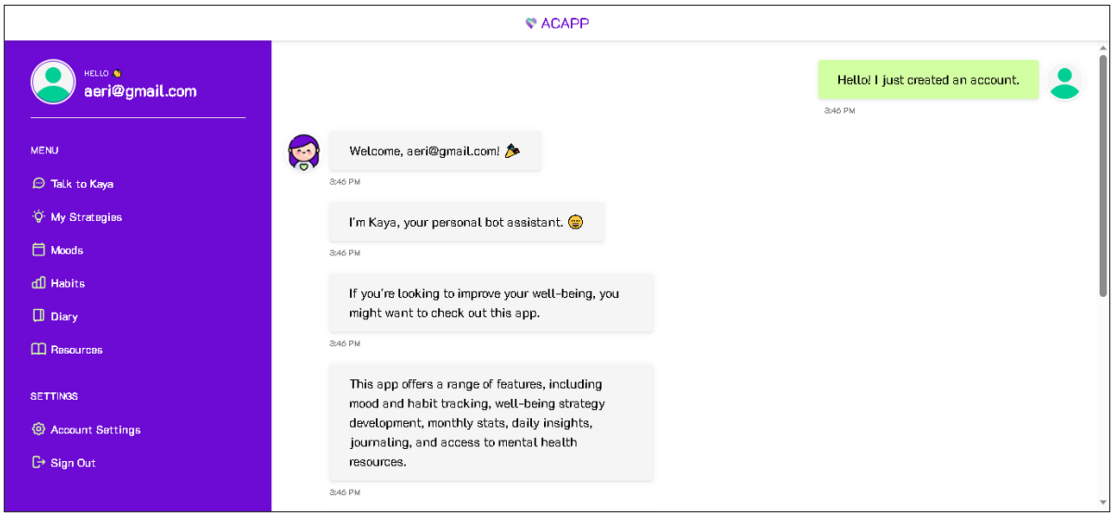

Figure 15: Chat Page – Welcome Message

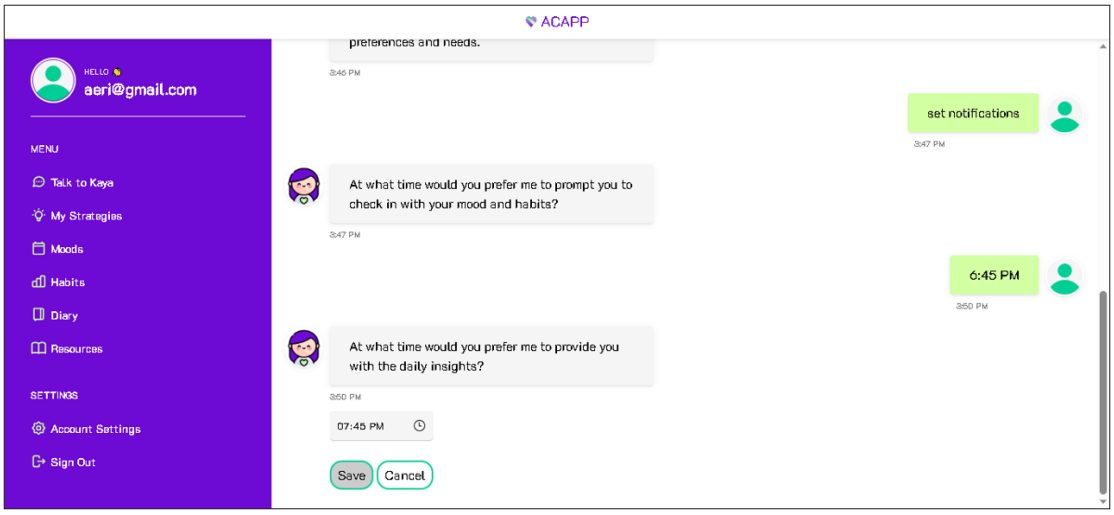

Figure 16: Chat Page – Set Notifications

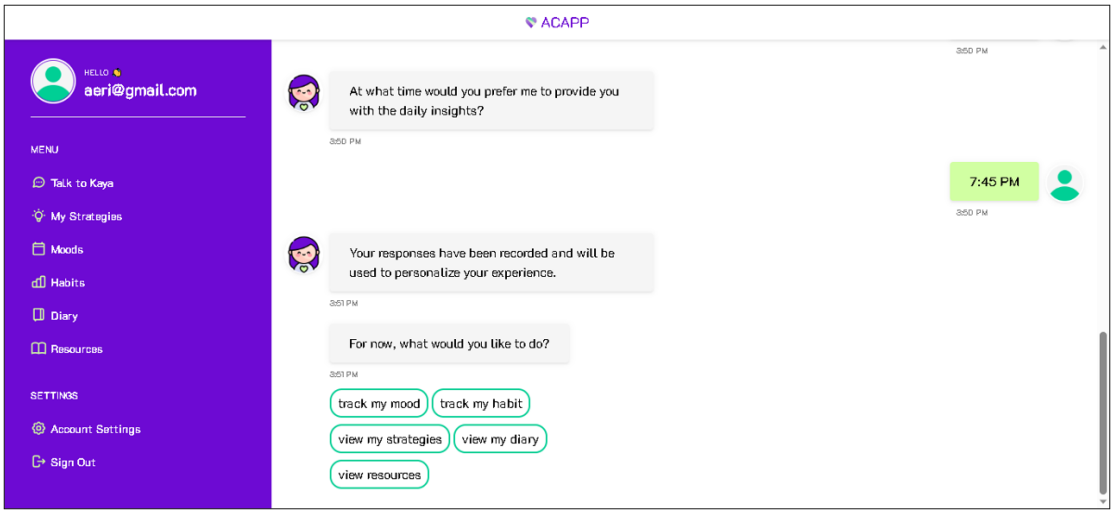

Figure 17: Chat Page – Main Menu

C. Talk to Kaya

The home page of the app is dedicated to the "Talk to Kaya" feature. It serves as the primary interface for student users to interact with the menu-based assistant named "Kaya". To initiate a conversation, student users simply need to click the "Start a Conversation" button. This action will redirect them to the chat page, where they can engage in a dialogue with the assistant.

The student user sidebar in ACAPP displays the profile image and email of the user (showing the email if the name has not been set yet). It provides a convenient navigation tool, allowing users to easily access different features within the app. The main features of the app are divided into six sections: Talk to Kaya, My Strategies, Moods, Habits, Diary, and Resources. Additionally, the sidebar allows users to navigate to the Account Settings page and the Sign Out page.

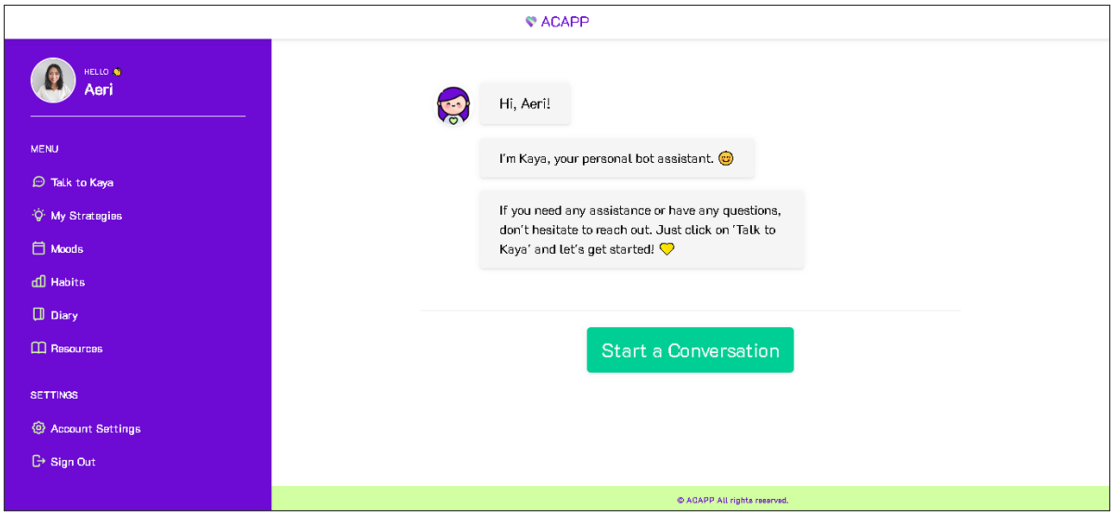

Figure 18: Home Page

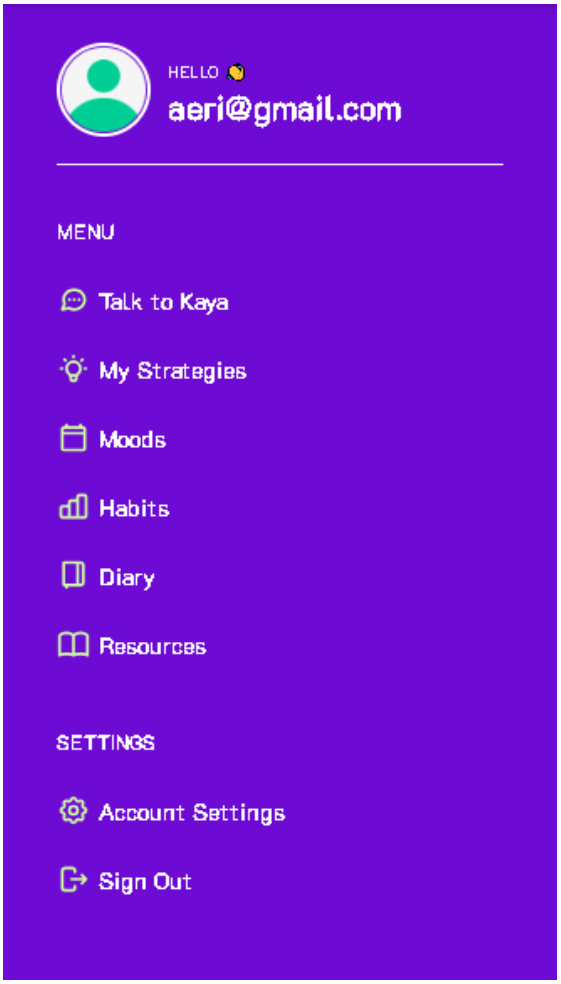

Figure 19: Student User Sidebar

D. My Strategies

When the student user clicks on "My Strategies" in the sidebar, they will be redirected to the My Strategies page. Alternatively, they can choose the "view my strategies" option within the assistant's conversation. On this page, the student user can view all the strategies they have tried, liked, and marked as effective. This provides a convenient overview of the strategies they have engaged with and found beneficial. Additionally, the student user can click on the "learn well-being strategies" button to engage in a conversation with the assistant and learn more well-being strategies.

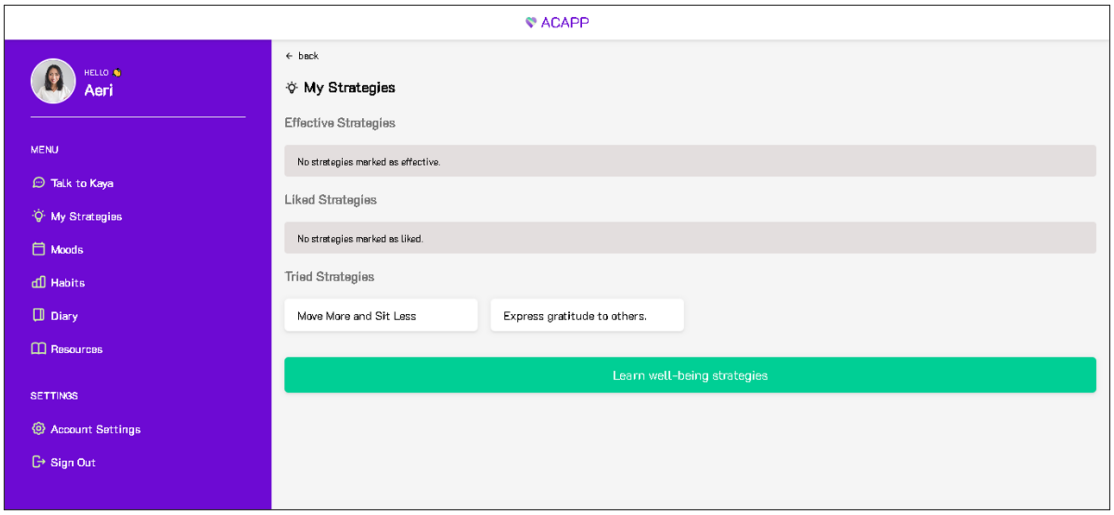

Figure 20: My Strategies Page

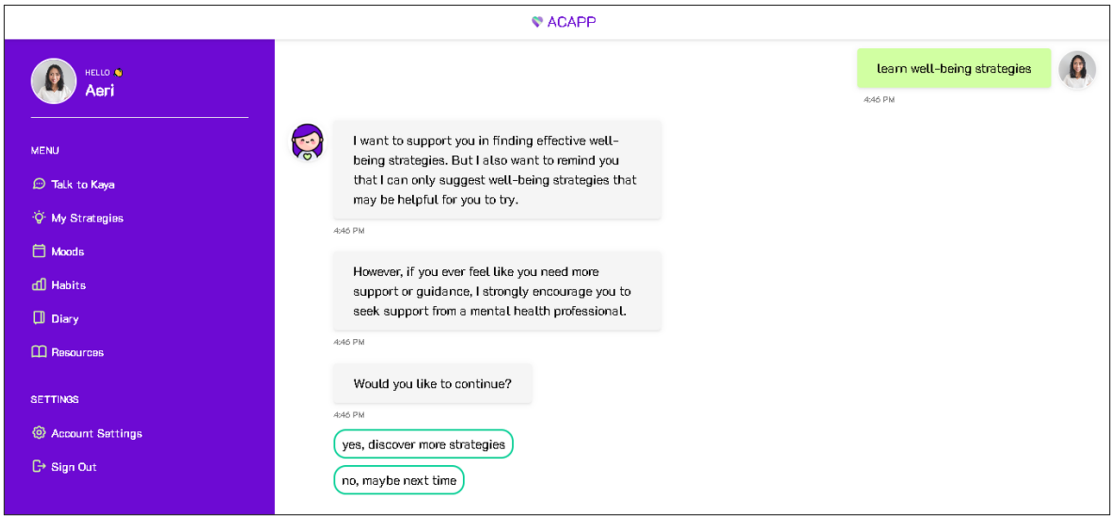

Figure 21: Chat Page - Learn Well-Being Strategies

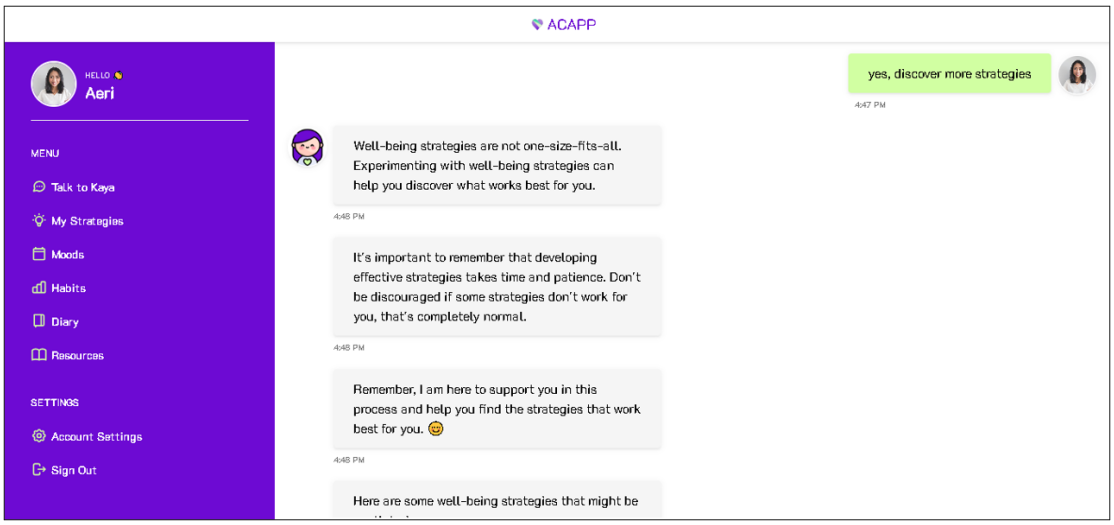

Figure 22: Chat Page - Discover More Strategies

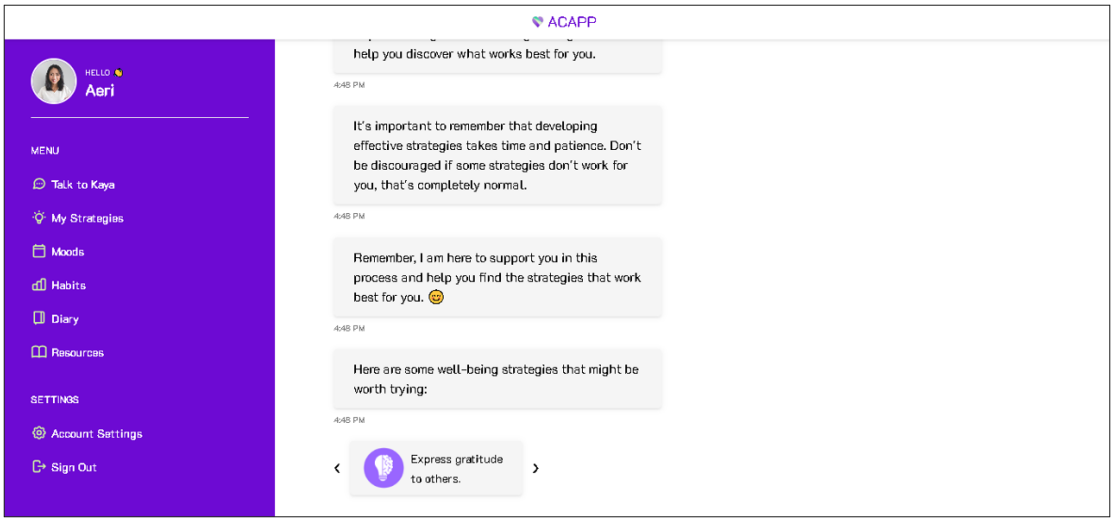

Figure 23: Chat Page - Discover More Strategies

E. View Strategy Page

After selecting a strategy suggested by the assistant, the student user will be directed to the View Strategy page. Here, they can find a detailed description of the chosen strategy along with references or sources.

To track their progress, the user must first click the "Mark as Tried" button to indicate they have attempted the strategy. Once marked as tried, they can then click the "Mark as Liked" or "Mark as Effective" buttons to express their feedback on the strategy's impact.

Clicking the "back" button will redirect the user back to the chat page.

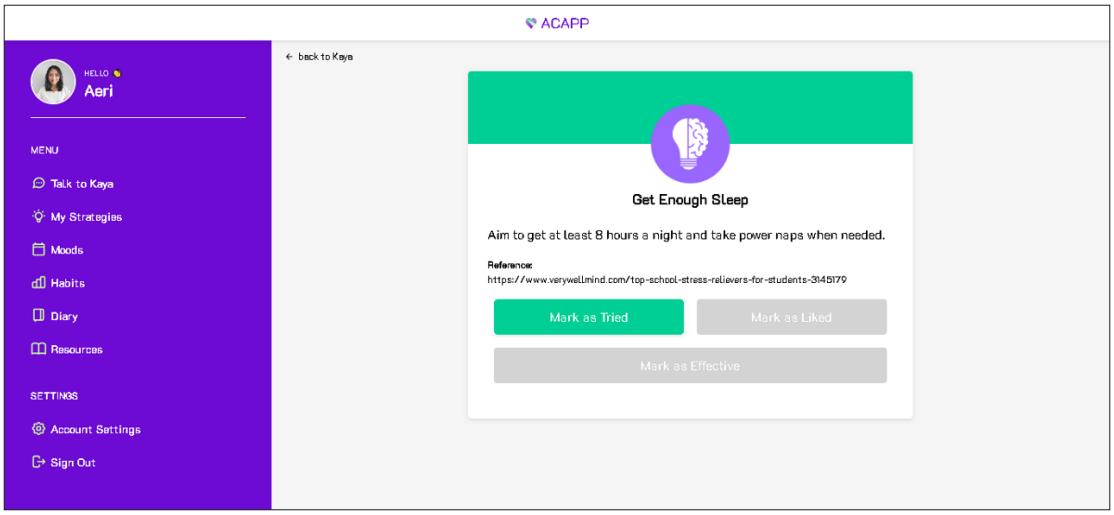

Figure 24: View Strategy Page

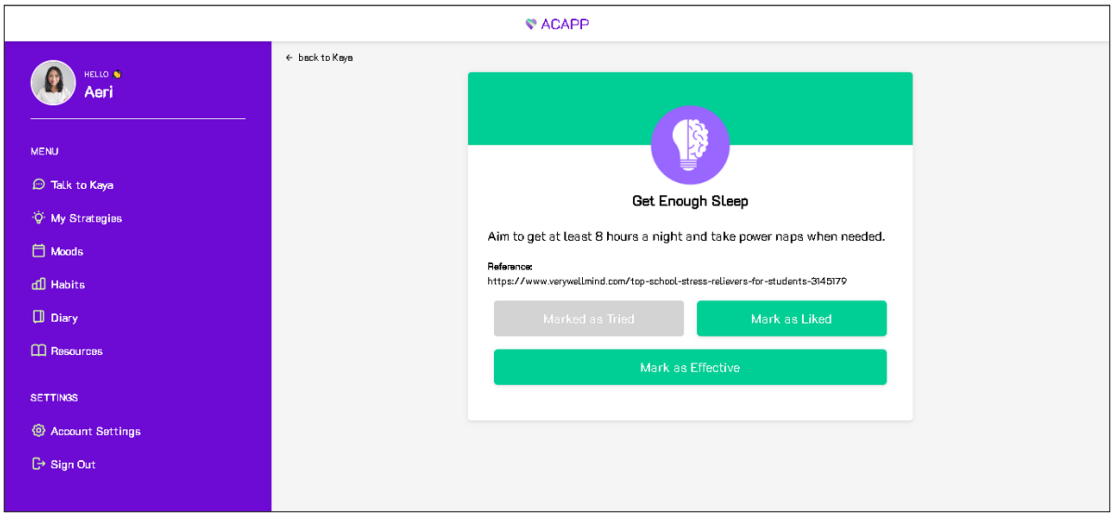

Figure 25: View Strategy Page - Marked as Tried

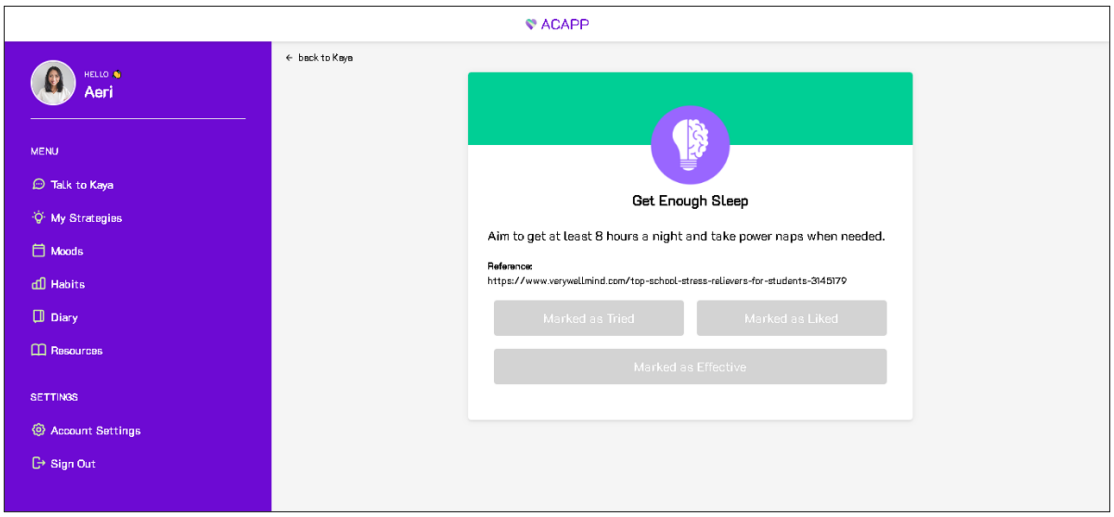

Figure 26: View Strategy Page - Marked as Effective

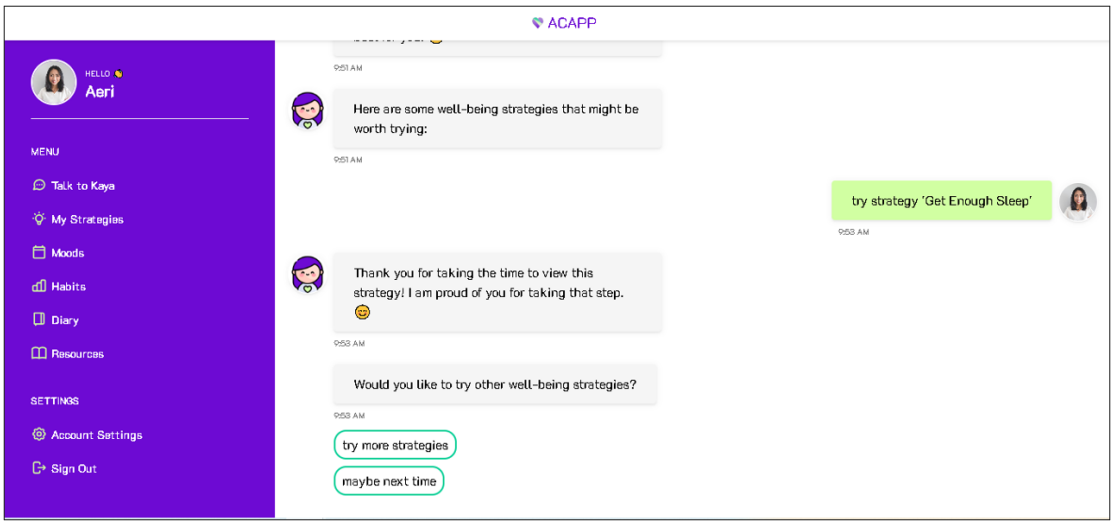

Figure 27: Chat Page - Back to Kaya

F. Moods

When the student user clicks on "Moods" in the sidebar or selects the "track my mood" option within the assistant's conversation, they will be redirected to the mood tracker page. This page allows students to monitor their moods over the past three months or 90 days.

To add a mood entry, the student user can click the "add mood" button. This action will redirect them to the chat page, where the assistant will guide them through capturing and logging their current mood. The user can provide details such as their mood and the trigger or reason for the mood.

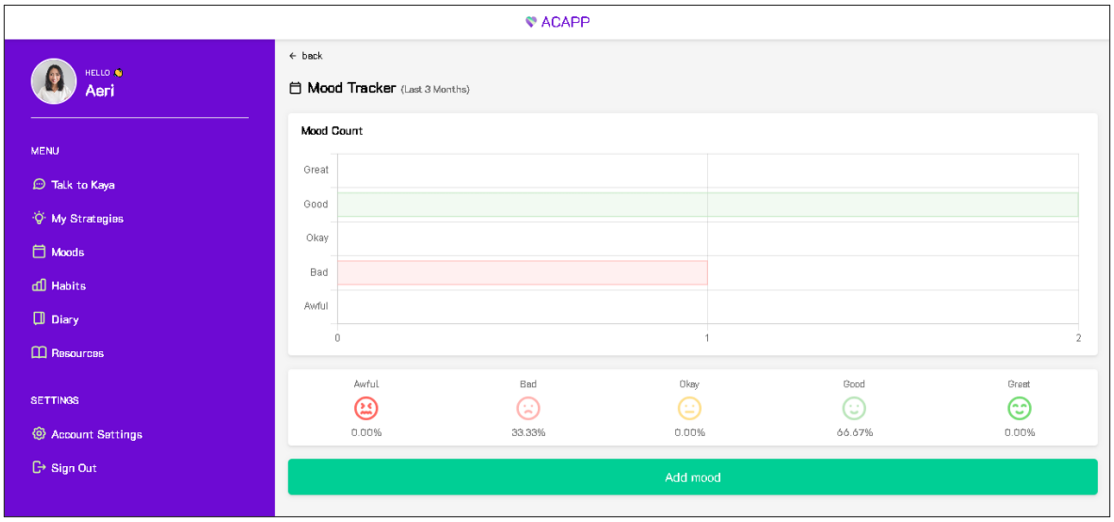

Figure 28: Mood Tracker Page

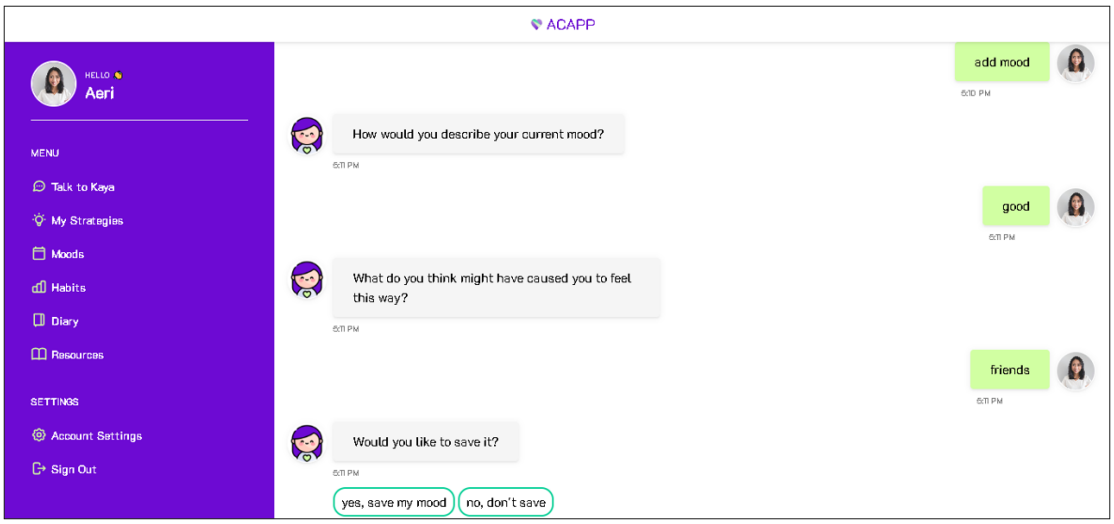

Figure 29: Chat Page – Add Mood

G. Habits

When the student user clicks on "Habits" in the sidebar or selects the "track my habit" option within the assistant's conversation, they will be redirected to the habit tracker page. By clicking the "add entry" button, users can record new habit entries. Additionally, users can customize the habits they want to track by clicking the "edit habits" button, allowing them to manage and adjust their habit tracking preferences.

To visually represent the habit status, the app utilizes color coding. The red bar indicates a bad or low habit, while the green bar signifies good or recommended habits. On this page, students can also track their habits over the past three months or 90 days.

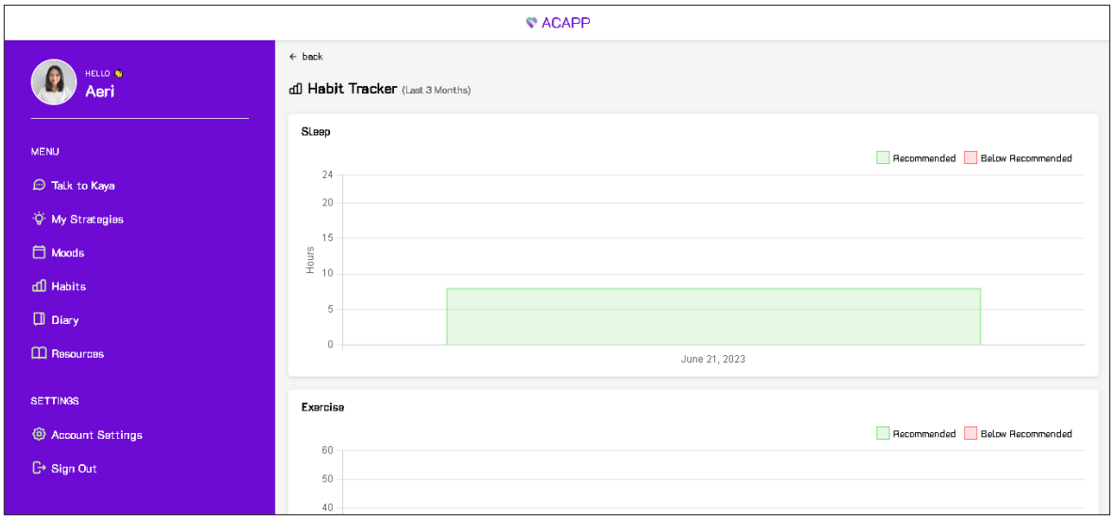

Figure 30: Habit Tracker Page

Figure 31: Habit Tracker Page

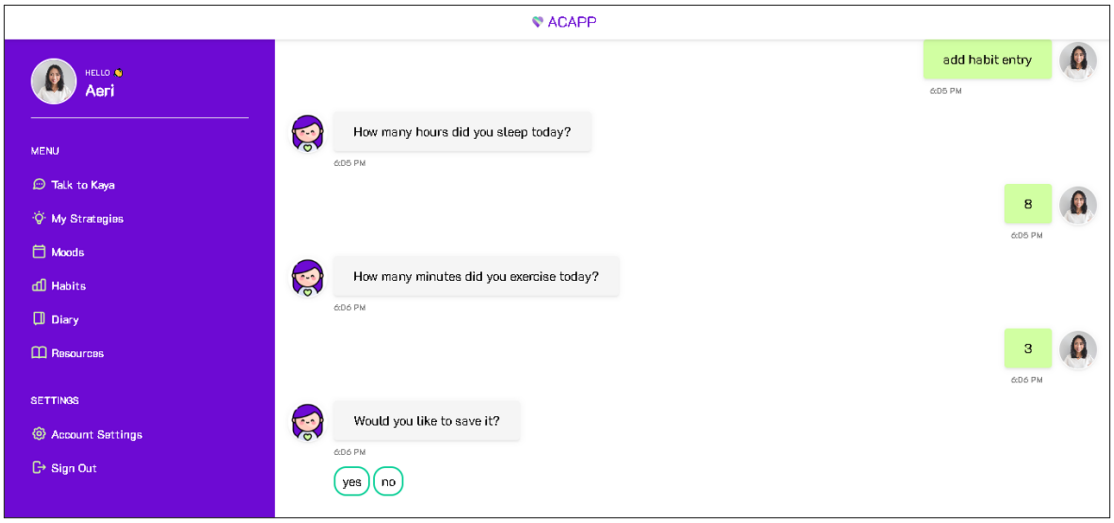

Figure 32: Chat Page – Add Habit Entry

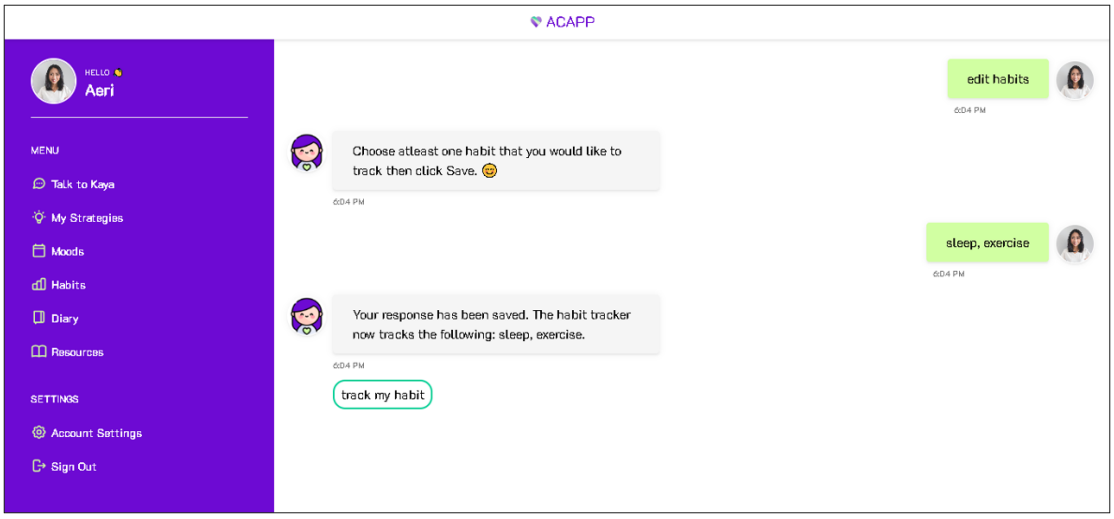

Figure 33: Chat Page – Edit Habits

H. Diary

Student users can access the diary by clicking on "Diary" in the sidebar or through the assistant by selecting the "view my diary" option. The Diary page showcases a chronological list of the user's diary entries per day, accompanied by recorded moods represented by emojis. By clicking on any specific entry, users will be directed to the corresponding Diary Entry page.

The Diary Entry page enables users to express their thoughts and reflections for a particular day. Users can freely write about their experiences, emotions, and any other relevant information by clicking the "edit diary" button. The availability of the "edit diary" button is limited to diary entries made on the current day. Additionally, on this page, users can view the moods they recorded for that day, track their habits, review the well-being strategies they tried, and access a record of the online articles they have read.

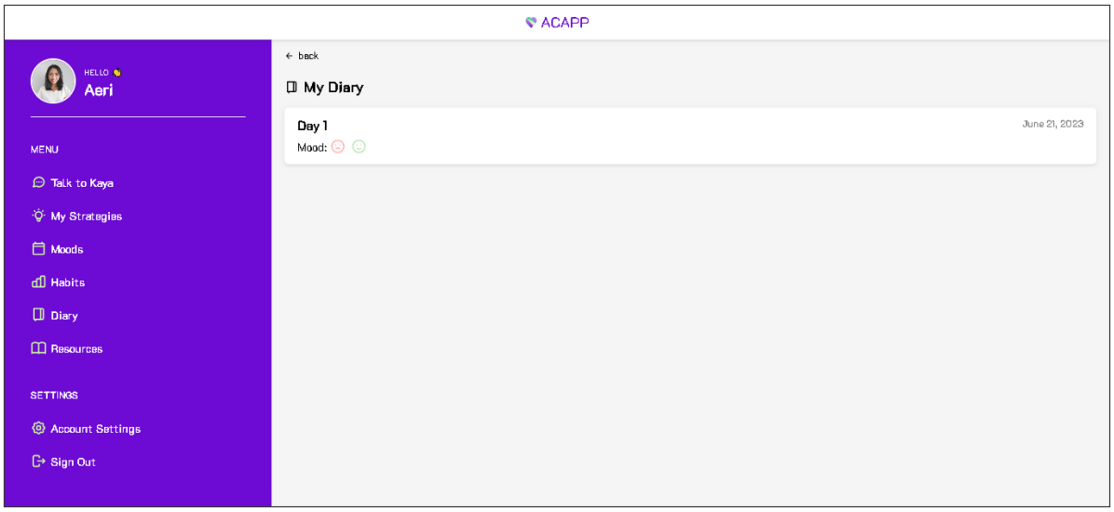

Figure 34: Diary Page

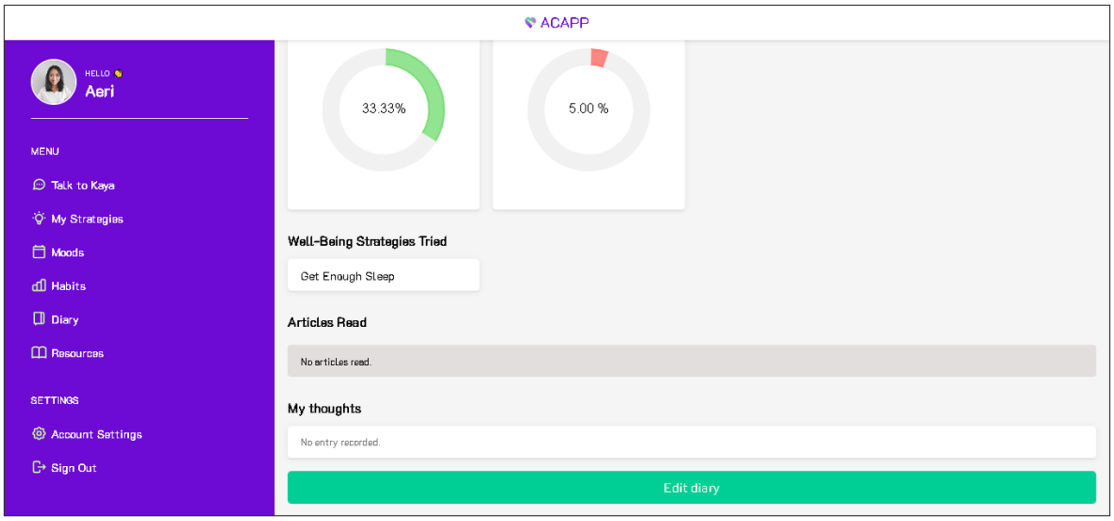

Figure 36: Diary Entry Page

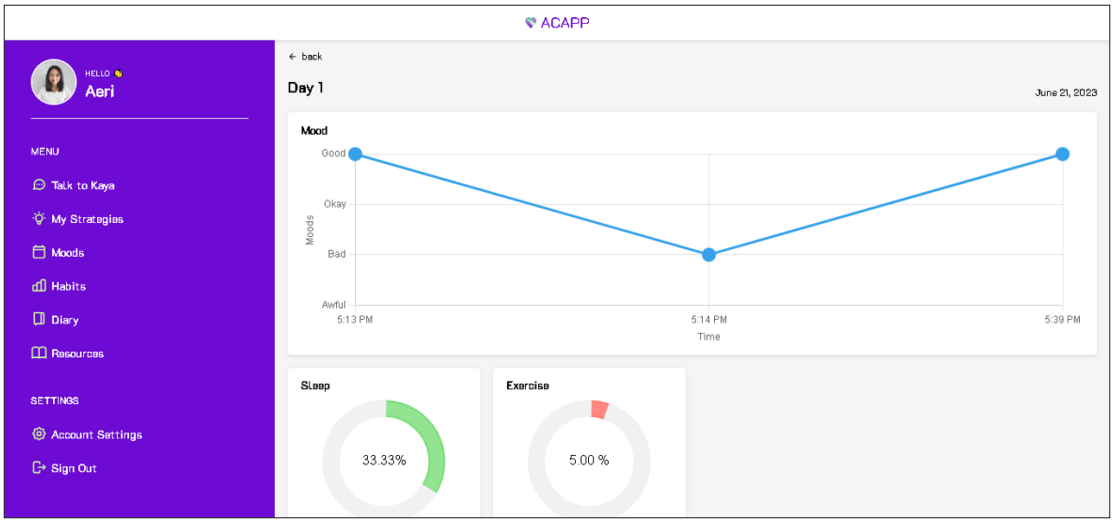

Figure 35: Diary Entry Page

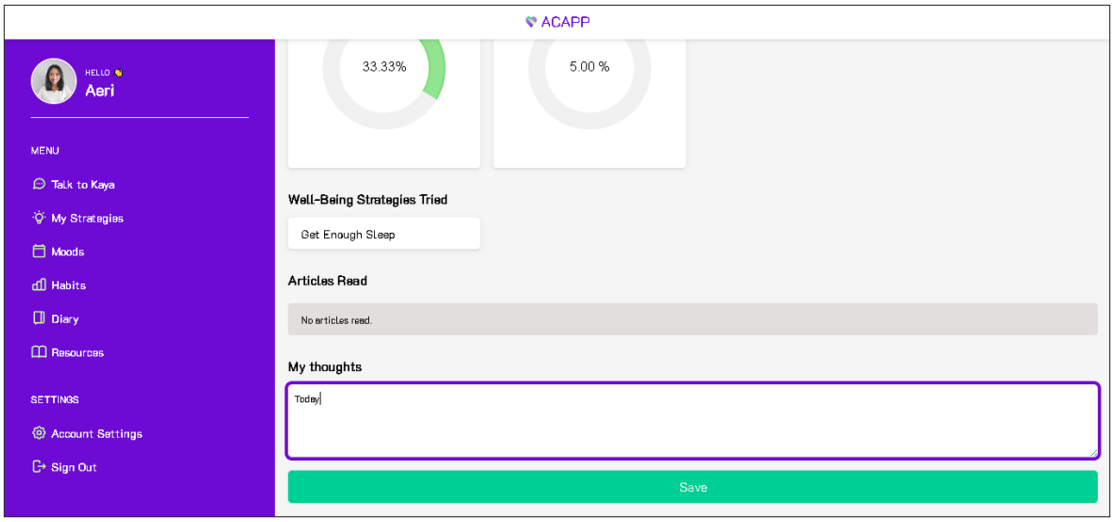

Figure 37: Diary Entry Page - Edit Diary

I. Resources

When the student user clicks on "Resources" in the sidebar or selects the "view resources" option within the assistant's conversation, they will be redirected to the resources page. On this page, the student user can view all the articles they have read, creating a convenient reference of their reading history.

To explore and read more articles, the user can click the "read articles" button. This action will redirect them to the chat page, where they can engage with the assistant. The assistant will then suggest online articles. The user can choose from the suggested articles and proceed to read them.

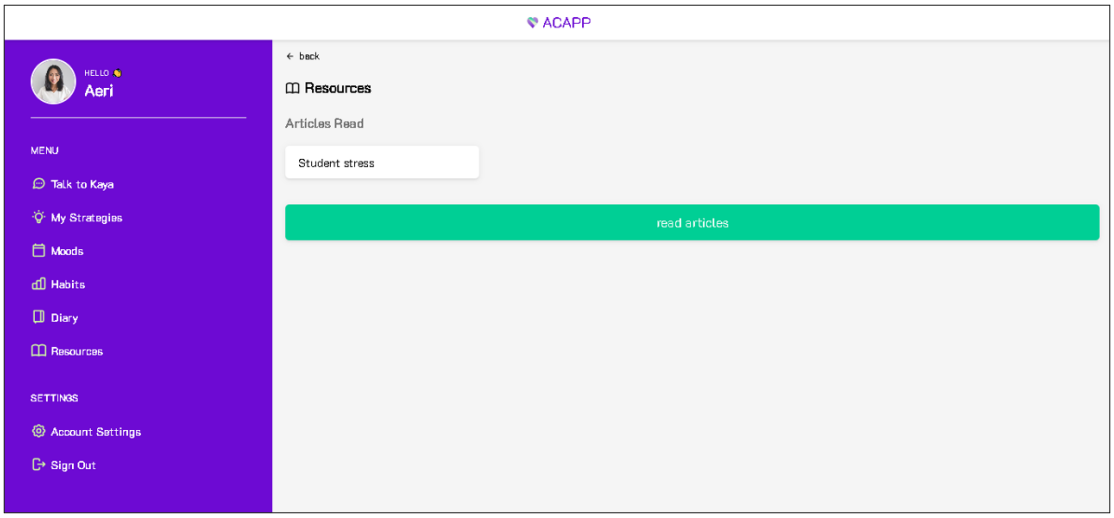

Figure 38: Resources Page

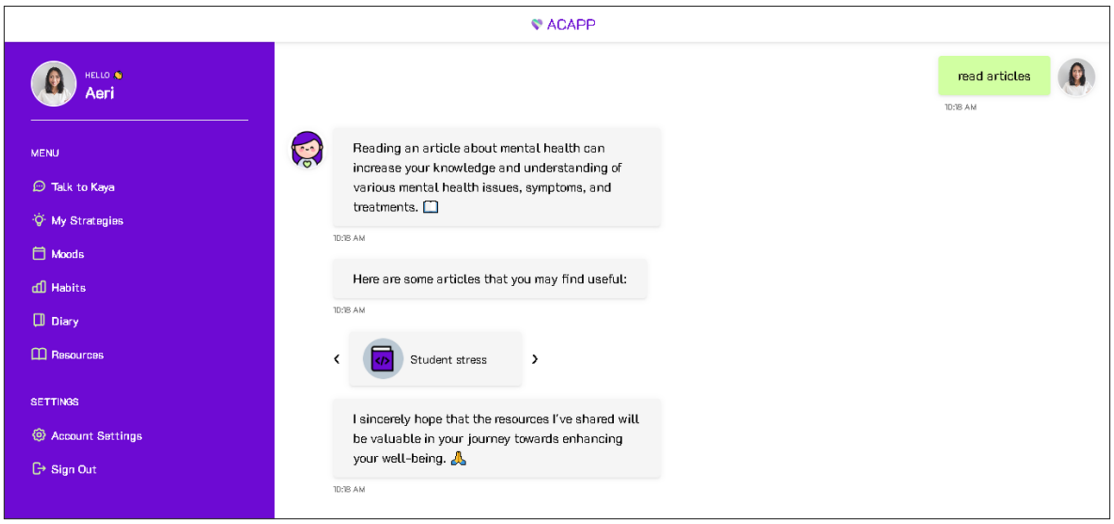

Figure 39: Chat Page - Read Articles

J. View Article Page

After selecting an article, the user will be redirected to the View Article page. On this page, the user can find detailed information about the online article, including the title, summary, and source.

To proceed with reading the article, the user can click the "Read the article" button. By clicking this button, the app automatically saves the user's reading history, so that they can easily track and revisit previously read articles at any time.

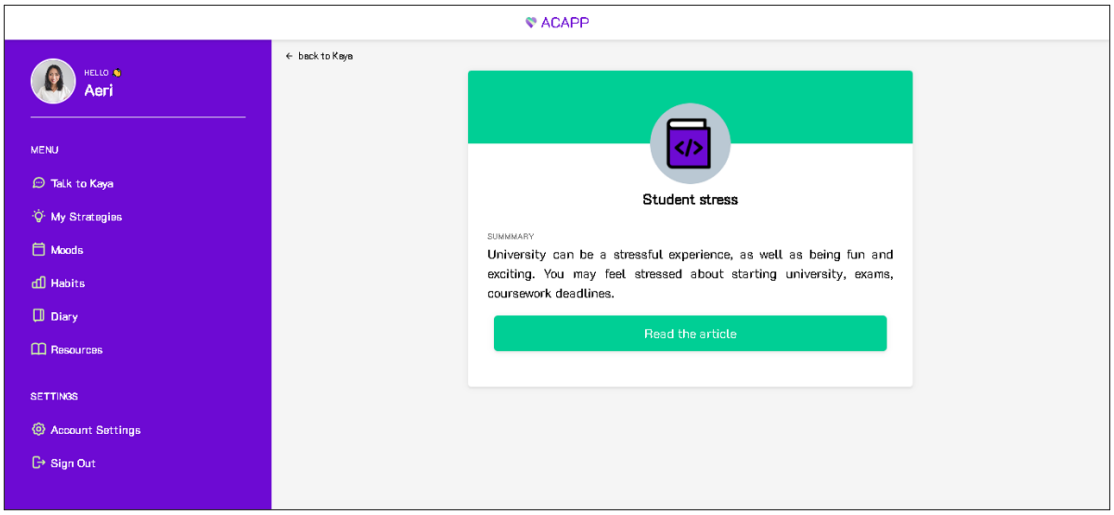

Figure 40: View Article Page

K. Account Settings

By clicking on "Account Settings" in the sidebar, users can access the Account Settings Page. On this page, users can view their profile details, including their name, contact information, and profile image. Additionally, users can review and modify their notification settings, which were initially set at the beginning. This page serves as a centralized location for users to manage their account preferences and ensure a personalized and tailored experience within the ACAPP application.

The Edit Profile Page allows users to modify their personal information, including their name, contact number, birthday, and profile image. To access this page, users can click on "Edit Profile" within the Account Settings Page.

The Notification Settings Page offers users the flexibility to customize the timing of prompts for mood and habit check-ins from the assistant. First, the user must click the "subscribe to push messaging" button to receive notifications. After that, users can adjust the specific times at which they receive reminders to track their moods and habits. Additionally, users can modify the time prompt for the delivery of daily insights.

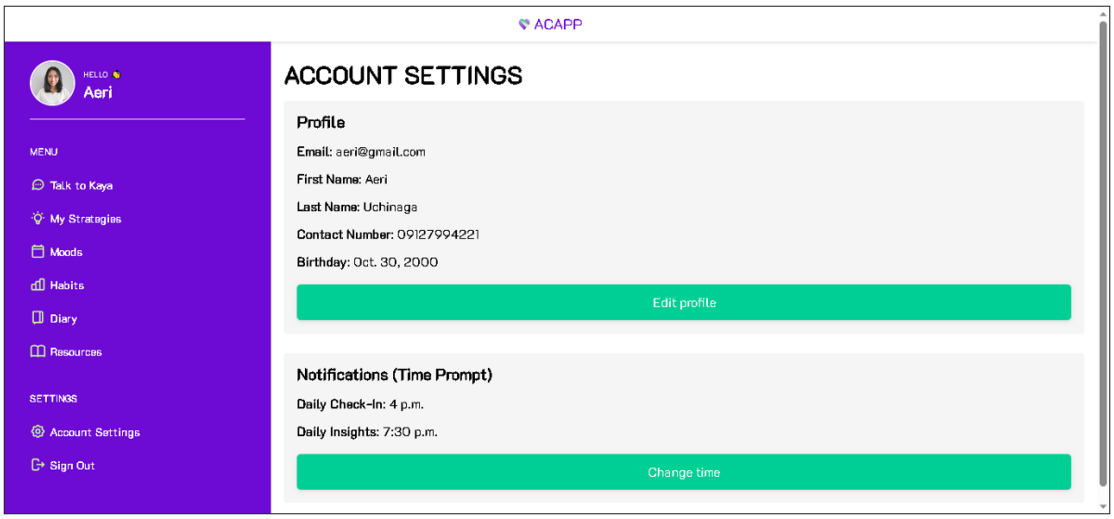

Figure 41: Account Settings Page

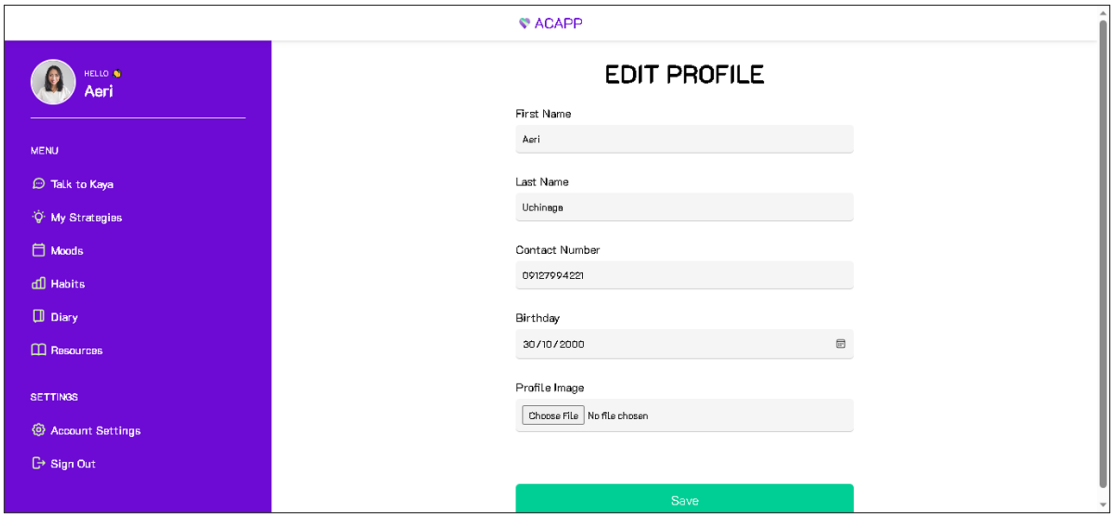

Figure 42: Edit Profile Page

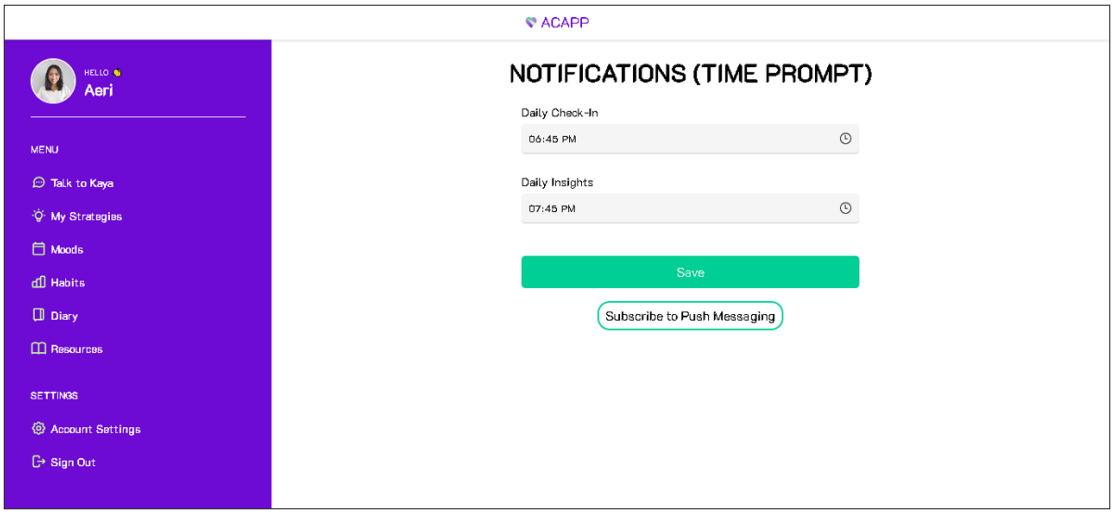

Figure 43: Notifications Settings Page

L. Notifications

After setting a check-in time, student users will receive notifications that will lead them to the mood tracker. Additionally, users have the option to set notifications for daily insights. The Daily Insights page can be accessed according to the user's chosen time for receiving these notifications. It serves as a dedicated diary entry page that becomes available only during the specified time set by the user.

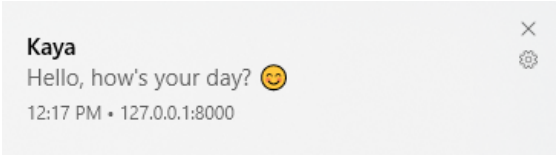

Figure 44: Notification - Check In

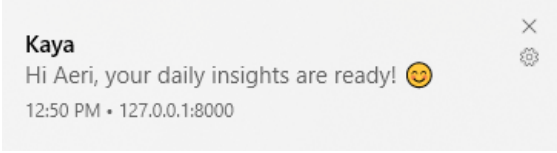

Figure 45: Notification - Daily Insights

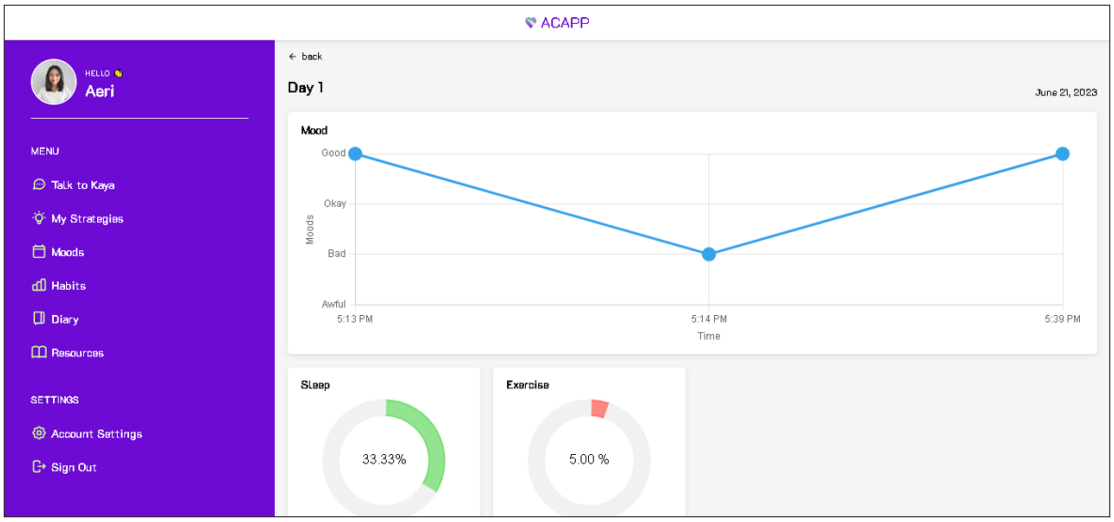

Figure 46: Daily Insights Page

M. Sign Out Page

Users have the option to sign out of the app whenever they choose not to use it. By accessing the Sign Out page, users can securely log out of their account, ensuring that their personal information and session details are protected.

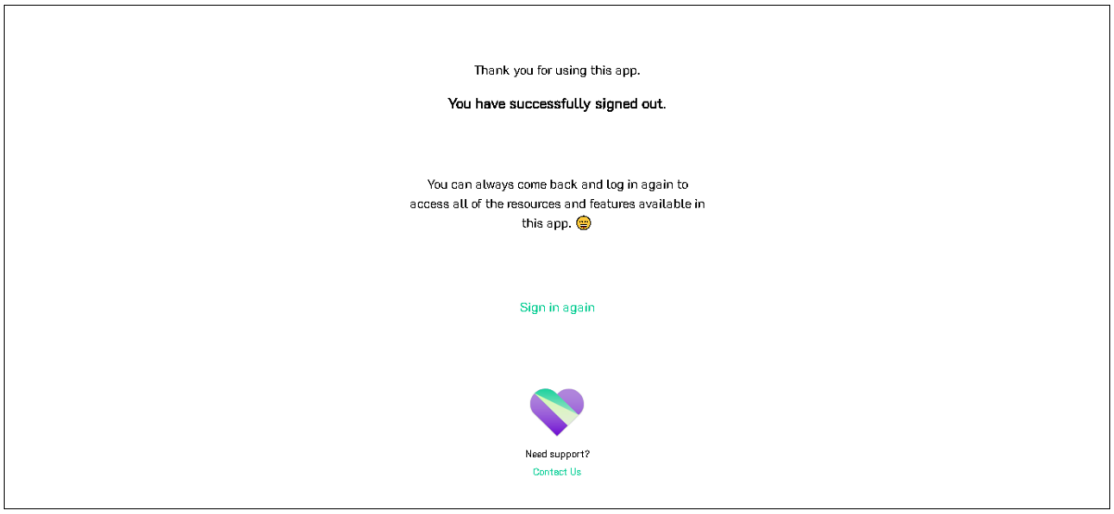

Figure 47: Sign Out Page

N. Sign In Page

To access the app, users can sign in through the Sign In page. Upon reaching this page, users will be prompted to enter their email and password, which are required for authentication.

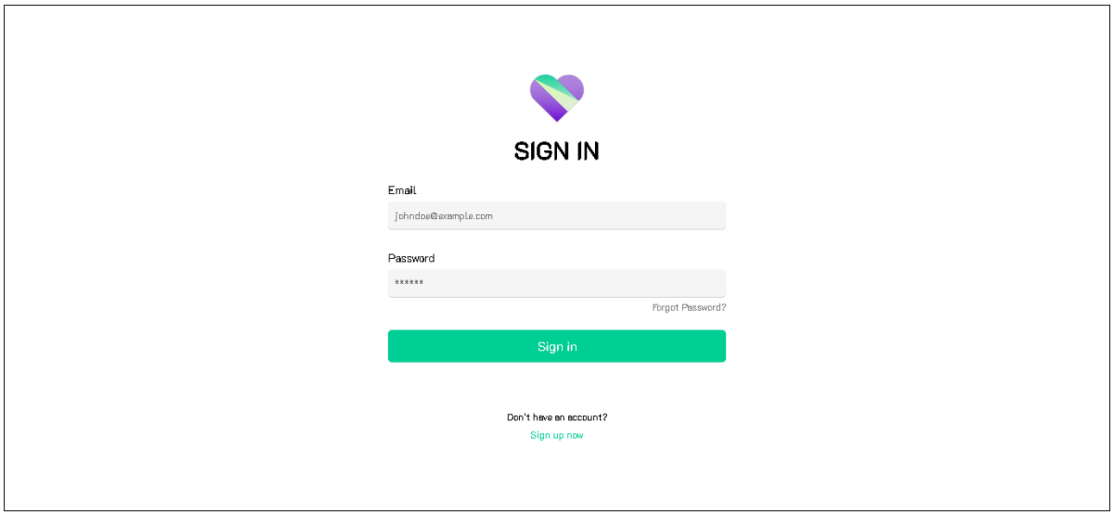

Figure 48: Sign In Page

O. Admin Dashboard

After signing in, the admin will be redirected to the admin dashboard page. On this page, the admin can see the number of well-being strategies and the number of articles available in the app. The admin's sidebar can be used to navigate to the dashboard, the list of well-being strategies, and the list of online articles.

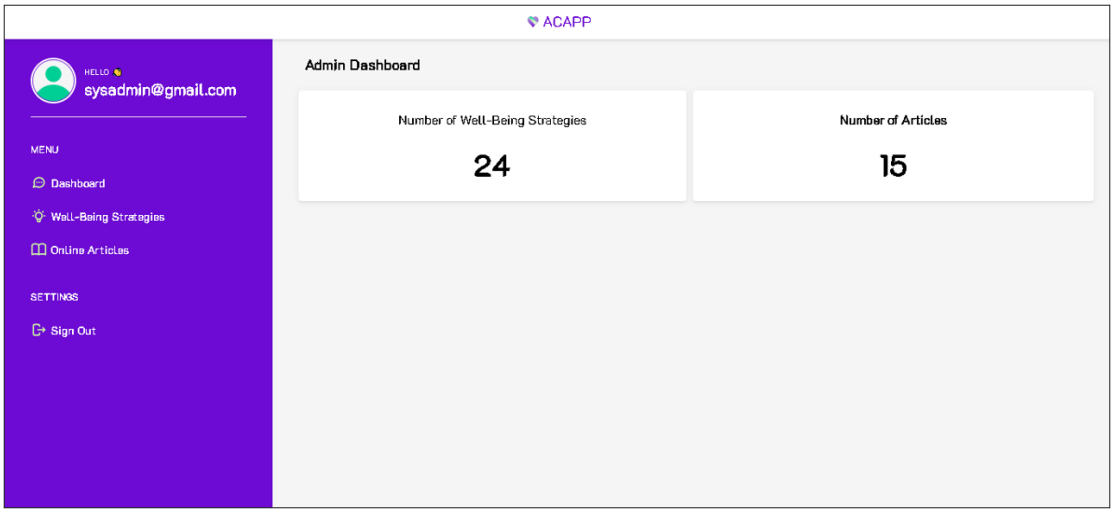

Figure 49: Admin Dashboard

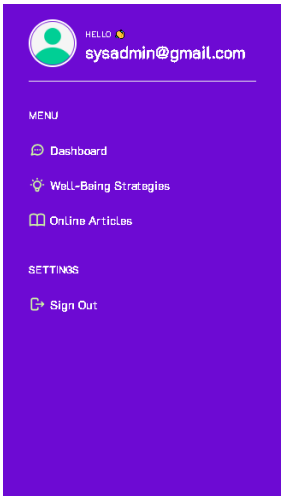

Figure 50: Admin Sidebar

P. Admin Add Well-Being Strategies

In the sidebar navigation, under "Well-Being Strategies," the admin user can access the list of available well-being strategies within the app as shown in figure 49. By clicking the "add strategy" button, the admin user can add a new strategy. Furthermore, clicking on a specific strategy will allow the user to view its details.

After clicking the "add strategy" button, the admin will be redirected to the Admin Add Strategy page. On this page, the admin is required to fill out the name, description, and source of the strategy. Once the admin has completed the required fields, they can click the "save" button to store the strategy in the database.

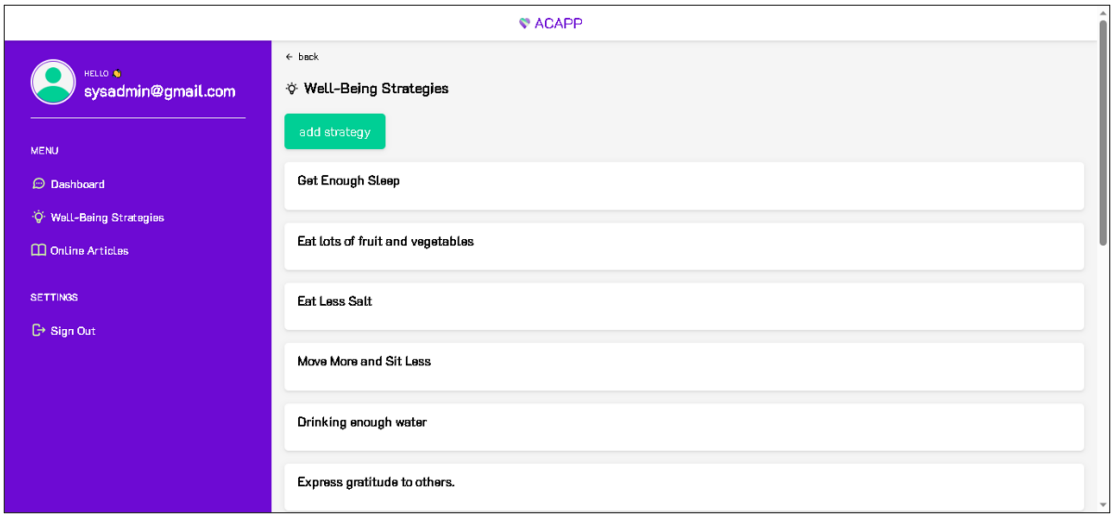

Figure 51: Admin Well-Being Strategies Page

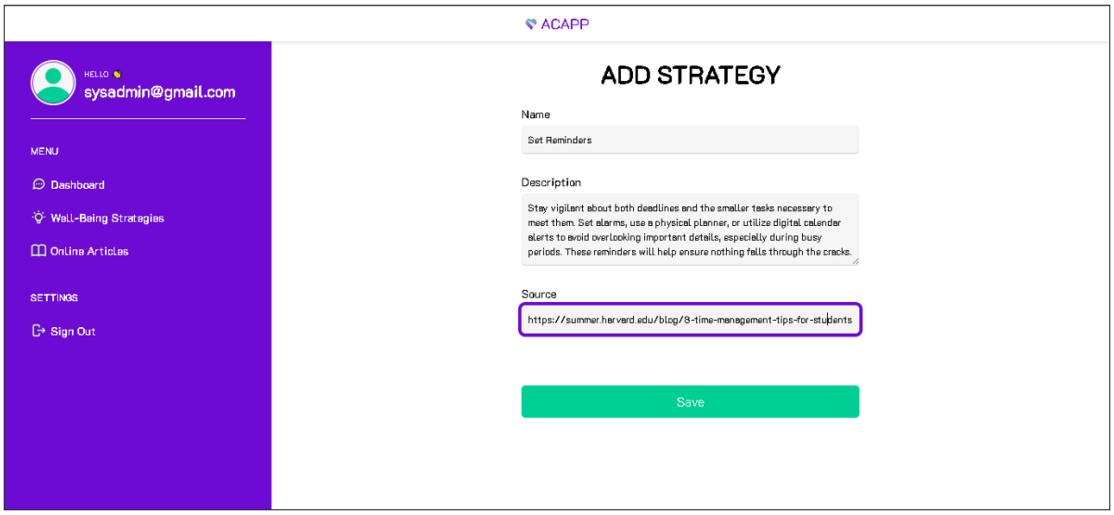

Figure 52: Admin Add Strategy Page

Q. Admin Edit Strategy

The admin can view the details of a strategy by clicking on it from the Admin Well-Being Strategies Page. Upon clicking the strategy, the admin will be directed to a page where they can view the strategy's details. On this page, the admin will also find an "edit strategy" button, which they can click to modify the strategy.

Once the "edit strategy" button is clicked, the admin will be directed to the Admin Edit Strategy page. On this page, the admin can modify the name, description, and source of the strategy. To delete the strategy, the admin can click

the "delete" button, while clicking the "save" button will save any changes made to the strategy.

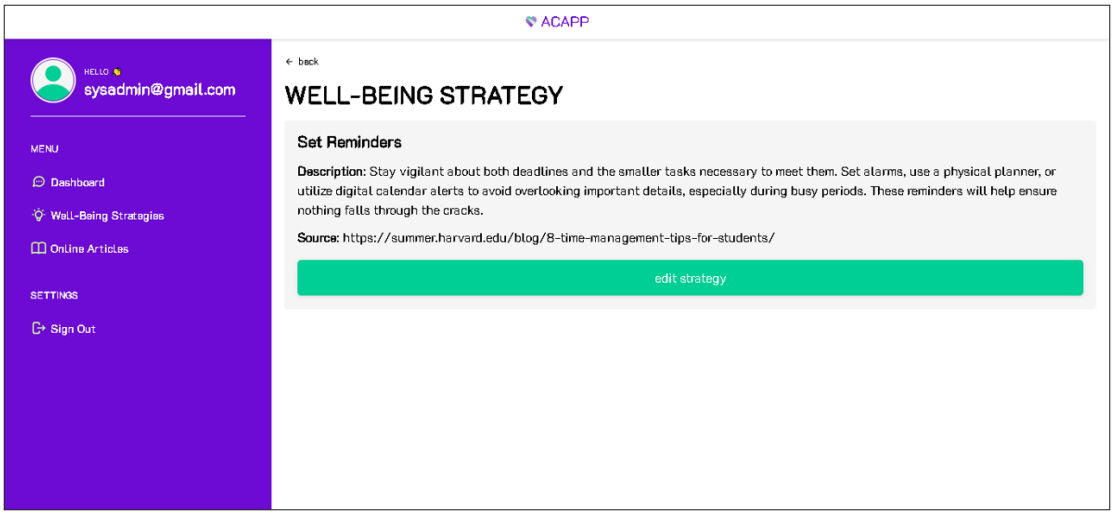

Figure 53: Admin View Strategy Page

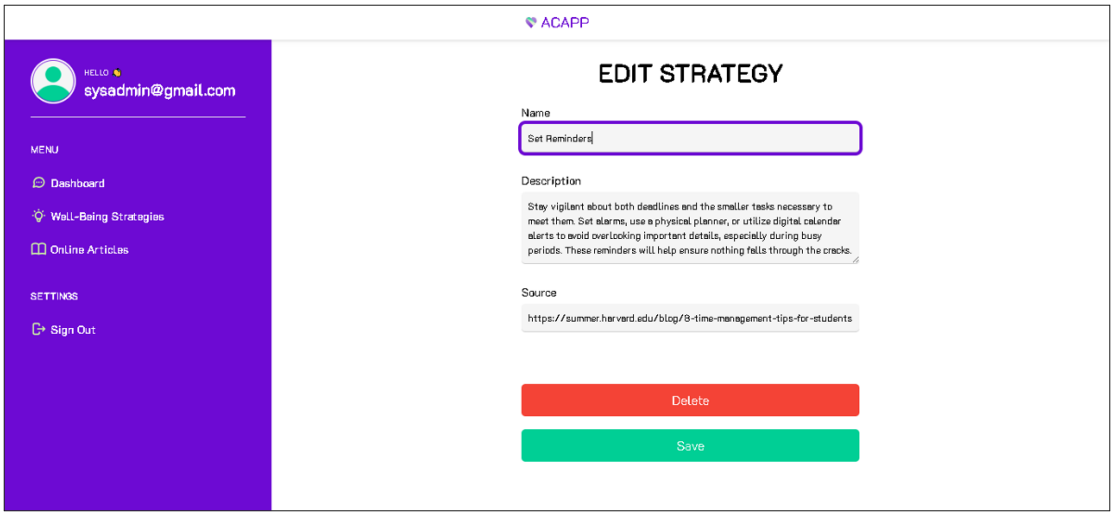

Figure 54: Admin Edit Strategy Page

R. Admin Add Online Articles

In the sidebar navigation, under "Online Articles," the admin user can access the list of available online articles within the app. By clicking the "add article" button, the admin user can add a new article. Furthermore, clicking on a specific article will allow the user to view its details.

After clicking the "add article" button, the admin will be redirected to the Admin Add Article page. On this page, the admin is required to fill out the

title, summary, and source of the online article. Once the admin has completed the required fields, they can click the "save" button to store the article in the database.

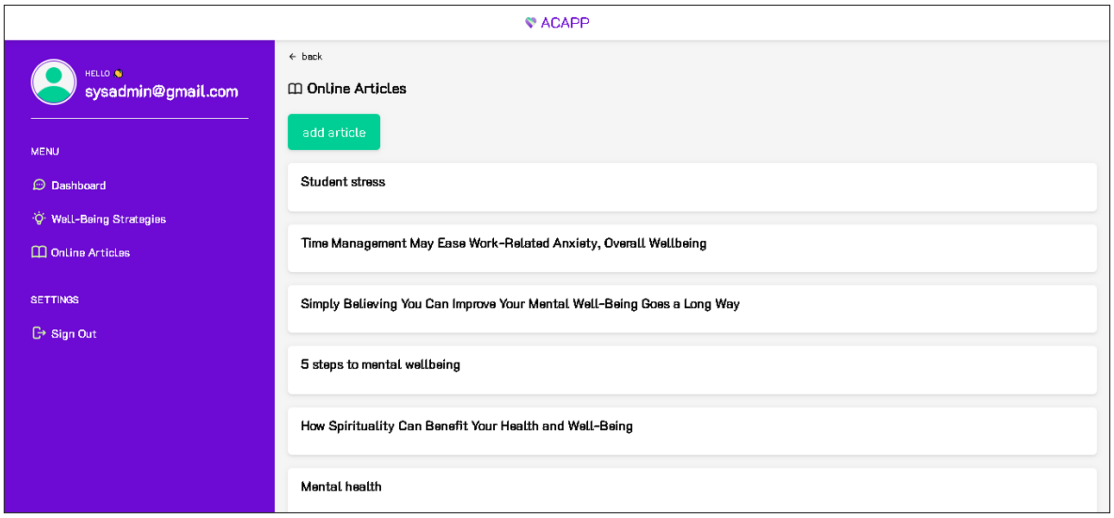

Figure 55: Admin Online Articles Page

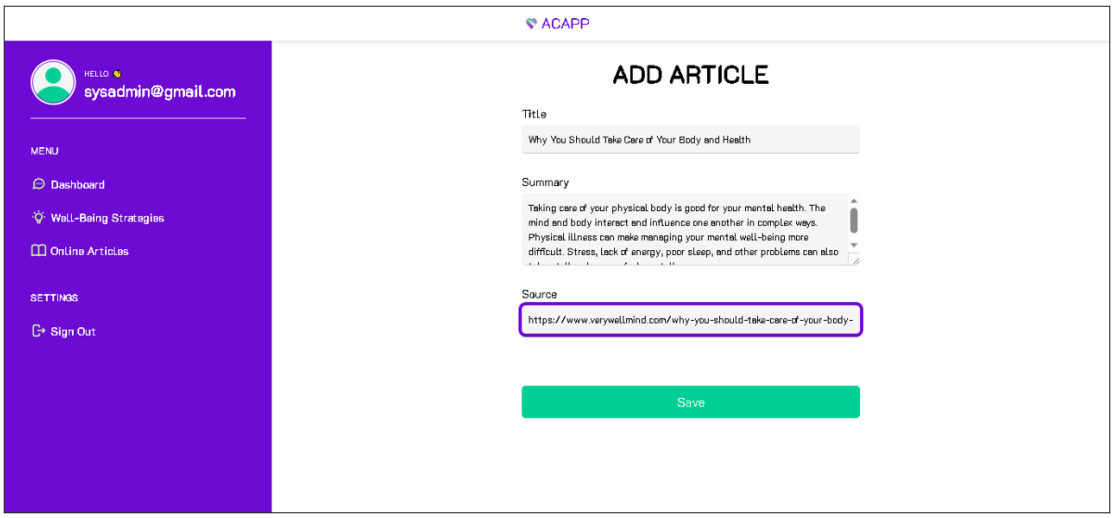

Figure 56: Admin Add Article Page

S. Admin Edit Article

The admin can view the details of an article by clicking on it from the Admin Online Articles Page. Upon clicking the article, the admin will be directed to a page where they can view the article's details. On this page, the admin will also find an "edit article" button, which they can click to modify the article.

Once the "edit article" button is clicked, the admin will be directed to the Admin Edit Article page. On this page, the admin can modify the title, summary, and source of the article. To delete the article, the admin can click the "delete" button, while clicking the "save" button will save any changes made to the article.

Figure 57: Admin View Article Page

Figure 58: Admin Edit Article Page

VI. Discussions

ACAPP is a web application specifically designed to enhance the well-being of university students. With the inclusion of a menu-based assistant named "Kaya," students now have an interactive platform that offers them additional support and guidance. The app's responsive design ensures compatibility across devices of various sizes, providing students with convenience and flexibility in accessing its features.

The app was able to implement tools for fostering self-awareness, cultivating healthy habits, and acquiring well-being strategies. The app was built using the Django framework, which enables users to record and manage their data. To get started, student users can create an account through the Sign-Up page. The Account Settings page allows users to personalize their profiles, including personal details and notification time prompts.

The app features a Mood Tracker section, which enables users to record and track their daily moods, along with the triggers that influence their moods. This tool fosters self-awareness, as users can review monthly mood statistics. Additionally, the Habit Tracker section allows users to record their habits and monitor their progress towards achieving recommended habits. Users can customize the habits they wish to track, and the tracker visually indicates whether they meet or fall below the recommended levels. The recommended habits were compiled by the developer and reviewed by a guidance counselor.

The My Strategies section of the app provides users with well-being strategies to explore. Users can choose their preferred strategies facilitating a personalized learning experience. The well-being strategies were categorized using the five dimensions that contribute to the well-being of Filipino university students identified by Cristobal and Bance (2021). The content in this section was curated by the developer and reviewed by a guidance counselor.

The Diary feature enables users to write their thoughts and feelings, promoting self-reflection and emotional awareness. The app also provides daily insights

to users, offering them a comprehensive overview of their moods, habits, well-being strategies, and online articles read. This helps users understand their daily experiences and track their progress in various aspects of their well-being journey.

The Resources section of the app provides users with access to online articles pertaining to mental health, enabling them to broaden their understanding of this expansive topic. In order to gain insight into their own well-being, it is crucial for users to familiarize themselves with relevant definitions and other relevant knowledge. The content within this section has been researched and compiled by the developer, with the guidance counselor providing review and input.

To enhance user engagement, the app incorporates a virtual assistant implemented using Rasa, a conversational AI platform. The assistant functions in a rule-based manner, offering predefined options to guide users and ensure productive conversations, as suggested by guidance counselors. It welcomes users, provides information about the app's functionalities and limitations, and gathers notification preferences. It sends reminders to encourage users to utilize the mood and habit trackers and notifies them when daily insights become available. The assistant also offers guidance and support on using key features of the app, such as mood and habit tracking, well-being strategy development, diary writing, and accessing mental health resources.

Additionally, an admin page was developed to manage user accounts and oversee app content, including well-being strategies and online articles.

VII. Conclusions

The development of ACAPP, a mental health application, was driven by the recognition of the vital importance of well-being among university students. ACAPP's primary objective is to support students in fostering self-awareness, cultivating healthy habits, and acquiring effective well-being strategies. The app's features, such as the mood tracker, habit tracker, and diary, empower users to develop self-awareness and establish positive routines. The My Strategies section further facilitates the development of personalized well-being strategies, tailored specifically for Filipino university students based on a dedicated model [28].

Moreover, the resources section of ACAPP plays a vital role in expanding users' knowledge of mental health, aiding in the comprehension of unfamiliar terms and concepts. To enhance the user experience, ACAPP offers customization options for personal profiles and notification prompts.

The incorporation of the menu-based assistant offers users a guided and interactive experience. Recognizing the time-consuming nature of developing recommended habits and well-being strategies, ACAPP seeks to improve user engagement to promote long-term positive outcomes.

By prioritizing student well-being, ACAPP strives to support individuals in cultivating healthier, more fulfilling lives, while also working towards preventing the onset of mental disorders. Through its comprehensive platform, ACAPP aims to facilitate positive change and promote the overall well-being of university students.

VIII. Recommendations

ACAPP, being in its early stage as a mental health app, acknowledges that there are still limitations and room for improvements. Based on the feedback from guidance counselors, the following recommendations can be considered for future enhancements:

1. To provide a more comprehensive experience for users, it would be beneficial to incorporate an interpretation feature for recorded moods on the mood tracker. This could involve analyzing the recorded moods, exploring their impact on the user's life, and offering insights or implications based on the patterns observed.

2. While the current habit tracker focuses on tracking positive habits, it would be valuable to include the ability to track unhealthy habits as well. This would enable users to identify and work towards reducing such habits. Additionally, offering flexibility in how habits are measured can provide a more personalized and customizable experience for users.

3. To enhance the effectiveness of suggested well-being strategies, consider integrating an evaluation tool that allows users to assess the strategies they have tried. Including online assessment tools that users can access independently can also promote self-awareness and help users discover new strategies that may align with their needs. Furthermore, incorporating audio-visual resources, particularly for practices like meditation, mindfulness, and breathing exercises, can enhance the variety and accessibility of well-being strategies.

IX. Bibliography

[1] J. O. Serrano, M. E. S. Reyes, and A. B. De Guzman, "Psychological distress and coping of filipino university students amidst the global pandemic: A mixed-method study," *Journal of Positive School Psychology*, vol. 6, no. 6, 2022.

[2] W. H. O. WHO, "Mental disorders." https://www.who.int/news-room/fact-sheets/detail/mental-disorders, June 2022.

[3] C. M. Toquero, "Provision of mental health services for people with disabilities in the philippines amid coronavirus outbreak," *Disability Society*, vol. 36, no. 6, p. 1026–1032, 2021.

[4] A. B. Martinez, M. Co, J. Lau, and J. S. Brown, "Filipino help-seeking for mental health problems and associated barriers and facilitators: A systematic review," *Social Psychiatry and Psychiatric Epidemiology*, vol. 55, no. 11, p. 1397–1413, 2020.

[5] M. L. Tee, C. A. Tee, J. P. Anlacan, K. J. Aligam, P. W. Reyes, V. Kuruchittham, and R. C. Ho, "Psychological impact of covid-19 pandemic in the philippines," *Journal of Affective Disorders*, vol. 277, p. 379–391, 2020.

[6] W. H. O. WHO, "Health promotion glossary of terms 2021." https://www.who.int/publications/i/item/9789240038349, December 2021.

[7] R. C. F. Sun and D. T. L. Shek, *Well-Being, Student*. December 2020.

[8] C. F. D. Leochico, E. M. V. Austria, and A. I. Espiritu, "Global online interest in telehealth, telemedicine, telerehabilitation, and related search terms amid the covid-19 pandemic: An infodemiological study," *Acta Medica Philippina*, January 2020.

[9] K. Stawarz, C. Preist, and D. Coyle, "Use of smartphone apps, social media, and web-based resources to support mental health and well-being: Online survey," *JMIR mental health*, vol. 6, p. e12546, July 2019.

[10] N. I. of Mental Health, "Caring for your mental health." https://www.nimh.nih.gov/health/topics/caring-for-your-mental-health, April 2021.

[11] J. Lally, J. C. Tully, and R. M. Samaniego, "Mental health services in the philippines," *BJPsych international*, vol. 16, pp. 62–64, August 2019.

[12] M. R. Hechanova-Alampay, P. L. Angeles, A. Tuliao, E. Hilario, A. F. Pagente, and C. V. Narra, "The development and pilot evaluation of a mental health mobile app in the philippines," *Mental Health and Social Inclusion*, November 2022.

[13] J. R. Melcher, E. Camacho, S. Lagan, and J. Torous, "College student engagement with mental health apps: analysis of barriers to sustained use," *Journal of American College Health*, vol. 70, pp. 1819–1825, October 2020.

[14] E. G. Lattie, K. A. Cohen, E. Hersch, K. D. A. Williams, K. P. Kruzan, C. MacIver, J. Hermes, K. Maddi, M. J. Kwasny, and D. C. Mohr, "Uptake and effectiveness of a self-guided mobile app platform for college student mental health," *Internet interventions*, vol. 27, p. 100493, December 2021.

[15] M. Eisenstadt, S. Liverpool, E. Infanti, R. M. Ciuvat, and C. Carlsson, "Mobile Apps That Promote Emotion Regulation, Positive Mental Health, and Well-being in the General Population: Systematic Review and Meta-analysis," *JMIR mental health*, vol. 8, p. e31170, November 2021.

[16] M. Al-Refae, A.-S. Al-Refae, M. Munroe, N. A. Sardella, and M. D. Ferrari, "A self-compassion and mindfulness-based cognitive mobile intervention (serene) for depression, anxiety, and stress: Promoting adaptive emotional regulation and wisdom," *Frontiers in Psychology*, vol. 12, March 2021.

[17] B. Chen, T. Yang, L. Xiao, C. Xu, and C. Zhu, "Effects of mobile mindfulness meditation on the mental health of university students: Systematic review and meta-analysis," *Journal of Medical Internet Research*, vol. 25, p. e39128, April 2022.

[18] O. Lahtinen, J. Aaltonen, J. K. Kaakinen, L. Franklin, and J. Hyönä, "The effects of app-based mindfulness practice on the well-being of university students and staff," *Current Psychology*, vol. 42, pp. 4412–4421, May 2021.

[19] J. A. Clarke and S. Draper, "Intermittent mindfulness practice can be beneficial, and daily practice can be harmful. an in depth, mixed methods study of the "calm" app's (mostly positive) effects," *Internet interventions*, vol. 19, p. 100293, March 2020.

[20] M. Jitanan, V. Somanandana, S. Jitanan, U. Lalitpasan, and S. Kham-In, "The development of "friend from heart" application based on line system to promote well-being of undergraduate students of faculty of education, kasetsart university," *Higher Education Studies*, vol. 11, p. 215, May 2021.

[21] S. Jeong, S. Alghowinem, L. Aymerich-Franch, K. Arias, A. Lapedriza, R. W. Picard, H.-S. Park, and C. Breazeal, "A robotic positive psychology coach to improve college students' wellbeing," August 2020.

[22] K. M. R. De Asis, E. J. P. Guillem, F. A. M. Reyes, and M. J. C. Samonte, "Serenity: A stress-relieving virtual reality application based on philippine environmental variables," June 2020.

[23] C. Oliveira, A. Pereira, P. Vagos, C. Nóbrega, J. Gonçalves, and B. Q. Afonso, "Effectiveness of mobile app-based psychological interventions for college students: A systematic review of the literature," *Frontiers in Psychology*, vol. 12, May 2021.

[24] O. Oti and I. Pitt, "Online mental health interventions designed for students in higher education: A user-centered perspective," *Internet interventions*, vol. 26, p. 100468, December 2021.

[25] F. Alqahtani and R. Orji, "Insights from user reviews to improve mental health apps," *Health Informatics Journal*, vol. 26, pp. 2042–2066, January 2020.

[26] A. Abd-Alrazaq, M. Alajlani, A. A. Alalwan, B. M. Bewick, P. Gardner, and M. Househ, "An overview of the features of chatbots in mental health: A scoping review," *International Journal of Medical Informatics*, vol. 132, p. 103978, September 2019.

[27] I. Hungerbuehler, K. Daley, K. Cavanagh, H. G. Claro, and M. Kapps, "Chatbot-based assessment of employees' mental health: Design process and pilot implementation," *JMIR formative research*, vol. 5, p. e21678, April 2021.

[28] N. V. Cristobal and L. O. Bance, "Come into bloom: A grounded theory of well-being among filipino university students," *Philippine social science journal*, vol. 4, pp. 40–49, December 2021.

[29] A. Morin, "Self-awareness part 1: Definition, measures, effects, functions, and antecedents," *Social and Personality Psychology Compass*, vol. 5, pp. 807–823, October 2011.

[30] T. Eurich, "What self-awareness really is (and how to cultivate it)." https://hbr.org/2018/01/what-self-awareness-really-is-and-how-to-cultivate-it, January 2018.

[31] R. T. Inc., "The rasa masterclass handbook: A companion guide to the rasa masterclass video series." https://info.rasa.com/masterclass-ebook.

X. Appendix

A. Source Code

A..1 ACAPP Web App

Listing 1: manage.py

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'acapp.
        ↪ settings')
    try:
        from django.core.management import
            ↪ execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed
                ↪ and "
            "available on your PYTHONPATH environment variable?
                ↪ Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

Listing 2: acapp/settings.py

```
"""
Django settings for acapp project.

Generated by 'django-admin startproject' using Django 4.2.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.2/ref/settings/
"""
import os

from pathlib import Path

from django_jinja.builtins import DEFAULT_EXTENSIONS

# Build paths inside the project like this: BASE_DIR / 'subdir
    ↪ '.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.2/howto/deployment/
    ↪ checklist/

# SECURITY WARNING: keep the secret key used in production
    ↪ secret!
SECRET_KEY = "django-insecure-2k3&avvla0vh@zhu2fd7sr&nw68w@1&#
    ↪ rn1nrs8o#i8iyt-pa^"

# SECURITY WARNING: don't run with debug turned on in
    ↪ production!
DEBUG = True

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "main_app",
    "django_apscheduler",
    "webpush",
]

MIDDLEWARE = [
    "django.middleware.security.SecurityMiddleware",
    "django.contrib.sessions.middleware.SessionMiddleware",
    "django.middleware.common.CommonMiddleware",
    "django.middleware.csrf.CsrfViewMiddleware",
    "django.contrib.auth.middleware.AuthenticationMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware",
    "django.middleware.clickjacking.XFrameOptionsMiddleware",
]

ROOT_URLCONF = "acapp.urls"

TEMPLATES = [
    {
        "BACKEND": "django.template.backends.django.
            ↪ DjangoTemplates",
        "DIRS": [],
        "APP_DIRS": True,
        "OPTIONS": {
            "context_processors": [
                "django.template.context_processors.debug",
                "django.template.context_processors.request",
                "django.contrib.auth.context_processors.auth",
                "django.contrib.messages.context_processors.
                    ↪ messages",
            ],
        },
    },
]


WSGI_APPLICATION = "acapp.wsgi.application"


# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.sqlite3",
        "NAME": BASE_DIR / "db.sqlite3",
    }
}


# Password validation
# https://docs.djangoproject.com/en/4.2/ref/settings/#auth-
    ↪ password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        "NAME": "django.contrib.auth.password_validation.
            ↪ UserAttributeSimilarityValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.
            ↪ MinimumLengthValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.
            ↪ CommonPasswordValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.
            ↪ NumericPasswordValidator",
    },
]

WEBPUSH_SETTINGS = {
    "VAPID_PUBLIC_KEY": "
        ↪ BBXssLfjrKsKMhzFfKpT3phRDBn8xf23qPAnKoXB_5SJSrsquccRmKQ82UypW8L7gjeTIFu
        ↪ -1Rnk8FMWwxvC07E",
    "VAPID_PRIVATE_KEY": "ynd16rg3148sAiZ-OCCkOjZiWZe0Wwc-
        ↪ RzVUi4JfApw",
    "VAPID_ADMIN_EMAIL": "admin@gmai.com",
}


# Internationalization
# https://docs.djangoproject.com/en/4.2/topics/i18n/

LANGUAGE_CODE = "en-us"

TIME_ZONE = "Asia/Manila"

USE_I18N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/

MEDIA_ROOT = os.path.join(BASE_DIR, "media")
```

```
MEDIA_URL = "/media/"

STATIC_URL = "static/"


# Default primary key field type
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-
      ↪ auto-field

DEFAULT_AUTO_FIELD = "django.db.models.BigAutoField"

AUTH_USER_MODEL = "main_app.CustomUser"

LOGIN_URL = "/signin"
```

Listing 3: acapp/asgi.py

```
"""
ASGI config for acapp project.

It exposes the ASGI callable as a module-level variable named
      ↪ ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/howto/deployment/asgi/
"""

import os

from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'acapp.settings
      ↪ ')

application = get_asgi_application()
```

Listing 4: acapp/urls.py

```
"""
URL configuration for acapp project.

The `urlpatterns` list routes URLs to views. For more
      ↪ information please see:
    https://docs.djangoproject.com/en/4.2/topics/http/urls/
Examples:
Function views
    1. Add an import: from my_app import views
    2. Add a URL to urlpatterns: path('', views.home, name='home
          ↪ ')
Class-based views
    1. Add an import: from other_app.views import Home
    2. Add a URL to urlpatterns: path('', Home.as_view(), name='
          ↪ home')
Including another URLconf
    1. Import the include() function: from django.urls import
          ↪ include, path
    2. Add a URL to urlpatterns: path('blog/', include('blog.
          ↪ urls'))
"""
from django.contrib import admin
from django.urls import path, include

from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path("admin/", admin.site.urls),
    path("", include("main_app.urls")),
    path("webpush/", include("webpush.urls")),
]

if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=
          ↪ settings.MEDIA_ROOT)
```

Listing 5: acapp/wsgi.py

```
"""
WSGI config for acapp project.

It exposes the WSGI callable as a module-level variable named
      ↪ ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'acapp.settings
      ↪ ')

application = get_wsgi_application()
```

Listing 6: main_app/apps.py

```
from django.apps import AppConfig


class MainAppConfig(AppConfig):
    default_auto_field = "django.db.models.BigAutoField"
    name = "main_app"
```

Listing 7: main_app/forms.py

```
from django import forms
from django.contrib.auth import get_user_model
from .models import Strategy, Article, Diary

CustomUser = get_user_model()


class UserProfileForm(forms.ModelForm):
    class Meta:
        model = CustomUser
        fields = [
            "first_name",
            "last_name",
            "contact_number",
            "birthday",
            "profile_image",
        ]
        widgets = {
            "first_name": forms.TextInput(attrs={"class": "w3-
                  ↪ input field-input"}),
            "last_name": forms.TextInput(attrs={"class": "w3-
                  ↪ input field-input"}),
            "contact_number": forms.NumberInput(
                attrs={
                    "class": "w3-input field-input",
                    "type": "tel",
                    "pattern": "09[0-9]{9}",
                    "placeholder": "09xxxxxxxx",
                }
            ),
            "birthday": forms.DateInput(
                attrs={"class": "w3-input field-input", "type": "
                      ↪ date"}
            ),
            "profile_image": forms.FileInput(attrs={"class": "w3-
                  ↪ input field-input"}),
        }

    def __init__(self, *args, **kwargs):
        super(UserProfileForm, self).__init__(*args, **kwargs)
        self.fields["first_name"].label = "First Name"
        self.fields["last_name"].label = "Last Name"
        self.fields["contact_number"].label = "Contact Number"
        self.fields["birthday"].label = "Birthday"
        self.fields["profile_image"].label = "Profile Image"
        self.fields["profile_image"].widget.attrs["value"] = ""


class TimePromptForm(forms.ModelForm):
    tracker_time_prompt = forms.TimeField(
        label="Daily Check-In",
        widget=forms.TimeInput(
            attrs={"class": "w3-input field-input", "type": "time
                  ↪ "}
        ), # Specify the display format for the time
    )

    insights_time_prompt = forms.TimeField(
        label="Daily Insights",
        widget=forms.TimeInput(
            attrs={"class": "w3-input field-input", "type": "time
                  ↪ "}
        ), # Specify the display format for the time
    )

    class Meta:
        model = CustomUser
        fields = ["tracker_time_prompt", "insights_time_prompt"]

    def clean(self):
        cleaned_data = super().clean()
        tracker_time_prompt = cleaned_data.get("
              ↪ tracker_time_prompt")
        insights_time_prompt = cleaned_data.get("
              ↪ insights_time_prompt")

        if (
            tracker_time_prompt
            and insights_time_prompt
            and tracker_time_prompt > insights_time_prompt
        ):
```

```
            raise forms.ValidationError(
                "Daily check-in cannot be later than the daily
                    ↪ insights."
            )

        return cleaned_data


class StrategyForm(forms.ModelForm):
    description = forms.CharField(
        widget=forms.Textarea(
            attrs={"rows": 4, "cols": 40, "class": "w3-input
                ↪ field-input"}
        )
    )

    class Meta:
        model = Strategy
        fields = ["name", "description", "source"]
        widgets = {
            "name": forms.TextInput(attrs={"class": "w3-input
                ↪ field-input"}),
            "source": forms.URLInput(attrs={"class": "w3-input
                ↪ field-input"}),
        }


class ArticleForm(forms.ModelForm):
    summary = forms.CharField(
        widget=forms.Textarea(
            attrs={"rows": 4, "cols": 40, "class": "w3-input
                ↪ field-input"}
        )
    )

    class Meta:
        model = Article
        fields = ["title", "summary", "source"]
        widgets = {
            "title": forms.TextInput(attrs={"class": "w3-input
                ↪ field-input"}),
            "source": forms.URLInput(attrs={"class": "w3-input
                ↪ field-input"}),
        }


class DiaryForm(forms.ModelForm):
    thoughts = forms.CharField(
        widget=forms.Textarea(
            attrs={"rows": 4, "cols": 40, "class": "w3-input
                ↪ diary-input"}
        )
    )

    class Meta:
        model = Diary
        fields = ["thoughts"]
```

Listing 8: main_app/models.py

```
from django.contrib.auth.models import AbstractUser
from django.db import models


class CustomUser(AbstractUser):
    email = models.EmailField(unique=True, blank=False, null=
        ↪ False)
    contact_number = models.CharField(max_length=20, blank=True)
    birthday = models.DateField(blank=True, null=True)
    tracker_time_prompt = models.TimeField(blank=True, null=True
        ↪ )
    insights_time_prompt = models.TimeField(blank=True, null=
        ↪ True)
    profile_image = models.ImageField(
        upload_to="profile_images/", blank=True, null=True
    )

    USERNAME_FIELD = "email"
    REQUIRED_FIELDS = ["username", "password"]


class MoodEntry(models.Model):
    mood_entry_id = models.AutoField(primary_key=True)
    user = models.ForeignKey(CustomUser, on_delete=models.
        ↪ CASCADE)
    mood = models.CharField(max_length=100, blank=False, null=
        ↪ False)
    mood_trigger = models.CharField(max_length=100, blank=False,
        ↪  null=False)
    date = models.DateTimeField(auto_now_add=True, blank=False,
        ↪ null=False)
    # date = models.DateTimeField()

    def __str__(self):
        return f"{self.user.email}"


class Habit(models.Model):
    habit_id = models.AutoField(primary_key=True)
    habit_name = models.CharField(max_length=100, blank=False,
        ↪ null=False)
    users = models.ManyToManyField(CustomUser)


class HabitEntry(models.Model):
    habit_entry_id = models.AutoField(primary_key=True)
    user = models.ForeignKey(CustomUser, on_delete=models.
        ↪ CASCADE)
    habit = models.CharField(max_length=100, blank=False, null=
        ↪ False)
    data = models.CharField(max_length=100, blank=False, null=
        ↪ False)
    date = models.DateTimeField(auto_now_add=True, blank=False,
        ↪ null=False)
    # date = models.DateTimeField()


class Strategy(models.Model):
    strategy_id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=200, blank=False, null=
        ↪ False)
    description = models.TextField()
    source = models.URLField(null=True, blank=True)


class StrategyTried(models.Model):
    strategy_tried_id = models.AutoField(primary_key=True)
    user = models.ForeignKey(CustomUser, on_delete=models.
        ↪ CASCADE)
    strategy = models.ForeignKey(Strategy, on_delete=models.
        ↪ CASCADE)
    is_liked = models.BooleanField(default=False)
    is_effective = models.BooleanField(default=False)
    date = models.DateTimeField(auto_now_add=True, blank=False,
        ↪ null=False)


class Article(models.Model):
    article_id = models.AutoField(primary_key=True)
    title = models.CharField(max_length=100, null=True, blank=
        ↪ True)
    summary = models.TextField()
    source = models.URLField(null=True, blank=True)


class ArticleRead(models.Model):
    article_read_id = models.AutoField(primary_key=True)
    user = models.ForeignKey(CustomUser, on_delete=models.
        ↪ CASCADE)
    article = models.ForeignKey(Article, on_delete=models.
        ↪ CASCADE)
    date = models.DateTimeField(auto_now_add=True, blank=False,
        ↪ null=False)


class Diary(models.Model):
    diary_id = models.AutoField(primary_key=True)
    user = models.ForeignKey(CustomUser, on_delete=models.
        ↪ CASCADE)
    thoughts = models.TextField(null=True, blank=True)
    date = models.DateTimeField(auto_now_add=True, blank=False,
        ↪ null=False)

    def __str__(self):
        return self.title()

    def title(self):
        return f"Day {self.get_entry_count()}"

    def get_entry_count(self):
        return Diary.objects.filter(user=self.user, date__lte=
            ↪ self.date).count()
```

Listing 9: main_app/urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    path("", views.home, name="home"),
    path("talk_to_kaya/", views.talk_to_kaya, name="talk_to_kaya
        ↪ "),
    path("chat/<str:message>/", views.chat, name="chat"),
    path("mood/", views.mood_tracker, name="mood_tracker"),
    path("add_mood/", views.add_mood, name="add_mood"),
    path("habit/", views.habit_tracker, name="habit_tracker"),
    path("habit/add_entry", views.add_habit_entry, name="
        ↪ add_habit_entry"),
    path("habit/add", views.add_habit, name="add_habit"),
    path("habit/edit", views.edit_habit, name="edit_habit"),
    path("signin/", views.signin, name="signin"),
    path("signup/", views.signup, name="signup"),
    path("signout/", views.signout, name="signout"),
    path("diary/", views.diary_entry_list, name="
        ↪ diary_entry_list"),
```

```
    path("diary/<int:pk>/", views.diary_entry, name="diary_entry
        ↪ "),
    path("my_strategies/", views.strategies, name="strategies"),
    path("strategies/<int:pk>/", views.view_strategy, name="
        ↪ view_strategy"),
    path("done_strategy/<str:strategy>/", views.done_strategy,
        ↪ name="done_strategy"),
    path("tried_strategy/<str:pk>/", views.tried_strategy, name
        ↪ ="tried_strategy"),
    path("liked_strategy/<str:pk>/", views.liked_strategy, name
        ↪ ="liked_strategy"),
    path(
        "effective_strategy/<str:pk>",
        views.effective_strategy,
        name="effective_strategy",
    ),
    path("resources/", views.resources, name="resources"),
    path("resources/<int:pk>/", views.view_article, name="
        ↪ view_article"),
    path("done_article/<str:article>/", views.done_article, name
        ↪ ="done_article"),
    path("read_article/<str:pk>/", views.read_article, name="
        ↪ read_article"),
    path("send_message/", views.send_message, name="send_message
        ↪ "),
    path("account_settings/", views.account_settings, name="
        ↪ account_settings"),
    path("edit_profile/", views.edit_profile, name="edit_profile
        ↪ "),
    path(
        "notification_settings/",
        views.notification_settings,
        name="notification_settings",
    ),
    # API
    path("api/get_user/", views.get_user, name="get_user"),
    path("api/save_mood/", views.save_mood, name="save_mood"),
    path("api/get_habits/", views.get_habits, name="get_habits")
        ↪ ,
    path("api/save_habit_entry/", views.save_habit_entry, name="
        ↪ save_habit_entry"),
    path("api/save_habits/", views.save_habits, name="
        ↪ save_habits"),
    path("api/get_strategies/", views.get_strategies, name="
        ↪ get_strategies"),
    path("api/get_articles/", views.get_articles, name="
        ↪ get_articles"),
    path(
        "api/save_notifications_settings/",
        views.save_notifications_settings,
        name="save_notifications_settings",
    ),
    # Admin
    path("admin_dashboard", views.admin_dashboard, name="
        ↪ admin_dashboard"),
    path("strategies_list/", views.strategies_list, name="
        ↪ strategies_list"),
    path("strategy/add/", views.add_strategy, name="add_strategy
        ↪ "),
    path("strategy/edit/<int:pk>/", views.edit_strategy, name="
        ↪ edit_strategy"),
    path(
        "strategy/view/<int:pk>/", views.admin_view_strategy,
            ↪ name="admin_view_strategy"
    ),
    path("strategy/delete/<int:pk>/", views.delete_strategy,
        ↪ name="delete_strategy"),
    path("articles_list/", views.articles_list, name="
        ↪ articles_list"),
    path("article/add/", views.add_article, name="add_article"),
    path("article/edit/<int:pk>/", views.edit_article, name="
        ↪ edit_article"),
    path("article/view/<int:pk>/", views.admin_view_article,
        ↪ name="admin_view_article"),
    path("article/delete/<int:pk>/", views.delete_article, name
        ↪ ="delete_article"),
]
```

Listing 10: main_app/views.py

```
from django.shortcuts import render, redirect,
    ↪ get_object_or_404
from django.contrib.auth import authenticate, login, logout
from django.contrib import messages
from django.utils import timezone

from django.contrib.auth.forms import UserCreationForm

from django.contrib.auth.decorators import login_required
from django.views.decorators.csrf import csrf_exempt

from django.http import HttpResponse, JsonResponse,
    ↪ HttpResponseRedirect
from django.urls import reverse

from .forms import *
from .models import (
    CustomUser,
    MoodEntry,
    Habit,
    HabitEntry,
    Strategy,
    StrategyTried,
    Article,
    ArticleRead,
    Diary,
)


# connects to rasa bot
from rasa.core.agent import Agent
from rasa.core.tracker_store import SQLTrackerStore
from rasa.core.utils import EndpointConfig


# enable async
from asgiref.sync import async_to_sync

from datetime import datetime, timedelta
import json

from webpush import send_user_notification


# Load the tracker store
tracker_store = SQLTrackerStore(
    host="localhost",
    username="postgres",
    password="exoshidae",
    port="5432",
    db="postgres",
    dialect="postgresql",
)


# Load the model
model_path = "./rasa_bot/models"
action_endpoint = EndpointConfig(url="http://localhost:5055/
    ↪ webhook")
agent = Agent.load(
    model_path, tracker_store=tracker_store, action_endpoint=
        ↪ action_endpoint
)


@login_required
@async_to_sync
async def chat(request, message):
    current_userid = str(request.user.id)

    # Load conversation history
    tracker = await tracker_store.get_or_create_tracker(
        ↪ sender_id=current_userid)
    conversation_history = []
    for event in tracker.events:
        event_dict = event.as_dict()
        msg_dict = {}

        # Convert the Unix timestamp to a datetime object
        formatted_timestamp = convert_timestamp(event_dict.get("
            ↪ timestamp"))

        # user messages
        if event_dict.get("event") == "user":
            msg_dict = {
                "sender": "user",
                "text": event_dict.get("text"),
                "timestamp": formatted_timestamp,
            }
            conversation_history.append(msg_dict)

        # bot messages w/out attachment
        elif event_dict.get("event") == "bot" and all(
            value is None for value in event_dict.get("data").
                ↪ values()
        ):
            # print(event_dict)
            msg_dict = {
                "sender": "bot",
                "text": event_dict.get("text"),
                "timestamp": formatted_timestamp,
            }
            conversation_history.append(msg_dict)

        # bot messages w/ attachment
        elif event_dict.get("event") == "bot" and not all(
            value is None for value in event_dict.get("data").
                ↪ values()
        ):
            msg_dict = {
                "sender": "bot",
                "text": event_dict.get("text"),
                "data": event_dict.get("data"),
                "timestamp": formatted_timestamp,
            }
```

```
            conversation_history.append(msg_dict)

    context = {"conversation": conversation_history, "
        ↪ user_message": message}

    return render(request, "main_app/chat.html", context)


def convert_timestamp(timestamp):
    # Convert the Unix timestamp to a datetime object
    dt = datetime.fromtimestamp(timestamp)
    formatted_timestamp = dt.strftime("%#I:%M %p")
    return formatted_timestamp


@login_required
@async_to_sync
async def send_message(request):
    if request.method == "POST":
        # Get the user's message from the request
        msgData = json.loads(request.body)
        message = msgData.get("message")
        userid = msgData.get("userid")

        menu_options = {
            "track my mood": "/mood",
            "track my habit": "/habit",
            "view my strategies": "/my_strategies",
            "view my diary": "/diary",
            "view resources": "/resources",
            "get daily insights": "/resources",
        }

        # Use the Rasa Agent to get the intent of the message
        parsed_msg = await agent.parse_message(message)
        print("\nIntent: ", parsed_msg["intent"]["name"])

        # Use the Rasa Agent to generate a response
        bot_responses = await agent.handle_text(text_message=
            ↪ message, sender_id=userid)

        tracker = await tracker_store.retrieve(userid)
        slot_values = tracker.current_slot_values()
        print("\nSlots: ", slot_values)

        responses_list = []

        if bot_responses:
            print("\nResponses: ", bot_responses)
            for response in bot_responses:
                if "custom" in response:
                    if (
                        response["custom"]["type"] == "strategies"
                        or response["custom"]["type"] == "
                            ↪ resources"
                    ):
                        # display slides for strategies/articles
                        response_dict = {
                            "type": response["custom"]["type"],
                            "slides": response["custom"]["slides
                                ↪ "],
                        }

                    elif response["custom"]["type"] == "
                        ↪ notifications":
                        # display time input for notifications
                        response_dict = {
                            "type": response["custom"]["type"],
                            "text": response["custom"]["text"],
                        }

                    elif response["custom"]["type"] == "slider":
                        # display slider for habits
                        response_dict = {
                            "type": response["custom"]["type"],
                            "text": response["custom"]["text"],
                            "min": response["custom"]["min"],
                            "max": response["custom"]["max"],
                        }

                    responses_list.append(response_dict)
                    print("\nresponses_list :", responses_list)

                else:
                    response_dict = {"text": response["text"]}

                    choices = []
                    if "buttons" in response:
                        buttons = response["buttons"]
                        html_buttons = []
                        button_counter = 0
                        for button in buttons:
                            button_counter += 1
                            title = button["title"]
                            payload = button["payload"]
                            if message == "edit habits":
                                choice = {"title": title, "payload
                                    ↪ ": payload}
                                choices.append(choice)
                            else:
                                html_button = f'<button type="
                                    ↪ submit" class="w3-button
                                    ↪ chat-btn" id="choice{
                                    ↪ button_counter}" value="{
                                    ↪ payload}">{title}</button
                                    ↪ >'
                                html_buttons.append(html_button)

                        # Join the HTML buttons into a string
                        html_buttons_str = " ".join(html_buttons)
                    else:
                        html_buttons_str = []

                    if choices != []:
                        response_dict.update({"choices": choices})
                    else:
                        response_dict.update({"html_buttons_str":
                            ↪ html_buttons_str})

                    responses_list.append(response_dict)

        else:
            print("\n Didn't receive response from bot\n")
            response = {"text": "Sorry, I didn't get that"}
            responses_list.append(response)

        result = {
            "responses_list": responses_list,
        }

        if message in menu_options:
            url = menu_options[message]
            result = {
                "responses_list": responses_list,
                "url": url,
            }
            return JsonResponse(result)
            # return HttpResponseRedirect(url)
        else:
            return JsonResponse(result)

    return HttpResponse("Error")


@csrf_exempt
def save_mood(request):
    if request.method == "POST":
        try:
            # Get the JSON data from the request
            data = json.loads(request.body)

            user_id = data["user_id"]
            mood = data["mood"]
            mood_trigger = data["mood_trigger"]

            # Retrieve the user instance
            user = CustomUser.objects.get(id=user_id)

            # Create a new mood entry
            mood_entry = MoodEntry(user=user, mood=mood,
                ↪ mood_trigger=mood_trigger)

            # Save the mood entry to the database
            mood_entry.save()

            # Return a JSON response (optional)
            return JsonResponse({"status": "success"})
        except Exception as e:
            return JsonResponse({"status": "error", "message":
                ↪ str(e)})

    return JsonResponse({"status": "error", "message": "Invalid
        ↪ request method"})


@login_required
def home(request):
    if request.user.is_staff:
        return redirect("admin_dashboard")

    daily_insights(request.user)
    return render(request, "main_app/home.html")


def check_in_user(user):
    text = "Hello, how's your day? "
    payload = {
        "head": "Kaya",
        "body": text,
        "icon": "https://drive.google.com/file/d/1nRIx1-1
            ↪ QvxvgirBdn2XFZWu5XsbUJ-uB/view?usp=sharing",
        "url": "http://127.0.0.1:8000/mood/",
    }
    send_user_notification(user=user, payload=payload, ttl=1000)


def daily_insights(user):
    current_date = datetime.now().date()
```

```
    diary = Diary.objects.get(user=user, date__date=current_date
        ↪ )
    diary_id = str(diary.diary_id)
    url = "http://127.0.0.1:8000/diary/" + diary_id

    text = f"Hi {user.first_name}, your daily insights are ready
        ↪ ! "
    payload = {
        "head": "Kaya",
        "body": text,
        "icon": "",
        "url": url,
    }
    send_user_notification(user=user, payload=payload, ttl=1000)


@login_required
def talk_to_kaya(request):
    message = "talk to kaya"
    return redirect("chat", message=message)


def signin(request):
    if request.method == "POST":
        email = request.POST["email"]
        password = request.POST["password"]
        user = authenticate(request, email=email, password=
            ↪ password)
        if user is not None:
            login(request, user)
            return redirect("home")
        else:
            messages.error(request, "Invalid email or password.")
    return render(request, "main_app/signin.html")


@login_required
def signout(request):
    logout(request)
    return render(request, "main_app/signout.html")


@login_required
def diary_entry_list(request):
    user_id = request.user.id

    diary_query = Diary.objects.filter(user_id=user_id)

    diary_entries = []

    for index, entry in enumerate(diary_query, start=1):
        mood_entries = MoodEntry.objects.filter(
            user_id=user_id, date__date=entry.date.date()
        )
        mood_values = ["great", "good", "okay", "bad", "awful"]
        mood_presence = [
            1 if mood_entries.filter(mood=mood).exists() else 0
                ↪ for mood in mood_values
        ]

        diary_entry = {
            "diary_id": entry.diary_id,
            "title": f"Day {index}",
            "date": entry.date.date(),
            "mood_present": mood_presence,
        }

        diary_entries.append(diary_entry)

    print(diary_entries)

    context = {"diary_entries": diary_entries}

    return render(request, "main_app/diary/entry_list.html",
        ↪ context)


@login_required
def diary_entry(request, pk):
    user = request.user
    user_id = user.id
    diary = Diary.objects.get(diary_id=pk, user=user)

    # all mood recorded this day
    mood_entries = MoodEntry.objects.filter(user=user,
        ↪ date__date=diary.date.date())

    allow_diary_edit = False
    entry_date = diary.date.date()
    if entry_date == datetime.now().date():
        allow_diary_edit = True

    # all moods recorded this day
    num_of_moods = mood_entries.count()
    mood_mapping = {"great": 4, "good": 3, "okay": 2, "bad": 1,
        ↪ "awful": 0}
    if num_of_moods == 1:
        moods = mood_entries[0].mood
        mood_triggers = mood_entries[0].mood_trigger
        local_datetime = timezone.localtime(mood_entries[0].date
            ↪ )
        times_of_the_day = local_datetime.strftime("%#I:%M %p")
        moods_int_values = mood_mapping[moods]
    else:
        moods = []
        mood_triggers = []
        times_of_the_day = []

        for mood_entry in mood_entries:
            moods.append(mood_entry.mood)
            mood_triggers.append(mood_entry.mood_trigger)
            local_datetime = timezone.localtime(mood_entry.date)
            times_of_the_day.append(local_datetime.strftime("%#I
                ↪ :%M %p"))

        # convert mood to corresponding integer value
        moods_int_values = [mood_mapping[mood] for mood in moods
            ↪ ]

    # all active habits recorded this day
    habits = Habit.objects.filter(users__id=user_id).values_list
        ↪ (
        "habit_name", flat=True
    )
    habits_to_track = list(habits)

    habits_data = {}
    for habit in habits_to_track:
        habit_query = HabitEntry.objects.filter(
            user=user, habit=habit, date__date=diary.date.date()
        )
        if habit_query.count() == 1:
            string_without_spaces = habit.replace(" ", "_")
            habits_data[string_without_spaces] = habit_query[0].
                ↪ data

    # all strategies tried this day
    strategies_tried = StrategyTried.objects.filter(user=user,
        ↪ date__date=entry_date)

    # all articles read this day
    articles_read = ArticleRead.objects.filter(user=user,
        ↪ date__date=entry_date)

    if request.method == "POST":
        print("\nENTER POST")
        form = DiaryForm(request.POST, instance=diary)
        if form.is_valid():
            form.save()
            print("\nSAVE CHANGES")
            messages.success(request, "Changes saved!")
        else:
            messages.error(
                request,
                "Form not valid",
            )
    else:
        form = DiaryForm(instance=diary)

    context = {
        "diary": diary,
        "num_of_moods": num_of_moods,
        "moods": moods,
        "moods_int_values": moods_int_values,
        "mood_triggers": mood_triggers,
        "times_of_the_day": times_of_the_day,
        "habits_data": habits_data,
        "strategies_tried": strategies_tried,
        "articles_read": articles_read,
        "allow_diary_edit": allow_diary_edit,
        "form": form,
    }

    return render(request, "main_app/diary/entry.html", context)


@login_required
def mood_tracker(request):
    user_id = request.user.id

    # Get the current month
    current_month_num = datetime.now().month
    current_month = datetime.now().strftime("%B")
    current_year = datetime.now().year
    three_months_ago = datetime.now() - timedelta(days=90)

    aware_datetime = timezone.make_aware(
        three_months_ago, timezone.get_current_timezone()
    )

    # Query the model for records within the last 3 months
    # results = YourModel.objects.filter(date_field__gte=
        ↪ three_months_ago)

    # great to awful
    mood_count_list = []

    moods = ["great", "good", "okay", "bad", "awful"]
```

```
    for mood in moods:
        # mood_count = MoodEntry.objects.filter(
        # user_id=user_id, mood=mood, date__month=
              ↪ current_month_num
        # ).count()
        mood_count = MoodEntry.objects.filter(
            user_id=user_id, mood=mood, date__gte=aware_datetime
        ).count()
        mood_count_list.append(mood_count)

    total_mood_count = sum(mood_count_list)
    mood_percent_list = []
    if total_mood_count > 0:
        for num in mood_count_list:
            percent = (num / total_mood_count) * 100
            mood_percent_list.append(percent)
    else:
        mood_count_list = [0, 0, 0, 0, 0]
        mood_percent_list = [0, 0, 0, 0, 0]

    context = {
        "current_month": current_month,
        "current_year": current_year,
        "total_mood_count": total_mood_count,
        "mood_count_list": mood_count_list,
        "mood_percent_list": mood_percent_list,
    }

    return render(request, "main_app/mood/mood_tracker.html",
          ↪ context)


@login_required
def add_mood(request):
    # message = "add mood"
    # return redirect("chat", message=message)
    return render(request, "main_app/mood/add_mood.html")


@login_required
def habit_tracker(request):
    user_id = request.user.id

    list_of_habits = ["sleep", "exercise", "water intake"]

    sleep_data = []
    exercise_data = []
    water_intake_data = []

    # Get the current month
    current_month_num = datetime.now().month
    current_month = datetime.now().strftime("%B")
    current_year = datetime.now().year

    three_months_ago = datetime.now() - timedelta(days=90)

    aware_datetime = timezone.make_aware(
        three_months_ago, timezone.get_current_timezone()
    )

    current_date = datetime.now().date()

    # _, num_days = calendar.monthrange(current_year,
          ↪ current_month)
    # dates_of_month = [f"{year}-{month:02d}-{day:02d}" for day
          ↪ in range(1, num_days + 1)]
    # print(len(dates_of_month))

    # habits_value = []
    # for habit in list_of_habits:
    # habit_query = HabitEntry.objects.filter(user_id=user_id,
          ↪ habit=habit)
    # num_of_habit = habit_query.count()
    # if num_of_habit == 0:
    # habits_value.append(habit_query)

    # get all habits of the user
    habits = Habit.objects.filter(users__id=user_id).values_list
          ↪ (
        "habit_name", flat=True
    )
    habits_to_track = list(habits)

    habits_data = {}
    requested_habits = []
    for habit in habits_to_track:
        habit_query = HabitEntry.objects.filter(
            user_id=user_id, habit=habit, date__gte=
                ↪ aware_datetime
        )
        num_of_habit = habit_query.count()
        habit_dates = habit_query.values_list("date", flat=True)
        formatted_dates = [datetime.strftime(date, "%B %d, %Y")
              ↪ for date in habit_dates]
        data = list(habit_query.values_list("data", flat=True))

        string_without_spaces = habit.replace(" ", "_")
        habits_data[string_without_spaces] = {
            "num_of_habit": num_of_habit,
            "dates": formatted_dates,
            "data": data,
        }

        # check if the user can add habit entry
        habit_entry = HabitEntry.objects.filter(
            user__id=user_id, habit=habit, date__date=
                ↪ current_date
        )
        if not habit_entry.exists():
            requested_habits.append(habit)

    if requested_habits != []:
        can_add_habit_entry = True
    else:
        can_add_habit_entry = False

    context = {
        "current_month": current_month,
        "current_year": current_year,
        "habits_data": habits_data,
        "can_add_habit_entry": can_add_habit_entry,
    }
    return render(request, "main_app/habit/habit_tracker.html",
          ↪ context)


@login_required
def add_habit_entry(request):
    message = "add habit entry"
    return redirect("chat", message=message)
    # return render(request, "main_app/habit/add_habit_entry.
          ↪ html")


@csrf_exempt
def get_habits(request):
    if request.method == "POST":
        try:
            # Get the JSON data from the request
            data = json.loads(request.body)

            user_id = data["user_id"]

            # Query all habits of the user
            habits_query = Habit.objects.filter(users__id=user_id
                  ↪ ).values_list(
                "habit_name", flat=True
            )
            habits_list = list(habits_query)

            current_date = datetime.now().date()
            requested_habits = []
            for habit in habits_list:
                habit_entry = HabitEntry.objects.filter(
                    user__id=user_id, habit=habit, date__date=
                        ↪ current_date
                )
                if not habit_entry.exists():
                    requested_habits.append(habit)

            # Return habits data as JSON response
            return JsonResponse({"habits": requested_habits})
        except Exception as e:
            return JsonResponse({"status": "error", "message":
                  ↪ str(e)})

    return JsonResponse({"status": "error", "message": "Invalid
          ↪ request method"})


@csrf_exempt
def save_habit_entry(request):
    if request.method == "POST":
        try:
            # Get the JSON data from the request
            data = json.loads(request.body)

            user_id = data["user_id"]
            habits_dict = data["habits"]

            # Retrieve the user instance
            user = CustomUser.objects.get(id=user_id)

            for habit in habits_dict:
                record = habits_dict[habit]
                current_time = datetime.now()
                habit = habit.replace("_", " ")

                # Create a new mood entry
                habit_entry = HabitEntry(
                    user=user, habit=habit, data=record, date=
                        ↪ current_time
                )

                print(habit_entry)

                # Save the mood entry to the database
                habit_entry.save()
```

```
            # Return a JSON response (optional)
            return JsonResponse({"status": "success"})
        except Exception as e:
            return JsonResponse({"status": "error", "message":
                ↪ str(e)})

    return JsonResponse({"status": "error", "message": "Invalid
        ↪ request method"})


@csrf_exempt
def save_habits(request):
    if request.method == "POST":
        try:
            # Get the JSON data from the request
            data = json.loads(request.body)

            user_id = data["user_id"]
            habits_to_track = data["habits"]

            user = CustomUser.objects.get(id=user_id)
            all_habits = Habit.objects.all()

            for habit in all_habits:
                if habit.habit_name in habits_to_track:
                    habit.users.add(user)
                    habit.save()
                else:
                    habit.users.remove(user)
                    habit.save()

            # Return a JSON response (optional)
            return JsonResponse({"status": "success"})
        except Exception as e:
            return JsonResponse({"status": "error", "message":
                ↪ str(e)})

    return JsonResponse({"status": "error", "message": "Invalid
        ↪ request method"})


@login_required
def edit_habit(request):
    message = "edit habits"
    return redirect("chat", message=message)
    # return render(request, "main_app/habit/add_habit_entry.
        ↪ html")


@login_required
def add_habit(request):
    return render(request, "main_app/habit/add_habit_entry.html
        ↪ ")


@login_required
def strategies(request):
    user = request.user
    strategy_tried = StrategyTried.objects.filter(user=user)
    liked_strategy = strategy_tried.filter(is_liked=True)
    effective_strategy = strategy_tried.filter(is_effective=True
        ↪ )

    context = {
        "strategy_tried": strategy_tried,
        "liked_strategy": liked_strategy,
        "effective_strategy": effective_strategy,
    }

    return render(request, "main_app/coping_strategies/
        ↪ strategy_list.html", context)


@csrf_exempt
def get_strategies(request):
    if request.method == "POST":
        try:
            # Get the JSON data from the request
            data = json.loads(request.body)

            user_id = data["user_id"]

            # Query all habits of the user
            strategies = Strategy.objects.all()
            strategies_tried = StrategyTried.objects.filter(
                user_id=user_id
            ).values_list("strategy__name", flat=True)
            other_strategies = [
                strategy
                for strategy in strategies
                if strategy.name not in strategies_tried
            ]

            # Serialize habits data
            strategies_data = []
            if other_strategies != []:
                for strategy in other_strategies:
                    temp_dict = {
                        "strategy_id": strategy.strategy_id,
                        "strategy_name": strategy.name,
                    }
                    strategies_data.append(temp_dict)

            # Return habits data as JSON response
            return JsonResponse({"strategies": strategies_data})
        except Exception as e:
            return JsonResponse({"status": "error", "message":
                ↪ str(e)})

    return JsonResponse({"status": "error", "message": "Invalid
        ↪ request method"})


@login_required
def view_strategy(request, pk):
    strategy = Strategy.objects.get(strategy_id=pk)

    try:
        strategy_tried = StrategyTried.objects.get(strategy=
            ↪ strategy)
    except StrategyTried.DoesNotExist:
        strategy_tried = None

    context = {
        "strategy": strategy,
        "strategy_tried": strategy_tried,
    }
    return render(request, "main_app/coping_strategies/
        ↪ view_strategy.html", context)


@login_required
def tried_strategy(request, pk):
    strategy_obj = Strategy.objects.get(strategy_id=pk)
    user = request.user
    strategy, created = StrategyTried.objects.get_or_create(
        user=user, strategy=strategy_obj
    )
    return redirect("view_strategy", pk)


@login_required
def liked_strategy(request, pk):
    strategy_obj = Strategy.objects.get(strategy_id=pk)
    user = request.user
    strategy, created = StrategyTried.objects.get_or_create(
        user=user, strategy=strategy_obj
    )
    strategy.is_liked = True
    strategy.save()
    return redirect("view_strategy", pk)


@login_required
def effective_strategy(request, pk):
    strategy_obj = Strategy.objects.get(strategy_id=pk)
    user = request.user
    strategy, created = StrategyTried.objects.get_or_create(
        user=user, strategy=strategy_obj
    )
    strategy.is_effective = True
    strategy.save()
    return redirect("view_strategy", pk)


@login_required
def done_strategy(request, strategy):
    message = f"try strategy '{strategy}' "
    return redirect("chat", message=message)


@login_required
def resources(request):
    user = request.user
    article_read = ArticleRead.objects.filter(user=user)

    context = {
        "article_read": article_read,
    }

    return render(request, "main_app/resources/article_list.html
        ↪ ", context)


@csrf_exempt
def get_articles(request):
    if request.method == "POST":
        try:
            # Get the JSON data from the request
            data = json.loads(request.body)

            user_id = data["user_id"]

            # Query all habits of the user
            articles = Article.objects.all()
            articles_tried = ArticleRead.objects.filter(user_id=
                ↪ user_id).values_list(
```

```
            "article__title", flat=True
        )
        other_articles = [
            article for article in articles if article.title
                ↪ not in articles_tried
        ]

        # Serialize habits data
        articles_data = []
        if other_articles != []:
            for article in other_articles:
                temp_dict = {
                    "article_id": article.article_id,
                    "article_title": article.title,
                }
                articles_data.append(temp_dict)

        # Return habits data as JSON response
        return JsonResponse({"articles": articles_data})
    except Exception as e:
        return JsonResponse({"status": "error", "message":
            ↪ str(e)})

return JsonResponse({"status": "error", "message": "Invalid
    ↪ request method"})


@login_required
def view_article(request, pk):
    article = Article.objects.get(article_id=pk)

    try:
        article_read = ArticleRead.objects.get(article=article)
    except ArticleRead.DoesNotExist:
        article_read = None

    context = {
        "article": article,
        "article_read": article_read,
    }
    return render(request, "main_app/resources/view_article.html
        ↪ ", context)


@login_required
def done_article(request, article):
    message = f"try article '{article}' "
    return redirect("chat", message=message)


@login_required
def read_article(request, pk):
    article_obj = Article.objects.get(article_id=pk)
    user = request.user
    article, created = ArticleRead.objects.get_or_create(user=
        ↪ user, article=article_obj)
    url = article.article.source
    return HttpResponseRedirect(url)


# def signup(request):
# return render(request, "main_app/signup.html")


def signup(request):
    if request.method == "POST":
        email = request.POST["email"]
        password1 = request.POST["password1"]
        password2 = request.POST["password2"]
        if password1 == password2:
            try:
                user = CustomUser.objects.get(email=email)
                messages.error(request, "Email already exists! Go
                    ↪  to sign in.")
            except CustomUser.DoesNotExist:
                new_user = CustomUser.objects.create_user(
                    email=email, username=email, password=
                        ↪ password1
                )
                new_user.save()
                diary = Diary(user=new_user)
                diary.save()
                messages.success(request, "Account successfully
                    ↪ created!")
                user = authenticate(request, email=email,
                    ↪ password=password1)
                login(request, user)
                message = "Hello! I just created an account."
                return redirect("chat", message=message)
        else:
            messages.error(request, "Password didn't match.")
    return render(request, "main_app/signup.html")


@csrf_exempt
def get_user(request):
    if request.method == "POST":
        try:
            # Get the JSON data from the request
            data = json.loads(request.body)

            user_id = data["user_id"]
            user = CustomUser.objects.get(id=user_id)
            user_email = user.email

            # Return habits data as JSON response
            return JsonResponse({"user": user_email})
        except Exception as e:
            return JsonResponse({"status": "error", "message":
                ↪ str(e)})

    return JsonResponse({"status": "error", "message": "Invalid
        ↪ request method"})


@login_required
def account_settings(request):
    return render(request, "main_app/account_settings/
        ↪ account_settings.html")


@login_required
def edit_profile(request):
    user = request.user
    if request.method == "POST":
        form = UserProfileForm(request.POST, request.FILES,
            ↪ instance=user)
        if form.is_valid():
            form.save()
            messages.success(request, "Profile updated!")
            return redirect(
                "account_settings"
            ) # Redirect to the profile page after successful
                ↪ update
    else:
        form = UserProfileForm(instance=user)
    return render(
        request, "main_app/account_settings/edit_profile.html",
            ↪ {"form": form}
    )


@login_required
def notification_settings(request):
    user = request.user
    if request.method == "POST":
        form = TimePromptForm(request.POST, instance=user)
        if form.is_valid():
            form.save()
            messages.success(request, "Changes saved!")
            return redirect(
                "account_settings"
            ) # Redirect to the profile page after successful
                ↪ update
        else:
            messages.error(
                request,
                "Daily check-in cannot be later than the daily
                    ↪ insights.",
            )
    else:
        form = TimePromptForm(instance=user)
    return render(
        request, "main_app/account_settings/
            ↪ notification_settings.html", {"form": form}
    )


@csrf_exempt
def save_notifications_settings(request):
    if request.method == "POST":
        try:
            # Get the JSON data from the request
            data = json.loads(request.body)

            user_id = data["user_id"]
            check_in_time = data["check_in_time"]
            insights_time = data["insights_time"]

            converted_check_in_time = datetime.strptime(
                ↪ check_in_time, "%I:%M %p")
            converted_insights_time = datetime.strptime(
                ↪ insights_time, "%I:%M %p")
            print("\nconverted_check_in_time: ",
                ↪ converted_check_in_time)
            print("\nconverted_insights_time: ",
                ↪ converted_insights_time)

            # Retrieve the user instance
            user = CustomUser.objects.get(id=user_id)
            user.tracker_time_prompt = converted_check_in_time
            print("\ncheckin: ", user.tracker_time_prompt)
            user.insights_time_prompt = converted_insights_time
            print("insights: ", user.insights_time_prompt)
            user.save()

            return JsonResponse({"status": "success"})
```

```
        except Exception as e:
            return JsonResponse({"status": "error", "message":
                ↪ str(e)})

    return JsonResponse({"status": "error", "message": "Invalid
        ↪ request method"})


@login_required
def admin_dashboard(request):
    strategy_count = Strategy.objects.count()
    article_count = Article.objects.count()

    context = {
        "strategy_count": strategy_count,
        "article_count": article_count,
    }
    return render(request, "main_app/admin/admin.html", context)


@login_required
def edit_strategy(request, pk):
    strategy = get_object_or_404(Strategy, pk=pk)
    if request.method == "POST":
        form = StrategyForm(request.POST, instance=strategy)
        if form.is_valid():
            form.save()
            messages.success(request, "Strategy updated!")
            return redirect("admin_view_strategy", pk=pk)
    else:
        form = StrategyForm(instance=strategy)

    context = {"strategy_id": pk, "form": form}
    return render(request, "main_app/admin/edit_strategy.html",
        ↪ context)


@login_required
def add_strategy(request):
    if request.method == "POST":
        form = StrategyForm(request.POST)
        if form.is_valid():
            form.save()
            messages.success(request, "Strategy added!")
            return redirect("strategies_list")
        else:
            messages.error(request, "Strategy not valid!")

    else:
        form = StrategyForm()

    context = {"form": form}
    return render(request, "main_app/admin/add_strategy.html",
        ↪ context)


@login_required
def admin_view_strategy(request, pk):
    strategy = get_object_or_404(Strategy, pk=pk)
    context = {"strategy": strategy}
    return render(request, "main_app/admin/view_strategy.html",
        ↪ context)


@login_required
def edit_article(request, pk):
    article = get_object_or_404(Article, pk=pk)
    if request.method == "POST":
        form = ArticleForm(request.POST, instance=article)
        if form.is_valid():
            form.save()
            messages.success(request, "Article updated!")
            return redirect("admin_view_article", pk=pk)
    else:
        form = ArticleForm(instance=article)

    context = {"article_id": pk, "form": form}
    return render(request, "main_app/admin/edit_article.html",
        ↪ context)


@login_required
def add_article(request):
    if request.method == "POST":
        form = ArticleForm(request.POST)
        if form.is_valid():
            form.save()
            messages.success(request, "Article added!")
            return redirect("articles_list")
        else:
            messages.error(request, "Article not valid!")

    else:
        form = ArticleForm()

    context = {"form": form}
    return render(request, "main_app/admin/add_article.html",
        ↪ context)


@login_required
def admin_view_article(request, pk):
    article = get_object_or_404(Article, pk=pk)
    context = {"article": article}
    return render(request, "main_app/admin/view_article.html",
        ↪ context)


@login_required
def strategies_list(request):
    strategies = Strategy.objects.all()

    context = {
        "strategies": strategies,
    }
    return render(request, "main_app/admin/strategies_list.html
        ↪ ", context)


@login_required
def articles_list(request):
    articles = Article.objects.all()

    context = {
        "articles": articles,
    }
    return render(request, "main_app/admin/articles_list.html",
        ↪ context)


def delete_strategy(request, pk):
    strategy = get_object_or_404(Strategy, pk=pk)
    strategy.delete()
    message = f"Strategy '{strategy.name}' is deleted!"
    messages.error(request, message)
    return redirect("strategies_list")


def delete_article(request, pk):
    article = get_object_or_404(Article, pk=pk)
    article.delete()
    message = f"Article '{article.title}' is deleted!"
    messages.error(request, message)
    return redirect("articles_list")
```

Listing 11: templates/main_app/base.html

```
{% load static %}
{% load webpush_notifications %}

<!DOCTYPE html>
<html>
<head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width,
            ↪ initial-scale=1">

        <!-- ICONS -->
        <script src="https://code.iconify.design/3/3.1.0/iconify
            ↪ .min.js"></script>

        <!-- CSS -->
        <link rel="stylesheet" href="https://www.w3schools.com/
            ↪ w3css/4/w3.css">
        <link rel="stylesheet" href="{% static 'css/style.css'
            ↪ %}">

        <!-- FONTS -->
        <link rel="stylesheet" href="https://fonts.googleapis.
            ↪ com/css?family=Poppins">
        <link rel="stylesheet" href="https://fonts.googleapis.
            ↪ com/css2?family=K2D">

        <!-- CHART.JS -->
        <script src="https://cdn.jsdelivr.net/npm/chart.js"></
            ↪ script>

        <!-- JQUERY -->
        <script src="https://code.jquery.com/jquery-3.7.0.js"
            ↪ integrity="sha256-
            ↪ JlqSTELeR4TLqP0OG9dxM7yDPqX1ox/HfgiSLBj8+kM="
            ↪ crossorigin="anonymous"></script>

        {% webpush_header %}

        <title>ACAPP</title>
        <link rel="icon" type="image/x-icon" href="{% static '
            ↪ img/logo.svg' %}">

</head>
<body>

        {% block content %}
        {% endblock %}

        <!-- JS -->
```

```
    <script type="text/javascript" src="{% static 'js/main.
        ↪ js' %}" ></script>

</body>
</html>
```

Listing 12: templates/main_app/chat.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

{% include "main_app/sidebar.html" %}

<div class="w3-main">

<div class="w3-padding-16" id="chat-container" style="height:
    ↪ 580px; overflow-y: scroll;">

{% for message in conversation %}

{% if message.sender == "bot" %}
{% if message.text != None %}
<div class="w3-container w3-padding">
<div class="w3-cell bot-image-col">
<img src="{% static 'img/bot.png' %}" alt="profile pic" class="
    ↪ bot-image">
</div>
<div class="w3-cell">
<div class="bot-message-div message-div w3-round w3-padding-
    ↪ large">
<span class="">{{ message.text }}</span>
</div>
<span class="w3-tiny w3-text-grey" style="margin-left: 8px;">{{
    ↪  message.timestamp }}</span>
</div>
</div>
{% endif %}

{% elif message.sender == "user" %}

<div class="w3-container w3-padding">
<div class="w3-right">
<div class="w3-cell">
<div class="user-message-div message-div w3-round w3-padding-
    ↪ large">
<span class="">{{ message.text }}</span>
</div>
<span class="w3-tiny w3-text-grey" style="margin-left: 8px;">{{
    ↪  message.timestamp }}</span>
</div>
<div class="w3-cell user-image-col">
{% if request.user.profile_image %}
<img src="{{ request.user.profile_image.url }}" alt="profile
    ↪ image" class="bot-image">
{% else %}
<img src="{% static 'img/user.png' %}" alt="profile image"
    ↪ class="bot-image">
{% endif %}
</div>
</div>
</div>

{% endif %}

{% endfor %}

</div>

<!-- end of main -->
</div>

<script>
const userId = '{{ request.user.id }}';
const userMessage = '{{ user_message }}';
console.log(userMessage)

// Function to display the message sent by the user
function displayUserMessage(message) {

var currentTime = getTime()

// user message
chatContainer.innerHTML +=
`<div class="w3-container w3-padding">
<div class="w3-right">
<div class="w3-cell">
<div class="user-message-div message-div w3-round w3-padding-
    ↪ large">
<span class="">${message}</span>
</div>
<span class="w3-tiny w3-text-grey" style="margin-left: 8px;">${
    ↪ currentTime}</span>
</div>
<div class="w3-cell user-image-col">
{% if request.user.profile_image %}
<img src="{{ request.user.profile_image.url }}" alt="profile
    ↪ image" class="user-image">
{% else %}
<img src="{% static 'img/user.png' %}" alt="profile image"
    ↪ class="user-image">
{% endif %}
</div>
</div>
</div>`

scrollToLatestMessage();
}

// Function to send user message to the bot
function sendMessageToBot(message, csrf_token) {

const userData = {"userid": userId, "message": message};

// Send the message to the bot using a POST request
fetch('/send_message/', {
method: 'POST',
headers: {
'Content-Type': 'application/json',
'X-CSRFToken': csrf_token
},
body: JSON.stringify(userData)
})
.then(response => response.json())
.then(data => {

var responses_list = data.responses_list;
// console.log(responses_list)

var currentTime = getTime();

// bot message
if(responses_list[0].text){
chatContainer.innerHTML +=
`<div class="w3-container w3-padding">
<div class="w3-cell bot-image-col">
<img src="{% static 'img/bot.png' %}" alt="profile pic" class="
    ↪ bot-image">
</div>
<div class="w3-cell">
<div class="bot-message-div message-div w3-round w3-padding-
    ↪ large">
<span class="">${responses_list[0].text}</span>
</div>
<span class="w3-tiny w3-text-grey" style="margin-left: 8px;">${
    ↪ currentTime}</span>
</div>
</div>`
}

// choice buttons
if(responses_list[0].html_buttons_str){
chatContainer.innerHTML +=
`<div class="w3-container choices-div">
<div class="w3-cell bot-image-col"></div>
<div class="w3-cell">
<div class="chat-btn-div">
<form class="form" method="POST">
{% csrf_token %}
${responses_list[0].html_buttons_str}
</form>
</div>
</div>
</div>`
}

// checkmark choice buttons
if(responses_list[0].choices){
var habits = responses_list[0].choices
var btnList = []
for(var i = 0; i < habits.length; i++){
var habit = habits[i];
checkmarkBtn = `<button class="w3-button checkmark-button"
    ↪ value="${habit.payload}" onclick="toggleCheckmark(this
    ↪ )">${habit.title}</button>`
btnList.push(checkmarkBtn)
}

var joinedString = btnList.join(" ");

chatContainer.innerHTML +=
`<div class="w3-container choices-div">
<div class="w3-cell bot-image-col"></div>
<div class="w3-cell">
<div class="chat-btn-div">
${joinedString}
</div>

<div class="chat-btn-div w3-margin-top">
<form class="form" method="POST">
{% csrf_token %}
<input type="hidden" id="checkedValues" name="checkedValues">
<button type="submit" class="w3-button chat-btn" id="savebtn"
    ↪ value="save habits" disabled="true">Save</button>
<button type="submit" class="w3-button chat-btn" id="savebtn"
```

```
      ↪ value="cancel">Cancel</button>
</form>
</div>
</div>
</div>`
}

// strategy slideshow
if(responses_list[0].slides){
var slides = responses_list[0].slides
var slidesType = responses_list[0].type
// Loop over the items and generate HTML code
let html_links = '';
for (var dict of slides) {
var id = dict.id;
var url = '/' + slidesType + '/' + id + '/'
html_links += `<a href="${url}" class="w3-container w3-display-
      ↪ container w3-round w3-padding mySlides strategy-box">
       <div class="w3-cell w3-cell-middle strategy-image-col">
       <img src="{% static 'img/strategy.png' %}" alt="strategy
            ↪  pic" class="strategy-image">
       </div>
       <div class="w3-cell w3-cell-middle strategy-text">
       <span class="chart-name">${dict.text}</span>
       </div>
       </a>`
}

chatContainer.innerHTML +=
`<div class="w3-container w3-padding strategy-div">
<div class="w3-cell bot-image-col"></div>
<div class="w3-cell w3-cell-middle ">
<div class="w3-display-container strategy-box-div">
${html_links}
<button class="w3-button w3-display-left slideshow-btn" onclick
      ↪ ="plusDivs(-1)">&#10094;</button>
<button class="w3-button w3-display-right slideshow-btn"
      ↪ onclick="plusDivs(1)">&#10095;</button>
</div>
</div>
</div>`

showDivs(slideIndex);

}

// notifications time input
if(responses_list[0].type){
if(responses_list[0].type == "notifications"){

chatContainer.innerHTML +=
`<div class="w3-container choices-div">
<div class="w3-cell bot-image-col"></div>
<div class="w3-cell">
<div class="chat-btn-div">
<input class="w3-input field-input" type="time" id="timeInput"
      ↪ name="timeInput" onchange="setTime()"><br>
<form class="form" method="POST">
{% csrf_token %}
<button type="submit" class="w3-button chat-btn" id="savebtn"
      ↪ value="">Save</button>
<button type="submit" class="w3-button chat-btn" id="savebtn"
      ↪ value="cancel">Cancel</button>
</form>
</div>
</div>
</div>`

} else if(responses_list[0].type == "slider"){
console.log("let me in")
var response = responses_list[0]
var max_value = response.max
var min_value = response.min

chatContainer.innerHTML +=
`<div class="w3-container w3-padding choices-div">
<div class="w3-cell bot-image-col">
</div>
<div class="w3-cell w3-padding w3-center">
<form class="form" method="POST">
              {% csrf_token %}
              <div class="slider-container w3-margin-bottom">
       <input type="range" id="slider" class="slider" min="${
            ↪ min_value}" max="${max_value}" value="${
            ↪ min_value}" step="1" onchange="setHabit()">
       <label for="slider" class="">
              <span id="sliderText" class="w3-large">${
                   ↪ min_value}</span>
       </label>
       </div>
       <button type="submit" class="w3-button chat-btn" id="
            ↪ savebtn" value="">Save</button>
       <button type="submit" class="w3-button chat-btn" id="
            ↪ savebtn" value="cancel">Cancel</button>
       </form>
</div>
</div>`
}
}

scrollToLatestMessage();

if(responses_list.length > 0){
for(var i = 1; i < responses_list.length; i++){
var response = responses_list[i];

// bot message
if(response.text){
chatContainer.innerHTML +=
`<div class="w3-container w3-padding bot-div">
<div class="w3-cell bot-image-col">
</div>
<div class="w3-cell">
<div class="bot-message-div message-div w3-round w3-padding-
      ↪ large">
<span class="">${response.text}</span>
</div>
<span class="w3-tiny w3-text-grey" style="margin-left: 8px;">${
      ↪ currentTime}</span>
</div>
</div>`
}

// choice buttons
if(response.html_buttons_str){
chatContainer.innerHTML +=
`<div class="w3-container choices-div">
<div class="w3-cell bot-image-col"></div>
<div class="w3-cell">
<div class="chat-btn-div">
<form class="form" method="POST">
{% csrf_token %}
${response.html_buttons_str}
</form>
</div>
</div>
</div>`
}

// checkmark choice buttons
if(response.choices){
var habits = response.choices
var btnList = []
for(var i = 0; i < habits.length; i++){
  var habit = habits[i];
  checkmarkBtn = `<button class="w3-button checkmark-button"
       ↪ value="${habit.payload}" onclick="toggleCheckmark(
       ↪ this)">${habit.title}</button>`
  btnList.push(checkmarkBtn)
}
var joinedString = btnList.join(" ");

chatContainer.innerHTML +=
`<div class="w3-container choices-div">
<div class="w3-cell bot-image-col"></div>
<div class="w3-cell">
  <div class="chat-btn-div">
  ${joinedString}
</div>

  <div class="chat-btn-div w3-margin-top">
  <form class="form" method="POST">
  {% csrf_token %}
  <input type="hidden" id="checkedValues" name="checkedValues">
  <button type="submit" class="w3-button chat-btn" id="savebtn"
       ↪ value="save habits">Save</button>
  <button type="submit" class="w3-button chat-btn" id="savebtn"
       ↪ value="cancel">Cancel</button>
  </form>
  </div>
</div>
</div>`
}

// strategy/articles slideshow
if(response.slides){
var slides = response.slides
var slidesType = response.type
// Loop over the items and generate HTML code
let html_links = '';
for (var dict of slides) {
var id = dict.id;
var url = '/' + slidesType + '/' + id + '/'

if(slidesType == "strategies"){
img = `<img src="{% static 'img/strategy.png' %}" alt="strategy
      ↪  pic" class="strategy-image">`
}else if(slidesType == "resources"){
img = `<img src="{% static 'img/resource.png' %}" alt="strategy
      ↪  pic" class="strategy-image">`
}

html_links += `<a href="${url}" class="w3-container w3-display-
      ↪ container w3-round w3-padding mySlides strategy-box">
           <div class="w3-cell w3-cell-middle strategy-image-col
                ↪ ">
           ${img}
           </div>
           <div class="w3-cell w3-cell-middle strategy-text">
```

```
            <span class="chart-name">${dict.text}</span>
            </div>
            </a>‘
}

chatContainer.innerHTML +=
‘<div class="w3-container w3-padding strategy-div">
<div class="w3-cell bot-image-col"></div>
<div class="w3-cell w3-cell-middle ">
<div class="w3-display-container strategy-box-div">
${html_links}
<button class="w3-button w3-display-left slideshow-btn" onclick
      ↪ ="plusDivs(-1)">&#10094;</button>
<button class="w3-button w3-display-right slideshow-btn"
      ↪ onclick="plusDivs(1)">&#10095;</button>
</div>
</div>
</div>‘

showDivs(slideIndex);

}

scrollToLatestMessage();
}
}

if(data.url){
// 3 second delay
setTimeout(function(){
window.location.href = "http://127.0.0.1:8000" + data.url;
}, 3000);
}

})
.catch(error => console.log(error));

}

function getTime(){
var date = new Date();
var hours = date.getHours();
var minutes = date.getMinutes();
var ampm = hours >= 12 ? ’PM’ : ’AM’;

// Convert to 12-hour format
hours = hours % 12;
hours = hours ? hours : 12;
var currentTime = hours + ":" + (minutes < 10 ? ’0’ : ’’) +
      ↪ minutes + " " + ampm
return currentTime;
}

</script>
<script type="text/javascript" src="{% static ’js/chat.js’ %}"
      ↪ ></script>
<script type="text/javascript" src="{% static ’js/slideshow.js’
      ↪  %}" ></script>



{% endblock %}
```

Listing 13: templates/-main_app/home.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

{% include "main_app/sidebar.html" %}

<div class="w3-main" >
<!-- style="background: #D1FFA2; height: 100vh;" -->

{% if messages %}
{% for message in messages %}
{% if message.tags == "success" %}
<div class="message success">
{{ message }}
</div>
{% else %}
<div class="message error">
{{ message }}
</div>
{% endif %}
{% endfor %}
{% endif %}

<div class="w3-container w3-auto home-div w3-round">

<div class="w3-container w3-padding w3-margin-top">
<div class="w3-cell home-bot-image-col">
<img src="{% static ’img/bot.png’ %}" alt="profile pic" class="
      ↪ home-bot-image">
</div>
<div class="w3-cell">
<div class="bot-message-div message-div w3-round w3-padding-
      ↪ large">
<span class="w3-large">Hi, {% if request.user.first_name %}{{
      ↪ request.user.first_name}}{% else %}{{request.user.
      ↪ email}}{% endif %}!
</span>
</div>
</div>
</div>
<div class="w3-container w3-padding">
<div class="w3-cell home-bot-image-col">
</div>
<div class="w3-cell">
<div class="bot-message-div message-div w3-round w3-padding-
      ↪ large">
<span class="">
I’m Kaya, your personal bot assistant. </span>
</div>
</div>
</div>
<div class="w3-container w3-padding">
<div class="w3-cell home-bot-image-col">
</div>
<div class="w3-cell">
<div class="bot-message-div message-div w3-round w3-padding-
      ↪ large">
<span class="">
If you need any assistance or have any questions, don’t
      ↪ hesitate to reach out. Just click on ’Talk to Kaya’
      ↪ and let’s get started! </span>
</div>
</div>
</div>

<br>
<hr>
<div class="w3-center">
<a href="{% url ’talk_to_kaya’ %}" class="w3-button add-btn w3-
      ↪ stretch w3-auto w3-center">
<span class="w3-xlarge">Start a Conversation</span>
</a>
</div>
</div>

<footer class="w3-container w3-auto w3-bottom" style="
      ↪ background: #D1FFA2; height:30px">
<p class="w3-center w3-tiny" style="color: #6D0AD3">&copy;
      ↪ ACAPP All rights reserved.</p>
</footer>

</div>

{% endblock %}
```

Listing 14: templates/-main_app/sidebar.html

```
{% load static %}


<!-- Sidebar/menu -->
<nav class="w3-sidebar w3-red w3-collapse w3-tiny w3-padding"
      ↪ id="mySidebar">

<br>

<a href="javascript:void(0)" onclick="w3_close()" class="w3-
      ↪ button w3-hide-large w3-display-topright" style="font-
      ↪ size:22px">
<span class="iconify nav-iconify" data-icon="eva:arrow-ios-back
      ↪ -outline"></span>
</a>

<div class="w3-container">
<div class="w3-cell w3-cell-middle profile-div">
{% if request.user.profile_image %}
<img src="{{ request.user.profile_image.url }}" alt="profile
      ↪ image" class="responsive-profile">
{% else %}
<img src="{% static ’img/user.png’ %}" alt="profile image"
      ↪ class="responsive-profile">
{% endif %}
</div>
<div class="w3-cell w3-cell-middle name-div">
<h6 class="w3-tiny">HELLO </h6>
<h4><b>
{% if request.user.first_name %}
{{ request.user.first_name}}
{% else %}
{{request.user.email}}
{% endif %}
</b></h4>
</div>
</div>
```

```
<hr class="w3-margin">


<div class="w3-bar-block menu-items">
<h6 class="w3-margin-left">MENU</h6>
{% if request.user.is_staff %}
<a href="{% url 'admin_dashboard' %}" class="w3-bar-item w3-
    ↪ button">
<span class="iconify nav-iconify" data-icon="humbleicons:chat
    ↪ "></span>
<span class="item-name">Dashboard</span>
</a>
<a href="{% url 'strategies_list' %}" class="w3-bar-item w3-
    ↪ button">
<span class="iconify nav-iconify" data-icon="tabler:bulb"></
    ↪ span>
<span class="item-name">Well-Being Strategies</span>
</a>
<a href="{% url 'articles_list' %}" class="w3-bar-item w3-
    ↪ button">
<span class="iconify nav-iconify" data-icon="akar-icons:book-
    ↪ open"></span>
<span class="item-name">Online Articles</span>
</a>
{% else %}
<a href="{% url 'home' %}" class="w3-bar-item w3-button">
<span class="iconify nav-iconify" data-icon="humbleicons:chat
    ↪ "></span>
<span class="item-name">Talk to Kaya</span>
</a>
<a href="{% url 'strategies' %}" class="w3-bar-item w3-button">
<span class="iconify nav-iconify" data-icon="tabler:bulb"></
    ↪ span>
<span class="item-name">My Strategies</span>
</a>
<a href="{% url 'mood_tracker' %}" class="w3-bar-item w3-button
    ↪ ">
<span class="iconify nav-iconify" data-icon="material-symbols:
    ↪ calendar-today-outline-rounded"></span>
<span class="item-name">Moods</span>
</a>
<a href="{% url 'habit_tracker' %}" class="w3-bar-item w3-
    ↪ button">
<span class="iconify nav-iconify" data-icon="tabler:chart-bar
    ↪ "></span>
<span class="item-name">Habits</span>
</a>
<a href="{% url 'diary_entry_list' %}" class="w3-bar-item w3-
    ↪ button">
<span class="iconify nav-iconify" data-icon="uil:diary"></span>
<span class="item-name">Diary</span>
</a>
<a href="{% url 'resources' %}" class="w3-bar-item w3-button">
<span class="iconify nav-iconify" data-icon="akar-icons:book-
    ↪ open"></span>
<span class="item-name">Resources</span>
</a>
{% endif %}
</div>



<div class="w3-bar-block settings-items">
<h6 class="w3-margin-left">SETTINGS</h6>
{% if not request.user.is_staff %}
<a href="{% url 'account_settings' %} " class="w3-bar-item w3-
    ↪ button w3-hover-white">
<span class="iconify nav-iconify" data-icon="tabler:settings
    ↪ "></span>
<span class="item-name">Account Settings</span>
</a>
{% endif %}
<a href="{% url 'signout' %}" class="w3-bar-item w3-button w3-
    ↪ hover-white">
<span class="iconify nav-iconify" data-icon="humbleicons:logout
    ↪ "></span>
<span class="item-name">Sign Out</span>
</a>

</nav>

<!-- header on small screens -->
<header class="w3-top w3-hide-large w3-red w3-xlarge" id="
    ↪ header_small">
<a href="javascript:void(0)" class="w3-button w3-red w3-margin-
    ↪ right" onclick="w3_open()">

</a>
<span class="w3-cell-middle">ACAPP</span>
</header>

<!-- header on large screens -->
<header class="w3-top w3-hide-small w3-hide-medium w3-large w3-
    ↪ padding w3-center" id="header_large">
<img src="{% static 'img/logo.svg' %}" alt="logo" class="logo-
    ↪ header">
<span>ACAPP</span>
</header>

<!-- Overlay effect when opening sidebar on small screens -->
<div class="w3-overlay w3-hide-large" onclick="w3_close()"
    ↪ title="close side menu" id="myOverlay"></div>
```

Listing 15: templates/main_app/signin.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

<div class="w3-container w3-padding w3-center w3-block w3-
    ↪ display-middle">

<img src="{% static 'img/logo.svg' %}" alt="logo" class="logo-
    ↪ signin">

<h2><b>SIGN IN</b></h2>

<div class="w3-container w3-auto w3-center signin-form-div">
{% if messages %}
{% for message in messages %}
{% if message.tags == "success" %}
<div class="message success">
{{ message }}
</div>
{% else %}
<div class="message error">
{{ message }}
</div>
{% endif %}
{% endfor %}
{% endif %}

<form method="post" action="{% url 'signin' %}">
{% csrf_token %}

<h6 class="w3-left field-name">Email</h6>
<input class="w3-input field-input" type="email" name="email"
    ↪ id="id_email" placeholder="johndoe@example.com">
<br>

<h6 class="w3-left field-name">Password</h6>
<input class="w3-input field-input" type="password" name="
    ↪ password" id="id_password" placeholder="******">

<a href="#" class="w3-right forgot-pw-name">
<span>Forgot Password?</span>
</a>

<br>

<button class="w3-button w3-margin-top signin-btn" type="submit
    ↪ "><span>Sign in</span></button>

</form>

<br><br>

<h6 class="no-acct-text">Don't have an account?</h6>
<a href="{% url 'signup' %}" class="signup-link">
<span>Sign up now</span>
</a>

</div>

</div>

{% endblock %}
```

Listing 16: templates/main_app/signout.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

<div class="w3-container w3-padding w3-center w3-block w3-
    ↪ display-middle">

<div class="w3-container w3-auto w3-center signout-div">

<p>Thank you for using this app.</p>

<h5><b>You have successfully signed out.</b></h5>

<br><br>

<p>You can always come back and log in again to access all of
    ↪ the resources and features available in this app.  </p
    ↪ >
```

```
<br><br>

<a href="{% url 'signin' %}" class="signup-link">
<p>Sign in again</p>
</a>

<br><br><br>

<img src="{% static 'img/logo.svg' %}" alt="logo" class="logo-
      ↪ signin">

<br>

<h6 class="no-acct-text">Need support?</h6>
<a href="#" class="signup-link">
<span>Contact Us</span>
</a>

</div>

</div>

{% endblock %}
```

Listing 17: templates/-main_app/signup.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

<div class="w3-container w3-padding w3-center w3-block w3-
      ↪ display-middle">

<img src="{% static 'img/logo.svg' %}" alt="logo" class="logo-
      ↪ signin">

<h2><b>CREATE ACCOUNT</b></h2>

<div class="w3-container w3-auto w3-center signin-form-div">

{% if messages %}
{% for message in messages %}
{% if message.tags == "success" %}
<div class="message success">
{{ message }}
</div>
{% else %}
<div class="message error">
{{ message }}
</div>
{% endif %}
{% endfor %}
{% endif %}

<form method="post" action="{% url 'signup' %}">
{% csrf_token %}

<h6 class="w3-left field-name">Email</h6>
<input class="w3-input field-input" type="email" name="email"
      ↪ id="id_email" placeholder="johndoe@example.com">
<br>

<h6 class="w3-left field-name">Password</h6>
<input class="w3-input field-input" type="password" name="
      ↪ password1" id="id_password1" placeholder="******">

<h6 class="w3-left field-name">Confirm Password</h6>
<input class="w3-input field-input" type="password" name="
      ↪ password2" id="id_password2" placeholder="******">

<br>

<button class="w3-button w3-margin-top signin-btn" type="submit
      ↪ "><span>Sign up</span></button>

</form>

<br><br>

</div>

</div>

{% endblock %}
```

Listing 18: templates/-main_app/article_list.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

{% include "main_app/sidebar.html" %}

<body style="background: #F5F5F5;">

<div class="w3-main w3-padding-16 w3-container">

<a href="{% url 'home' %}">
<h6 class="w3-small">
<span class="iconify back-icon" data-icon="material-symbols:
      ↪ arrow-back-rounded"></span>
back
</h6>
</a>

<div>
<h3>
<span class="iconify title-icon" data-icon="akar-icons:book-
      ↪ open"></span>
<span class="w3-large"><b>Resources</b></span>
</h3>
</div>

<div class="w3-row-padding w3-section w3-container w3-stretch">
<span class="w3-left w3-text-grey"><b>Articles Read</b></span>
</div>

{% if article_read %}
<div class="w3-row-padding w3-stretch">
{% for article in article_read %}
<div class="w3-col s12 m12 l3 w3-margin-bottom">
<a href="{% url 'view_article' article.article.article_id %}"
      ↪ class="w3-button w3-card w3-container ">
<h6 class="chart-name w3-left-align">{{article.article.title
      ↪ }}</h6>
</a>
</div>
{% endfor %}
</div>
{% else %}
<div class="w3-row-padding w3-container w3-section w3-stretch">
<div class="w3-container w3-round" style="background: #e3dede">
<h6 class="w3-small">No article marked as read.</h6>
</div>
</div>
{% endif %}

<div class="w3-row-padding w3-section w3-container w3-stretch">
<a href="{% url 'chat' message='read articles' %}" class="w3-
      ↪ button add-btn w3-block" >
read articles
</a>
</div>

</div>

</body>

{% endblock %}
```

Listing 19: templates/-main_app/view_article.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

{% include "main_app/sidebar.html" %}

<body style="background: #F5F5F5;">

<div class="w3-main w3-padding-16 w3-container">

{% if request.user.is_staff == False %}
<a href="{% url 'done_article' article.title %}">
<h6 class="w3-small">
<span class="iconify back-icon" data-icon="material-symbols:
      ↪ arrow-back-rounded"></span>
back to Kaya
</h6>
</a>
{% else %}
<a href="{% url 'articles_list' %}">
<h6 class="w3-small">
<span class="iconify back-icon" data-icon="material-symbols:
      ↪ arrow-back-rounded"></span>
back
</h6>
</a>
{% endif %}

<div class="w3-card w3-display-container w3-round view-strategy
      ↪ -div">

<div style="height: 90px; background: #00CF95; border-radius: 4
      ↪ px 4px 0px 0px;"></div>
```

```
<div class="view-strategy-img-div w3-display-topmiddle" style="
      ↪ margin-top:40px">
<img src="{% static 'img/resource.png' %}" alt="article pic"
      ↪ class="strategy-image">
</div>

<div class="w3-padding-large w3-padding-top-48">

<div>
<h3 class="w3-center no-margin">
<span class="w3-large"><b>{{article.title}}</b></span>
</h3>

<p class="w3-justify">
<span class="w3-text-grey w3-tiny">SUMMMARY</span><br>
{{article.summary}}
</p>

</div>

<div class="w3-margin-top">

<div class="w3-row-padding w3-section w3-container w3-stretch">

<div class="w3-col s12 m12 w3-margin-bottom">
<a href="{% url 'read_article' article.article_id %}" class="w3
      ↪ -button add-btn w3-block" >
Read the article
</a>
</div>

</div>

</div>

</div>

</div>


<!-- end of main -->
</div>

</body>

{% endblock %}
```

Listing 20: templates/-main_app/mood_tracker.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

{% include "main_app/sidebar.html" %}

<body style="background: #F5F5F5;">

<div class="w3-main w3-padding-16 w3-container">

<a href="{% url 'home' %}">
<h6 class="w3-small">
<span class="iconify back-icon" data-icon="material-symbols:
      ↪ arrow-back-rounded"></span>
back
</h6>
</a>

<div>
<h3>
<span class="iconify title-icon" data-icon="material-symbols:
      ↪ calendar-today-outline-rounded"></span>
<span class="w3-large"><b>Mood Tracker</b></span>
<span class="w3-small w3-text-grey"><b>(Last 3 Months)</b></
      ↪ span>
</h3>
</div>

{% if total_mood_count == 0 %}

<div class="w3-row-padding w3-container w3-section w3-stretch">
<div class="w3-container w3-card w3-round ">
<h6 class="chart-name"><b>Mood Count</b></h6>
<h6 class="w3-text-grey"> Not enough data </h6>
</div>
</div>

{% else %}

<div class="w3-row-padding w3-container w3-section w3-stretch">
<div class="w3-container w3-card w3-round ">
<h6 class="chart-name"><b>Mood Count</b></h6>
<div class="chart-div">
<canvas id="moodBarChart" style="width: 100%; height: 250px;
      ↪ margin-bottom: 10px"></canvas>
</div>
</div>
</div>

<div class="w3-row-padding w3-container w3-stretch">

<div class="w3-cell-row w3-card w3-round">
<div class="w3-cell w3-center w3-container w3-padding w3-text-
      ↪ grey">
<h6 class="w3-small no-margin">Awful</h6>
<h6 class="no-margin">
<span class="iconify mood-tracker-icon mood-wrrr-icon" data-
      ↪ icon="tabler:mood-wrrr"></span>
</h6>
<h6 class="w3-small no-margin">{{ mood_percent_list.4|
      ↪ floatformat:2 }}%</h6>
</div>

<div class="w3-cell w3-center w3-container w3-padding w3-text-
      ↪ grey">
<h6 class="w3-small no-margin">Bad</h6>
<h6 class="no-margin">
<span class="iconify mood-tracker-icon mood-sad-icon" data-icon
      ↪ ="tabler:mood-sad"></span>
</h6>
<h6 class="w3-small no-margin">{{ mood_percent_list.3|
      ↪ floatformat:2 }}%</h6>
</div>

<div class="w3-cell w3-center w3-container w3-padding w3-text-
      ↪ grey">
<h6 class="w3-small no-margin">Okay</h6>
<h6 class="no-margin">
<span class="iconify mood-tracker-icon mood-empty-icon" data-
      ↪ icon="tabler:mood-empty"></span>
</h6>
<h6 class="w3-small no-margin">{{ mood_percent_list.2|
      ↪ floatformat:2 }}%</h6>
</div>

<div class="w3-cell w3-center w3-container w3-padding w3-text-
      ↪ grey">
<h6 class="w3-small no-margin">Good</h6>
<h6 class="no-margin">
<span class="iconify mood-tracker-icon mood-smile-icon" data-
      ↪ icon="tabler:mood-smile"></span>
</h6>
<h6 class="w3-small no-margin">{{ mood_percent_list.1|
      ↪ floatformat:2 }}%</h6>
</div>

<div class="w3-cell w3-center w3-container w3-padding w3-text-
      ↪ grey">
<h6 class="w3-small no-margin">Great</h6>
<h6 class="no-margin">
<span class="iconify mood-tracker-icon mood-smile-beam-icon"
      ↪ data-icon="tabler:mood-smile-beam"></span>
</h6>
<h6 class="w3-small no-margin">{{ mood_percent_list.0|
      ↪ floatformat:2 }}%</h6>
</div>

</div>
</div>

{% endif %}

<div class="w3-row-padding w3-section w3-container w3-stretch">
<a href="{% url 'chat' 'add mood' %}" class="w3-button add-btn
      ↪ w3-block" >
Add mood
</a>
</div>

</div>
</body>


<!-- JS -->
<script>
const moodCount = {{mood_count_list}}
</script>
<script type="text/javascript" src="{% static 'js/mood.js' %}"
      ↪ ></script>



{% endblock %}
```

Listing 21: templates/-main_app/habit_tracker.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}
```

```
{% include "main_app/sidebar.html" %}

<body style="background: #F5F5F5;">

<div class="w3-main w3-padding-16 w3-container">

<a href="{% url 'home' %}">
<h6 class="w3-small">
<span class="iconify back-icon" data-icon="material-symbols:
      ↪ arrow-back-rounded"></span>
back
</h6>
</a>

<div>
<h3>
<span class="iconify title-icon" data-icon="tabler:chart-bar
      ↪ "></span>
<span class="w3-large"><b>Habit Tracker</b></span>
<span class="w3-small w3-text-grey"><b>(Last 3 Months)</b></
      ↪ span>
</h3>
</div>

{% if habits_data %}

{% if "sleep" in habits_data %}
{% if habits_data.sleep.num_of_habit == 0 %}

<div class="w3-row-padding w3-container w3-section w3-stretch">
<div class="w3-container w3-card w3-round ">
<h6 class="chart-name"><b>Sleep</b></h6>
<h6 class="w3-text-grey"> Not enough data </h6>
</div>
</div>

{% else %}
<div class="w3-row-padding w3-container w3-section w3-stretch">
<div class="w3-container w3-card w3-round ">
<h6 class="chart-name"><b>Sleep</b></h6>

<div class="w3-container w3-display-container">
<div class="w3-row w3-display-top w3-right">
<div class="w3-show-inline-block" style="width: 16px; height:16
      ↪ px; background: rgba(119,221,118, 0.2); border: 1px
      ↪ solid rgb(119,221,118); vertical-align: middle;"></div
      ↪ >
<span class="w3-show-inline-block w3-small" style="vertical-
      ↪ align: middle;">Recommended</span>

<div class="w3-show-inline-block" style="width: 16px; height:16
      ↪ px; background: rgba(255,105,98, 0.2); border: 1px
      ↪ solid rgb(255, 105, 98); vertical-align: middle;
      ↪ margin-left: 6px;"></div>
<span class="w3-show-inline-block w3-small" style="vertical-
      ↪ align: middle;">Below Recommended</span>
</div>
</div>

<div class="chart-div">
<canvas id="sleepBarChart" style="width: 100%; height: 250px;
      ↪ margin-bottom: 10px"></canvas>
</div>
</div>
</div>

{% endif %}
{% endif %}

{% if "exercise" in habits_data %}
{% if habits_data.exercise.num_of_habit == 0 %}

<div class="w3-row-padding w3-container w3-section w3-stretch">
<div class="w3-container w3-card w3-round ">
<h6 class="chart-name"><b>Exercise</b></h6>
<h6 class="w3-text-grey"> Not enough data </h6>
</div>
</div>

{% else %}

<div class="w3-row-padding w3-container w3-section w3-stretch">
<div class="w3-container w3-card w3-round ">
<h6 class="chart-name"><b>Exercise</b></h6>

<div class="w3-container w3-display-container">
<div class="w3-row w3-display-top w3-right">
<div class="w3-show-inline-block" style="width: 16px; height:16
      ↪ px; background: rgba(119,221,118, 0.2); border: 1px
      ↪ solid rgb(119,221,118); vertical-align: middle; "></
      ↪ div>
<span class="w3-show-inline-block w3-small" style="vertical-
      ↪ align: middle;">Recommended</span>

<div class="w3-show-inline-block" style="width: 16px; height:16
      ↪ px; background: rgba(255,105,98, 0.2); border: 1px
      ↪ solid rgb(255, 105, 98); vertical-align: middle;
      ↪ margin-left: 6px;"></div>
<span class="w3-show-inline-block w3-small" style="vertical-
      ↪ align: middle;">Below Recommended</span>
</div>
</div>

<div class="chart-div">
<canvas id="exerciseBarChart" style="width: 100%; height: 250px
      ↪ ; margin-bottom: 10px"></canvas>
</div>
</div>
</div>

{% endif %}
{% endif %}


{% if "water_intake" in habits_data %}
{% if habits_data.water_intake.num_of_habit == 0 %}

<div class="w3-row-padding w3-container w3-section w3-stretch">
<div class="w3-container w3-card w3-round ">
<h6 class="chart-name"><b>Water Intake</b></h6>
<h6 class="w3-text-grey"> Not enough data </h6>
</div>
</div>

{% else %}

<div class="w3-row-padding w3-container w3-section w3-stretch">
<div class="w3-container w3-card w3-round ">
<h6 class="chart-name"><b>Water Intake</b></h6>

<div class="w3-container w3-display-container">
<div class="w3-row w3-display-top w3-right">
<div class="w3-show-inline-block" style="width: 16px; height:16
      ↪ px; background: rgba(119,221,118, 0.2); border: 1px
      ↪ solid rgb(119,221,118); vertical-align: middle;"></div
      ↪ >
<span class="w3-show-inline-block w3-small" style="vertical-
      ↪ align: middle;">Recommended</span>

<div class="w3-show-inline-block" style="width: 16px; height:16
      ↪ px; background: rgba(255,105,98, 0.2); border: 1px
      ↪ solid rgb(255, 105, 98); vertical-align: middle;
      ↪ margin-left: 6px;"></div>
<span class="w3-show-inline-block w3-small" style="vertical-
      ↪ align: middle;">Below Recommended</span>
</div>
</div>

<div class="chart-div">
<canvas id="waterIntakeBarChart" style="width: 100%; height:
      ↪ 250px; margin-bottom: 10px"></canvas>
</div>
</div>
</div>

{% endif %}
{% endif %}


{% else %}

<div class="w3-row-padding w3-container w3-section w3-stretch">
<div class="w3-container w3-card w3-round ">
<h6 class="w3-text-grey"> No habits to track. </h6>
</div>
</div>

{% endif %}

<div class="w3-row-padding w3-section w3-container w3-stretch">
<div class="w3-col s12 m6 w3-margin-bottom">
<a href="{% url 'chat' message='edit habits' %}" class="w3-
      ↪ button add-btn w3-block" >
Edit habits
</a>
</div>

{% if can_add_habit_entry %}

<div class="w3-col s12 m6 w3-margin-bottom">
<a href="{% url 'add_habit_entry' %}" class="w3-button add-btn
      ↪ w3-block" >
Add entry
</a>
</div>

{% else %}

<div class="w3-col s12 m6 w3-margin-bottom">
<a class="w3-button add-btn w3-block w3-disabled">
Add entry
</a>
</div>

{% endif %}
</div>

</div>
```

```
<script type="text/javascript">


// Sleep Bar Chart
const sleepBarChart = document.getElementById('sleepBarChart');
const sleepData = '{{habits_data.sleep.data|escapejs}}'
const modifiedSleepData = sleepData.replace(/'/g, '"');
const parsedSleepData = JSON.parse(modifiedSleepData);

const sleepDatesData = '{{habits_data.sleep.dates|escapejs}}'
const modifiedSleepDatesData = sleepDatesData.replace(/'/g,
      ↪ '"');
const parsedSleepDatesData = JSON.parse(modifiedSleepDatesData)
      ↪ ;

new Chart(sleepBarChart, {
type: 'bar',
data: {
labels: parsedSleepDatesData,
datasets: [
{
axis: 'y',
label: 'Hours of Sleep',
data: parsedSleepData,
backgroundColor: function (context) {
var value = context.dataset.data[context.dataIndex];
return value < 7 ? 'rgba(255,105,98, 0.2)' : 'rgba(119,221,118,
      ↪  0.2)';
},
borderColor: function (context) {
var value = context.dataset.data[context.dataIndex];
return value < 7 ? 'rgb(255, 105, 98)' : 'rgb(119,221,118)';
},
borderWidth: 1
}
]
},
options: {
plugins: {
legend: {
display: false
}
},
scales: {
y: {
beginAtZero: true, // Start the y-axis at zero
precision: 0, // Display only integer values
min: 0, // Set the minimum value for the y-axis
max: 24, // Set the maximum value for the y-axis
title: {
display: true,
text: 'Hours' // Specify the title text for the y-axis
}
}
}
},
});

// Exercise Bar Chart
const exerciseBarChart = document.getElementById('
      ↪ exerciseBarChart');
const exerciseData = '{{habits_data.exercise.data|escapejs}}'
const modifiedExerciseData = exerciseData.replace(/'/g, '"');
const parsedExerciseData = JSON.parse(modifiedExerciseData);

const exerciseDatesData = '{{habits_data.exercise.dates|
      ↪ escapejs}}'
const modifiedExerciseDatesData = exerciseDatesData.replace(/'/
      ↪ g, '"');
const parsedExerciseDatesData = JSON.parse(
      ↪ modifiedExerciseDatesData);

new Chart(exerciseBarChart, {
type: 'bar',
data: {
labels: parsedExerciseDatesData,
datasets: [
{
axis: 'y',
label: 'Minutes of Exercise',
data: parsedExerciseData,
backgroundColor: function (context) {
var value = context.dataset.data[context.dataIndex];
return value < 5 ? 'rgba(255,105,98, 0.2)' : 'rgba(119,221,118,
      ↪  0.2)';
},
borderColor: function (context) {
var value = context.dataset.data[context.dataIndex];
return value < 5 ? 'rgb(255, 105, 98)' : 'rgb(119,221,118)';
},
borderWidth: 1
}
]
},
options: {
plugins: {
legend: {
display: false
}
},
scales: {
y: {
beginAtZero: true, // Start the y-axis at zero
precision: 0, // Display only integer values
min: 0, // Set the minimum value for the y-axis
max: 60, // Set the maximum value for the y-axis
title: {
display: true,
text: 'Minutes' // Specify the title text for the y-axis
}
}
}
},
});


// Water Intake Bar Chart
const waterIntakeBarChart = document.getElementById('
      ↪ waterIntakeBarChart');
const waterIntakeData = '{{habits_data.water_intake.data|
      ↪ escapejs}}'
const modifiedWaterIntakeData = waterIntakeData.replace(/'/g,
      ↪ '"');
const parsedWaterIntakeData = JSON.parse(
      ↪ modifiedWaterIntakeData);

const waterIntakeDatesData = '{{habits_data.water_intake.dates|
      ↪ escapejs}}'
const modifiedWaterIntakeDatesData = waterIntakeDatesData.
      ↪ replace(/'/g, '"');
const parsedWaterIntakeDatesData = JSON.parse(
      ↪ modifiedWaterIntakeDatesData);

new Chart(waterIntakeBarChart, {
type: 'bar',
data: {
labels: parsedWaterIntakeDatesData,
datasets: [
{
axis: 'y',
label: 'Glasses of Water',
data: parsedWaterIntakeData,
backgroundColor: function (context) {
var value = context.dataset.data[context.dataIndex];
return value < 8 ? 'rgba(255,105,98, 0.2)' : 'rgba(119,221,118,
      ↪  0.2)';
},
borderColor: function (context) {
var value = context.dataset.data[context.dataIndex];
return value < 8 ? 'rgb(255, 105, 98)' : 'rgb(119,221,118)';
},
borderWidth: 1
}
]
},
options: {
plugins: {
legend: {
display: false
}
},
scales: {
y: {
beginAtZero: true, // Start the y-axis at zero
precision: 0, // Display only integer values
min: 0, // Set the minimum value for the y-axis
max: 25, // Set the maximum value for the y-axis
title: {
display: true,
text: 'Glasses' // Specify the title text for the y-axis
}
}
}
},
});

</script>
</body>

{% endblock %}
```

Listing 22: templates/main_app/entry.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

{% include "main_app/sidebar.html" %}

<body style="background: #F5F5F5;">

<div class="w3-main w3-container w3-padding-16 grey-bg" >
```

```
<a href="{% url 'diary_entry_list' %}">
<h6 class="w3-small">
<span class="iconify back-icon" data-icon="material-symbols:
      ↪ arrow-back-rounded"></span>
back
</h6>
</a>

{% if messages %}
{% for message in messages %}
{% if message.tags == "success" %}
<div class="message success">
{{ message }}
</div>
{% else %}
<div class="message error">
{{ message }}
</div>
{% endif %}
{% endfor %}
{% endif %}

<div class="w3-row-padding w3-section w3-container w3-stretch">
<span class="w3-left w3-large"><b>{{diary.title}}</b></span>
<h6><span class="w3-right w3-small">{{diary.date.date}}</span
      ↪ ></h6>
</div>

{% if num_of_moods > 1 %}

<div class="w3-row-padding w3-container w3-section w3-stretch">
<div class="w3-container w3-round w3-card" >
<h6 class="chart-name"><b>Mood</b></h6>
<div class="chart-div">
<canvas id="lineChart" style="width: 100%; height: 250px;
      ↪ margin-bottom: 10px"></canvas>
</div>
</div>
</div>

{% endif %}



<div class="w3-row-padding w3-stretch">

{% if num_of_moods == 1 %}

<div class="w3-col s6 m4 l3 w3-margin-bottom">
<div class="w3-container w3-round w3-card" style="aspect-ratio:
      ↪  1/1;">
<h6 class="chart-name"><b>Mood</b></h6>

<div class="w3-container w3-auto w3-center">
{% if moods == "awful" %}
<span class="iconify diary-entry-mood-icon mood-wrrr-icon" data
      ↪ -icon="tabler:mood-wrrr"></span>
{% endif %}

{% if moods == "bad" %}
<span class="iconify diary-entry-mood-icon mood-sad-icon" data-
      ↪ icon="tabler:mood-sad"></span>
{% endif %}

{% if moods == "okay" %}
<span class="iconify diary-entry-mood-icon mood-empty-icon"
      ↪ data-icon="tabler:mood-empty"></span>
{% endif %}

{% if moods == "good" %}
<span class="iconify diary-entry-mood-icon mood-smile-icon"
      ↪ data-icon="tabler:mood-smile"></span>
{% endif %}

{% if moods == "great" %}
<span class="iconify diary-entry-mood-icon mood-smile-beam-icon
      ↪ " data-icon="tabler:mood-smile-beam"></span>
{% endif %}

</div>
<h6 class="w3-small no-margin w3-margin-top"><b>Mood Trigger:</
      ↪ b> {{mood_triggers}}</h6>
<h6 class="w3-small no-margin w3-margin-bottom"><b>Time:</b> {{
      ↪ times_of_the_day}}</h6>

</div>
</div>

{% endif %}



{% if habits_data.sleep %}
<div class="w3-col s6 m4 l3 w3-margin-bottom">
<div class="w3-container w3-round w3-card" style="aspect-ratio:
      ↪  1/1;">
<h6 class="chart-name"><b>Sleep</b></h6>
<div class="donut-div">
<canvas id="myDonut" class="donut-canvas"></canvas>
</div>
</div>
</div>
{% endif %}

{% if habits_data.exercise %}
<div class="w3-col s6 m4 l3 w3-margin-bottom">
<div class="w3-container w3-round w3-card" style="aspect-ratio:
      ↪  1/1;">
<h6 class="chart-name"><b>Exercise</b></h6>
<div class="donut-div">
<canvas id="myDonut2" class="donut-canvas"></canvas>
</div>
</div>
</div>

{% endif %}

{% if habits_data.water_intake %}
<div class="w3-col s6 m4 l3 w3-margin-bottom">
<div class="w3-container w3-round w3-card" style="aspect-ratio:
      ↪  1/1;">
<h6 class="chart-name"><b>Water Intake</b></h6>
<div class="donut-div">
<canvas id="myDonut3" class="donut-canvas"></canvas>
</div>
</div>
</div>
{% endif %}

</div>

<div class="w3-row-padding w3-stretch w3-container ">
<h6><b>Well-Being Strategies Tried</b></h6>

{% if strategies_tried %}
<div class="w3-row-padding w3-stretch">
{% for strategy in strategies_tried %}
<div class="w3-col s6 m4 l3 w3-margin-bottom">
<a href="{% url 'view_strategy' strategy.strategy.strategy_id
      ↪ %}" class="w3-button w3-card w3-container ">
<h6 class="chart-name w3-left-align">{{strategy.strategy.name
      ↪ }}</h6>
</a>
</div>
{% endfor %}
</div>
{% else %}
<div class="w3-row-padding w3-container w3-section w3-stretch">
<div class="w3-container w3-round" style="background: #e3dede">
<h6 class="w3-small">No strategies tried.</h6>
</div>
</div>
{% endif %}



</div>

<div class="w3-row-padding w3-container w3-stretch">

<h6><b>Articles Read</b></h6>

{% if articles_read %}
<div class="w3-row-padding w3-stretch">
{% for article in articles_read %}
<div class="w3-col s12 m12 l3 w3-margin-bottom">
<a href="{% url 'view_article' article.article.article_id %}"
      ↪ class="w3-button w3-card w3-container ">
<h6 class="chart-name w3-left-align">{{article.article.title
      ↪ }}</h6>
</a>
</div>
{% endfor %}
</div>
{% else %}
<div class="w3-row-padding w3-container w3-section w3-stretch">
<div class="w3-container w3-round" style="background: #e3dede">
<h6 class="w3-small">No articles read.</h6>
</div>
</div>
{% endif %}

</div>

<div class="w3-row-padding w3-container w3-stretch">

<h6><b>My thoughts</b></h6>

<div class="w3-container w3-round w3-card" id="thoughtsDiv">
{% if diary.thoughts %}
<h6>
{{diary.thoughts}}
</h6>
{% else %}
<h6 class="w3-text-grey w3-small">
No entry recorded.
</h6>
{% endif %}
</div>
```

```
{% if allow_diary_edit %}
<button id="revealButton" class="w3-button w3-margin-top signin
      ↪ -btn" onclick="revealTextBox()">Edit diary</button>
<div id="textBoxContainer" class="hidden">

<form method="post" action="{% url 'diary_entry' diary.diary_id
      ↪  %}">
{% csrf_token %}
{% for field in form %}
{{field}}
{% endfor %}

<button class="w3-button w3-margin-top signin-btn" type="submit
      ↪ "><span>Save</span></button>
</form>

</div>
{% endif %}

</div>

</div>

<!-- JS -->
<script type="text/javascript" src="{% static 'js/diary.js' %}"
      ↪  ></script>

<script>
const moods = ["Awful", "Bad", "Okay", "Good", "Great"];
const moodsIntValues = "{{moods_int_values}}"
const modifiedMoodsIntValues = moodsIntValues.replace(/'/g,
      ↪ '"');
const parsedMoodsIntValues = JSON.parse(modifiedMoodsIntValues)
      ↪ ;

const moodTriggers = "{{mood_triggers|escapejs}}";
const modifiedMoodTriggers = moodTriggers.replace(/'/g, '"');
const parsedMoodTriggers = JSON.parse(modifiedMoodTriggers);

const timesOfTheDay = "{{times_of_the_day|escapejs}}"
const modifiedTimesOfTheDay = timesOfTheDay.replace(/'/g, '"');
const parsedTimesOfTheDay = JSON.parse(modifiedTimesOfTheDay);

const habitData = {{habits_data.sleep}}
const quotient = (habitData / 24) * 100
const finalHabitData = quotient.toFixed(2);

const exerciseData = {{habits_data.exercise}}
const exerciseQuotient = (exerciseData / 60) * 100
const finalExerciseHabitData = exerciseQuotient.toFixed(2);

const waterIntakeData = {{habits_data.water_intake}}
const waterIntakeQuotient = (waterIntakeData / 20) * 100
const finalWaterIntakeHabitData = waterIntakeQuotient.toFixed
      ↪ (2);
</script>

</body>

{% endblock %}
```

Listing 23: templates/-main_app/entry_list.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

{% include "main_app/sidebar.html" %}

<body style="background: #F5F5F5;">

<div class="w3-main w3-padding-16 w3-container">

<a href="{% url 'home' %}">
<h6 class="w3-small">
<span class="iconify back-icon" data-icon="material-symbols:
      ↪ arrow-back-rounded"></span>
back
</h6>
</a>

<div>
<h3>
<span class="iconify title-icon" data-icon="uil:diary"></span>
<span class="w3-large"><b>My Diary</b></span>
</h3>
</div>

{% if diary_entries %}
{% for entry in diary_entries %}
<a href="{% url 'diary_entry' entry.diary_id %}" class="w3-
      ↪ container w3-button w3-card w3-block w3-margin-bottom
      ↪ w3-left-align">
<h6 style="margin-bottom: 0px">
<b>{{entry.title}}</b>
<span class="w3-small w3-text-grey w3-right">{{entry.date}}</
      ↪ span>
</h6>

<h6 style="margin-top: 0px">
<span style="font-size:14px"> Mood: </span>

{% if entry.mood_present.4 %}
<span class="iconify diary-mood-icon mood-wrrr-icon" data-icon
      ↪ ="tabler:mood-wrrr"></span>
{% endif %}

{% if entry.mood_present.3 %}
<span class="iconify diary-mood-icon mood-sad-icon" data-icon="
      ↪ tabler:mood-sad"></span>
{% endif %}

{% if entry.mood_present.2 %}
<span class="iconify diary-mood-icon mood-empty-icon" data-icon
      ↪ ="tabler:mood-empty"></span>
{% endif %}

{% if entry.mood_present.1 %}
<span class="iconify diary-mood-icon mood-smile-icon" data-icon
      ↪ ="tabler:mood-smile"></span>
{% endif %}

{% if entry.mood_present.0 %}
<span class="iconify diary-mood-icon mood-smile-beam-icon" data
      ↪ -icon="tabler:mood-smile-beam"></span>
{% endif %}
</h6>
</a>
{% endfor %}
{% else %}

<div class="w3-row-padding w3-container w3-section w3-stretch">
<div class="w3-container w3-card w3-round ">
<h6 class="w3-text-grey"> No entries yet. </h6>
</div>
</div>

{% endif %}

</div>

</body>

{% endblock %}
```

Listing 24: templates/-main_app/view_strategy.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

{% include "main_app/sidebar.html" %}

<body style="background: #F5F5F5;">

<div class="w3-main w3-padding-16 w3-container">

{% if request.user.is_staff == False %}
<a href="{% url 'done_strategy' strategy.name %}">
<h6 class="w3-small">
<span class="iconify back-icon" data-icon="material-symbols:
      ↪ arrow-back-rounded"></span>
back to Kaya
</h6>
</a>
{% else %}
<a href="{% url 'articles_list' %}">
<h6 class="w3-small">
<span class="iconify back-icon" data-icon="material-symbols:
      ↪ arrow-back-rounded"></span>
back
</h6>
</a>
{% endif %}

<div class="w3-card w3-display-container w3-round view-strategy
      ↪ -div">

<div style="height: 90px; background: #00CF95; border-radius: 4
      ↪ px 4px 0px 0px;"></div>

<div class="view-strategy-img-div w3-display-topmiddle" style="
      ↪ margin-top:40px">
<img src="{% static 'img/strategy.png' %}" alt="strategy pic"
      ↪ class="strategy-image">
</div>

<div class="w3-padding-large w3-padding-top-48">
```

```
<div>
<h3 class="w3-center no-margin">
<span class="w3-large"><b>{{strategy.name}}</b></span>
</h3>

<p class="w3-justify">
{{strategy.description}}
</p>

<h6 class="w3-small no-margin"><b>Reference:</b></h6>

<h6 class="w3-small no-margin w3-margin-bottom">
<a href="{{strategy.source}}">{{strategy.source}}</a>
</h6>

</div>

<div class="w3-margin-top">

<div class="w3-row-padding w3-section w3-container w3-stretch">

{% if strategy_tried %}
<div class="w3-col s12 m6 w3-margin-bottom">
<div class="disabled-btn w3-round w3-center">
Marked as Tried
</div>
</div>
{% else %}
<div class="w3-col s12 m6 w3-margin-bottom">
<a href="{% url 'tried_strategy' strategy.strategy_id %}" class
      ↪ ="w3-button add-btn w3-block" >
Mark as Tried
</a>
</div>
{% endif %}

{% if strategy_tried.is_liked == False %}
<div class="w3-col s12 m6 w3-margin-bottom">
<a href="{% url 'liked_strategy' strategy.strategy_id %}" class
      ↪ ="w3-button add-btn w3-block" >
Mark as Liked
</a>
</div>
{% elif strategy_tried.is_liked == True %}
<div class="w3-col s12 m6 w3-margin-bottom">
<div class="disabled-btn w3-round w3-center">
Marked as Liked
</div>
</div>
{% else %}
<div class="w3-col s12 m6 w3-margin-bottom">
<div class="disabled-btn w3-round w3-center">
Mark as Liked
</div>
</div>
{% endif %}

{% if strategy_tried.is_effective == False %}
<div class="w3-col s12 m12 w3-margin-bottom">
<a href="{% url 'effective_strategy' strategy.strategy_id %}"
      ↪ class="w3-button add-btn w3-block" >
Mark as Effective
</a>
</div>
{% elif strategy_tried.is_effective == True %}
<div class="w3-col s12 m12 w3-margin-bottom">
<div class="disabled-btn w3-round w3-center w3-block">
Marked as Effective
</div>
</div>
{% else %}
<div class="w3-col s12 m12 w3-margin-bottom">
<div class="disabled-btn w3-round w3-center w3-block">
Mark as Effective
</div>
</div>
{% endif %}
</div>

</div>

</div>

</div>


<!-- end of main -->
</div>

<!-- JS -->
<script type="text/javascript" src="{% static 'js/rating.js'
      ↪ %}" ></script>

</body>

{% endblock %}
```

Listing 25: templates/-main_app/strategy_list.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

{% include "main_app/sidebar.html" %}

<body style="background: #F5F5F5;">

<div class="w3-main w3-padding-16 w3-container">

<a href="{% url 'home' %}">
<h6 class="w3-small">
<span class="iconify back-icon" data-icon="material-symbols:
      ↪ arrow-back-rounded"></span>
back
</h6>
</a>

<div>
<h3>
<span class="iconify title-icon" data-icon="tabler:bulb"></span
      ↪ >
<span class="w3-large"><b>My Strategies</b></span>
</h3>
</div>

<div class="w3-row-padding w3-section w3-container w3-stretch">
<span class="w3-left w3-text-grey"><b>Effective Strategies</b
      ↪ ></span>
</div>

{% if effective_strategy %}
<div class="w3-row-padding w3-stretch">
{% for strategy in effective_strategy %}
<div class="w3-col s6 m4 l3 w3-margin-bottom">
<a href="{% url 'view_strategy' strategy.strategy.strategy_id
      ↪ %}" class="w3-button w3-card w3-container ">
<h6 class="chart-name w3-left-align">{{strategy.strategy.name
      ↪ }}</h6>
</a>
</div>
{% endfor %}
</div>
{% else %}
<div class="w3-row-padding w3-container w3-section w3-stretch">
<div class="w3-container w3-round" style="background: #e3dede">
<h6 class="w3-small">No strategies marked as effective.</h6>
</div>
</div>
{% endif %}

<div class="w3-row-padding w3-section w3-container w3-stretch">
<span class="w3-left w3-text-grey"><b>Liked Strategies</b></
      ↪ span>
</div>

{% if liked_strategy %}
<div class="w3-row-padding w3-stretch">
{% for strategy in liked_strategy %}
<div class="w3-col s6 m4 l3 w3-margin-bottom">
<a href="{% url 'view_strategy' strategy.strategy.strategy_id
      ↪ %}" class="w3-button w3-card w3-container">
<h6 class="chart-name w3-left-align">{{strategy.strategy.name
      ↪ }}</h6>
</a>
</div>
{% endfor %}
</div>
{% else %}
<div class="w3-row-padding w3-container w3-section w3-stretch">
<div class="w3-container w3-round" style="background: #e3dede">
<h6 class="w3-small">No strategies marked as liked.</h6>
</div>
</div>
{% endif %}

<div class="w3-row-padding w3-section w3-container w3-stretch">
<span class="w3-left w3-text-grey"><b>Tried Strategies</b></
      ↪ span>
</div>

{% if strategy_tried %}
<div class="w3-row-padding w3-stretch">
{% for strategy in strategy_tried %}
<div class="w3-col s6 m4 l3 w3-margin-bottom">
<a href="{% url 'view_strategy' strategy.strategy.strategy_id
      ↪ %}" class="w3-button w3-card w3-container ">
<h6 class="chart-name w3-left-align">{{strategy.strategy.name
      ↪ }}</h6>
</a>
</div>
{% endfor %}
</div>
{% else %}
```

```
<div class="w3-row-padding w3-container w3-section w3-stretch">
<div class="w3-container w3-round" style="background: #e3dede">
<h6 class="w3-small">No strategies marked as tried.</h6>
</div>
</div>
{% endif %}

<div class="w3-row-padding w3-section w3-container w3-stretch">
<a href="{% url 'chat' message='learn well-being strategies'
      ↪ %}" class="w3-button add-btn w3-block" >
Learn well-being strategies
</a>
</div>

</div>

</body>

{% endblock %}
```

Listing 26: templates/-main_app/view_strategy.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

{% include "main_app/sidebar.html" %}

<body style="background: #F5F5F5;">

<div class="w3-main w3-padding-16 w3-container">

{% if request.user.is_staff == False %}
<a href="{% url 'done_strategy' strategy.name %}">
<h6 class="w3-small">
<span class="iconify back-icon" data-icon="material-symbols:
      ↪ arrow-back-rounded"></span>
back to Kaya
</h6>
</a>
{% else %}
<a href="{% url 'articles_list' %}">
<h6 class="w3-small">
<span class="iconify back-icon" data-icon="material-symbols:
      ↪ arrow-back-rounded"></span>
back
</h6>
</a>
{% endif %}

<div class="w3-card w3-display-container w3-round view-strategy
      ↪ -div">

<div style="height: 90px; background: #00CF95; border-radius: 4
      ↪ px 4px 0px 0px;"></div>

<div class="view-strategy-img-div w3-display-topmiddle" style="
      ↪ margin-top:40px">
<img src="{% static 'img/strategy.png' %}" alt="strategy pic"
      ↪ class="strategy-image">
</div>

<div class="w3-padding-large w3-padding-top-48">

<div>
<h3 class="w3-center no-margin">
<span class="w3-large"><b>{{strategy.name}}</b></span>
</h3>

<p class="w3-justify">
{{strategy.description}}
</p>

<h6 class="w3-small no-margin"><b>Reference:</b></h6>

<h6 class="w3-small no-margin w3-margin-bottom">
<a href="{{strategy.source}}">{{strategy.source}}</a>
</h6>

</div>

<div class="w3-margin-top">

<div class="w3-row-padding w3-section w3-container w3-stretch">

{% if strategy_tried %}
<div class="w3-col s12 m6 w3-margin-bottom">
<div class="disabled-btn w3-round w3-center">
Marked as Tried
</div>
</div>
{% else %}
<div class="w3-col s12 m6 w3-margin-bottom">
<a href="{% url 'tried_strategy' strategy.strategy_id %}" class
      ↪ ="w3-button add-btn w3-block" >
Mark as Tried
</a>
</div>
{% endif %}

{% if strategy_tried.is_liked == False %}
<div class="w3-col s12 m6 w3-margin-bottom">
<a href="{% url 'liked_strategy' strategy.strategy_id %}" class
      ↪ ="w3-button add-btn w3-block" >
Mark as Liked
</a>
</div>
{% elif strategy_tried.is_liked == True %}
<div class="w3-col s12 m6 w3-margin-bottom">
<div class="disabled-btn w3-round w3-center">
Marked as Liked
</div>
</div>
{% else %}
<div class="w3-col s12 m6 w3-margin-bottom">
<div class="disabled-btn w3-round w3-center">
Mark as Liked
</div>
</div>
{% endif %}

{% if strategy_tried.is_effective == False %}
<div class="w3-col s12 m12 w3-margin-bottom">
<a href="{% url 'effective_strategy' strategy.strategy_id %}"
      ↪ class="w3-button add-btn w3-block" >
Mark as Effective
</a>
</div>
{% elif strategy_tried.is_effective == True %}
<div class="w3-col s12 m12 w3-margin-bottom">
<div class="disabled-btn w3-round w3-center w3-block">
Marked as Effective
</div>
</div>
{% else %}
<div class="w3-col s12 m12 w3-margin-bottom">
<div class="disabled-btn w3-round w3-center w3-block">
Mark as Effective
</div>
</div>
{% endif %}
</div>

</div>

</div>


<!-- end of main -->
</div>

<!-- JS -->
<script type="text/javascript" src="{% static 'js/rating.js'
      ↪ %}" ></script>

</body>

{% endblock %}
```

Listing 27: templates/-main_app/account_settings.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

{% include "main_app/sidebar.html" %}

<div class="w3-main">

<div class="w3-container w3-padding">

{% if messages %}
{% for message in messages %}
{% if message.tags == "success" %}
<div class="message success">
{{ message }}
</div>
{% else %}
<div class="message error">
{{ message }}
</div>
{% endif %}
{% endfor %}
{% endif %}

<h2><b>ACCOUNT SETTINGS</b></h2>
```

```
<div class="w3-container w3-round" style="background: #F5F5F5
      ↪ ;">
<h4><strong>Profile</strong></h4>

<h6><b>Email</b>: <span>{{request.user.email}}</span></h6>

<h6><b>First Name</b>:
{% if request.user.first_name %}
{{request.user.first_name}}
{% else%}
<span class="w3-text-grey w3-small"> First name not set.</span>
{% endif%}
</h6>

<h6><b>Last Name</b>:
{% if request.user.last_name %}
{{request.user.last_name}}
{% else%}
<span class="w3-text-grey w3-small"> Last name not set.</span>
{% endif%}
</h6>

<h6><b>Contact Number</b>:
{% if request.user.contact_number %}
{{request.user.contact_number}}
{% else%}
<span class="w3-text-grey w3-small"> Contact number not set.</
      ↪ span>
{% endif%}
</h6>

<h6><b>Birthday</b>:
{% if request.user.birthday %}
{{request.user.birthday}}
{% else%}
<span class="w3-text-grey w3-small"> Birthday not set.</span>
{% endif%}
</h6>


<div class="w3-row-padding w3-section w3-container w3-stretch">
<a href="{% url 'edit_profile' %}" class="w3-button add-btn w3-
      ↪ block" >
Edit profile
</a>
</div>
</div>

<br>

<div class="w3-container w3-round" style="background: #F5F5F5
      ↪ ;">
<h4><strong>Notifications (Time Prompt)</strong></h4>

<h6><b>Daily Check-In</b>: <span>{{request.user.
      ↪ tracker_time_prompt}}</span></h6>

<h6><b>Daily Insights</b>: <span>{{request.user.
      ↪ insights_time_prompt}}</span></h6>

<div class="w3-row-padding w3-section w3-container w3-stretch">
<a href="{% url 'notification_settings' %}" class="w3-button
      ↪ add-btn w3-block" >
Change time
</a>
</div>
</div>

<br>


</div>

<!-- end of main -->
</div>

{% endblock %}
```

Listing 28: templates/-main_app/edit_profile.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

{% include "main_app/sidebar.html" %}

<div class="w3-main">

<div class="w3-container w3-padding w3-center w3-block">


<h2><b>EDIT PROFILE</b></h2>

<div class="w3-container w3-auto w3-center signin-form-div">
{% if messages %}
{% for message in messages %}
{% if message.tags == "success" %}
<div class="message success">
{{ message }}
</div>
{% else %}
<div class="message error">
{{ message }}
</div>
{% endif %}
{% endfor %}
{% endif %}

<form method="post" action="{% url 'edit_profile' %}" enctype="
      ↪ multipart/form-data">
{% csrf_token %}

{% for field in form %}
<h6 class="w3-left field-name">{{field.label}}</h6>
{{field}}
<br>
{% endfor %}

<br>

<button class="w3-button w3-margin-top signin-btn" type="submit
      ↪ "><span>Save</span></button>

</form>

</div>

</div>

<!-- end of main -->
</div>

{% endblock %}
```

Listing 29: templates/-main_app/notification_settings.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

{% include "main_app/sidebar.html" %}

<div class="w3-main">

<div class="w3-container w3-padding w3-center w3-block">


<h2><b>NOTIFICATIONS (TIME PROMPT)</b></h2>

<div class="w3-container w3-auto w3-center signin-form-div">
{% if messages %}
{% for message in messages %}
{% if message.tags == "success" %}
<div class="message success">
{{ message }}
</div>
{% else %}
<div class="message error">
{{ message }}
</div>
{% endif %}
{% endfor %}
{% endif %}

{% if form.errors %}
{% for field in form %}
{% for error in field.errors %}
<div class="message error">
{{ error }}
</div>
{% endfor %}
{% endfor %}
{% endif %}

<form method="post" action="{% url 'notification_settings' %}"
      ↪ class="w3-margin-bottom">
{% csrf_token %}

{% for field in form %}
<h6 class="w3-left field-name">{{field.label}}</h6>
{{field}}
<br>
{% endfor %}


<button class="w3-button w3-margin-top signin-btn" type="submit
      ↪ "><span>Save</span></button>
</form>

{% load webpush_notifications %}
```

```
{% webpush_button with_class="w3-button chat-btn" %}

</div>

</div>

<!-- end of main -->
</div>

{% endblock %}
```

Listing 30: templates/-main_app/admin/add_article.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

{% include "main_app/sidebar.html" %}

<div class="w3-main">

<div class="w3-container w3-padding w3-center w3-block">


<h2><b>ADD ARTICLE</b></h2>

<div class="w3-container w3-auto w3-center signin-form-div">
{% if messages %}
{% for message in messages %}
{% if message.tags == "success" %}
<div class="message success">
{{ message }}
</div>
{% else %}
<div class="message error">
{{ message }}
</div>
{% endif %}
{% endfor %}
{% endif %}

<form method="post" action="{% url 'add_article' %}" enctype="
      ↪ multipart/form-data">
{% csrf_token %}

{% for field in form %}
<h6 class="w3-left field-name">{{field.label}}</h6>
{{field}}
<br>
{% endfor %}

<br>

<button class="w3-button w3-margin-top signin-btn" type="submit
      ↪ "><span>Save</span></button>

</form>

</div>

</div>

<!-- end of main -->
</div>

{% endblock %}
```

Listing 31: templates/-main_app/admin/add_strategy.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

{% include "main_app/sidebar.html" %}

<div class="w3-main">

<div class="w3-container w3-padding w3-center w3-block">


<h2><b>ADD STRATEGY</b></h2>

<div class="w3-container w3-auto w3-center signin-form-div">
{% if messages %}
{% for message in messages %}
{% if message.tags == "success" %}
<div class="message success">
{{ message }}
</div>
{% else %}
<div class="message error">
{{ message }}
</div>
{% endif %}
{% endfor %}
{% endif %}

<form method="post" action="{% url 'add_strategy' %}" enctype="
      ↪ multipart/form-data">
{% csrf_token %}

{% for field in form %}
<h6 class="w3-left field-name">{{field.label}}</h6>
{{field}}
<br>
{% endfor %}


<br>

<button class="w3-button w3-margin-top signin-btn" type="submit
      ↪ "><span>Save</span></button>

</form>

</div>

</div>

<!-- end of main -->
</div>

{% endblock %}
```

Listing 32: templates/-main_app/admin/admin.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

{% include "main_app/sidebar.html" %}

<body style="background: #F5F5F5;">

<div class="w3-main w3-padding-16 w3-container">

{% if messages %}
{% for message in messages %}
{% if message.tags == "success" %}
<div class="message success">
{{ message }}
</div>
{% else %}
<div class="message error">
{{ message }}
</div>
{% endif %}
{% endfor %}
{% endif %}

<div class="w3-row-padding">
<div class="w3-row w3-padding">
<h3 class="w3-large"><b>Admin Dashboard</b></h3>
</div>

<div class="w3-half w3-margin-bottom">
<div class="w3-card w3-padding w3-round w3-center">
<p>Number of Well-Being Strategies</p>
<h1><b>{{ strategy_count }}<b/></h1>
</div>
</div>

<div class="w3-half w3-margin-bottom">
<div class="w3-card w3-padding w3-round w3-center">
<p>Number of Articles</p>
<h1><b>{{ article_count }}</b></h1>
</div>
</div>
</div>

</div>

</body>

{% endblock %}
```

Listing 33: templates/-main_app/admin/articles_list.html

```
{% extends "main_app/base.html" %}
{% load static %}
```

```
{% block content %}

{% include "main_app/sidebar.html" %}

<body style="background: #F5F5F5;">

<div class="w3-main w3-padding-16 w3-container">

<a href="{% url 'home' %}">
<h6 class="w3-small">
<span class="iconify back-icon" data-icon="material-symbols:
      ↪ arrow-back-rounded"></span>
back
</h6>
</a>

{% if messages %}
{% for message in messages %}
{% if message.tags == "success" %}
<div class="message success">
{{ message }}
</div>
{% else %}
<div class="message error">
{{ message }}
</div>
{% endif %}
{% endfor %}
{% endif %}

<div>
<h3>
<span class="iconify title-icon" data-icon="akar-icons:book-
      ↪ open"></span>
<span class="w3-large"><b>Online Articles</b></span>
</h3>
</div>

<div class="w3-row-padding w3-section w3-container w3-stretch">
<a href="{% url 'add_article' %}" class="w3-button add-btn">
<span>add article</span>
</a>
</div>

{% if articles %}
{% for article in articles %}
<a href="{% url 'admin_view_article' article.article_id %}"
      ↪ class="w3-container w3-button w3-card w3-block w3-
      ↪ margin-bottom w3-left-align">
<h6 style="margin-bottom: 0px">
<b>{{article.title}}</b>
</h6><br>
</a>
{% endfor %}
{% else %}

<div class="w3-row-padding w3-container w3-section w3-stretch">
<div class="w3-container w3-card w3-round ">
<h6 class="w3-text-grey"> No articles yet. </h6>
</div><br>
</div>

{% endif %}

</div>

</body>

{% endblock %}
```

Listing 34: templates/-main_app/admin/edit_article.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

{% include "main_app/sidebar.html" %}

<div class="w3-main">

<div class="w3-container w3-padding w3-center w3-block">


<h2><b>EDIT ARTICLE</b></h2>

<div class="w3-container w3-auto w3-center signin-form-div">
{% if messages %}
{% for message in messages %}
{% if message.tags == "success" %}
<div class="message success">
{{ message }}
</div>
{% else %}
<div class="message error">
{{ message }}
</div>
{% endif %}
{% endfor %}
{% endif %}

<form method="post" action="{% url 'edit_article' article_id
      ↪ %}" enctype="multipart/form-data">
{% csrf_token %}

{% for field in form %}
<h6 class="w3-left field-name">{{field.label}}</h6>
{{field}}
<br>
{% endfor %}

<br>

<a href="{% url 'delete_article' article_id %}" class="w3-
      ↪ button w3-margin-top signin-btn w3-red" ><span>Delete
      ↪ </span></a>
<button class="w3-button w3-margin-top signin-btn" type="submit
      ↪ "><span>Save</span></button>

</form>

</div>

</div>

<!-- end of main -->
</div>

{% endblock %}
```

Listing 35: templates/-main_app/admin/edit_strategy.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

{% include "main_app/sidebar.html" %}

<div class="w3-main">

<div class="w3-container w3-padding w3-center w3-block">


<h2><b>EDIT STRATEGY</b></h2>

<div class="w3-container w3-auto w3-center signin-form-div">
{% if messages %}
{% for message in messages %}
{% if message.tags == "success" %}
<div class="message success">
{{ message }}
</div>
{% else %}
<div class="message error">
{{ message }}
</div>
{% endif %}
{% endfor %}
{% endif %}

<form method="post" action="{% url 'edit_strategy' strategy_id
      ↪ %}" enctype="multipart/form-data">
{% csrf_token %}

{% for field in form %}
<h6 class="w3-left field-name">{{field.label}}</h6>
{{field}}
<br>
{% endfor %}

<br>

<a href="{% url 'delete_strategy' strategy_id %}" class="w3-
      ↪ button w3-margin-top signin-btn w3-red" ><span>Delete
      ↪ </span></a>
<button class="w3-button w3-margin-top signin-btn" type="submit
      ↪ "><span>Save</span></button>

</form>

</div>

</div>

<!-- end of main -->
</div>

{% endblock %}
```

Listing 36: templates/-main_app/admin/strategies_list.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

{% include "main_app/sidebar.html" %}

<body style="background: #F5F5F5;">

<div class="w3-main w3-padding-16 w3-container">

<a href="{% url 'home' %}">
<h6 class="w3-small">
<span class="iconify back-icon" data-icon="material-symbols:
      ↪ arrow-back-rounded"></span>
back
</h6>
</a>

{% if messages %}
{% for message in messages %}
{% if message.tags == "success" %}
<div class="message success">
{{ message }}
</div>
{% else %}
<div class="message error">
{{ message }}
</div>
{% endif %}
{% endfor %}
{% endif %}

<div>
<h3>
<span class="iconify title-icon" data-icon="tabler:bulb"></span
      ↪ >
<span class="w3-large"><b>Well-Being Strategies</b></span>
</h3>
</div>

<div class="w3-row-padding w3-section w3-container w3-stretch">
<a href="{% url 'add_strategy' %}" class="w3-button add-btn">
<span>add strategy</span>
</a>
</div>

{% if strategies %}
{% for strategy in strategies %}
<a href="{% url 'admin_view_strategy' strategy.strategy_id %}"
      ↪ class="w3-container w3-button w3-card w3-block w3-
      ↪ margin-bottom w3-left-align">
<h6 style="margin-bottom: 0px">
<b>{{strategy.name}}</b>
</h6><br>
</a>
{% endfor %}
{% else %}

<div class="w3-row-padding w3-container w3-section w3-stretch">
<div class="w3-container w3-card w3-round ">
<h6 class="w3-text-grey"> No strategies yet. </h6>
</div>
</div>

{% endif %}

</div>

</body>

{% endblock %}
```

Listing 37: templates/-main_app/admin/view_article.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

{% include "main_app/sidebar.html" %}

<div class="w3-main">

<div class="w3-container w3-padding">

<a href="{% url 'articles_list' %}">
<h6 class="w3-small">
<span class="iconify back-icon" data-icon="material-symbols:
      ↪ arrow-back-rounded"></span>
back
</h6>
</a>

{% if messages %}
{% for message in messages %}
{% if message.tags == "success" %}
<div class="message success">
{{ message }}
</div>
{% else %}
<div class="message error">
{{ message }}
</div>
{% endif %}
{% endfor %}
{% endif %}

<h2><b>ONLINE ARTICLE</b></h2>

<div class="w3-container w3-round" style="background: #F5F5F5
      ↪ ;">
<h4><strong>{{article.title}}</strong></h4>

<h6><b>Description</b>:
{{article.summary}}
</h6>

<h6><b>Source</b>:
{{article.source}}
</h6>

<div class="w3-row-padding w3-section w3-container w3-stretch">
<a href="{% url 'edit_article' article.article_id %}" class="w3
      ↪ -button add-btn w3-block" >
edit article
</a>
</div>
</div>

<br>


</div>

<!-- end of main -->
</div>

{% endblock %}
```

Listing 38: templates/-main_app/admin/view_strategy.html

```
{% extends "main_app/base.html" %}
{% load static %}

{% block content %}

{% include "main_app/sidebar.html" %}

<div class="w3-main">

<div class="w3-container w3-padding">

<a href="{% url 'strategies_list' %}">
<h6 class="w3-small">
<span class="iconify back-icon" data-icon="material-symbols:
      ↪ arrow-back-rounded"></span>
back
</h6>
</a>

{% if messages %}
{% for message in messages %}
{% if message.tags == "success" %}
<div class="message success">
{{ message }}
</div>
{% else %}
<div class="message error">
{{ message }}
</div>
{% endif %}
{% endfor %}
{% endif %}

<h2><b>WELL-BEING STRATEGY</b></h2>

<div class="w3-container w3-round" style="background: #F5F5F5
      ↪ ;">
<h4><strong>{{strategy.name}}</strong></h4>

<h6><b>Description</b>:
{{strategy.description}}
</h6>

<h6><b>Source</b>:
{{strategy.source}}
</h6>
```

```
<div class="w3-row-padding w3-section w3-container w3-stretch">
<a href="{% url 'edit_strategy' strategy.strategy_id %}" class
      ↪ ="w3-button add-btn w3-block" >
edit strategy
</a>
</div>
</div>

<br>


</div>

<!-- end of main -->
</div>

{% endblock %}
```

Listing 39: management/commands/runapscheduler.py

```
# runapscheduler.py
import logging

from django.conf import settings

from apscheduler.schedulers.blocking import BlockingScheduler
from apscheduler.triggers.cron import CronTrigger
from django.core.management.base import BaseCommand
from django_apscheduler.jobstores import DjangoJobStore
from django_apscheduler.models import DjangoJobExecution
from django_apscheduler import util
from main_app.models import CustomUser, Diary
from main_app.views import check_in_user, daily_insights
from datetime import datetime

logger = logging.getLogger(__name__)


def create_diary_job():
    users = CustomUser.objects.all()
    for user in users:
        diary = Diary(user=user)
        diary.save()


def check_in_user_job():
    # users = CustomUser.objects.all()
    current_time = datetime.now().time().replace(second=0,
          ↪ microsecond=0)
    # print(current_time)
    users = CustomUser.objects.filter(tracker_time_prompt=
          ↪ current_time)

    for user in users:
        check_in_user(user)


def daily_insights_job():
    # users = CustomUser.objects.all()

    current_time = datetime.now().time().replace(second=0,
          ↪ microsecond=0)
    # print(current_time)
    users = CustomUser.objects.filter(insights_time_prompt=
          ↪ current_time)
    for user in users:
        daily_insights(user)


# The `close_old_connections` decorator ensures that database
      ↪ connections, that have become
# unusable or are obsolete, are closed before and after your
      ↪ job has run. You should use it
# to wrap any jobs that you schedule that access the Django
      ↪ database in any way.
@util.close_old_connections
def delete_old_job_executions(max_age=604_800):
    """
    This job deletes APScheduler job execution entries older
          ↪ than `max_age` from the database.
    It helps to prevent the database from filling up with old
          ↪ historical records that are no
    longer useful.

    :param max_age: The maximum length of time to retain
          ↪ historical job execution records.
                Defaults to 7 days.
    """
    DjangoJobExecution.objects.delete_old_job_executions(max_age
          ↪ )


class Command(BaseCommand):
    help = "Runs APScheduler."

    def handle(self, *args, **options):
        scheduler = BlockingScheduler(timezone=settings.
              ↪ TIME_ZONE)
        scheduler.add_jobstore(DjangoJobStore(), "default")

        scheduler.add_job(
            create_diary_job,
            # trigger=CronTrigger(second="*/10"), # Every 10
                  ↪ seconds
            trigger=CronTrigger(
                day="*", hour="0", minute="0", second="0"
            ), # trigger everyday
            id="create_diary_job", # The `id` assigned to each
                  ↪ job MUST be unique
            max_instances=1,
            replace_existing=True,
        )
        logger.info("Added job 'create_diary_job'.")

        # schedule_job(scheduler)

        scheduler.add_job(
            check_in_user_job,
            trigger=CronTrigger(minute="*"),
            # trigger=CronTrigger(day="*", hour="0", minute="1",
                  ↪ second="0"),
            # trigger=CronTrigger(second="*/30"), # Every 10
                  ↪ seconds
            id="check_in_user_job", # The `id` assigned to each
                  ↪ job MUST be unique
            max_instances=1,
            replace_existing=True,
        )
        logger.info("Added job 'check_in_user_job'.")

        scheduler.add_job(
            daily_insights_job,
            trigger=CronTrigger(minute="*"),
            # trigger=CronTrigger(day="*", hour="0", minute="1",
                  ↪ second="0"),
            # trigger=CronTrigger(second="*/50"), # Every 10
                  ↪ seconds
            id="daily_insights_job", # The `id` assigned to each
                  ↪ job MUST be unique
            max_instances=1,
            replace_existing=True,
        )
        logger.info("Added job 'daily_insights_job'.")

        scheduler.add_job(
            delete_old_job_executions,
            trigger=CronTrigger(
                day_of_week="mon", hour="00", minute="00"
            ), # Midnight on Monday, before start of the next
                  ↪ work week.
            id="delete_old_job_executions",
            max_instances=1,
            replace_existing=True,
        )
        logger.info("Added weekly job: '
              ↪ delete_old_job_executions'.")

        try:
            logger.info("Starting scheduler...")
            scheduler.start()
        except KeyboardInterrupt:
            logger.info("Stopping scheduler...")
            scheduler.shutdown()
            logger.info("Scheduler shut down successfully!")
```

Listing 40: static/chat.js

```
var slideIndex = 1;
var chatContainer = document.getElementById('chat-container');

const csrf_token = document.cookie
.split('; ')
.find(cookie => cookie.startsWith('csrftoken='))
.split('=')[1];


// Function to scroll the chat container to the latest message
function scrollToLatestMessage() {
    chatContainer.scrollTop = chatContainer.scrollHeight;
}

// Checkmark Buttons
function toggleCheckmark(button) {
    button.classList.toggle("checked");
    var checkedValues = []
    var checkedButtons = document.querySelectorAll('.checkmark-
          ↪ button.checked')
    checkedButtons.forEach(function(button) {
        checkedValues.push(button.value);
    });

    if (checkedValues.length > 0) {
```

```
        $('#savebtn').prop('disabled', false)
        $('#savebtn').val(checkedValues.join(', '));
    } else {
        $('#savebtn').prop('disabled', true)
        $('#savebtn').val("");
    }
}

// notifications time
function setTime() {
    var timeInput = document.getElementById("timeInput");
    var selectedTime = timeInput.value
    var formattedTime = formatTime(selectedTime)

    var saveBtn = document.getElementById("savebtn");
    saveBtn.value = formattedTime
    console.log(saveBtn.value)
}

// Format the time in "h:mmA" format
function formatTime(time) {
  var hour = parseInt(time.substring(0, 2));
  var minute = time.substring(3, 5);
  var meridiem = " AM";
  if (hour >= 12) {
    meridiem = " PM";
    if (hour > 12) {
      hour -= 12;
    }
  }
  if (hour === 0) {
    hour = 12;
  }
  return hour + ":" + minute + meridiem;
}

// habits slider
function setHabit() {
    var slider = document.getElementById("slider");
    var sliderValue = slider.value

    var saveBtn = document.getElementById("savebtn");
    saveBtn.value = sliderValue
    console.log(saveBtn.value)

    var sliderText = document.getElementById("sliderText");
    sliderText.innerText = sliderValue;

}

$(document).ready(function() {

    scrollToLatestMessage();
    displayUserMessage(userMessage);
    sendMessageToBot(userMessage, csrf_token);

    $(document).on('click', '.chat-btn', function(e) {
        e.preventDefault(); // Prevent the default form
            ↪ submission or link behavior

        var buttonValue = this.value;
        console.log("Clicked button value: " + buttonValue);

        const csrfToken = document.querySelector('[name=
            ↪ csrfmiddlewaretoken]').value;

        $(".choices-div").remove();
        displayUserMessage(buttonValue);
        sendMessageToBot(buttonValue, csrfToken);

    });
    // end of click event

})
```

Listing 41: static/diary.js

```
// Mood Chart
const ctx = document.getElementById("lineChart").getContext("2d
    ↪ ");
const lineChart = new Chart(ctx, {
type: "line",
data: {
labels: parsedTimesOfTheDay,
datasets: [{
label: "Moods",
data: parsedMoodsIntValues,
borderColor: "rgba(54, 162, 235, 1)",
backgroundColor: "rgba(54, 162, 235, 0.2)",
pointRadius: 8,
pointHoverRadius: 10,
pointBackgroundColor: "rgba(54, 162, 235, 1)",
}]
},
options: {
responsive: true,
plugins: {
legend: {
display: false
},

tooltip: {
callbacks: {
label: function(context) {
var index = context.dataIndex;
var value = context.dataset.data[index];
var label = "Mood trigger: " + parsedMoodTriggers[index] + "";
return label;
}
}
}
},
scales: {
x: {
title: {
display: true,
text: "Time"
}
},
y: {
beginAtZero: true,
title: {
display: true,
text: "Moods"
},
ticks: {
stepSize: 1,
callback: function(value, index, values) {
return moods[value];
}
}
}
}
}
});



// Sleep Chart
var myChartCircle = new Chart('myDonut', {
type: 'doughnut',
data: {
datasets: [{
// label: 'Sle',
percent: finalHabitData,
backgroundColor: [(finalHabitData < 28) ? 'rgba(255, 105, 98,
    ↪ 0.8)': 'rgba(119,221,118, 0.8)', '#F1F1F1'],
borderColor: [(finalHabitData < 28) ? 'rgba(255, 105, 98, 0.8)
    ↪ ': '#77DD76', '#F1F1F1']
}]
},
plugins: [{
beforeInit: (chart) => {
const dataset = chart.data.datasets[0];
chart.data.labels = [dataset.label];
dataset.data = [dataset.percent, 100 - dataset.percent];
}
},
{

beforeDraw: (chart, args, options) => {
// console.log(chart.width)
var width = chart.width, height = chart.height, ctx = chart.ctx
    ↪ ;
ctx.restore();
var fontSize = (height / 150).toFixed(2);
ctx.font = fontSize + "em sans-serif";
ctx.fillStyle = "#000000";
ctx.textBaseline = "middle";
// var text = chart.data.datasets[0].percent + "%",
var text = finalHabitData + "%"
textX = Math.round((width - ctx.measureText(text).width) / 2),
textY = height / 2;
ctx.fillText(text, textX, textY);
ctx.save();
}
},

],
options: {
plugins: {
legend: {
display: false,
},
tooltip: {
callbacks: {
label: function(context) {
let label = "Sleep: " + habitData + " hrs";
return label;
}
}
}
},
maintainAspectRatio: false,
cutout: '75%',
rotation: Math.PI / 2,
```

```
}
});


// Exercise Chart
var myChartCircle = new Chart('myDonut2', {
type: 'doughnut',
data: {
datasets: [{
percent: finalExerciseHabitData,
backgroundColor: [(finalExerciseHabitData < 8) ? 'rgba(255,
      ↪ 105, 98, 0.8)': 'rgba(119,221,118, 0.8)', '#F1F1F1'],
borderColor: [(finalExerciseHabitData < 8) ? 'rgba(255, 105,
      ↪ 98, 0.8)': '#77DD76', '#F1F1F1']
}]
},
plugins: [{
beforeInit: (chart) => {
const dataset = chart.data.datasets[0];
chart.data.labels = [dataset.label];
dataset.data = [dataset.percent, 100 - dataset.percent];
}
},
{

beforeDraw: (chart, args, options) => {
// console.log(chart.width)
var width = chart.width, height = chart.height, ctx = chart.ctx
      ↪ ;
ctx.restore();
var fontSize = (height / 150).toFixed(2);
ctx.font = fontSize + "em sans-serif";
ctx.fillStyle = "#000000";
ctx.textBaseline = "middle";
// var text = chart.data.datasets[0].percent + "%",
var text = finalExerciseHabitData + " %";
textX = Math.round((width - ctx.measureText(text).width) / 2),
textY = height / 2;
ctx.fillText(text, textX, textY);
ctx.save();
}
},

],
options: {
plugins: {
legend: {
display: false,
},
tooltip: {
callbacks: {
label: function(context) {
let label = "Exercise: " + exerciseData + " mins";
return label;
}
}
}
},
maintainAspectRatio: false,
cutout: '75%',
rotation: Math.PI / 2,
}
});


// Water Intake Chart
var myChartCircle = new Chart('myDonut3', {
type: 'doughnut',
data: {
datasets: [{
percent: finalWaterIntakeHabitData,
backgroundColor: [(finalWaterIntakeHabitData < 40) ? 'rgba(255,
      ↪  105, 98, 0.8)': 'rgba(119,221,118, 0.8)', '#F1F1F1'],
borderColor: [(finalWaterIntakeHabitData < 40) ? 'rgba(255,
      ↪ 105, 98, 0.8)': '#77DD76', '#F1F1F1']
}]
},
plugins: [{
beforeInit: (chart) => {
const dataset = chart.data.datasets[0];
chart.data.labels = [dataset.label];
dataset.data = [dataset.percent, 100 - dataset.percent];
}
},
{

beforeDraw: (chart, args, options) => {
// console.log(chart.width)
var width = chart.width, height = chart.height, ctx = chart.ctx
      ↪ ;
ctx.restore();
var fontSize = (height / 150).toFixed(2);
ctx.font = fontSize + "em sans-serif";
ctx.fillStyle = "#000000";
ctx.textBaseline = "middle";
// var text = chart.data.datasets[0].percent + "%",
var text = finalWaterIntakeHabitData + " %";
textX = Math.round((width - ctx.measureText(text).width) / 2),
textY = height / 2;
ctx.fillText(text, textX, textY);
ctx.save();
}
},

],
options: {
plugins: {
legend: {
display: false,
},
tooltip: {
callbacks: {
label: function(context) {
let label = "Water Intake: " + waterIntakeData + " glasses";
return label;
}
}
}
},
maintainAspectRatio: false,
cutout: '75%',
rotation: Math.PI / 2,
}
});



function revealTextBox() {
var textBoxContainer = document.getElementById("
      ↪ textBoxContainer");
var thoughtsDiv = document.getElementById("thoughtsDiv");
var revealButton = document.getElementById("revealButton");

textBoxContainer.classList.remove("hidden");
thoughtsDiv.classList.add("hidden");
revealButton.classList.add("hidden");
}
```

Listing 42: static/main.js

```
// Script to open and close sidebar
function w3_open() {
  document.getElementById("mySidebar").style.display = "block";
  document.getElementById("myOverlay").style.display = "block";
}

function w3_close() {
  document.getElementById("mySidebar").style.display = "none";
  document.getElementById("myOverlay").style.display = "none";
}

// Modal Image Gallery
function onClick(element) {
  document.getElementById("img01").src = element.src;
  document.getElementById("modal01").style.display = "block";
  var captionText = document.getElementById("caption");
  captionText.innerHTML = element.alt;
}
```

Listing 43: static/mood.js

```
// MOOD BAR CHART
const ctx = document.getElementById('moodBarChart');
const mood_labels = ["Great", "Good", "Okay", "Bad", "Awful"]

new Chart(ctx, {
       type: 'bar',
       data: {
         labels: mood_labels,
         datasets: [{
           axis: 'y',
           label: 'Mood Count',
           data: moodCount,
           fill: false,
           backgroundColor: [
             'rgba(119,221,118, 0.2)',
             'rgba(189, 231, 189, 0.2)',
             'rgba(254,225,145, 0.2)',
             'rgba(255,182,179, 0.2)',
             'rgba(255, 105, 98, 0.2)'
           ],
           borderColor: [
             'rgb(119,221,118)',
             'rgb(189, 231, 189)',
             'rgb(254,225,145)',
             'rgb(255,182,179)',
             'rgb(255, 105, 98)',
           ],
           borderWidth: 1
         }]
       },
       options: {
           indexAxis: 'y',
```

```
            scales: {
                   x: {
                     beginAtZero: true, // Start the x-axis at
                           ↪ zero
                     precision: 0 // Display only integer values
                   }
                 },
               plugins: {
            legend: {
               display: false,
            }
        },
        scales: {
          x: {
            beginAtZero: true, // Start the y-axis at zero
            ticks: {
                       stepSize: 1,
                       callback: function (value) {
                         if (Number.isInteger(value)) {
                           return value;
                         }
                       }
                     }
          }
        }

        },
});
```

Listing 44: static/slider.js

```
var slider = document.getElementById("slider");
var sliderValue = document.getElementById("sliderValue");

slider.addEventListener("input", function() {
  sliderValue.textContent = slider.value;
});
```

Listing 45: static/slideshow.js

```
function plusDivs(n) {
  showDivs(slideIndex += n);
}

function showDivs(n) {
  var i;
  var x = document.getElementsByClassName("mySlides");
  if (n > x.length) {slideIndex = 1}
  if (n < 1) {slideIndex = x.length}
  for (i = 0; i < x.length; i++) {
     x[i].style.display = "none";
  }
  x[slideIndex-1].style.display = "block";
}
```

Listing 46: static/style.css

```
body, h1, h2, h3, h4, h5, h6 {
        font-family: 'K2D', sans-serif;
}

body {
        font-size: 16px;
}

a {
        text-decoration: none;
}


/* div */

.w3-card {
        width:100%;
        background: #FFFFFF;
        box-shadow: 0 2px 2px 0 rgba(0, 0, 0, 0.06), 0 1px 8px 0
              ↪  rgba(0, 0, 0, 0.06);
}

input[type="radio"] {
        display: none;
}


/* buttons */

.w3-button {
        border-radius: 5px;
}

.add-btn {
        background: #00CF95;
        border: 2px solid #00CF95;
        color: white;
        box-shadow: 0 2px 2px 0 rgba(0, 0, 0, 0.06), 0 1px 8px 0
              ↪  rgba(0, 0, 0, 0.06);
}

.disabled-btn {
        background: lightgrey;
        border: 2px solid lightgrey;
        color: white;
        padding-top:8px;padding-bottom:8px;
}

.cancel-btn {
        color: #00CF95;
        border: 2px solid #00CF95;
        background: #FFFFFF;
}

.chat-btn {
        border-radius: 15px;
        padding: 4px 8px;
        background: #fff;
        border: 2px solid #00CF95;
        margin-bottom: 8px;
}

.chat-btn-div {
        margin-left: 8px;
        max-width: 270px;
}

/* any */

.no-margin {
        margin: 0;
}

.hidden {
        display: none;
}

/* slider */

.slider-container {
        width: 238px;
}

.slider {
  -webkit-appearance: none;
  width: 100%;
  height: 15px;
  border-radius: 15px;
  background: #F5F5F5;
  outline: none;
  opacity: 0.7;
  -webkit-transition: .2s;
  transition: opacity .2s;
  overflow: hidden;
  box-shadow: 0 0 0 2px #F5F5F5;
}

.slider:hover {
  opacity: 1;
}

:root {
  --thumb-color: #04AA6D; /* default thumb color */
  --thumb-shadow-color: -208px 0 0 200px #04AA6D; /* default
        ↪ thumb shadow color */
}

.slider::-webkit-slider-thumb {
  -webkit-appearance: none;
  appearance: none;
  width: 15px;
  height: 15px;
  border-radius: 50%;
  cursor: pointer;
  border: 1px solid #fff;
  background: var(--thumb-color);
  box-shadow: var(--thumb-shadow-color);

}

.slider::-moz-range-thumb {
  width: 25px;
  height: 25px;
  border-radius: 50%;
  background: #04AA6D;
  cursor: pointer;
}

/* CSS for success message */
.message.success {
  background-color: #dff0d8;
  color: #3c763d;
  padding: 10px;
  margin-bottom: 10px;
}
```

```
/* CSS for error message */
.message.error {
  background-color: #f2dede;
  color: #a94442;
  padding: 10px;
  margin-bottom: 10px;
}

.checkmark-button {
       border-radius: 15px;
       padding: 4px 8px;
       background: #fff;
       border: 2px solid #00CF95;
       margin-bottom: 8px;
  cursor: pointer;
}

.checkmark-button.checked {
  background-color: #00CF95;
  color: white;
}

.checkmark-button.checked::after {
  content: " \2713";
  color: white;
  font-size: 14px;
}

/* header and sidebar */

#header_large {
       z-index: 4;
       background: #FFFFFF;
       box-shadow: 0 2px 2px 0 rgba(0, 0, 0, 0.06), 0 1px 8px 0
            ↪  rgba(0, 0, 0, 0.06);
}

#header_large span {
       color: #6D0AD3 !important;
}

.logo-header {
       width: 14px !important;
       height: 14px !important;
       margin-bottom: 3px;
}

#header_small, #header_small a {
       background: #6D0AD3 !important;
}

#header_small {
       padding-top: 8px;
       padding-bottom: 8px;
}

.w3-sidebar {
       z-index: 3;
       width: 330px;
       font-weight: bold;
       background: #6D0AD3 !important;
       margin-top: 43px;
}

.w3-main {
       margin-left: 330px;
       padding-top: 43px !important;
}

.profile-div {
       width: 20%;
}

.responsive-profile {
       border: 2px solid white;
       border-radius: 50%;
       width: 100%;
       object-fit: cover;
       aspect-ratio: 1/1;
}

.name-div {
       width: 75%;
       padding-left: 10px;
       line-height: 10px;
}

.name-div h6 {
       font-size: 14px;
}

.menu-items, .settings-items {
       margin-top: 30px;
}

.w3-bar-block h6 {
       font-size: 12px;
       font-weight: bold;
}

#myOverlay {
       cursor:pointer;
}

.w3-bar-item {
       text-decoration: none !important;
       padding: 8px 16px !important;
}

.w3-bar-item:hover {
       background: white !important;
}

.w3-bar-item:hover span, .w3-bar-item:hover i {
  color: black !important;
}

.nav-iconify {
       color: #D1FFA2 !important;
       width: 20px;
       height: 20px;
       position: relative;
       top: 4px;
       margin-right: 2px;
}

.back-icon {
       width: 12px;
       height: 12px;
       position: relative;
       top: 2px;
       margin-right: 2px;
}

.item-name {
       font-size: 14px !important;
}

/* chat */

.bot-image-col, .user-image-col {
       width: 48px;
}

.bot-image, .user-image {
       border: 1px solid white;
       border-radius: 50%;
  object-fit: cover;
  aspect-ratio: 1/1;
       width: 100%;
       box-shadow: 0 2px 2px 0 rgba(0, 0, 0, 0.06), 0 1px 8px 0
            ↪  rgba(0, 0, 0, 0.06);
}

.message-div {
       max-width: 400px;
       max-height: 300px;
}

.bot-message-div {
       margin-left: 8px;
       background: #F5F5F5;
       color: black;
       box-shadow: 0 2px 2px 0 rgba(0, 0, 0, 0.06), 0 1px 8px 0
            ↪  rgba(0, 0, 0, 0.06);
}

.user-message-div {
       margin-right: 8px;
       background: #D1FFA2;
       color: black;
       box-shadow: 0 2px 2px 0 rgba(0, 0, 0, 0.06), 0 1px 8px 0
            ↪  rgba(0, 0, 0, 0.06);
}

/* home */

.home-div {
       margin-top: 30px;
       max-width: 700px;
}

.home-bot-image-col {
       width: 50px;
}

.home-bot-image {
       border: 0.5px solid white;
       border-radius: 50%;
  object-fit: cover;
  aspect-ratio: 1/1;
       width: 100%;
       box-shadow: 0 2px 2px 0 rgba(0, 0, 0, 0.06), 0 1px 8px 0
            ↪  rgba(0, 0, 0, 0.06);
}

.convo-div {
```

```
        background: #F5F5F5;
        box-shadow: 0 2px 2px 0 rgba(0, 0, 0, 0.06), 0 1px 8px 0
            ↪ rgba(0, 0, 0, 0.06);
}

.convo-bot-img-col {
        width: 48px;
}

.convo-msg-div {
        padding-left: 10px;
        line-height: 10px;
}

.bot-message {
        font-size: 12px !important;
        color: #616161 !important;
}

.convo-date {
        font-size: 10px !important;
        color: #616161 !important;
        line-height: 10px !important;
}


/* sign in */

.logo-signin {
        width: 70px;
}

.signin-form-div {
        margin-top: 16px;
        max-width: 450px;
}

.field-name {
        margin-top: 0;
        margin-bottom: 4px;
        font-size: 14px;
}

.forgot-pw-name {
        color: grey;
        font-size: 12px;
        margin-top: 2px;
        text-decoration: none;
}

.field-input {
        border-radius: 5px;
        font-size: 12px;
        margin-bottom: 0 !important;
        background: #F5F5F5;
}

.field-input:focus, .diary-input:focus {
  outline: 4px solid #6D0AD3 !important;
}

.diary-input {
        border-radius: 5px;
        font-size: 12px;
        margin-bottom: 0 !important;
        background: white;
}

.signin-btn {
        background: #00CF95;
        color: white;
        width: 100%;
}

.no-acct-text {
        font-size: 12px;
        margin-bottom: 0;
}

.signup-link {
        color: #00CF95;
        font-size: 12px;
        text-decoration: none;
}


/* signout */

.signout-div {
        margin-top: 16px;
        max-width: 400px;
}

.signout-div p {
        font-size: 16px;
}


/* diary */

.diary-mood-icon {
        width: 20px;
        height: 20px;
        position: relative;
        top: 4px;
        margin-right: 2px;
}

.mood-wrrr-icon {
        color: #FF6962;
}

.mood-sad-icon {
        color: #FFB6B3;
}

.mood-empty-icon {
        color: #FEE191;
}

.mood-smile-icon {
        color: #BDE7BD;
}

.mood-smile-beam-icon {
        color: #77DD76;
}

.rating-icon {
        font-size: 20px;
        color: gold;
}

.chart-name {
        margin-bottom: 8px;
        font-size: 14px;
}

.donut-canvas {
        width: 100%;
        aspect-ratio: 1/1;
        height: 150px;
}

.diary-entry-mood-icon {
        width: 70%;
        height: 70%;
}

/* mood tracker */

.title-icon {
        font-size: 18px;
        position: relative;
        top: 3px;
}

.mood-tracker-icon {
        font-size: 35px;
        position: relative;
        top: 4px;
}


/* strategy */

.strategy-box-div {
        max-width: 400px;
}

.strategy-box {
        background: #F5F5F5;
        color: black;
        box-shadow: 0 2px 2px 0 rgba(0, 0, 0, 0.06), 0 1px 8px 0
            ↪ rgba(0, 0, 0, 0.06);
        margin-left: 28px !important;
        margin-right: 28px !important;
        width: 210px;
}

.strategy-image-col {
        width: 50px;
}

.strategy-image {
        border-radius: 50%;
  object-fit: cover;
  aspect-ratio: 1/1;
        width: 100%;
}

.strategy-text {
        padding-left: 8px;
        overflow: hidden;
}

.mySlides {
        display: none;
```

```
}

.slideshow-btn {
       color: black;
       background: transparent !important;
       padding:8px;
}

.view-strategy-div {
       max-width: 550px !important;
       margin: 0 auto;
}

.view-strategy-img-div {
       width: 100px;
       height: 100px;
       margin: 0 auto;
}

.rating-box {
  background: #fff;
  padding: 16px;
  box-shadow: 0 5px 10px rgba(0, 0, 0, 0.05);
  max-width: 250px !important;
  margin: 0 auto;
}

.rating-box header {
  font-size: 22px;
  font-weight: 500;
  text-align: center;
  margin-top: 16px;
}

.star-icon {
       color: #e6e6e6;
  font-size: 35px;
  cursor: pointer;
  transition: color 0.2s ease;
}

label .star-icon.active {
       color: gold;
}


/* resources */

.article-box {
       padding: 0 0 5px;
       background: #F5F5F5;
       color: black;
       box-shadow: 0 2px 2px 0 rgba(0, 0, 0, 0.06), 0 1px 8px 0
              ↪  rgba(0, 0, 0, 0.06);
       margin-left: 28px !important;
       margin-right: 28px !important;
       width: 210px;
}

.article-img {
       height: 100px;
       width: 100%;
       object-fit: cover;
       aspect-ratio: 1/1;
       border-radius: 4px 4px 0px 0px;
}

.article-img-cover {
       height: 150px !important;
}

/* media screen */

@media screen and (max-width: 380px) {
  .chart-name {
       font-size: 12px;
       margin-bottom: 4px;
  }

  .mood-tracker-icon {
              font-size: 20px;
              position: relative;
              top: 4px;
       }

}

@media screen and (max-width: 420px) {
  .w3-sidebar {
       width: 100%;
  }

}

@media screen and (max-width: 640px) {
  .bot-message-div, .user-message-div {
              max-width: 238px;
              max-height: 300px;
       }

       .rating-icon {
              width: 18px;
       }

       .strategy-box {
              width: 190px;
       }

       .strategy-image-col {
              width: 40px;
       }

       .slider-container {
       width: 200px;
  }

}

@media screen and (max-width: 992px) {

       .w3-sidebar {
              margin-top: 0px;
       }

       .w3-main {
         padding-top: 68px !important;
       }
}
```

A..2 RASA Assistant

Listing 47: rasa_bot/config.yml

```
recipe: default.v1
assistant_id: 20230516-112913-warm-plisse
language: en

pipeline:
  - name: WhitespaceTokenizer
  - name: LexicalSyntacticFeaturizer
  - name: CountVectorsFeaturizer
    analyzer: word
  - name: DucklingEntityExtractor
    url: http://localhost:8000
    dimensions:
    - number
    - time
  - name: DIETClassifier
    epochs: 250
    # epochs: 100
  - name: EntitySynonymMapper
  - name: FallbackClassifier
    # If the highest ranked intent has a confidence lower than
          ↪ the threshold than
    # the NLU pipeline predicts an intent ‘nlu_fallback‘ which
          ↪ you can then use in
    # stories / rules to implement an appropriate fallback.
    threshold: 0.3

policies:
  - name: RulePolicy
    core_fallback_threshold: 0.3
    # Name of the action which should be predicted if no rule
          ↪ matched.
    core_fallback_action_name: "action_default_fallback"
    # If ‘True‘ ‘core_fallback_action_name‘ is predicted in case
          ↪  no rule matched.
    enable_fallback_prediction: True
```

Listing 48: rasa_bot/credentials.yml

```
rest:
# # you don’t need to provide anything here - this channel
      ↪ doesn’t
# # require any credentials

rasa:
  url: "http://localhost:5002/api"
```

Listing 49: rasa_bot/domain.yml

```
version: ’3.1’
intents:
- ’*’
- affirm
- bot_challenge
- deny
- inform
- talk_to_kaya
# Sign Up
- sign_up
```

```
- confirm_set_notification
# Mood Tracker
- track_mood
- add_mood
- confirm_save_mood
# Habit Tracker
- track_habit
- add_habits_to_track
- add_habit_entry
- confirm_save_habit_entry
- edit_habits_to_tracker
- save_habits_to_tracker
# Strategies
- view_strategies
- learn_strategies
- discover_more_strategies
- viewed_strategy
- try_more_strategies
# Resources
- view_resources
- read_articles
- viewed_article
- suggest_more_articles
# Diary
- view_diary


entities:
- trigger
- mood
- sleep
- exercise
- water_intake
- habit
- time
- number

slots:
  mood:
    type: text
    influence_conversation: false
    mappings:
    - type: from_entity
      entity: mood
      conditions:
      - active_loop: mood_entry_form
        requested_slot: mood
  mood_trigger:
    type: text
    influence_conversation: false
    mappings:
    - type: from_entity
      entity: trigger
      conditions:
      - active_loop: mood_entry_form
        requested_slot: mood_trigger
  sleep:
    type: float
    influence_conversation: false
    mappings:
    - type: from_entity
      entity: number
      conditions:
      - active_loop: habit_entry_form
        requested_slot: sleep
  exercise:
    type: float
    influence_conversation: false
    mappings:
    - type: from_entity
      entity: number
      conditions:
      - active_loop: habit_entry_form
        requested_slot: exercise
  water_intake:
    type: float
    influence_conversation: false
    mappings:
    - type: from_entity
      entity: number
      conditions:
      - active_loop: habit_entry_form
        requested_slot: water_intake
  check_in_time:
    type: text
    influence_conversation: false
    mappings:
    - type: from_entity
      entity: time
      conditions:
      - active_loop: time_prompt_form
        requested_slot: check_in_time
  insights_time:
    type: text
    influence_conversation: false
    mappings:
    - type: from_entity
      entity: time
      conditions:
      - active_loop: time_prompt_form
        requested_slot: insights_time

forms:
  mood_entry_form:
    required_slots:
    - mood
    - mood_trigger
  habit_entry_form:
    required_slots:
    - sleep
    - exercise
    - water_intake
  time_prompt_form:
    required_slots:
    - check_in_time
    - insights_time

actions:
- utter_ask_menu
- action_reset_slots
- action_save_mood
- utter_ask_save_mood
- utter_going_to_mood_tracker
- utter_greet
- utter_cheer_up
- utter_did_that_help
- utter_happy
- utter_goodbye
- utter_iamabot
- action_hello_world
- utter_menu
- utter_ask_mood_entry_form_mood
- utter_ask_mood_entry_form_mood_trigger
- utter_going_to_habit_tracker
- validate_habit_entry_form
- action_check_habits_to_track
- action_ask_sleep
- action_ask_exercise
- action_ask_water_intake
- action_save_habit_entry
- action_suggest_habits_to_track
- action_save_habits_to_track
# - utter_would_you_like_to_add_habit_entry
- action_welcome_message
- utter_i_am_kaya
- utter_might_check_out_app
- utter_app_purpose
- utter_ask_to_set_notification
- action_save_notification_settings
- action_ask_check_in_time
- action_ask_insights_time
- utter_going_to_strategies
- utter_strategies_disclaimer
- utter_encourage_to_mhp
- utter_would_you_like_to_continue
- utter_encourage_to_experiment_strategies
- utter_remind_to_be_patient
- utter_remember_me_strategies
- utter_suggest_well_being_strategies
- action_suggest_well_being_strategies
- utter_thank_time_to_view_strategy
- utter_would_you_like_to_try_more_strategies
- utter_going_to_resources
- utter_reading_article_benefits
- utter_suggest_articles
- action_suggest_articles
- utter_hope_to_be_helpful
- utter_thank_time_to_view_article
- utter_motivate_user
- utter_would_you_like_to_read_more_strategies
- utter_going_to_diary

responses:
  utter_greet:
  - text: Hey! How are you?
  utter_cheer_up:
  - text: 'Here is something to cheer you up:'
    image: https://i.imgur.com/nGF1K8f.jpg
  utter_did_that_help:
  - text: Did that help you?
  utter_happy:
  - text: Great, carry on!
  utter_goodbye:
  - text: Bye
  utter_iamabot:
  - text: I am a bot, powered by Rasa.
  utter_default:
  - text: Sorry I didn't get that. I am only able to assist with
      ↪  certain options. For now, what would you like to do
      ↪ ?
    buttons:
    - title: track my mood
      payload: track my mood
    - title: track my habit
      payload: track my habit
    - title: view my strategies
      payload: view my strategies
    - title: view my diary
```

```
    payload: view my diary
  - title: view resources
    payload: view resources
utter_menu:
- text: Hello!  What would you like to do?
  buttons:
  - title: track my mood
    payload: track my mood
  - title: track my habit
    payload: track my habit
  - title: view my strategies
    payload: view my strategies
  - title: view my diary
    payload: view my diary
  - title: view resources
    payload: view resources
utter_ask_menu:
- text: For now, what would you like to do?
  buttons:
  - title: track my mood
    payload: track my mood
  - title: track my habit
    payload: track my habit
  - title: view my strategies
    payload: view my strategies
  - title: view my diary
    payload: view my diary
  - title: view resources
    payload: view resources
utter_going_to_mood_tracker:
- text: Going to Mood Tracker...
utter_ask_mood_entry_form_mood:
- text: How would you describe your current mood?
  buttons:
  - title: great
    payload: great
  - title: good
    payload: good
  - title: okay
    payload: okay
  - title: bad
    payload: bad
  - title: awful
    payload: awful
utter_ask_mood_entry_form_mood_trigger:
- text: What do you think might have caused you to feel this
      ↪ way?
  buttons:
  - title: school
    payload: school
  - title: family
    payload: family
  - title: relationship
    payload: relationship
  - title: friends
    payload: friends
utter_ask_save_mood:
- text: Would you like to save it?
  buttons:
  - title: yes, save my mood
    payload: yes, save my mood
  - title: no, don't save
    payload: no, don't save
utter_going_to_habit_tracker:
- text: Going to Habit Tracker...
utter_i_am_kaya:
- text: I'm Kaya, your personal bot assistant.
utter_might_check_out_app:
- text: If you're looking to improve your well-being, you
      ↪ might want to check out this app.
utter_app_purpose:
- text: This app offers a range of features, including mood
      ↪ and habit tracking, well-being strategy development,
      ↪  monthly stats, daily insights, journaling, and
      ↪ access to mental health resources.
utter_ask_to_set_notification:
- text: To provide you with a personalized experience, I would
      ↪  like to ask you a few questions. Your answers will
      ↪ help me better understand your preferences and needs
      ↪ .
  buttons:
  - title: set notifications
    payload: set notifications
utter_going_to_strategies:
- text: Going to My Strategies...
utter_strategies_disclaimer:
- text: I want to support you in finding effective well-being
      ↪ strategies. But I also want to remind you that I can
      ↪  only suggest well-being strategies that may be
      ↪ helpful for you to try.
utter_encourage_to_mhp:
- text: However, if you ever feel like you need more support
      ↪ or guidance, I strongly encourage you to seek
      ↪ support from a mental health professional.
utter_would_you_like_to_continue:
- text: Would you like to continue?
  buttons:
  - title: yes, discover more strategies
    payload: yes, discover more strategies
  - title: no, maybe next time
    payload: no, maybe next time
utter_encourage_to_experiment_strategies:
- text: Well-being strategies are not one-size-fits-all.
      ↪ Experimenting with well-being strategies can help
      ↪ you discover what works best for you.
utter_remind_to_be_patient:
- text: It's important to remember that developing effective
      ↪ strategies takes time and patience. Don't be
      ↪ discouraged if some strategies don't work for you,
      ↪ that's completely normal.
utter_remember_me_strategies:
- text: Remember, I am here to support you in this process and
      ↪  help you find the strategies that work best for you
      ↪ .
utter_suggest_well_being_strategies:
- text: 'Here are some well-being strategies that might be
      ↪ worth trying:'
utter_thank_time_to_view_strategy:
- text: Thank you for taking the time to view this strategy! I
      ↪  am proud of you for taking that step.
utter_would_you_like_to_try_more_strategies:
- text: Would you like to try other well-being strategies?
  buttons:
  - title: try more strategies
    payload: try more strategies
  - title: maybe next time
    payload: maybe next time
utter_going_to_resources:
- text: Going to Resources...
utter_reading_article_benefits:
- text: Reading an article about mental health can increase
      ↪ your knowledge and understanding of various mental
      ↪ health issues, symptoms, and treatments.
utter_suggest_articles:
- text: 'Here are some articles that you may find useful:'
utter_hope_to_be_helpful:
- text: I sincerely hope that the resources I've shared will
      ↪ be valuable in your journey towards enhancing your
      ↪ well-being.
utter_thank_time_to_view_article:
- text: Thank you for taking the time to view this article! I
      ↪ appreciate your interest and dedication to learning
      ↪ more about this important topic.
utter_motivate_user:
- text: I'm here to support you on your journey. Keep up the
      ↪ great work!
utter_would_you_like_to_read_more_strategies:
- text: Would you like to read more articles?
  buttons:
  - title: yes, suggest more articles
    payload: yes, suggest more articles
  - title: maybe next time
    payload: maybe next time
# utter_would_you_like_to_add_habit_entry:
# - text: What would you like to do?
# buttons:
# - title: add habit entry
# payload: add habit entry
# - title: track my habit
# payload: track my habit
utter_going_to_diary:
- text: Going to Diary...
session_config:
  session_expiration_time: 60
  carry_over_slots_to_new_session: false
```

Listing 50: rasa_bot/endpoints.yml

```
action_endpoint:
 url: "http://localhost:5055/webhook"
```

Listing 51: rasa_bot/actions/actions.py

```
from typing import Any, Text, Dict, List

from rasa_sdk.forms import FormValidationAction
from rasa_sdk import Action, Tracker
from rasa_sdk.executor import CollectingDispatcher
from rasa_sdk.events import SlotSet
import requests, json


ALLOWED_MOOD = ["great", "good", "okay", "bad", "awful"]
ALLOWED_MOOD_TRIGGERS = ["school", "family", "relationship", "
      ↪ friends"]


class ActionHelloWorld(Action):
    def name(self) -> Text:
        return "action_hello_world"

    def run(
        self,
        dispatcher: CollectingDispatcher,
```

```
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        dispatcher.utter_message(text="Hello World!")

        return []


class ActionWelcomeMessage(Action):
    def name(self) -> Text:
        return "action_welcome_message"

    def run(
        self,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        user_id = tracker.sender_id

        # Make an HTTP request to the Django API endpoint
        api_endpoint = "http://127.0.0.1:8000/api/get_user/"
        data = {"user_id": user_id}
        response = requests.post(api_endpoint, json=data)

        # Handle the response from the API
        if response.status_code == 200:
            json_data = response.json()
            user = json_data["user"]
            dispatcher.utter_message(text=f"Welcome, {user}! ")
            return []

        dispatcher.utter_message(text="Invalid JSON response")
        return []


class AskForCheckInTimeAction(Action):
    def name(self) -> Text:
        return "action_ask_check_in_time"

    def run(
        self,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        payload = {
            "type": "notifications",
            "text": "At what time would you prefer me to prompt
                ↪ you to check in with your mood and habits
                ↪ ?",
        }
        dispatcher.utter_message(json_message=payload)
        return []


class AskForInsightsTimeAction(Action):
    def name(self) -> Text:
        return "action_ask_insights_time"

    def run(
        self,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        payload = {
            "type": "notifications",
            "text": "At what time would you prefer me to provide
                ↪ you with the daily insights?",
        }
        dispatcher.utter_message(json_message=payload)
        return []


class ActionSaveNotificationSettings(Action):
    def name(self) -> Text:
        return "action_save_notification_settings"

    def run(
        self,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        user_id = tracker.sender_id
        check_in_time = tracker.get_slot("check_in_time")
        insights_time = tracker.get_slot("insights_time")

        # Make an HTTP request to the Django API endpoint
        api_endpoint = "http://127.0.0.1:8000/api/
            ↪ save_notifications_settings/"
        data = {
            "user_id": user_id,
            "check_in_time": check_in_time,
            "insights_time": insights_time,
        }
        response = requests.post(api_endpoint, json=data)

        # Handle the response from the API
        if response.status_code == 200:
            # slots_to_reset = ["mood", "mood_trigger"]
            # reset_events = [SlotSet(slot, None) for slot in
                ↪ slots_to_reset]

            dispatcher.utter_message(
                text="Your responses have been recorded and will
                    ↪ be used to personalize your experience
                    ↪ ."
            )
            # return reset_events
        else:
            dispatcher.utter_message(text="Failed to save slot
                ↪ value.")

        return []


class ValidateMoodEntryForm(Action):
    def name(self) -> Text:
        return "validate_mood_entry_form"

    def validate_mood(
        self,
        slot_value: Any,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        """validate ‘mood‘ value"""

        if slot_value not in ALLOWED_MOOD:
            dispatcher.utter_message(
                text=f"I don’t recognize that mood. I can only
                    ↪ recognize {’/’.join(ALLOWED_MOOD)}."
            )
            return {"mood": None}
        return {"mood": slot_value}

    def validate_mood_trigger(
        self,
        slot_value: Any,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        """validate ‘mood trigger‘ value"""

        if slot_value not in ALLOWED_MOOD_TRIGGERS:
            dispatcher.utter_message(
                text=f"I don’t recognize that mood trigger. I can
                    ↪  only recognize {’/’.join(
                    ↪ ALLOWED_MOOD_TRIGGERS)}."
            )
            return {"mood_trigger": None}
        return {"mood_trigger": slot_value}


class ActionSaveMood(Action):
    def name(self) -> Text:
        return "action_save_mood"

    def run(
        self,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        user_id = tracker.sender_id
        mood = tracker.get_slot("mood")
        mood_trigger = tracker.get_slot("mood_trigger")

        # Make an HTTP request to the Django API endpoint
        api_endpoint = "http://127.0.0.1:8000/api/save_mood/"
        data = {"user_id": user_id, "mood": mood, "mood_trigger
            ↪ ": mood_trigger}
        response = requests.post(api_endpoint, json=data)

        # Handle the response from the API
        if response.status_code == 200:
            # slots_to_reset = ["mood", "mood_trigger"]
            # reset_events = [SlotSet(slot, None) for slot in
                ↪ slots_to_reset]
            print("\nMood saved")
            dispatcher.utter_message(
                text="Thank you for your response. It has been
                    ↪ saved."
            )
            # return reset_events
        else:
            dispatcher.utter_message(text="Failed to save slot
                ↪ value.")

        return []


class ActionResetSlots(Action):
    def name(self) -> Text:
```

```
        return "action_reset_slots"

    def run(
        self,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        # Reset all slots by creating SlotSet events with None
            ↪ as the value
        reset_slots = [SlotSet(slot, None) for slot in tracker.
            ↪ slots.keys()]
        print("\nSlots reset")
        # Return the events to be processed by the dialogue
            ↪ engine
        return reset_slots


class ValidateHabitEntryForm(FormValidationAction):
    def name(self) -> Text:
        return "validate_habit_entry_form"

    async def required_slots(
        self,
        domain_slots: List[Text],
        dispatcher: "CollectingDispatcher",
        tracker: "Tracker",
        domain: "DomainDict",
    ) -> List[Text]:
        print("\nCHECKING REQUIRED SLOTS\n")
        user_id = tracker.sender_id

        api_endpoint = "http://127.0.0.1:8000/api/get_habits/"
        data = {"user_id": user_id}
        response = requests.post(api_endpoint, json=data)

        if response.status_code == 200:
            json_data = response.json()
            if "habits" in json_data:
                updated_slots = json_data["habits"]
                updated_slots = [habit.replace(" ", "_") for
                    ↪ habit in updated_slots]
                return updated_slots
            return []

        dispatcher.utter_message(text="Invalid JSON response")
        return []

    def validate_sleep(
        self,
        slot_value: Any,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        """validate ‘sleep‘ value"""

        print("\nVALIDATING SLEEP\n")

        print("\n", slot_value)
        return {"sleep": slot_value}

    def validate_exercise(
        self,
        slot_value: Any,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        """validate ‘exercise‘ value"""

        print("\nVALIDATING EXERCISE\n")
        print("\n", slot_value)
        return {"exercise": slot_value}

    def validate_water_intake(
        self,
        slot_value: Any,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        """validate ‘water_intake‘ value"""

        print("\nVALIDATING WATER INTAKE\n")
        print("\n", slot_value)
        return {"water_intake": slot_value}


class AskForSleepAction(Action):
    def name(self) -> Text:
        return "action_ask_sleep"

    def run(
        self,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        payload = {
            "type": "slider",
            "text": "How many hours did you sleep today?",
            "max": 24,
            "min": 0,
        }
        dispatcher.utter_message(json_message=payload)
        return []


class AskForExerciseAction(Action):
    def name(self) -> Text:
        return "action_ask_exercise"

    def run(
        self,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        payload = {
            "type": "slider",
            "text": "How many minutes did you exercise today?",
            "max": 120,
            "min": 0,
        }
        dispatcher.utter_message(json_message=payload)
        return []


class AskForWaterIntakeAction(Action):
    def name(self) -> Text:
        return "action_ask_water_intake"

    def run(
        self,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        payload = {
            "type": "slider",
            "text": "How many glasses of water did you to drink
                ↪ today?",
            "max": 20,
            "min": 0,
        }
        dispatcher.utter_message(json_message=payload)
        return []


class ActionCheckHabitsToTrack(Action):
    def name(self) -> Text:
        return "action_check_habits_to_track"

    def run(
        self,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        print("\nCHECKING HABITS TO TRACK\n")
        user_id = tracker.sender_id

        dispatcher.utter_message(
            text="Would you like to save it?",
            buttons=[
                {"title": "yes", "payload": "save habit entry"},
                {"title": "no", "payload": "no"},
            ],
        )
        return []


class ActionSaveHabitEntry(Action):
    def name(self) -> Text:
        return "action_save_habit_entry"

    def run(
        self,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        user_id = tracker.sender_id
        slots_to_save = {}
        for slot in tracker.current_slot_values():
            slot_value = tracker.get_slot(slot)
            if slot_value != None:
                print(f"{slot}: {slot_value}")
                # habit_slot[slot] = slot_value
                slots_to_save.update({slot: slot_value})

        # Make an HTTP request to the Django API endpoint
        api_endpoint = "http://127.0.0.1:8000/api/
            ↪ save_habit_entry/"
        data = {"user_id": user_id, "habits": slots_to_save}
        response = requests.post(api_endpoint, json=data)
```

```
        # Handle the response from the API
        if response.status_code == 200:
            dispatcher.utter_message(
                text="Thank you for your response. It has been
                    ↪ saved."
            )
        else:
            dispatcher.utter_message(text="Failed to save slot
                ↪ value.")

        return []


class ActionSuggestHabitsToTrack(Action):
    def name(self) -> Text:
        return "action_suggest_habits_to_track"

    def run(
        self,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        print("\nSUGGEST HABITS TO TRACK\n")
        user_id = tracker.sender_id

        dispatcher.utter_message(
            text="Choose atleast one habit that you would like to
                ↪  track then click Save. ",
            buttons=[
                {"title": "sleep", "payload": "sleep"},
                {"title": "exercise", "payload": "exercise"},
                {"title": "water intake", "payload": "water
                    ↪ intake"},
            ],
        )

        return []


class ActionSaveHabitsToTrack(Action):
    def name(self) -> Text:
        return "action_save_habits_to_track"

    def run(
        self,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        print("\nSAVE HABITS TO TRACK\n")
        user_id = tracker.sender_id

        entities = tracker.latest_message.get("entities")

        if entities:
            entity_values = [entity.get("value") for entity in
                ↪ entities]

            entity_values_text = ", ".join(entity_values)
            # response = f"The entity values are: {
                ↪ entity_values_text}"

            # Make an HTTP request to the Django API endpoint
            api_endpoint = "http://127.0.0.1:8000/api/save_habits
                ↪ /"
            data = {"user_id": user_id, "habits": entity_values}
            response = requests.post(api_endpoint, json=data)

            # Handle the response from the API
            if response.status_code == 200:
                dispatcher.utter_message(
                    text=f"Your response has been saved. The
                        ↪ habit tracker now tracks the
                        ↪ following: {entity_values_text}.",
                    buttons=[
                        {
                            "title": "track my habit",
                            "payload": "track my habit",
                        },
                    ],
                )
            else:
                dispatcher.utter_message(text="Failed to save
                    ↪ habits.")
        else:
            dispatcher.utter_message("No entities found.")

        return []


class ActionSuggestWellBeingStrategies(Action):
    def name(self) -> Text:
        return "action_suggest_well_being_strategies"

    def run(
        self,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        print("\nSUGGEST WELL-BEING STRATEGIES\n")
        user_id = tracker.sender_id

        api_endpoint = "http://127.0.0.1:8000/api/get_strategies
            ↪ /"
        data = {"user_id": user_id}
        response = requests.post(api_endpoint, json=data)

        if response.status_code == 200:
            json_data = response.json()
            if json_data["strategies"]:
                payload = {
                    "type": "strategies",
                    "slides": [],
                }

                for strategy in json_data["strategies"]:
                    print(strategy)
                    temp_dict = {
                        "id": strategy["strategy_id"],
                        "text": strategy["strategy_name"],
                    }
                    payload["slides"].append(temp_dict)

                print("\n", payload)
                dispatcher.utter_message(json_message=payload)
                return []
            else:
                bot_message = "You've already tried all the
                    ↪ available strategies. Great job on
                    ↪ exploring them! If you need further
                    ↪ assistance or have any questions, feel
                    ↪ free to ask. I'm here to help! "
                dispatcher.utter_message(
                    text=bot_message,
                    buttons=[
                        {
                            "title": "view my strategies",
                            "payload": "view my strategies",
                        },
                    ],
                )
                return []

        dispatcher.utter_message(text="Invalid JSON response")
        return []


class ActionSuggestArticles(Action):
    def name(self) -> Text:
        return "action_suggest_articles"

    def run(
        self,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
        print("\nSUGGEST ONLINE ARTICLES\n")
        user_id = tracker.sender_id

        api_endpoint = "http://127.0.0.1:8000/api/get_articles/"
        data = {"user_id": user_id}
        response = requests.post(api_endpoint, json=data)

        if response.status_code == 200:
            json_data = response.json()
            if json_data["articles"]:
                payload = {
                    "type": "resources",
                    "slides": [],
                }

                for article in json_data["articles"]:
                    print(article)
                    temp_dict = {
                        "id": article["article_id"],
                        "text": article["article_title"],
                    }
                    payload["slides"].append(temp_dict)

                print("\n", payload)
                dispatcher.utter_message(json_message=payload)
                return []
            else:
                bot_message = "You've already tried all the
                    ↪ available articles. Great job on
                    ↪ exploring them! If you need further
                    ↪ assistance or have any questions, feel
                    ↪ free to ask. I'm here to help! "
                dispatcher.utter_message(
                    text=bot_message,
                    buttons=[
                        {"title": "go to menu", "payload": "go to
                            ↪ menu"},
                    ],
                )
```

```
            return []

        dispatcher.utter_message(text="Invalid JSON response")
        return []
```

Listing 52: rasa_bot/data/nlu.yml

```
version: "3.1"
nlu:
- intent: affirm
  examples: |
    - yes
    - indeed
    - of course
    - correct

- intent: deny
  examples: |
    - no
    - n
    - never
    - I don't think so
    - don't like that
    - no way
    - not really
    - maybe next time
    - don't save
    - cancel

- intent: bot_challenge
  examples: |
    - are you a bot?
    - are you a human?
    - am I talking to a bot?
    - am I talking to a human?

- intent: sign_up
  examples: |
    - Hello
    - hello
    - hi
    - created new account
    - Hello! I just created an account.
    - created an account

- intent: confirm_set_notification
  examples: |
    - set notifications
    - set notification
    - i want to set my notifications
    - setting notification

- intent: talk_to_kaya
  examples: |
    - talk to kaya
    - talk to the bot
    - talk to a bot
    - go to menu
    - go to main menu
    - menu

- intent: inform
  examples: |
    - [great](mood)
    - [good](mood)
    - [okay](mood)
    - [bad](mood)
    - [awful](mood)
    - [school](trigger)
    - [family](trigger)
    - [relationship](trigger)
    - [friends](trigger)
    - [0](number)
    - [1](number)
    - [2](number)
    - [3](number)
    - [4](number)
    - [5](number)
    - [7](number)
    - [10](number)
    - [12](number)
    - [20](number)
    - [30](number)
    - [50](number)
    - [60](number)
    - [100](number)
    - [2:45 PM](time)
    - [7:00 AM](time)
    - [good](mood)
    - [family](trigger)
    - [great](mood)
    - [friends](trigger)

- intent: track_mood
  examples: |
    - track my mood
    - i want to track my mood
    - i would like to see my mood
    - go to mood tracker

- intent: add_mood
  examples: |
    - add mood
    - add a mood
    - adding mood
    - i want to add mood
    - i will add mood

- intent: confirm_save_mood
  examples: |
    - yes, save mood
    - yes, save my mood
    - Save current mood
    - please save the mood
    - Confirm and save mood

- intent: save_habits_to_tracker
  examples: |
    - [sleep](habit)
    - [exercise](habit)
    - [water intake](habit)
    - [sleep](habit), [exercise](habit)
    - [exercise](habit), [water intake](habit)
    - [sleep](habit), [water intake](habit)
    - [water intake](habit), [sleep](habit), [exercise](habit)
    - [sleep](habit), [exercise](habit), [water intake](habit)

- intent: track_habit
  examples: |
    - track my habit
    - I want to track a habit
    - Start habit tracking
    - habit tracking

- intent: add_habits_to_track
  examples: |
    - add habits to track
    - add habits
    - Add new habits for tracking
    - Include habits in tracker

- intent: add_habit_entry
  examples: |
    - add habit entry
    - habit entry
    - I want to add a habit entry
    - Log habit entry
    - add a new habit entry

- intent: confirm_save_habit_entry
  examples: |
    - save habit entry
    - confirm saving habit entry
    - Save the habit entry
    - saving the created habit entry

- intent: edit_habits_to_tracker
  examples: |
    - edit habits
    - edit habit
    - modify habits to track
    - Change tracked habits

- intent: view_strategies
  examples: |
    - view my strategies
    - i want to view my strategies
    - show me my strategies
    - give me my strategies

- intent: learn_strategies
  examples: |
    - learn well-being strategies
    - learn strategies
    - learning strategies
    - i want to learn strategies
    - i want to learn well-being strategies
    - learn

- intent: discover_more_strategies
  examples: |
    - yes, discover more strategies
    - discover more strategies
    - more strategies
    - know more strategies
    - i want to discover more strategies
    - discover strategy
    - discover strategies

- intent: viewed_strategy
  examples: |
    - try strategy 'Get enough sleep'
    - try strategy 'Eat lots of fruits and vegetable'
    - try strategy 'Meditation'
    - strategy tried 'Meditation'
```

```
- intent: try_more_strategies
  examples: |
    - try more strategies
    - try other strategies
    - i want to try other strategies
    - i want to try more strategies

- intent: view_resources
  examples: |
    - view resources
    - i want to view resources
    - wanna view resources
    - want to view resource
    - view my resources

- intent: read_articles
  examples: |
    - read article
    - read articles
    - i want to read article
    - i want to read articles
    - want to read article
    - read
    - article
    - article read

- intent: viewed_article
  examples: |
    - try article 'Well-being'
    - try article 'Spiritual Well-being'
    - try article 'What is Well-being'
    - try article 'What is Mental Well-being'
    - article tried 'Well-being'

- intent: suggest_more_articles
  examples: |
    - suggest more articles
    - suggest other articles
    - read other articles
    - i want to read other articles
    - i want to read more articles

- intent: view_diary
  examples: |
    - view my diary
    - view diary
    - see my diary
    - open my diary
    - want my diary
```

Listing 53: rasa_bot/data/rules.yml

```
version: "3.1"

rules:

- rule: Say 'I am a bot' anytime the user challenges
  steps:
  - intent: bot_challenge
  - action: utter_iamabot

- rule: Initiate talk to kaya
  steps:
  - intent: talk_to_kaya
  - action: action_reset_slots
  - action: utter_menu

- rule: Go to menu anytime the user denies
  steps:
  - intent: deny
  - action: utter_ask_menu
  - action: action_reset_slots

### Signup Rules

- rule: Disclaimer after sign up
  steps:
  - intent: sign_up
  - action: action_welcome_message
  - action: utter_i_am_kaya
  - action: utter_might_check_out_app
  - action: utter_app_purpose
  - action: utter_ask_to_set_notification

- rule: Activate time prompt form
  steps:
  - intent: confirm_set_notification
  - action: action_reset_slots
  - action: time_prompt_form
  - active_loop: time_prompt_form

- rule: Submit time prompt form
  condition:
  # Condition that form is active.
  - active_loop: time_prompt_form
  steps:
  # Form is deactivated
  - action: time_prompt_form
  - active_loop: null
  - slot_was_set:
    - requested_slot: null
  # The actions we want to run when the form is submitted.
  - action: action_save_notification_settings
  - action: action_reset_slots
  - action: utter_ask_menu

### Mood Tracker Rules
- rule: Go to mood tracker
  steps:
  - intent: track_mood
  - action: utter_going_to_mood_tracker

- rule: Activate mood entry form
  steps:
  - intent: add_mood
  - action: mood_entry_form
  - active_loop: mood_entry_form

- rule: Submit mood entry form
  condition:
  # Condition that form is active.
  - active_loop: mood_entry_form
  steps:
  # Form is deactivated
  - action: mood_entry_form
  - active_loop: null
  - slot_was_set:
    - requested_slot: null
  # The actions we want to run when the form is submitted.
  - action: utter_ask_save_mood

- rule: Save mood entry in the database
  steps:
  - intent: confirm_save_mood
  - action: action_save_mood
  - action: utter_ask_menu
  - action: action_reset_slots

### Habit Tracker Rules
- rule: Go to habit tracker
  steps:
  - intent: track_habit
  - action: utter_going_to_habit_tracker

- rule: Activate habit entry form
  steps:
  - intent: add_habit_entry
  - action: habit_entry_form
  - active_loop: habit_entry_form

- rule: Submit habit entry form
  condition:
  # Condition that form is active.
  - active_loop: habit_entry_form
  steps:
  # Form is deactivated
  - action: habit_entry_form
  - active_loop: null
  - slot_was_set:
    - requested_slot: null
  # The actions we want to run when the form is submitted.
  - action: action_check_habits_to_track
  # - action: utter_ask_save_habit_entry

- rule: Save habit entry in the database
  steps:
  - intent: confirm_save_habit_entry
  - action: action_save_habit_entry
  - action: utter_ask_menu
  - action: action_reset_slots

- rule: Add habits to tracker
  steps:
  - intent: add_habits_to_track
  - action: action_suggest_habits_to_track

- rule: Edit habits to tracker
  steps:
  - intent: edit_habits_to_tracker
  - action: action_suggest_habits_to_track

- rule: Save habits to tracker
  steps:
  - intent: save_habits_to_tracker
  - action: action_save_habits_to_track
  # - action: utter_would_you_like_to_add_habit_entry

### Learn Well-Being Strategies Rules
- rule: Go to my strategies
  steps:
  - intent: view_strategies
  - action: utter_going_to_strategies

- rule: Well-being strategies disclaimer
  steps:
  - intent: learn_strategies
```

```
  - action: utter_strategies_disclaimer
  - action: utter_encourage_to_mhp
  - action: utter_would_you_like_to_continue

- rule: Learn well-being strategies
  steps:
  - intent: discover_more_strategies
  - action: utter_encourage_to_experiment_strategies
  - action: utter_remind_to_be_patient
  - action: utter_remember_me_strategies
  - action: utter_suggest_well_being_strategies
  - action: action_suggest_well_being_strategies

- rule: Viewed strategy
  steps:
  - intent: viewed_strategy
  - action: utter_thank_time_to_view_strategy
  - action: utter_would_you_like_to_try_more_strategies

- rule: Learn more strategies
  steps:
  - intent: try_more_strategies
  - action: utter_suggest_well_being_strategies
  - action: action_suggest_well_being_strategies

### Resource Rules

- rule: Go to resource
  steps:
  - intent: view_resources
  - action: utter_going_to_resources

- rule: Read online articles
  steps:
  - intent: read_articles
  - action: utter_reading_article_benefits
  - action: utter_suggest_articles
  - action: action_suggest_articles
  - action: utter_hope_to_be_helpful

- rule: Viewed article
  steps:
  - intent: viewed_article
  - action: utter_thank_time_to_view_article
  - action: utter_motivate_user
  - action: utter_would_you_like_to_read_more_strategies

- rule: Suggest more articles
  steps:
  - intent: suggest_more_articles
  - action: utter_suggest_articles
  - action: action_suggest_articles

- rule: Go to diary
  steps:
  - intent: view_diary
  - action: utter_going_to_diary
```

Listing 54: rasa_bot/data/stories.yml

```
version: "3.1"

stories:
### Main menu
- story: Talk to kaya path
  steps:
  - intent: talk_to_kaya
  - action: action_reset_slots
  - action: utter_menu

### Mood Tracker Stories
- story: Go to mood tracker and add mood entry
  steps:
  - intent: track_mood
  - action: utter_going_to_mood_tracker
  - intent: add_mood
  - action: mood_entry_form
  - active_loop: mood_entry_form
  - slot_was_set:
    - mood: great
  - intent: inform
    entities:
    - trigger: family
  - action: mood_entry_form
  - active_loop: mood_entry_form
  - slot_was_set:
    - mood: great
  - active_loop: null
  - slot_was_set:
    - requested_slot: null
  - action: utter_ask_save_mood
  - intent: confirm_save_mood
  - action: action_save_mood
  - action: utter_ask_menu
  - action: action_reset_slots

  ## Story: User adds a mood entry
- story: User adds a mood entry
  steps:
  - intent: add_mood
  - action: mood_entry_form
  - active_loop: mood_entry_form
  - slot_was_set:
    - mood: great
  - intent: inform
    entities:
    - trigger: family
  - action: mood_entry_form
  - active_loop: mood_entry_form
  - slot_was_set:
    - mood: great
  - active_loop: null
  - slot_was_set:
    - requested_slot: null
  - action: utter_ask_save_mood
  - intent: confirm_save_mood
  - action: action_save_mood
  - action: utter_ask_menu
  - action: action_reset_slots

### Habit Tracker Stories

  ## Story: User adds a habit entry
- story: Go to habit tracker and add habit entry
  steps:
  - intent: track_habit
  - action: utter_going_to_habit_tracker
  - intent: add_habit_entry
  - action: habit_entry_form
  - active_loop: habit_entry_form
  - slot_was_set:
    - sleep: 5
  - intent: inform
    entities:
    - exercise: 10
  - action: habit_entry_form
  - active_loop: habit_entry_form
  - slot_was_set:
    - sleep: 5
  - active_loop: null
  - slot_was_set:
    - requested_slot: null
  - action: action_check_habits_to_track
  - intent: confirm_save_habit_entry
  - action: action_save_habit_entry
  - action: utter_ask_menu
  - action: action_reset_slots

- story: User adds a habit entry
  steps:
  - intent: add_habit_entry
  - action: habit_entry_form
  - active_loop: habit_entry_form
  - slot_was_set:
    - sleep: 5
  - intent: inform
    entities:
    - water_intake: 10
  - action: habit_entry_form
  - active_loop: habit_entry_form
  - slot_was_set:
    - water_intake: 10
  - active_loop: null
  - slot_was_set:
    - requested_slot: null
  - action: action_check_habits_to_track
  - intent: confirm_save_habit_entry
  - action: action_save_habit_entry
  - action: utter_ask_menu
  - action: action_reset_slots

  ## Story: User add habits to track
- story: Go to habit tracker and add habits
  steps:
  - intent: track_habit
  - action: utter_going_to_habit_tracker
  - intent: add_habit_entry
  - action: habit_entry_form
  - active_loop: habit_entry_form
  - slot_was_set:
    - sleep: 5
  - active_loop: null
  - slot_was_set:
    - requested_slot: null
  - action: action_check_habits_to_track
  - intent: add_habits_to_track
  - action: action_suggest_habits_to_track
  - intent: save_habits_to_tracker
  - action: action_save_habits_to_track
  # - action: utter_would_you_like_to_add_habit_entry

- story: User add habits to track
  steps:
  - intent: add_habits_to_track
  - action: action_suggest_habits_to_track
  - intent: save_habits_to_tracker
```

```
  - action: action_save_habits_to_track
  # - action: utter_would_you_like_to_add_habit_entry

  ## Story: User edit habits to track
- story: Go to habit tracker and edit habits
  steps:
  - intent: track_habit
  - action: utter_going_to_habit_tracker
  - intent: edit_habits_to_tracker
  - action: action_suggest_habits_to_track
  - intent: save_habits_to_tracker
  - action: action_save_habits_to_track
  # - action: utter_would_you_like_to_add_habit_entry

- story: User edit habits to track
  steps:
  - intent: edit_habits_to_tracker
  - action: action_suggest_habits_to_track
  - intent: save_habits_to_tracker
  - action: action_save_habits_to_track
  # - action: utter_would_you_like_to_add_habit_entry
```

XI. Acknowledgement

Hindi ko akalaing mararating ko ang puntong ito. Bagamat hindi ako pumasa sa UPCAT, magtatapos pa rin ako sa UP. At siyempre, hindi ito magiging posible kung hindi sa mga taong tumulong at nagbigay ng kanilang suporta.

Sa aking adviser na si Ma'am Carpio, maraming salamat po sa iyong mahabang pasensya at sa paghikayat sa akin upang matapos ang SP na ito. Sa mga guidance counselors ng UPM Guidance and Counseling Program, maraming salamat sa pagbahagi ng inyong mga opinyon, mungkahi at komento, lalo na kay Ma'am Micah na nag-review ng nilalaman ng app na ito. Hindi ko po kayo makakalimutan sa inyong kabutihan.

Sa aking mga kaibigan, maraming salamat sa inyo, THC, dahil kahit hindi tayo magkasama sa kolehiyo, nandiyan pa rin kayo. Sa mga naging kaibigan ko sa UPM, lalo na sa OM, maraming salamat sa pagsama kahit sa isa't kalahating semestre lang na may face-to-face, one chat away pa rin kayo noong online class. Kung hindi rin dahil sa inyong tulong, hindi ko makakayanan ang online class.

Sa aking pamilya, maraming salamat sa inyong suporta, lalo na sa aspeto ng pinansyal. At sa ating Panginoon, maraming salamat dahil hindi Mo ako pinabayaan at inakay Mo ako patungo sa mas magandang landas.

Bagamat mahirap ang buhay sa kolehiyo, mahalaga na hindi natin kalimutan ang pangangalaga sa ating sarili lalo na sa ating mental health. Bilang pagpapahalaga sa mga mag-aaral sa kolehiyo, partikular na sa mga nagdaranas ng mga kondisyon sa kalusugan ng isipan, iniaalay ko ang pag-aaral na ito.