Princess Nourah Bint Abdulrahman University

SoulGlow

Graduation project (2)

Mental Health Journaling System (SoulGlow)

Presented by:

ID	Name
442002956	Reeman Algarni
442004138	Reem Alzahrani
442004084	Ruba Alghamdi
442001079	Lamia Alasmari

Section: 8CE

Supervised by: Dr. Kashish Ara Shakil

Graduation Project Report Submitted to College of Computer and Information Sciences at PNU in Partial Fulfillment of the Requirements for the
Degree of: Bachelor of Computer Science
CCIS, PNU Riyadh, KSA 2024-1445H

Table of Contents

TABLE OF FIGURES

TABLE OF TABLES

Acknowledgments

We would like to express our deepest gratitude and appreciation to all those who contributed to the successful completion of our graduation project.

First and foremost, we are thankful to our supervisor Dr. Kashish Ara Shakil, for her guidance, support, and invaluable insights throughout the entire project. Her expertise and commitment played a great role in shaping and refining our work.

We would also like to extend our appreciation to our university which provided us with a great environment to thrive and an incredible professor, whose feedback and constructive criticism improved the quality of our project.

This project has been a significant learning experience, and we are proud to have had the opportunity to work with such a talented and supportive community. Thank you all for being part of this journey and for contributing to the culmination of our academic endeavors.

Abstract

In this fast-paced digital age, where mental health is a growing concern affecting many individuals, our project aims to enhance individuals' well-being through the development of an innovative web platform. The proposed platform aims to integrate therapeutic daily writing practices with modern technologies such as natural language processing (NLP) and emotion analysis.

The "Soul Glow" platform is a safe space for users to journal their thoughts, emotions, and experiences through daily reflective writing. It utilizes natural language processing to record user emotions and generate their mental state, providing insights into their emotional well-being over time. This website offers a confidential outlet for reflective journaling, allowing users to express and understand their thoughts and feelings.

Emotion analysis algorithms provide visualizations of emotional well-being. The platform aims to empower individuals through self-awareness and consistent journaling. It has an easy-to-use interface, privacy protection, and mental health resources. Challenges are offered for continuous progress. The project aims to make a meaningful impact on mental health, fostering self-discovery and emotional resilience.

In summary, the "Soul Glow" platform aims to promote mental health and well-being by integrating therapeutic daily writing practices with emotion analysis and artificial intelligence. With its user-friendly interface and smart technologies, the platform will help you understand your emotional health and develop the necessary self-empowerment skills to achieve balance and emotional happiness in your life.

Keywords

Mental Health, Website, Journaling, Natural Language Processing (NLP), Emotion tracking, Self-reflection, Mental State detection, Digital Mental Health, machine learning.

Chapter 1:

1. Introduction

In ancient times, most diaries were handwritten using pens and paper. These diaries took the form of small books in which they were written regularly. In today's digital age, a remarkable shift has been made to the online world. People now document their thoughts, experiences, and emotions in digital journals, and various online platforms. This digital transition has not only made journaling more accessible and convenient but has also opened new possibilities for understanding the power of human emotions. Emotions are fundamental to human experience and have a significant impact on our mental health. Understanding and monitoring emotions can provide valuable insights into an individual's psychological well-being. Advances in technology have enabled the development of tools and applications that can capture and analyze emotional data. These technologies can range from wearable devices and smartphone apps to more complex AI-driven systems .

The connection between emotions and mental health is deep, with emotions often serving as indicators of an individual's overall mental well-being. Leveraging technology to monitor emotions can therefore be instrumental in promoting mental wellness and addressing mental health concerns. Emotion detection and overall mental health monitoring in online journals is an emerging field that combines technology and psychology to analyze and interpret emotional content. By harnessing natural language processing (NLP), machine learning, and data analytics, these systems can uncover the emotional and mental undercurrents of the content people create and share online. There are various methods employed in emotion detection and mental health assessment, including voice analysis, facial expression analysis, physiological measurements, and text and sentiment analysis.

Our system aims to utilize these methods to provide comprehensive insights into individuals' emotional and mental states. By discerning the emotional nuances embedded within the text, it becomes possible to gain deep insights into the writer's emotional struggles, moments of joy, or potential indicators of depression such as persistent feelings of sadness. This integration of technology with emotional and mental health monitoring signifies a promising avenue for enhancing mental healthcare. As technology continues to advance and data collection methods become more refined, the role of emotion detection in mental health is assured to expand further. Therefore, in this project, we aim to harness the emotional health of individuals as a means to detect and address their mental health status effectively. Our mental health journaling system, SoulGlow will primarily focus on emotion detection and depression detection based on the user journal entries enabling the user to track their mental health over time.

1.1 Problem statement and significance

In today's world, many people find themselves dealing with intense levels of depression, and sadness. The modern life, compounded by the global effect of the passed pandemic, have significantly impacted people's mental well-being, often leaving them feeling overwhelmed and isolated. Unfortunately, the shame associated with discussing mental health challenges and asking for help prevents people from getting the necessary support and guidance they require to get better. Ignoring this problem can extend the suffering of humans with mental health in silence, and the consequences will be unpleasant [2].

In this project we will be developing a mental health journalling system in the form of a web-based platform. Our website will be a vital solution to this universal problem by providing a safe and inclusive space where individuals can freely express their innermost thoughts and emotions without the fear of judgment. By creating an environment of understanding, our platform encourages users to journal about their experiences openly, with access to resources to help them improve mentally.

Our system will analyze their journal entries to help users explore the depths of their emotions and gain valuable insights into the root causes and triggers behind their feelings.

Through a combination of user-driven journaling, analysis of emotions, depression detection, and a big repository of mental health resources, our website encourages to give attention to mental health. We aim to inspire a culture of openness and acceptance, where individuals feel empowered to prioritize their mental well-being and ask for the necessary help and resources they need to thrive and glow.

1.2 Proposed solution

Our goal is to develop a solution that provides a private space for users to freely express their thoughts and emotions without the fear of being judged.

This solution involves creating a welcoming and user-friendly website that offers a secure environment for users.

By allowing users to write down their feelings daily, they can better understand the reasons behind their emotions, encouraging reflection and emotional discharge [3]. The website will provide daily affirmations and challenges for users, along with various well-being books to keep them motivated and engaged. This approach aims to support users in managing their mental health effectively.

Additionally, this project includes emotion detection through sentiment analysis of journal entries, categorizing feelings as Joy, sadness, anger, fear, surprise and love. Users can track their emotional progress over time. The process of analyzing the journal entries is done by detecting the user emotions from text. The system will also analyze the journals for depression and display the results, allowing users to visualize their emotions and depression rate through graphs, enhancing their understanding of mental well-being.

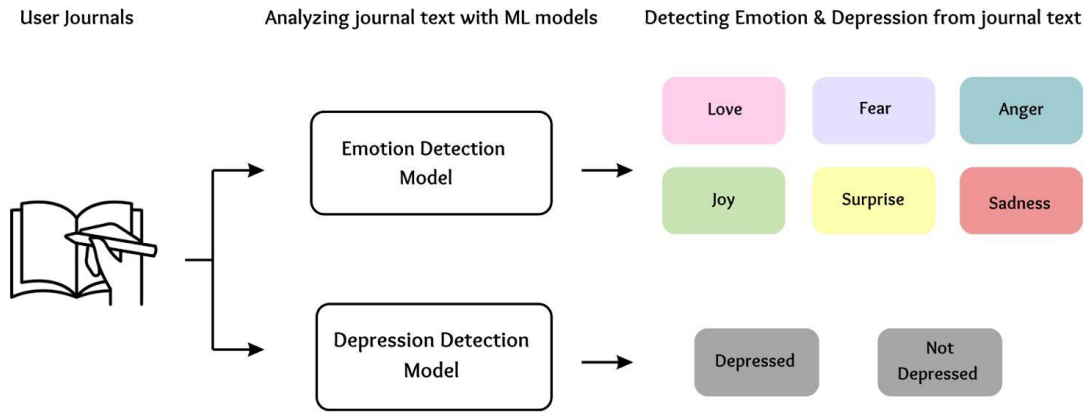


Figure 1: proposed emotions and depression detection system

Development Environment:

Utilize both Visual Studio (VS) for robust Python development and Jupyter Notebook for fine-tuning the model and data analysis and visualization.

Data Collection:

1. **Emotion Dataset**

The dataset contains text data labeled with six basic emotions: anger, fear, joy, love, sadness, and surprise [4].

- Size: The dataset comprises 34792 rows.

2. **Depression Dataset:**

the dataset includes texts related to depression, categorized as "0" for non-depressive texts and "1" for depressive texts [37].

- Size: The dataset contains 27977 rows.

Data Preprocessing:

Data preprocessing was done using the following steps:

Cleaning: Apply standard techniques like lowercasing, removing special characters, URLs, mentions/hashtags, emoji's and tokenizing the text for both datasets.

Normalization: Consider stemming, lemmatization to reduce word variations and improve model efficiency.

Balancing: Address class imbalances if certain emotions are overrepresented in the data. This ensures fair model training, but in our case, we used a dataset that was already balanced.

Feature Extraction:

Feature extraction is an important process in natural language processing (NLP) that converts preprocessed text into numerical features suitable for machine learning algorithms.

Term Frequency-Inverse Document Frequency (TF-IDF): is a commonly used technique for this purpose, enhancing Bag of Words (BoW) by considering the importance of words across the entire dataset. TF-IDF assigns higher weights to unique words in a document while reducing the weight of common words across documents, making it effective for text classification tasks like sentiment analysis.

Additionally, feature extraction can be implicitly performed by pre-trained models, including Transformer-based models. These models tokenize input text, encode tokens into high-dimensional vectors, and pass them through layers to capture contextual information. The final hidden states represent learned features that can be used for downstream tasks like sentiment analysis, offering an efficient approach to feature extraction without explicit engineering

Model Development:

1. Emotion Detection:

Emotion detection can benefit from leveraging powerful pre-trained language models such as DeBERTa, RoBERTa, Distilroberta, BERT, and XLNet. These models, pre-trained on extensive text corpora, offer robust representations that can be fine-tuned for specific tasks like emotion detection. Some of the models used in this work includes:

1. DeBERTa (Decoding-enhanced BERT): improves upon BERT with enhanced decoding mechanisms, achieving state-of-the-art results in various NLP tasks.

2. RoBERTa (Robustly optimized BERT approach): an enhanced version of BERT, achieves superior performance by optimizing training procedures and using a larger corpus.

3. BERT (Bidirectional Encoder Representations from Transformers): a pioneering model in NLP, utilizes transformer architecture and pre-training on large corpora for various downstream tasks.

4. Distilroberta: it's a distilled version of RoBERTa, offering a balance between efficiency and performance. Its streamlined architecture makes it suitable for resource-constrained environments without sacrificing accuracy in emotion detection tasks.

5. XLNet: leveraging autoregressive pre-training and permutation language modeling, attains state-of-the-art performance across NLP benchmarks.

2. Depression Detection:

Following machine learning models have been explored in this project like:

1. Logistic Regression: is a statistical model used for binary classification tasks, estimating the probability that an instance belongs to a particular class.

2. Naive Bayes: is a probabilistic classifier based on Bayes' theorem, assuming independence between features, often used for text classification tasks.

3. Decision Tree: It partitions the feature space into regions and make predictions based on majority class or average target value within each region, suitable for classification and regression tasks.

4. Random Forest: an ensemble learning method that constructs multiple decision trees during training, outputting the mode of the classes or mean prediction of the individual trees.

5. Support Vector Machine (SVM): finds the hyperplane that best separates classes in the feature space, maximizing the margin between classes, commonly used for classification and regression tasks.

Model Training:

Train separate models using labeled data for both emotion detection and depression detection tasks.

Model Evaluation:

1. Use relevant metrics like accuracy, loss, precision, recall, F1-score and ROC curves for both emotion and depression detection.
2. Employ cross-validation techniques to ensure model generalizability to unseen data.
3. Analyze model outputs and interpretability to understand factors influencing predictions.

1.3. Domain and limitations

1.3.1. Domain

The domain of the project is the intersection of mental wellness and technology, focusing on emotional well-being and mental health support. The project utilizes an online journaling system that incorporates Natural Language Processing (NLP) technology. This technology enables a deeper understanding of user-written content by analyzing the emotions behind the words. The system also includes emotion detection capabilities to identify signs of depression through the analysis of emotional tone and context. The goal is to provide a supportive environment where users can freely express their thoughts and receive understanding and assistance related to mental health concerns.

1.3.2. Limitations

The SoulGlow journaling system has several limitations that should be taken into consideration:

1. Language Limitation: The system is currently available only in English, which may hinder individuals who speak other languages from fully benefiting from the platform.

2. Text-based Analysis: The system can analyze text-based content only and does not have the ability to understand emotions conveyed through images or other non-textual forms.

3. Age Restriction: The system is designed for individuals older than 13 years of age due to legal and privacy considerations. This age restriction may limit access for younger individuals who could benefit from mental health support.

4. Lack of Mobile Application: The system is accessible solely through a website and does not have a dedicated mobile application. This may inconvenience users who prefer or rely on mobile apps for accessing such services.

5. Limited Mental Illness Coverage: The system is primarily designed to detect and provide insights into specific mental health conditions such as depression.

1.4 Definition of new terms

SoulGlow: the name of the proposed system, that captures our purpose, which is to help users connect with their inner selves and shine a light on their emotional well-being. The word "Soul" signifies the core of one's being, encompassing emotions, thoughts, and personal experiences. "Glow" represents the idea of radiance, positivity, and growth.

Sentiment analysis: Figuring out the overall feeling in a piece of writing. In our project, it helps us understand the emotional tone of your journal entries.

NLP: a field of artificial intelligence that focuses on enabling computers to understand, interpret, and generate human language, allowing for effective communication between humans and machines.

Emotion detection: Identifying and understanding feelings expressed in words. In our project, this means our system can understand the emotions you write about.

Online Journal: a digital platform where individuals can record and organize their thoughts, experiences, and emotions.

Mental health resource: Various tools like podcasts, books, and challenges aimed at making you feel better emotionally.

Chapter 2:
2. Background Information & Related work

2.1. Background Information.

This chapter describes the background information about the proposed SoulGlow system.

1.What is Depression detection?

Depression detection involves various tools and techniques to identify signs and symptoms of depression in individuals. These tools can range from standardized clinical assessments to innovative technological solutions like machine learning algorithms and natural language processing. One of the significant advancements in this field is the integration of artificial intelligence, which enables the analysis of vast amounts of textual data, including social media posts, online forums, or electronic health records. By leveraging machine learning models, mental health professionals can detect patterns in language and emotions that might indicate the presence of depression. For instance, sentiment analysis plays a vital role in this context, as it can reveal subtle shifts in emotions expressed in texts, aiding in the early detection of depressive symptoms. Additionally, linguistic analysis techniques, such as examining linguistic markers associated with depression (e.g., pronoun usage, sentence structure), can provide further insights into an individual's mental state. Moreover, topic modeling algorithms can help identify prevalent themes or topics in textual data related to depression, allowing for a deeper understanding of the issues individuals are facing. These approaches, including sentiment analysis, linguistic analysis, and topic modeling, contribute to a more comprehensive understanding of an individual's mental health status and facilitate early intervention and support for those at risk of or experiencing depression. [7].

2.Sentiment analysis

Sentiment Analysis, also known as opinion mining, is a vital component of natural language processing (NLP) used to determine and extract the emotional tone behind a piece of text. It involves analyzing text data to identify the sentiment expressed. While sentiment analysis is primarily concerned with a text's polarity (positive, negative, or neutral), it also goes beyond polarity to identify feelings and emotions such as anger, happiness or sadness. This analysis is particularly valuable in the context of social media posts, customer reviews, surveys, and other forms of textual data, helping researchers understand public opinions, attitudes, and emotions. Sentiment Analysis algorithms use natural language processing and machine learning techniques to recognize and categorize sentiments, allowing organizations to gain valuable insights from large volumes of unstructured text data. It has also shown improvements in the medical field such as psychology, it has emerged as a powerful tool in the realm of mental health, offering invaluable insights into patients' emotional states and well-being. By analyzing sentiments expressed in online forums, social media posts, or even electronic health records, mental health professionals can gain a deeper understanding of individuals' feelings, anxieties, and struggles. Sentiment Analysis can be divided into several categories based on the scope and focus of the analysis. One primary division is document-level sentiment analysis, which assesses the overall sentiment expressed in an entire document, such as an article, review, or social media post. Another category is sentence-level sentiment analysis, where the sentiment of individual sentences within a document is evaluated. More granularly, aspect-based sentiment analysis delves into specific aspects or features mentioned in a text (like product features in a review) and evaluates sentiment associated with each aspect [8].

3.Emotion Detection

Emotion detection is the process of identifying and categorizing human emotions expressed in various forms of data, such as text, speech, facial expressions, or physiological signals. This technology leverages techniques from natural language processing, machine learning, and computer vision to analyze and interpret emotional cues accurately. In text-based emotion detection, algorithms analyze written text to identify the underlying emotions conveyed by the author, such as joy, sadness, anger, or fear. These algorithms examine linguistic features, semantic context, and syntactic patterns to infer emotional states from text data. By analyzing words, phrases, and sentence structures, text-based emotion detection algorithms can discern the emotional tone of a piece of text, whether it's a social media post, customer review, or email message.

Research studies have demonstrated the effectiveness of text-based emotion detection in various domains. Emotion detection in textual conversations aids sales and marketing by analyzing customer sentiments in interactions, allowing businesses to tailor their strategies and offerings accordingly. In psychology, it provides insights into emotional states and behaviors, facilitating research and therapeutic interventions. Additionally, it enhances customer engagement and satisfaction by enabling personalized communication and support [9].

Moreover, a study have shown that text-based emotion detection has significant implications for mental health monitoring and intervention [10]. by analyzing linguistic features in text data can help detect early signs of depression and anxiety in individuals, facilitating timely interventions by mental health professionals. Additionally, research exploring how politeness impacts the user experience of chatbots for mental health support has shed light on the importance of emotion-aware conversational agents in providing empathetic and effective support to users [11].

Text-based emotion detection is a powerful technology with diverse applications across various domains, including social media analysis, market research, mental health monitoring, and human-computer interaction. Through advanced machine learning techniques and linguistic analysis, it enables organizations to gain valuable insights into human sentiment, behavior, and emotional well-being, ultimately enhancing decision-making processes, user experiences, and mental health outcomes.

4.The process of Sentiments Analysis

Sentiment analysis is a technique which belongs to the broader category of text analysis, the process of sorting and understanding textual data. The goal is to interpret and classify the emotions or sentiments conveyed within textual data. Emotion and depression detection model is a type of sentiment analysis that relies on Machine Learning and Natural Language Processing to make analysis from textual data.

Analytics focuses on developing new insights and understanding of business performance based on data and statistical methods by applying six phases called Data analytics lifecycle. These phases include the following:

Data Discovery:

In the first phase of data analytics, the goal is to define the problem, understand the context, and set clear objectives. Key tasks include problem formulation, goal setting, and data source identification [10].

Data Preparation:

In the second phase, the Data analysis involves collecting, cleaning, and transforming data to make it suitable for analysis. It includes data cleaning, integration, feature engineering, and transformation [10].

Model design:

In the third phase, you select the appropriate algorithms and methodologies. Tasks include algorithm selection, model architecture design, and feature selection [10].

Model Building:

In the fourth phase, the emphasis is on implementing and training the selected models. Key steps include model implementation, training, validation, and fine-tuning [10].

Communicate Results:

In the fifth phase, prepare presentation for sponsors and analyst and will share the project result with various audiences to draw inferences on the key findings and summarize the entire work done [10].

Operationalize:

The final report includes monitoring its performance in real-time, establishing feedback loops for ongoing enhancement, ensuring data and model security, and meticulously documenting the entire process for future reference [10].

5.What is natural language processing (NLP)?

Natural Language Processing (NLP) is a significant field in computer science and linguistics dedicated to understanding and processing human language using computers. The division of texts into basic units, known as "tokens," is a fundamental step in NLP. These units include words, phrases, and other linguistic concepts.

"tokens" refer to linguistic units that can be processed as the smallest units in text analysis and generation. NLP deals with single-word tokens like words, but it also considers compound tokens such as idioms and ready-made expressions. This concept is used to better understand human language and enable computers to interact with it more effectively.

Additionally, the concept of tokens highlights the importance of interpreting complex linguistic expressions as indivisible units in certain contexts. These complex linguistic units, known as compound tokens, play a critical role in text analysis and their effective use in machine translation and other linguistic processes [11].

.

6.Text-based Emotion detection and Depression detection and sentiment analysis in NLP

Natural Language Processing (NLP) in texts is a significant area within Emotion detection, aiming to analyze texts to understand and extract the emotions and sentiments they convey, Emotion detection involves using NLP techniques to extract words and phrases related to emotions like happiness, sadness, anger, and fear.

Emotion and depression detection techniques share similarities, especially when it comes to Natural Language Processing (NLP) techniques. Both fields utilize NLP methods to analyze text and extract relevant information. Some of these techniques:

Tokenization in NLP:

Tokenization is a common preprocessing step in NLP, where text or protein sequences are split into distinct tokens, which are atomic units of information. Tokenization can be done at various levels, such as character-level, word-level, or subword-level, depending on the application [12].

Bag-of-Words (BoW):

BoW representation is used to count unique tokens in a text and create a fixed-size vector for analysis. BoW is a simple and popular feature extraction method in text analysis. It can also be used to represent proteins by counting character-level tokens, which are amino acids in this case [12].

Word Embeddings:

The use of word embeddings in NLP can significantly improve the performance of various NLP applications such as sentiment analysis, language translation, natural language understanding, and text search. These algorithms like word2vec and FastText use dense vectors to represent tokens in a way that captures semantic information, allowing for more accurate and nuanced analysis of text data [12].

7.Machine learning techniques for emotion detection and depression detection

Machine learning (ML) is a technique that concerns building models that learn automatically by experience. The general idea of these approaches is to build a learning model using sample data, then make predictions, suggestions, or decisions without being given explicit instructions. ML approaches can be broadly divided into two main categories, Shallow machine learning and deep learning. Shallow ML approaches are also divided into three categories, Supervised, unsupervised, and semi-supervised learning. The key difference between these three categories is data labels; Supervised ML approaches learn from fully labelled data, example SVM, decision tree, Naïve Bayes. while unsupervised ML learns from unlabeled data example K-Mean, KNN, PCA. Semi-supervised ML is an intermediate approach that uses partially labeled data to learn the unlabeled portion of the data. ML approaches have been applied successfully in diverse fields, such as pattern recognition, finance, learning, as well as emotion detection & depression detection.

ML approaches show promising results in detecting users' emotions from several data sources, including text, speech, video, and more. Also, some ML approaches have shown high performance in processing and learning time, making them suitable for detecting emotions in interactive and adaptive systems in real- time. Such as SVM algorithm [13].

ML techniques offer a unique opportunity for emotion detection and depression detection through linguistic analysis, providing a non-invasive and scalable approach to mental health monitoring. By analyzing linguistic patterns in textual data, such as social media posts, online forums, or electronic health records, ML models can identify subtle indicators associated with various emotions and depressive symptoms. These indicators may include changes in language use, such as increased frequency of negative words, reduced linguistic complexity, shifts in tone and style, or alterations in sentiment expression.

Through sophisticated algorithms, ML models can recognize patterns and correlations within textual data that may signify underlying emotional states and mental health conditions like depression. By leveraging large datasets and advanced natural language processing techniques, these models can learn to distinguish between normal variations in language and those indicative of depressive states. Additionally, it can also learn to distinguish between different emotional states, such as joy, sadness, anger, or excitement. As a result, ML-based depression emotion detection and detection systems can provide valuable insights into individuals' emotional and mental well-being, enabling early intervention and support.

8.Deep Learning models for emotion detection & depression detection

In emotion and depression detection using Deep Learning, neural networks are trained on labeled datasets to understand sentence semantics and structures. Deep Learning permits the system to comprehend the semantics and building of sentences the interdependency of the sentence. The emotion dataset & depression dataset is first built, which is tagged. This tagged dataset is then fed to the neural network which trains the dataset for more accurateness and handles new data. There are different options for selecting training models, like Recurrent Neural Network and Convolution Neural Network. Afterward training the neural network, analytic reports are produced until the desired accuracy is not attained. Before employing the algorithms on the input, pre-processing on the text is completed. This conversion on the raw input into another format is easy and efficient for processing. There are different approaches for pre-processing data like Cleaning in which it deals with stop words, punctuation, capitalization, repeated letters, etc. Annotation in which the tokens are markup as part of speech, Standardization in which the input is prearranged for effective access, and extracting the valuable features is important for a specific task or application [14].

Convolution neural network (CNN) is a feed-forward neural network, mainly consisting of an input layer, a convolution layer, a pooling layer, a full connection layer, and an output layer.

Recurrent neural network (RNN) is a feed-forward neural network with a ring structure and a specific memory function. Its input includes current input samples as well as information obtained in the previous time so that information can be cycled in the network at any time.

Long short-term memory network (LSTM) is a kind of RNN that can learn the relationship of long-term dependency, characterize the information of a time sequence, and effectively solve the problem of gradient vanishing or gradient exploding faced during RNN training. Each unit consists of four components: a memory cell, input gate, output gate, and forget gate [15].

Bi-LSTM is an improved LSTM model that uses two LSTM networks to train together and start their respective sequences from opposite ends while being connected back to the same output layer, it can integrate the past and future information of each point.

Support Vector Machine (SVM) is a versatile algorithm for classification and regression tasks, aiming to find the best hyperplane to separate different classes in high-dimensional spaces. It effectively handles linear and non-linear problems using various kernel functions, making it widely applicable in text classification, image recognition, and bioinformatics.

2.2 Related work survey

2.2.1 Similar Studies

The surveyed research papers highlight various applications of natural language processing (NLP) and machine learning (ML) in mental health monitoring and support. Techniques like sentiment analysis, text classification, and thematic analysis are utilized to extract insights from social media data, including platforms like Reddit, Twitter, and clinical texts. These methods enable the identification of mental health-related content, such as depression or anxiety indicators, and facilitate early detection and intervention. Additionally, studies explore the efficacy of mental health apps, incorporating user reviews and sentiment analysis to evaluate their effectiveness and address user needs. Such approaches offer promising avenues for integrating into a Journaling system, enhancing its capacity to monitor users' mental health states, provide personalized support, and contribute to reducing stigma while improving overall mental well-being.

Study 1: A deep learning model for detecting mental illness from user content on social media. (Date: 2020)

This study collected posts from mental health communities on Reddit. By analyzing and learning posting information written by users, our proposed model could accurately identify whether a user's post belongs to one of six mental-health-related subreddits which is: r/depression, r/Anxiety, r/bipolar, r/BPD, r/ schizophrenia, and r/autism. this will be a supplementary tool for monitoring mental health states of individuals who frequently use social media, the study focused on text processing and machine learning to classify users' text into relevant mental health subreddits, using SMOTE to handle data [16].

Study 2: Detecting presence of mental illness using NLP sentiment analysis. (Date: 2022)

Using NLP and machine learning algorithms sentiment analysis which is also known as opinion mining helps automatically identify the emotional tone behind the online conversation. A rule-based system performs sentiment analysis based on set of rules crafted manually an automatic system relies on machine learning techniques to learn from data how to analyze the emotions, this will help people who are suffering from mental illness and thus help in correct treatment. Data is collected from social media platforms, including user details and their posted content. This information is processed and organized into datasets. Posts are categorized as positive, negative, or neutral. Positive content suggests good mental health, while negative content may indicate mental illness. Healthcare professionals can identify users with negative posts for early intervention and support [17].

Study 3: Natural language processing in mental health applications using non-clinical texts. (Date: 2017)

People use writing to express their activities, and convey their feelings, mental states, hopes and desires. Recipients then use this written information from emails and other forms of social media texts to make inferences, Natural language processing (NLP) techniques make inferences about what people say and feel, and these inferences can trigger messages or other actions. This paper aims to help researchers in these areas envision and work toward new mental health applications. the different data sources (ranging from lengthy diaries to brief tweets), and existing datasets that have been gathered and annotated (with moods, suicidal intent, etc..) using NLP are discussed. the techniques that researchers have applied to make inferences or generate diagnoses about the emotional state or mental health of the authors of these texts. This work draws widely upon text classification and NLP [18].

Study 4: Using Machine Learning and Thematic Analysis Methods to Evaluate Mental Health Apps Based on User Reviews. (Date: 2020)

Over the years, research has applied sentiment analysis on mobile health (mHealth) app reviews and social media data to understand user opinions or perceptions. Greaves et al. applied the machine learning (ML) technique to classify healthcare-related online comments into negative and positive sentiments to understand patients' perception about their care in meeting users' expectations and needs. In this paper, we utilized both machine learning-based sentiment analysis and thematic analysis to identify negative and positive factors affecting the effectiveness of mental health apps. In addition, we offer design recommendations based on the positive factors to address the negative issues, and we evaluate 104 mental health apps on Google Play and App Store by performing sentiment analysis of 88125 user reviews using machine learning approach [20].

Study 5: Mental Health Monitoring using Sentiment Analysis. (Date: 2020)

Mental State of a person is the perspective of that person and furthermore gives a sign of his/her general nature. The assessment of mental well-being is important to comprehend and propose treatments for patients with stray mental conduct. Sentimental Analysis using Natural Language Processing has been proposed to be applied to the collected data to predict user information such as location, mood, activity, mental status, depression, anxiety, stress and so on. This paper surveys and proposes a methodology for data extraction and a model using Lambda Architecture to study the social media presence and other available data to predict the mental state of the user along with taking additional measures to maintain the secrecy of the user along with data privacy. Estimation and emotional detecting technology could assist with handling these goals by giving successful apparatuses and frameworks for target appraisal. Such devices and frameworks don't expect to supplant the clinician or therapist, yet they could bolster their choices [21].

Study 6: Machine learning-based proactive social-sensor service for mental health monitoring using twitter data. (Date: 2022)

The social media platforms are considered an ecosystem of social sensors where each social media platform user is treated as a social sensor cloud. Focusing on the Twitter platform, a generic framework is designed in this paper to support proactive mental health monitoring Detailed data cleaning and preprocessing of the tweets are offered using regular expressions based on observed patterns to ensure accurate results in sentiment analysis. It is challenging to address the large-scale mental health surveillance using traditional health administration systems to address these challenges, we are exploring innovative solutions using machine learning algorithms on social media platforms for early detection of mental illness. Social media platforms such as Facebook, Instagram, and Twitter are being used daily by millions of worldwide users to communicate and share information [22].

Study 7: A review on sentiment analysis from social media platforms. (Date: 2023)

Sentiment analysis, a crucial aspect of the modern era, relies on textual data from various sources like social media, blogs, and interviews. Its main purpose is to understand and assess emotions, including positive and negative sentiments, expressed in these texts. This tool has diverse applications, from detecting cyberbullying and identifying suicide ideation in social media data to analyzing sentiment changes in product reviews over time. However, it's essential to note that different sentiment analysis tools may yield varying results due to differences in techniques and vocabulary. Furthermore, this analysis highlights the prevalence of patents related to sentiment analysis in social networks, with major companies like Google, Facebook, and Microsoft dominating this field. Sentiment analysis has primarily been applied in domains such as marketing and politics, driven by factors like data accessibility and interdisciplinary collaboration. In contrast, domains like health and emergencies may face data protection challenges [24].

Study 8: Utilizing Neural Networks and Linguistic Metadata for Early Detection of Depression Indications in Text Sequences. (Date: 2018)

The use of neural networks and linguistic data for early detection of depression indications in text sequences. It begins with a review of relevant research, emphasizing the relationship between depression and language and how language reflects psychological states. Studies show distinct linguistic patterns for individuals with depression, such as increased use of first-person singular pronouns and negative emotion words. also addresses the ethical dimensions in the field of Natural Language Processing (NLP), shedding light on concerns regarding privacy, data protection, and bias in training data. Furthermore, it explores the use of natural language in monitoring mental health on social media and delves into the history of natural language processing and text classification. It also discusses the evolution of sentiment

analysis and the utilization of social media data in depression detection [25].

Study 9: Mental health monitoring apps for depression and anxiety in children and young people. (Date: 2022)

Depression in children and young people (CYP) begins with an introduction to the prevalence of depression and anxiety and the emergence of digital health interventions, particularly smartphone apps. These apps often include mood monitoring, which is a key part of cognitive-behavioral therapy (CBT). However, the research on the efficacy and user acceptance of these apps remains limited. Highlighting results from studies and reviews, some studies demonstrate potential benefits in reducing anxiety and depression symptoms, reflecting the growing interest in mental health apps. The perspective on mental health apps emphasizes viewing them as part of a larger mental health care system rather than standalone solutions. It underscores the need to explore the long-term consequences of these technologies, especially as passive monitoring becomes more prevalent. Overall, it encourages a holistic understanding of mental health apps and their societal implications within complex assemblages of factors [26].

Study 10: Detecting Arabic Depressed Users from Twitter Data. (Date: 2022)

This study focuses on depression, a severe global health issue that affects millions of people, especially in Arab countries where mental illness often goes undiagnosed and untreated due to stigma. Unlike previous studies that used English data, this research aims to detect depression among Arabic-speaking users on social media. Social media posts offer a more timely and accurate way to identify depression compared to traditional methods. The study crawled a dataset of Arabic tweets, classifying users as depressed or non-depressed based on their self-declared status and clinical scales. It examines depression-related behaviors in Arabic users to develop features that can classify tweets and improve early identification and intervention, reduce stigma, and enhance outcomes for individuals with depression [28].

Study 11: Detecting social media users' mental health issues with sentiment analysis. (Date: 2020)

This study explores the role of social media, particularly Twitter, as a platform for individuals with mental health disorders to express themselves. It addresses the challenges posed by cultural stigmas related to mental health, which often hinder individuals from openly discussing their conditions. The research collected 5537 tweets in Indonesian containing keywords associated with mental health disorders, such as "emotions," "hallucinations," "panic," "mental illness," "stress," and "fear." These tweets were analyzed using sentiment analysis and categorized as Positive, Negative, or Neutral. The study found that Twitter is effective in identifying symptoms of mental health disorders, offering a safe and comfortable space for individuals to express their emotions. The research aims to contribute to early identification, intervention, and reducing the stigma surrounding mental illness [29].

Study 12: Natural language processing to extract symptoms of severe mental illness from clinical text. (Date: 2020)

The Clinical Record Interactive Search Comprehensive Data Extraction (CRIS-CODE) project employs natural language processing (NLP) to capture symptoms of severe mental illness (SMI) from clinical text, aiming to enhance the utility of mental healthcare data for research. This effort involves developing information extraction applications to identify SMI symptoms within routine mental health records using the Clinical Record Interactive Search (CRIS) data resource. While NLP and Information Extraction (IE) have been applied to process clinical records, their use in mental healthcare data is limited, especially for symptom identification. The project focuses on creating sentence classification models for various SMI symptoms to facilitate automatic extraction from patient narratives. Initial results show the extraction of 46 symptoms from discharge summaries, cutting across diagnoses and revealing that symptoms often overlap between different patient groups. However, some challenges persist, such as issues with symptom differentiation due to common clinical language and variations in record types and completeness. This work opens possibilities for using NLP to extract valuable information on SMI symptoms from electronic health records, potentially improving research and clinical applications in the field of mental health [30].

2.2.2 Similar platforms

The surveyed AI-powered mental health support systems, like Journaly, Better.me, Menty, and MoodJournal, employ NLP and ML for emotion analysis and personalized recommendations. They assess users' journal entries, offer insights into emotional trends, and provide support, including suicide prevention measures. Menty gamifies mental health activities and triggers support notifications based on sentiment analysis. MoodJournal emphasizes data sharing with healthcare providers. Integrating similar features into our AI-powered Journaling website can enhance user engagement, promote emotional awareness, and provide proactive support for mental well-being.

Platform 1: better.me

It’s an AI journaling tool that helps you analyze your emotions and provides you with smart recommendations for your well-being. We used NLP emotion analytics to process text data and incorporated a suicidal prevention algorithm that will help you make better informed decisions about your mental health. Their mission is to provide a private environment to help people analyze their emotions and receive mental health support. With the help of natural language process, Better Me will classify the user's emotional situation and keep track of the data. To take a step forward towards suicide prevention, it also incorporates a suicidal text detection algorithm that triggers a preventive measure [19].

Platform 2: Menty

Menty is A Personalized Mental Health Platform. A lot of people want to do things to take care of their mental health but don't do it because it is hard to get into it at first. We realized that we could motivate people to do these things by making it fun with an interactive website where they can go on a mental health journey with their friends. A machine learning model running in a flask server analyzes the sentiment of the journal entry. The sentiment for the day's entry is reflected in the opacity of the calendar UI elements with darker being more positive and more transparent being more negative. If the sentiment is negative, the flask server makes SMTP (Simple Mail Transfer Protocol) API requests to send an email to all your friends on the app letting them know that they should check in on you. It Built with CSS, firebase, flask, JavaScript, python, react, and TensorFlow [23].

Platform 3: MoodJournal

The "MoodJournal" app emphasizes the importance of sharing data with healthcare providers. It also recognizes limitations in analysing app reviews, such as selection bias and undisclosed user conditions, while supporting all four stages of mood tracking and adapting to various user needs. Consumer reviews reveal users' motivations for using these apps, including understanding mood patterns, managing mental illnesses, and improving emotional well-being. Users appreciate the ability to log their mood frequently and share data with healthcare providers. They express a desire for personalized features and better data organization. Overall, the study underscores the potential benefits of using mood tracking apps like "MoodJournal" while calling for enhancements in their design and functionality [27].

Platform 4: Journaly

Journaly is an AI-powered app that enables users to enter their daily emotions and analyze them by using Sentiment analysis and Natural language processing (NLP). After the users input their thoughts, it will use machine learning to analyze the text and output a positive or negative value, words will be assigned to either –1 or 1, -1 which corresponds to strongly negative and 1 corresponding to strongly positive. Next, the entry is evaluated as a whole, considering these individual values collectively. Then they are added to a database so they can be analyzed. The app provides an average score among all entries, their lowest score and its corresponding post/date, their highest score and its corresponding post/date, and a chart showing the progression of the user's scores [31].

2.3 Proposed & Similar Platforms and Studies Comparison

Our System: Mental health journaling system (SoulGlow)

Mental health journaling system (SoulGlow)	
Problem solved	Our system offers a powerful solution to those who suffer from mental illnesses by offering an online journaling system, where patients can enter their daily emotions and express their feelings openly. The system uses NLP and sentiment analysis to analyze the patient's text and detect their emotions. The system outputs the user's mental state and helps them to improve it by providing recourses.
Domain & users	The system will benefit individuals who suffer mentally by providing a place to write their struggles. The system is accessible to anyone who's older than 12 years with knowledge of the English language.
Design method	The system uses a machine and deep learning algorithms that can be trained to recognize specific emotions in text such as Fear, Anger, Joy, Surprise, etc.
Software& Hardware	The system is developed using Python. It will be available on PC.
Output	The system provides emotion tracking using the journaling entries from the user, visualize it using coloured charts, generates users' depression states based on their expressed feelings and recommend helpful resources.
Features	This analysis covers linguistic features such as word frequency and pronoun usage, journal mood monitoring, detecting specific keywords related to depression.
Limitations	Only support English language

Table1: Mental health journaling system (SoulGlow)

Study 1 (A deep learning model for detecting mental illness from user content on social media)

A deep learning model for detecting mental illness from user content on social media.	
Problem solved	Detecting mental illness problems in early stages and providing appropriate solutions can help potential mental disorder sufferers. By collecting and analyzing data from mental-health-related subreddits in Reddit that focus on mental disorder issues, to identify the users with potential mental illness based on their posts [16].
Domain & users	Users: people that regularly post on Reddit.
Design method	To quantitatively represent each post, we converted the words in the training set to numerical representations using TF-IDF and CNN model uses pre-trained word embeddings generated through the word2vec API with a focus on capturing subreddit-specific language styles [16].
Software& Hardware	XGBoost and convolutional neural network (CNN) were employed [16].
Output	Develop a deep learning model to identify a user's mental state based on their posting information, identify whether a user's post belongs to a specific mental disorder [16].
Features	six binary classification models, each of which categorizes a user' specific post into one of the following subreddits: r/depression, r/Anxiety, r/bipolar, r/BPD, r/schizophrenia, and r/autism [16].
Limitations	Only support Reddit application [16].

Table2: Study 1: A deep learning model for detecting mental illness from user content on social media.

Study 2 (Detecting presence of mental illness using NLP sentiment analysis)

DETECTING PRESENCE OF MENTAL ILLNESS USING NLP SENTIMENT ANALYSIS	
Problem solved	detect depression among social media users. A person's psychological status can be determined by analyzing their tweets. Doctors can use this data to treat patients effectively and determine if someone is suicidal [17].
Domain & users	Users: people that regularly post on social media [17].
Design method	Data is gathered from social media platforms, classification to analyze the positive, negative, and neutral emotions. if the outcome is negative then the user is considered to have metal illness that needs help [17].
Software& Hardware	Python libraries for visualization, Access to APIs of twitter, machine learning algorithms to determine emotions [17].
Output	create a system which gathers data from the APIs of social media to measure the negative post and provides guidance for them so that early detection and prevention of mental health Issues can be performed on the user [17].
Features	Mental state detection using social media posts, histograms to visualize their mental health, and give treatment plans [17].
Limitations	Only support English language [17].

Table3: Study 2: Detecting presence of mental illness using NLP sentiment analysis

Study 3 (Natural language processing in mental health applications using non-clinical texts)

Natural language processing in mental health applications using non-clinical texts.	
Problem solved	The prevalence of mental health disorders implies that mental health is as crucial as physical health. Using NLP to detect mental health disorders based on sentiment analysis of data sources like social media, which can be applied by health care professionals to get accurate information and therefore give proper treatment [18].
Domain & users	look for emotions in general day-to-day sources such as Twitter.
Design method	Using the percentage of emotional terms in the posts. computed using LIWC to provide psychologically grounded lists of positive and negative emotional terms that will help to indicate mental state [18].
Software& Hardware	LIWC software tools. Hardware: large storage [18].
Output	Model uses data from social media services to learn about people's behavior, emotions, social communications skills, classify it to several mental illness to use it in e-therapy [18].
Features	diagnosis of specific mental health conditions and automatic labeling of suicidal thoughts [18].
Limitations	Only focus on depression and suicide. Only support English language [18].

Table4: Study 3: Natural language processing in mental health applications using non-clinical texts.

Study 4 (Using Machine Learning and Thematic Analysis Methods to Evaluate Mental Health Apps Based on User Reviews)

Using Machine Learning and Thematic Analysis Methods to Evaluate Mental Health Apps Based on User Reviews	
Problem solved	using machine learning and thematic analysis of positive and negative reviews to identify themes representing several factors affecting the effectiveness of mental health apps positively and negatively [20].
Domain & users	General people.
Design method	(Five Machine Learning Algorithms (SVM, MNB, SGD, LR, RF), natural language processing techniques, and DATA ANNOTATION) [20].
Software& Hardware	Mobile health apps, Social Media data, Deep learning model CNN, Heedzy tool, NVivo 12 Windows, and BOW approach [20].
Output	The evaluated mental health apps determine their positive and negative effectiveness based on user review [20].
Features	(Applied natural language processing techniques to preprocess the data and prepare it for analysis. prepared ground truth data by automatically annotating reviews based on user ratings. developed five supervised machine learning classifiers for predicting sentiment polarity (positive or negative) of reviews. used the best performing classifier to predict the sentiment polarity of unlabeled or unannotated reviews. conducted thematic analysis on positive and negative reviews using NVivo 12 (Plus Edition) qualitative analysis tool. utilized both machine learning-based sentiment analysis and thematic analysis to identify negative and positive factors affecting the effectiveness of mental health apps. In addition, they offer design recommendations based on the positive factors to address the negative issues) [20].
Limitations	excluded apps whose description does not relate to mental health, apps with less than five user reviews, and non-English apps [20].

Table 5: Study 4: Using Machine Learning and Thematic Analysis Methods to Evaluate Mental Health Apps Based on User Reviews

Study 5 (Mental Health Monitoring using Sentiment Analysis)

Mental Health Monitoring using Sentiment Analysis	
Problem solved	Using sentiment analysis and natural language processing to monitor mental health on the gathered data to predict user information such as location, mood, activity, mental status, depression, anxiety, stress, and other critical parameters [21].
Domain & users	General people
Design method	Natural Language processing Machine Learning. [21]
Software & Hardware	R programming language Lambda architecture. [21]
Output	The recognition of the text to be positive, negative, or neutral.
Features	data help to determine the sentiment analysis. The Target field has 3 outputs {0,2,4} where 0 represents neutral, 2 represents positive and 4 represents negative. data set contains only 3. fields of id, post and the date of the post based on these post's and using NLP inbuilt library of Nltk corpus, sentiment analysis is derived from this data. Feature Extraction and Selection Offline processing. [21]
Limitations	Analyze English texts only [21].

Table6: Study 5: Mental Health Monitoring using Sentiment Analysis

Study 6 (Machine learning-based proactive social-sensor service for mental health monitoring using twitter data)

Machine learning-based proactive social-sensor service for mental health monitoring using twitter data	
Problem solved	exploring innovative solutions using machine learning algorithms on social media platforms for early detection of mental illness using twitter data [22].
Domain & users	Twitter Platform General people
Design method	Machine Learning models Natural language processing
Software & Hardware	Twitter platform, LSTM, SVM, Python, Deep Learning packages, MongoDB being a NoSQL database, TensorFlow, and Windows 10 desktop with a 3.40 GHz Intel i7-6700 CPU and 16 GB RAM [22].
Output	Social-sensor cloud services from the Twitter platform.
Features	(Detect the likelihood of a person suffering from mental illness "as early as" possible using social sensor cloud services by using historical sentiment analysis and machine learning. user can be sarcastic or joke about an issue, or sometimes it could be a genuine health-related message but not relating to the user itself. Health experts are used to classifying each message with a Yes/No (binary classification) Polarity values closer to 1.0 indicate positive sentiment, closer to −1.0 indicate negative sentiment, while polarity values equal to 0.0 indicate neutral sentiment) [22].
Limitations	Focused on a few illnesses Alzheimer's, Asperger's, bipolar, dementia, depression, and schizophrenia [22]. Analyze English tweets only [22].

Table7: Study 6: Machine learning-based proactive social-sensor service for mental health monitoring using twitter data.

Study 7 (A review on sentiment analysis from social media platforms)

A review on sentiment analysis from social media platforms	
Problem solved	The program aims to identify signs of depression in text-based communication, extending support to individuals struggling with depression. Additionally, it provides tools for mental health professionals to assess and manage depression-related language effectively. By leveraging machine learning, lexicon-based methods, deep learning, and transformer-based models, the system contributes to the early detection and intervention of depression [24].
Domain & users	The domain of this system is mental health and psychology. Typically, users include researchers, mental health professionals, and and developers working on NLP and sentiment analysis [24].
Design method	The program employs a variety of design methods, including machine learning, lexicon-based approaches, deep learning, and transformer-based models. These methods utilize algorithms to analyze and classify text based on sentiment, extracting features and patterns indicative of positive, negative, or neutral sentiment [24].
Software & Hardware	Software tools integral to the program include Python libraries, TensorFlow (for deep learning experiments), and scikit-learn (for machine learning experiments). While the software is clearly outlined, the review lacks information on the hardware requirements [24].
Output	The system generates sentiment scores or labels assigned to texts, offering insights into positive, negative, or neutral sentiments expressed in the text. Additionally, it provides reports and summaries, aiding in the analysis of sentiment trends, changes over time, and the impact of specific events or topics on sentiment [24].
Features	Key features of the sentiment analysis system include temporal analysis, examining sentiment changes over time, and causality analysis, exploring relationships between sentiment and other variables. Evaluation metrics such as accuracy, precision, recall, F1 score, coverage, and interpretability are used to assess the system's performance [24].
Limitations	The challenges in depression detection through text-based communication include false positives/negatives, limited access for those not using text, ethical concerns about user data privacy [24].

Table 8: Study 7: A review on sentiment analysis from social media platforms

Study 8 (Utilizing Neural Networks and Linguistic Metadata for Early Detection of Depression Indications in Text Sequences)

Utilizing Neural Networks and Linguistic Metadata for Early Detection of Depression Indications in Text Sequences	
Problem solved	The program addresses challenges in early depression detection on Reddit, utilizing machine learning to classify users. It employs NLP techniques, such as text classification and sentiment analysis, to detect and monitor depression using social media data, exploring the relationship between depression and language [25].
Domain & users	Focused on mental health and NLP, the study involves Reddit users expressing depression signs and a control group. Target users include NLP researchers and those interested in language-depression links [25].
Design method	The system leverages a Convolutional Neural Network (CNN) that utilizes specialized word embeddings to analyze and categorize text [25].
Software & Hardware	The neural network models impact word embeddings, specifically using GloVe/fastText, Python, NLTK, and an Intel Xeon processor. Tools like LIWC, DLATK, and pre-trained word vectors are used for text analysis [25].
Output	The main goal is early detection of depression in users' text data. Results are evaluated with ERDE, F1 score, ERDE% displaying neural network effectiveness [25].
Features	Features involve pre-trained embeddings, metadata covering language aspects, and user-level information like emotions and sentiment [25].
Limitations	influenced by the quality and diversity of data available on social media, potential biases in training data, and the challenge of early detection [25].

Table9: Study 8: Utilizing Neural Networks and Linguistic Metadata for Early Detection of Depression Indications in Text Sequences

Study 9 (Mental health monitoring apps for depression and anxiety in children and young people: A scoping review and critical ecological analysis)

Mental health monitoring apps for depression and anxiety in children and young people: A scoping review and critical ecological analysis	
Problem solved	The app's primary objective is to help users monitor their mental health and seek appropriate support if needed. It does so by identifying and tracking symptoms of depression among individuals who speak a specific language [26].
Domain & users	The study explores mental health monitoring apps, with a specific focus on their application in the context of anxiety and depression. It examines their usage in clinical settings and their acceptability and usability among target users, including young patients, adolescents, and young people [26].
Design method	It reviews relevant studies using a scoping review methodology and charts the characteristics of the included studies. also touches upon the use of emojis and other predictive elements in monitoring users' moods [26].
Software & Hardware	It does not explicitly detail the specific software and hardware aspects of the mental health monitoring apps. However, it mentions the use of smartphones and sensors in some apps for data collection [26].
Output	Users receive a mental health assessment report, recommendations for seeking professional help, and access to mental health resources in their language [26].
Features	The app offers features such as mood tracking, access to therapeutic exercises or meditations, and a multilingual interface to accommodate users who speak the targeted language [26].
Limitations	It acknowledges limitations in the research, such as the possibility of missing research due to limited databases searched and the focus on English language studies, and the need for human intervention in severe cases [26].

Table 10: Study 9: Mental health monitoring apps for depression and anxiety in children and young people: A scoping review and critical ecological analysis

Study 10 (Detecting Arabic Depressed Users from Twitter Data)

Detecting Arabic Depressed Users from Twitter Data	
Problem solved	Identifying depression in individuals can be challenging, especially within the Arabic-speaking population, due to limited awareness of mental health issues. The study proposes a solution using Twitter as a valuable source for monitoring mental health. Researchers aim to develop a predictive model through machine learning algorithms, leveraging data from depressed and non-depressed users' tweets. The objective is to enhance early detection of depression by employing social media data and linguistic features to classify tweets as indicative of depression or not [28].
Domain & users	The system is utilized for individuals who have engaged with social media, particularly Twitter users [28].
Design method	The researchers conducted a supervised experiment using four popular classifiers: Random Forest, Naïve Bayes, AdaBoostM1, and Liblinear. They employed Weka, a machine learning tool, and the AffectiveTweets package, which provided filters for converting tweets into feature vectors. However, some modifications were needed to make this package compatible with the Arabic language [28].
Software & Hardware	Not mentioned [28].
Output	The study constructed a predictive model employing four classifiers (Random Forest, Naïve Bayes, AdaBoostM1, and Liblinear) trained on features like bag-of-unigrams and negation handling to distinguish depressed from non-depressed tweets. Performance evaluation, conducted through 10-fold cross-validation, used metrics such as accuracy, precision, recall, and F-measure. Among the classifiers, Liblinear achieved the highest accuracy (87.5%) and recall rate (87.5%), while AdaBoostM1 exhibited the lowest accuracy (55.2%) and a recall rate of 55.3% [28].
Features	The predictive model was built using four different classifiers: Random Forest, Naïve Bayes, AdaBoostM1, and Liblinear. These programs were taught to tell the difference between tweets that showed signs of depression and tweets that did not, using information we got from the tweets [28].
Limitations	It works within a limited range, specifically for individuals using Twitter. The system can only be applied to Arabic text [28].

Table 11: Study 10: Detecting Arabic Depressed Users from Twitter Data.

Study 11 (Detecting social media users' mental health issues with sentiment analysis)

Detecting social media users' mental health issues with sentiment analysis	
Problem solved	In this study, scientists developed a program that could predict if someone has mental health problems based on the words they use in their tweets. They did this by looking at specific words that are often linked to mental health issues. By doing this, they could learn more about people's behavior on Twitter and find out more about mental health [29].
Domain & users	The system is used on all those how have used social media, especially Twitter [29].
Design method	The methodology for this study involved collecting Twitter data totaling 10,000 tweets, which was subsequently reduced to 5,537 clean tweets after removing duplicates and irrelevant content. These tweets contained keywords related to mental health, such as "emotions," "hallucinations," "panic," "mental illness," "stress," and "fear." The next step involved manually labeling these tweets into three categories: positive (indicating a mental health disorder), negative (no indication of a mental health disorder), and neutral (unclear if there's a mental health disorder). Positive labels included tweets mentioning symptoms or actions related to mental health, while negative labels were assigned to tweets with no such mentions, even if they contained the keyword "mental illness." Additionally, tweets expressing positive emotions were labeled as negative for mental health disorders. Any remaining tweets were labeled as neutral [29].
Software & Hardware	RapidMiner for data collecting and cleaning [29].
Output	Individuals with mental disorders often express panic in their tweets, seeking attention, help, or simply sharing their experiences on social media. "Emotion," "stress," and "fear," suggesting that these emotions are common indicators of mental health issues after panic. Individuals with severe mental disorders may not actively communicate their condition on social media, due to the urgency of seeking professional help, such as psychologists and psychiatrists [29].
Features	API enables researchers to fetch a substantial volume of tweets, specifically 10,000 in this instance. Subsequently, RapidMiner is also employed for data cleaning, where it filters and eliminates duplicate Twitter data. The cleaning process involves identifying and removing tweets with blank text, "@" characters, "?", retweets (RT), and other redundant information, ensuring a refined dataset for further analysis [29].
Limitations	It operates within a constrained scope, only for Twitter users. The dataset consists of a small sample of 5,537 tweets, this limited dataset might not represent the full diversity of Twitter users and their mental health expressions [29].

Table 12: Study 11: Detecting social media users' mental health issues with sentiment

analysis.

Study 12 (Natural language processing to extract symptoms of severe mental illness from clinical text)

Natural language processing to extract symptoms of severe mental illness from clinical text	
Problem solved	The CRIS-CODE project utilizes Natural Language Processing (NLP) to extract symptoms of severe mental illness (SMI) from clinical text, with the goal of enhancing mental healthcare data for research purposes. This involves developing applications to identify SMI symptoms within mental health records using the Clinical Record Interactive Search (CRIS) data resource. While NLP and Information Extraction (IE) have been applied to clinical records, their use in mental healthcare, particularly for symptom identification, is limited. The project focuses on creating sentence classification models for various SMI symptoms to automate extraction from patient narratives [30].
Domain & users	Can be used in diagnosis by Psychiatrists. The expected result of this project is to benefit various future research and clinical applications [30].
Design method	Psychiatrists created a keyword list for severe mental illness (SMI) symptoms, focusing on commonly used, consistently recorded terms relevant to standard symptom scales. Their goal was to classify sentences in clinical records into five domains: positive, negative, disorganization, manic, and catatonic symptoms. The NLP task involved identifying symptom-related sentences, considering modifiers within eight words of a keyword as possible relations. The aim was to pinpoint clinician-assigned symptom constructs, avoiding descriptions of experiences for clarity and alignment with standard symptom documentation methods [30].
Software & Hardware	Not mentioned
Output	Extracted data for 46 symptoms with a median F1 score of 0.88. Four symptom models performed poorly and were excluded. From the corpus of discharge summaries, it was possible to extract symptomatology in 87% of patients with SMI and 60% of patients with non-SMI diagnosis [30].
Features	The system represents numerous and diverse symptomatology concepts, indicating that natural language processing is well-suited for this task [30].
Limitations	The models were tested using English text from a specific healthcare facility in the UK. It's possible that their performance may not apply as effectively when applied to texts from other institutions or regions with different medical language variations [30].

Table 13: Study 12: Natural language processing to extract symptoms of severe mental illness from clinical text

Platform 1 (Better.me)

Better.me	
Problem solved	Poor mental health is a growing pandemic, numbers of adults attempting suicide are growing, this website is a private environment to help people analyze their emotions and receive mental health support and lifestyle advice [19].
Domain & users	Adults
Design method	Used Google T5 NLP model for emotional recognition and categorizing emotions. trained data set with deep learning to develop a fine-tuned BERT model to prevent suicide [19].
Software& Hardware	Software: Python libraries, Streamlit. Hardware: not mentioned.
Output	AI journaling tool that helps analyze emotions, with a suicidal prevention algorithm and provide recommendations for mental health [19].
Features	Personal journal, NLP emotion analytics, smart recommendation for mental health resources, suicide detection [19].
Limitations	Only support English [19].

Table 14: System 1: Better.me

Platform 2 (Menty)

Menty	
Problem solved	They motivate people to address mental health concerns in a non-traditional fashion by making it fun with an interactive website where they can go on a mental health journey with their friends [23].
Domain & users	General people (English language) [23]
Design method	Machine Learning model SMTP API [23].
Software & Hardware	Software: CSS, firebase, flask, JavaScript, python, react and TensorFlow Hardware: not mentioned
Output	The sentiment analysis of the journal entry of the user mental health
Features	(Make sure that the user is continuously engaged. The sentiment for the day's entry is reflected in the opacity of the calendar UI (User Interface) elements with darker being more positive and more transparent being more negative. Also, if the sentiment is negative, the flask server makes SMTP API requests to send an email to all your friends on the app letting them know that they should check in on you. Future features: Make optimizations to their FireStore database architecture to make the project more scalable. add more rewards and incentives for the users to use their points on. They also like to research better tasks to add to the website and add an option for the user to enter their own tasks. They would like to implement a CI/CD pipeline to enable automation with the addition of new features) [23]
Limitations	Analyze English texts only.

Table 15: System 2: Menty

Platform 3 (MoodJournal)

IMoodJournal	
Problem solved	iMoodJournal addresses the need for individuals to gain insights into their moods, identify patterns, and take agile steps towards emotional well-being. The app provides a self-tracking platform, allowing users to record, analyze, and reflect on their emotional states. It aims to enhance awareness, proactive self-regulation, and accessibility to mental health resources [27].
Domain & users	Targeting the general population, iMoodJournal is also utilized by individuals with mental illnesses, assistance in tracking moods alongside other symptoms or conditions. The app supports users in gaining insights into emotional well-being and making informed decisions to improve their mood [27].
Design method	The app was evaluated for its support across the preparation, collection, reflection, and action stages of mood tracking. This analysis aimed to assess how iMoodJournal aligns with the holistic process of self-tracking and its effectiveness in supporting users through all stages [27].
Software & Hardware	iMoodJournal is a mobile app available for both Android and iOS devices. Designed to run on modern smartphones, it requires the appropriate operating system version. The app utilizes built-in features like touch screens and camera functionalities for diverse ways of recording moods [27].
Output	The output includes personalized mood tracking data, visualized in formats such as bar graphs, line graphs, and pie charts. Users can gain insights into mood patterns over time, identifying trends with other factors. Additionally, iMoodJournal allows exporting mood data in free text or spreadsheet format for further analysis or sharing [27].
Features	Examined across the stages of preparation, collection, reflection, and action, iMoodJournal offers features supporting mood tracking. It includes sharing functionalities for users to share mood data and exporting options for additional analysis or sharing [27].
Limitations	The app is limited to self-tracking and is not a replacement for professional help; its effectiveness depends on consistent user engagement, as users must actively log their moods for the app to provide meaningful insights [27].

Table 16: System 3: MoodJournal.

Platform 4 (Journly)

Journly	
Problem solved	The system offers an efficient way to express your feelings and keep track of your mental health through journaling. It allows users to track the progression of their mental health and see how it changes over time. The system will evaluate the user journal entries by using machine learning to determine their emotional polarity (-1,1). The insight section users can view their average score across all entries, identify their lowest score/ highest score along with its corresponding post and date, and observe a graphical representation illustrating the progression of their mental health over time [31].
Domain & users	Anyone who suffers mentally and has knowledge of the English language can use the app [31].
Design method	The system used the natural library, a Node.js library, which provides tools and functionality for natural language processing (NLP). It's used to perform tasks like sentiment analysis, tokenization, and other text processing tasks [31].
Software & Hardware	Software: JavaScript programming language, Node.js & Express.js were used for the backend, and HTML/CSS/Bootstrap for the frontend. Hardware was not mentioned [31].
Output	The system provides a mental health tracker that encourages users to express their emotions and understand them, leading to better self-awareness [31].
Features	The system presents an online journal that can be analyzed; each post is scored from -1 to 1, with -1 meaning negative and 1 meaning positive. A graphical representation of the user's mental health growth is provided, as it shows the progression over periods of time [31].
Limitations	The system can analyze text in the English language only. Keeps track of the user's mental health but lacks other features like recommendations and warning messages when professional help is needed [31].

Table 17: System 4: Journly

Comparison table between our System and other Platform (features)

Platform / feature	Sentiment analysis by journaling.	Tracks the user's emotions and depression rate.	Alert the user when the journals are indicating depression.	Provide a diagnosis based on the user's entry.	Offers recommendations for mantal health books.	A visual chart/graphs that illustrate how the user's emotional & mental health well-being progresses over time.	Supports Englisgh language.
(SoulGlow)	✓	✓	✓	✓	✓	✓	✓
Better.me		✓	✓				✓
Menty	✓		✓	✓			✓
Mood journal	✓	✓			✓	✓	✓
Journaly		✓	✓				✓

Table 18: Comparing SoulGlow System with other Platforms.

Comparison table between our System and other Studies (ai side)

Study/ai feature	Text-based emotion detection	Keeping track of the user's mental well-being.	Utilizing (SVN) for early detection of depression in text.	Utilizing NLP techniques such as Tokenization.	Utilizing deep learning techniques for text classification tasks.	Supports Englisgh language.
Our System (SoulGlow)	✓	✓	✓	✓	✓	✓
Study 1					✓	✓
Study 2	✓	✓	✓			✓
Study 3	✓			✓	✓	✓
Study 4		✓				
Study 5	✓	✓				✓
Study 6					✓	✓
Study 7	✓					✓
Study 8	✓		✓		✓	✓
Study 9					✓	
Study 10				✓		✓
Study 11						✓
Study 12	✓	✓			✓	✓

Table 19: Comparing SoulGlow System with other studies.

Chapter 3:
3.System Analysis

In this chapter, we aim to provide a complete understanding of what our system needs, including both technical and operational aspects. We will also look at the functional and non-functional requirements, and we'll outline the necessary hardware and software for the system to work.
To perform system analysis for the project, we gathered user requirements using a structured questionnaire. After that, we carefully studied these inputs to identify user requirements. This systematic approach helped us identify user problems, set clear goals, and establish efficient methods to achieve our project's objectives.

3.1 Requirements specification

Our main project goal is to create an online journaling system. To achieve this, we carefully looked at existing software in this area and found areas where they fell short. These shortcomings became the basis for our project's key criteria. To better understand our needs, we designed a questionnaire and determined both functional and non-functional requirements from the responses we received.

3.1.1 Questionnaires

We developed a questionnaire to efficiently gather information, and it was answered by 34 individuals. This questionnaire functions similarly to a group interview within an organization, making it easy to reach a broad audience. Respondents can fill out the questionnaire via phone, email, or online forms before sending it back. Using a questionnaire, instead of individual interviews, makes it easier to reach more people and lets them complete it when it suits them. The questionnaire covered various topics, including information security and decryption. We distributed it to computer science students and people interested and got 34 answers.

Following is our questionnaire:

Question 1.

The analysis of question 1 in Figure, shows that 58.8% of people are from (18 to 25) years old, 20.6% are from (12 to 17) years old, 14.7% are from (25 to 40) years old, and the rest are (above 40) years old, as shown in (figure 2).

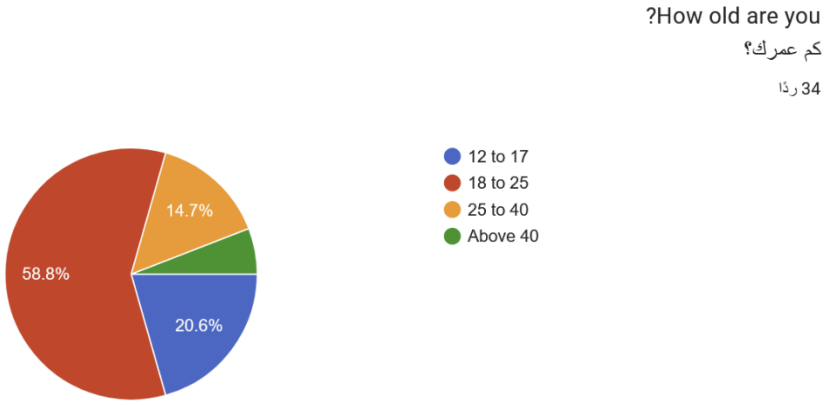


Figure 2: Result of question 1.

Question 2.

The analysis of question 2 in Figure, shows that people's answers of have you ever tried journaling before and 35.3% choses (yes, but it didn't last long), 29.4% choses (No, but I always wanted to), 20.6% choses (No, not interested), and 14.7% choses (yes, I always journal), as shown in (figure 3).

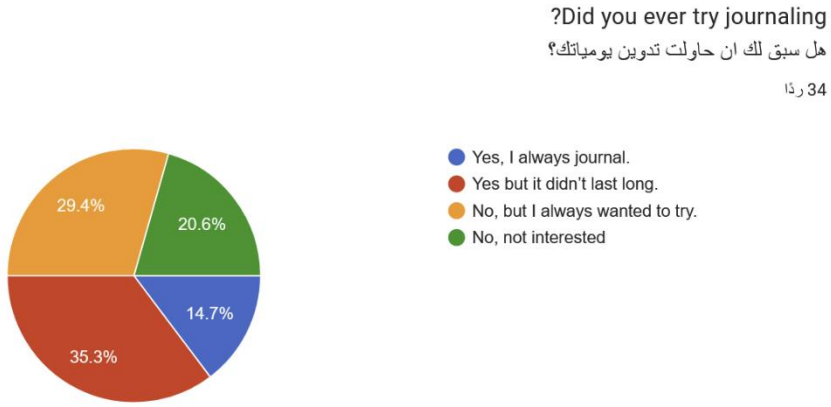


Figure 3: Result of question 2.

Question 3.

The analysis of question 3 in Figure, shows that 85.3% of people have liked the idea of analyzing their journal entries to track their mental health and provide tips to improve it and 14.7% didn't like the idea, as shown in (figure 4).

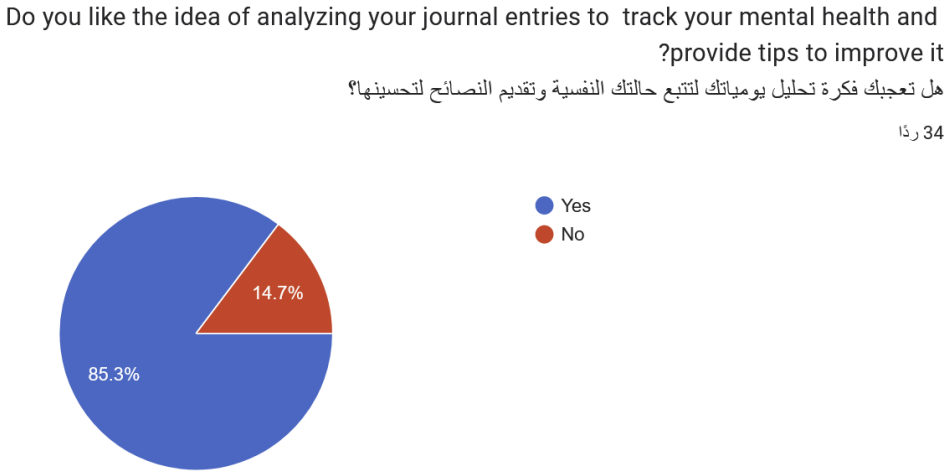


Figure 4: Result of question 3.

Question 4.

The analysis of question 4 in Figure, shows that 70.6% of people prefer to use the app with user profile for personalized features, 120.6% anonymously, and 8.8% not sure, as shown in (figure 5).

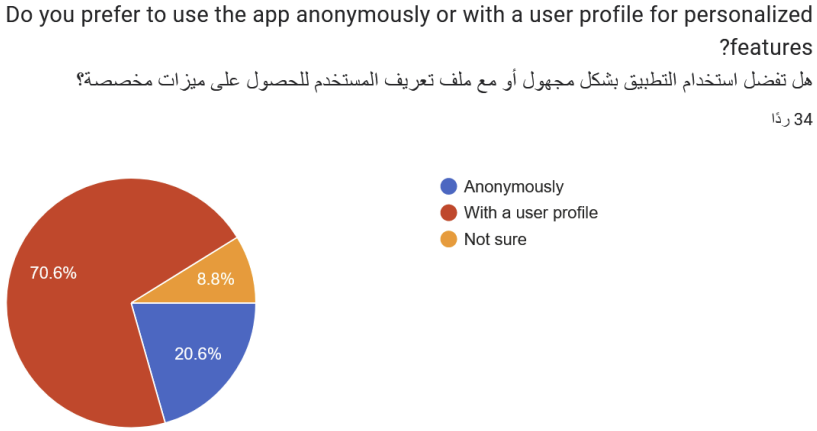


Figure 5: Result of question 4.

Question 5.

The analysis of question 5 in Figure, shows that 50% of people often plan to use a journaling app to support mental health and track their mood and thoughts weekly, 23.5% not sure, 20.6% daily, and 2.9% others, as shown in (figure 6).

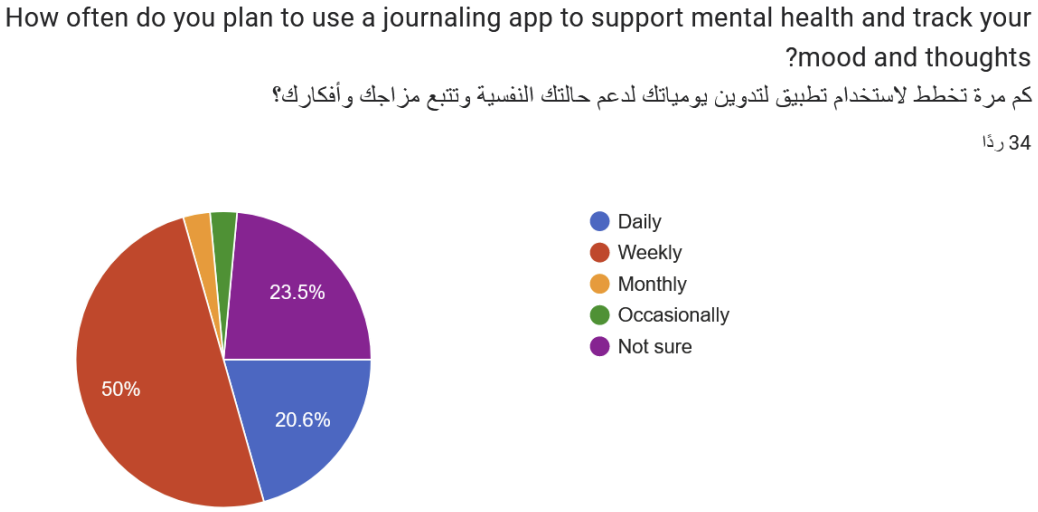


Figure 6: Result of question 5.

Question 6.

The analysis of question 6 in Figure, shows that 85.3% of people have never used a mental health journaling app before and 14.7% have used it, as shown in (figure 7).

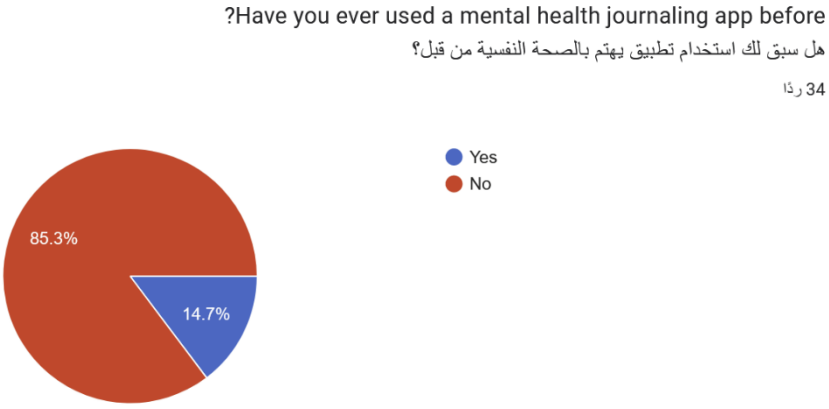


Figure 7: Result of question 6

Question 7.

The analysis of question 7 in Figure, shows that the people who have used a mental health journaling app before, have named what application they used, as shown in (figure 8).

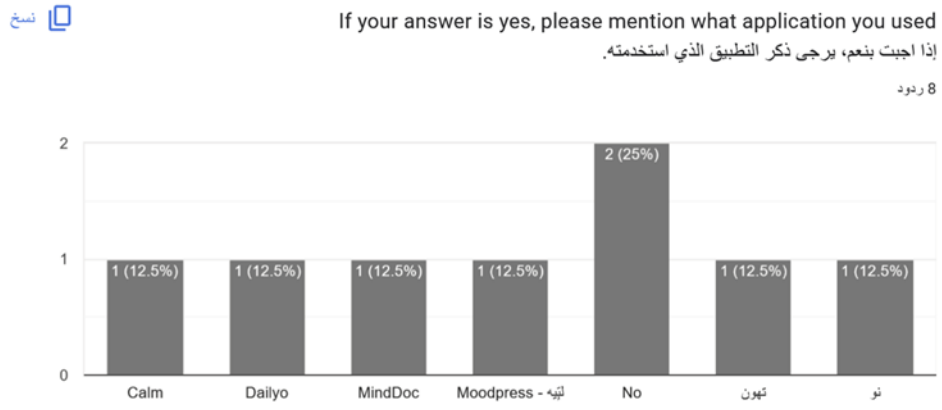


Figure 8: Result of question 7.

Question 8.

The analysis of question 3 in Figure, shows the answers of how people like to receive support or guidance for their mental state within the app and 73.5% was by tips and suggestions based on mood patterns, as shown in (figure 9).

Figure 9: Result of question 8.

Question 9.

The analysis of question 9 in Figure, shows the people's answers of do they have any specific mental health goals they hope to achieve through journaling and 79.4% was managing stress and anxiety, as shown in (figure 10).

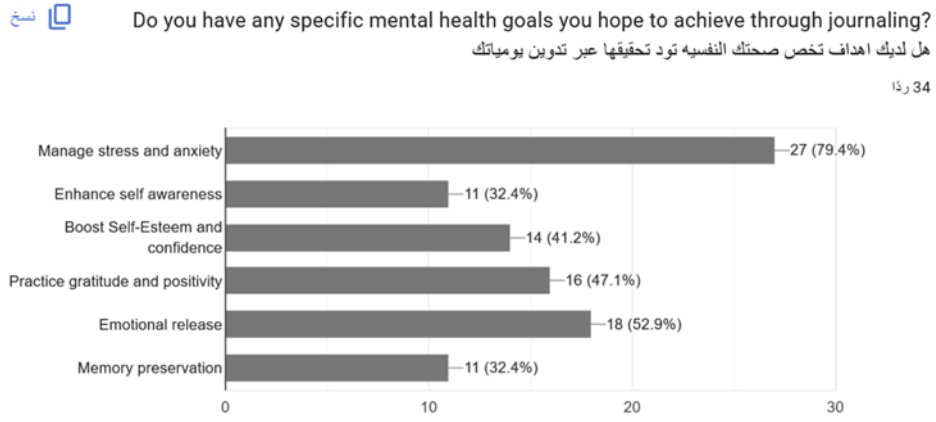


Figure 10: Result of question 9.

Question 10.

The analysis of question 10 in Figure shows the people's answers of what specific mental health condition or concern they would like the app to address and 73.5% was stress management, as shown in (figure 11).

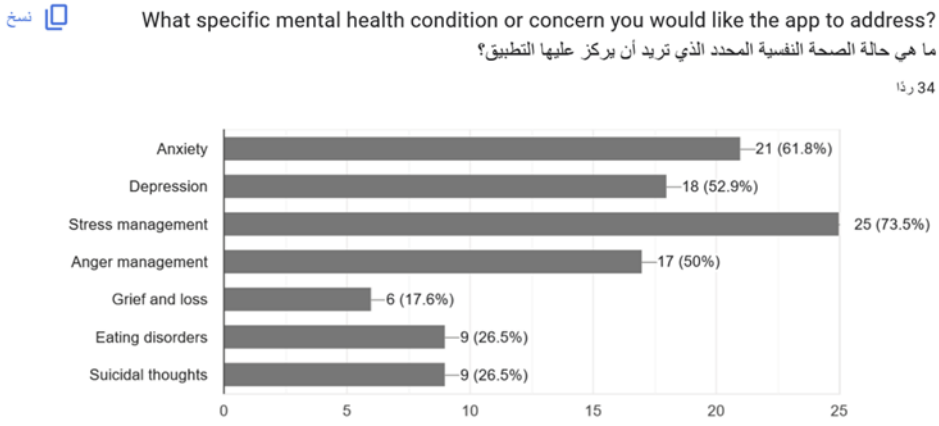


Figure 11: Result of question 10.

Question 11.

The analysis of question 11 in Figure shows people's answers of what specific aspects of their mental health are they interested in tracking and 82.4% was stress levels, as shown in (figure 12).

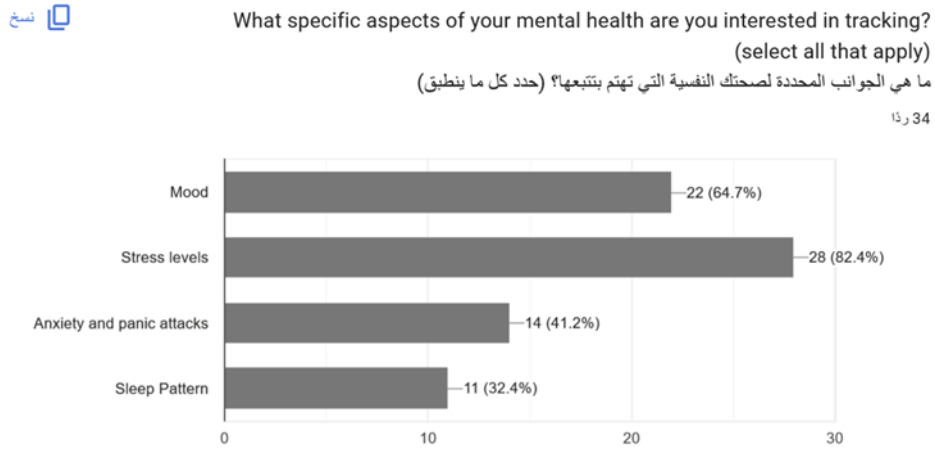


Figure 12: Result of question 11.

Question 12.

The analysis of question 12 in Figure shows people's answers of what would encourage you to use this application regularly, and 67.6% was user friendly interface, as shown in (figure 13).

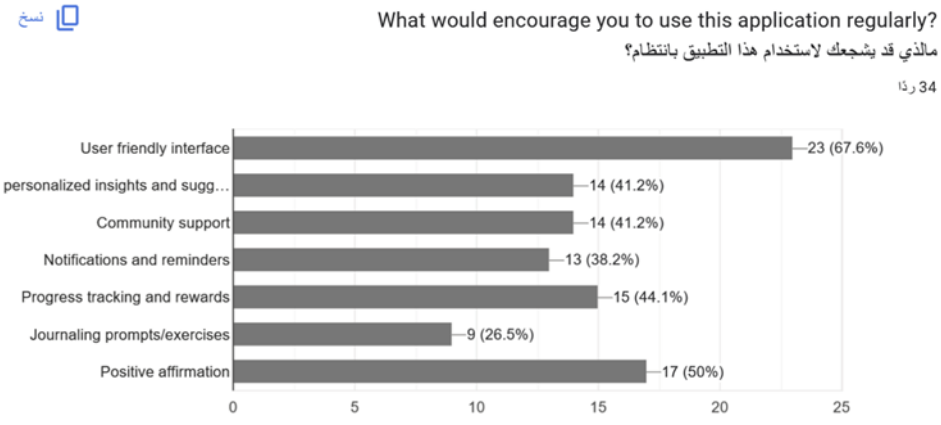


Figure 13: Result of question 12.

Question 13.

The analysis of question 11 in Figure shows people's answers of how you would maintain your mental well-being and 50% was self-help books or resources, as shown in (figure 14).

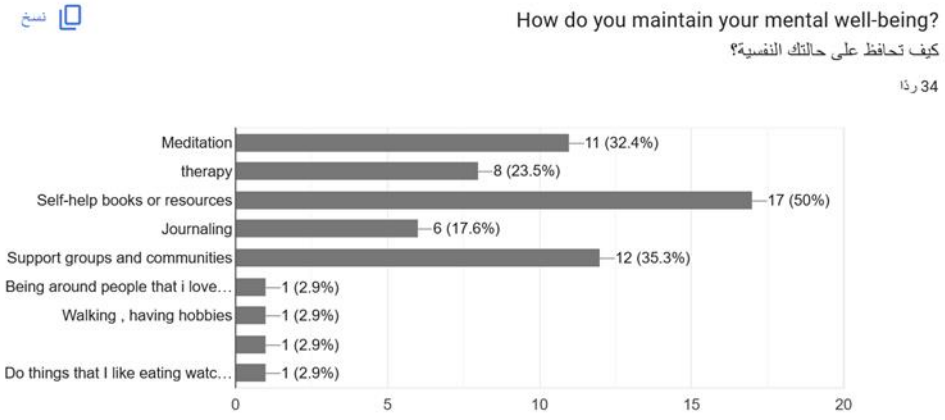


Figure 14: Result of question 13.

Question 14.

The analysis of question 14 in Figure, shows the people specific features or functionalities they expect from a mental health app, as shown in (figure 15).

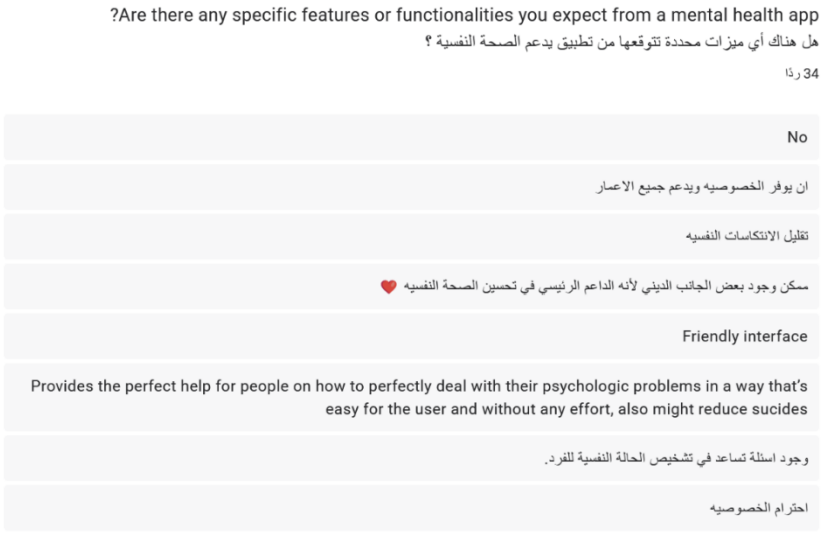


Figure 15: Result of question 14.

Question 15.

The analysis of question 15 in Figure, shows how people picture the ideal user experience while using a mental health journaling app, as shown in (figure 16).

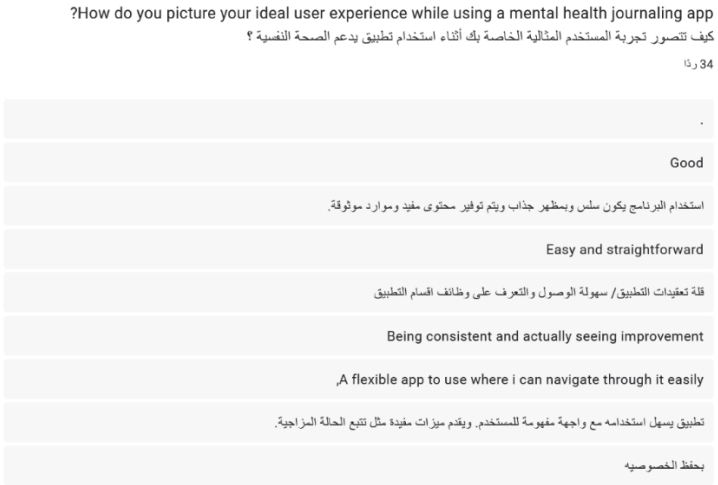


Figure 16: Result of question 15.

3.2.2 Software and Hardware requirements

1. Software requirements

Programming language: Python, HTML (Hypertext Markup Language), CSS, JavaScript.
framework: Django and Bootstrap.
Integrated development environment (IDE): Visual Studio Code, Jupyter and Google Colab.
Database management system: SQLite3.
Operating system platforms: windows, iOS, mac.
Version control: Git.

2. Hardware requirements

A computer with the following specifications:
Processor: Intel Core i7 or higher
Hard Disk Storage: Minimum of 128 GB available
Memory (RAM): At least 16 GB of RAM Internet Connection.

3.2.3. Functional non-functional requirement

1. Functional requirements

1. The user shall be able to login.
2. The user shall be able to sign up.
3. The user can logout.
4. The user can reset his/her password.
5. The user can update his/her information via settings.
6. The user can contact the SoulGlow team via email.
7. The user can view frequently asked questions (FAQ).
8. The user can delete his/her account.
9. The user can enter his/her journal.
10. The user can save his/her journal.
11. The user can delete his/her journal.
12. The user can view his/her profile.
13. The user can view mental health improvement resources.
14. The user can view daily affirmations and challenges.
15. The user can view previous journal entries (history).
16. A user will receive alerts when his/her journals are indicating depression.
17. The admin can log in.
18. The admin can logout.
19. The admin can delete user accounts.
20. The admin can view user accounts.

2. Non-functional requirements

1. Performance

The system should consistently achieve a page loading time of 5 seconds or less, ensuring that 95% of web pages load within this specified 5-second timeframe.

2. Security

Sensitive user data, including journal entries and personal information, should be encrypted, ensuring that it remains inaccessible to administrators.
Administrator access should be restricted to only essential functionalities.

3. Usability

The system's interface should successfully pass usability testing, achieving a score of at least 80 out of 100 on standard usability scales.
User feedback surveys should indicate a minimum satisfaction rate of 85% with the system's interface and navigation.

3.2 Requirements Analysis

This chapter explains the most important tasks that the proposed system performs by describing the functional and non-functional requirements that the system achieves. The requirement analysis in our project was performed through Use Case Diagram, Data Flow Diagram Level 0, Data Flow Diagram Level 1, Software and Hardware requirements. The same have been illustrated below:

3.2.1 Data flow diagrams

A data flow diagram (DFD) is like a map that shows how information moves through a system. It uses shapes and arrows to show where the information comes from, where it goes, and how it gets transformed along the way. It helps us to understand how different parts of a system work together and how data is processed and shared in our system.

1. Data flow diagram (Level 0)

Here a level zero of the data flow diagram which shows that our system will take input then generate outputs and will deal with a database to store all necessary data.

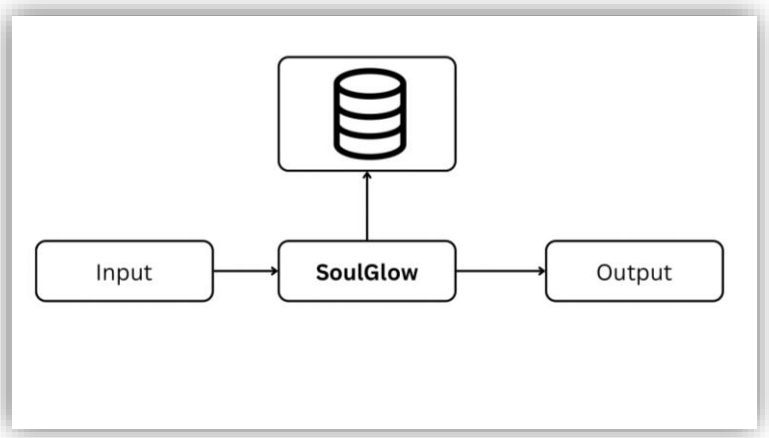


Figure 17: DFD Level0

2. Data flow diagram (Level 1)

Here is a more detailed data flow diagram than the previous one, it shows journal text as input to generate the emotions analysis report which will be the input of emotions tracking to generate graphs and mental states.

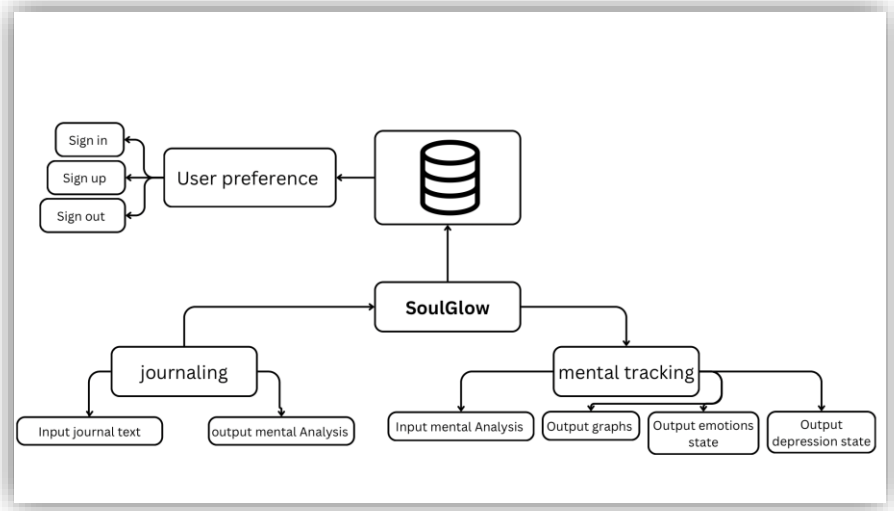


Figure 18: DFD Level 1

3.2.2 Use case diagram

In this section we describe the use cases of the user as shown in (figure 20), where the actors are the user and admin. Soul Glow system allows users to sign up or log in to document their daily experiences through journaling, Users can also delete journal entries and receive depression alerts if the journals were indicating depression after being analyzed by the depression model. Additionally, users can view their profiles where they can see the emotion and depression analysis based on the journals they entered. Users can also view mental health resources which are recommendation of mental wellness books along with viewing daily affirmations and challenges to keep them motivated. The system administrator can login, logout, delete user accounts and view user account.

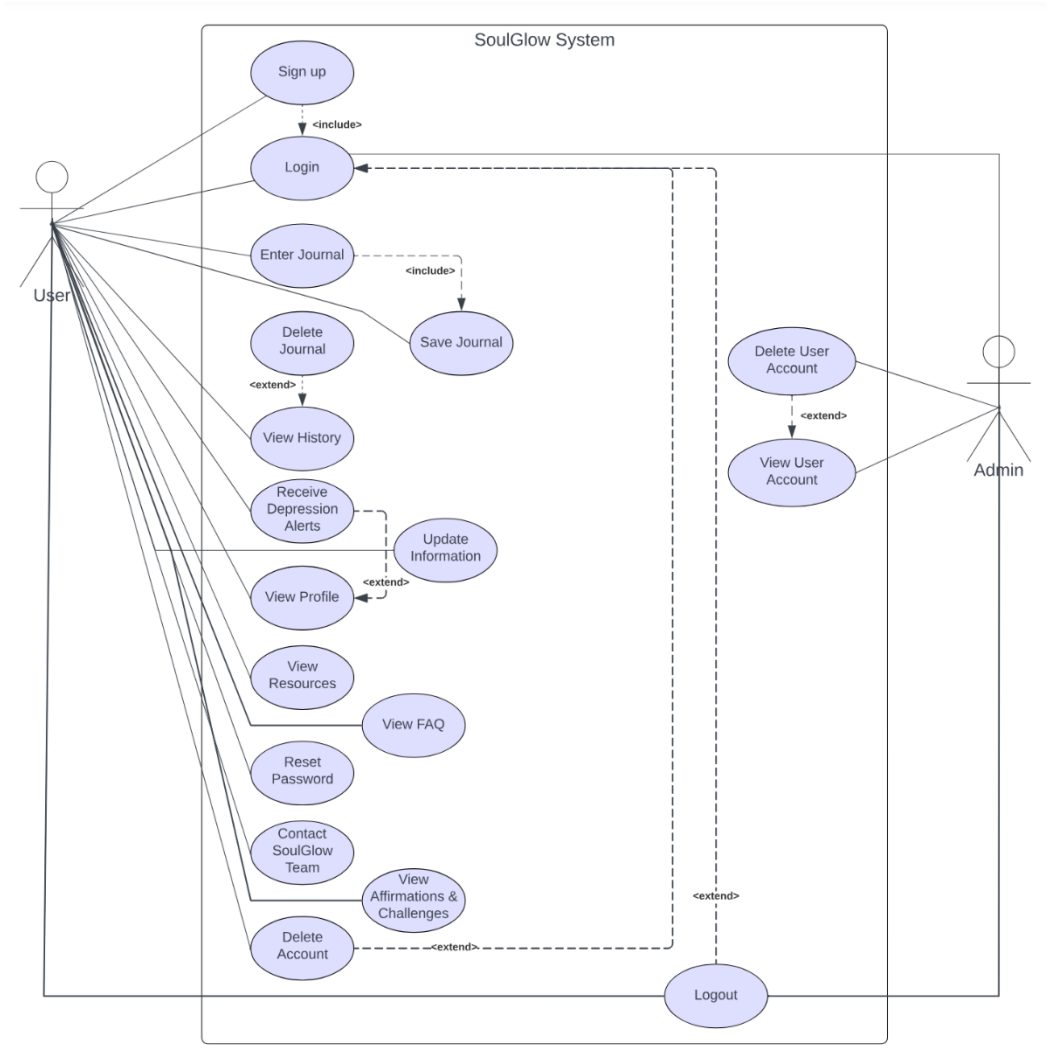


Figure 19: Use case diagram.

Use case 1 (Sign up)

UC-1	
Use case	Sign up
Actors	User
description	The "Sign up" use case involves creating a new account in the system.
precondition	The user must have access to the internet and the sign-up interface.
Post-condition	The user is successfully registered and logged in to the system.
Normal flow	The user navigates to the registration page. They provide the necessary information, including name, age, email, username, password, and confirm password. The system validates the provided information, ensuring that the email is in a valid format and the password meets the required strength criteria. Once the information is validated, the user or admin submits the registration form. The system creates a new user account and securely stores the provided information.
Alternative flow	If the provided registration information is invalid or incomplete, the system displays an error message. The user or admin is prompted to correct the information by re-entering the necessary details and resubmitting the registration form.

Table 20: Sign up.

Use case 2 (Login)

UC-2	
Use case	Login
Actors	User
Description	Users can login to the system to access their account.
Precondition	User must have access to the internet and the login interface.
Post-condition	The user is successfully logged in, gaining access to their account.
Normal flow	User navigates to the login page. User enter his registered username and password. System verifies the entered information with the stored user data. If the entered information matches the stored data, the user is successfully logged in. User gains access to their account dashboard.
Alternative flow	If the entered login information doesn't match the stored data: The system displays an error message indicating an incorrect username or password. The user is prompted to re-enter the login information.

Table 21: login.

Use case 3 (Login- Admin)

UC-3	
Use case	Login -Admin
Actors	Admin.
Description	Admin can login to the system to access their account.
Precondition	Admin must have access to the internet and the login interface.
Post-condition	Admin is successfully logged in, gaining access to his account.
Normal flow	Admin navigates to the login page. Admin enters their registered username, password, and click on the admin check button. The system verifies the entered information with the stored admin data. If the entered information matches the stored data, the admin is successfully logged in. Admin gains access to their administrative dashboard.
Alternative flow	If the entered login information does not match the stored data: The system displays an error message indicating an incorrect username or password. The admin is prompted to re-enter the login information.

Table 22: login-Admin.

Use case 4 (Save journal)

UC-4	
Use case	Save Journal
Actors	User
Description	Users have the option to express their thoughts, emotions, and experiences by writing text entries, then user can save it.
Precondition	The user must be logged into their account and have access to the journal page, and there should be an active internet connection.
Post-condition	The user successfully creates a journal entry, which may include text, picture, or both, based on their preference. The journal entry is saved in the user's account for journal analysis.
Normal flow	User navigates to the journal page. User write their thoughts, feelings, or experiences in the text box provided. Users have the option to upload pictures to their journal entry. User confirms that they have finished journaling by clicking on save for an emotion and depression analysis. The user will receive a pop-up message “your journal have been saved”. System saves the journal entry in the user's account, associating it with the current date and time.
Alternative flow	If the user didn’t enter any text or choose mood, he will receive a pop-up message to provide information

Table 23: Save journal.

Use case 5 (Enter journal)

UC-5	
Use case	Enter Journal
Actors	User
Description	Users can create meaningful journal entries within the system. Users have the option to express their thoughts, emotions, and experiences by writing text entries. Users may also enhance their entries by uploading relevant pictures, capturing moments visually.
Precondition	The user must be logged into their account and have access to the journal page, and there should be an active internet connection.
Post-condition	The user successfully creates a journal entry, which may include text, picture, or both, based on their preference. The journal entry is saved in the user's account for journal analysis.
Normal flow	User navigates to the journal page. User write their thoughts, feelings, or experiences in the text box provided. Users have the option to upload pictures to their journal entry. User confirms that they have finished journaling by clicking on save for an emotion and depression analysis. The user will receive a pop-up message “your journal have been saved”. System saves the journal entry in the user's account, associating it with the current date and time.
Alternative flow	N/A

Table 24: Enter journal.

Use case 6 (Delete journal)

UC-6	
Use case	Delete journal
Actors	User
Description	Users can manage their journal entries by deleting their previous journal entries. Users can enhance their journaling experience by removing entries that are no longer desired.
Precondition	The user must be logged into their account and have navigated to the history page of the system. The user should have at least one journal entry created.
Post-condition	After completion, the selected journal entry is deleted from the user's account. The system reflects the changes in the user's previous journal history.
Normal flow	User navigates to the history page from the navigation bar within the system. After viewing the previous journals, user chooses the specific journal entry they want to delete from the history page. User selects the delete button option for the chosen journal entry. The user will receive a pop-up message if he is sure to delete the journal entry or not. System removes the entry from the user's journal history.
Alternative flow	At any point during the delete process, the user can cancel the operation by clicking on the "cancel" button. If canceled, any changes made are discarded.

Table 25: Delete journal.

Use case 7 (View Profile)

UC-7	
Use case	View Profile
Actors	User.
Description	Users can view their previous journals which will be analyzed using NLP algorithm to perform sentiment analysis, extract keywords, detect emotions and depression, and generate insights based on the user's journal entry.
Precondition	The user must have an account and have previously recorded journal entries related to their mental health.
Post-condition	The users can view their previous journal entries to share with mental health professionals if needed.
Normal flow	Users must log in to the journaling system. Users navigate to the profile page. The system displays information and different graphical representations to the user for their emotions and depression analysis. The user can see the journals during the current week. The user can see the monthly emotional overview chart. The user can see the depression prediction and probability.
Alternative flow	If the user didn't provide any entry text, the system cannot make the analyzing.

Table 26: View Profile.

Use case 8 (View History)

UC-8	
Use case	View History
Actors	User
Description	The user accesses their history page to view their previous journal entries, by displaying the date and the time of the journal entry.
Precondition	The user must be logged into their account and have navigated to the profile page of the system. The user should have at least one journal entry created.
Post-condition	They can receive depression detection alert if found, access graphs and display their emotions over time and their depression state.
Normal flow	The user logs into their account They navigate to the history page. The system retrieves and displays a list of the user's previous journal entries. User selects a specific journal entry to view its details. User can delete any journals entries by clicking on “delete” button, then will receive a pop-up message to confirm the deletion.
Alternative flow	If there are no journal entries from the user, the system displays a message indicating that there are no journal entries.

Table 27: View History.

Use case 9 (Receive Depression Alerts)

UC-9	
Use case	Receive depression alerts
Actors	User
Description	The system detects depression signs in user's journal entry and displays an alert message that a user may need professional help to give them support and guidance.
Precondition	The user must be actively using the system that monitors their status by providing texts to help detect depression.
Post-condition	The alerts messages are generated and displayed to the user when certain criteria are met.
Normal flow	User: will enter their journal entries as text. SoulGlow System: Will continuously analyze the content provided by the user. Detects signs of Depression. Sends an alert notification to the users encouraging them to seek professional help support.
Alternative flow	The user will not receive any alert notification if the system doesn't detect any signs of depression.

Table 28: Receive Depression Alerts

Use case 10 (View resources)

UC-10	
Use case	View resources
Actors	User
Description	The system displays suggestions of various resources of books, to help users improve their mental state.
Precondition	The user must be logged into their account and navigate the explore page of the system.
Post-condition	The user has discovered the resources they desired and accessed detailed information or purchased it from external sources.
Normal flow	The user logs into their account They navigate to the explore page. The user clicks on a specific resource, transferring to another website for more information.
Alternative flow	If the user faces technical difficulties accessing the external website

Table 29: View resources.

Use case 11 (Update Information)

UC-11	
Use case	Update Information
Actors	User
Description	In the system the user can update his account information such as the username, name and email.
Precondition	The users must be logged into the system.
Post-condition	The information will be updated based on the information provided.
Normal flow	The user logs into their account. The user navigates to the “Account Information” in the settings page. It will show a pop-up message with three fields name, username, and email. After filling the fields, click on “save changes” button. The data will be updated also in the database system.
Alternative flow	Users can modify their information by returning to their "Account Information” on the settings page.

Table 30: Update Information.

Use case 12 (View FAQ)

UC-12	
Use case	View FAQ
Actors	User
Description	In the system the user can view the Frequently Asked Questions (FAQ), to help users understand a particular subject.
Precondition	The users must be logged into the system.
Post-condition	The user will view the FAQ.
Normal flow	The user logs into their account. The user navigates to the "FAQ" in the settings page. It will show a pop-up message with the Frequently Asked Question.
Alternative flow	N/A

Table 31: View FAQ.

Use case 13 (Reset Password)

UC-13	
Use case	Reset Password
Actors	User
Description	In the system if the user forgot the password, he could reset it.
Precondition	The user must already have an account on the system.
Post-condition	the user successfully reset his password.
Normal flow	The user must have an account. The user navigates to the log in page. The user will click on the "forgot password?" button. The user will provide his email that is associated with his account so he can receive a reset link in the email. Then the user will enter a new valid password then click on "confirm" button. The user will log in again with the new password.
Alternative flow	If the user entered the password less than 8 characters, he would receive a message that the password is short.

Table 32: Reset Password.

Use case 14 (Contact SoulGlow Team)

UC-14	
Use case	Contact SoulGlow Team
Actors	User
Description	The user can contact the SoulGlow team for any questions, Feedback, or suggestions, by providing the name, email address, and the message.
Precondition	The user does not need to have an account to contact SoulGlow system, only fill the fields with the invalid information.
Post-condition	the user will get a confirmation message after submission.
Normal flow	The user does not need to have an account. The user navigates to the home page. The user will go to the “contact us” section. The user will fill in the fields name, email, and the message with valid information. After the user clicks on “submit” button, he will receive a confirmation message.
Alternative flow	The user will receive appropriate warnings for incomplete or incorrect inputs.

Table 33: Contact SoulGlow Team.

Use case 15 (View Affirmation and Challenges)

UC-15	
Use case	View Affirmation and Challenges
Actors	User
Description	Users can view the daily challenges and affirmation to improve their mental health and overall well-being.
Precondition	The user must be logged into their account.
Post-condition	The user has discovered the affirmation and challenges.
Normal flow	The users log into their account. They navigate to the explore page. They can locate the section containing affirmations and challenges.
Alternative flow	If the user faces technical issues or challenges during the login process, they reach out to the technical support team for assistance.

Table 34: Vie Affirmation and Challenges.

Use case 16 (Delete account)

UC-16	
Use case	Delete account
Actors	User
Description	The user can permanently delete their account from the system.
Precondition	The user must be logged into their account. For account deletion, the user must confirm their intention to delete the account.
Post-condition	The user's account is permanently removed from the system. The user cannot log in using the deleted account anymore.
Normal flow	User navigates to the account settings. User selects the "Delete Account" option. The system presents a confirmation message, explaining the consequences of account deletion. Users confirm their intention to delete the account by clicking on Delete. The system permanently removes the user's account and associated data from the system.
Alternative flow	If the user hesitates or changes their mind after initiating the account deletion process, they can click on the “X” button. The system cancels the account deletion process, and the user's account remains active.

Table 35: Delete account.

Use case 17 (Delete User Account)

UC-17	
Use case	Delete User Account
Actors	Admin
Description	The administrator is responsible for deleting user accounts.
Precondition	For deleting user account, the admin must be logged into their account.
Post-condition	The Administrator performs the desired actions or tasks within the system.
Normal flow	The Administrator logs into the system using the admin page. The admin is presented with the system's administrative dashboard. The admin can perform administrative tasks such as managing user accounts which involves deleting them as needed. After completing administrative tasks, the admin can log out of the system.
Alternative flow	If the administrator encounters a technical issue or challenge during the login process, they reach out to the technical support team for assistance.

Table 36: Delete User Account.

Use case 18 (Logout)

UC-18	
Use case	Logout
Actors	User
Description	The User can log out from their account, terminating their current session.
Precondition	The user must be logged into their account.
Post-condition	The user's current session is terminated.
Normal flow	User navigates to the logout button on the navigation bar or account settings. User selects the "Logout" option. The user is logged out.
Alternative flow	If the user hesitates or changes their mind after initiating the account deletion process, they can click on the “X” button. The system cancels the account deletion process, and the user's account remains active.

Table 37: Logout.

Use case 19 (Logout-Admin)

UC-19	
Use case	Logout-Admin
Actors	Admin
Description	The admin can log out from their account, terminating their current session.
Precondition	The admin must be logged into their account.
Post-condition	The admin's current session is terminated.
Normal flow	Admin navigates to the logout button on the navigation bar of the administrative dashboard. Admin selects the "Logout" option. The admin is logged out.
Alternative flow	If the admin hesitates or changes their mind after initiating the account deletion process, they can click on the "X" button. The system cancels the account deletion process, and the user's account remains active.

Table 38: Logout-Admin.

Use case 20 (View User Account- Admin)

UC-3	
Use case	View User Account -Admin
Actors	Admin.
Description	Admin can login to the system to access their account.
Precondition	Admin must have access to the internet and the login interface.
Post-condition	Admin is successfully logged in, gaining access to his account.
Normal flow	Admin navigates to the login interface. Admin enters their username and password. Admin submits the login form. System validates the credentials and logs in the admin user. Admin successfully views their account details.
Alternative flow	If the admin enters invalid credentials, he won't access the admin interface. He will have to re-enter valid credentials to gain access to his account and the admin panel.

Table 39: View User Account-Admin.

3.2.3 Class Diagram

The UML diagram represents the structure of a system with four main classes: Person, User, Admin, and JournalEntry. It showcases relationships like association, composition, and inheritance between these classes.

The Person class is characterized by attributes such as name, username, password, age, email, is_staff, is_active, and date_joined, enabling interactions like signing up, logging in, and sign-out. Both Admin and User classes inherit these attributes and methods from the Person class, with additional functionalities for their roles. For instance, Admins can manage User accounts, including deletion and viewing, while Users can save journals, view profiles, and perform other actions.

The User class has a composition relationship with the JournalEntry class, implying that a User can have many JournalEntries, and if the User account is deleted, all associated JournalEntries are also deleted. JournalEntry represents entries made by Users, with attributes like text, mood, image, created_at, predicted_emotion, probability, and depression.

.

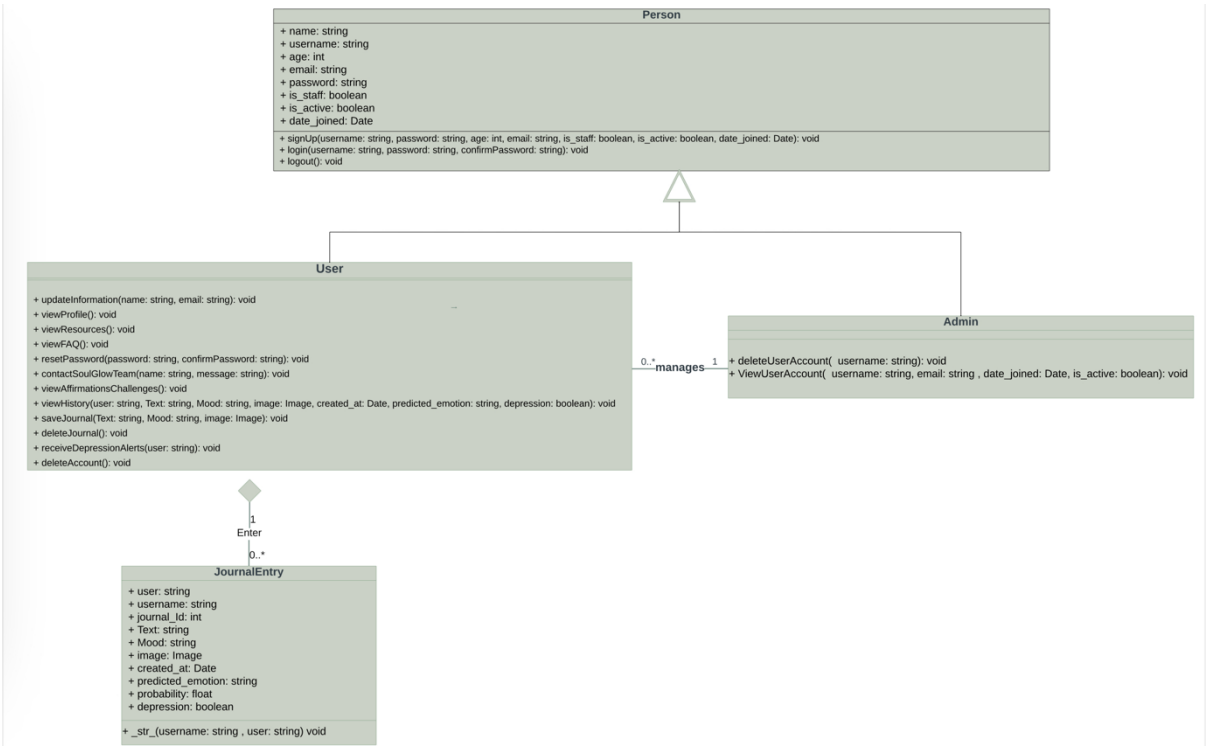


Figure 20: Class Diagram.

Chapter 4:
4.System Design

4.1 System Architecture

The system architecture is a model that defines the system's views, structure, and behavior. In simpler words is the representation and description of how the system works and communicates with other system components in general.

4.1.1 Application Architecture

In this chapter, we show the system architecture of our website, presenting the interfaces of our application and explaining the Model-View-Template (MVT) architecture that underpins it. Figure 21 illustrates the architecture of our proposed system, which follows the Model-View-Template (MVT) architecture. This architectural pattern, akin to MVC (Model-View-Controller), separates the application into three distinct components: the model, view, and template. This division enhances maintainability, understandability, and modifiability of the codebase while promoting code reusability and supporting testability by allowing individual components to be tested independently [34].

The user initiates a request through the system, which is then handled by the controller. The controller processes the user request and communicates commands to the model. The model, in turn, interacts with the database to retrieve the required data. Subsequently, the model sends the data back to the controller, which passes it to the view for rendering and display.

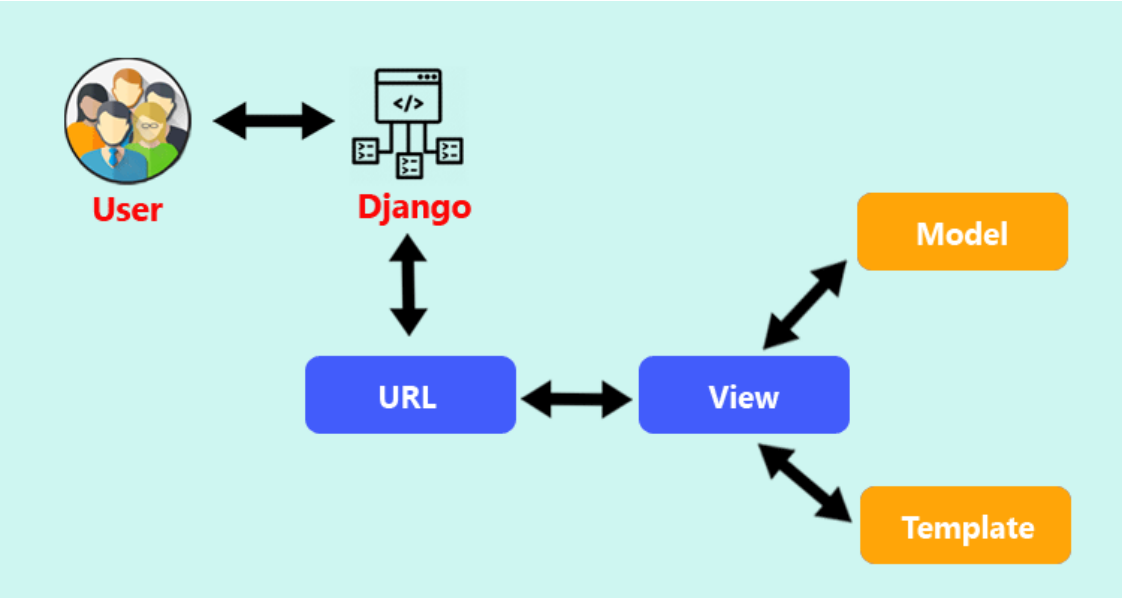


Figure 21: (MVT) architecture.

Model

The model manages the data of the application, responding to view requests and controller instructions to update the user's state [34]. In our system, the model encompasses the logic and data for emotion analysis, incorporating NLP algorithms and machine learning models for sentiment analysis. It handles the analysis of text data from user journal entries and generates results based on user requests received through the view module.

View

The view represents the user interface (UI) logic of the application, displaying information retrieved from the model [34]. In our system, the view enables users to journal and receive graphical analysis. It displays the visual graph representations of the user's emotions and depression state, as well as books aimed at improving mental well-being.

Template

The template defines the structure of the final HTML output sent to the user's browser. It integrates HTML with the data sent from the view and provides a dynamic user interface [34]. In our system, the template assists in rendering the view components, ensuring a cohesive and interactive user experience.

4.1.2 Flow Chart

The flow chart (Figure 22) illustrates the operation of the SoulGlow system.First the SoulGlow system starts at the home page and give you two option to sign up or to sign in the system, If the user chooses sign up it will lead you to Account Registration New User and it contains boxes that the user should fill such as the name, age, email, username, password, and confirm password, after that the user account will be created and confirmed.

If the user chooses to sign in, he will fill in the username, and password information only. And if the user forgot the password, he would click on the forgot password button, provide his email, then he will receive an email, and then he will reset the password.

The system will check the password validity of the user's entry in Sign in. The user will be checked if it's not admin eventually will log in to the explore page. Ther will be four pages. The user will choose one of these pages based on the task he wants to do as for the settings page, profile page, history page, and journal page.

1. In the settings page the user can check the account information, FAQ, and delete account.
2. In the journal page is where the user will make daily journal and he must click on save the text he entered, then will return to the explore page after the text is analyzed.
3. In the history page the user can see his previous journals entries.
4. In the profile page the user can view the emotion and depression analysis based on the journals he entered.

If the user is an admin, he will go to the admin page and can check on the user's page.

Finally, to log out from the system the user will click on log out button or delete account from the settings page, the administrator will go to the admin page and click on log out.

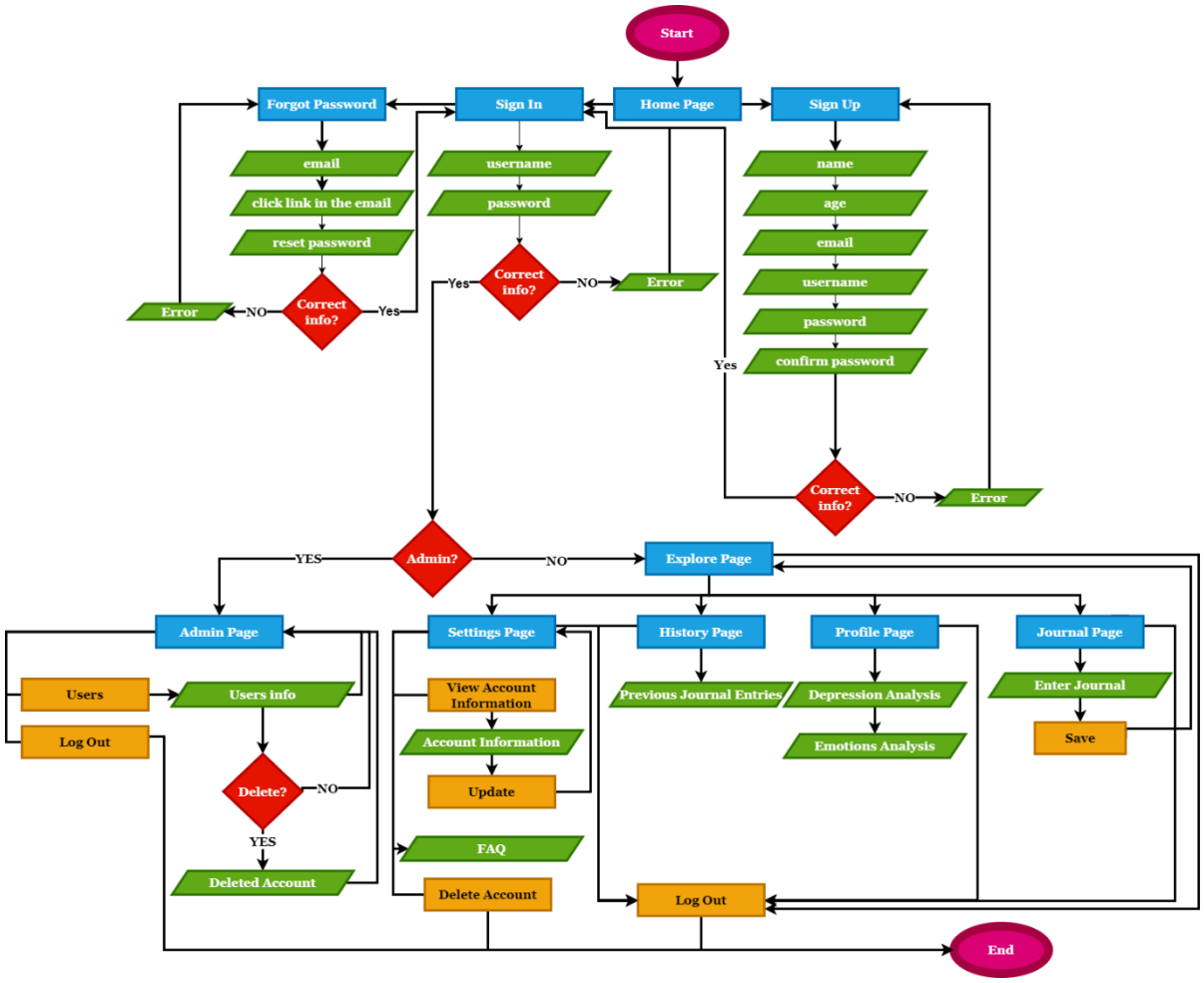


Figure 22: Flow chart.

4.1.3 Database Description (Project Database)

The project database serves as a vital repository for managing user data and interactions. It comprises two main tables: the 'User' table, storing essential user information, and the 'Journal Entries' table, enabling users to record daily experiences and emotions. These tables support key functionalities such as authentication, content management, and user engagement within the system.

1. User Table

Table [40] The "User" table serves as a central repository for storing user information within the project database. It contains essential attributes related to user accounts, facilitating authentication, authorization, and user management functionalities, this table is based on the default user table provided by Django, ensuring seamless integration with Django's user authentication system.

Attribute	Data type	Note
id	INT	Unique identifier for each user (Primary Key)
username	VARCHAR(150)	User's login username
first_name	VARCHAR(30)	First name of the user
password	VARCHAR(128)	User's password (hashed and secured)
email	VARCHAR(255)	User's email address
is_staff	BOOLEAN	Permission to access Django admin site (True/False)
is_active	BOOLEAN	Active user account status (True/False)
date_joined	DATETIME	Date and time of user’s account creation
Last_login	DATETIME	Date and time of last login

Table 40: User table

2. Journal Entries Table

Table [41] The "Journal Entries" table serves as a repository for users to log their daily entries. It enables users to record various attributes related to each entry, including the mood, associated data analysis, and optional picture.

Attribute	Data type	Note
journal_id	INT	Unique identifier for each journal entry (Primary Key)
user_id	INT	Foreign key referencing the user who created the entry
text	Byte array	Content of the journal entry which are encrypted
mood	VARCHAR(50)	User-reported mood
image	BLOB	Optional image associated with the entry
created_at	DATETIME	Timestamp indicating when the entry was created
predicted_emotion	VARCHAR(50)	System's predicted emotional tone of the entry text
Emotion_prob	FLOAT	System's confidence level in the predicted emotion
Depression_prob	FLOAT	Indicates the likelihood of the user experiencing symptoms of depression, derived from the entry's content

Table 41: JournalEntries table

4.2 User interface design

4.2.1 Home Page

SoulGlow's landing page utilizes a scrolling format to promote a healthy and mindful lifestyle. The page features a calming aesthetic, inspirational messaging ("Embrace your Journey with SoulGlow") and highlights key features ("Our Features") to address mental well-being. Users can learn more about SoulGlow's mission in the "About Us" section and connect with us through the "Contact Us" section. Overall, the page aims to inspire users and position SoulGlow as a supportive resource for achieving mental well-being.

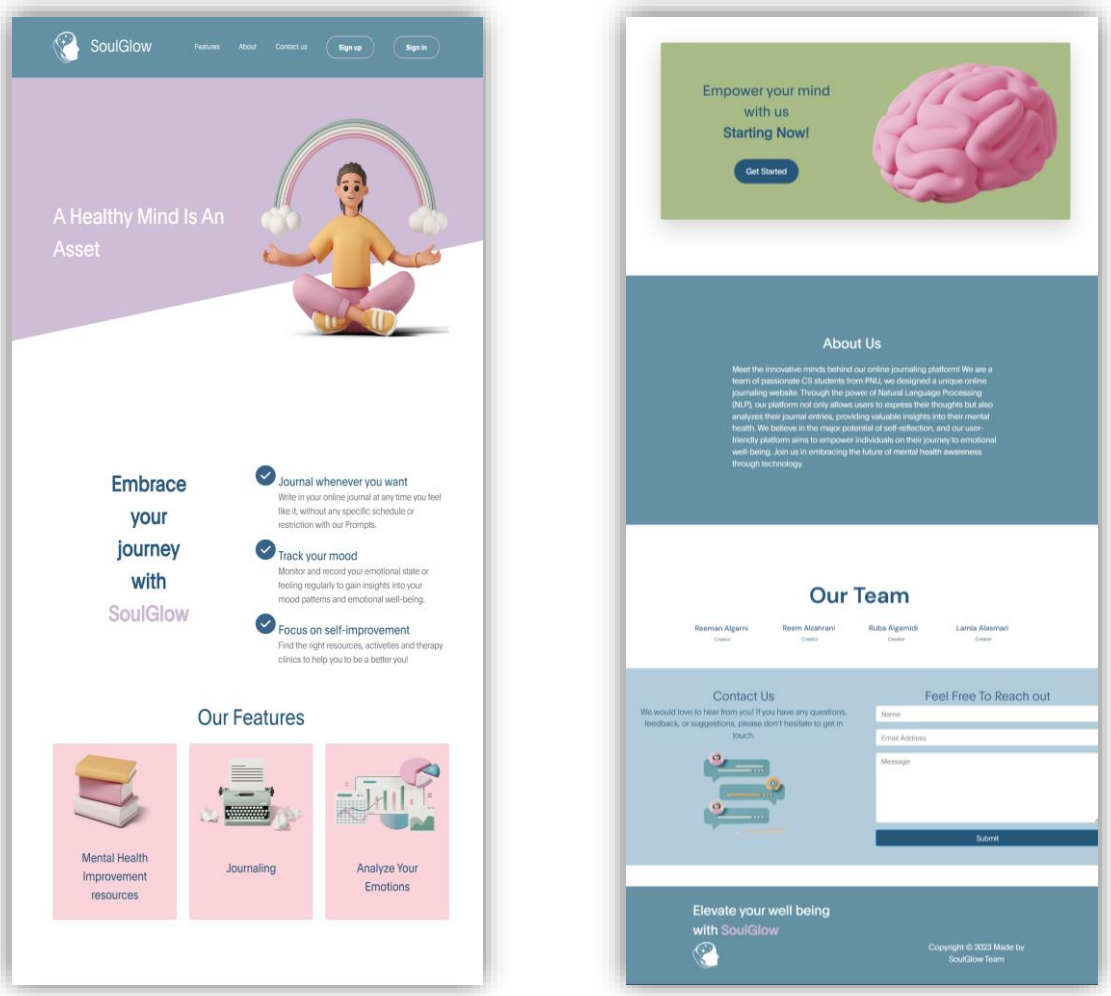


Figure 23: Home page.

4.2.2 Sign up Page

SoulGlow sign-up page captures the following information for new user registration:

1. Name field.
2. Age field.
3. Email field.
4. Username field.
5. Password field.
6. Confirm Password field to re-enter the password for verification.

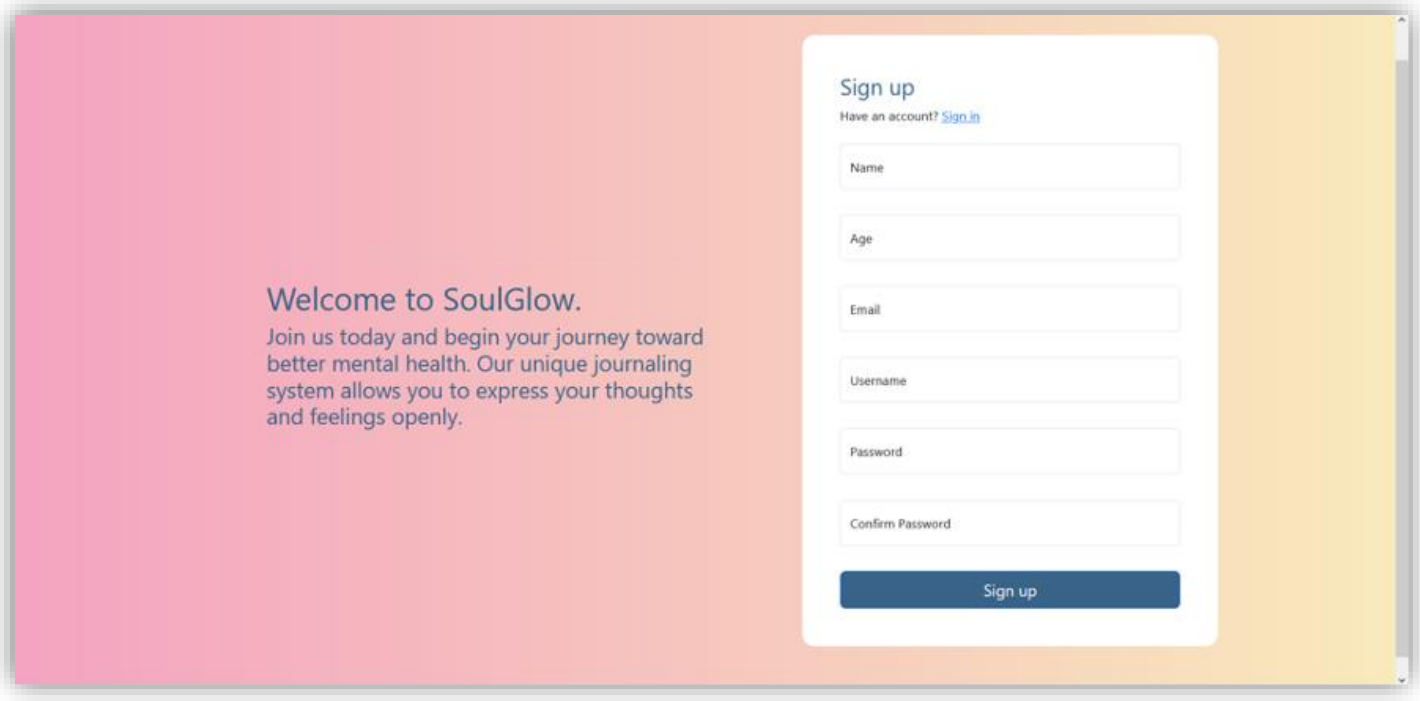


Figure 24: Sign up page.

4.2.3 Sign in page

The SoulGlow sign-in page prioritizes user access with a clear layout. It features the SoulGlow logo, username, and password fields, and a "Sign in" button. A Captcha verification step ensures security, while the accompanying text ("welcome back to soulglow. You safe space for emotional well-being") reinforces SoulGlow's focus on providing a secure and supportive environment for users. A "Forgot password" link offers help for those who have forgotten their credentials.

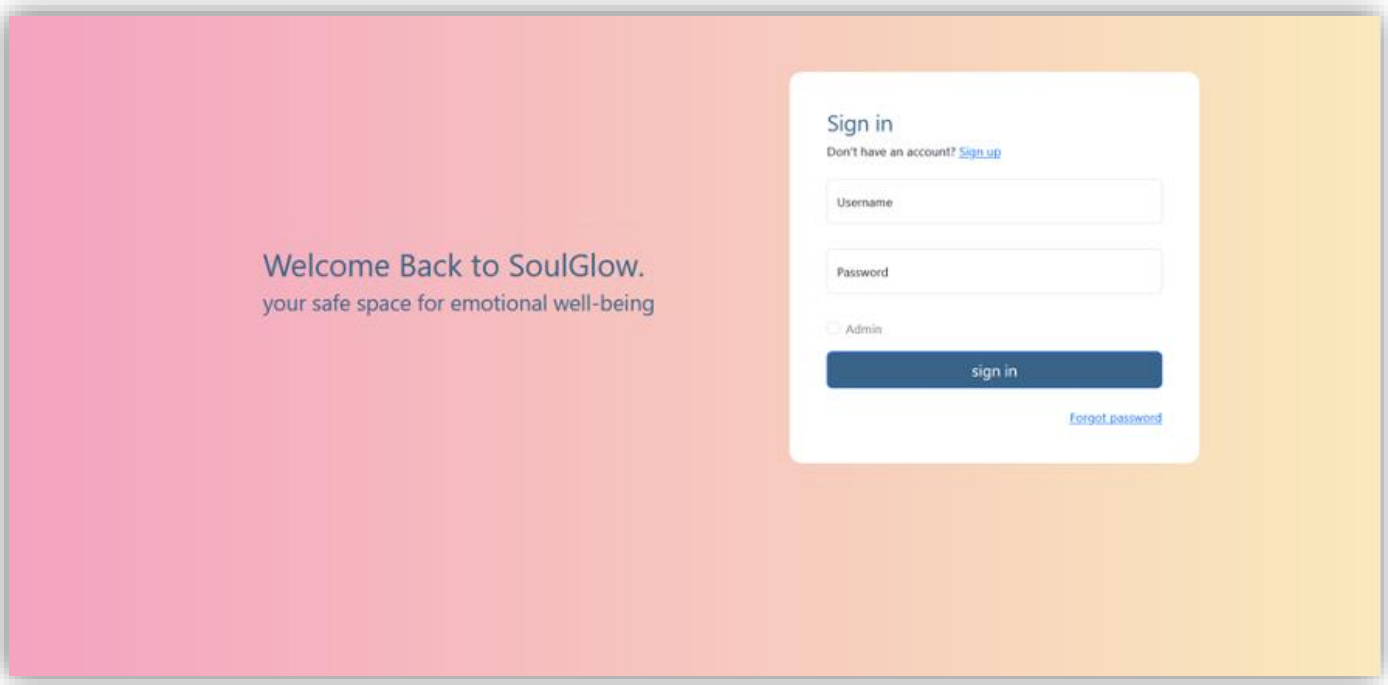


Figure 25: Sign in page.

4.2.4 Explore Page

After users sign in or sign up, they are directed to the Explore page, where they are greeted with a scroll-down interface offering various features to enhance their well-being. The Explore page presents users with daily affirmations to uplift and motivate them, along with a challenge of the day to encourage growth and self-improvement. Additionally, users can explore a selection of mental improvement books, each accompanied by detailed information about the book's content, author, and reviews. Users also have the option to purchase the books directly from Amazon for further exploration and self-development.

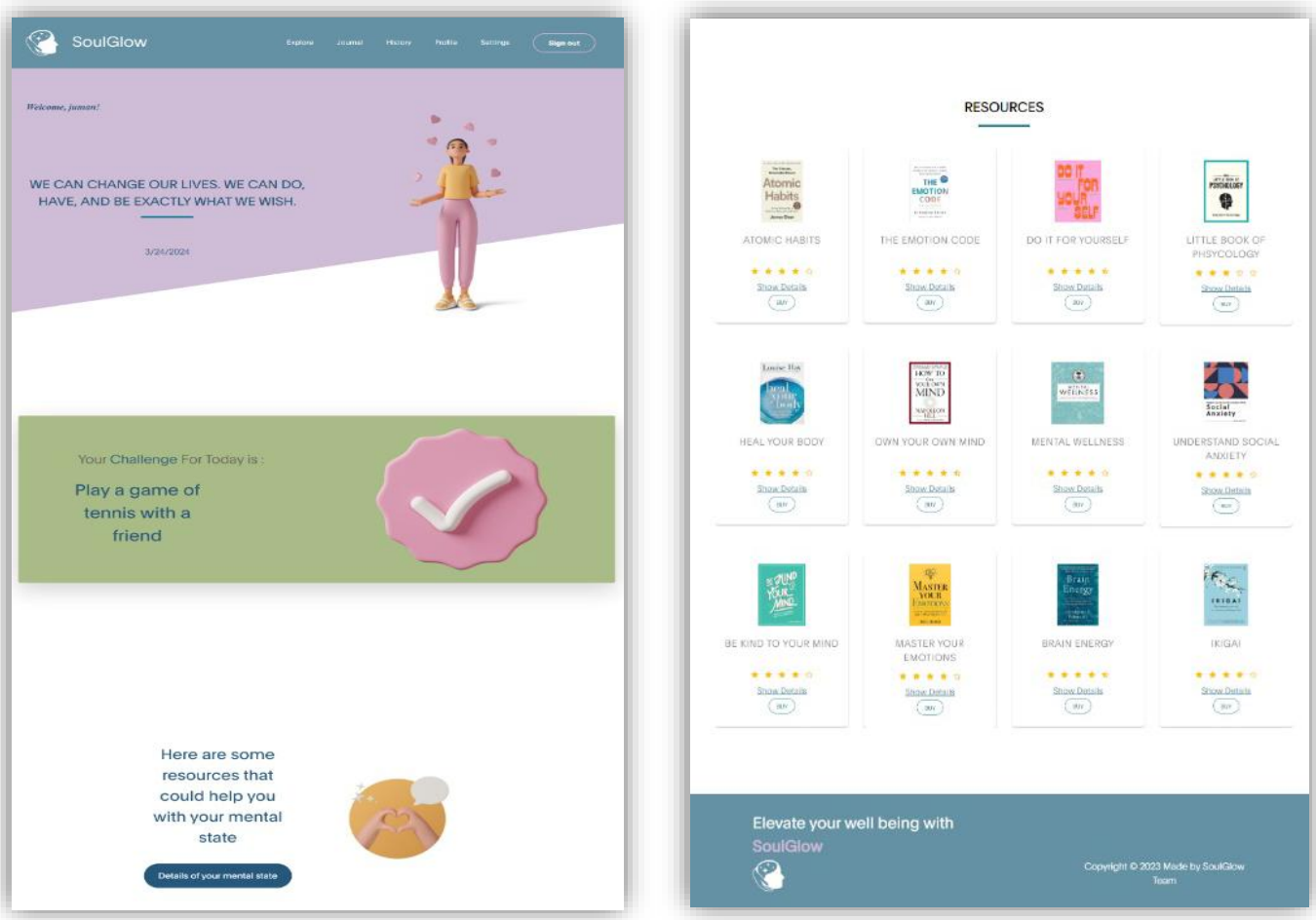


Figure 26: Explore page.

4.2.5 Journal page

journaling page prioritizes a user-friendly way to capture daily emotions. The check-in mood question offers a quick way to record a user's emotional state, while the journal entry text box allows for more detailed reflection. The loading picture section to have a better record of your day, this will help Our system to offer an analyzing service which includes NLP algorithms that analyze the journal entries, detect the user's emotions and depression, and display them in visual graphs, and gaining personalized understanding from journaling.

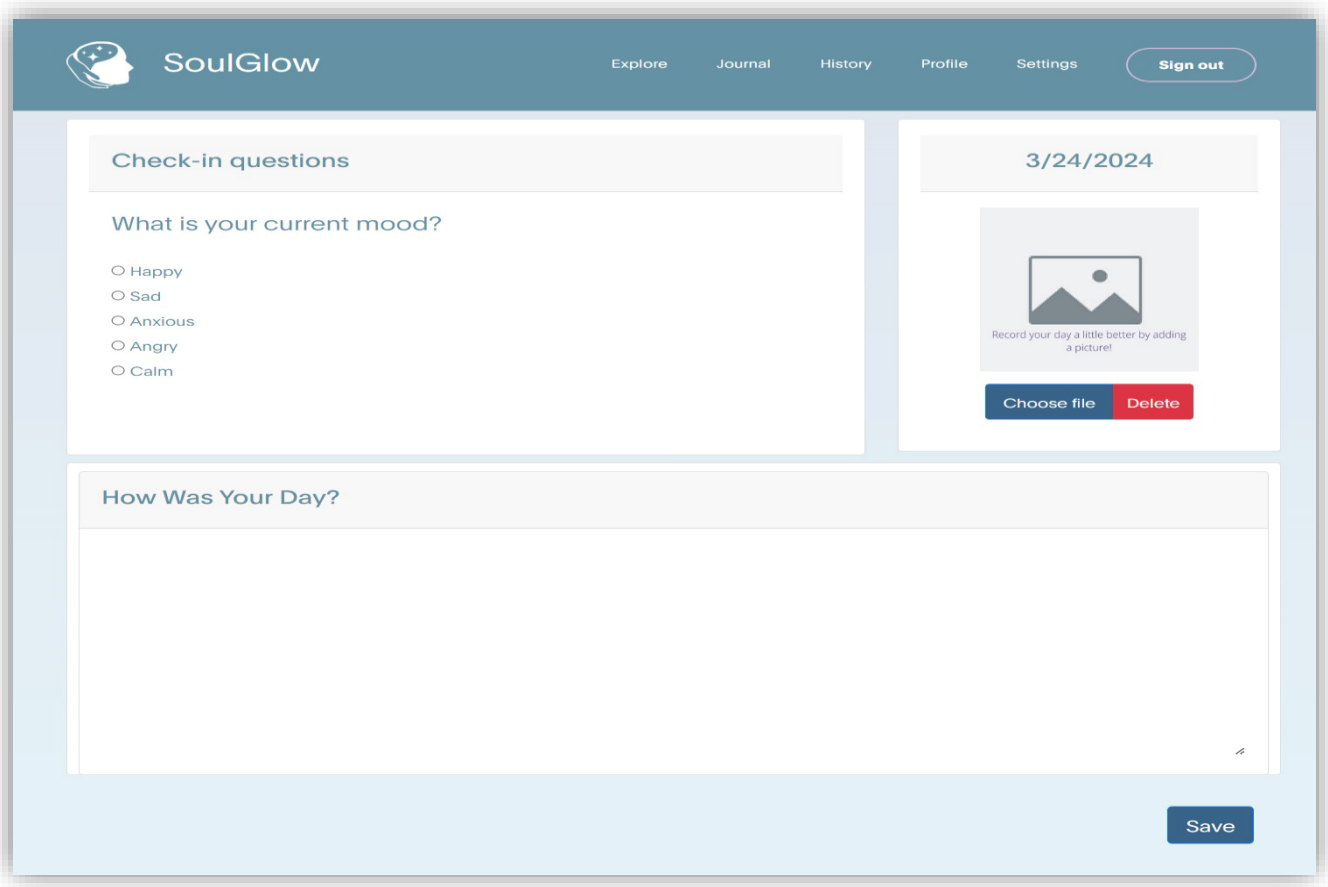


Figure 27: Journal page.

4.2.6 Profile page

In this page we display the user's mental state by detecting it from the journal entries. The system offers three graphs which are Weekly depression check, Weekly emotional chart, Monthly emotional overview where the user can have a better understanding of their mental state, also, there is two analysis sections which show the weekly depression analysis that contain the depression probability and a weekly emotional analysis that list all the emotions detected for each journal in this week.

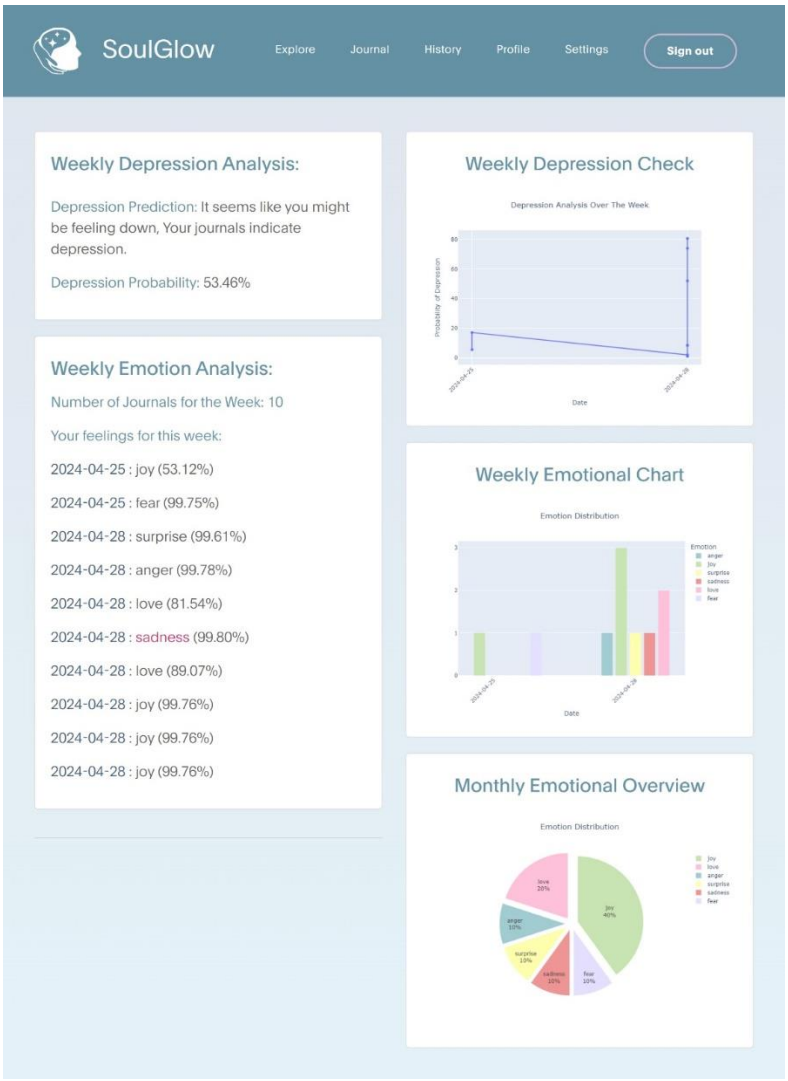


Figure 28: Profile page

4.2.7 Previous journals

Here are the previous journals for the user as shown in (Figure 29). Which list all journals of the user with ability to delete it.

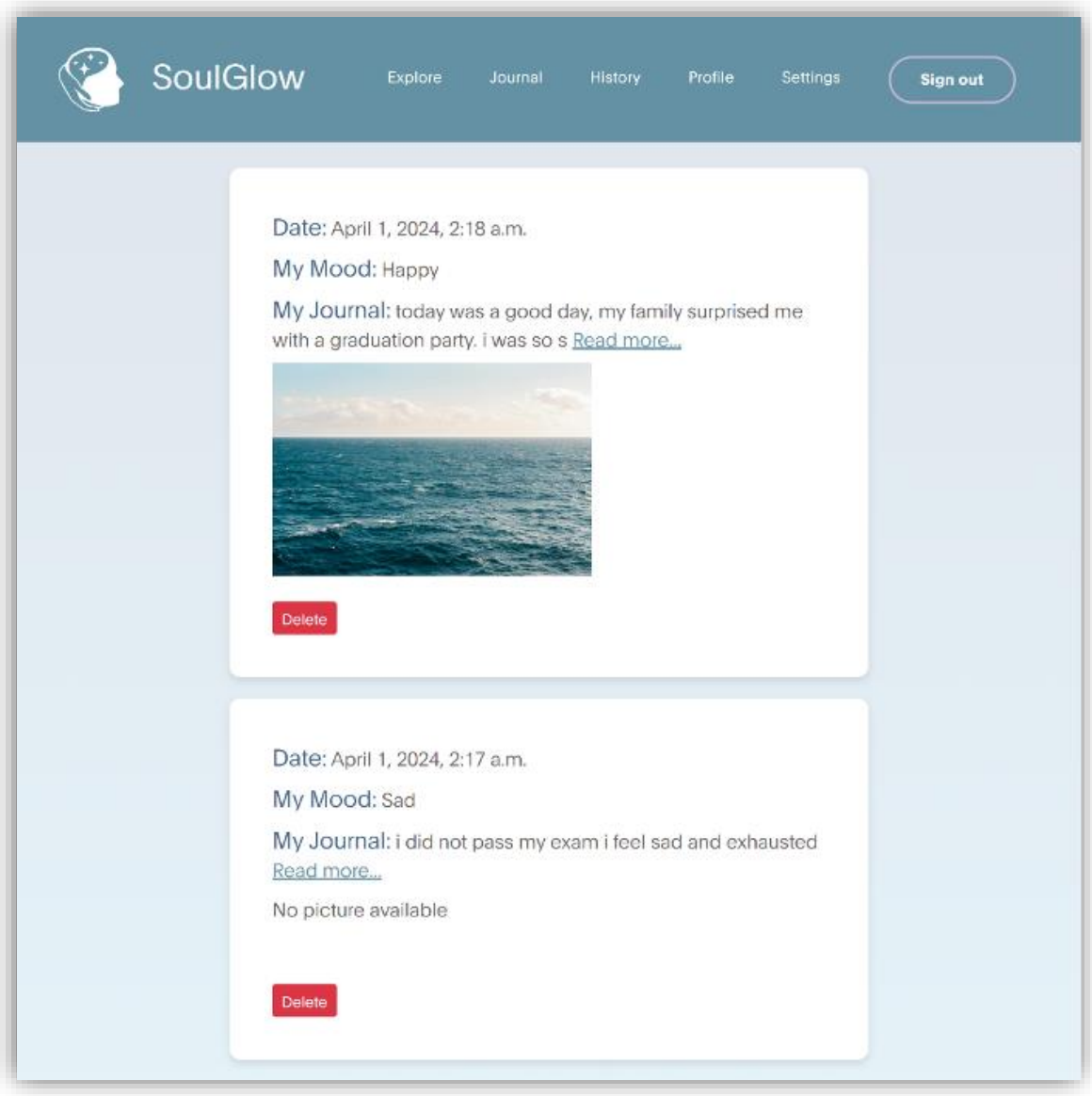


Figure 29: Previous journal page

4.2.8 Settings page

The settings page includes Account information, frequently asked questions (FAQ), and deleted account. As shown in (Figure 30).

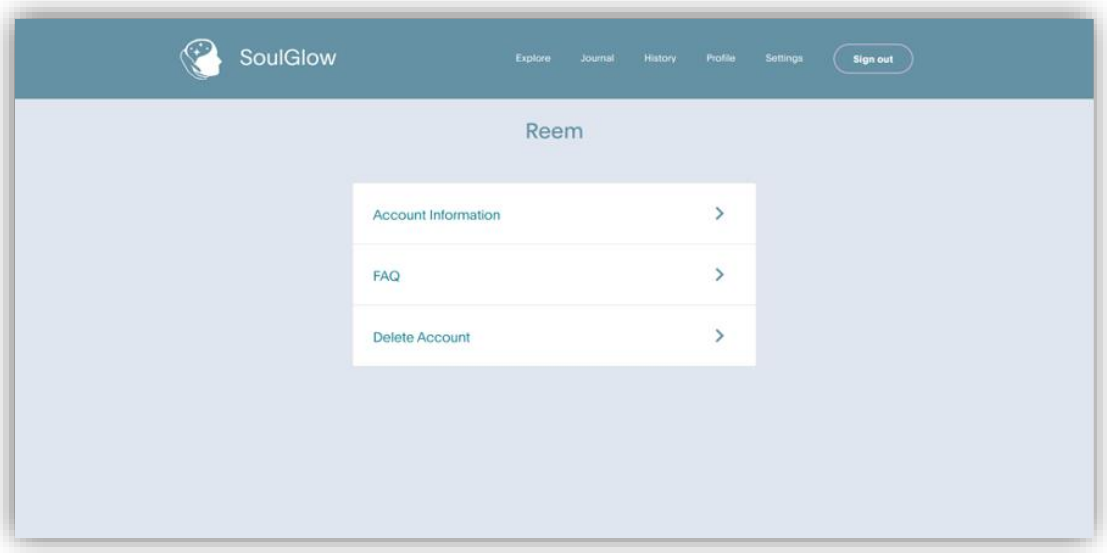


Figure 30: Settings page

The settings page content will appear as a pop-up message as shown in (figure 31)

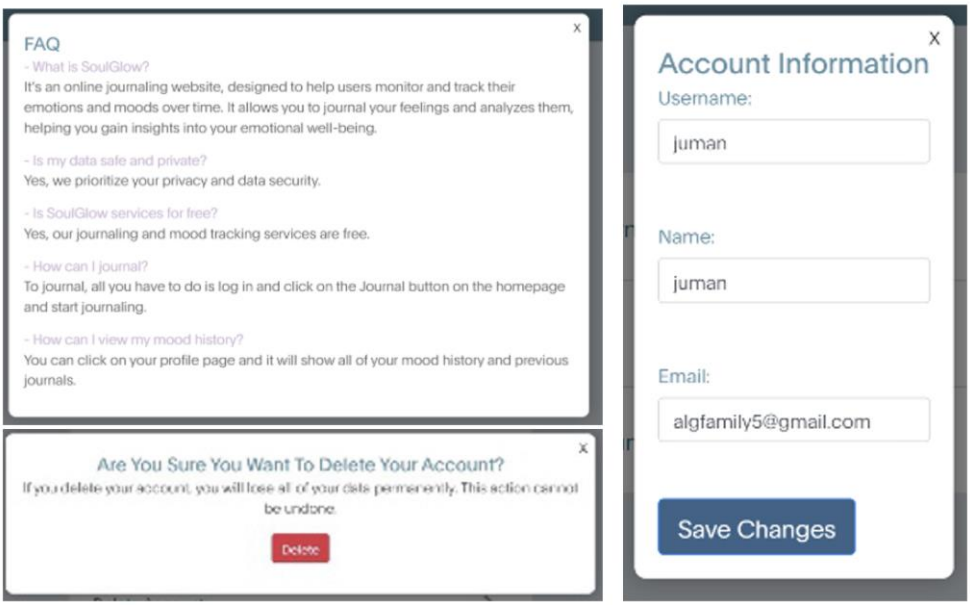


Figure 31: Settings page content

4.2.9 Admin page

This page will only be shown to the administrator to help them manage users, as shown in (Figure 32).

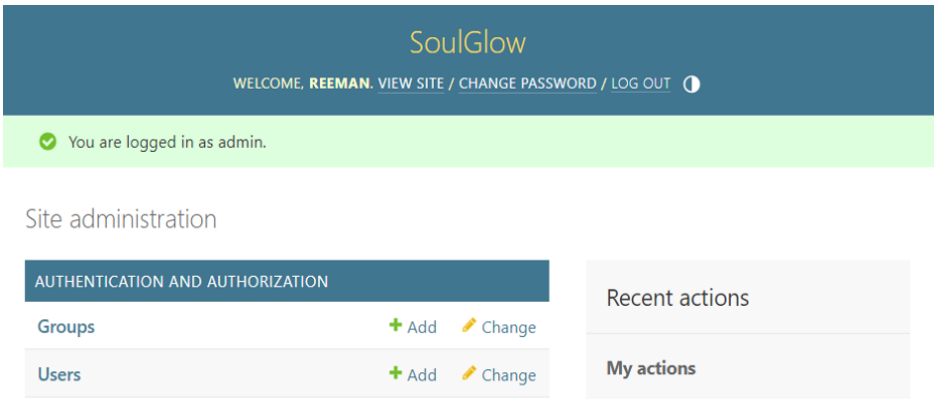


Figure 32: Admin page

As shown in (Figure 33) This window will show the users of our system.

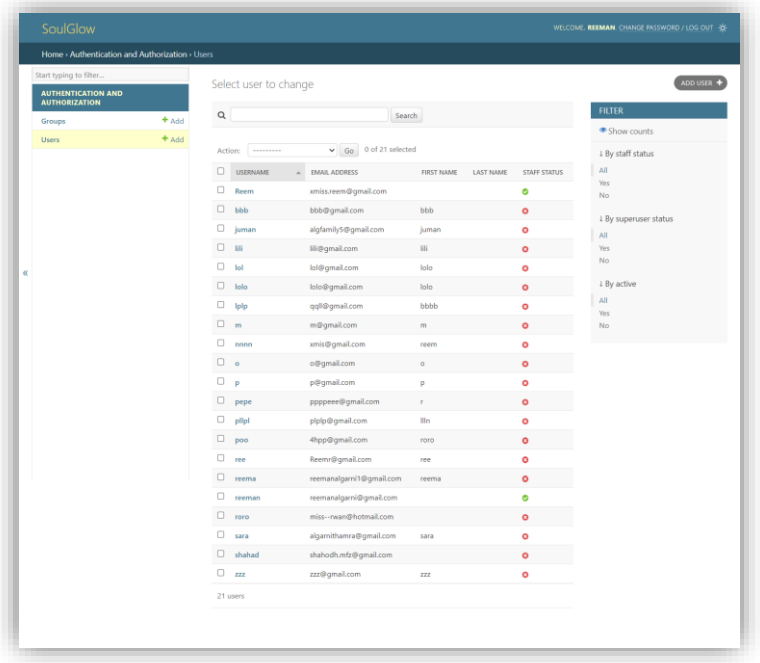


Figure 33: Users page

Chapter 5:
5. Implementation

Throughout this section, we outline the utilization of various technologies, methodologies, and tools that were employed to bring our project to fruition. From the selection of programming languages and frameworks to the configuration of development environments, we document the key components that contributed to the successful realization of our objectives.

5.1. Implementation Requirements

To implement our project, we utilized the following tools and technologies:

5.1.1 Software requirements

Integrated Development Environment (IDE):
Visual Studio Code: We used Visual Studio Code as our IDE for coding and development tasks. It provided us with a user-friendly interface and various features to enhance our productivity.

Framework:
Django: We chose the Django framework for implementing the required functionalities. Django simplified tasks such as URL routing, database interactions, and template rendering, making our development process more efficient.

Bootstrap: We utilized Bootstrap, which offers a collection of pre-designed CSS styles, components, and JavaScript plugins. It helped us create visually appealing and consistent user interfaces quickly and efficiently.

Model Training and Testing Environments:
Jupyter Notebook: We employed Jupyter Notebook for model training and testing. It provided us with an interactive environment for data exploration, model development, and evaluation, facilitating our machine learning tasks.

Google Colab: We utilized Google Colab because it provides a cloud-based environment with free access to GPU and TPU resources. This helped us run resource-intensive computations without the need for powerful local hardware. Additionally, it allowed easy sharing and collaboration on notebooks.

Programming Language:

Python: Python served as the primary programming language throughout our project. Its simplicity, readability, and extensive library ecosystem allowed us to develop both the models and system implementation with ease.

HTML, CSS, and JavaScript: These languages were essential for building and enhancing the frontend components of our web application, ensuring a seamless user experience.

Database Management System:

SQLite3: we employed SQLite3 as our database management system, providing a lightweight and efficient solution for managing structured data. Our software stack was designed to be platform-agnostic, ensuring compatibility with Windows, iOS, and macOS operating systems.

These tools and technologies played a significant role in the successful implementation of our project, providing us with efficient development environments and robust frameworks for our system.

5.1.2 Hardware requirements

we specified a computer with robust specifications to support our development and testing needs. This included a processor of Intel Core i7 or higher, ensuring sufficient computing power for demanding tasks. We also required a minimum of 128 GB of available hard disk storage and at least 16 GB of RAM to accommodate our project's data and processing requirements. An internet connection was essential for accessing online resources, collaborating with team members, and leveraging cloud-based services.

5.2. Implementation Details

The implementation of this project heavily relies on the versatility of Visual Studio Code, which serves as the primary integrated development environment (IDE) for both front-end and back-end development tasks. The project leverages a combination of core web development technologies, including HTML, CSS, and JavaScript, for front-end development, while employing Django, a high-level Python framework, for back-end development and handling front-end components. Here are the key implementation details:

5.2.1 Front-End Development

The front-end development of the project centers around the following technologies:

HTML (Hypertext Markup Language): HTML is used to structure the content of web pages, defining the layout and hierarchy of elements.

CSS (Cascading Style Sheets): CSS is employed for styling the web pages, providing visual enhancements such as colors, fonts, and layouts.

JavaScript: JavaScript is utilized to add interactivity and dynamic behavior to the web pages. It enables features like form validation, event handling, and manipulation of DOM elements.

Bootstrap: The project integrates the Bootstrap front-end framework, which offers a wide range of pre-built components and responsive design utilities. This inclusion streamlines development by providing ready-to-use components and ensuring a consistent and visually appealing user interface across different devices.

Here we present our primary and intricate HTML pages in this section, while the remaining content can be found in the appendix.

index.html

The index.html page of our website serves as the starting point for users. It provides information about our website and its benefits, including details about our features and advantages. We encourage users to sign up to fully experience our services. Additionally, there is a section that allows users to contact us.

After the user logs in, they will be directed to the "explore" page. Here, they will find affirmations and challenges of the day that change each time they refresh the page. The purpose of this is to encourage growth and self-improvement.

On the "explore" page, users can explore a collection of mentally empowering books. They will have the opportunity to learn more about each book and, if interested, they can even purchase them on Amazon.

We have implemented an if statement in the HTML code to check if the user is authenticated. This allows us to show different content based on whether the user is logged in or not. If the user is authenticated, they will see the "explore" page content; otherwise, they will only see the general introduction content.

Before user is authenticated:
Before the user is authenticated, the HTML/CSS navigation bar (navbar) on our website is designed to cater to non-authenticated users. Its purpose is to provide a simple and user-friendly interface for individuals exploring our website before signing in. When a user is not authenticated, the navbar prominently displays links such as "Features," "About," "Contact Us," "Sign Up," and "Sign In." These links enable users to navigate the website, learn about its features and offerings, and easily access the sign-up and sign-in pages when they are ready to create an account or log in.

```
<!-- MENU BAR -->
    <nav class="navbar navbar-expand-lg" style="background-color: #6491A4;">
        <div class="container">
            <a class="navbar-brand" href="#">
              <!-- <i class="fa fa-line-chart"></i>       MENU BAR -->
              <img src="{% static 'images/Logo.png' %}" class="img-fluid" alt="Logo" width="80" height="40">
              SoulGlow
            </a>

            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false"
                aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>

            <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav ml-auto">
                    <li class="nav-item">
                        <a href="#Features" class="nav-link smoothScroll">Features</a>
                    </li>
                    <li class="nav-item">
                        <a href="#about" class="nav-link smoothScroll">About</a>
                    </li>
                    <li class="nav-item">
                        <a href="#contact" class="nav-link">Contact us</a>
                    </li>
                    <li class="nav-item">
                        <a href="/signup" class="nav-link contact">Sign up</a>
                    </li>
                    <li class="nav-item">
                        <a href="/signin" class="nav-link contact">Sign in</a>
                     </li>
                </ul>
            </div>
        </div>
    </nav>
```

Figure 34 :Navbar before user is authenticated.html

The "About Us" section of the website page provides information about the team behind SoulGlow, their mission, and their values.

```
<!-- PROJECT -->
        <section class="project section-padding" id="about">
             <div class="container-fluid">
                  <div class="row">
                       <div class="col-lg-7 mx-auto col-md-10 col-12">
                            <div class="about-info">

   <h2 class="mb-4" data-aos="fade-up" style="color:white">About Us</h2>
   <p class="mb-0" data-aos="fade-up" style="color:white">
Meet the innovative minds behind our online journaling platform!
We are a team of passionate CS students from PNU, we designed a unique online journaling website. Through the power of Natural Language
Processing (NLP), our platform not only allows users to express their thoughts but also analyzes their journal entries, providing
valuable insights into their mental health. We believe in the major potential of self-reflection, and our user-friendly platform aims to
empower individuals on their journey to emotional well-being. Join us in embracing the future of mental health awareness through
technology.
                                 </p>
                            </div>
                  </div>
             </div>
        </section>
        <section class="about section-padding pb-0" id="about">
             <div class="container">
                  <div class="row">
   <div class="hero-image" data-aos="fade-up" data-aos-delay="200">
                              <img src="{% static 'images/team.png' %}"  class="img-fluid" alt="office" width= 800 height= 100>
                         </div>
                    </div>
                  </div>
             </div>
        </section>
```

Figure 35 : About Us.html

the "Contact Us" section is an important part of the website as it provides a way for users to connect with the team and get the support they need.

```
<section id="contact">
           <div class="overflow-hidden my-5">
               <div class="row justify-content-center">
                   <div class="">
                       <div class="card border-0 rounded-3 shadow-lg overflow-hidden">
                           <div class="">
                               <div class="card-body" style="background-color:#B2CCDB">
                                   <div class="row g-0">
                                       <div class="col-sm-6 d-none d-sm-block bg-image">
                                           <div class="text-center p-4">
                                               <div class="h3 fw-light" style="color:#27577B">Contact Us</div>
                                               <p class="mb-4" style="color:#27577B">We would love to hear from you! If you have any
questions, feedback, or suggestions, please don’t hesitate to get in touch.</p>
                                               <img src="{% static 'images/woman talking with chatbot.png' %}" style="width: 200px;"
alt="">
                                           </div>
                                       </div>
                                       <div class="col-sm-6 p-4">
                                           <div class="text-center">
                                               <div class="h3 fw-light" style="color:#27577B">Feel Free To Reach out</div>
                                           </div>
                                           <form method="post" action="{% url 'contact_submit' %}">
                                               {% csrf_token %}
                                               <div class="form-floating mb-3">
                                                   <input class="form-control" name="name" id="name" type="text" placeholder="Name"
required />

                                               </div>
                                               <div class="form-floating mb-3">
                                                   <input class="form-control" name="email" id="emailAddress" type="email"
placeholder="Email Address" required />
                                               </div>
                                               <div class="form-floating mb-3">
                                                   <textarea class="form-control" name="message" id="message" style="height:10rem;"
placeholder="Message" required></textarea>
                                               </div>
                                               <div class="d-grid">
```

```
                                        <button class="btn btn-primary" style="background-color:#27577B;  width:100%"
type="submit">Submit</button>
                                    </div>
                                </form>
                                {% if thanks %}
                                <div>{{ thanks }}</div>
                            {% endif %}
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </section>
```

Figure 36 : Contact Us.html

After user is authenticated:

After the user is authenticated, the navigation bar (navbar) in the web application's user interface will display links like 'Explore', 'Journal', 'History', 'Profile', 'Settings', and 'Sign out'. These links provide access to different sections and features of the website. The navbar helps users move between pages and access the functionalities available to them after logging in.

```
<nav class="navbar navbar-expand-lg" style="background-color: #6491A4;">
  <div class="container">
    <a class="navbar-brand" href="index.html">
      <!-- <i class="fa fa-line-chart"></i>       MENU BAR -->
      <img src="{% static 'images/Logo.png' %}" class="img-fluid" alt="Logo" width="80" height="40">
      SoulGlow
    </a>

    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false"
      aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>

    <div class="collapse navbar-collapse" id="navbarNav">
      <ul class="navbar-nav ml-auto">
        <li class="nav-item">
          <a href="{% url 'home' %}"; class="nav-link smoothScroll">Explore</a>
        </li>
        <li class="nav-item">
          <a href="{% url 'journal' %}";  class="nav-link smoothScroll">Journal</a>
        </li>
        <li class="nav-item">
          <a href="{% url 'prevjournal' %}";  class="nav-link smoothScroll">History</a>
        </li>
        <li class="nav-item">
          <a href="{% url 'profile' %}"; class="nav-link smoothScroll">Profile</a>
        </li>
        <li class="nav-item">
          <a href="{% url 'settings' %}"; class="nav-link">Settings</a>
        </li>
        <li class="nav-item">
          <a href="/signout" class="nav-link contact">Sign out</a>
        </li>

      </ul>
    </div>
  </div>
</nav>
```

Figure 37 :Navbar After user is authenticated.html

Affirmation API:

The server-side code (views.py) requests an affirmation from an external website called "affirmation.dev"

```python
def get_affirmation(request):
    try:
        # Send a GET request to the affirmations API
        response = requests.get('https://www.affirmations.dev/')

        # Parse the JSON response
        data = response.json()

        # Return the affirmation as JSON response
        return JsonResponse(data)

    except Exception as e:
        # If an exception occurs during the request, return an error message
        # along with the appropriate HTTP status code (500 - Internal Server Error)
        return JsonResponse({'error': str(e)}, status=500)
```

Figure 38 : Affirmation.views.py

- It converts the response into a format the server can understand (JSON).
- If successful, it sends the affirmation back to the webpage, otherwise, it sends an error message.
- The webpage has a special place for displaying affirmation messages, identified by its id "affirmation".
- Using JavaScript, the webpage requests affirmations from the server at a specific URL.
- When the server responds, JavaScript displays the affirmation on the webpage.
- If there is an error (like a connection issue), JavaScript handles it and notifies the user.

```javascript
<script>
    const affirmationElement = document.getElementById("affirmation");
    const api_url = "{% url 'get_affirmation' %}";

    async function getAffirmation(url) {
     try {
         const response = await fetch(url);
         if (!response.ok) {
             throw new Error(`HTTP error! Status: ${response.status}`);
         }
         const data = await response.json();
         console.log(data);
         affirmationElement.innerHTML = data.affirmation;
     } catch (error) {
         console.error('Error fetching affirmation:', error);
     }
    }
    getAffirmation(api_url);
    </script>
```

Figure 39 : Affirmation.JavaScript

Challenge API:

- The webpage requests a random activity from an external API called "http://www.boredapi.com/api/activity/".
- Using JavaScript, it sends a request to the API endpoint and waits for a response.
- Upon receiving the response, it extracts the activity from the JSON data.
- If there is an error during the process, it logs the error for debugging purposes.
- The webpage has a designated area for displaying the daily challenge, identified by its id "challenge".
- JavaScript updates this area with the retrieved activity from the API.
- The activity is then displayed to the user in a paragraph element with a specific styling for better visibility.

```
<p class="mb-0" data-aos="fade-up" style="font-size: 27px;">
                            {{ user.username|capfirst }} Your<strong style="color: #27577B;">Challenge</strong> For Today is
:</p><br>
<script>
    const challenge = document.getElementById("challenge");
    const api_url2 = "http://www.boredapi.com/api/activity/";

    async function getChallenge(url) {
      try {
        const response = await fetch(url);
        const data = await response.json();
        // Assuming you want to display the activity in the "challenge" element
        challenge.innerHTML = data.activity;
      } catch (error) {
        console.error('Error fetching activity:', error);
      }
    }
    getChallenge(api_url2);
  </script>
```

Figure 40: Challenge.JavaScript

By using JavaScript and APIs to dynamically generate the affirmation and challenge, the website can provide users with fresh and relevant content each time they visit the page. This helps keep the content engaging and encourages users to return for daily inspiration and motivation.

• The "Resources" section of the website features a carousel that displays a collection of book resources. Each book in the carousel is presented with an image, a title, a description, and a button.
• Users can navigate through the carousel by clicking on the arrow buttons, allowing them to explore different books in the collection.
• To provide users with more information about each book, the website utilizes the Google Books API. When a user clicks on the "Show Details" button associated with a particular book, the JavaScript function `showDetails(isbn)` is triggered, passing the book's ISBN as an argument.

Google Books API:

- Using the fetch function in JavaScript, an asynchronous request is sent to the Google Books API endpoint (https://www.googleapis.com/books/v1/volumes?q=isbn:${isbn}) to retrieve the book's details.
- Once the API response is received, the data is parsed as JSON, and relevant information such as the book's title, author(s), and description are extracted.
- These book details are dynamically inserted into the HTML modal element with the id "bookDetails" using JavaScript template literals, ensuring they are visible to the user.
- The modal, which displays the retrieved book details, is centered on the webpage to enhance the user experience.
- Error handling mechanisms are implemented to capture and log any errors that may occur during the API request or data extraction process.
- When the user chooses to close the modal pop-up, the JavaScript function `closeModal()` is invoked. This function hides the modal from view and re-enables scrolling on the webpage.
All the books in the "Resources" section focus on mental health and well-being, aiming to help users gain a better understanding of mental health, effectively managing symptoms, and enhancing their overall well-being.
After viewing the book details, users can click on the "Buy" button, which redirects them to an Amazon page where they can purchase the selected book.

Here is the closeModal(), showDetails() JavaScript function that is responsible for fetching the books information from Google Books api

```
<script>
    function showDetails(isbn) {
      // Fetch book details from Google Books API using the provided ISBN
      fetch(`https://www.googleapis.com/books/v1/volumes?q=isbn:${isbn}`)
        .then(response => response.json())
        .then(data => {
          // Extract relevant details from the API response
          const bookDetails = {
            title: data.items[0].volumeInfo.title,
            author: data.items[0].volumeInfo.authors.join(', '),
            description: data.items[0].volumeInfo.description || 'No description available.'
          };

          // Display details in the modal
          document.getElementById('bookDetails').innerHTML = `
            <p><strong>Title:</strong> ${bookDetails.title}</p>
            <p><strong>Author:</strong> ${bookDetails.author}</p>
            <p><strong>Description:</strong> ${bookDetails.description}</p>
          `;

          // Show the modal and center it on the page
          const modal = document.getElementById('myModal');
```

```
            modal.style.display = 'block';

    // Disable scrolling on the body
           document.body.style.overflow = 'hidden';
           document.getElementById("overlay").style.display = "block";
          })
          .catch(error => {
            console.error('Error fetching book details:', error);
          });
      }

        function closeModal() {
          // Close the modal
          document.getElementById("overlay").style.display = "none";
          document.getElementById('myModal').style.display = 'none';
// Enable scrolling on the body
document.body.style.overflow = 'auto';
        }
      </script>
```

Figure 41 : GoogleBooksAPI.JavaScript

Here is the HTML code responsible for fetching book information from the Google Books API:

```
  <section>
  <div class="container">
    <div class="row">
      <div class="col-md-12">
        <h2 class="affirmation">Resources</h2>
        <div id="myCarousel" class="carousel slide" data-ride="carousel" data-interval="0">
        <!-- Carousel indicators -->
        <ol class="carousel-indicators">
          <li data-target="#myCarousel" data-slide-to="0" class="active"></li>
          <li data-target="#myCarousel" data-slide-to="1"></li>
          <li data-target="#myCarousel" data-slide-to="2"></li>
        </ol>
        <!-- Wrapper for carousel items -->
        <div class="carousel-inner">
          <div class="item active">
            <div class="row">
              <div class="col-sm-3">
                <div class="thumb-wrapper">

                  <div class="img-box"> <!--1-->
                    <img src="{% static 'images/atomic habits.jpg' %}" class="img-responsive" alt="">
                  </div>
                  <div class="thumb-content">
                    <h4>Atomic Habits</h4><br>
                    <div class="star-rating">
                      <ul class="list-inline">
                        <li class="list-inline-item"><i class="fa fa-star"></i></li>
                        <li class="list-inline-item"><i class="fa fa-star"></i></li>
                        <li class="list-inline-item"><i class="fa fa-star"></i></li>
                        <li class="list-inline-item"><i class="fa fa-star"></i></li>
                        <li class="list-inline-item"><i class="fa fa-star-o"></i></li>
                      </ul>
                    </div>

                    <span  class="clickable-text" onclick="showDetails('0735211299')">Show Details</span><br>

                    <a href="https://www.amazon.sa/dp/0735211299?_encoding=UTF8&psc=1&ref_=cm_sw_r_cp_ud_dp_FNKVG3A46Y4AJWYTNZ0H" class="btn btn-
primary">Buy</a>
                  </div>
                </div>
              </div>
              <div class="col-sm-3">
                <div class="thumb-wrapper">

                  <div class="img-box"> <!--2-->
                    <img src="{% static 'images/b2.jpg' %}"class="img-responsive" alt="">
                  </div>

                  <div class="thumb-content">
                    <h4>The emotion code</code></h4><br>
                    <div class="star-rating">
                      <ul class="list-inline">
                        <li class="list-inline-item"><i class="fa fa-star"></i></li>
                        <li class="list-inline-item"><i class="fa fa-star"></i></li>
                        <li class="list-inline-item"><i class="fa fa-star"></i></li>
                        <li class="list-inline-item"><i class="fa fa-star"></i></li>
                        <li class="list-inline-item"><i class="fa fa-star-o"></i></li>
                      </ul>
                    </div>
                    <span  class="clickable-text" onclick="showDetails('1785042874')">Show Details</span><br>

                    <a href="https://amzn.eu/d/gsYPMgo" class="btn btn-primary">Buy</a>
```

```
                </div>
            </div>
        </div>

                <a href="https://www.amazon.sa/dp/1981089152?ref_=cm_sw_r_cp_ud_dp_QT3SZ65FEJS8EQYATFB2" class="btn btn-primary">Buy</a>
            </div>
        </div>
    </div>
    <div class="col-sm-3">
        <div class="thumb-wrapper">

            <div class="img-box"> <!--12-->
                <img src="{% static 'images/b111.jpg' %}" class="img-responsive" alt="">
            </div>
            <div class="thumb-content">
                <h4>Brain energy</h4><br>

                <div class="star-rating">
                    <ul class="list-inline">
                        <li class="list-inline-item"><i class="fa fa
```

Figure 42 : GoogleBooksAPI.Html

profile.html

The profile page consists of several sections that provide valuable information to the user. Here is a breakdown of each section:

1. Weekly Depression Analysis: This section presents the user's weekly depression analysis, indicating whether they are experiencing depression and their depression probability.

2. Weekly Emotion Analysis: In this section, the user can find insights from their weekly emotion analysis. It includes the number of journals written during the week and displays their emotions for each day.

3. Weekly Depression Check: This section features a chart displaying the user's cumulative depression score over time. It allows for a visual understanding of their depression levels throughout the week.

4. Weekly Emotional Chart: Here, the user can view a chart representing their emotion distribution for the week. It provides an overview of the range and frequency of different emotions experienced.

5. Monthly Emotional Overview: This section offers a chart illustrating the user's emotion distribution for the entire month. It provides a broader perspective on their emotional patterns and trends.

Additionally, within the page's body, there is a pop-up window that appears when the page is loaded. This pop-up window contains a message and a button that allows the user to close the window.
Here's an example of the code:

```
<body style="background: linear-gradient( rgb(224, 230, 237) 0%, rgba(228,242,248,1) 100%); color: #6491A4;" >

<br>
{% if no_journals_message  %}
<div class="section">
  <div class="card">
    <div class="card-body">
      <p><strong style="color: #6491A4;">{{no_journals_message }}</strong></p>
    </div>
  </div>
</div><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>
{% elif journalnum != 0 %}
<div class="container" style="padding: 2%;">
<div class="row">
  <!-- First column -->
  <div class="col-md-6">
    <!-- First Row -->
    <div class="row">
      <div class="col-md-12">
        <div class="section">
          <div class="card">
            <div class="card-body">
              <h3>Weekly Depression Analysis:</h3><br>
              {% if weekly_dep.is_depressed %}
                <p><strong style="color: #6491A4;">Depression Prediction:</strong> It seems like you might be feeling down, Your journals indicate depression.</p>

                <!-- Modal pop-up -->
                <!-- popup and overlay -->
              <div class="overlay" id="overlay"></div>
              <div class="popup text-center " id="popup">

                <h3 id="modalContent" style="padding: 20px; color: #396389;">We've noticed some concerning signs in your recent journals. It seems like you might be experiencing depression. We want you to know that you're not alone, and support is available. We encourage you to seek help and talk to someone you trust.</h3>
                <button class="redirect-button  btn-primary btn-lg" style="background: #396389; width: 60%;" id="close">OK</button>

              </div>

                <!-- JavaScript for pop-up -->
                <script>
                  // JavaScript for modal pop-up
                </script>
              {% else %}
                <p><strong style="color: #6491A4;">Depression Prediction:</strong> You're doing well! Keep it up, Your journals do not indicate depression.</p>
              {% endif %}

                <p><strong style="color: #6491A4;">Depression Probability:</strong> {{ weekly_dep.dep_probability|floatformat:"2"  }}%</p>

            </div>
          </div>
        </div>
      </div>
    </div>

    <!-- Second Row -->
    <div class="row">
```

```
        <div class="col-md-12">
          <div class="section">
            <div class="card">
              <div class="card-body">
                <h3 class="card-title">Weekly Emotion Analysis:</h3>
                <p class="card-text"><strong style="color: #6491A4;">Number of Journals for the Week: {{ journalnum }}</strong></p>
                <p class="card-text"><strong style="color: #6491A4;">Your feelings for this week:</strong></p>
                {% for  entry in current_week_entries %}
                {% with journal_date=entry.created_at %}
                  <p>
                    <span style="color: #455d7a;">{{ entry.created_at|date:"Y-m-d" }}</span> :
                    {% if "sadness" in entry.predicted_emotion %}
                      <span style="color: #c9356c;">{{ entry.predicted_emotion }} </span>
                    {% else %}
                      {{ entry.predicted_emotion }}
                    {% endif %}
                    {% if entry.probability %}
                      ({{ entry.probability|floatformat:"2" }}%)
                    {% endif %}
                  </p>
                {% endwith %}
                {% endfor %}

                {% if sadness_count > 1 %}
                <p class="card-text"><strong style="color: #6491A4;">You have been feeling <span style="color: #c9356c;">sad</span> multiple times this week, which could potentially be indicative of depression.</strong></p>
                {% endif %}
              </div>
            </div>
          </div>
        </div>
      </div>
      <hr>

    </div>

    <!-- Second column -->
    <div class="col-md-6">
      <!-- First Row -->
      <div class="row">
        <div class="col-md-12">
          <div class="section">
            <div class="card">
              <div class="card-body">
                <h3 class="card-title" style="text-align: center;">Weekly Depression Check</h3>
                <div style="text-align: center;">
                  <img src="{{ cumulative_chart_image }}" alt="Cumulative Depression Chart" class="img-fluid">
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>

      <!-- Second Row -->
      <div class="row">
        <div class="col-md-12">
          <div class="section">
            <div class="card">
              <div class="card-body">
                <h3 class="card-title" style="text-align: center;">Weekly Emotional Chart</h3>
                <div style="text-align: center;">
                  <img src="{{ emotion_chart_image }}" alt="Emotion Distribution Chart" class="img-fluid">
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>

      <!-- Third Row -->
      <div class="row">
        <div class="col-md-12">
          <div class="section">
            <div class="card">
              <div class="card-body">
                <h3 class="card-title" style="text-align: center;">Monthly Emotional Overview</h3>
                <div style="text-align: center;">
                  <img src="{{ emotion_monthly_chart_image }}" alt="Emotion Distribution Chart" class="img-fluid" >

                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
</div>
{%else %}
<div class="row">
  <div class="col-md-12">
    <div class="section">
      <div >
        <div class="card-body">
          <h3 class="card-title" style="text-align: center;">Monthly Emotional Overview</h3>
          <div style="text-align: center;">
            <img src="{{ emotion_monthly_chart_image }}" alt="Emotion Distribution Chart" class="img-fluid" height="500" width="600">

          </div>
        </div>
      </div>
    </div>
  </div>
</div>
{% endif %}

  <!-- SCRIPTS -->
  <script src= "{% static 'js/jquery.min.js' %}" ></script>
  <script src="{% static 'js/bootstrap.min.js' %}"></script>
  <script src="{% static 'js/aos.js' %}" ></script>
  <script src="{% static 'js/owl.carousel.min.js' %}" ></script>
  <script src="{% static 'js/smoothscroll.js' %}" ></script>
  <script src="{% static 'js/custom.js' %}"></script>

<script>
  window.onload = function() {
    showPopUp(); // Call the function to display the popup when the page loads

    // Attach event listener to the close button
    document.getElementById('close').addEventListener('click', function() {
      // Hide the overlay and popup
      document.getElementById('overlay').style.display = 'none';
      document.getElementById('popup').style.display = 'none';
    });
  };

  function showPopUp() {
    document.getElementById('overlay').style.display = 'block';
    document.getElementById('popup').style.display = 'block';
  }
</script>
</body>
```

Figure 43 :profile.html

5.2.2 Back-end web development

urls.py

After completing the design of all interfaces, we proceeded to the stage of linking these pages together. This was achieved by utilizing the `urls.py` file, which serves as a mapping mechanism between URL paths and view functions defined in `views.py`. `urls.py` contains Python code that defines routes for each interface on the server, allowing users to access different parts of the website/application through specific URLs.

Here is a breakdown of each pattern:

```
from django.contrib import admin
from django.urls import path, include
from . import views
from django.contrib.auth.views import (
    PasswordResetView,
    PasswordResetDoneView,
    PasswordResetConfirmView,
    PasswordResetCompleteView
)

urlpatterns = [
path('', views.home,name="home"),
path('signup', views.signup, name="signup"),
path('signin', views.signin, name="signin"),
path('signout', views.signout, name="signout"),
path('activate/<uidb64>/<token>', views.activate, name="activate"),
path('contact_submit/', views.contact_submit, name='contact_submit'),

    path('password-reset/',
        PasswordResetView.as_view(
            template_name='SoulGlow/password_reset.html',
            html_email_template_name='SoulGlow/password_reset_email.html'
        ),
        name='password-reset'
    ),
    path('password-reset/done/',
PasswordResetDoneView.as_view(template_name='SoulGlow/password_reset_done.html'),name='password_reset_done'),
    path('password-reset-confirm/<uidb64>/<token>/',
PasswordResetConfirmView.as_view(template_name='SoulGlow/password_reset_confirm.html'),name='password_reset_confirm'),
    path('password-reset-
complete/',PasswordResetCompleteView.as_view(template_name='SoulGlow/password_reset_complete.html'),name='password_reset_complete'),
    path('journal/', views.journal, name="journal"),
    path('save_journal/', views.save_journal, name='save_journal'),
    path('profile/', views.profile, name='profile'),
    path('prevjournal/', views.prevjournal, name='prevjournal'),
    path('delete_journal/<int:entry_id>/', views.delete_journal_entry, name='delete_journal_entry'),
    path('delete/<str:pk>/', views.delete_account, name='delete_account'),
    path('settings', views.settings, name='settings'),
    path('analyze_emotions/', views.analyze_emotions, name='analyze_emotions'),
    path('get_affirmation/', views.get_affirmation, name='get_affirmation'),

]
```

Figure 44 : urls.py

These URL patterns define the structure of the our website and specify which view functions should handle requests to each URL. When a user visits a particular URL, the corresponding view function will be called to generate the appropriate response.

Views.py

First off, we start by importing the necessary modules and libraries that will be used throughout our Django application. The views.py file contains the views that handle HTTP requests and return responses.

```
# Django modules
from django.shortcuts import get_object_or_404, render, redirect
from django.http import HttpResponse, JsonResponse
from django.contrib.auth.models import User
from django.contrib import messages
from django.contrib.auth import authenticate, login, logout
from django.core.mail import send_mail
from django.contrib.sites.shortcuts import get_current_site
from django.template.loader import render_to_string
from django.utils.http import urlsafe_base64_encode, urlsafe_base64_decode
from django.utils.encoding import force_bytes, force_str
from django.views.decorators.csrf import csrf_exempt
from django.contrib.auth.decorators import login_required
from django.conf import settings as conf_settings
from .tokens import generate_token
from django.conf import settings
from my_app import settings
from email.message import EmailMessage
from django.core.mail import EmailMessage

from SoulGlow.utils import preprocess_text

# Journal related
from .models import JournalEntry

# Plotting related
import plotly.graph_objs as go
import plotly.io as pio
import base64
from io import BytesIO
from datetime import datetime, timedelta, timezone

# Cryptography related
from cryptography.fernet import Fernet
cipher_suite = Fernet(settings.SECRET_KEY)

# Other
import calendar
import requests

# TensorFlow and NumPy
import tensorflow as tf
import numpy as np
from scipy.special import softmax
from datetime import datetime

# PIL and File handling
from PIL import Image
from django.core.files.uploadedfile import InMemoryUploadedFile
import base64
import json

# Depression model related
import os
import joblib
import numpy as np
```

Figure 45 : views.py

get_current_time():

- This function retrieves the current date and time in Coordinated Universal Time (UTC) and returns the current UTC datetime object. It will be used later in filtering the journals by using the current date.

```
#function returns the current time as a datetime object.
def get_current_time():
    return datetime.now(timezone.utc)
```

Figure 46 : get_current_time()

Loading pre-trained models:

- Depression Model

– svm_model and tfidf_vectorizer are loaded using joblib's load function from the files "SoulGlow\depression_svm_model.pkl" and "SoulGlow\depression_tfidf_vectorizer.pkl".
  - **svm_model** is a Support Vector Machine model trained for depression analysis.
  - **tfidf_vectorizer** is a TF-IDF vectorizer used for feature extraction in the same context.

```
svm_model = joblib.load(open("SoulGlow\\depression_svm_model.pkl", "rb"))
tfidf_vectorizer = joblib.load(open("SoulGlow\\depression_tfidf_vectorizer.pkl", "rb"))
```

Figure 47 : SVM_model

- Emotion Model

– TFAutoModelForSequenceClassification and AutoTokenizer are imported from the Hugging face transformers library.
– **model** is loaded using **TFAutoModelForSequenceClassification.from_pretrained()** from the directory **"SoulGlow\BertEmotion"**, which is directory containing the pre-trained model for emotion analysis using BERT.
– Similarly, tokenizer is loaded using AutoTokenizer.from_pretrained() from the same directory "SoulGlow\BertEmotion".

```
from transformers import TFAutoModelForSequenceClassification, AutoTokenizer
import tensorflow as tf

# Load the Emotion Model
model = TFAutoModelForSequenceClassification.from_pretrained("SoulGlow\BertEmotion")
# Load the tokenizer
tokenizer = AutoTokenizer.from_pretrained("SoulGlow\BertEmotion")
```

Figure 48 : BertEmotion_model

User Management

- **signup** and **signin** views handle user registration and login, respectively.

```
    # If the request method is not POST, return a 405 Method Not Allowed
    return HttpResponse(status=405)
def signup(request):
    # Check if the request method is POST
    if request.method == 'POST':
        # Extract form data
        name = request.POST['name']
        username = request.POST['username']
        age = request.POST['age']
        email = request.POST['email']
        password = request.POST['password']
        confirm = request.POST['confirm']
        # Check if username already exists
        if User.objects.filter(username=username):
            messages.error(request,"username already exist")
            return redirect("home")
        # Check if email already exists
        if User.objects.filter(email=email):
            messages.error(request,"email already exist")
            return redirect("home")
        # Check if username is too long
        if len(username) > 10:
            messages.error(request,"username is too long")
        # Check if passwords match
        if password !=  confirm:
            messages.error(request, "passwords don't match")
        # Check if username is alphanumeric
        if not username.isalnum():
            messages.error(request,"username is too long")
            return redirect("home")
        # Convert age to an integer before comparing
        age = int('0'+age)
        # Check if age is at least 12 years old
        if age < 12:
            messages.error(request, "You must be at least 12 years old to sign up")
            return redirect("home")
        # Create the user
        myuser = User.objects.create_user(username, email, password)
        myuser.first_name = name
        myuser.Age = age
        myuser.is_active= False
        myuser.save()
        # Success message
        messages.success(request, "Please check your email inbox and follow the instructions to confirm your email address.")
        # Send welcome email
        subject = "Welcome to SoulGlow"
        message = "hello " + myuser.first_name +"\n hope you have a lovely journey with us"
        from_email = conf_settings.EMAIL_HOST_USER
        to_list = [myuser.email]
        send_mail(subject, message, from_email, to_list )
        # Send confirmation email
        current_site = get_current_site(request)
        email_subject = "Confirm your email @ SoulGlow "
        message2 = render_to_string('email_confirmation.html', {
            'name': myuser.first_name,
            'domain': current_site.domain,
            'uid': urlsafe_base64_encode(force_bytes(myuser.pk)),
            'token' : generate_token.make_token(myuser)
        })
        email = EmailMessage(
            email_subject,
            message2,
            conf_settings.EMAIL_HOST_USER,
            [myuser.email],
        )
        email.fail_silently = True
        email.send()
        return redirect('signin')
    # Render the signup page if request method is not POST
    return render(request, "SoulGlow/signup.html")
def signin(request):
    # Check if the request method is POST
    if request.method == 'POST':
        # Extract username and password from the POST request
        username = request.POST['username']
        password = request.POST['password']
        # Authenticate user credentials
        user = authenticate(username=username, password=password)

        if user is not None:
            if user.is_staff:
                # If user is staff/admin, login and redirect to admin index page
                login(request, user)
                messages.success(request, 'You are logged in as admin.')
                return redirect('admin:index')
            else:
                # If user is a regular user, login and redirect to home page
                login(request, user)
                messages.success(request, 'You are logged in.')
                return redirect('home')
        else:
            # If authentication fails, display error message
            messages.error(request, 'Invalid credentials.')

    # Render the signin page if request method is not POST
    return render(request, 'SoulGlow/signin.html')
```

Figure 49: signup and signin views.py

- **signout** logs out the user.

```
def signout(request):
    # Logout the user
    logout(request)
    messages.success(request, "logged out "  )
    # Redirect to the home page after logout
    return redirect("home")
```

*Figure 50: **signout** views.py*

- **activate** activates the user's account after email verification.

```
def activate(request, uidb64, token ):


    try:


        # Decode the user ID from base64


        uid = force_str(urlsafe_base64_decode(uidb64))


        # Retrieve user with the decoded ID


        myuser = User.objects.get(pk=uid)




    except (TypeError, ValueError, OverflowError, User.DoesNotExist):


        myuser = None




    # Check if user exists and the token is valid


    if myuser is not None and generate_token.check_token(myuser, token):


        # Activate the user and login


        myuser.is_active = True


        myuser.save()


        login(request, myuser)


        return redirect('home')


    else:


        # Render activation failed page if activation fails


        return render(request, "activation_failed.html" )
```

Figure 51: activate views.py

- **Settings** view displays the settings page.
  1. Checks if the request method is POST to handle form data submission for updating user settings.
  2. Retrieves the currently logged-in user from the request.
  3. Updates the user's first name, email, and username based on the submitted form data.
  4. Saves the updated user information.
  5. Displays a success message using Django's messages framework to notify the user about the successful update.

6. Redirects the user back to the explore (home) page after successful submission.
7. Renders the settings.html template with the user object if the request method is not POST.

```
@login_required
def settings(request):
    if request.method == 'POST':
        user = request.user
        user.first_name = request.POST.get('name')
        user.email = request.POST.get('email')
        user.username = request.POST.get('username')
        user.save()
        messages.success(request, "Your settings have been updated successfully.")
        return redirect('home')  # Redirect to the settings page again
    return render(request, "SoulGlow/settings.html", {'user':request.user})
```

Figure 52: setting views.py

- **Delete_account** view is responsible for deleting the users account per user request from the settings page.

```
def delete_account(request,pk):
        queryset = User.objects.get(id = pk)
        if request.method == 'POST':
          queryset.delete()
        return redirect('home')  # Redirect to home page after deletion
```

Figure 53:Delete_account views.py

- **Home** view displays the index page, which the user sees before signing in.

```
def home(request):
    return render(request, "SoulGlow/index.html")
```

Figure 54:Home views.py

- **Journal** view displays the journaling page.

```
def journal(request):
     return render(request, "SoulGlow/journal.html")
```

Figure 55:journal views.py

- **contact_submit**

This function handles the submission of a contact form on a website. it uses Django's built-in EmailMessage class from the django.core.mail module to compose an email.

1. Checks if the request method is POST to ensure form data submission.
2. Retrieves form data including name, email, and message fields from the request.
3. Composes an email using Django's EmailMessage class, containing the form data, then Sends the email to Soulglow.website@gmail.com.
4. Renders a template (index.html) with a thank you message upon successful submission.
5. Returns a 405 Method Not Allowed response if the request method is not POST.

```
def contact_submit(request):
    if request.method == 'POST':
        # Retrieve form data
```

```
        name = request.POST.get('name')
        email = request.POST.get('email')
        message = request.POST.get('message')

        # Do something with the form data (e.g., send an email)
        # For example, you can use Django's built-in EmailMessage class:
        from django.core.mail import EmailMessage
        email = EmailMessage(
           subject='New Contact Form Submission',
           body=f'Name: {name}\nEmail: {email}\nMessage: {message}',
            to=['Soulglow.website@gmail.com']
        )
        email.send()

        # Return a success response
        return render(request, "SoulGlow/index.html", {'thanks': 'Thank you for your
message!','scroll_to_contact': True})

    # If the request method is not POST, return a 405 Method Not Allowed
    return HttpResponse(status=405)
```

Figure 56:contact_submit views.py

- **save_journal** view saves the journal entry, here are the steps:

1. This function is triggered when a POST request is made.
2. It retrieves the journal text and mood from the request's POST data.
3. It checks if both the journal text and mood are provided. If not, it returns an error response.
4. It tries to retrieve an image file if uploaded.
5. It encrypts the journal text using a the function **encrypt_data**.
6. It analyzes the emotions of the journal text using the **analyze_emotions** function which returns the predicted emotion along with its probability.
7. It saves the journal entry, including the encrypted journal text, mood, image (if provided), predicted emotion, emotion probability, and depression probability analysis, to the database.
8. If any exception occurs during this process, it returns an error response.
9. If the request method is not POST, it returns an error response.

```
@csrf_exempt
# This decorator exempts the view from CSRF protection, allowing POST requests without a CSRF token.
def save_journal(request):
    if request.method == 'POST':
        # Retrieve journal text and mood from the POST data
        journal_text = request.POST.get('journalText', '')
        mood = request.POST.get('mood', '')

        # Check if both text and mood are provided
        if not journal_text or not mood:
            return JsonResponse({'status': 'error', 'message': 'Both text and mood fields are required.'})

        try:
            # Retrieve optional image file
            image_file = request.FILES.get('image', None)

            # Encrypt the journal text
            encrypted_journal_text = encrypt_data(journal_text)

            # Analyze emotions and probabilities
            emotion_analysis = analyze_emotions(journal_text)
            predicted_emotion = emotion_analysis['prediction']
            probability = emotion_analysis['probability']
            depression= analyze_depression(journal_text)

            # Save the data to the database
            entry = JournalEntry(
                user=request.user,
                text=encrypted_journal_text,
                mood=mood,
```

```
                image=image_file,
                predicted_emotion=predicted_emotion,
                probability=probability,
                depression=depression,
            )
            entry.save()

            return JsonResponse({'status': 'success'})
        except Exception as e:
            # Return error response if any exception occurs
            return JsonResponse({'status': 'error', 'message': str(e)})
    else:
        # Return error response for invalid request method
        return JsonResponse({'status': 'error', 'message': 'Invalid request method'})
```

Figure 57: Save journal views.py

- **prevjournal** view displays a list of previous journal entries.

1. This function retrieves previous journal entries for the currently logged-in user.
2. It retrieves the user's journal entries ordered by their creation time in descending order.
3. It iterates over each entry, decrypts the encrypted text using a function called **decrypt_data**, and updates the entry with the decrypted text.
4. It prepares a context containing the decrypted journal entries for rendering.
5. Finally, it renders a template named "prevjournal.html" with the decrypted journal entries.

```
def prevjournal(request):
    # Assuming you have a ForeignKey relationship with User in your
JournalEntry model
    user = request.user
    journal_entries = JournalEntry.objects.filter(user=user).order_by('-
created_at')
    # Decrypt journal text for display
    for entry in journal_entries:
        # Debug statement to inspect the encrypted data
        #print("Encrypted data:", entry.text)

        # Decrypt the data and print the decrypted text
        decrypted_text = decrypt_data(entry.text)
        #print("Decrypted text:", decrypted_text)

        # Update the entry with the decrypted text
        entry.text = decrypted_text

    context = {'journal_entries': journal_entries}
    return render(request, "SoulGlow/prevjournal.html", context)
```

Figure 58: prev journal views.py

- **delete_journal_entry**

1. This function is responsible for deleting a specific journal entry.
2. It takes two parameters: **request**, which represents the HTTP request made to the server, and **entry_id**, which is the unique identifier of the journal entry to be deleted.
3. It attempts to retrieve the journal entry with the given **entry_id** from the database.
4. If the entry does not exist (raises **JournalEntry.DoesNotExist**).
5. If the entry exists, it deletes the entry from the database using the **delete()** method.
6. After deleting the entry, it redirects the user back to the previous journal (history) page using the **redirect()** function, which typically redirects the user to a specified URL pattern named **prevjournal**.

```
def delete_journal_entry(request, entry_id):
    # Retrieve the journal entry to delete
    try:
        journal_entry = JournalEntry.objects.get(id=entry_id)
    except JournalEntry.DoesNotExist:
        # Handle case where entry does not exist (optional)
        pass
    else:
        # Delete the journal entry
        journal_entry.delete()
    # Redirect back to the previous journal page
    return redirect('prevjournal')
```

Figure 59 :delete_journal_entry views.py

Emotion Analysis

- **analyze_emotions** function analyzes the journal text and predicts the emotion using a pre-trained model.

1. This function takes journal text as input and analyzes the emotions expressed in the text.
2. It starts by tokenizing the input text using a tokenizer.
3. Then, it makes predictions using a pre-trained model.
4. After obtaining the model outputs, it extracts relevant information such as logits and probabilities.
5. It calculates the softmax probabilities to convert logits into probabilities.
6. Finally, it returns the predicted emotion label along with the maximum probability as a percentage.

- softmax(x):

1. This function calculates the softmax values for the input scores to convert them into probabilities.
2. It first exponentiates the input scores and subtracts the maximum value to prevent numerical instability.
3. Then, it divides each exponentiated score by the sum of all exponentiated scores to normalize them.
4. The normalized scores represent probabilities and are returned as the output.

```
def analyze_emotions(journal_text):
    # Tokenize the input text
    inputs = tokenizer(journal_text, return_tensors="tf")

    # Make prediction using the model
    outputs = model(**inputs)

    # Extract relevant information from the outputs
    logits = outputs.logits.numpy()
    probabilities = softmax(logits)[0]
    predicted_class_id = int(tf.argmax(logits, axis=-1)[0])
    predicted_emotion_label = model.config.id2label[predicted_class_id]

    # Return the results with the maximum probability
    max_probability = np.max(probabilities) * 100  # Convert to percentage
    return {'prediction': predicted_emotion_label, 'probability': max_probability}

# function calculates the softmax values for the input scores to convert them into probabilities
def softmax(x):
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum()
```

Figure 60: softmax

Depression Analysis

- **analyze_depression** function analyzes the journal text for signs of depression using a pre-trained SVM model.

1. This function analyzes the level of depression indicated by a given text.
2. It preprocesses the input text using a function named **preprocess_text**.
3. After preprocessing, the text is vectorized using a TF-IDF vectorizer (**tfidf_vectorizer**), which converts the text into a numerical representation suitable for machine learning models.
4. The vectorized text is then passed through a Support Vector Machine (SVM) model (**svm_model**) to predict the probability of depression.
5. The probability of the depression class is extracted from the prediction probabilities returned by the SVM model.
6. Finally, the depression probability is converted to a percentage and returned.

```
def analyze_depression(text):
    # Preprocess text using the same function as in your Jupyter notebook
    preprocessed_input = preprocess_text(text)

    # Vectorize the preprocessed text
    input_vectorized = tfidf_vectorizer.transform([preprocessed_input]).toarray()

    # Use the loaded model for prediction
    prediction_prob = svm_model.predict_proba(input_vectorized)

    # Extract the probability of depression class
    depression_probability = prediction_prob[0][1] * 100  # Convert to percentage

    return depression_probability
```

Figure 61: analyze_depression

- **analyze_weekly_depression** analyzes depression for the current week's journal entries.

1. This function analyzes the level of depression based on a collection of journal entries from a user over a week.
2. It first decrypts the text of each journal entry using the **decrypt_data** function.
3. The decrypted text of all journal entries is then combined into a single text.
4. This combined text is preprocessed using the **preprocess_text** function.
5. After preprocessing, the text is vectorized using the same TF-IDF vectorizer as in the previous function.
6. The vectorized text is passed through the SVM model to predict the probability of depression and the class label.
7. The probability of the depression class is extracted from the prediction probabilities returned by the SVM model.
8. Additionally, it determines if the text indicates depression based on the predicted class label.
9. The function returns a dictionary containing the depression probability and a Boolean indicating whether the user is depressed or not.

```
def analyze_weekly_depression(user_entries):
    decrypted_entries = []

    # Decrypt the text of each journal entry
    for entry in user_entries:
        decrypted_entry_text = decrypt_data(entry.text)
        decrypted_entries.append(decrypted_entry_text)

    # Combine the decrypted text of all selected journals
    combined_text = '\n'.join(decrypted_entries)
    # Preprocess the combined text
    preprocessed_input = preprocess_text(combined_text)

    # Vectorize the preprocessed text
    input_vectorized = tfidf_vectorizer.transform([preprocessed_input]).toarray()

    # Use the loaded model for prediction
    prediction = svm_model.predict(input_vectorized)
    prediction_prob = svm_model.predict_proba(input_vectorized)

    # Extract the probability of depression class
    depression_probability = prediction_prob[0][1] * 100  # Convert to percentage

    # Determine if the text indicates depression
    is_depressed = prediction[0] == 1

    result = {
        'dep_probability': depression_probability,
        'is_depressed': is_depressed,
          }
    print(result)

    return result
```

Figure 62: analyze_Weekly_depression

Profile Page

- **profile** view displays the user's profile page, including journal entries, emotional and depression analysis including their charts as well.

1. This function represents a view for the user profile page.
2. It starts by fetching the user's journal entries from the database using **JournalEntry.objects.filter(user=request.user)**.
3. Then, it analyzes the entries for the current week and current month by filtering entries based on their creation dates.
4. If there are no journal entries for the current month, it prepares a message indicating the absence of journals.
5. If there are entries for the current week, it performs several tasks:
    - It analyzes the depression level of the user for the current week using the **analyze_weekly_depression** function.
    - It counts the number of journal entries for the current week.
    - It generates cumulative depression chart and an emotion chart based on the weekly entries.
    - It calculates the count of entries associated with the emotion 'sadness' for the current week.
6. Additionally, it generates an emotion chart for the current month based on the entries.
7. Finally, it prepares a context dictionary containing all the necessary data to render the profile page.
8. If depression is predicted for the current week, it sends a warning email to the user.
9. The function returns a rendered template for the user profile page (**'SoulGlow/profile.html'**) along with the context data.

```
def profile(request):
    # Fetch the user's journal entries
    user_entries = JournalEntry.objects.filter(user=request.user)

    # Analyze weekly entries
    current_date = get_current_time()
    start_of_week = current_date - timedelta(days=current_date.weekday())
    end_of_week = start_of_week + timedelta(days=6)
    current_week_entries = user_entries.filter(created_at__date__range=[start_of_week, end_of_week])

    # Calculate the start date of the current month
    start_of_month = current_date.replace(day=1)

    # Calculate the end date of the current month
    next_month = current_date.replace(day=28) + timedelta(days=4)
    end_of_month = next_month - timedelta(days=next_month.day)

    # Filter entries for the current month
    current_month_entries = user_entries.filter(created_at__date__range=[start_of_month,
end_of_month])

    if current_month_entries.count()==0:
        # No journals for the current week
        context = {
            'no_journals_message': "You haven't entered any journals this Month."
```

```
        }
        return render(request, 'SoulGlow/profile.html', context)

if current_week_entries.exists():
        weekly_dep = analyze_weekly_depression(current_week_entries)
        print("done with weekly_dep ")

        # Count the number of entries in current_week_entries
        journalnum = current_week_entries.count()
        print("done with journalnum")


            # depression_data = analyze_depression(current_week_entries)
        cumulative_chart_image = plot_depression_chart(current_week_entries)
        print("done with ccumulative_chart_image")


        # emotion_chart_data = analyze_emotions_for_week(current_week_entries)
        emotion_chart_image = plot_emotions_weekly_analysis(current_week_entries)
        print("done with emotion_chart_image")


    # Compute sadness count
        sadness_count = sum(1 for entry in current_week_entries if entry.predicted_emotion ==
'sadness')
        print("done with sadness_count")

    # Monthly Emotion Analysis
    emotion_monthly_chart_image = plot_emotions_monthly_analysis(current_month_entries)
    print("done with emotion_monthly_chart_image")

    context = {
        'current_week_entries': current_week_entries,
        'weekly_dep': weekly_dep,
        'cumulative_chart_image': cumulative_chart_image,
        'emotion_chart_image': emotion_chart_image,
        'sadness_count': sadness_count,
        'emotion_monthly_chart_image': emotion_monthly_chart_image,
        'journalnum': journalnum,
    }

    # Check if depression is predicted and send a warning message
    if weekly_dep['is_depressed']:
        subject = "Concern About Your Wellbeing"
        message = (
            f"Hello {request.user.first_name},\n\n"
            "We've noticed some concerning signs in your recent journals. "
            "It seems like you might be experiencing depression. "
            "We want you to know that you're not alone, and support is available. "
            "We encourage you to seek help and talk to someone you trust.\n\n"
            "Best regards,\n"
            "The SoulGlow Team"
        )
        send_mail(subject, message, conf_settings.EMAIL_HOST_USER, [request.user.email])

    return render(request, 'SoulGlow/profile.html', context)
```

Figure 63: profile

Encryption and Decryption

- The `**encrypt_data**` and `**decrypt_data**` functions encrypt and decrypt journal text for security. `encrypt_data` takes a string, encrypts it using the Fernet class from the `cryptography` library, and returns the encrypted byte array. The encryption key is derived from `**conf_settings.SECRET_KEY**` in Django settings. `decrypt_data` reverses this process, decrypting the input string and returning the decrypted text. These functions are vital for securing sensitive data in Django applications, such as user passwords or confidential information stored in databases, enhancing security and preventing unauthorized access.

```python
import os
from cryptography.fernet import Fernet
from django.conf import settings as conf_settings

# Create an instance of the Fernet cipher
cipher_suite = Fernet(conf_settings.SECRET_KEY)

# Function to encrypt data
def encrypt_data(data):
    encrypted_data = cipher_suite.encrypt(data.encode())
    return encrypted_data

# Function to decrypt data
def decrypt_data(encrypted_data):
    decrypted_data = cipher_suite.decrypt(encrypted_data).decode()
    return decrypted_data
```

Figure 64 : Encryption and Decryption

Data Visualization (Graphs)

- **plot_depression_chart** function generates a scatter plot to visualize depression analysis over the current week. It extracts dates and corresponding depression probabilities from journal entries, creates a scatter plot using Plotly, customizes layout settings like the title, axis labels, tick angle, tick format, and title alignment, then converts the plot to a base64-encoded image to return it as a string, which can be embedded in a web page or rendered in an application

```
#generates a scatter plot to visualize the depression analysis over the week

def plot_depression_chart(current_week_entries):

    # Extract dates and corresponding depression probabilities from journal entries

    dates = [entry.created_at.date() for entry in current_week_entries if entry.depression is not None]

    probabilities = [entry.depression for entry in current_week_entries if entry.depression is not None]

    # Print the extracted dates for debugging purposes

    print("\nDepression analysis dates: \n", dates)

    # Create a scatter plot of depression probabilities over dates

    fig = go.Figure()

    fig.add_trace(go.Scatter(x=dates, y=probabilities, mode='lines+markers'))

    # Update layout with chart title and axis labels

    fig.update_layout(

        title='Depression Analysis Over The Week',

        xaxis_title='Date',

        yaxis_title='Probability of Depression',

        xaxis_tickangle=-45,

        xaxis_tickformat='%Y-%m-%d',

        title_x=0.5  # Center alignment for the title

    )

 # Set the x-axis tickvals and ticktext to display only the dates in the 'dates' variable

    fig.update_xaxes(tickvals=dates, ticktext=[date.strftime('%Y-%m-%d') for date in dates])

   # Convert the plot to a base64-encoded image

    image_stream = BytesIO()
```

```
    pio.write_image(fig, image_stream, format='png')

    image_stream.seek(0)

    image_base64 = base64.b64encode(image_stream.getvalue()).decode('utf-8')

    return image_base64
```

Figure 65: plot_depression_chart

- **plot_emotions_weekly_analysis** and **plot_emotions_monthly_analysis** functions generate bar and pie charts to depict emotion distribution over the week and month, respectively. They count emotion occurrences, create the charts using Plotly, customize layout settings, and encode the plots as base64 strings for embedding. Additionally, colors are defined for the emotions in the charts: anger (#A1CCD1), joy (#C8E4B2), surprise (#FDFFAE), sadness (#EF9595), love (#FFC0D9), and fear (#E5E0FF).

```
#generates a bar chart to visualize the distribution of emotions detected in the entries over a weekly period
def plot_emotions_weekly_analysis(journal_entries):
    # Define emotions and colors
    emotions = ["anger", "joy", "surprise", "sadness", "love", "fear"]
    colors = ['#A1CCD1', '#C8E4B2', '#FDFFAE', '#EF9595', '#FFC0D9', '#E5E0FF']

    # Count occurrences of each emotion for each date
    date_counts = {}
    for entry in journal_entries:
        predicted_emotion_label = entry.predicted_emotion
        date = entry.created_at.date()

        if date in date_counts:
            date_counts[date][predicted_emotion_label] = date_counts[date].get(predicted_emotion_label, 0) + 1
        else:
            date_counts[date] = {emotion: 0 for emotion in emotions}
            date_counts[date][predicted_emotion_label] = 1

    # Sort dates
    dates = sorted(date_counts.keys())

    # Count occurrences of each emotion for each date
    counts = {emotion: [date_counts[date].get(emotion, 0) for date in dates] for emotion in emotions}

    # Create a bar chart of emotion distribution over dates
    fig = go.Figure()
    bar_width = 0.1
    index = np.arange(len(dates))

    for i, emotion in enumerate(emotions):
        fig.add_trace(go.Bar(x=index + (i - 2) * bar_width, y=counts[emotion], name=emotion, marker_color=colors[i]))

    # Update layout with chart title and axis labels
    fig.update_layout(
        xaxis=dict(tickmode='array', tickvals=index, ticktext=dates),
        xaxis_tickangle=-45,
        title='Emotion Distribution',
        xaxis_title='Date',
        yaxis=dict(tickvals=list(range(max(max(counts.values())) + 1))),
        legend_title='Emotion',
        title_x=0.5
    )

    # Convert the plot to a base64-encoded image
    image_stream = BytesIO()
    pio.write_image(fig, image_stream, format='png')
    image_stream.seek(0)
    image_base64 = base64.b64encode(image_stream.getvalue()).decode('utf-8')

    return image_base64
```

Figure 66: plot_emotions_weekly_analysis

```python
# generates a pie chart to visualize the distribution of emotions detected in the entries over a monthly period
def plot_emotions_monthly_analysis(journal_entries):
    # Define emotions and colors
    emotions = ["anger", "joy", "surprise", "sadness", "love", "fear"]
    colors = ['#A1CCD1', '#C8E4B2', '#FDFFAE', '#EF9595', '#FFC0D9', '#E5E0FF']

    # Count occurrences of each emotion for each date
    date_counts = {}
    for entry in journal_entries:
        predicted_emotion_label = entry.predicted_emotion
        date = entry.created_at.date()

        if date in date_counts:
            date_counts[date][predicted_emotion_label] = date_counts[date].get(predicted_emotion_label, 0) + 1
        else:
            date_counts[date] = {emotion: 0 for emotion in emotions}
            date_counts[date][predicted_emotion_label] = 1

    # Count total occurrences of each emotion
    counts = {emotion: sum(date_counts[date].get(emotion, 0) for date in date_counts.keys()) for emotion in emotions}

    # Filter out emotions with zero occurrences
    emotions_detected = [emotion for emotion in emotions if counts[emotion] > 0]
    counts_detected = {emotion: counts[emotion] for emotion in emotions_detected}

    # Create a pie chart of emotion distribution
    fig = go.Figure(data=[go.Pie(labels=emotions_detected, values=[counts_detected[emotion] for emotion in emotions_detected])])
    fig.update_traces(marker=dict(colors=colors), pull=[0.1, 0.1, 0.1, 0.1, 0.1, 0.1], textinfo='percent+label')

    # Update layout with chart title
    fig.update_layout(
        title='Emotion Distribution',
        title_x=0.5
    )

    # Convert the plot to a base64-encoded image
    image_stream = BytesIO()
    pio.write_image(fig, image_stream, format='png')
    image_stream.seek(0)
    image_base64 = base64.b64encode(image_stream.getvalue()).decode('utf-8')

    return image_base64
```

Figure 67: plot_emotions_monthly_analysis

models.py

JournalEntry represents a journal entry that a user can create and store in the database.
First import necessary modules including models from Django and Fernet from the cryptography library. These modules provide the required functionalities for defining the model and implementing encryption.
The `JournalEntry` model represents a journal entry in the database. It includes the following fields:

- `user`: A foreign key to the `User` model, indicating the user who created the entry.
- `text`: Binary field for storing the main content or text of the entry after encryption.
- `mood`: char field for representing the associated mood.
- `image`: Optional field for storing an image related to the entry.
- `created_at`: DateTime field automatically set to the entry's creation timestamp.
- `predicted_emotion`: char field for representing the predicted emotion associated with the entry.
- `probability`: float field for storing the probability value of the predicted emotion.
- `depression`: float field for storing the probability related to depression.

```
class JournalEntry(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    text = models.BinaryField()
    mood = models.CharField(max_length=50, null=True, blank=True)
    image = models.ImageField(upload_to='journal_images/', null=True, blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
    predicted_emotion = models.CharField(max_length=50, null=True, blank=True)
    probability = models.FloatField(null=True, blank=True)
    depression = models.FloatField(null=True, blank=True)
```

Figure 68: `JournalEntry` model

- __str__ method: This method defines how the JournalEntry object should be represented as a string. In this case, it returns a string that includes the username of the user who created the journal entry.

```
    def __str__(self):
        return f"{self.user.username}'s Journal Entry"
```

Figure 69:__str__ method

Admin.py

in the `admin.py` , the code creates a customized admin interface for the User model in Django. It introduces an "inline model" feature for the JournalEntry model, which allows you to view JournalEntry entries within the User admin interface, to provide a seamless way to manage user-related journal entries without navigating to a separate interface.

1. This class defines the inline view for the JournalEntry model. It allows you to view JournalEntry instances within the User admin page.

```
class JournalEntryInline(admin.TabularInline):
    model = JournalEntry
    extra = 0  # Remove the ability to add new JournalEntry instances
    list_display = ('text', 'mood', 'predicted_emotion', 'probability', 'depression', 'display_id', 'created_at')
    readonly_fields = ('text', 'mood', 'predicted_emotion', 'probability', 'depression', 'display_id' , 'created_at')  #   , 'created_at')  Make fields read-only
    can_delete = False  # Remove the ability to delete JournalEntry instances

    def display_id(self, instance):
        return f"Journal ID: {instance.id}"

    def created_at(self, instance):
        return instance.created_at.strftime('%Y-%m-%d %H:%M:%S')  # Format the date as desired

    display_id.short_description = 'Journal ID'
    created_at.short_description = 'Created At'
    # Remove image field from the inline
    exclude = ('image',)

    def has_change_permission(self, request, obj=None):
         return False # Remove the ability to change existing user journals

    def has_add_permission(self, request, obj=None):
        return False  # Remove the ability to add new user journals
```

Figure 70: Defining the JournalEntryInline class

This code snippet customizes the Django admin interface by incorporating an inline view for JournalEntry within the User admin page using CustomUserAdmin. It unregisters the default UserAdmin and registers the custom UserAdmin, also, it sets the admin site header to 'SoulGlow' for branding and removes the default URL from the site header to disallow visiting the website.

```
class CustomUserAdmin(UserAdmin):
    inlines = [JournalEntryInline]

# Unregister the default UserAdmin
admin.site.unregister(User)

# Register the custom UserAdmin
admin.site.register(User, CustomUserAdmin)

# Customize the admin site header with your logo
admin.site.site_header = 'SoulGlow'

admin.site.site_url= None
```

Figure 71: Defining the CustomUserAdmin class

settings.py

The Django settings file snippet configures email settings for sending emails via SMTP. We enable TLS for secure communication with Gmail's SMTP server (`smtp.gmail.com` on port `587`). Authentication details, including the email address and application password, are provided.

Additionally, we import the `Fernet` class for cryptographic key generation and retrieval. The `SECRET_KEY` setting retrieves the secret key value from the 'SECRET_KEY' environment variable using `os.environ.get('SECRET_KEY') `.

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_USE_TLS = True
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_HOST_USER = 'soulglow.website@gmail.com'  # Your Gmail address
EMAIL_HOST_PASSWORD =  'zzrq hagv aeby agxh' # Your Gmail password
from cryptography.fernet import Fernet
# # Generate a key
# key = Fernet.generate_key()
# # Convert the key to a string for storage or transmission
# key_string = key.decode()
# # Print or store the key securely
# print("Generated Key:", key_string)
import os
# Secret key settings
SECRET_KEY = os.environ.get('SECRET_KEY')
```

Figure 72: Settings.py

Utils.py

Preprocessing Text:

The preprocess_text function takes a text string as input and performs the following preprocessing steps: converting the text to lowercase, removing non-alphabetic characters and symbols, expanding contractions, performing part-of-speech (POS) tagging, lemmatizing tokens to their base form, removing stop words, and eliminating duplicate words. Finally, it returns the preprocessed text.

```
def preprocess_text(text):
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer()

    # Convert text to lowercase
    text = text.lower()

    # Remove characters and symbols
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    contractions = {
        # ... (your contractions dictionary)
    }

    # Replace contractions
    for contraction, expanded_form in contractions.items():
        text = text.replace(contraction, expanded_form)

    # Tokenize the text into words using wordninja
    tokens = wordninja.split(text)

    # Perform POS tagging
    pos_tags = pos_tag(tokens)

    # Lemmatization
    lemmatized_tokens = []
    for word, tag in pos_tags:
        pos = get_wordnet_pos(tag)
        lemma = lemmatizer.lemmatize(word, pos=pos)
        lemmatized_tokens.append(lemma)

    # Remove stop words
    filtered_tokens = [word for word in lemmatized_tokens if word.lower() not in stop_words]

    # Remove duplicate words
    unique_tokens = []
    for word in filtered_tokens:
        if word not in unique_tokens:
            unique_tokens.append(word)

    # Recreate text from unique tokens
    filtered_text = ' '.join(unique_tokens)

    return filtered_text

import joblib
from sklearn.svm import SVC
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import make_pipeline

# Load the SVM model
svm_model_file = open("SoulGlow\\depression_svm_model.pkl", "rb")
svm = joblib.load(svm_model_file)
svm_model_file.close()

# Load the TF-IDF vectorizer
tfidf_vectorizer_file = open("SoulGlow\\depression_tfidf_vectorizer.pkl", "rb")
tfidf_vectorizer = joblib.load(tfidf_vectorizer_file)
tfidf_vectorizer_file.close()
```

Figure 73: Preprocessing Text

Predicting Depression

The predict_depression function takes a text string as input and performs the following steps: preprocessing the text using the preprocess_text function, vectorizing it using TF-IDF, predicting depression status (0 for no depression, 1 for depression) using SVM model, and calculating the probability of depression. It returns a tuple with the predicted depression status and its probability.

```
def predict_depression(input_text):
    preprocessed_input = preprocess_text(input_text)
    input_vectorized = tfidf_vectorizer.transform([preprocessed_input]).toarray()
    prediction = svm.predict(input_vectorized)
    probability = svm.predict_proba(input_vectorized)

    if prediction[0] == 1:
        result = "The input text indicates depression."
    else:
        result = "The input text does not indicate depression."

    return result, probability
```

Figure 74: Predicting Depression

5.2.3 Machine Learning Models

Here, we will examine the data preprocessing steps and the accuracy of machine learning algorithms as we discussed in Chapter 2 regarding data preparation and model training for Depression Detection model and emotion detection model.

Depression Detection Model

Data pre-processing:

In [8]:
```
from nltk import pos_tag
nltk.download('averaged_perceptron_tagger')

def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return 'a'  # Adjective
    elif tag.startswith('V'):
        return 'v'  # Verb
    elif tag.startswith('N'):
        return 'n'  # Noun
    elif tag.startswith('R'):
        return 'r'  # Adverb
    else:
        return 'n'  # Default to noun if not found
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /Users/iremhatipoglu/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

In [10]:
```
nltk.download('wordnet')

def preprocess_text(text):
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer()

    # Convert text to lowercase
    text = text.lower()

    # Remove characters and symbols
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    contractions = {
        "aint": "is not", "arent": "are not","cant": "can not",
        "cause": "because", "couldve": "could have", "couldnt": "could not",
        "didnt": "did not",  "doesnt": "does not", "dont": "do not", "hadnt": "had not",
        "hasnt": "has not", "havent": "have not", "hed": "he would", "hes": "he is",
        "howd": "how did", "howll": "how will", "hows": "how is",  "I'd": "I would",
        "I'd've": "I would have", "Ill": "I will",  "Ill've": "I will have","Im": "I am", "I've": "I have",
        "i'd": "i would", "i'd've": "i would have", "i'll": "i will", "im": "i am" , "i'll've": "i will have","i'm": "i am",
        "i've": "i have", "isnt": "is not", "it'd": "it would", "it'd've": "it would have", "it'll": "it will",
        "it'll've": "it will have","it's": "it is", "let's": "let us", "ma'am": "madam", "mayn't": "may not",
        "might've": "might have","mightn't": "might not","mightn't've": "might not have", "must've": "must have",
        "mustn't": "must not", "mustn't've": "must not have", "needn't": "need not", "needn't've": "need not have",
        "o'clock": "of the clock", "oughtn't": "ought not", "oughtn't've": "ought not have", "shan't": "shall not",
        "sha'n't": "shall not", "shan't've": "shall not have", "she'd": "she would", "she'd've": "she would have",
        "she'll": "she will", "she'll've": "she will have", "she's": "she is", "should've": "should have",
        "shouldn't": "should not", "shouldn't've": "should not have", "so've": "so have","so's": "so as",
        "this's": "this is","that'd": "that would", "that'd've": "that would have", "that's": "that is",
        "there'd": "there would", "there'd've": "there would have", "there's": "there is", "here's": "here is",
        "they'd": "they would", "they'd've": "they would have", "they'll": "they will", "they'll've": "they will have",
        "they're": "they are", "they've": "they have", "to've": "to have", "wasn't": "was not", "we'd": "we would",
        "we'd've": "we would have", "we'll": "we will", "we'll've": "we will have", "we're": "we are", "we've": "we have",
        "weren't": "were not", "what'll": "what will", "what'll've": "what will have", "what're": "what are",  "what's": "what
        "what've": "what have", "when's": "when is", "when've": "when have", "where'd": "where did", "where's": "where is",
        "where've": "where have", "who'll": "who will", "who'll've": "who will have", "who's": "who is", "who've": "who have",
        "why's": "why is", "why've": "why have", "will've": "will have", "won't": "will not", "won't've": "will not have",
        "would've": "would have", "wouldn't": "would not", "wouldn't've": "would not have", "y'all": "you all",
        "y'all'd": "you all would","y'all'd've": "you all would have","y'all're": "you all are","y'all've": "you all have",
        "you'd": "you would", "you'd've": "you would have", "you'll": "you will", "you'll've": "you will have",
        "you're": "you are", "you've": "you have"

    }
```

```
    # Replace contractions
    for contraction, expanded_form in contractions.items():
        text = text.replace(contraction, expanded_form)

    # Tokenize the text into words using wordninja
    tokens = wordninja.split(text)

    # Perform POS tagging
    pos_tags = pos_tag(tokens)

    # Lemmatization
    lemmatized_tokens = []
    for word, tag in pos_tags:
        pos = get_wordnet_pos(tag)
        lemma = lemmatizer.lemmatize(word, pos=pos)
        lemmatized_tokens.append(lemma)

    # Remove stop words
    filtered_tokens = [word for word in lemmatized_tokens if word.lower() not in stop_words]

     # Remove duplicate words
    unique_tokens = []
    for word in filtered_tokens:
        if word not in unique_tokens:
            unique_tokens.append(word)

    # Recreate text from unique tokens
    filtered_text = ' '.join(unique_tokens)

    return filtered_text
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/iremhatipoglu/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [11]:

```
# Apply the preprocess_text function to the cleaned_text column
mental_data['processed_text'] = mental_data['text'].apply(preprocess_text)

mental_data
```

Out[11]:

	text	label	processed_text
0	dear american teens question dutch person hear...	0	dear american teen question dutch person hear ...
1	nothing look forward lifei dont many reasons k...	1	nothing look forward life many reason keep go ...
2	music recommendations im looking expand playli...	0	music recommendation look expand playlist usua...
3	im done trying feel betterthe reason im still ...	1	try feel better reason still alive know mum de...
4	worried year old girl subject domestic physic...	1	worried year old girl subject domestic physica...
...	...	...	...
27972	posting everyday people stop caring religion ...	0	post everyday people stop care religion matter...
27973	okay definetly need hear guys opinion ive pret...	0	okay definetly need hear guys opinion ive pret...
27974	cant get dog think ill kill myselfthe last thi...	1	get dog think ill kill last thing hold yup fir...
27975	whats point princess bridei really think like ...	1	point princess bride really think like wesley ...
27976	got nudes person might might know snapchat do ...	0	get nude person might know snap chat ok chick ...

27977 rows × 3 columns

Figure 75: Data pre-processing

Training the model:

In [16]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from scikitplot.metrics import plot_confusion_matrix, plot_roc
X = mental_data["processed_text"]
y = mental_data['label'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2, random_state= 42, stratify = y)
```

In [17]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(max_features= 2500, min_df= 2)
X_train = tfidf.fit_transform(X_train).toarray()
X_test = tfidf.transform(X_test).toarray()
```

In [18]:
```
X_test
```

Out[18]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [19]:
```
def train_model(model):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)
    accuracy = round(accuracy_score(y_test, y_pred), 3)
    precision = round(precision_score(y_test, y_pred), 3)
    recall = round(recall_score(y_test, y_pred), 3)

    print(f'Accuracy of the model: {accuracy}')
    print(f'Precision Score of the model: {precision}')
    print(f'Recall Score of the model: {recall}')

    sns.set_context('notebook', font_scale= 1.3)
    fig, ax = plt.subplots(1, 2, figsize = (25,  8))
    ax1 = plot_confusion_matrix(y_test, y_pred, ax= ax[0], cmap='Blues')
    ax2 = plot_roc(y_test, y_prob, ax= ax[1], plot_macro= False, plot_micro= False, cmap='summer')
```

Figure 76: Training the model

We will start with training our dataset with the 5 Machine learning algorithms and compare the results.

Logistic Regression:

```
In [20]: from sklearn.linear_model import LogisticRegression

         logistic_reg = LogisticRegression()
         train_model(logistic_reg)
```

```
Accuracy of the model: 0.917
Precision Score of the model: 0.923
Recall Score of the model: 0.909
```

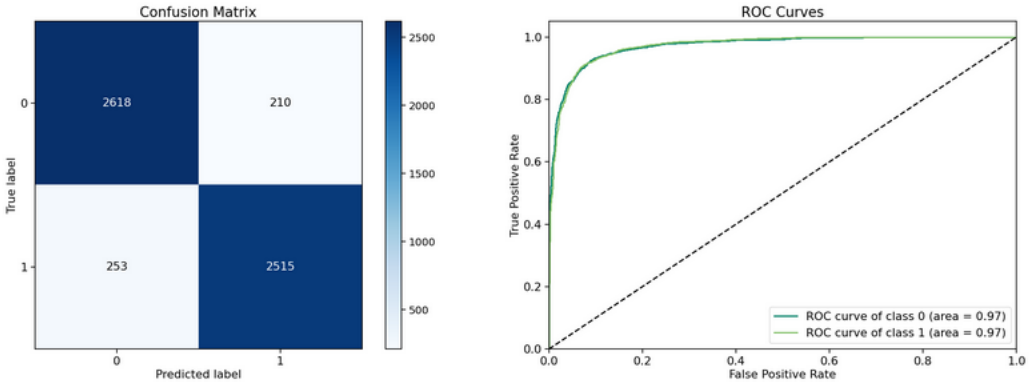


Figure 77: Logistic Regression model

Naïve Bayes:

```
In [21]: from sklearn.naive_bayes import MultinomialNB

         nb = MultinomialNB()
         train_model(nb)
```

```
Accuracy of the model: 0.889
Precision Score of the model: 0.847
Recall Score of the model: 0.947
```

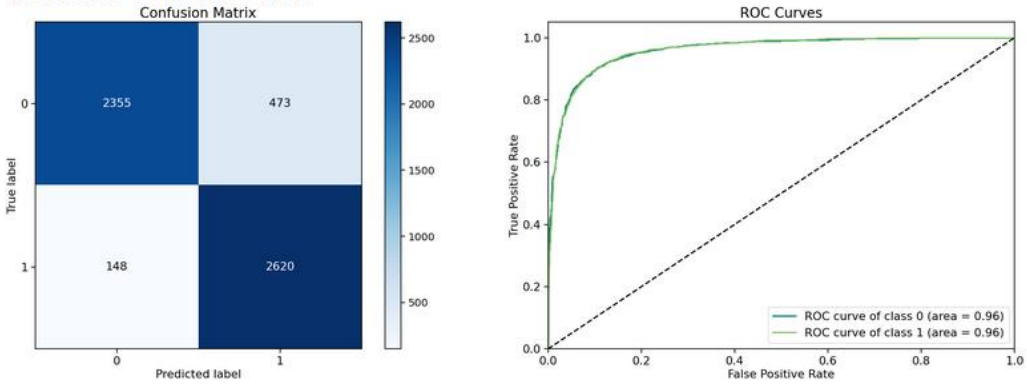


Figure 78: Naïve Bayes model

Decision Tree:

```
In [22]: from sklearn.tree import DecisionTreeClassifier

         decision_tree = DecisionTreeClassifier()
         train_model(decision_tree)
```

```
Accuracy of the model: 0.83
Precision Score of the model: 0.831
Recall Score of the model: 0.824
```

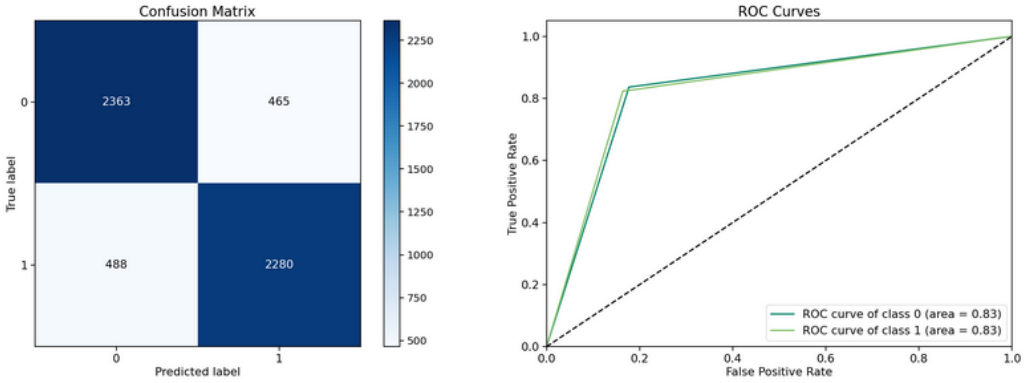


Figure 79: Decision Tree model

Random Forest:

```
In [23]: from sklearn.ensemble import RandomForestClassifier

         rf = RandomForestClassifier(n_estimators= 300)
         train_model(rf)
```

```
Accuracy of the model: 0.893
Precision Score of the model: 0.882
Recall Score of the model: 0.905
```

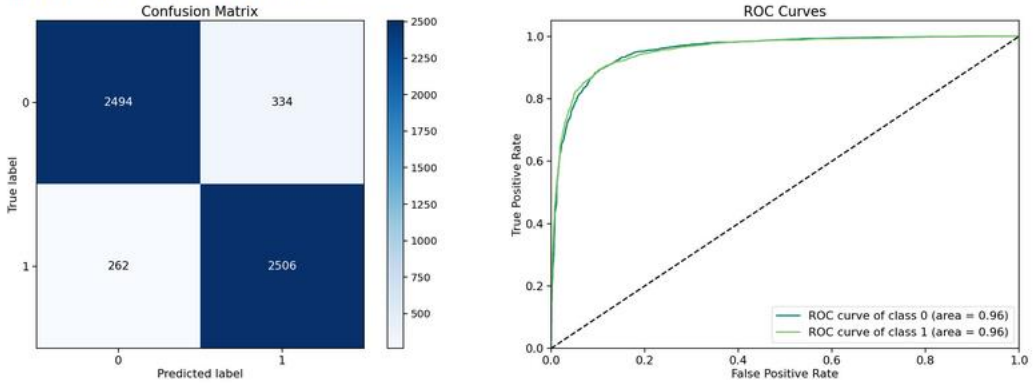


Figure 80: Random Forest model

Support Vector Machine (SVM):

```
In [24]:
from sklearn.svm import SVC

svm = SVC(probability=True)  # Setting probability=True for using predict_proba
train_model(svm)
```

```
Accuracy of the model: 0.92
Precision Score of the model: 0.926
Recall Score of the model: 0.911
```

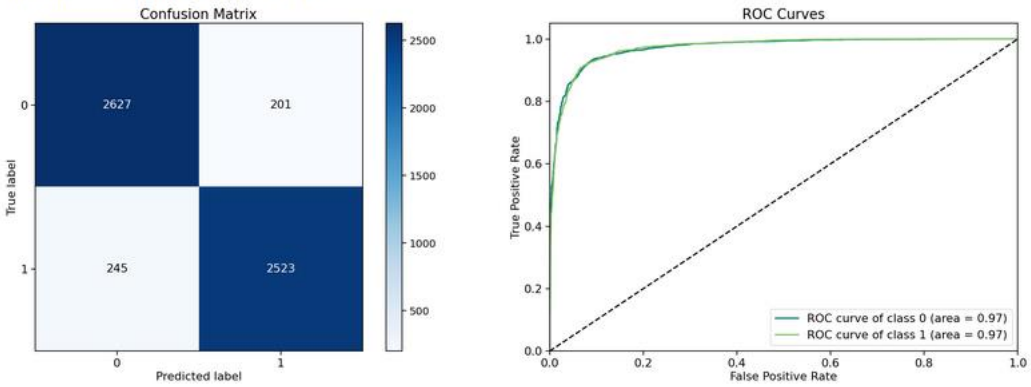


Figure 81: Support Vector Machine (SVM) model

Results:

According to the accuracy results of the training model, Support Vector Machine Algorithm (SVM) is the most effective for detecting depression. The figure below shows the comparison between accuracy scores for the algorithms: LR, NB, DT, RF, and SVM.

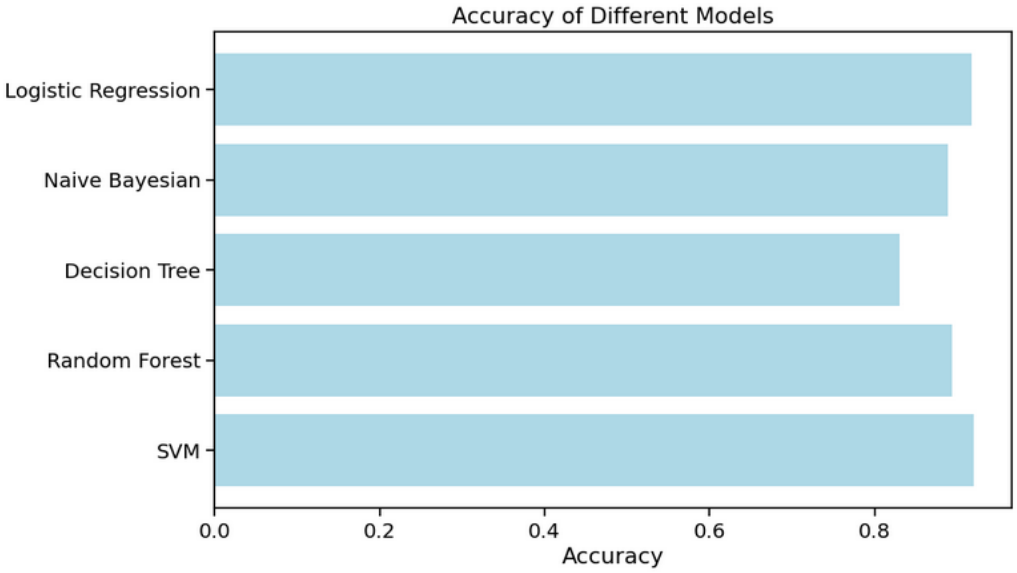


Figure 82: Results model

Logistic Regression	Use logistic regression to estimate the probability that an event will occur as a function of other variables.[12]
Accuracy	The accuracy of the model is 0.917, which means that the model correctly classified approximately 91.7% of the instances in the dataset.
Precision Accuracy	The precision score of the model is 0.923, which means the model predicted a positive outcome, it was correct about 92.3% of the time.
Recall Score	The recall score of the model is 0.909, which means that the model predicted about 90.9% of all actual positive instances in the dataset.
Confusion Matrix	A 2x2 matrix displaying the number of true positives (2500), false positives (210), true negatives (2515), and false negatives (253).
ROC Curves	A good model has a high TPR with a low FPR, thus the area under the curve (AUC) is 0.97.

Table 42: Performance evaluation of logistic regression

Naïve Bayes	The Naïve Bayes classifier is a probabilistic classifier based on Bayes' Law and naïve conditional independence assumptions.[12]
Accuracy	The accuracy of the model is 0.889, which means that the model correctly classified approximately 88.9% of the instances in the dataset.
Precision Accuracy	The precision score of the model is 0.847, which means the model predicted a positive outcome, it was correct about 84.7% of the time.
Recall Score	The recall score of the model is 0.947, which means that the model predicted about 94.7% of all actual positive instances in the dataset.
Confusion Matrix	A 2x2 matrix displaying the number of true positives (2355), false positives (148), true negatives (2620), and false negatives (473).
ROC Curves	A good model has a high TPR with a low FPR, thus the area under the curve (AUC) is 0.96.

Table 43: Performance evaluation of Naïve Bayes

Decision Tree	Decision Trees are a flexible method commonly deployed in data mining applications.[12]
Accuracy	The accuracy of the model is 0.83, which means that the model correctly classified approximately 83% of the instances in the dataset.
Precision Accuracy	The precision score of the model is 0.831, which means the model predicted a positive outcome, it was correct about 83.1% of the time.
Recall Score	The recall score of the model is 0.824, which means that the model predicted about 82.4% of all actual positive instances in the dataset.
Confusion Matrix	A 2x2 matrix displaying the number of true positives (2363), false positives (465), true negatives (2280), and false negatives (187).
ROC Curves	A good model has a high TPR with a low FPR, thus the area under the curve (AUC) is 0.83.

Table 44: Performance evaluation of decision tree

Random Forest	A random forest is a set of decision trees built on random samples with a different policy for splitting a node, trying to find the threshold that best separates the data.[35]
Accuracy	The accuracy of the model is 0.893, which means that the model correctly classified approximately 89.3% of the instances in the dataset.
Precision Accuracy	The precision score of the model is 0.882, which means the model predicted a positive outcome, it was correct about 88.2% of the time.
Recall Score	The recall score of the model is 0.905 means that the model

	predicted about 90.5% of all actual positive instances in the dataset.
Confusion Matrix	A 2x2 matrix displaying the number of true positives (2494), false positives (334), true negatives (2280), and false negatives (187).
ROC Curves	A good model has a high TPR with a low FPR, thus the area under the curve (AUC) is 0.96.

Table 45: Performance evaluation of Random Forest

Support Vector Machine (SVM)	Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression.[36]
Accuracy	The accuracy of the model is 0.92 means that the model correctly classified approximately 92% of the instances in the dataset.
Precision Accuracy	The precision score of the model is 0.926 means the model predicted a positive outcome, it was correct about 92.6% of the time.
Recall Score	The recall score of the model is 0.911, which means that the model predicted about 91.1% of all actual positive instances in the dataset.
Confusion Matrix	A 2x2 matrix displaying the number of true positives (2494), false positives (201), true negatives (2280), and false negatives (187).
ROC Curves	A good model has a high TPR with a low FPR, thus the area under the curve (AUC) is 0.97.

Table 46: Performance evaluation of SVM

Emotion Detection Model

In emotion detection we have fine-tuned five different models **“microsoft/deberta-v3-xsmall”**, **“xlnet/xlnet-base-cased”**, **“google-bert/bert-base-uncased”, “xlm-roberta-base”** and **“distilreberta-base”** on a dataset called **"AdamCodd/emotion-balanced"**.

According to the accuracy results of the training model, **“google-bert/bert-base-uncased”** are the most effective for emotion detection with Accuracy = 0.945.

fine-tuning script:
Installing the Python packages transformers and datasets and evaluate then loads a dataset named "AdamCodd/emotion-balanced"

```
[3] from datasets import load_dataset

    dataset = load_dataset("AdamCodd/emotion-balanced")
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
Downloading readme: 100% 9.06k/9.06k [00:00<00:00, 388kB/s]
Downloading data: 100% 1.97M/1.97M [00:00<00:00, 6.96MB/s]
Downloading data: 100% 248k/248k [00:00<00:00, 1.32MB/s]
Downloading data: 100% 244k/244k [00:00<00:00, 1.70MB/s]
Generating train split: 16000/0 [00:00<00:00, 123862.80 examples/s]
Generating validation split: 2000/0 [00:00<00:00, 23151.46 examples/s]
Generating test split: 2000/0 [00:00<00:00, 23926.37 examples/s]
```

```
[4] dataset["test"][0]
```

```
{'text': 'i feel enraged by the amount of people participating for the chance to break things or those who treat it as a tourist event',
 'label': 3}
```

Figure 83: load datasets

using the AutoTokenizer.from_pretrained method from the transformers library to initialize a tokenizer for the model we chose.

```
[5] from transformers import AutoTokenizer

    tokenizer = AutoTokenizer.from_pretrained("xlnet/xlnet-base-cased")
```

```
config.json: 100% 760/760 [00:00<00:00, 24.6kB/s]
spiece.model: 100% 798k/798k [00:00<00:00, 4.07MB/s]
tokenizer.json: 100% 1.38M/1.38M [00:00<00:00, 6.89MB/s]
```

Figure 84: AutoTokenizer

defines a function named preprocess_function that takes a dictionary of examples as input and preprocesses the text data using a tokenizer.

```
[6] def preprocess_function(examples):
        return tokenizer(examples["text"], truncation=True)

[7] tokenized_data = dataset.map(preprocess_function, batched=True)
```

```
Map: 100% 16000/16000 [00:04<00:00, 3872.93 examples/s]
Asking to truncate to max_length but no maximum length is provided and the model has no predefined maximum length. Default to no truncation
Map: 100% 2000/2000 [00:00<00:00, 4327.70 examples/s]
Map: 100% 2000/2000 [00:00<00:00, 3240.46 examples/s]
```

Figure 85: preprocess_function

using the DataCollatorWithPadding class from the transformers library to create a data collecter.

```
[8] from transformers import DataCollatorWithPadding

    data_collator = DataCollatorWithPadding(tokenizer=tokenizer, return_tensors="tf")
```

Figure 86: DataCollatorWithPadding

importing the evaluate library and then a function named load is being called from it with the argument "accuracy".

```
[10] import evaluate

     accuracy = evaluate.load("accuracy")

     Downloading builder script: 100%  4.20k/4.20k [00:00<00:00, 293kB/s]
```

Figure 87: evaluate

importing the NumPy library with the alias np, then defines a function named compute_metrics that calculates evaluation metrics based on predictions and labels, it will return the accuracy, precision, recall and F1.

```
[11] import numpy as np
     from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score



     def compute_metrics(eval_pred):
         predictions, labels = eval_pred
         predictions = np.argmax(predictions, axis=1)

         accuracy = accuracy_score(labels, predictions)
         precision = precision_score(labels, predictions, average='weighted')
         recall = recall_score(labels, predictions, average='weighted')
         f1 = f1_score(labels, predictions, average='weighted')

         return {
             'accuracy': accuracy,
             'precision': precision,
             'recall': recall,
             'f1': f1
         }
```

Figure 88: compute_metrics

Here we set up an optimizer for training a TensorFlow model using the Hugging Face Transformers library.

```
[12] from transformers import create_optimizer
     import tensorflow as tf

     batch_size = 16
     num_epochs = 5
     batches_per_epoch = len(tokenized_data["train"]) // batch_size
     total_train_steps = int(batches_per_epoch * num_epochs)
     optimizer, schedule = create_optimizer(init_lr=2e-5, num_warmup_steps=0, num_train_steps=total_train_steps)
```

Figure 89: TensorFlow

imports the TFAutoModelForSequenceClassification class from the transformer's library. This class is used to create a sequence classification model that can be fine-tuned on specific tasks.

```
[13] from transformers import TFAutoModelForSequenceClassification

     label2int = {
         "sadness": 0,
         "joy": 1,
         "love": 2,
         "anger": 3,
         "fear": 4,
         "surprise": 5
     }

     # Create id2label dictionary by reversing key-value pairs of label2int
     id2label = {v: k for k, v in label2int.items()}

     # Create label2id dictionary from label2int
     label2id = label2int

     model = TFAutoModelForSequenceClassification.from_pretrained(
         "xlnet/xlnet-base-cased", num_labels=6, id2label=id2label, label2id=label2id
     )
```

Figure 90: TFAutoModelForSequenceClassification

prepares TensorFlow datasets for training and validation and testing using a pre-trained sequence classification model and compiles the TensorFlow model for training using the specified optimizer.

```
[14] tf_train_set = model.prepare_tf_dataset(
         tokenized_data["train"],
         shuffle=True,
         batch_size=16,
         collate_fn=data_collator,
     )

     tf_validation_set = model.prepare_tf_dataset(
         tokenized_data["validation"],
         shuffle=False,
         batch_size=16,
         collate_fn=data_collator,
     )
     tf_test_set = model.prepare_tf_dataset(
         tokenized_data["test"],
         shuffle=False,
         batch_size=16,
         collate_fn=data_collator,
     )

[15] import tensorflow as tf

     model.compile(optimizer=optimizer, metrics='accuracy')  # No loss argument!
```

Figure 91: compiles the TensorFlow model

creates a Keras callback for computing metrics during training.

```
[16] from transformers.keras_callbacks import KerasMetricCallback

     metric_callback = KerasMetricCallback(metric_fn=compute_metrics, eval_dataset=tf_validation_set)

[17] from transformers.keras_callbacks import PushToHubCallback

     push_to_hub_callback = PushToHubCallback(
         output_dir="./xlnet-new",
         tokenizer=tokenizer,
     )

[18] callbacks = [metric_callback, push_to_hub_callback]
```

Figure 92: callback model

trains the TensorFlow model using the provided training and validation datasets, over a specified number of epochs, and using specified callbacks during training

```
[19] model.fit(x=tf_train_set, validation_data=tf_validation_set, epochs=3, callbacks=callbacks)

Epoch 1/3
WARNING:tensorflow:AutoGraph could not transform <function infer_framework at 0x7b53aea65630> and will run it as-is.
Cause: for/else statement not yet supported
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function infer_framework at 0x7b53aea65630> and will run it as-is.
Cause: for/else statement not yet supported
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING:tensorflow:Gradients do not exist for variables ['tfxl_net_for_sequence_classification/transformer/mask_emb:0'] when minimizing the
WARNING:tensorflow:Gradients do not exist for variables ['tfxl_net_for_sequence_classification/transformer/mask_emb:0'] when minimizing the
WARNING:tensorflow:Gradients do not exist for variables ['tfxl_net_for_sequence_classification/transformer/mask_emb:0'] when minimizing the
WARNING:tensorflow:Gradients do not exist for variables ['tfxl_net_for_sequence_classification/transformer/mask_emb:0'] when minimizing the
1000/1000 [==============================] - 399s 311ms/step - loss: 0.6884 - accuracy: 0.9130 - val_loss: 0.2723 - val_accuracy: 0.9130 -
Epoch 2/3
1000/1000 [==============================] - 281s 281ms/step - loss: 0.1980 - accuracy: 0.9345 - val_loss: 0.1866 - val_accuracy: 0.9345 -
Epoch 3/3
1000/1000 [==============================] - 288s 288ms/step - loss: 0.1361 - accuracy: 0.9340 - val_loss: 0.1800 - val_accuracy: 0.9340 -
<tf_keras.src.callbacks.History at 0x7b53465caa40>
```

Figure 93: fit the model

```
[20] # Evaluate your model on the test set
     evaluation_result = model.evaluate(tf_test_set)

125/125 [==============================] - 9s 73ms/step - loss: 0.1658 - accuracy: 0.9410
```

Figure 94: evaluation the model

The previous code formed the basis for training the other models for emotion detection, below are the results of all the models

xlnet/xlnet-base-cased model:

Loss	The loss of the model is 0.1361 which means that the model has a 13.61% lower loss value, and it is better to predict the correct output for a given input.
Accuracy	The accuracy of the model is 0.934, which means that the model correctly predicted the output for 93.4% of the inputs.
Precision	The precision score of the model is 0.9357 means the model predicted a positive outcome, it was correct about 93.57% of the time.
Recall	The recall score of the model is 0.934, which means that the model predicted about 93.4% of all actual positive instances in the dataset.
F1	The F1 score of the model is 0.9335, which means that the model has a 93.35% higher value and it indicates a good balance between precision and recall.

Table 47: Performance evaluation of xlnet

Xlm-roberta-base model:

Loss	The loss of the model is 0.1361means that the model has a 13.61% lower loss value and it's better to predict the correct output for a given input.
Accuracy	The accuracy of the model is 0.944, which means that the model correctly predicted the output for 94.4% of the inputs.
Precision	The precision score of the model is 0.9458, which means the model predicted a positive outcome, it was correct about 94.58% of the time.
Recall	The recall score of the model is 0.944, which means that the model predicted about 94.4% of all actual positive instances in the dataset.
F1	The F1 score of the model is 0.9436, which means that the model has a 94.36% higher value and it indicates a good balance between precision and recall.

Table 48: Performance evaluation of Roberta

Distilroberta-base model:

Loss	The loss of the model is 0.13661 which means that the model has a 13.66% lower loss value and it's better to predict the correct output for a given input.
Accuracy	The accuracy of the model is 0.942, which means that the model correctly predicted the output for 94.2% of the inputs.
Precision	The precision score of the model is 0.9442, which means the model predicted a positive outcome, it was correct about 94.42% of the time.
Recall	The recall score of the model is 0.942, which means that the model predicted about 94.2% of all actual positive instances in the dataset.
F1	The F1 score of the model is 0.9417, which means that the model has a 94.17% higher value and it indicates a good balance between precision and recall.

Table 49: Performance evaluation of Distilroberta

Microsoft/deberta-v3-xsmall model:

Loss	The loss of the model is 0.1951means that the model has a 19.51% lower loss value and it's better to predict the correct output for a given input.
Accuracy	The accuracy of the model is 0.933, which means that the model correctly predicted the output for 93.3% of the inputs.
Precision	The precision score of the model is 0.9356, which means the model predicted a positive outcome, it was correct about 93.56% of the time.
Recall	The recall score of the model is 0.933, which means that the model predicted about 93.3% of all actual positive instances in the dataset.
F1	The F1 score of the model is 0.9326, which means that the model has a 93.26% higher value and it indicates a good balance between precision and recall.

Table 50: Performance evaluation of Deberta

google-bert/bert-base-uncased model:

Loss	The loss of the model is 0.1018, which means that the model has a 10.18% lower loss value and it's better to predict the correct output for a given input.
Accuracy	The accuracy of the model is 0.945 means that the model correctly predicted the output for 94.5% of the inputs.
Precision	The precision score of the model is 0.9466, which means the model predicted a positive outcome, it was correct about 94.66% of the time.
Recall	The recall score of the model is 0.945, which means that the model predicted about 94.5% of all actual positive instances in the dataset.
F1	The F1 score of the model is 0.9446, which means that the model has a 94.46% higher value, and it indicates a good balance between precision and recall.

Table 51: Performance evaluation of Bert

5.3. I/O Screens

Signup screen

The sign-up page requires 6 fields name, age, email, username, password, and confirm password, provided by the new user. Which will be used to create a new account for them. The data provided will be stored in the system's database.

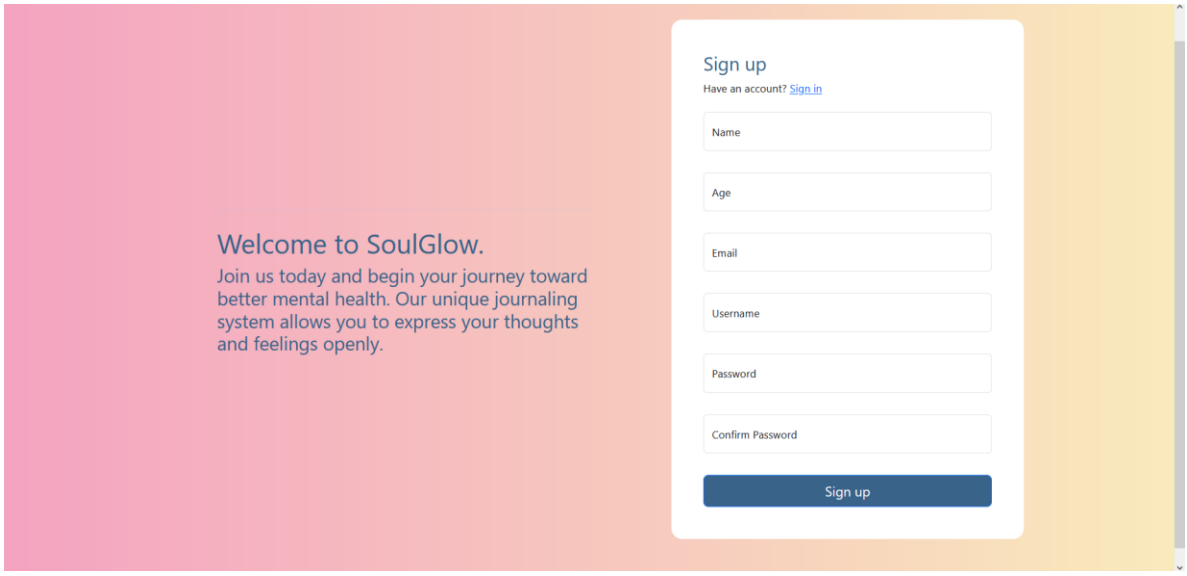


Figure 95: Signup screen

Sign in screen

On the sign in page, the user who already has an account can log in by providing two inputs, Username, and Password. After providing the information it must be compared to the system's database to ensure that it matches, and the user is then able to sign in. They can also retrieve their password if they have forgotten it. Or create an account if they have not made one yet.

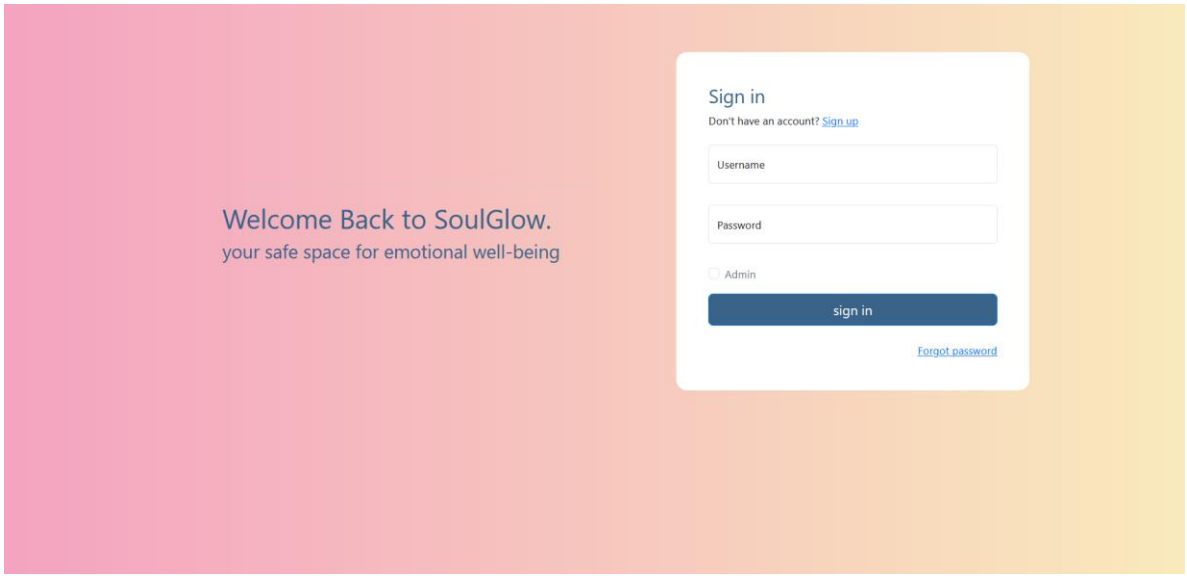


Figure 96: Sign in screen

Reset password screen

On this page, from the (forgot password) option, the user will be able to change the password. The user will provide the email, after that he will receive an email to reset the password.

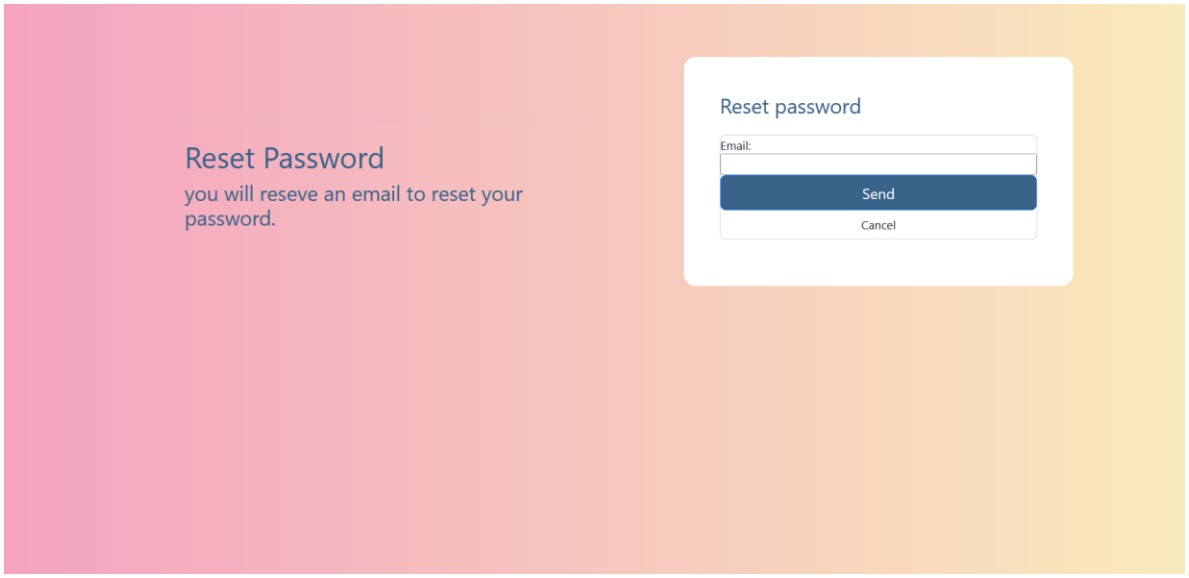


Figure 97: Reset password screen

Settings screen

On this page, the user can change the account information, read the FAQ, and delete the account.

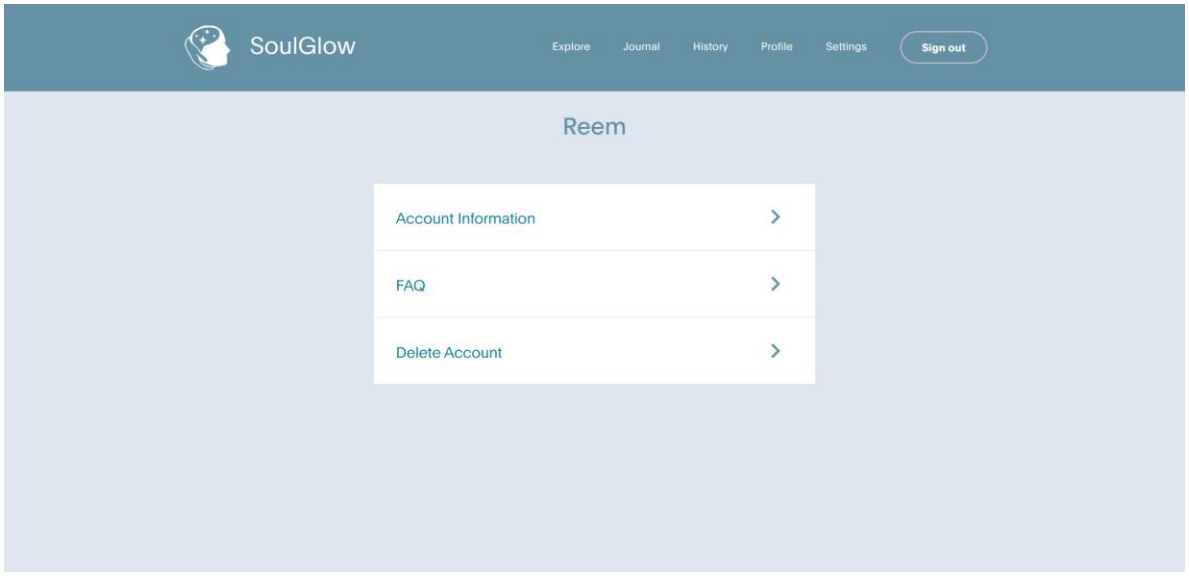


Figure 98: Settings screen

Contact us screen

On this page, the user can contact the SoulGlow system for any questions, Feedback, or suggestions, by providing the name, email address, and the message.

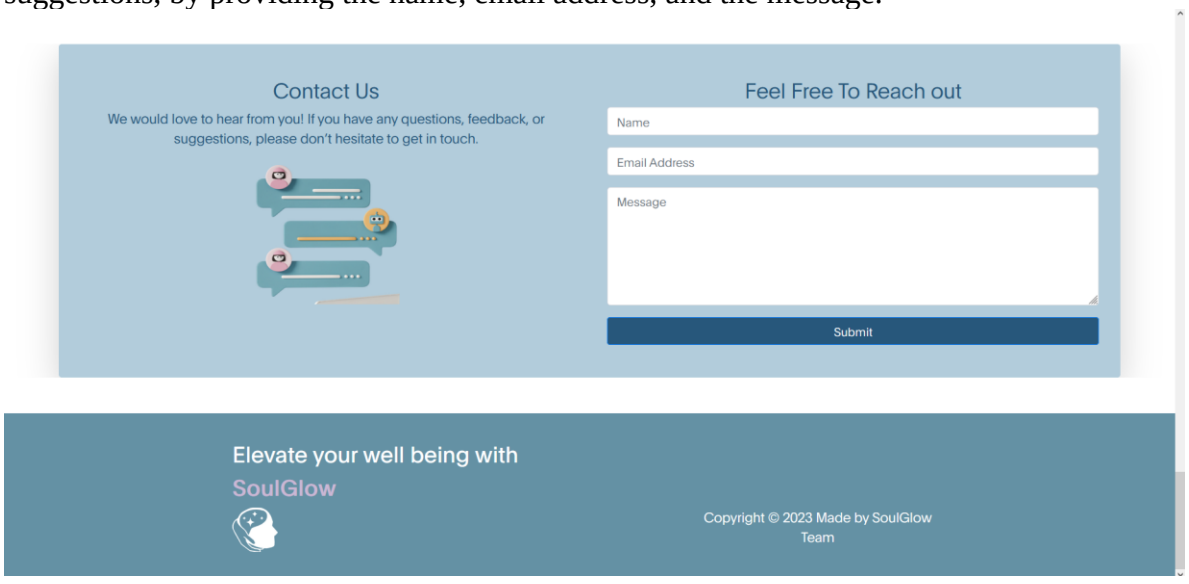


Figure 99: Contact us screen

Journal screen

On this page, the user will provide answers about his current mood, and how their day was. The user can upload an image that represents their mood.

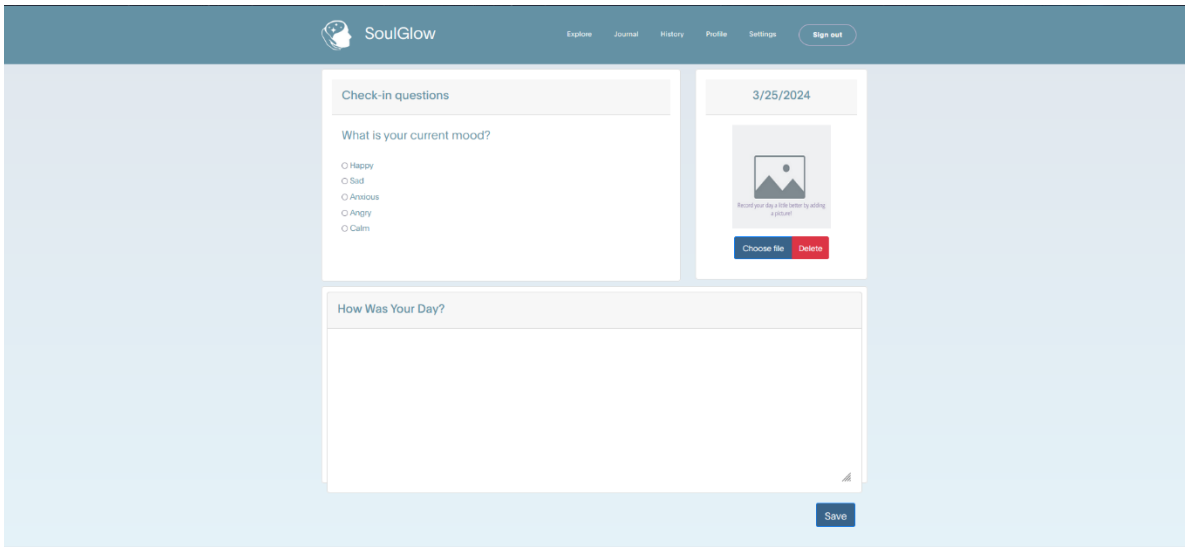


Figure 100: journal screen

History screen

On this page, you will display all the Journals for the user by providing the date of the journal, mood, the journal text, and the picture if available.

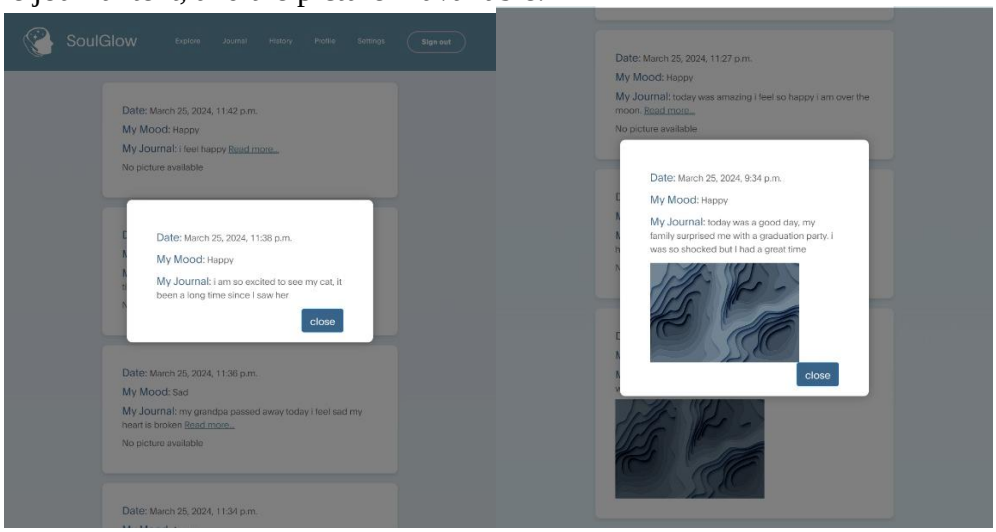


Figure 101: History screen

Profile screen

On this page, it will analyze the user journals, first the user will receive a warning message if a depression is predicted with ok button.

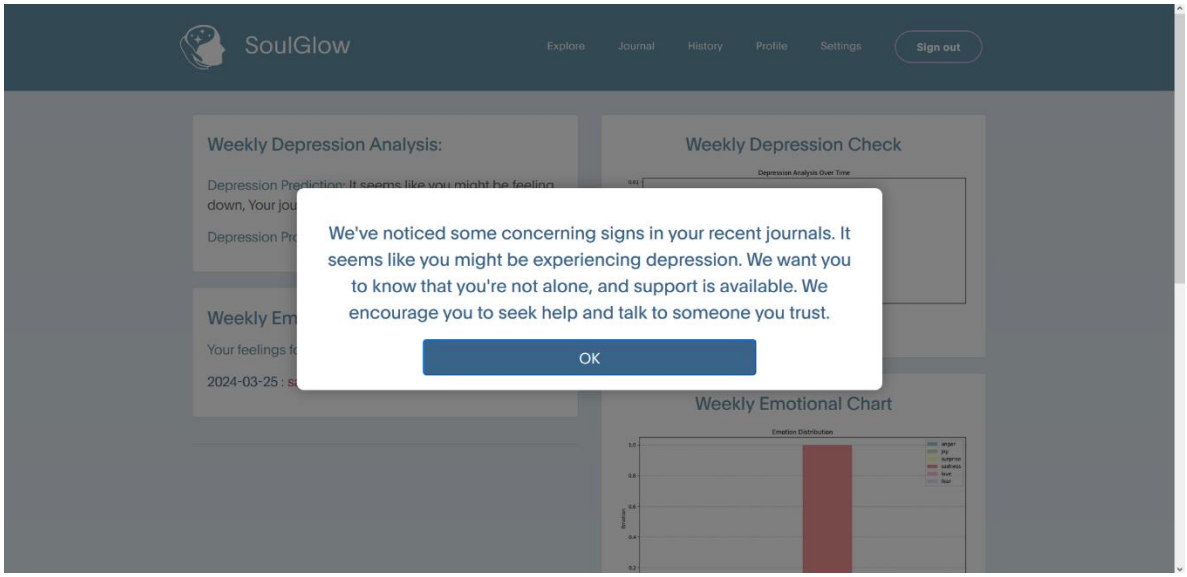


Figure 102: warning message screen

It will show the user weekly depression analysis by providing the prediction and the probability of depression, also weekly emotion analysis shows the user's emotion for the current day with the probability of it, and provides three graphs the weekly depression analysis, the weekly emotion distribution chart, and monthly emotional chart.

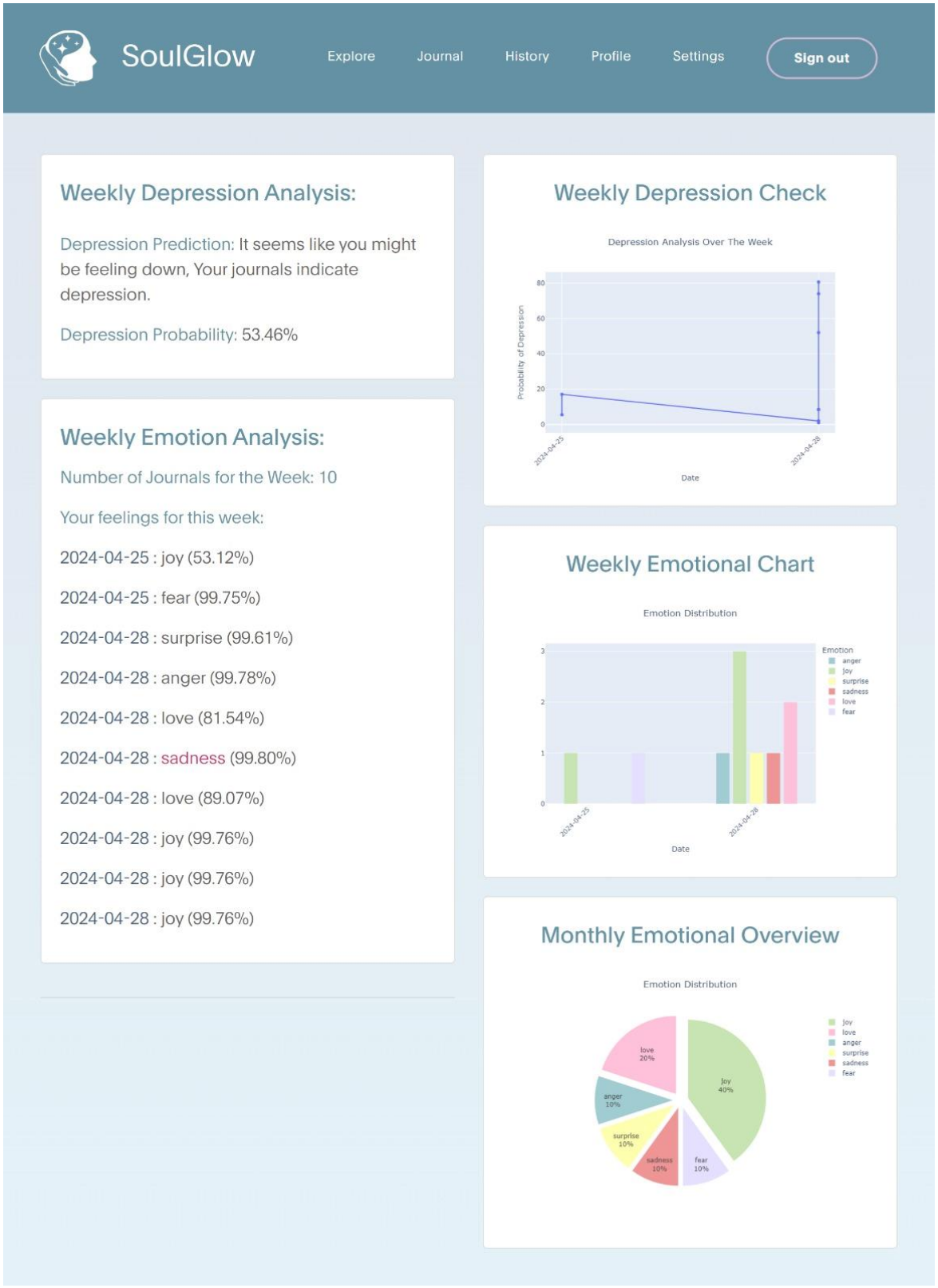


Figure 103: Profile screen

Chapter 6:

6. Testing

6.1 Test Plan

In this section, we'll put our system through its paces to see how it performs in real-world situations. We will be evaluating its features and functionalities thoroughly to ensure it meets our standards for effectiveness and reliability. This chapter marks a crucial step in validating our system's performance and suitability for practical use by conducting both functional testing and non-functional testing.
We will conduct complete unit testing (functional Testing), where individual components like functions and methods are tested in isolation to ensure that the web application's features, user interactions, and error handling work as intended, validating it against predefined criteria to ensure smooth operation and user satisfaction, here we will go through all the16 features we have.

6.2 Test Cases

Functional Testing

Test Case 1: (Sign up - user)

TEST ID		SIGNUP			
TEST CASE DESCRIPTION		Users can create an account by providing valid information and successfully register on the platform.			
S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT
1	Launch website	Launch website	Main page	Mainpage	Pass
2	click on signup button	Click on signup button	Launch signup page	Launch signup page	Pass
3	Enter valid user details then click on the signup button	Username: Sama Age:22 Email: Sama@gmail.com Name: Sama Password: ********* Confirm password: *********	receive a welcoming email, an activation email, and a notification to check their inbox	receive a welcoming email, an activation email, and a notification to check their inbox	Pass
4	Enter Existed email then click on the signup button	Username: Sama2 Age:22 Email: Sama@gmail.com Name: Sama Password: ********* Confirm password: ********	warning massage “email already exist”	warning massage “email already exist”	Pass
5	Enter Existed Username then click on the signup button	Username: Sama Age:22 Email: Sama3@gmail.com Name: Sama Password: **** Confirm password: ****	warning massage “Username already exist”	warning massage “Username already exist”	Pass

TEST ID		SIGNUP			
TEST CASE DESCRIPTION		Users can create an account by providing valid information and successfully register on the platform.			
S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT
6	Enter null value into the signup form.	Username: Sama3 Age:22 Email: Sama3@gmail.com Name: _____ Password: ********* Confirm password: ********	warning massage “please fill out this field”	warning massage “please fill out this field”	Pass
7	Enter invalid email into the signup form.	Username: Sama4 Age:22 Email: a Name: Sama Password: ********* Confirm password: ********	warning massage “Please include @ in the email address, ‘a’ is missing a @”	warning massage “Please include @ in the email address, ‘a’ is missing a @”	Pass
8	Enter unmatched passwords into the signup form.	Username: Sama5 Age:22 Email: Sama5@gmail.com Name: Sama Password: ********* Confirm password: ****	warning massage “The passwords you entered do not match.”	warning massage “The passwords you entered do not match.”	Pass
9	Enter username longer than 10 char into the signup form.	Username: Sama6666666 Age:22 Email: Sama6@gmail.com Name: Sama Password: ********* Confirm password: ********	warning massage “Username cannot exceed 10 characters.”	warning message “Username cannot exceed 10 characters.	Pass

TEST ID		SIGNUP			
TEST CASE DESCRIPTION		Users can create an account by providing valid information and successfully register on the platform.			
S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT
9	Enter Username contains other than letters and numbers into the sign-up form.	Username: Sama@7 Age:22 Email: Sama7@gmail.com Name: Sama Password: ********** Confirm password: *********	warning message "Username must contain only letters and numbers.	warning massage "Username must contain only letters and numbers."	Pass
10	Enter Age under 12 then click on the signup button	Username: Sama2 Age:11 Email:Sama@gmail.com Name: Sama Password: ********** Confirm password: *********	warning message "Sorry, you must be at least 12 years old to sign up. Please try again later."	warning message "Sorry, you must be at least 12 years old to sign up. Please try again later."	Pass
11	click on the activation link that was sent to the user Email.	Activation link to the email	Logged in and Redirected to the explore page	Logged in and Redirected to the explore page	Pass

Table 52: Test Case 1: (Sign up - user)

These warning messages should provide users with clear feedback on what information is missing or incorrect, helping them to correct their input and successfully complete the signup process.

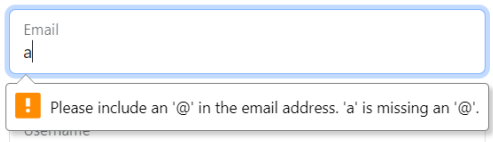
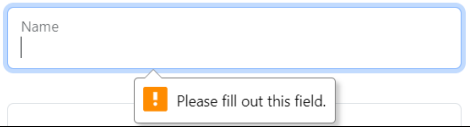
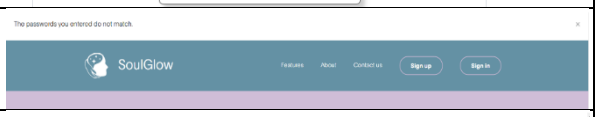
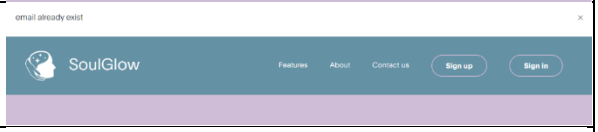
invalid email	
null value	
unmatched passwords	
Username contains other than letters and numbers	
Age under 12	
Existed email	
Existed username	

Table 53: Warning messages

These messages serve to reassure users that their information has been successfully processed and validated, providing a positive user experience during the signup process, the message will inform user to check their inbox, to received welcoming and confirmation email.

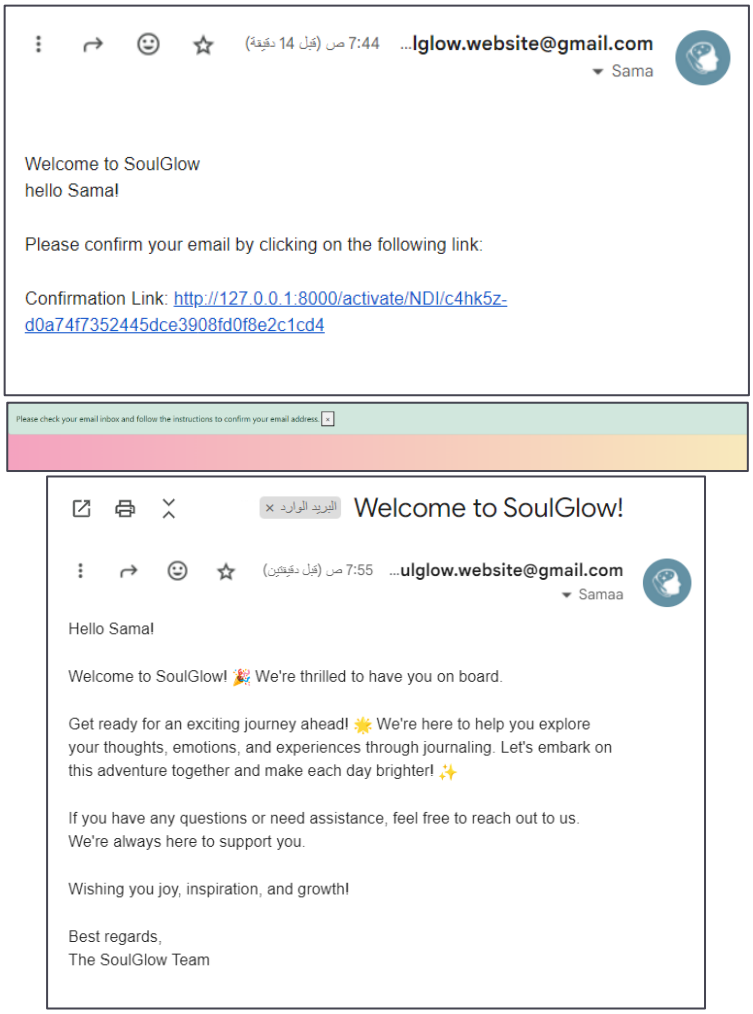


Figure 104: welcoming and confirmation email

Test Case 2: (Sign in-user)

TEST ID		SIGNIN			
TEST CASE DESCRIPTION		Users can sign in to their account with valid credentials.			
S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT
1	Launch website	Launch website	Main page	Main page	Pass
2	click on sign in button	Click on sign in button	Launch sign in page	Launch sign in page	Pass
3	Enter valid user credentials then click on the sign-in button.	Username: Sama, Password: *****	View logged in message and redirect to explore page	View logged in message and redirect to explore page	Pass
4	Enter invalid user credentials then click on the sign-in button.	Username: Sama, Password: ***	Error message indicating invalid credentials	Error message indicating invalid credentials	Pass

Table 54: Test Case 2: (Sign in-user)

These messages provide clear feedback to users regarding the outcome of their sign-in attempts, ensuring a smooth and informative user experience.

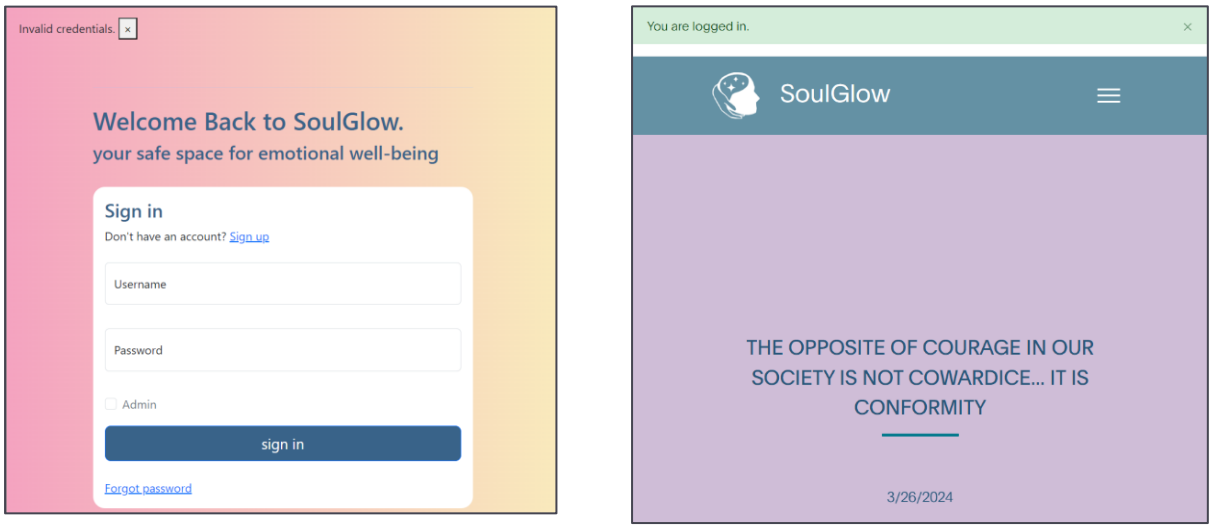


Figure 105: sign-in notification

Test Case 3: (Sign out – user and admin)

TEST ID		SIGNOUT			
TEST CASE DESCRIPTION		Users and administrators can sign out of their account successfully.			
	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT
1	Click on the sign-out button.	Click on the sign-out button.	successfully logged out message and redirected the main page	successfully logged out message and redirected the main page	Pass

Table 56: Test Case 3: (Sign out – user and admin)

After clicking on the sign-out button in the navigation bar, users and administrators will see a message confirming their action and notifying them that they have been successfully logged out of their account.

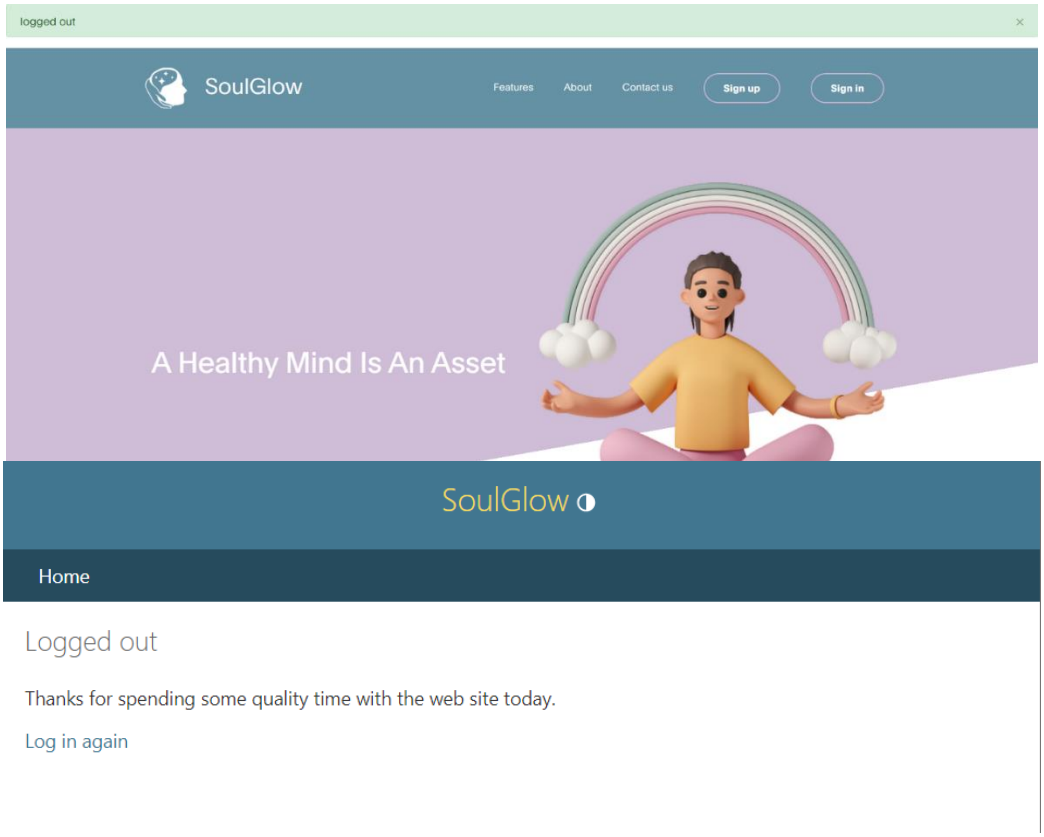


Figure 106: Sign out notification for user or administrator

Test Case 4: (Reset Password - User)

TEST ID		RESET PASSWORD			
TEST CASE DESCRIPTION		Users can reset their password using the password reset functionality.			
S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT
1	locate the "Forgot Password" link on the sign in page.	Click on forgot password	enter email address	enter email address	pass
2	Enter the email address associated with the account	Enter the email address associated with the account	“Check your inbox” message	“Check your inbox” message	pass
3	click on the reset link that was sent to the user email.	click on the reset link that was sent to the user email.	Enter new password	Enter new password	pass
4	Enter a valid new password then click on confirm	New password: ********* then confirm	“Your password has been changed successfully. Click on log in “message	“Your password has been changed successfully. Click on log in “message	pass
5	Enter an invalid new password	New password: **** then confirm	“This password is short, it must contain at least 8 characters” message	“This password is short, it must contain at least 8 characters” message	pass
6	Click on log in hyperlink.	Click on log in hyperlink.	Launch sign in page	Launch sign in page	pass

Table 57: Test Case 4: (Reset Password - User)

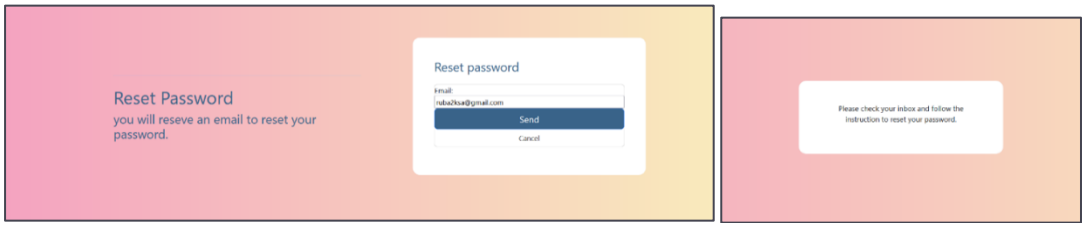


Figure 107: Reset password

This page allows users to initiate the password reset process. They can access this page by clicking on a "Forgot Password" link on the login page.

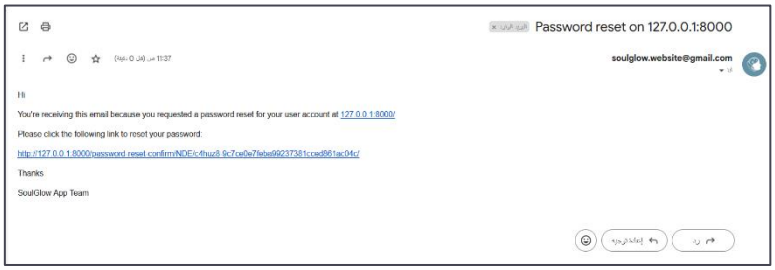


Figure 108: Forgot password

After the user submits their email address on the reset page, an email containing a password reset link is sent to the user's email address. This email serves as a verification step and contains instructions on how to proceed with resetting the password.

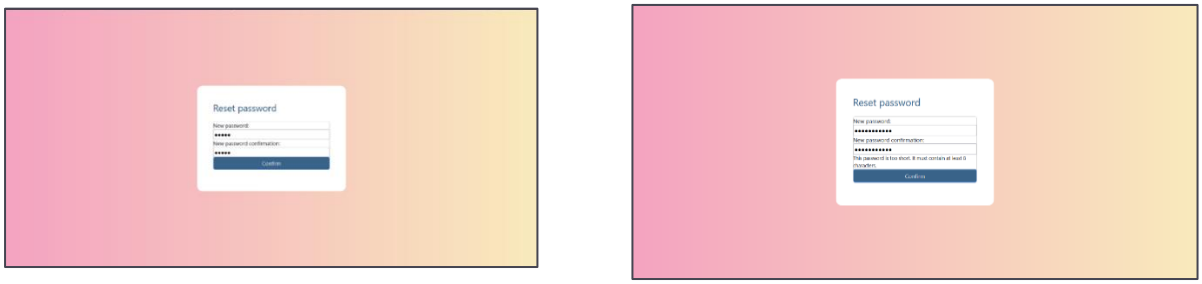


Figure 109: verification of resetting password

The password reset link in the email directs the user to a password reset page. Here, the user can enter a new password for their account.

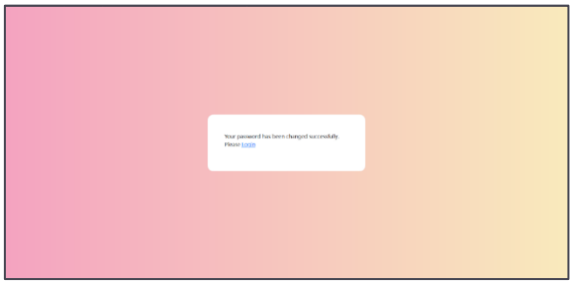


Figure 110: new password

After successfully resetting their password, users are redirected to a confirmation page. This page reassures the user that their password has been changed successfully and provides them with instructions to log in using their new password.

Test Case 5: (Delete Account - user)

TEST ID		DELETE ACOUNT			
TEST CASE DESCRIPTION		users can delete their account.			
S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT
1	Navigate to settings page	Navigate to settings page	settings page	settings page	Pass
2	Click on the delete account	Click on the delete account	Confirmation message	Confirmation message	Pass
3	Click on Delete button	Click on Delete button	Redirect to main page and delete account from database	Redirect to main page and delete account from database	Pass

Table 58: Test Case 5: (Delete Account - user)

This is a confirmation popup message designed to ensure that users are certain they want to delete their account.

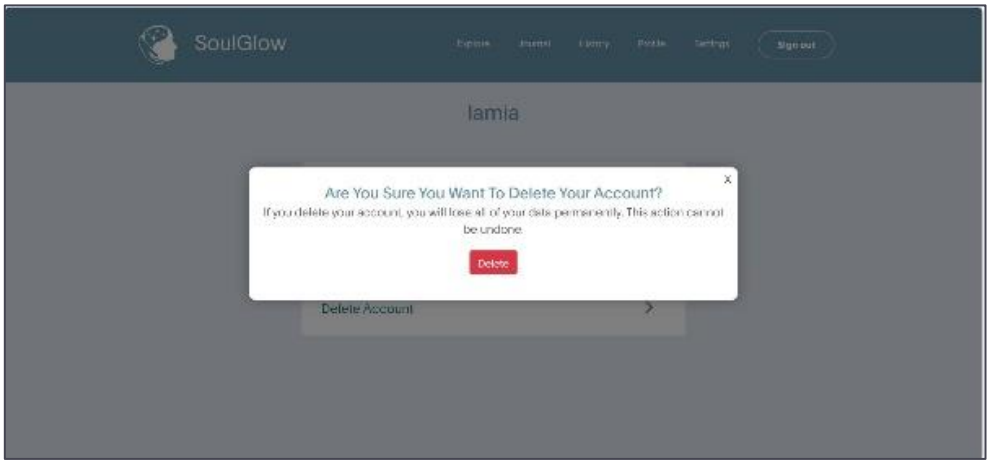


Figure 111: delete account popup message

Test Case 6: (Enter & Save journal - user)

TEST ID		Enter/ save journal.			
TEST CASE DESCRIPTION		User enters a journal which consists of 3 fields mood, journal text and option to upload an image then user can save it.			
S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT
1	Launch the Journal page	Click on Journal page	Journal page	Journal page	pass
2	Users enter a mood and journal text and uploads an image then click on "Save"	Mood, journal text, image	The journal saved pop up message appear	The journal saved pop up message appear	pass
3	Users enter a mood and journal text then click on "Save"	Mood, journal text	The journal saved pop up message appear	The journal saved pop up message appear	pass
4	Users enter a journal text then click on "Save"	Journal text	The "Mood not entered" error message appear	The "Mood not entered" error message appear	pass
5	Users enter a mood then click on "Save"	mood	The "Journal not entered" error message appear	The "Journal not entered" error message appear	pass

Table 59: Test Case 6: (Enter & Save journal - user)

Both mood and journal were entered therefore the journal was saved successfully.

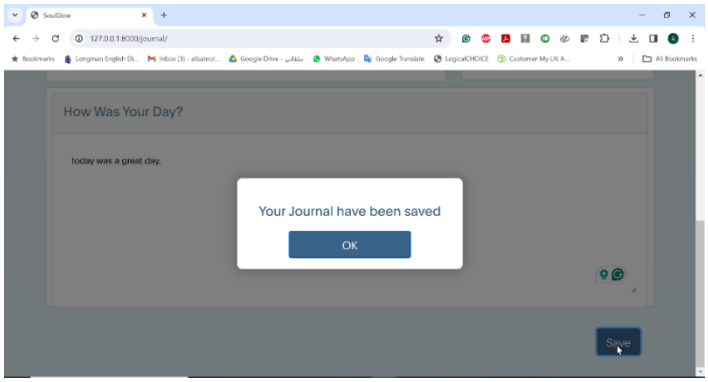


Figure 112: save journal

Here the journal was entered but the mood wasn't entered.

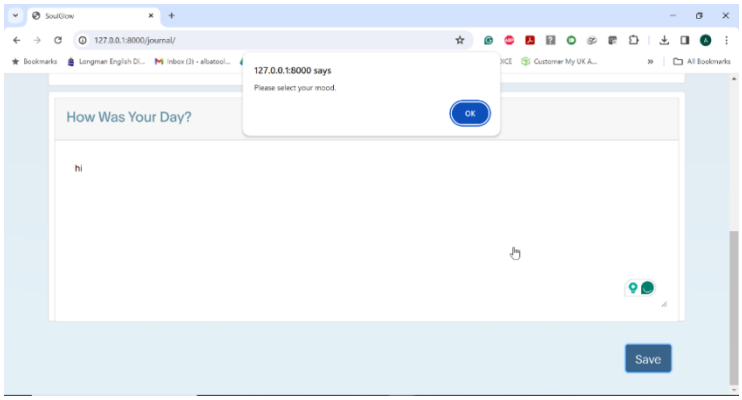


Figure 113: enter mood, popup message

Here the mood was entered but the journal wasn't.

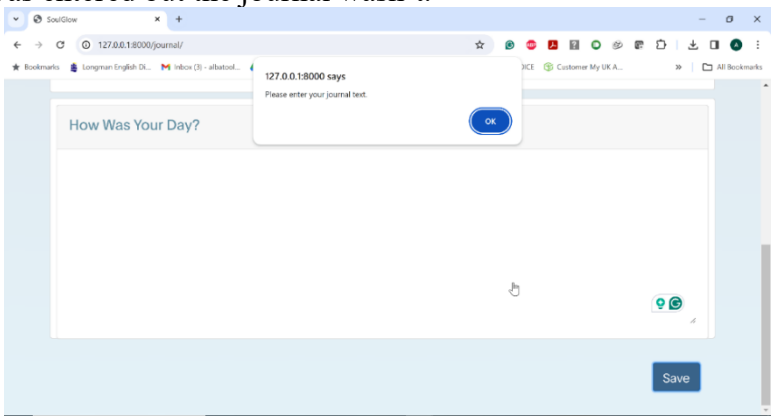


Figure 114: enter journal, popup message

Test Case 7: (View History - user)

TEST ID		View history- user			
TEST CASE DESCRIPTION		Users can view a collection of journal entries. Each entry contains the date, mood, a shortened snippet of the journal text, and a picture if available.			
S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT
1	User has no journals and clicks on the history page	Click on History page	The “no journal entries available” message in the history page	The “no journal entries available” message in the history page	pass
2	User has journals and clicks on the history page	Click on History page	History page with all the users' previous journals	History page with all the users' previous journals	pass
3	Click on “Read more” to see details of the journal	Click on “Read more”	Journal details pop up message	Journal details pop up message	pass
4	User clicks on “Delete” to delete his journal	Click on “Delete”	The journal disappears from the history page	The journal disappears from the history page	pass

Table 60: Test Case 7: (View History - user)

Here the user has no journal entries and they clicked on the History page, as shown in the figure above the “no journal entries available” message appeared.

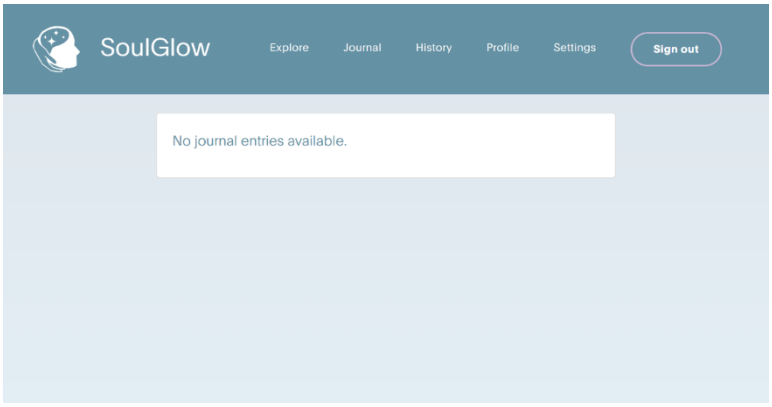


Figure 115: no journal entries message

Here the User has journals and he clicked on the history page where he can see his previous journals and they could delete the journal if he wanted to.

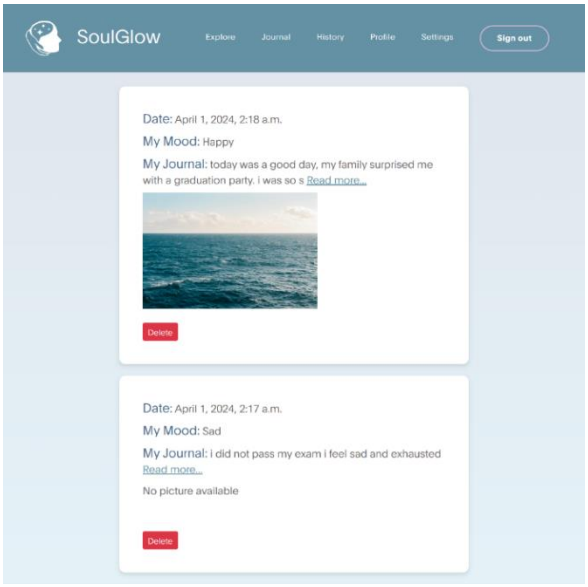


Figure 116: previous journals

The user can see his journal details by clicking on the “Read more” button.

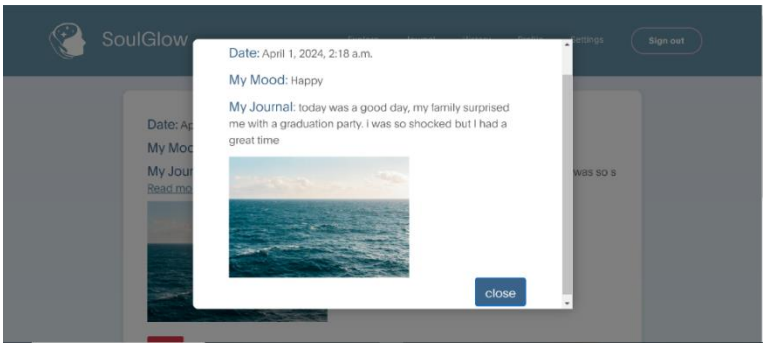


Figure 117: journal details

After clicking on” Delete” the journal disappeared from the history page.

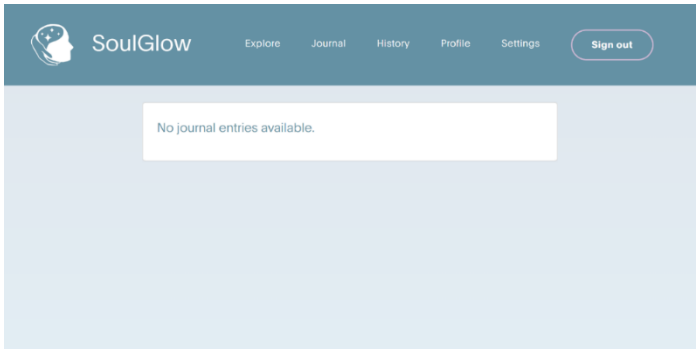


Figure 118: after Delete journal details

Test Case 8: (View Profile - user)

TEST ID		View profile- user			
TEST CASE DESCRIPTION		Users can view the journal entries analysis. It provides a user interface for displaying information and data visualization for the analysis of depression and emotions over time.			
S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT
1	User has not entered journals for the month, and clicks on the profile page	Click on profile page	The “You haven’t entered any journals this month” will appear.	The “You haven’t entered any journals this month” will appear.	pass
2	User has not entered journals for the week, and clicks on the profile page	Click on profile page	The monthly overview chart will appear in profile page	The Monthly Emotional Overview chart will appear in profile page	pass
3	User entered journals for the week and clicks on the profile page	Click on profile page	Profile page with emotion and depression analysis	Profile page with emotion and depression analysis	pass

Table 61: Test Case 8: (View Profile - user)

If the user has not entered any journals during the month, then the “You haven’t entered any journals this month” message will appear.

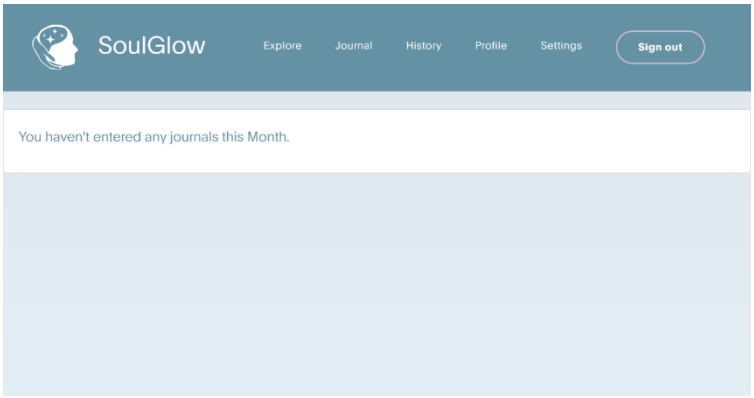


Figure 119: User hasn’t entered monthly journals

Here the user has not entered any journals for the current weak, after clicking on the profile button in the navigation bar the Monthly Emotional Overview chart will appear in profile page which is the emotional analysis of the journals that were entered during the month.

Figure 120: monthly emotion overview

The user can see his profile if he has entered journals during the current week, after clicking on the “Profile” on the navigation bar.

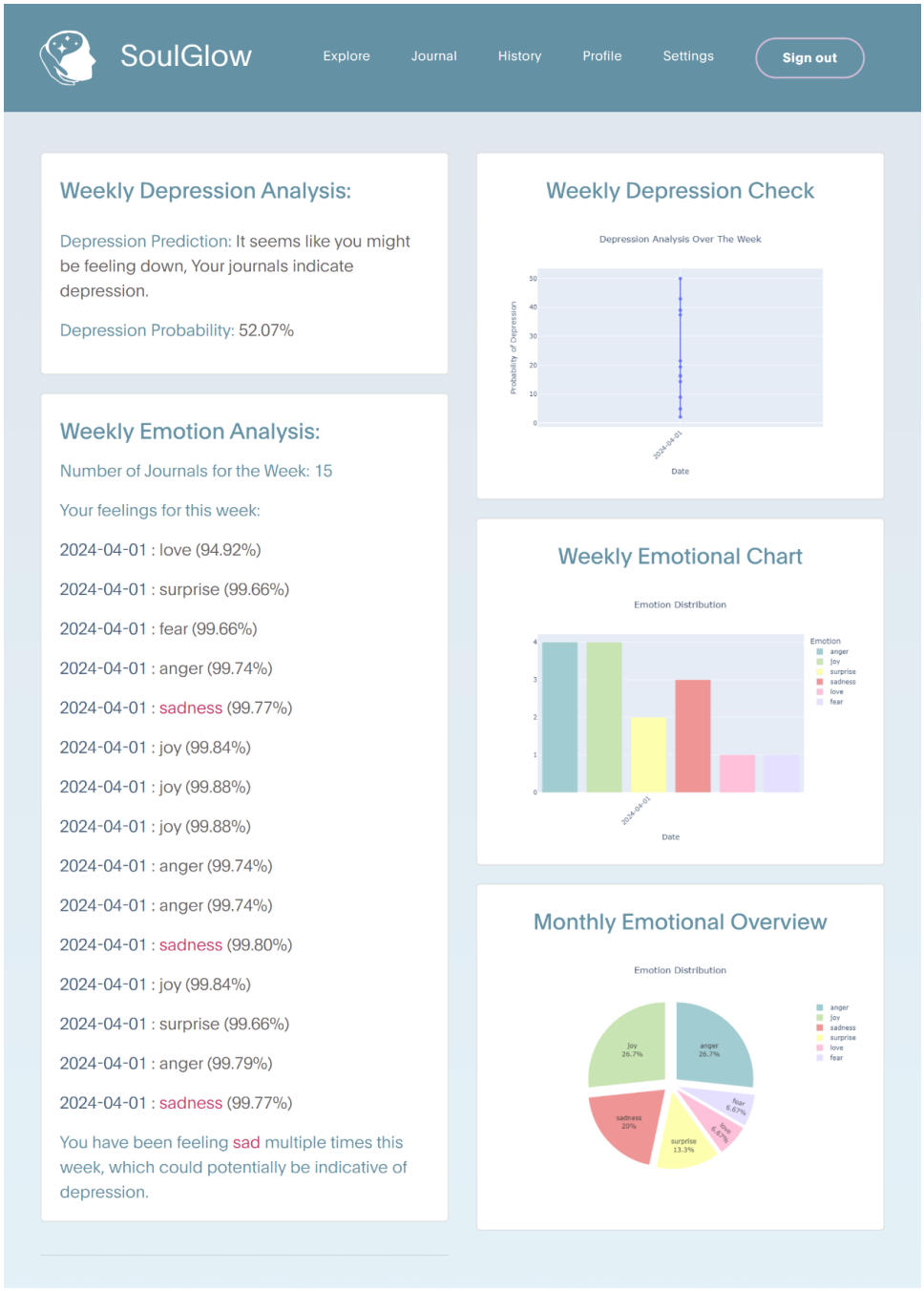


Figure 121: User see profile weekly, monthly

Test Case 9: (View mental health resources – user)

TEST ID		view mental health resources-user			
TEST CASE DESCRIPTION		Users can view mental health resources within the explore page and access detailed information or purchase them from external sources.			
S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT
1	Navigate to the resource section in explore page	Click on the explore page	Explore page	Explore page	pass
2	Click on a resource to view its details	Click on a resource to view its details	Resource details	Resource details	pass
3	click "Buy it" to be redirected to the Amazon page.	Click “Buy it”	redirected to the Amazon page.	redirected to the Amazon page.	pass

Table 62: Test Case 9: (View mental health resources – user)

Clicking on "Show Details" prompts a popup displaying the book's title, author, and description.

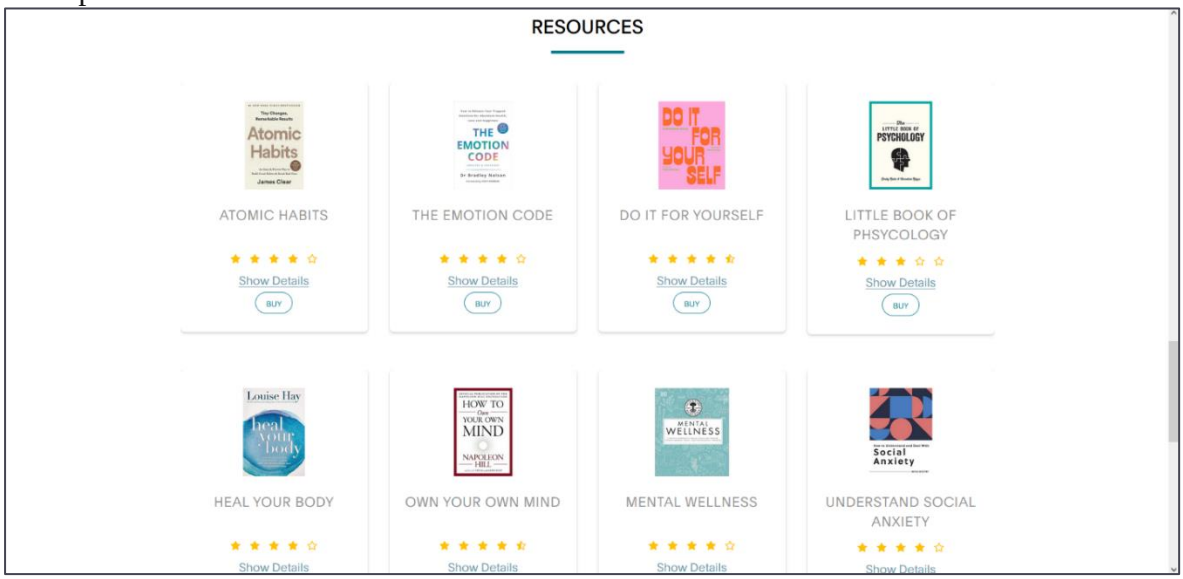


Figure 122: resources

Clicking on "Buy It" redirects users to the Amazon store page for purchasing the book.

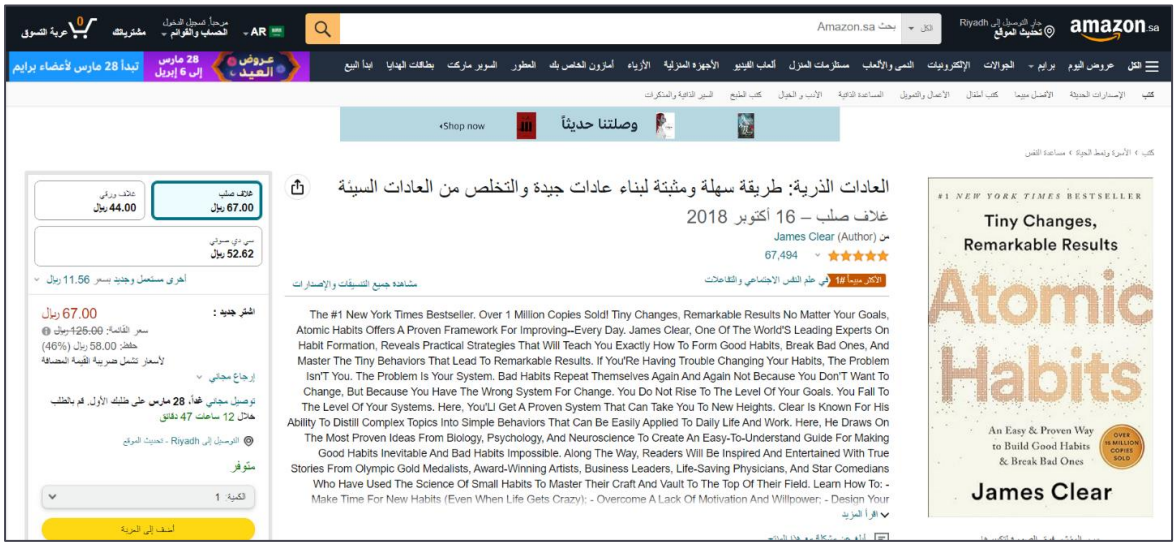


Figure 123: amazon page of the resources

Test Case 10: (View affirmations & Challenges - user)

TEST ID		view affirmation and challenges-user			
TEST CASE DESCRIPTION		Users can display the daily challenges and affirmation to improve their mental health and overall well-being.			
S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT
1	Launch explore page	Click on explore page	Explore page	Explore page	pass
2	Locate the section containing affirmations and challenges.	Locate the section containing affirmations and challenges.	Affirmations and challenges section	Affirmations and challenges section	pass

Table 63: Test Case 10: (View affirmations & Challenges - user)

Users can locate the section containing affirmations and challenges. This section is appropriately labeled and accessible, ensuring users can easily find affirmations and challenges.

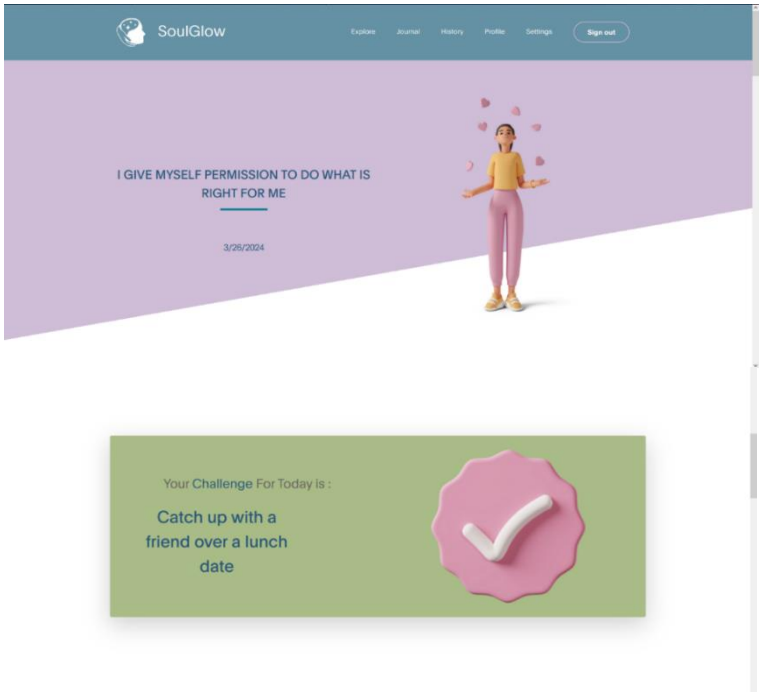


Figure 124: affirmations and challenges.

Test Case 11: (Receive depression alerts - user)

TEST ID		RECEIVE DEPRESSION ALERTS			
TEST CASE DESCRIPTION		Users receive depression alerts when accessing their profile if they are identified as depressed.			
S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT
1	Log in as a user identified as experiencing depression.	Log in	"You are logged in." message	"You are logged in." message	pass
2	Launch profile page	Click on profile page	receives both a popup message and an email expressing concern about their well-being.	receives both a popup message and an email expressing concern about their well-being.	pass

Table 64: Test Case 11: (Receive depression alerts - user)

Upon detection of potential signs of depression, users receive both a popup message and an email expressing genuine concern for their mental health and well-being. These notifications aim to provide support and encourage users to seek assistance if needed.

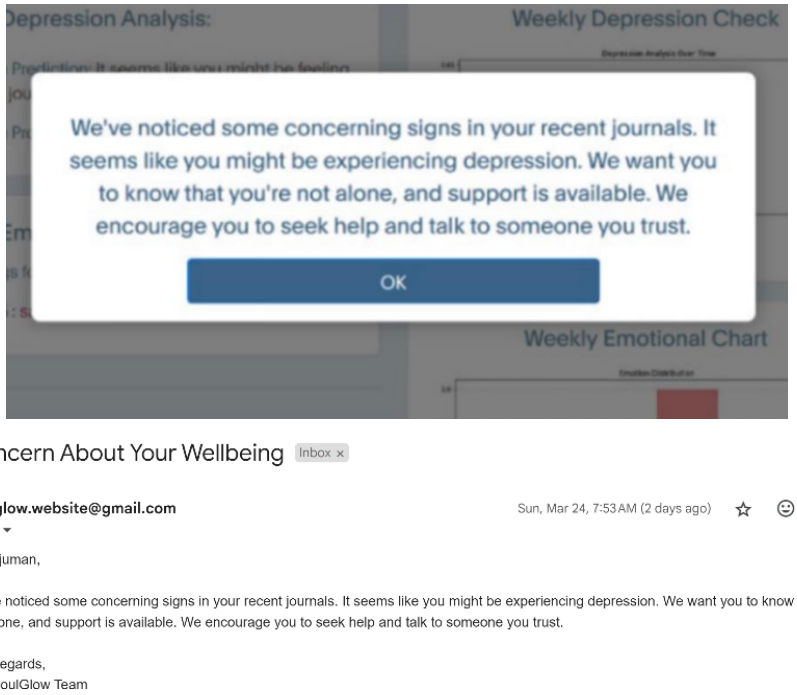


Figure 125: Receive depression alerts.

Test Case 12: (Delete user account - admin)

TEST ID		DELETE USERS ACCOUNT-ADMIN			
TEST CASE DESCRIPTION		deleting a user account by an administrator.			
S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT
1	Launch admin page	Launch admin page	admin page	admin page	Pass
2	Click on “users”	Click on “users”	Table of all accounts	Table of all accounts	Pass
3	Click on the username of account to delete	Click on the username of account to delete	Account details page	Account details page	Pass
4	Click on delete button	Click on delete button	Confirmation message	Confirmation message	Pass
5	Click on “Yes, I’m sure”	Click on “Yes, I’m sure”	Popup message informing the user was deleted	Popup message informing the user was deleted	Pass

Table 65: Test Case 12: (Delete user account - admin)

Clicking on the username of the account navigates users to the account details page. Upon clicking the delete button, admin is presented with a confirmation message. Subsequently clicking on "Yes, I'm sure" triggers a popup message confirming the successful deletion of the user's account.

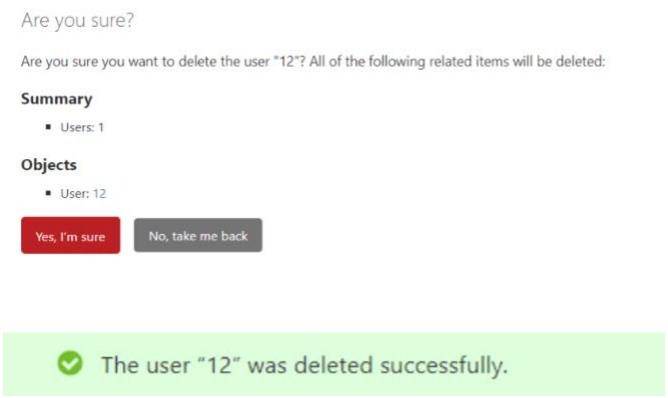


Figure 126: Delete user account – admin.

Test Case 13: (Sign in - admin)

TEST ID					
TEST CASE DESCRIPTION		administrators can sign in to their account with valid credentials.			
S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT
1	Launch the sign in page	Launch the sign in page	Redirect to the sign in page	Redirect to the sign in page	Pass
2	Enter valid admin credentials then click on both the admin check button and the sign-in button.	Username: reeman Password: ******** And click on both the admin check button and the sign-in button.	View logged in message and redirect to admin page	View logged in message and redirect to admin page	Pass
3	Enter invalid user credentials then click on both the admin check button and the sign-in button.	Username: reeman Password: ******** And click on both the admin check button and the sign-in button.	Error message indicating invalid credentials	Error message indicating invalid credentials	Pass

Table 66: Test Case 13: (Sign in - admin)

This is the admin page, accompanied by a popup indicating successful login.

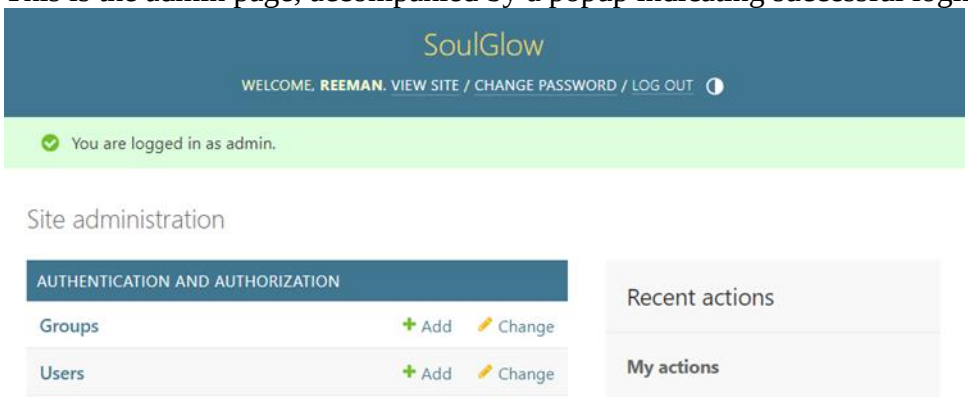


Figure 127: sign-in admin notification

If the admin enters incorrect credentials, a notification will be displayed.

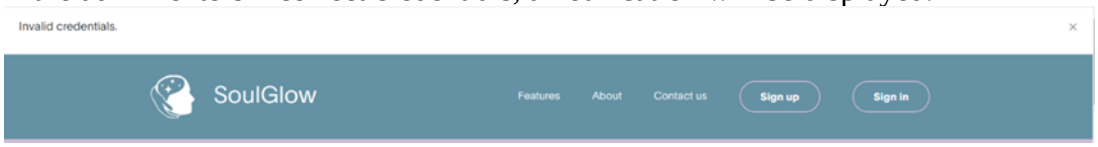


Figure 128: incorrect credentials popup

Test Case 14: (Contact us - anyone)

TEST ID		CONTACT US			
TEST CASE DESCRIPTION		the "Contact Us" form is submitted successfully.			
S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT
1	Launch website and	Launch website	Main page	Main page	Pass
2	Navigate to the "Contact Us" section.	Click on contact us in the navigation bar	"Contact Us" section	"Contact Us" section	Pass
3	Fill out the form with valid information.	Name: Reem Email:reemrashed@gmail.com Message: HI.	“Thank you for your message” message. The email is sent	“Thank you for your message” message. The email is sent	Pass
4	Fill out the form with null value then click on the "Submit" button.	Name: ____ Email:reemrashed@gmail.com Message: HI.	warning massage “please fill out this field”	warning massage “please fill out this field”	Pass
5	Fill out the form with	Name: Reem Email:a	warning massage	warning massage	Pass

	invalid email then click on the "Submit" button.	Message: HI.	“Please include @ in the email address, ‘a’ is missing a @”	“Please include @ in the email address, ‘a’ is missing a @”	

Table 67: Test Case 14: (Contact us - anyone)

This process ensures that users can successfully submit the with valid information"Contact Us" form, receive appropriate warnings for incomplete or incorrect inputs, and get a confirmation message upon submission.

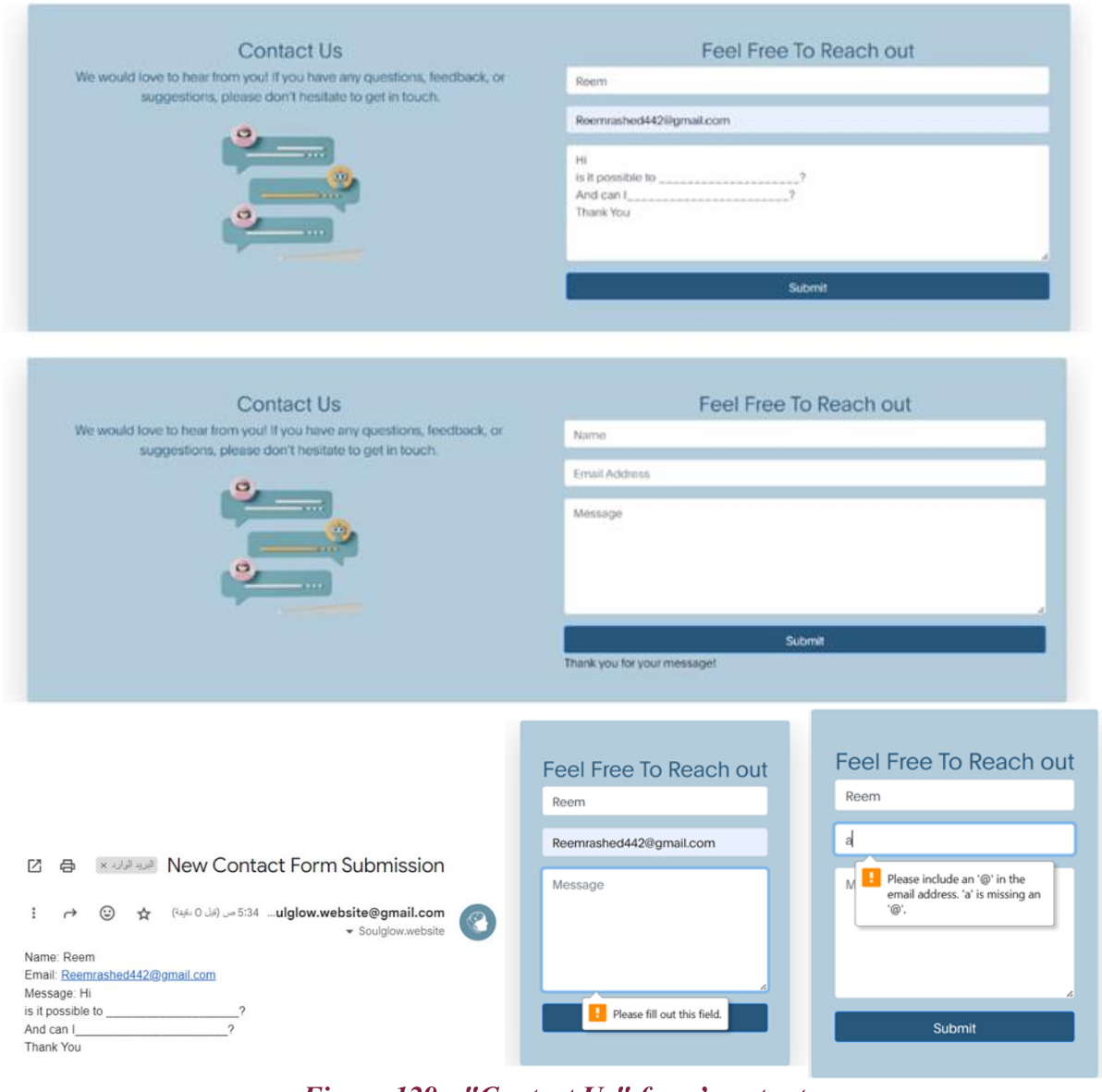


Figure 129: "Contact Us" form’s outputs

Test Case 15: (Update information - user)

TEST ID		UPDATE INFORMATION			
TEST CASE DESCRIPTION		updating user information.			
S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT
1	Navigate to settings page	Navigate to settings page	settings page	settings page	Pass
2	Click on the account information	Click on the account information	Popup with name, username, and email	Popup with name, username, and email	Pass
3	Update information and click on “save changes” button	Name:jumana	Updated database and “Your settings have been updated successfully” message showed	Redirect to main page and delete account from database	Pass

Table 68: Test Case 15: (update information - user)

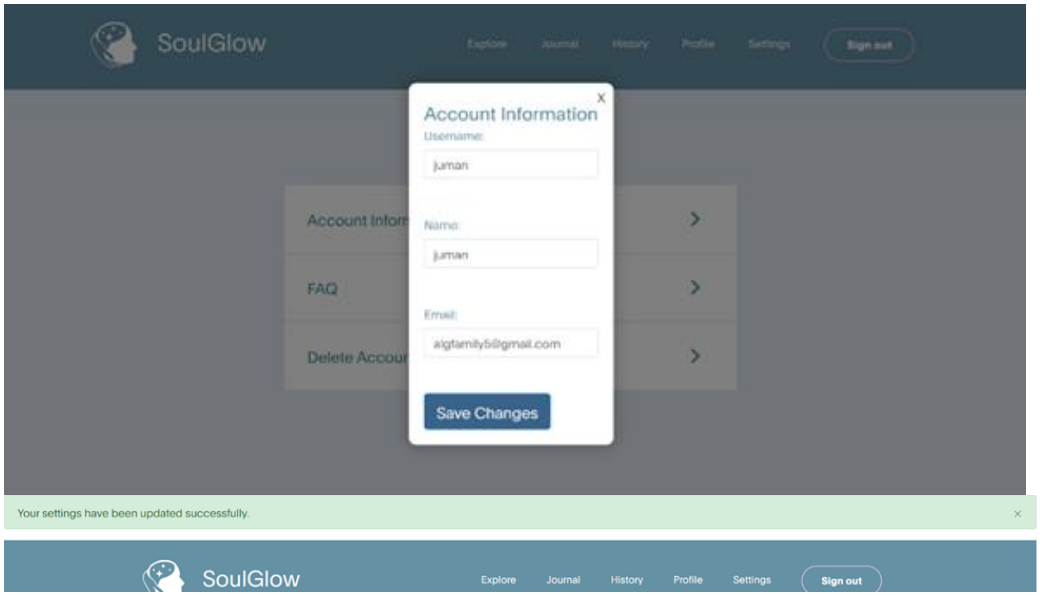


Figure 130: pop up of account information

Test Case 16: (View FAQ - user)

TEST ID		VIEW FAQ			
TEST CASE DESCRIPTION		viewing the Frequently Asked Questions (FAQ)			
S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULT
1	Navigate to settings page	Navigate to settings page	settings page	settings page	Pass
2	Click on the FAQ section	Click on the FAQ section	Popup with all Frequently Asked Questions	Popup with all Frequently Asked Questions	Pass

Table 69: Test Case 16: (View FAQ - user)

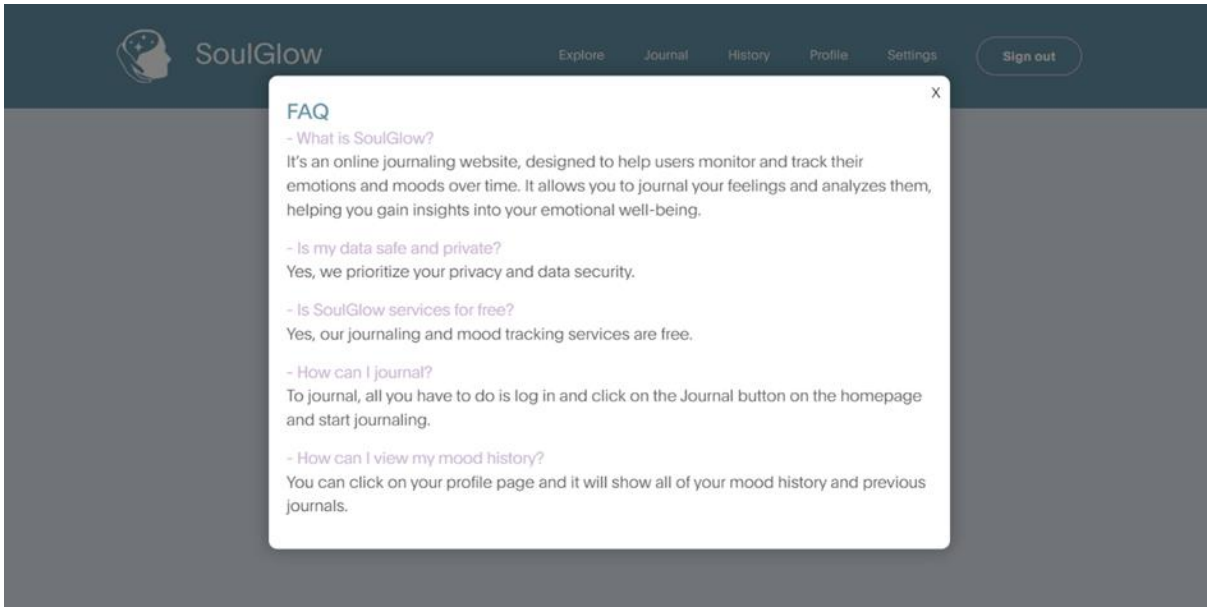


Figure 131: View FAQ - user

Non-Functional Testing

	Performance	Security	Usability
Description	The system must consistently load web pages within 5 seconds or less, ensuring a responsive and efficient user experience.	Sensitive user data such as journals must be encrypted to prevent unauthorized access.	The system's interface must undergo usability testing, achieving a score of at least 80 out of 100 on standard usability scales.
Expected output	Page loading time of 5 seconds or less	User's data should be encrypted	User feedback surveys should indicate satisfaction with the interface and navigation, ensuring user-friendly interaction.
Test result	pass	pass	pass

Table 70: Non-Functional Testing

Usability Testing

We conducted a comprehensive usability test, employing a structured approach to evaluate the effectiveness and efficiency of our website's interface and navigation, consisting of yes/no questions designed to assess various aspects of usability, including ease of navigation, clarity of instructions, and overall user satisfaction. This test was administered to 10 different users, allowing us to gather valuable feedback on the usability of our web application. By analyzing the responses and calculating usability test scores, we gained insights into areas of strength and areas for improvement in our website's usability.

Question \ User	**U1**	**U3**	**U2**	**U4**	**U5**	**U6**	**U8**	**U7**	**U9**	**U10**
1. Did you find it easy to navigate through the website?	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
2. Was the layout of the website intuitive and easy to understand?	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3. Were you able to quickly find the information you were looking for?	✓	✗	✓	✓	✓	✓	✓	✓	✗	✓
4. Were you able to complete tasks on the website without encountering any difficulties?	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓
5. Were the buttons and links clearly labeled and easy to identify?	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
6. Did you feel confident in using the website's features and functionalities?	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
7. Were you able to easily recover from any errors or mistakes while using the website?	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
8. Did the website respond quickly to your actions and commands?	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
9. Did you find the instructions provided on the website clear and unambiguous?	✓	✗	✓	✓	✓	✓	✓	✓	✗	✓
10. Overall, did you find the website user-friendly and enjoyable to use?	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Score	**10**	**8**	**10**	**10**	**10**	**9**	**10**	**9**	**8**	**10**
Total score	**94%**									

Table 71: Usability Testing results

6.3 Test Results

Based on the results of the functional, non-functional and usability testing with the end users, through our SoulGlow website with the user-friendly interfaces, we have successfully tested all the functionality with the above 16 cases scenarios along with the non-functional testing to ensure that SoulGlow website performance, features, user interactions, and error handling work as intended. All requirements have been met and are in full operation.

Chapter 7:

7.Conclusion

7.1 Evaluation

In this project, we successfully developed a digital journaling website powered by machine learning models using the Django framework for rapid development, Jupyter Notebook, and Visual Studio Code as an IDE. We created a database using SQLite. For front-end development, we utilized HTML and CSS to create user-friendly interfaces. Additionally, we incorporated JavaScript for handling user interactions and used the Plotly library for data visualizations. The system enables users to journal and track their mental and emotional journey by analyzing these journals and displaying the results in their profiles. SoulGlow can benefit those who are suffering mentally and are hesitant to seek professional help due to fear of judgment and the stigma associated with discussing mental health challenges, especially depression. The system can serve as a pre-screening depression detection technique, helping with early detection of depression without the need to visit a psychologist. Additionally, users will receive alerts if they are suffering mentally.

System features include:

- Sign up.
- Login.
- Emotion & Depression detection.
- Graph visualizations of analyzed journals.
- Daily challenges & affirmations.
- Mental health resources.

7.2 Future work

For further improvements to our system, SoulGlow, we can add functionalities and features such as:

- Multilingual support.
- Detection of other types of mental illnesses.
- Emotion & Depression detection through images and voice recognition.
- More security by including image encryption.

- Development of a mobile version of the website.

References

[1] Machová, K. et al. (2023) Detection of emotion by Text Analysis Using Machine Learning, Frontiers. Available at: https://www.frontiersin.org/articles/10.3389/fpsyg.2023.1190326/full (Accessed: 21 November 2023).
[2] DeVries, M., & Wilkerson, B. (2003). Stress, work and mental health: A global perspective. Acta Neuropsychiatrica.
[3] Goel, Vidhit. 20 HABITS of ENTREPRENEURS: THAT CAN CHANGE YOUR LIFE. Vidhit Goel, 27 Apr. 2021.
[4] "AdamCodd/emotion-balanced · Datasets at Hugging Face," huggingface.co, Nov. 10, 2023. https://huggingface.co/datasets/AdamCodd/emotion-balanced (accessed May 05, 2024).
[5] Brownlee, Jason. Deep Learning for Natural Language Processing. Machine Learning Mastery, 21 Nov. 2017.
[6] Zeberga, Kamil, et al. "A Novel Text Mining Approach for Mental Health Prediction Using Bi-LSTM and BERT Model." Computational Intelligence and Neuroscience, vol. 2022, 3 Mar. 2022, pp. 1–18, https://doi.org/10.1155/2022/7893775. Accessed 12 May 2022.
[7] T. Zhang, A. M. Schoene, S. Ji, and S. Ananiadou, "Natural language processing applied to mental illness detection: a narrative review - npj Digital Medicine," Nature, Apr. 08, 2022. [Online]. Available: https://www.nature.com/articles/s41746-022-00589-7
[8] "Sentiment Analysis and Opinion Mining," Google Books. [Online]. Available: https://books.google.com/books/about/Sentiment_Analysis_and_Opinion_Mining.html?hl=ar&id=xYhyEAAAQBAJ
[9] A. Thiab, L. Alawneh, and M. Al-Smadi, "Contextual emotion detection using ensemble deep learning," Computer speech & language, Jun. 01, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0885230823001237
[10] "MOODIFY: Tailored, Personal and Multifaceted AI Assistant for Young Adult Mental Health Issues," IEEE Conference Publication | IEEE Xplore, Dec. 08, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10455044
[11] R. L. Bowman et al., "Exploring how politeness impacts the user experience of chatbots for mental health support," International journal of human-computer studies, Nov. 01, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S1071581923001908
[12] D. Dietrich, B. Heller, B. Yang, Electric Music Collective, and Emc Education Services, Data science & big data analytics discovering, analyzing, visualizing and presenting data. Indianapolis Wiley, 2015.
[13] Webster, J.J. and Kit, C. (1992) 'Tokenization as the initial phase in NLP', Proceedings of the 14th conference on Computational linguistics - [Preprint]. doi:10.3115/992424.992434.
[14] Author links open overlay panelDan Ofer a et al. (2021) The language of proteins: NLP, Machine Learning & Protein Sequences, Computational and Structural Biotechnology Journal. Available at:

https://www.sciencedirect.com/science/article/pii/S2001037021000945 (Accessed: 03 October 2023).
[15] Alslaity, A. and Orji, R. (2022) 'Machine learning techniques for emotion detection and sentiment analysis: Current State, Challenges, and Future Directions', Behaviour & Information Technology, pp. 1–26. doi:10.1080/0144929x.2022.2156387.
[16] Guo, J. (2022) 'Deep Learning Approach to text analysis for human emotion detection from Big Data', Journal of Intelligent Systems, 31(1), pp. 113–126. doi:10.1515/jisys-2022-0001.
[17] Peng, S. et al. (2022) 'A survey on Deep Learning for textual emotion analysis in social networks', Digital Communications and Networks, 8(5), pp. 745–762. doi: 10.1016/j.dcan.2021.10.003.
[18] Kim, J. et al. (2020) A deep learning model for detecting mental illness from user content on social media, Nature News. Available at: https://www.nature.com/articles/s41598-020-68764-y (Accessed: 16 September 2023).
[19] Yadav S. Detecting presence of mental illness using NLP sentiment analysis. Available at: https://www.irjmets.com/uploadedfiles/paper/issue_6_june_2022/27031/final/fin_irjmets1656403378.pdf (Accessed: 16 September 2023).
[20] Journals & Books Online: Cambridge University Press, Cambridge Core. Available at: https://www.cambridge.org/core (Accessed: 16 September 2023).
[21] Better.me - Ai Mental Health Recommendations, Devpost. Available at: https://devpost.com/software/better-me-5w3lpj (Accessed: 16 September 2023).
[22] Using machine learning and thematic analysis methods to ... - IEEE xplore. Available at: https://ieeexplore.ieee.org/document/9115602 (Accessed: 16 September 2023).
[23] (No date) PESTLE technique a tool to identify external risks in ... - IRJET. Available at: https://www.irjet.net/archives/V3/i1/IRJET-V3I165.pdf (Accessed: 21 November 2023).
[24] Author links open overlay panelShailesh Hinduja a et al. (2022) Machine learning-based proactive social-sensor service for Mental Health Monitoring using Twitter data, International Journal of Information Management Data Insights. Available at: https://www.sciencedirect.com/science/article/pii/S2667096822000568 (Accessed: 16 September 2023).
[25] Nishant Nagururu started this project-Jan 29 (no date) Menty, Devpost. Available at: https://devpost.com/software/menty (Accessed: 24 September 2023).
[26] Author links open overlay panelMargarita Rodríguez-Ibánez a et al. (2023) A review on sentiment analysis from social media platforms, Expert Systems with Applications. Available at: https://www.sciencedirect.com/science/article/pii/S0957417423003639 (Accessed: 26 September 2023).
[27] (No date) Utilizing neural networks and linguistic metadata for ... - IEEE xplore. Available at: https://ieeexplore.ieee.org/document/8580405 (Accessed: 26 September 2023).
[28] Author links open overlay panelJessy E. Williams, Highlights•Scoping review of apps for anxiety and depression for children and young people.•Provides typology and

critical ecological analysis of mental health monitoring apps.•Effectiveness of mental health monitoring apps is uncertain.•Apps are not disc and AbstractThere is considerable concern about increasing rates of anxiety and depression among children and young people (CYP). Mental health technologies (2022) Mental Health Monitoring Apps for depression and anxiety in children and young people: A scoping review and critical ecological analysis, SocialScience&Medicine. Available at: https://www.sciencedirect.com/science/article/pii/S0277953622001083 (Accessed: 26 September 2023).
[29] Caldeira, C. et al. (2018a) Mobile apps for Mood Tracking: An analysis of features and user reviews, AMIA ... Annual Symposium proceedings. AMIA Symposium. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5977660/ (Accessed: 26 September 2023).
[30] Author links open overlay panelSalma Almouzini a et al. (2020) Detecting Arabic depressed users from Twitter data, Procedia Computer Science. Available at: https://www.sciencedirect.com/science/article/pii/S1877050919321465#keys0001
[31] Author links open overlay panelHaris Herdiansyah a, and AbstractMental health disorders remain a problem that always appears throughout the ages because its originators are related to everyday social phenomena that are always changing. One of the serious obstacles is cultural factors that view people with ment (2023) Their post tells the truth: Detecting social media users mental health issues with sentiment analysis, Procedia Computer Science. Available at: https://www.sciencedirect.com/science/article/pii/S1877050922022633
[32] Jackson, R.G. et al. (2017) Natural language processing to extract symptoms of severe mental illness from clinical text: The Clinical Record Interactive Search Comprehensive Data Extraction (Cris-code) project, BMJ Open. Available at: https://bmjopen.bmj.com/content/7/1/e012012.info (Accessed: 16 September 2023).
[33] Michelle Liu started this project Oct 25 (no date) Journly, Devpost. Available at: https://devpost.com/software/journly
[34] R. Buczyński, “MVT Architecture in Django: Introduction and Comparison with MVC,” Medium, Sep. 21, 2023. https://python.plainenglish.io/mvt-architecture-in-django-introduction-and-comparison-with-mvc-37cd617b542e
[35] (No date) Machine learning algorithms. Available at: https://balasahebtarle.files.wordpress.com/2020/01/machine-learning-algorithms_text-book.pdf (Accessed: 02 May 2024).
[36] Support Vector Machine (SVM) algorithm (2023) GeeksforGeeks. Available at: https://www.geeksforgeeks.org/support-vector-machine-algorithm/ (Accessed: 02 May 2024).
[37]“joangaes/depression · Datasets at Hugging Face,” huggingface.co. https://huggingface.co/datasets/joangaes/depression

Appendices

Appendix A: The questionnaire of the system requirements specification: (Questionnaire)

Appendix B: The suitable datasets for our system: (Emotion dataset & Depression dataset)

Appendix C: The interactive prototype of SoulGlow system: (Prototype)

Appendix D: The interactive prototype of SoulGlow system for Administrator: (Prototype)

Appendix E: The SoulGlow Source code (Source Code)

Appendix F: How to use the SoulGlow system

• To begin using the SoulGlow system, you'll need to create an account if you haven't already. Simply click on "Sign up" to access the registration form. If you've already registered, proceed by clicking on "Login" and enter your username and password.

• Once logged in, you'll land on the Explore page, where you'll find various options. Here, you can explore your daily affirmations, challenges, and book resources. Alternatively, you can start journaling by navigating to the Journal page from the navigation bar.

• In the Journal page, select your current mood and enter your journal text. You also have the option to upload an image to accompany your entry. Afterward, click on "Save" to store your journal in the database. You can access your previously entered journals on the History page.

• After entering one or more journals, you can analyze your emotions and depression trends by visiting the Profile page, where you'll find visualizations based on your journal entries.

• For account management, navigate to the Settings page. Here, you can update your information, access FAQs, and delete your account if needed.

• To learn more about SoulGlow, visit the "About Us" section on the landing page before logging in. Additionally, if you have feedback or inquiries, you can contact us by filling out the form in the "Contact Us" section on the landing page.

Appendix G: Code

Index.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>SoulGlow</title>

    <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">
    <link rel="stylesheet" href="{% static 'css/bootstrap.min.css' %}">
    <link rel="stylesheet" href= "{% static 'css/font-awesome.min.css' %}">
    <link rel="stylesheet" href="{% static 'css/aos.css' %}">
    <link rel="stylesheet" href="{% static 'css/owl.carousel.min.css' %}">
    <link rel="stylesheet" href="{% static 'css/owl.theme.default.min.css' %}">
    <!-- MAIN CSS -->
    <link rel="stylesheet" href="{% static 'css/templatemo-digital-trend.css' %}">

     <!-- MAIN CSS -->
     <link rel="stylesheet" href="css/templatemo-digital-trend.css">

     <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Roboto|Open+Sans">
     <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/font-
awesome.min.css">
     <!--<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"> -->

     <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
     <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>

<style>

     .affirmation {
        color: #000;
        font-size: 26px;
        font-weight: 300;
        text-align: center;
        text-transform: uppercase;
        position: relative;
        margin: 30px 0 60px;
    }   */
    p {
        color: #000;
        font-size: 14px;
```

```
}
.affirmation::after {
    content: "";
    width: 100px;
    position: absolute;
    margin: 0 auto;
    height: 4px;
    border-radius: 1px;
    background: #057a8d;
    left: 0;
    right: 0;
    bottom: -20px;
}*/
.carousel {
    margin: 50px auto;
    padding: 0 70px;
}
.carousel .item {
    color: #057a8d;
    min-height: 325px;
    text-align: center;
    overflow: hidden;
}
.carousel .thumb-wrapper {
    padding: 25px 15px;
    background: #fff;
    border-radius: 6px;
    text-align: center;
    position: relative;
    box-shadow: 0 2px 3px rgba(0,0,0,0.2);
}
.carousel .item .img-box {
    height: 120px;
    margin-bottom: 20px;
    width: 100%;
    position: relative;
}
.carousel .item img {
    max-width: 100%;
    max-height: 100%;
    display: inline-block;
    position: absolute;
    bottom: 0;
    margin: 0 auto;
    left: 0;
```

```
        right: 0;
    }
    .carousel .item h4 {
        font-size: 18px;
    }
    .carousel .item h4, .carousel .item p, .carousel .item ul {
        margin-bottom: 5px;
    }
     .carousel .thumb-content .btn {
        color: #057a8d;
        font-size: 11px;
        text-transform: uppercase;
        font-weight: bold;
        background: none;
        border: 1px solid #057a8d;
        padding: 6px 14px;
        margin-top: 5px;
        line-height: 16px;
        border-radius: 20px;
    }
    .carousel .thumb-content .btn:hover, .carousel .thumb-content .btn:focus {
        color: #fff;
        background: #057a8d;
        box-shadow: none;
    }
    .carousel .thumb-content .btn i {
        font-size: 14px;
        font-weight: bold;
        margin-left: 5px;
    }
    .carousel .carousel-control {
        height: 44px;
        width: 40px;
        background: #057a8d;
        margin: auto 0;
        border-radius: 4px;
        opacity: 0.8;
    }


    .star-rating li {
        padding: 0;
    }
    .star-rating i {
        font-size: 14px;
```

```
    color: #ffc000;
}


/* Styles for the close button */
.close {
  position: absolute;
  top: 10px;
  right: 10px;
  cursor: pointer;
}

.clickable-text {
  color: #6491A4; /* Text color */
  background-color: white; /* Background color */
  padding: 10px; /* Padding for better visual appearance */
  text-decoration: none; /* Remove default underline */
  text-decoration: underline;
  cursor: pointer; /* Change cursor on hover */
}
.overlay {
        display: none;
        position: fixed;
        top: 0;
        left: 0;
        width: 100%;
        height: 100%;
        background-color: rgba(0, 0, 0, 0.5); /* Semi-transparent black */
        z-index: 998; /* Ensure overlay is below the popup */
    }
    .popup {
 display: none; /* Hide the popup initially */
 position: fixed;
 top: 50%;
 left: 50%;
 transform: translate(-50%, -50%);
 max-width: 80%;
 max-height: 80%;
 overflow: auto; /* Add overflow to handle content overflow */
 background-color: white;
 padding: 20px;
 border-radius: 10px;
 box-shadow: 0 0 20px rgba(0, 0, 0, 0.3);
 z-index: 999;
}
```

```
    .username{
  font-family: "times new roman", times, serif;
  font-size: 20px;
  color: #27577B;
  font-style: italic;
  font-weight: bold;
  font-variant: normal;
  text-align: left;
  letter-spacing: -0.3px;
  line-height: 19px;
}
    </style>

<!--     ------------------------------------------------------------------    -->


</head>
<body>




    {% for message in messages %}
    <div class="alert alert-{{ message.tags }} alert-dismissible fade show" role="alert">
        {{ message }}
        <button type="button" class="close" data-dismiss="alert" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
   {% endfor %}

    {% if user.is_authenticated %}
        <!-- LOGGED IN USER-->



     <!-- MENU BAR -->
     <nav class="navbar navbar-expand-lg" style="background-color: #6491A4;">
        <div class="container">
            <a class="navbar-brand" href="index.html">
              <!-- <i class="fa fa-line-chart"></i>       MENU BAR -->
              <img src="{% static 'images/logo.png' %}"class="img-fluid" alt="Logo" width="80"
height="40">
              SoulGlow
            </a>
```

```
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarNav" aria-controls="navbarNav" aria-expanded="false"
                aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>

            <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav ml-auto">

                    <li class="nav-item">
                        <a href="{% url 'home' %}"; class="nav-link smoothScroll">Explore</a>
                    </li>
                    <li class="nav-item">
                        <a href="{% url 'journal' %}";  class="nav-link smoothScroll">Journal</a>
                    </li>
                    <li class="nav-item">
                        <a href="{% url 'prevjournal' %}";  class="nav-link smoothScroll">History</a>
                    </li>
                    <li class="nav-item">
                        <a href="{% url 'profile' %}"; class="nav-link smoothScroll">Profile</a>
                    </li>
                    <li class="nav-item">
                        <a href="{% url 'settings' %}"; class="nav-link">Settings</a>
                    </li>
                    <li class="nav-item">
                         <a href="/signout" class="nav-link contact">Sign out</a>
                     </li>

                </ul>
            </div>
        </div>
    </nav>

<!-- affirmation -->
<section class="hero hero-bg d-flex justify-content-center align-items-center">
    <div class="container">
         <div class="row">

            <div class="col-lg-6 col-md-10 col-12 d-flex flex-column justify-content-center align-
items-center">
                 <div class="hero-text">
                     <h1 id="affirmation" class="affirmation" style="color: #27577B;"></h1><br>
                     <p id="datetime" style="text-align: center; color:#27577B;" class="mb-0" data-
aos="fade-up"></p><br><br>
```

```
                </div>
            </div>

            <div class="col-lg-6 col-12">
              <div data-aos="fade-up" data-aos-delay="300">

                <img src="{% static 'images/explorepic.png' %}" class="img-fluid" alt="working girl"
width="300" height="250" style="margin-left: 140px;">
              </div>
            </div>

        </div>
    </div>
</section>




  <!-- CHALLENGES -->

<section class="testimonial section-padding">
    <div class="container card border-0 rounded-4 shadow-lg overflow-hidden" style="padding: 30px;
background-color: #A9BB87;">
        <div class="row align-items-center">
            <div class="col-lg-6 col-md-7 col-12">
                <div class="row">
                    <div class="col-md-10 offset-md-1 text-center" data-aos="fade-up" data-aos-
delay="300">
                        <div class="about-info">
                            <p class="mb-0" data-aos="fade-up" style="font-size: 27px;">{{
user.username|capfirst }} Your <strong style="color: #27577B;">Challenge</strong> For Today is
:</p><br>
                            <h1 id="challenge" data-aos="fade-up" style="font-size: 35px; color:
#27577B;"></h1>
                        </div>
                    </div>
                </div>
            </div>
            <div class="col-lg-6 col-md-5 col-12 d-flex justify-content-center align-items-center">
                <div data-aos="fade-up">
                    <img src="{% static 'images/exp1.png' %}" class="img-fluid" alt="website"
style="width: 400px;">
                </div>
            </div>
```

```
        </div>
    </div>
</section>




 <!-- Details of your mental state -->
    <section class="testimonial section-padding" style="padding: 20%;">
        <div class="row align-items-center">
            <div class="col-lg-6 col-md-7 col-12">
                <div class="row">
                    <div class="col-md-10 offset-md-1 text-center" data-aos="fade-up" data-aos-
delay="300">
                        <h2 class="mb-4" style="color: #27577B;">Here are some resources that could
help you with your mental state</h2>
                        <a href="{% url 'profile' %}" class="custom-btn btn-bg btn mt-3" data-
aos="fade-up" data-aos-delay="100" style="background-color: #27577B; color: aliceblue;">Details of
your mental state</a>
                    </div>
                </div>
            </div>
            <div class="col-lg-6 col-md-5 col-12">
                <div data-aos="fade-up">
                    <img src="{% static 'images/heart.png' %}"  class="img-fluid" alt="website"
style="margin-top: 10px;">
                </div>
            </div>
        </div>
    </div>
</section>


 <!-- RESOURCES -->

   <section>
    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <h2 class="affirmation">Resources</h2>
                <div id="myCarousel" class="carousel slide" data-ride="carousel" data-interval="0">
                <!-- IN FUTURE -->
                <ol class="carousel-indicators">
                    <li data-target="#myCarousel" data-slide-to="0" class="active"></li>
                    <li data-target="#myCarousel" data-slide-to="1"></li>
```

```
                    <li data-target="#myCarousel" data-slide-to="2"></li>
                </ol>
                <!-- BOOKS -->
                <div class="carousel-inner">
                    <div class="item active">
                        <div class="row">
                            <div class="col-sm-3">
                                <div class="thumb-wrapper">

                                    <div class="img-box"> <!--1-->
                                        <img src="{% static 'images/atomic habits.jpg' %}" class="img-
responsive" alt="">
                                    </div>
                                    <div class="thumb-content">
                                        <h4>Atomic Habits</h4><br>
                                        <div class="star-rating">
                                            <ul class="list-inline">
                                                <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                                <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                                <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                                <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                                <li class="list-inline-item"><i class="fa fa-star-
o"></i></li>
                                            </ul>
                                        </div>

                                        <span  class="clickable-text"
onclick="showDetails('0735211299')">Show Details</span><br>

                                        <a
href="https://www.amazon.sa/dp/0735211299?_encoding=UTF8&psc=1&ref_=cm_sw_r_cp_ud_dp_FNKVG3A46Y4AJWYTN
Z0H" class="btn btn-primary">Buy</a>
                                    </div>
                                </div>
                            </div>
                            <div class="col-sm-3">
                                <div class="thumb-wrapper">

                                    <div class="img-box"> <!--2-->
                                        <img src="{% static 'images/b2.jpg' %}"class="img-responsive"
alt="">
```

```
                                    </div>

                                    <div class="thumb-content">
                                        <h4>The emotion
code</code></h4><br>
                                        <div class="star-rating">
                                            <ul class="list-inline">
                                                <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                                <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                                <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                                <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                                <li class="list-inline-item"><i class="fa fa-star-
o"></i></li>
                                            </ul>
                                        </div>
                                        <span  class="clickable-text"
onclick="showDetails('1785042874')">Show Details</span><br>

                                        <a href="https://amzn.eu/d/gsYPMgo" class="btn btn-
primary">Buy</a>
                                    </div>
                                </div>
                            </div>
                            <div class="col-sm-3">
                                <div class="thumb-wrapper">

                                    <div class="img-box"> <!--3-->
                                        <img src="{% static 'images/b3.jpg' %}" class="img-responsive"
alt="">
                                    </div>
                                    <div class="thumb-content">
                                        <h4>Do it for yourself</h4><br>
                                        <div class="star-rating">
                                            <ul class="list-inline">
                                                <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                                <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                                <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                                <li class="list-inline-item"><i class="fa fa-
```

```
star"></i></li>
                                        <li class="list-inline-item"><i class="fa fa-star-
half-o"></i></li>
                                    </ul>
                                </div>
                                <span  class="clickable-text"
onclick="showDetails('1419743465')">Show Details</span><br>

                                <a href="https://amzn.eu/d/66ZRkY8" class="btn btn-
primary">Buy</a>
                            </div>
                        </div>
                    </div>
                    <div class="col-sm-3">
                        <div class="thumb-wrapper">

                            <div class="img-box"> <!--4-->
                                <img src="{% static 'images/book2.jpg' %}" class="img-
responsive" alt="">
                            </div>
                            <div class="thumb-content">
                                <h4>Little book of Phsycology</h4>
                                <div class="star-rating">
                                    <ul class="list-inline">
                                        <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                        <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                        <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                        <li class="list-inline-item"><i class="fa fa-star-
o"></i></li>
                                        <li class="list-inline-item"><i class="fa fa-star-
o"></i></li>
                                    </ul>
                                </div>
                                <span class="clickable-text"
onclick="showDetails('178685807X')">Show Details</span><br>

                                <a href="https://amzn.eu/d/ee8bGaE" class="btn btn-
primary">Buy</a>
                            </div>
                        </div><br><br>
                    </div>
                </div>
```

```
                    </div>
                    <div class="item">
                        <div class="row">
                            <div class="col-sm-3">
                                <div class="thumb-wrapper">

                                    <div class="img-box"> <!--5-->
                                        <img src="{% static 'images/b5.jpg' %}"class="img-responsive"
alt="">
                                    </div>
                                    <div class="thumb-content">
                                        <h4>Heal your body</h4><br>

                                        <div class="star-rating">
                                            <ul class="list-inline">
                                                <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                                <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                                <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                                <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                                <li class="list-inline-item"><i class="fa fa-star-
o"></i></li>
                                            </ul>
                                        </div>
                                        <span class="clickable-text"
onclick="showDetails('9788190416986')">Show Details</span><br>

                                        <a
href="https://www.amazon.sa/dp/8190416987?_encoding=UTF8&psc=1&ref_=cm_sw_r_cp_ud_dp_GGVD130ERZ5EP857C
2JY" class="btn btn-primary">Buy</a>
                                    </div>
                                </div>
                            </div>
                            <div class="col-sm-3">
                                <div class="thumb-wrapper">

                                    <div class="img-box"> <!--6-->
                                        <img src="{% static 'images/b6.jpg' %}" class="img-responsive"
alt="">
                                    </div>
                                    <div class="thumb-content">
                                        <h4>Own your own mind</h4><br>
```

```
                                    <div class="star-rating">
                                        <ul class="list-inline">
                                            <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                            <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                            <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                            <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                            <li class="list-inline-item"><i class="fa fa-star-
half-o"></i></li>
                                        </ul>
                                    </div>
                                    <span  class="clickable-text"
onclick="showDetails('9780143111528')">Show Details</span><br>

                                    <a href="https://amzn.eu/d/8NgGbA5" class="btn btn-
primary">Buy</a>
                                </div>
                            </div>
                        </div>
                        <div class="col-sm-3">
                            <div class="thumb-wrapper">

                                <div class="img-box"> <!--7-->
                                    <img src="{% static 'images/b7.jpg' %}" class="img-responsive"
alt="">
                                </div>
                                <div class="thumb-content">
                                    <h4>Mental wellness</h4><br>

                                    <div class="star-rating">
                                        <ul class="list-inline">
                                            <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                            <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                            <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                            <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                            <li class="list-inline-item"><i class="fa fa-star-
o"></i></li>
```

```
                                        </ul>
                                    </div>
                                    <span  class="clickable-text"
onclick="showDetails('9780241480069')">Show Details</span><br>

                                    <a href="https://amzn.eu/d/7gl2V7P" class="btn btn-
primary">Buy</a>
                                </div>
                            </div>
                        </div>
                        <div class="col-sm-3">
                            <div class="thumb-wrapper">

                                <div class="img-box"> <!--8-->
                                    <img src="{% static 'images/b8.jpg' %}" class="img-responsive"
alt="">
                                </div>
                                <div class="thumb-content">
                                    <h4>Understand social anxiety</h4>

                                    <div class="star-rating">
                                        <ul class="list-inline">
                                            <li class="list-inline-item"><i class="fa fa-
star"></i></li>

                                            <li class="list-inline-item"><i class="fa fa-
star"></i></li>

                                            <li class="list-inline-item"><i class="fa fa-
star"></i></li>

                                            <li class="list-inline-item"><i class="fa fa-
star"></i></li>

                                            <li class="list-inline-item"><i class="fa fa-star-
o"></i></li>

                                        </ul>
                                    </div>
                                    <span  class="clickable-text"
onclick="showDetails('9781800074965')">Show Details</span><br>

                                    <a href="https://amzn.eu/d/fot9WxG" class="btn btn-
primary">Buy</a>
                                </div>
                            </div><br><br>
                        </div>
                    </div>
                </div>
                <div class="item">
```

```
                    <div class="row">
                        <div class="col-sm-3">
                            <div class="thumb-wrapper">

                                <div class="img-box"> <!--10-->
                                    <img src="{% static 'images/b9.jpg' %}" class="img-responsive"
alt="">
                                </div>
                                <div class="thumb-content">
                                    <h4>Be kind to your mind</h4><br>

                                    <div class="star-rating">
                                        <ul class="list-inline">
                                            <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                            <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                            <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                            <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                            <li class="list-inline-item"><i class="fa fa-star-
o"></i></li>
                                        </ul>
                                    </div>
                                    <span  class="clickable-text"
onclick="showDetails('9781787832565')">Show Details</span><br>

                                    <a href="https://amzn.eu/d/2xiU4xv" class="btn btn-primary"
>Buy</a>
                                </div>
                            </div>
                        </div>
                        <div class="col-sm-3">
                            <div class="thumb-wrapper">

                                <div class="img-box"> <!--11-->
                                    <img src="{% static 'images/b12.jpg' %}" class="img-
responsive" alt="">
                                </div>
                                <div class="thumb-content">
                                    <h4>Master your emotions</h4>

                                    <div class="star-rating">
                                        <ul class="list-inline">
```

```
                                        <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                        <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                        <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                        <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                        <li class="list-inline-item"><i class="fa fa-star-
o"></i></li>
                                    </ul>
                                </div>
                                <span  class="clickable-text"
onclick="showDetails('1981089152')">Show Details</span><br>

                                <a
href="https://www.amazon.sa/dp/1981089152?ref_=cm_sw_r_cp_ud_dp_QT3SZ65FEJS8EQYATFB2" class="btn btn-
primary">Buy</a>
                            </div>
                        </div>
                    </div>
                    <div class="col-sm-3">
                        <div class="thumb-wrapper">

                            <div class="img-box"> <!--12-->
                                <img src="{% static 'images/b111.jpg' %}" class="img-
responsive" alt="">
                            </div>
                            <div class="thumb-content">
                                <h4>Brain energy</h4><br>

                                <div class="star-rating">
                                    <ul class="list-inline">
                                        <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                        <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                        <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                        <li class="list-inline-item"><i class="fa fa-
star"></i></li>
                                        <li class="list-inline-item"><i class="fa fa-star-
half-o"></i></li>
                                    </ul>
                                </div>
```

```
                                    <span class="clickable-text"
onclick="showDetails('1637741588')">Show Details</span><br>

                                    <a href="https://amzn.eu/d/adZRygK" class="btn btn-
primary">Buy</a>
                                </div>
                            </div>
                        </div>
                        <div class="col-sm-3">
                            <div class="thumb-wrapper">

                                <div class="img-box">
                                    <img src="{% static 'images/b10.jpg' %}" class="img-
responsive" alt="">
                                </div>
                                <div class="thumb-content">
                                    <h4>Ikigai</h4><br>

                                    <div class="star-rating">
                                        <ul class="list-inline">
                                            <li class="list-inline-item"><i class="fa fa-
star"></i></li>

                                            <li class="list-inline-item"><i class="fa fa-
star"></i></li>

                                            <li class="list-inline-item"><i class="fa fa-
star"></i></li>

                                            <li class="list-inline-item"><i class="fa fa-
star"></i></li>

                                            <li class="list-inline-item"><i class="fa fa-star-
o"></i></li>

                                        </ul>
                                    </div>
                                    <span  class="clickable-text"
onclick="showDetails('0143130722')">Show Details</span><br>

                                    <a href="https://amzn.eu/d/aaaDqLT" class="btn btn-primary"
onclick="showDetails()">Buy</a>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>

            </div>
```

```
        </div>
    </div>
    </section>


    <section class="about section-padding pb-0">   </section>
    <!-- Pop up -->

    <div class="overlay" id="overlay"></div>
    <div id="myModal" class="popup">
        <span class="close" onclick="closeModal()">&times;</span>
        <h2 class="affirmation">Book Details</h2>
        <div id="bookDetails"></div>
      </div>



         <!-- FOOTER -->

    <footer class="site-footer" style="background-color: #6491A4; padding: 30px 0">
        <div class="container">
          <div class="row">

            <div class="col-lg-5 mx-lg-auto col-md-8 col-10">
              <h1 class="text-white" style="font-size: 30px;"data-aos="fade-up" data-aos-
delay="100">Elevate your well being with <strong>SoulGlow</strong></h1>
              <img src="{% static 'images/Logo.png' %}"  class="img-fluid" alt="Logo" width="80"
height="40">
            </div>
            <div class="col-lg-4 mx-lg-auto text-center col-md-8 col-12" data-aos="fade-up" data-aos-
delay="100">
              <p class="copyright-text">Copyright &copy; 2023 Made by SoulGlow Team
              <br>
            </div>
          </div>
        </div>
      </footer>
```

```
 <!--
_____
_____
_____-->

    {%else%}
         <!-- MAINPAGE (LANDINGPAGE) -->



         <!-- MENU BAR -->
        <nav class="navbar navbar-expand-lg" style="background-color: #6491A4;">
            <div class="container">
                <a class="navbar-brand" href="#">
                  <!-- <i class="fa fa-line-chart"></i>       MENU BAR -->
                  <img src="{% static 'images/Logo.png' %}" class="img-fluid" alt="Logo" width="80"
height="40">
                  SoulGlow
                </a>

                <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarNav" aria-controls="navbarNav" aria-expanded="false"
                    aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>

                <div class="collapse navbar-collapse" id="navbarNav">
                    <ul class="navbar-nav ml-auto">
                        <li class="nav-item">
                            <a href="#Features" class="nav-link smoothScroll">Features</a>
                        </li>
                        <li class="nav-item">
                            <a href="#about" class="nav-link smoothScroll">About</a>
                        </li>
                        <li class="nav-item">
                            <a href="#contact" class="nav-link">Contact us</a>
                        </li>
                        <li class="nav-item">
                            <a href="/signup" class="nav-link contact">Sign up</a>
                        </li>
                        <li class="nav-item">
                             <a href="/signin" class="nav-link contact">Sign in</a>
                         </li>
                    </ul>
                </div>
```

```
            </div>
        </nav>


        <!-- HERO -->
        <section class="hero hero-bg d-flex justify-content-center align-items-center">
                <div class="container">
                    <div class="row">

                        <div class="col-lg-6 col-md-10 col-12 d-flex flex-column justify-content-
center align-items-center">
                            <div class="hero-text">

                                <h1 class="text-white" data-aos="fade-up">A Healthy Mind Is An
Asset</h1>

                            </div>
                        </div>
                        <div class="col-lg-6 col-12">
                          <div class="hero-image" data-aos="fade-up" data-aos-delay="300">
                            <img src="{% static 'images/pic.png' %}" class="img-fluid"
alt="working girl">
                          </div>
                        </div>
                    </div>
                </div>
        </section>


        <!-- What we do  -->
        <section class="about section-padding pb-0" >
            <div class="container">
                        <div class="row g-0">
                          <div class="col-sm-6 d-none d-sm-block bg-image">
                           <div class="text-center p-4">
                    <div class="col-lg-7 mx-auto col-md-10 col-12">
                        <div class="about-info">
                            <h1 class="mb-4" data-aos="fade-up" style="color: #27577B; font-
weight: bold;">Embrace your journey with <strong  style="font-weight: bolder;">SoulGlow</strong></h2>
                        </div>
                    </div>
                </div>
            </div>
            <div class="col-sm-6 p-4">
                            <ul style="color: #27577B; list-style:
```

```
url('../static/images/check.svg');">
                                    <li class="mb-0" data-aos="fade-up"  >
                                        <h3>
                                            Journal whenever you want
                                        </h3>
                                        <p>
                                            Write in your online journal at any time you feel
like it, without any specific schedule or restriction with our Prompts.
                                        </p></li>
                                    <li class="mb-0" data-aos="fade-up">  <h3>
                                        Track your mood
                                                                        </h3>
                                    <p>
                                        Monitor and record your emotional state or feeling
regularly to gain insights into your mood patterns and emotional well-
being.                                  </p></li>
                                    <li class="mb-0" data-aos="fade-up">  <h3>
                                        Focus on self-
improvement                                  </h3>
                                    <p>
                                        Find the right resources, activeties and therapy clinics
to help you to be a better you!                                     </p></li>
                                </ul>
                            </div>
                        </div>
                </div>
            </section>


            <div class="container" id="Features">

              <!-- OUR FEATURES -->
              <h1 class="my-4 text-center" style="color: #27577B;">Our Features
              </h1>

              <div class="row">
                <div class="col-lg-4 col-sm-6 mb-4" >
                  <div class="card h-100 d-flex flex-column justify-content-center" style="background-
color: #FBD3DB;">
                    <img class="card-img-top" style="height: 180px; padding-top: 20px; " src="{% static
'images/yellow, pink and beige books lying on top of each other.png' %}" alt="">
                    <div class="card-body text-center">
                      <h3 class="card-title" style="color: #27577B;">
                          Mental Health Improvement resources
                      </h3>
```

```
                </div>
              </div>
            </div>
            <div class="col-lg-4 col-sm-6 mb-4">
              <div class="card h-100 d-flex flex-column justify-content-center" style="background-
color: #FBD3DB;">
                <img class="card-img-top " style="height: 200px; padding: 20px;" src= "{% static
'images/green typewriter.png' %}" alt="">
                <div class="card-body text-center">
                  <h3 class="card-title" style="color: #27577B;">
                      Journaling
                              </h3>
                </div>
              </div>
            </div>
            <div class="col-lg-4 col-sm-6 mb-4">
             <div class="card h-100 d-flex flex-column justify-content-center" style="background-
color: #FBD3DB;">
                <img class="card-img-top" style="max-height: 200px; max-width: 100%; padding:
20px;"  src= "{% static 'images/data analysis.png' %}" alt="">
                <div class="card-body text-center">
                    <h3 class="card-title" style="color: #27577B;">Analyze Your Emotions</h3>
                </div>
             </div>
          </div>
           </div>
         </div>


        <!-- CALL FOR ACTION -->
        <section class="testimonial section-padding">
         <div class="container card border-0 rounded-4 shadow-lg overflow-hidden" style="background-
color: #A9BB87; padding: 30px;">
            <div class="row align-items-center">
                <div class="col-lg-6 col-md-7 col-12">
                    <div class="row">
                        <div class="col-md-10 offset-md-1 text-center" data-aos="fade-up" data-aos-
delay="300">
                            <h2 class="mb-4" style="color: #27577B;">Empower your mind with
us<br><span style="font-weight: bold;">Starting Now!</span><br></h2>
                            <a href="/signup" class="custom-btn btn-bg btn mt-3" data-aos="fade-up"
data-aos-delay="100" style="background-color: #27577B; color: aliceblue;">Get Started</a>
                        </div>
                    </div>
                </div>
```

```
                <div class="col-lg-6 col-md-5 col-12">
                    <div data-aos="fade-up">
                        <img src="{% static 'images/Pink brain.png' %}"  class="img-fluid"
alt="website" style="margin-top: 10px;">
                    </div>
                </div>
            </div>
        </div>
    </section>
        <!-- ABOUT US -->
        <section class="project section-padding" id="about">
            <div class="container-fluid">
                <div class="row">
                    <div class="col-lg-7 mx-auto col-md-10 col-12">
                        <div class="about-info">

                            <h2 class="mb-4" data-aos="fade-up" style="color:white">About
Us</h2>

                            <p class="mb-0" data-aos="fade-up" style="color:white">Meet the
innovative minds behind our online journaling platform!
    We are a team of passionate CS students from PNU, we designed a unique online journaling website.
Through the power of Natural Language Processing (NLP), our platform not only allows users to express
their thoughts but also analyzes their journal entries, providing valuable insights into their mental
health. We believe in the major potential of self-reflection, and our user-friendly platform aims to
empower individuals on their journey to emotional well-being. Join us in embracing the future of
mental health awareness through technology.
                            </p>
                        </div>
                </div>
            </div>
        </section>
            <!-- SOULGLOW TEAM -->

        <section class="about section-padding pb-0" id="about">
            <div class="container">
                <div class="row">
                        <div class="hero-image" data-aos="fade-up" data-aos-delay="200">
                         <img src="{% static 'images/team.png' %}"  class="img-fluid"
alt="office" width= 800 height= 100>
                        </div>
                    </div>
                </div>
            </div>
        </section>
```

```
            <!-- CONTACTUS -->

        <section id="contact">
           <div class="overflow-hidden my-5">
               <div class="row justify-content-center">
                   <div class="">
                       <div class="card border-0 rounded-3 shadow-lg overflow-hidden">
                           <div class="">
                               <div class="card-body" style="background-color:#B2CCDB">
                                   <div class="row g-0">
                                       <div class="col-sm-6 d-none d-sm-block bg-image">
                                           <div class="text-center p-4">
                                               <div class="h3 fw-light" style="color:#27577B">Contact
Us</div>
                                               <p class="mb-4" style="color:#27577B">We would love to
hear from you! If you have any questions, feedback, or suggestions, please don't hesitate to get in
touch.</p>
                                               <img src="{% static 'images/woman talking with
chatbot.png' %}" style="width: 200px;" alt="">
                                           </div>
                                       </div>
                                       <div class="col-sm-6 p-4">
                                           <div class="text-center">
                                               <div class="h3 fw-light" style="color:#27577B">Feel
Free To Reach out</div>
                                           </div>
                                           <form method="post" action="{% url 'contact_submit' %}">
                                               {% csrf_token %}
                                               <div class="form-floating mb-3">
                                                   <input class="form-control" name="name" id="name"
type="text" placeholder="Name" required />
                                               </div>
                                               <div class="form-floating mb-3">
                                                   <input class="form-control" name="email"
id="emailAddress" type="email" placeholder="Email Address" required />
                                               </div>
                                               <div class="form-floating mb-3">
                                                   <textarea class="form-control" name="message"
id="message" style="height: 10rem;" placeholder="Message" required></textarea>
                                               </div>
                                               <div class="d-grid">
                                                   <button class="btn btn-primary" style="background-
```

```
color:#27577B;  width: 100%" type="submit">Submit</button>
                                            </div>
                                        </form>
                                        {% if thanks %}
                                        <div>{{ thanks }}</div>
                                    {% endif %}
                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </section>


             <!-- FOOTER -->

        <footer class="site-footer" style="background-color: #6491A4; padding: 30px 0">
          <div class="container">
            <div class="row">
              <div class="col-lg-5 mx-lg-auto col-md-8 col-10">
                <h1 class="text-white" style="font-size: 30px;"data-aos="fade-up" data-aos-
delay="100">Elevate your well being with <strong>SoulGlow</strong></h1>
                <img src="{% static 'images/Logo.png' %}"  class="img-fluid" alt="Logo" width="80"
height="40">
              </div>
              <div class="col-lg-4 mx-lg-auto text-center col-md-8 col-12" data-aos="fade-up" data-
aos-delay="100">
                <p class="copyright-text">Copyright &copy; 2023 Made by SoulGlow Team
                <br>
              </div>
            </div>
          </div>
        </footer>


    {% endif %}

         <!-- SCRIPTS -->
         <script src= "{% static 'js/jquery.min.js' %}" ></script>
         <script src="{% static 'js/bootstrap.min.js' %}"></script>
         <script src="{% static 'js/aos.js' %}" ></script>
         <script src="{% static 'js/owl.carousel.min.js' %}" ></script>
```

```
    <script src="{% static 'js/smoothscroll.js' %}" ></script>
    <script src="{% static 'js/custom.js' %}"></script>
    {% if scroll_to_contact %}
    <script>
        document.addEventListener("DOMContentLoaded", function() {
            document.querySelector("#contact").scrollIntoView({
                behavior: 'smooth',
                block: 'start', // Scroll to the top of the target element

            });
        });
    </script>
    {% endif %}

<script>

    function showDetails(isbn) {
      // Fetch book details from Google Books API using the provided ISBN
      fetch(`https://www.googleapis.com/books/v1/volumes?q=isbn:${isbn}`)
        .then(response => response.json())
        .then(data => {
          // Extract relevant details from the API response
          const bookDetails = {
            title: data.items[0].volumeInfo.title,
            author: data.items[0].volumeInfo.authors.join(', '),
            description: data.items[0].volumeInfo.description || 'No description available.'
          };

          // Display details in the modal
          document.getElementById('bookDetails').innerHTML = `
            <p><strong>Title:</strong> ${bookDetails.title}</p>
            <p><strong>Author:</strong> ${bookDetails.author}</p>
            <p><strong>Description:</strong> ${bookDetails.description}</p>
          `;

          // Show the modal and center it on the page
          const modal = document.getElementById('myModal');
          modal.style.display = 'block';

  // Disable scrolling on the body
          document.body.style.overflow = 'hidden';
          document.getElementById("overlay").style.display = "block";
        })
        .catch(error => {
          console.error('Error fetching book details:', error);
```

```
            });
        }

          function closeModal() {
            // Close the modal
            document.getElementById("overlay").style.display = "none";
            document.getElementById('myModal').style.display = 'none';
  // Enable scrolling on the body
  document.body.style.overflow = 'auto';

          }
            // Get current date
            var now = new Date();
            var date = now.toLocaleDateString(); // Extract only the date part

            // Insert date into HTML
            document.getElementById("datetime").innerHTML = date;


//CHALLENGES
        const challenge = document.getElementById("challenge");
        const api_url2 = "http://www.boredapi.com/api/activity/";

        async function getChallenge(url) {
          try {
            const response = await fetch(url);
            const data = await response.json();

            // Assuming you want to display the activity in the "challenge" element
            challenge.innerHTML = data.activity;
          } catch (error) {
            console.error('Error fetching activity:', error);
          }
        }

        getChallenge(api_url2);

//AFFIRMATION

    const affirmationElement = document.getElementById("affirmation");
    const api_url = "{% url 'get_affirmation' %}";

    async function getAffirmation(url) {
     try {
         const response = await fetch(url);
```

```
        if (!response.ok) {
            throw new Error(`HTTP error! Status: ${response.status}`);
        }
        const data = await response.json();
        console.log(data);
        affirmationElement.innerHTML = data.affirmation;
    } catch (error) {
        console.error('Error fetching affirmation:', error);
    }
   }

   getAffirmation(api_url);
       </script>

</body>
</html>
```

Journal.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>

     <title>SoulGlow</title>

<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=Edge">
<meta name="description" content="">
<meta name="keywords" content="">
<meta name="author" content="">
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">
<link rel="stylesheet" href="{% static 'css/bootstrap.min.css' %}">
<link rel="stylesheet" href= "{% static 'css/font-awesome.min.css' %}">
<link rel="stylesheet" href="{% static 'css/aos.css' %}">
<link rel="stylesheet" href="{% static 'css/owl.carousel.min.css' %}">
<link rel="stylesheet" href="{% static 'css/owl.theme.default.min.css' %}">
<!-- MAIN CSS -->
<link rel="stylesheet" href="{% static 'css/templatemo-digital-trend.css' %}">

<style>
    .Radiogroup {
        display: flex;
        flex-direction: column;
        height: 100%;
    }

    .modd {
        height: 100%;
    }

    .card-body .row .col-md-6 {
        padding-right: 15px;
        padding-left: 15px;

    height: 100%;
    }

    .card-body .row {
        margin-right: -15px;
        margin-left: -15px;
            height: 100%;
```

```
    }

.overlay {
            display: none;
            position: fixed;
            top: 0;
            left: 0;
            width: 100%;
            height: 100%;
            background-color: rgba(0, 0, 0, 0.5); /* Semi-transparent black */
            z-index: 998; /* Ensure overlay is below the popup */
        }
        .popup {
     display: none; /* Hide the popup initially */
     position: fixed;
     top: 50%;
     left: 50%;
     transform: translate(-50%, -50%);
     max-width: 80%;
     max-height: 80%;
     overflow: auto; /* Add overflow to handle content overflow */
     background-color: white;
     padding: 20px;
     border-radius: 10px;
     box-shadow: 0 0 20px rgba(0, 0, 0, 0.3);
     z-index: 9999;
    }
</style>
</head>
<body style="background: linear-gradient( rgb(224, 230, 237) 0%, rgba(228,242,248,1) 100%); color:
#6491A4;" >

<!-- MENU BAR -->
<nav class="navbar navbar-expand-lg"  style="background-color: #6491A4;">
   <div class="container">
       <a class="navbar-brand" href="index.html">
         <img src="{% static 'images/Logo.png' %}" class="img-fluid" alt="Logo" width="80"
height="40">
         SoulGlow
       </a>
       <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav"
aria-controls="navbarNav" aria-expanded="false"
           aria-label="Toggle navigation">
           <span class="navbar-toggler-icon"></span>
       </button>
```

```
      <div class="collapse navbar-collapse" id="navbarNav">
          <ul class="navbar-nav ml-auto">
           <li class="nav-item">
               <a href="{% url 'home' %}"; class="nav-link smoothScroll">Explore</a>
           </li>
           <li class="nav-item">
               <a href="{% url 'journal' %}";  class="nav-link smoothScroll">Journal</a>
           </li>
           <li class="nav-item">
               <a href="{% url 'prevjournal' %}";  class="nav-link smoothScroll">History</a>
           </li>
           <li class="nav-item">
               <a href="{% url 'profile' %}"; class="nav-link smoothScroll">Profile</a>
           </li>
           <li class="nav-item">
               <a href="{% url 'settings' %}"; class="nav-link">Settings</a>
           </li>
           <li class="nav-item">
                <a href="/signout" class="nav-link contact">Sign out</a>
            </li>
          </ul>
      </div>
   </div>
</nav>

<div class="container">
    <!-- Two-column layout -->
    <div class="row" style="padding-top: 10px;">
        <!-- Left column -->
        <div class="col-md-8">
            <!-- Check-in questions card -->
            <div class="card">
                <div class="card-body">
                    <!-- Card header -->
                    <div class="card-header">
                        <h3 class="card-title">Check-in questions</h3>
                    </div>
                    <div class="card-body">
                        <div class="row">
                            <div class="col-md-6">
                                <!-- Mood form -->
                                <div class="Radiogroup">
                                    <div class="modd">
                                        <form id="moodForm">
                                            <fieldset>
```

```
                                        <!-- Mood options -->
                                          <legend>What is your current mood? </legend><br>
                                        <div>
                                            <input type="radio" id="Happy" name="mood"
value="Happy" required>
                                            <label for="Happy">Happy </label>
                                        </div>
                                        <div>
                                            <input type="radio" id="Sad" name="mood"
value="Sad" required>
                                            <label for="Sad">Sad </label>
                                        </div>
                                        <div>
                                            <input type="radio" id="Anxious" name="mood"
value="Anxious" required>
                                            <label for="Anxious">Anxious </label>
                                        </div>
                                        <div>
                                            <input type="radio" id="Angry" name="mood"
value="Angry" required>
                                            <label for="Angry">Angry </label>
                                        </div>
                                        <div>
                                            <input type="radio" id="Calm" name="mood"
value="Calm" required>
                                            <label for="Calm">Calm </label>
                                        </div><br><br>

                                    </fieldset>
                                </form>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
        <!-- Error message display area -->
        <div id="errorMessage" style="color: red; display: none;"></div>

        <!-- Right column -->
        <div class="col-md-4">
            <!-- Date display card -->
            <div class="card">
```

```
                <div class="card-body">
                    <div class="card-header">
                        <!-- Current date display -->
                        <h3 class="card-title" style="text-align: center;" id="datetime"></h3>
                    </div>
                    <!-- Image upload section -->
                    <div class="card-body d-flex flex-column">
                        <form method="post" enctype="multipart/form-data">
                            <!-- Selected image display -->
                            <div class="mb-3 d-flex justify-content-center align-items-center">
                                <img id="selectedImage" src="{% static 'images/photo.png' %}"
alt="example placeholder" style="width: 200px; height: 210px;" />
                            </div>
                            <!-- Image upload button -->
                            <div class="d-flex justify-content-center">
                                <div class="btn-group">
                                    <div class="btn btn-primary btn-rounded" style="background-color:
#396389;">
                                        <label class="form-label text-white m-1"
for="customFile1">Choose file</label>
                                        <input type="file" class="form-control d-none"
id="customFile1" onchange="displaySelectedImage(event, 'selectedImage')" />
                                    </div>
                                    <!-- Image delete button -->
                                    <button class="btn btn-danger btn-rounded"
onclick="deleteSelectedImage()">Delete</button>
                                </div>
                            </div>
                        </form>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <!-- Full-width row -->
    <div class="row" style="padding-top: 10px;">
        <!-- Full-width column -->
        <div class="col-12">
            <!-- Journal entry card -->
            <div class="card" style="height: 400px;">
                <div class="card-body " style="padding: 10px;">
                    <div class="card">
                        <div class="card-header ">
                            <!-- Journal entry title -->
                            <h3 class="card-title">How Was Your Day?</h3>
```

```
                        </div>
                        <!-- Journal entry text area -->
                        <div class="card-body" style="position: relative;">
                            <form style="height: 100%;">
                                <textarea id="journalText" name="journalText" placeholder=" "
rows="10" myTextBox style="padding: 15px; width: 100%; height: 100%; border: none;"
required></textarea>
                            </form>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>

<!-- Save button -->

<div class="container">
    <div class="row">
        <div class="col-12 mt-auto d-flex justify-content-end" style="padding: 40px;">
            <div class="d-grid">
                <button  class="btn btn-primary btn-lg" type="button" style="background: #396389;"
onclick="saveJournal()" >Save</button>
            </div>
        </div>
    </div>
</div>

<!-- popup and overlay -->
<div class="overlay" id="overlay"></div>
<div class="popup text-center " id="popup">
  <h3 id="modalContent" style="padding: 20px; color: #396389;">Your Journal have been saved</h3>
  <button class="redirect-button  btn-primary btn-lg" style="background: #396389; width: 60%;"
id="redirectButton">OK</button>
</div>


<!-- SCRIPTS -->
<script src="{% static 'js/jquery.min.js' %}"></script> <!-- jQuery library -->
<script src="{% static 'js/bootstrap.min.js' %}"></script> <!-- Bootstrap library -->
<script src="{% static 'js/aos.js' %}"></script> <!-- AOS library for scroll animations -->
<script src="{% static 'js/owl.carousel.min.js' %}"></script> <!-- Owl Carousel library for sliders --
>
<script src="{% static 'js/smoothscroll.js' %}"></script> <!-- Smooth scroll functionality -->
```

```
<script src="{% static 'js/custom.js' %}"></script> <!-- Custom JavaScript -->

<script>
    // Function to display selected image
    function displaySelectedImage(event, elementId) {
        const selectedImage = document.getElementById(elementId);
        const fileInput = event.target;

        if (fileInput.files && fileInput.files[0]) {
            const reader = new FileReader();

            reader.onload = function(e) {
                selectedImage.src = e.target.result;
            };

            reader.readAsDataURL(fileInput.files[0]);
        }
    }

    // Function to delete selected image
    function deleteSelectedImage() {
        // Set the source of the selected image back to the placeholder image
        document.getElementById('selectedImage').src =
"https://mdbootstrap.com/img/Photos/Others/placeholder.jpg";
        // Clear the file input
        document.getElementById('customFile1').value = "";
    }

    // Function to display current date
    var now = new Date();
    var date = now.toLocaleDateString(); // Extract only the date part
    document.getElementById("datetime").innerHTML = date; // Insert date into HTML

    // Function to show pop-up
    function showPopUp() {
        document.getElementById('overlay').style.display = 'block';
        document.getElementById('popup').style.display = 'block';
        // Redirect button action
        document.getElementById('redirectButton').onclick = function() {
            window.location.href = "{% url 'home' %}"; // Redirect to the specified URL
        };
    }

    // Function to save journal entry
    function saveJournal() {
```

```
    var journalText = document.getElementById('journalText').value.trim();
    if (journalText === '') {
        alert('Please enter your journal text.');
        return;
    }
    var moodForm = document.getElementById('moodForm');
    var mood = getSelectedRadioValue(moodForm);
    if (!mood) {
        alert('Please select your mood.');
        return;
    }
    var imageInput = document.getElementById('customFile1');
    var imageFile = imageInput.files[0];
    var formData = new FormData();
    formData.append('journalText', journalText);
    formData.append('mood', mood);
    formData.append('image', imageFile);
    fetch('/save_journal/', {
        method: 'POST',
        headers: {
            'X-CSRFToken': '{{ csrf_token }}'
        },
        body: formData
    })
    .then(response => response.json())
    .then(data => {
        console.log('Journal saved:', data);
        showPopUp();
        setTimeout(function() {
            window.location.href = "{% url 'home' %}";
        }, 2000);
    })
    .catch(error => {
        console.error('Error saving journal:', error);
    });
}

// Function to get value of selected radio button
function getSelectedRadioValue(form) {
    var selectedRadio = form.querySelector('input[type="radio"]:checked');
    return selectedRadio ? selectedRadio.value : null;
}


document.getElementById('saveButton').addEventListener('click', function(event) {
```

```
    event.preventDefault(); // Prevent the default button click action

    // Get the values of the text and mood fields
    var journalText = document.getElementById('journalText').value.trim();
    var moodInputs = document.getElementsByName('mood');
    var selectedMood = false;
    for (var i = 0; i < moodInputs.length; i++) {
        if (moodInputs[i].checked) {
            selectedMood = true;
            break;
        }
    }

    // Check if both text and mood fields are provided
    if (!journalText || !selectedMood) {
        // Display an error message
        document.getElementById('errorMessage').textContent = 'Both mood and journal text fields are
required.';
        document.getElementById('errorMessage').style.display = 'block';
    } else {
        // Submit the form
        saveJournal();
    }
});



</script>




</body>
</html>
```

Prevjournal.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>

    <title>SoulGlow</title>


<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=Edge">
<meta name="description" content="">
<meta name="keywords" content="">
<meta name="author" content="">
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">
<link rel="stylesheet" href="{% static 'css/bootstrap.min.css' %}">
<link rel="stylesheet" href= "{% static 'css/font-awesome.min.css' %}">
<link rel="stylesheet" href="{% static 'css/aos.css' %}">
<link rel="stylesheet" href="{% static 'css/owl.carousel.min.css' %}">
<link rel="stylesheet" href="{% static 'css/owl.theme.default.min.css' %}">
<!-- MAIN CSS -->
<link rel="stylesheet" href="{% static 'css/templatemo-digital-trend.css' %}">

<style>
    .journal-entry {

        background-color: white;
        padding: 40px;
        margin-bottom: 20px;
        border-radius: 10px;
        box-shadow: 0px 4px 8px rgba(0, 0, 0, 0.1);
    }

    .journal-entry p {
        margin-bottom: 10px;
    }

    .read-more {
        color: #6491A4;
        cursor: pointer;
        text-decoration: underline;
    }
    .overlay {
            display: none;
            position: fixed;
```

```
            top: 0;
            left: 0;
            width: 100%;
            height: 100%;
            background-color: rgba(0, 0, 0, 0.5); /* Semi-transparent black */
            z-index: 998; /* Ensure overlay is below the popup */
        }
        .popup {
     display: none; /* Hide the popup initially */
     position: fixed;
     top: 50%;
     left: 50%;
     transform: translate(-50%, -50%);
     max-width: 80%;
     max-height: 80%;
     overflow: auto; /* Add overflow to handle content overflow */
     background-color: white;
     padding: 20px;
     border-radius: 10px;
     box-shadow: 0 0 20px rgba(0, 0, 0, 0.3);
     z-index: 9999;
    }
.popup-content {
    padding: 40px;
    border-radius: 10px;
    width: 100%; /* Make sure content doesn't exceed the popup width */
    height: 100%; /* Make sure content doesn't exceed the popup height */
    box-sizing: border-box;
}
.close-button {
    position: fixed;
    bottom: 10px; /* Adjust as needed */
    right: 10px; /* Adjust as needed */
    margin-bottom: -20px; /* Adjust as needed */
    margin-right: -20px; /* Adjust as needed */
    cursor: pointer;
    font-size: 20px;
    color: #333;
}

.entry-label {
    font-size:larger;
    font-weight: 900;
    color: #396389;
}
```

```
</style>

<script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>

</head>

<!-- MENU BAR -->
<nav class="navbar navbar-expand-lg"  style="background-color: #6491A4;">
    <div class="container">
        <a class="navbar-brand" href="index.html">
          <!-- <i class="fa fa-line-chart"></i>       MENU BAR -->
          <img src="{% static 'images/Logo.png' %}" class="img-fluid" alt="Logo" width="80"
height="40">
          SoulGlow
        </a>

        <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav"
aria-controls="navbarNav" aria-expanded="false"
            aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>

        <div class="collapse navbar-collapse" id="navbarNav">
            <ul class="navbar-nav ml-auto">

             <li class="nav-item">
                 <a href="{% url 'home' %}"; class="nav-link smoothScroll">Explore</a>
             </li>
             <li class="nav-item">
                 <a href="{% url 'journal' %}";  class="nav-link smoothScroll">Journal</a>
             </li>
             <li class="nav-item">
                 <a href="{% url 'prevjournal' %}";  class="nav-link smoothScroll">History</a>
             </li>
             <li class="nav-item">
                 <a href="{% url 'profile' %}"; class="nav-link smoothScroll">Profile</a>
             </li>
             <li class="nav-item">
                 <a href="{% url 'settings' %}"; class="nav-link">Settings</a>
             </li>
             <li class="nav-item">
                  <a href="/signout" class="nav-link contact">Sign out</a>
```

```
            </li>

        </ul>
    </div>
</div>
</nav>


<body style="background: linear-gradient( rgb(224, 230, 237) 0%, rgba(228,242,248,1) 100%); color:
#6491A4;" >

<br>
{% block content %}
<div class="journal-container" style="padding-left:200px ; padding-right:200px ;">
    {% for entry in journal_entries %}
        <div class="journal-entry">
            <p><span class="entry-label">Date:</span> {{ entry.created_at }}</p>

            <p><span class="entry-label">My Mood:</span> {{ entry.mood }}</p>

            <p><span class="entry-label">My Journal:</span> {{ entry.text|slice:":80" }}
                {% if entry.text|length > 10 %}
                    <span class="read-more" onclick="openPopup('{{ entry.created_at }}', '{{
entry.mood }}', '{{ entry.text|escapejs }}', '{% if entry.image %}{% if entry.image.url %}{{
entry.image.url }}{% endif %}{% endif %}')">Read more...</span>
                {% endif %}
            </p>

            {% if entry.image %}
                {% if entry.image.url %}
                    <img src="{{ entry.image.url }}" alt="Journal Image" width="300" height="200" >
                {% else %}
                    <p>No picture available</p>
                {% endif %}
            {% else %}
                <p>No picture available</p>
            {% endif %}
            <br><br>
        <!-- Add delete button -->
        <form id="delete-form-{{ entry.id }}" action="{% url 'delete_journal_entry' entry.id %}"
method="post">
            {% csrf_token %}
            <button type="button" class="btn btn-danger btn-sm" onclick="confirmDelete({{ entry.id
}})">Delete</button>
        </form>
```

```
        <script>
          function confirmDelete(entryId) {
            if (confirm("Are you sure you want to delete this journal entry?")) {
              document.getElementById("delete-form-" + entryId).submit();
            }
          }
        </script>



        </div>
    {% empty %}

        <div class="section">
            <div class="card">
                <div class="card-body">
                    <p><strong style="color: #6491A4;">No journal entries available.</strong></p>
                </div>
            </div>
        </div><br><br><br><br><br><br><br><br><br><br><br><br><br><br>

    {% endfor %}
</div>
{% endblock %}



<div class="overlay" id="overlay"></div>

<div class="popup" id="popup">
    <div class="popup-content">
        <p id="popup-date"></p>
        <p id="popup-mood"></p>
        <p id="popup-text"></p>
        <img id="popup-image" alt="Journal Image" width="300" height="200">
        <button class="btn btn-primary btn-lg float-right" type="button" style="background: #396389;"
onclick="closePopup()">close</button>

    </div>
</div>


<script>
   function openPopup(date, mood, text, imageUrl) {
    console.log("Opening popup with:", date, mood, text, imageUrl);
```

```
    document.getElementById("popup-date").innerHTML = "<span class='entry-label'>Date:</span> " +
date;
    document.getElementById("popup-mood").innerHTML = "<span class='entry-label'>My Mood:</span> " +
mood;
    document.getElementById("popup-text").innerHTML = "<span class='entry-label'>My Journal:</span> "
+ text;

    if (imageUrl) {
        document.getElementById("popup-image").src = imageUrl;
        document.getElementById("popup-image").style.display = "block";
    } else {
        document.getElementById("popup-image").style.display = "none";
    }

    // Show the overlay and the popup
    document.getElementById("overlay").style.display = "block";
    document.getElementById("popup").style.display = "block";
}

function closePopup() {
    console.log("Closing popup");
    // Hide the overlay and the popup
    document.getElementById("overlay").style.display = "none";
    document.getElementById("popup").style.display = "none";
}

</script>
  <!-- SCRIPTS -->
  <script src= "{% static 'js/jquery.min.js' %}" ></script>
  <script src="{% static 'js/bootstrap.min.js' %}"></script>
  <script src="{% static 'js/aos.js' %}" ></script>
  <script src="{% static 'js/owl.carousel.min.js' %}" ></script>
  <script src="{% static 'js/smoothscroll.js' %}" ></script>
  <script src="{% static 'js/custom.js' %}"></script>

</body>
</html>
```

Profile.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>

    <title>SoulGlow</title>

<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=Edge">
<meta name="description" content="">
<meta name="keywords" content="">
<meta name="author" content="">
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">
<link rel="stylesheet" href="{% static 'css/bootstrap.min.css' %}">
<link rel="stylesheet" href= "{% static 'css/font-awesome.min.css' %}">
<link rel="stylesheet" href="{% static 'css/aos.css' %}">
<link rel="stylesheet" href="{% static 'css/owl.carousel.min.css' %}">
<link rel="stylesheet" href="{% static 'css/owl.theme.default.min.css' %}">
<!-- MAIN CSS -->
<link rel="stylesheet" href="{% static 'css/templatemo-digital-trend.css' %}">

<!-- Custom CSS -->
<style>
        /* Custom styling */
        .card-body .row .col-md-6 {
            padding-right: 15px;
            padding-left: 15px;
            height: 100%;
        }

        .card-body .row {
            margin-right: -15px;
            margin-left: -15px;
            height: 100%;
        }

        .overlay {
            display: none;
            position: fixed;
            top: 0;
            left: 0;
            width: 100%;
            height: 100%;
            background-color: rgba(0, 0, 0, 0.5); /* Semi-transparent black */
```

```
        z-index: 998; /* Ensure overlay is below the popup */
    }

    .popup {
        display: none; /* Hide the popup initially */
        position: fixed;
        top: 50%;
        left: 50%;
        transform: translate(-50%, -50%);
        max-width: 80%;
        max-height: 80%;
        overflow: auto; /* Add overflow to handle content overflow */
        background-color: white;
        padding: 20px;
        border-radius: 10px;
        box-shadow: 0 0 20px rgba(0, 0, 0, 0.3);
        z-index: 9999;
    }

    .section {
        margin-bottom: 20px;
    }

    .section:nth-child(2) {
        display: flex;
        justify-content: space-between;
    }

    .chart-container {
        display: flex;
    }

    .chart-container > div {
        margin-right: 20px;
        margin-left: 20px;
    }

    /* Adjust the styling of the charts */
    .chart-container img {
        display: block;
        margin: 0 auto;
        height: 350px;
        width: 600px;
    }
```

```
        .modal {
            display: none;
            position: fixed;
            top: 50%;
            left: 50%;
            transform: translate(-50%, -50%);
            padding: 20px;
            background-color: #fff;
            z-index: 1000;
            height: 220px;
            width: 700px;
        }

        /* Styles for the close button */
        .close {
            display: block;
            margin: 0 auto; /* Center the button */
            padding: 10px 20px;
            background-color: #6491A4;
            color: #fff;
            border: none;
            cursor: pointer;
            font-size: 16px;
            border-radius: 10px;
        }
</style>

</head>


<body style="background: linear-gradient( rgb(224, 230, 237) 0%, rgba(228,242,248,1) 100%); color:
#6491A4;" >

<!-- MENU BAR -->
<nav class="navbar navbar-expand-lg"  style="background-color: #6491A4;">
    <div class="container">
        <a class="navbar-brand" href="index.html">
          <img src="{% static 'images/Logo.png' %}" class="img-fluid" alt="Logo" width="80"
height="40">
          SoulGlow
        </a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav"
aria-controls="navbarNav" aria-expanded="false"
            aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
```

```
        </button>
        <div class="collapse navbar-collapse" id="navbarNav">
            <ul class="navbar-nav ml-auto">
             <li class="nav-item">
                 <a href="{% url 'home' %}"; class="nav-link smoothScroll">Explore</a>
             </li>
             <li class="nav-item">
                 <a href="{% url 'journal' %}";  class="nav-link smoothScroll">Journal</a>
             </li>
             <li class="nav-item">
                 <a href="{% url 'prevjournal' %}";  class="nav-link smoothScroll">History</a>
             </li>
             <li class="nav-item">
                 <a href="{% url 'profile' %}"; class="nav-link smoothScroll">Profile</a>
             </li>
             <li class="nav-item">
                 <a href="{% url 'settings' %}"; class="nav-link">Settings</a>
             </li>
             <li class="nav-item">
                  <a href="/signout" class="nav-link contact">Sign out</a>
              </li>

            </ul>
        </div>
    </div>
 </nav>
 <br>
 <!-- for user with NO entries this week -->
 {% if no_journals_message  %}
 <div class="section">
    <div class="card">
        <div class="card-body">
            <p><strong style="color: #6491A4;">{{no_journals_message }}</strong></p>
        </div>
    </div>
</div>
<br><br><br><br><br><br><br><br><br><br><br><br><br>
<!-- for user with entries this week -->
{% elif journalnum != 0 %}
<div class="container" style="padding: 2%;">
 <div class="row">
    <!-- First column Weekly Depression and emotion Analysis -->
    <div class="col-md-6">
        <!-- First Row for Weekly Depression Analysis-->
        <div class="row">
```

```
            <div class="col-md-12">
                <div class="section">
                    <div class="card">
                        <div class="card-body">
                            <h3>Weekly Depression Analysis:</h3><br>
                            {% if weekly_dep.is_depressed %}
                                <p><strong style="color: #6491A4;">Depression Prediction:</strong> It
seems like you might be feeling down, Your journals indicate depression.</p>

                                <!-- Modal pop-up -->
                                <!-- popup and overlay -->
                            <div class="overlay" id="overlay"></div>
                            <div class="popup text-center " id="popup">

                                <h3 id="modalContent" style="padding: 20px; color: #396389;">We've
noticed some concerning signs in your recent journals. It seems like you might be experiencing
depression. We want you to know that you're not alone, and support is available. We encourage you to
seek help and talk to someone you trust.</h3>
                                <button class="redirect-button  btn-primary btn-lg" style="background:
#396389; width: 60%;" id="close">OK</button>
                            </div>
                            {% else %}
                                <p><strong style="color: #6491A4;">Depression Prediction:</strong>
You're doing well! Keep it up, Your journals do not indicate depression.</p>
                            {% endif %}
                                <p><strong style="color: #6491A4;">Depression Probability:</strong> {{
weekly_dep.dep_probability|floatformat:"2"  }}%</p>
                        </div>
                    </div>
                </div>
            </div>
        </div>

        <!-- Second Row for Weekly Emotion Analysis -->
        <div class="row">
            <div class="col-md-12">
                <div class="section">
                    <div class="card">
                        <div class="card-body">
                            <h3 class="card-title">Weekly Emotion Analysis:</h3>
                            <p class="card-text"><strong style="color: #6491A4;">Number of Journals
for the Week: {{ journalnum }}</strong></p>
                            <p class="card-text"><strong style="color: #6491A4;">Your feelings for
this week:</strong></p>
                            {% for  entry in current_week_entries %}
```

```
                            {% with journal_date=entry.created_at %}
                                <p>
                                    <span style="color: #455d7a;">{{ entry.created_at|date:"Y-m-d"
}}</span> :
                                    {% if "sadness" in entry.predicted_emotion %}
                                        <span style="color: #c9356c;">{{ entry.predicted_emotion }}
</span>
                                    {% else %}
                                        {{ entry.predicted_emotion }}
                                    {% endif %}
                                    {% if entry.probability %}
                                        ({{ entry.probability|floatformat:"2" }}%)
                                    {% endif %}
                                </p>
                            {% endwith %}
                            {% endfor %}
                            <!--show text if user is mostly sad -->
                            {% if sadness_count > 1 %}
                            <p class="card-text"><strong style="color: #6491A4;">You have been feeling
<span style="color: #c9356c;">sad</span> multiple times this week, which could potentially be
indicative of depression.</strong></p>
                            {% endif %}
                        </div>
                    </div>
                </div>
            </div>
        </div>
        <hr>
    </div>

    <!-- Second column contain all 3 chars -->
    <div class="col-md-6">
        <!-- First Row for Weekly Depression Check -->
        <div class="row">
            <div class="col-md-12">
                <div class="section">
                    <div class="card">
                        <div class="card-body">
                            <h3 class="card-title" style="text-align: center;">Weekly Depression
Check</h3>
                            <div style="text-align: center;">
                                <img src="{{ cumulative_chart_image }}"
alt="Cumulative Depression Chart" class="img-fluid">
                            </div>
                        </div>
```

```
                    </div>
                </div>
            </div>
        </div>

        <!-- Second Row for Weekly Emotional Chart -->
        <div class="row">
            <div class="col-md-12">
                <div class="section">
                    <div class="card">
                        <div class="card-body">
                            <h3 class="card-title" style="text-align: center;">Weekly Emotional
Chart</h3>
                            <div style="text-align: center;">
                                <img src="{{ emotion_chart_image }}"
alt="Emotion Distribution Chart" class="img-fluid">
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>

        <!-- Third Row for Monthly Emotional Overview-->
        <div class="row">
            <div class="col-md-12">
                <div class="section">
                    <div class="card">
                        <div class="card-body">
                            <h3 class="card-title" style="text-align: center;">Monthly Emotional
Overview</h3>
                            <div style="text-align: center;">
                                <img src="{{ emotion_monthly_chart_image }}"
alt="Emotion Distribution Chart" class="img-fluid" >
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
</div>
{%else %}
<!-- show Monthly Emotional Overview only, for users with entries this month without the current
```

```
week  -->

<div class="row">
    <div class="col-md-12">
        <div class="section">
            <div >
                <div class="card-body">
                    <h3 class="card-title" style="text-align: center;">Monthly Emotional Overview</h3>
                    <div style="text-align: center;">
                        <img src="{{ emotion_monthly_chart_image }}"
alt="Emotion Distribution Chart" class="img-fluid" height="500" width="600">
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
{% endif %}

    <!-- SCRIPTS -->
    <script src= "{% static 'js/jquery.min.js' %}" ></script>
    <script src="{% static 'js/bootstrap.min.js' %}"></script>
    <script src="{% static 'js/aos.js' %}" ></script>
    <script src="{% static 'js/owl.carousel.min.js' %}" ></script>
    <script src="{% static 'js/smoothscroll.js' %}" ></script>
    <script src="{% static 'js/custom.js' %}"></script>

<script>
    window.onload = function() {
        showPopUp(); // Call the function to display the popup when the page loads

        // Attach event listener to the close button
        document.getElementById('close').addEventListener('click', function() {
            // Hide the overlay and popup
            document.getElementById('overlay').style.display = 'none';
            document.getElementById('popup').style.display = 'none';
        });
    };

    function showPopUp() {
        document.getElementById('overlay').style.display = 'block';
        document.getElementById('popup').style.display = 'block';
    }
</script>
</body>
```

```
</html>
```

Settings.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- Meta tags -->
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=Edge">
    <meta name="description" content="">
    <meta name="keywords" content="">
    <meta name="author" content="">
    <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">

    <!-- External CSS -->
    <link rel="stylesheet" href="{% static 'css/bootstrap.min.css' %}">
    <link rel="stylesheet" href="{% static 'css/font-awesome.min.css' %}">
    <link rel="stylesheet" href="{% static 'css/aos.css' %}">
    <link rel="stylesheet" href="{% static 'css/owl.carousel.min.css' %}">
    <link rel="stylesheet" href="{% static 'css/owl.theme.default.min.css' %}">

    <!-- Main CSS -->
    <link rel="stylesheet" href="{% static 'css/templatemo-digital-trend.css' %}">

    <title>Settings</title>

    <!-- Internal CSS -->
    <style type="text/css">
        /* Body styling */
        body {
            background-color: #E0E6EF;
            color: #6491A4;
        }

        /* Centering the container */
        .container {
            display: flex;
            justify-content: center;
            align-items: center;
        }

        /* Styling for the list */
        .list-group {
            list-style: none;
            border-radius: 50px;
            width: 600px;
```

```
}

/* Styling for list items */
.list-group-item {
    padding: 30px 30px;
    position: relative;
    font-size: 20px;
}

/* Styling for arrow button */
.arrow-button {
    position: absolute;
    right: 30px;
    top: 50%;
    transform: translateY(-50%);
    background-color: transparent;
    border: none;
    cursor: pointer;
}

/* Styling for arrow icon */
.arrow-icon {
    width: 40px;
    height: 40px;
    fill: #6491A4;
}

/* Styling for overlay */
.overlay {
    display: none;
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-color: rgba(0, 0, 0, 0.5); /* Semi-transparent black */
    z-index: 998; /* Ensure overlay is below the popup */
}

/* Styling for popup */
.popup {
    display: none; /* Hide the popup initially */
    position: fixed;
    top: 50%;
    left: 50%;
```

```
        transform: translate(-50%, -50%);
        max-width: 80%;
        max-height: 80%;
        overflow: auto; /* Add overflow to handle content overflow */
        background-color: white;
        padding: 20px;
        border-radius: 10px;
        box-shadow: 0 0 20px rgba(0, 0, 0, 0.3);
        z-index: 999;
    }

    /* Styling for FAQ popup */
    .faq-popup h3 {
        font-size: 16px;
        line-height: 1.5;
        text-align: center;
    }

    /* Styling for close button */
    .close-button {
        position: absolute;
        top: 5px;
        right: 5px;
        background: none;
        border: none;
        cursor: pointer;
    }

    /* Styling for toggle switch container */
    .toggle-switch-container {
        display: flex;
        justify-content: space-between;
        align-items: center;
        margin-bottom: 10px;
    }

    /* Styling for main text */
    .main-text {
        text-align: center;
        margin: 20px;
        color: #6491A4;
        font-size: larger;
    }

    /* Styling for popup text */
```

```
        .popup-text {
            margin-bottom: 20px;
        }

        /* Styling for main button */
        .main-button {
            display: block;
            margin: 20px auto;
        }
    </style>
</head>
<body>

    <!-- MENU BAR -->
    <nav class="navbar navbar-expand-lg"  style="background-color: #6491A4;">
        <div class="container">
            <a class="navbar-brand" href="index.html">
              <img src="{% static 'images/Logo.png' %}" class="img-fluid" alt="Logo" width="80"
height="40">
              SoulGlow
            </a>

            <!-- Toggle button for responsive navbar -->
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarNav" aria-controls="navbarNav" aria-expanded="false"
                aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>

            <!-- Navigation links -->
            <div class="collapse navbar-collapse" id="navbarNav">
                <ul class="navbar-nav ml-auto">
                    <li class="nav-item">
                        <a href="{% url 'home' %}"; class="nav-link smoothScroll">Explore</a>
                    </li>
                    <li class="nav-item">
                        <a href="{% url 'journal' %}";  class="nav-link smoothScroll">Journal</a>
                    </li>
                    <li class="nav-item">
                        <a href="{% url 'prevjournal' %}";  class="nav-link smoothScroll">History</a>
                    </li>
                    <li class="nav-item">
                        <a href="{% url 'profile' %}"; class="nav-link smoothScroll">Profile</a>
                    </li>
                    <li class="nav-item">
```

```
                    <a href="{% url 'settings' %}"; class="nav-link">Settings</a>
                </li>
                <li class="nav-item">
                    <a href="/signout" class="nav-link contact">Sign out</a>
                </li>
            </ul>
        </div>
    </div>
</nav>


<div class="main-text">
    <!-- Displaying user's username -->
    <h2>{{ user.username }}</h2>
    <br>
</div>

<!-- Container for list group -->
<div class="container">
    <ul class="list-group">
        <!-- List item for Account Information -->
        <li class="list-group-item">Account Information<button class="arrow-button"
id="item1Button"><svg class="arrow-icon" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24"><path
d="M8.59 16.59L13.17 12 8.59 7.41 10 6l6 6-6 6-1.41-1.41z"/></svg></button></li>
        <!-- List item for FAQ -->
        <li class="list-group-item">FAQ<button class="arrow-button" onclick="showPopupFAQ()"><svg
class="arrow-icon" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24"><path d="M8.59 16.59L13.17
12 8.59 7.41 10 6l6 6-6 6-1.41-1.41z"/></svg></button></li>
        <!-- List item for Delete Account -->
        <li class="list-group-item" onclick="showDeleteAccountPopup()">Delete Account<button
class="arrow-button" id="deleteAccountButton"><svg class="arrow-icon"
xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24"><path d="M8.59 16.59L13.17 12 8.59 7.41 10 6l6
6-6 6-1.41-1.41z"/></svg></button></li>
    </ul>
</div>

<!-- Overlay for popups -->
<div class="overlay" id="overlay"></div>
<!-- Popup container -->
<div id="popup" class="popup"></div>
<script>
 // Flag to keep track of whether a popup is open
 var isPopupOpen = false;

 // Function to show popup with message
```

```
 function showPopup(message) {
    document.getElementById("overlay").style.display = "block";
     var popup = document.getElementById("popup");
     popup.innerHTML = "<p>" + message + "</p>" + '<button class="close-button"
onclick="hidePopup()">X</button>';
     popup.style.display = "block";
     isPopupOpen = true;
     disableArrowButtons();
 }

 // Function to hide popup
 function hidePopup() {
    document.getElementById("overlay").style.display = "none";
     var popup = document.getElementById("popup");
     popup.style.display = "none";
     isPopupOpen = false;
     enableArrowButtons();
 }

 // Function to disable arrow buttons
 function disableArrowButtons() {
     var arrowButtons = document.querySelectorAll(".arrow-button");
     arrowButtons.forEach(function (button) {
         button.disabled = true;
     });
 }

 // Function to enable arrow buttons
 function enableArrowButtons() {
     var arrowButtons = document.querySelectorAll(".arrow-button");
     arrowButtons.forEach(function (button) {
         button.disabled = false;
     });
 }


// Flag to keep track of whether a popup is open
var isPopupOpen = false;

// Function to show popup with message
function showPopup(message) {
    document.getElementById("overlay").style.display = "block";
    var popup = document.getElementById("popup");
    popup.innerHTML = "<p>" + message + "</p>" + '<button class="close-button"
onclick="hidePopup()">X</button>';
```

```
    popup.style.display = "block";
    isPopupOpen = true;
    disableArrowButtons();
}

// Function to hide popup
function hidePopup() {
    document.getElementById("overlay").style.display = "none";
    var popup = document.getElementById("popup");
    popup.style.display = "none";
    isPopupOpen = false;
    enableArrowButtons();
}

// Function to disable arrow buttons
function disableArrowButtons() {
    var arrowButtons = document.querySelectorAll(".arrow-button");
    arrowButtons.forEach(function(button) {
        button.disabled = true;
    });
}

// Function to enable arrow buttons
function enableArrowButtons() {
    var arrowButtons = document.querySelectorAll(".arrow-button");
    arrowButtons.forEach(function(button) {
        button.disabled = false;
    });
}

// Function to show popup with form for item 1
function showPopupItem1() {
    document.getElementById("overlay").style.display = "block";
    var popup = document.getElementById("popup");
    // Dynamically generated HTML for the popup form
    popup.innerHTML = `
        <div>
            <h3>Account Information</h3>
            <form action="{% url "settings" %}" method="POST">
                {% csrf_token %}
                <div>
                    <label for="name">name:</label>
                    <input class="form-control" type="text" id="name" name="name" value="{{
user.first_name }}" required><br><br>
                </div>
```

```
                <div>
                    <label for="username">username:</label>
                    <input class="form-control" type="text" id="username" name="username" value="{{
user.username }}" required><br><br>
                </div>
                <div>
                    <label for="email">Email:</label>
                    <input class="form-control" type="email" id="email" name="email" value="{{
user.email }}" required><br><br>
                </div>
                <div>
                    <input type="submit" class="btn btn-primary btn-lg" style="background: #396389;
margin: 0 auto;" value="Save Changes">
                </div>
            </form>
            <button class="close-button" onclick="hidePopup()">X</button>
        </div>
    `;

    popup.style.display = "block";
    isPopupOpen = true;
    disableArrowButtons();
}

// Add event listener to item 1 button to show popup
document.getElementById("item1Button").addEventListener("click", showPopupItem1);

// Function to show FAQ popup
function showPopupFAQ() {
    document.getElementById("overlay").style.display = "block";
    var popup = document.getElementById("popup");
    // Dynamically generated HTML for the FAQ popup
    popup.innerHTML = '<div class="popup-header" >' +
        '<h3>FAQ</h3>' +
        '<p><strong>- What is SoulGlow?</strong><br>It’s an online journaling website, designed to
help users monitor and track their emotions and moods over time. It allows you to journal your
feelings and analyzes them, helping you gain insights into your emotional well-being.</p>' +
        '<p><strong>- Is my data safe and private?</strong><br>Yes, we prioritize your privacy and
data security.</p>' +
        '<p><strong>- Is SoulGlow services for free?</strong><br>Yes, our journaling and mood tracking
services are free.</p>' +
        '<p><strong>- How can I journal?</strong><br>To journal, all you have to do is log in and
click on the Journal button on the homepage and start journaling.</p>' +
        '<p><strong>- How can I view my mood history?</strong><br>You can click on your profile page
and it will show all of your mood history and previous journals.</p>' +
```

```
        '<button class="close-button" onclick="hidePopup()">X</button>' +
        '</div>';
    popup.style.display = "block";
    isPopupOpen = true;
    disableArrowButtons();
}

// Function to show popup for deleting account
function showDeleteAccountPopup() {
    document.getElementById("overlay").style.display = "block";
    var popup = document.getElementById("popup");
    // Dynamically generated HTML for the delete account popup
    popup.innerHTML =
        '<form id="deleteForm" action="{% url "delete_account" pk=user.pk %}" method="post">'+
            '{% csrf_token %}'+
            '<div class="popup-header" style="text-align: center;">' +
                '   <h3>Are You Sure You Want To Delete Your Account?</h3>' +
                '   <p class="popup-text">If you delete your account, you will lose all of your data
permanently. This action cannot be undone.</p>' +
                '</div>' +
            '<button type="submit" class="btn btn-danger main-button"
onclick="deleteAccount()">Delete</button>'+
         '</form>'+
         '<button class="close-button" onclick="hidePopup()">X</button>';
     popup.style.display = "block";
     isPopupOpen = true;
     disableArrowButtons();
}

// Function to check if another popup is open before showing a new one
function showPopupSafely(func) {
    if (!isPopupOpen) {
        func();
    }
}

// Event listener for delete account button
document.getElementById("deleteAccountButton").addEventListener("click", function () {
    showPopupSafely(showDeleteAccountPopup);
});

// Event listener for FAQ button
document.getElementById("faqButton").addEventListener("click", function () {
    showPopupSafely(showPopupFAQ);
});
```

```
</script>
<!-- SCRIPTS -->
<script src= "{% static 'js/jquery.min.js' %}" ></script>
<script src="{% static 'js/bootstrap.min.js' %}"></script>
<script src="{% static 'js/aos.js' %}" ></script>
<script src="{% static 'js/owl.carousel.min.js' %}" ></script>
<script src="{% static 'js/smoothscroll.js' %}" ></script>
<script src="{% static 'js/custom.js' %}"></script>

</body>
</html>
```

Signin.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://unpkg.com/bootstrap@5.3.2/dist/css/bootstrap.min.css">
    <!-- Custom CSS for login -->
    <link rel="stylesheet" href="https://unpkg.com/bs-brain@2.0.3/components/logins/login-
9/assets/css/login-9.css">

    <title>Sign in</title>
</head>
<body style="background: linear-gradient(to right, rgba(244,163,192,1) 0%, rgba(249,234,188,1)
100%);">

    <!-- Display messages -->
    {% for message in messages %}
    <div class="alert alert-{{ message.tags }} alert-dismissible fade show" role="alert">
        {{ message }}
        <!-- Button to close the message -->
        <button type="button" class="close" data-dismiss="alert" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
   {% endfor %}
<!-- Section for sign-in form -->
<section class=" py-3 py-md-5 py-xl-8">
  <div class="container">
      <div class="row gy-4 align-items-center">
          <!-- Left column with welcome message -->
          <div class="col-12 col-md-6 col-xl-7">
              <div class="d-flex justify-content-center">
                  <div class="col-12 col-xl-9">
                      <hr class="border-primary-subtle mb-4">
                      <h1 style="color: #396389;">Welcome Back to SoulGlow.</h1>
                      <h3 style="color: #396389;">Your safe space for emotional well-being</h3>
                      <div class="text-endx">
                      </div>
                  </div>
              </div>
          </div>
          <!-- Right column with sign-in form -->
          <div class="col-12 col-md-6 col-xl-5">
```

```
            <div class="card border-0 rounded-4">
                <div class="card-body p-3 p-md-4 p-xl-5">
                    <div class="row">
                        <div class="col-12">
                            <div class="mb-4">
                                <h3 style="color: #396389;">Sign in</h3>
                                <p>Don't have an account? <a href="/signup">Sign up</a></p>
                            </div>
                        </div>
                    </div>
                    <!-- Sign-in form -->
                    <form action="/signin" method="POST">
                        {% csrf_token %}
                        <div class="row gy-3 overflow-hidden">
                            <!-- Username input -->
                            <div class="col-12">
                                <div class="form-floating mb-3">
                                    <input type="text" class="form-control" name="username"
id="username" placeholder="Username" required>
                                    <label for="username" class="form-label">Username</label>
                                </div>
                            </div>
                            <!-- Password input -->
                            <div class="col-12">
                                <div class="form-floating mb-3">
                                    <input type="password" class="form-control" name="password"
id="password" value="" placeholder="Password" required>
                                    <label for="password" class="form-label">Password</label>
                                </div>
                            </div>
                            <!-- Admin checkbox -->
                            <div class="col-12">
                                <div class="form-check">
                                    <input class="form-check-input" type="checkbox" value=""
name="admin" id="admin">
                                    <label class="form-check-label text-secondary" for="admin">
                                        Admin
                                    </label>
                                </div>
                            </div>
                            <!-- Sign-in button -->
                            <div class="col-12">
                                <div class="d-grid">
                                    <button class="btn btn-primary btn-lg" type="submit"
style="background: #396389;">Sign in</button>
```

```
                                </div>
                            </div>
                        </div>
                    </form>
                    <!-- Forgot password link -->
                    <div class="row">
                        <div class="col-12">
                            <div class="d-flex gap-2 gap-md-4 flex-column flex-md-row justify-
content-md-end mt-4">
                                <a href="{% url 'password-reset' %}">Forgot password</a>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
  </div>
</section>
</body>
</html>
```

Signup.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- Meta tags -->
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://unpkg.com/bootstrap@5.3.2/dist/css/bootstrap.min.css">
    <!-- Custom CSS -->
    <link rel="stylesheet" href="https://unpkg.com/bs-brain@2.0.3/components/logins/login-
9/assets/css/login-9.css">
    <title>Sign Up</title>
</head>
<body style="background: linear-gradient(to right, rgba(244,163,192,1) 0%, rgba(249,234,188,1)
100%);">

    <!-- Display messages -->
    {% for message in messages %}
    <div class="alert alert-{{ message.tags }} alert-dismissible fade show" role="alert">
        {{ message }}
        <!-- Close button -->
        <button type="button" class="close" data-dismiss="alert" aria-label="Close">
            <span aria-hidden="true">&times;</span>
        </button>
    </div>
   {% endfor %}
       <!-- Sign up section -->
       <section class=" py-3 py-md-5 py-xl-8">
        <div class="container">
          <div class="row gy-4 align-items-center">
            <!-- Introduction -->
            <div class="col-12 col-md-6 col-xl-7">
              <div class="d-flex justify-content-center">
                <div class="col-12 col-xl-9">
                  <!-- Title and description -->
                  <hr class="border-primary-subtle mb-4">
                  <h1 style="color: #396389;">Welcome to SoulGlow.</h1>
                  <h3 style="color: #396389;"> Join us today and begin your journey toward better
mental health. Our unique journaling system allows you to express your thoughts and feelings
openly.</h3>
                  <div class="text-endx">

                  </div>
                </div>
```

```
            </div>
          </div>
          <!-- Sign up form -->
          <div class="col-12 col-md-6 col-xl-5">
            <div class="card border-0 rounded-4">
              <div class="card-body p-3 p-md-4 p-xl-5">
                <div class="row">
                  <div class="col-12">
                    <div class="mb-4">
                      <!-- Sign up title and link to sign in -->
                      <h3 style="color: #396389;">Sign up</h3>
                      <p>Have an account? <a href="/signin">Sign in</a></p>
                    </div>
                  </div>
                </div>
                <!-- Sign up form -->
                <form action="/signup" method="POST">
                  {% csrf_token %}
                  <div class="row gy-3 overflow-hidden">
                      <!-- Name field -->
                      <div class="col-12">
                          <div class="form-floating mb-3">
                            <input type="text" class="form-control" name="name" id="name"
placeholder="Name" required>
                            <label for="name" class="form-label">Name</label>
                          </div>
                        </div>
                        <!-- Age field -->
                        <div class="col-12">
                          <div class="form-floating mb-3">
                            <input type="text" class="form-control" name="age" id="age" value=""
placeholder="Age" required>
                            <label for="age" class="form-label">Age</label>
                          </div>
                        </div>
                    <!-- Email field -->
                    <div class="col-12">
                      <div class="form-floating mb-3">
                        <input type="email" class="form-control" name="email" id="email"
placeholder="name@example.com" required>
                        <label for="email" class="form-label">Email</label>
                      </div>
                    </div>
                    <!-- Username field -->
                    <div class="col-12">
```

```
                        <div class="form-floating mb-3">
                          <input type="text" class="form-control" name="username" id="uasename"
value="" placeholder="Username" required>
                          <label for="username" class="form-label">Username</label>
                        </div>
                      </div>
                      <!-- Password field -->
                      <div class="col-12">
                        <div class="form-floating mb-3">
                          <input type="password" class="form-control" name="password" id="password"
value="" placeholder="Password" required>
                          <label for="password" class="form-label">Password</label>
                        </div>
                      </div>
                      <!-- Confirm password field -->
                      <div class="col-12">
                        <div class="form-floating mb-3">
                          <input type="password" class="form-control" name="confirm" id="confirm"
value="" placeholder="Confirm Password" required>
                          <label for="confirm" class="form-label">Confirm Password</label>
                        </div>
                      </div>
                      <!-- Sign up button -->
                      <div class="col-12">
                        <div class="d-grid">
                          <button class="btn btn-primary btn-lg" type="submit"style="background:
#396389;">Sign up</button>
                        </div>
                      </div>
                    </div>
                  </form>
                </div>
              </div>
            </div>
          </div>
        </div>
      </section>
</body>
</html>
```

Views.py

```
# Django modules
from django.shortcuts import get_object_or_404, render, redirect
from django.http import HttpResponse, JsonResponse
from django.contrib.auth.models import User
from django.contrib import messages
from django.contrib.auth import authenticate, login, logout
from django.core.mail import send_mail
from django.contrib.sites.shortcuts import get_current_site
from django.template.loader import render_to_string
from django.utils.http import urlsafe_base64_encode, urlsafe_base64_decode
from django.utils.encoding import force_bytes, force_str
from django.views.decorators.csrf import csrf_exempt
from django.contrib.auth.decorators import login_required
from django.conf import settings as conf_settings
from .tokens import generate_token
from django.conf import settings
from my_app import settings
from email.message import EmailMessage
from django.core.mail import EmailMessage

from SoulGlow.utils import preprocess_text

# Journal related
from .models import JournalEntry

# Plotting related
import plotly.graph_objs as go
import plotly.io as pio
import base64
from io import BytesIO
from datetime import datetime, timedelta, timezone

# Cryptography related
from cryptography.fernet import Fernet
cipher_suite = Fernet(settings.SECRET_KEY)
# Other
import calendar
import requests
# TensorFlow and NumPy
import tensorflow as tf
import numpy as np
from scipy.special import softmax
from datetime import datetime
# PIL and File handling
```

```
from PIL import Image
from django.core.files.uploadedfile import InMemoryUploadedFile
import base64
import json
# Depression model related
import os
import joblib
import numpy as np

#function returns the current time as a datetime object.
def get_current_time():
    return datetime.now(timezone.utc)

#_____load the Models_____


svm_model = joblib.load(open("SoulGlow\\depression_svm_model.pkl", "rb"))
tfidf_vectorizer = joblib.load(open("SoulGlow\\depression_tfidf_vectorizer.pkl", "rb"))


from transformers import TFAutoModelForSequenceClassification, AutoTokenizer
import tensorflow as tf

# Load the Emotion Model
model = TFAutoModelForSequenceClassification.from_pretrained("SoulGlow\BertEmotion")
# Load the tokenizer
tokenizer = AutoTokenizer.from_pretrained("SoulGlow\BertEmotion")


#_____users (sign up- sign in- sign out-activate- contact
us)_____

def home(request):
    return render(request, "SoulGlow/index.html")


def contact_submit(request):
    if request.method == 'POST':
        # Retrieve form data
        name = request.POST.get('name')
        email = request.POST.get('email')
        message = request.POST.get('message')

        # Do something with the form data (e.g., send an email)
        # For example, you can use Django's built-in EmailMessage class:
        from django.core.mail import EmailMessage
```

```
        email = EmailMessage(
           subject='New Contact Form Submission',
           body=f'Name: {name}\nEmail: {email}\nMessage: {message}',
            to=['Soulglow.website@gmail.com']
        )
        email.send()

        # Return a success response
        return render(request, "SoulGlow/index.html", {'thanks': 'Thank you for your
message!','scroll_to_contact': True})


    # If the request method is not POST, return a 405 Method Not Allowed
    return HttpResponse(status=405)


def signup(request):
    # Check if the request method is POST
    if request.method == 'POST':
        # Extract form data
        name = request.POST['name']
        username = request.POST['username']
        age = request.POST['age']
        email = request.POST['email']
        password = request.POST['password']
        confirm = request.POST['confirm']

        # Check if username already exists
        if User.objects.filter(username=username):
            messages.error(request,"username already exist")
            return redirect("home")

        # Check if email already exists
        if User.objects.filter(email=email):
            messages.error(request,"email already exist")
            return redirect("home")

        # Check if username is too long
        if len(username) > 10:
            messages.error(request,"username is too long")

        # Check if passwords match
        if password !=  confirm:
            messages.error(request, "passwords don't match")
```

```
        # Check if username is alphanumeric
        if not username.isalnum():
            messages.error(request,"username is too long")
            return redirect("home")

        # Convert age to an integer before comparing
        age = int('0'+age)
        # Check if age is at least 12 years old
        if age < 12:
            messages.error(request, "You must be at least 12 years old to sign up")
            return redirect("home")

        # Create the user
        myuser = User.objects.create_user(username, email, password)
        myuser.first_name = name
        myuser.Age = age
        myuser.is_active= False
        myuser.save()

        # Success message
        messages.success(request, "Please check your email inbox and follow the instructions to
confirm your email address.")

        # Send welcome email
        subject = "Welcome to SoulGlow"
        message = "hello " + myuser.first_name +"\n hope you have a lovely journey with us"
        from_email = conf_settings.EMAIL_HOST_USER
        to_list = [myuser.email]
        send_mail(subject, message, from_email, to_list )

        # Send confirmation email
        current_site = get_current_site(request)
        email_subject = "Confirm your email @ SoulGlow "
        message2 = render_to_string('email_confirmation.html', {
            'name': myuser.first_name,
            'domain': current_site.domain,
            'uid': urlsafe_base64_encode(force_bytes(myuser.pk)),
            'token' : generate_token.make_token(myuser)
        })
        email = EmailMessage(
            email_subject,
            message2,
            conf_settings.EMAIL_HOST_USER,
            [myuser.email],
        )
```

```
        email.fail_silently = True
        email.send()

        return redirect('signin')

    # Render the signup page if request method is not POST
    return render(request, "SoulGlow/signup.html")


def signin(request):
    # Check if the request method is POST
    if request.method == 'POST':
        # Extract username and password from the POST request
        username = request.POST['username']
        password = request.POST['password']
        # Authenticate user credentials
        user = authenticate(username=username, password=password)

        if user is not None:
            if user.is_staff:
                # If user is staff/admin, login and redirect to admin index page
                login(request, user)
                messages.success(request, 'You are logged in as admin.')
                return redirect('admin:index')
            else:
                # If user is a regular user, login and redirect to home page
                login(request, user)
                messages.success(request, 'You are logged in.')
                return redirect('home')
        else:
            # If authentication fails, display error message
            messages.error(request, 'Invalid credentials.')

    # Render the signin page if request method is not POST
    return render(request, 'SoulGlow/signin.html')

def signout(request):
    # Logout the user
    logout(request)
    messages.success(request, "logged out "  )
    # Redirect to the home page after logout
    return redirect("home")

def activate(request, uidb64, token ):
    try:
```

```
        # Decode the user ID from base64
        uid = force_str(urlsafe_base64_decode(uidb64))
        # Retrieve user with the decoded ID
        myuser = User.objects.get(pk=uid)

    except (TypeError, ValueError, OverflowError, User.DoesNotExist):
        myuser = None

    # Check if user exists and the token is valid
    if myuser is not None and generate_token.check_token(myuser, token):
        # Activate the user and login
        myuser.is_active = True
        myuser.save()
        login(request, myuser)
        return redirect('home')
    else:
        # Render activation failed page if activation fails
        return render(request, "activation_failed.html" )

#_____________________________for journal page_____________________________________________

def journal(request):
     return render(request, "SoulGlow/journal.html")

@csrf_exempt
# This decorator exempts the view from CSRF protection, allowing POST requests without a CSRF token.

def save_journal(request):
    if request.method == 'POST':
        # Retrieve journal text and mood from the POST data
        journal_text = request.POST.get('journalText', '')
        mood = request.POST.get('mood', '')

        # Check if both text and mood are provided
        if not journal_text or not mood:
            return JsonResponse({'status': 'error', 'message': 'Both text and mood fields are
required.'})

        try:
            # Retrieve optional image file
            image_file = request.FILES.get('image', None)

            # Encrypt the journal text
            encrypted_journal_text = encrypt_data(journal_text)
```

```
            # Analyze emotions and probabilities
            emotion_analysis = analyze_emotions(journal_text)
            predicted_emotion = emotion_analysis['prediction']
            probability = emotion_analysis['probability']
            depression= analyze_depression(journal_text)

            # Save the data to the database
            entry = JournalEntry(
                user=request.user,
                text=encrypted_journal_text,
                mood=mood,
                image=image_file,
                predicted_emotion=predicted_emotion,
                probability=probability,
                depression=depression,
            )
            entry.save()

            return JsonResponse({'status': 'success'})
        except Exception as e:
            # Return error response if any exception occurs
            return JsonResponse({'status': 'error', 'message': str(e)})
    else:
        # Return error response for invalid request method
        return JsonResponse({'status': 'error', 'message': 'Invalid request method'})

#_____________________________for history page_____________________________________________

def prevjournal(request):
    # Assuming you have a ForeignKey relationship with User in your JournalEntry model
    user = request.user
    journal_entries = JournalEntry.objects.filter(user=user).order_by('-created_at')

    # Decrypt journal text for display
    for entry in journal_entries:
        # Debug statement to inspect the encrypted data
        #print("Encrypted data:", entry.text)

        # Decrypt the data and print the decrypted text
        decrypted_text = decrypt_data(entry.text)
        #print("Decrypted text:", decrypted_text)

        # Update the entry with the decrypted text
        entry.text = decrypted_text
```

```
    context = {'journal_entries': journal_entries}
    return render(request, "SoulGlow/prevjournal.html", context)


def delete_journal_entry(request, entry_id):
    # Retrieve the journal entry to delete
    try:
        journal_entry = JournalEntry.objects.get(id=entry_id)
    except JournalEntry.DoesNotExist:
        # Handle case where entry does not exist (optional)
        pass
    else:
        # Delete the journal entry
        journal_entry.delete()

    # Redirect back to the previous journal page
    return redirect('prevjournal')

#_____________________________for settings page_____________________________________________

def delete_account(request,pk):

        # Delete account logic here
        # For example:
        # request.user.delete()
        # Redirect to a confirmation page or homepage
        queryset = User.objects.get(id = pk)
        if request.method == 'POST':
          queryset.delete()
        return redirect('home')  # Redirect to home page after deletion


@login_required
def settings(request):
    if request.method == 'POST':
        user = request.user
        user.first_name = request.POST.get('name')
        user.email = request.POST.get('email')
        user.username = request.POST.get('username')
        user.save()
        messages.success(request, "Your settings have been updated successfully.")
        return redirect('home')  # Redirect to the settings page again
    return render(request, "SoulGlow/settings.html", {'user':request.user})
```

```python
#_____________________________analyze emotions_____________________________________________


def analyze_emotions(journal_text):
    # Tokenize the input text
    inputs = tokenizer(journal_text, return_tensors="tf")

    # Make prediction using the model
    outputs = model(**inputs)

    # Extract relevant information from the outputs
    logits = outputs.logits.numpy()
    probabilities = softmax(logits)[0]
    predicted_class_id = int(tf.argmax(logits, axis=-1)[0])
    predicted_emotion_label = model.config.id2label[predicted_class_id]

    # Return the results with the maximum probability
    max_probability = np.max(probabilities) * 100  # Convert to percentage
    return {'prediction': predicted_emotion_label, 'probability': max_probability}


# function calculates the softmax values for the input scores to convert them into probabilities
def softmax(x):
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum()



#_____________________________analyze depression_____________________________________________

def analyze_depression(text):
    # Preprocess text using the same function as in your Jupyter notebook
    preprocessed_input = preprocess_text(text)

    # Vectorize the preprocessed text
    input_vectorized = tfidf_vectorizer.transform([preprocessed_input]).toarray()

    # Use the loaded model for prediction
    prediction_prob = svm_model.predict_proba(input_vectorized)

    # Extract the probability of depression class
    depression_probability = prediction_prob[0][1] * 100  # Convert to percentage

    return depression_probability
```

```python
def analyze_weekly_depression(user_entries):
    decrypted_entries = []

    # Decrypt the text of each journal entry
    for entry in user_entries:
        decrypted_entry_text = decrypt_data(entry.text)
        decrypted_entries.append(decrypted_entry_text)

    # Combine the decrypted text of all selected journals
    combined_text = '\n'.join(decrypted_entries)

    # Preprocess the combined text
    preprocessed_input = preprocess_text(combined_text)

    # Vectorize the preprocessed text
    input_vectorized = tfidf_vectorizer.transform([preprocessed_input]).toarray()

    # Use the loaded model for prediction
    prediction = svm_model.predict(input_vectorized)
    prediction_prob = svm_model.predict_proba(input_vectorized)

    # Extract the probability of depression class
    depression_probability = prediction_prob[0][1] * 100  # Convert to percentage

    # Determine if the text indicates depression
    is_depressed = prediction[0] == 1

    result = {
        'dep_probability': depression_probability,
        'is_depressed': is_depressed,
          }
    print(result)

    return result




#_____________________________profile_____________________________________________

def profile(request):
    # Fetch the user's journal entries
    user_entries = JournalEntry.objects.filter(user=request.user)
```

```
    # Analyze weekly entries
    current_date = get_current_time()
    start_of_week = current_date - timedelta(days=current_date.weekday())
    end_of_week = start_of_week + timedelta(days=6)
    current_week_entries = user_entries.filter(created_at__date__range=[start_of_week, end_of_week])

    # Calculate the start date of the current month
    start_of_month = current_date.replace(day=1)

    # Calculate the end date of the current month
    next_month = current_date.replace(day=28) + timedelta(days=4)
    end_of_month = next_month - timedelta(days=next_month.day)

    # Filter entries for the current month
    current_month_entries = user_entries.filter(created_at__date__range=[start_of_month,
end_of_month])


    if current_month_entries.count()==0:
        # No journals for the current week
        context = {
            'no_journals_message': "You haven't entered any journals this Month."
        }
        return render(request, 'SoulGlow/profile.html', context)



    if current_week_entries.exists():
        weekly_dep = analyze_weekly_depression(current_week_entries)
        print("done with weekly_dep ")

        # Count the number of entries in current_week_entries
        journalnum = current_week_entries.count()
        print("done with journalnum")


        # depression_data = analyze_depression(current_week_entries)
        cumulative_chart_image = plot_depression_chart(current_week_entries)
        print("done with ccumulative_chart_image")


        # emotion_chart_data = analyze_emotions_for_week(current_week_entries)
        emotion_chart_image = plot_emotions_weekly_analysis(current_week_entries)
        print("done with emotion_chart_image")
```

```
    # Compute sadness count
        sadness_count = sum(1 for entry in current_week_entries if entry.predicted_emotion ==
'sadness')
        print("done with sadness_count")

    # Monthly Emotion Analysis
    emotion_monthly_chart_image = plot_emotions_monthly_analysis(current_month_entries)
    print("done with emotion_monthly_chart_image")

    context = {
        'current_week_entries': current_week_entries,
        'weekly_dep': weekly_dep,
        'cumulative_chart_image': cumulative_chart_image,
        'emotion_chart_image': emotion_chart_image,
        'sadness_count': sadness_count,
        'emotion_monthly_chart_image': emotion_monthly_chart_image,
        'journalnum': journalnum,
    }

    # Check if depression is predicted and send a warning message
    if weekly_dep['is_depressed']:
        subject = "Concern About Your Wellbeing"
        message = (
            f"Hello {request.user.first_name},\n\n"
            "We've noticed some concerning signs in your recent journals. "
            "It seems like you might be experiencing depression. "
            "We want you to know that you're not alone, and support is available. "
            "We encourage you to seek help and talk to someone you trust.\n\n"
            "Best regards,\n"
            "The SoulGlow Team"
        )
        send_mail(subject, message, conf_settings.EMAIL_HOST_USER, [request.user.email])

    return render(request, 'SoulGlow/profile.html', context)

#_____________________________Affirmation_____________________________________________

def get_affirmation(request):
    try:
        # Send a GET request to the affirmations API
        response = requests.get('https://www.affirmations.dev/')

        # Parse the JSON response
        data = response.json()
```

```
        # Return the affirmation as JSON response
        return JsonResponse(data)

    except Exception as e:
        # If an exception occurs during the request, return an error message
        # along with the appropriate HTTP status code (500 - Internal Server Error)
        return JsonResponse({'error': str(e)}, status=500)

#_____________________________to encrypt and decrypt_____________________________________________

# Create an instance of the Fernet cipher
cipher_suite = Fernet(conf_settings.SECRET_KEY)

# Function to encrypt data
def encrypt_data(data):
    encrypted_data = cipher_suite.encrypt(data.encode())
    return encrypted_data

# Function to decrypt data
def decrypt_data(encrypted_data):
    decrypted_data = cipher_suite.decrypt(encrypted_data).decode()
    return decrypted_data

#_____________________________to visualize_____________________________________________

#generates a scatter plot to visualize the depression analysis over the week
def plot_depression_chart(current_week_entries):
    # Extract dates and corresponding depression probabilities from journal entries
    dates = [entry.created_at.date() for entry in current_week_entries if entry.depression is not
None]
    probabilities = [entry.depression for entry in current_week_entries if entry.depression is not
None]

    # Print the extracted dates for debugging purposes
    print("\nDepression analysis dates: \n", dates)

    # Create a scatter plot of depression probabilities over dates
    fig = go.Figure()
    fig.add_trace(go.Scatter(x=dates, y=probabilities, mode='lines+markers'))

    # Update layout with chart title and axis labels
    fig.update_layout(
        title='Depression Analysis Over The Week',
        xaxis_title='Date',
        yaxis_title='Probability of Depression',
```

```
        xaxis_tickangle=-45,
        xaxis_tickformat='%Y-%m-%d',
        title_x=0.5  # Center alignment for the title
    )

    # Set the x-axis tickvals and ticktext to display only the dates in the 'dates' variable
    fig.update_xaxes(tickvals=dates, ticktext=[date.strftime('%Y-%m-%d') for date in dates])

    # Convert the plot to a base64-encoded image
    image_stream = BytesIO()
    pio.write_image(fig, image_stream, format='png')
    image_stream.seek(0)
    image_base64 = base64.b64encode(image_stream.getvalue()).decode('utf-8')

    return image_base64




#generates a bar chart to visualize the distribution of emotions detected in the entries over a weekly
period
def plot_emotions_weekly_analysis(journal_entries):
    # Define emotions and colors
    emotions = ["anger", "joy", "surprise", "sadness", "love", "fear"]
    colors = ['#A1CCD1', '#C8E4B2', '#FDFFAE', '#EF9595', '#FFC0D9', '#E5E0FF']

    # Count occurrences of each emotion for each date
    date_counts = {}
    for entry in journal_entries:
        predicted_emotion_label = entry.predicted_emotion
        date = entry.created_at.date()

        if date in date_counts:
            date_counts[date][predicted_emotion_label] =
date_counts[date].get(predicted_emotion_label, 0) + 1
        else:
            date_counts[date] = {emotion: 0 for emotion in emotions}
            date_counts[date][predicted_emotion_label] = 1

    # Sort dates
    dates = sorted(date_counts.keys())

    # Count occurrences of each emotion for each date
    counts = {emotion: [date_counts[date].get(emotion, 0) for date in dates] for emotion in emotions}
```

```
    # Create a bar chart of emotion distribution over dates
    fig = go.Figure()
    bar_width = 0.1
    index = np.arange(len(dates))

    for i, emotion in enumerate(emotions):
        fig.add_trace(go.Bar(x=index + (i - 2) * bar_width, y=counts[emotion], name=emotion,
marker_color=colors[i]))

    # Update layout with chart title and axis labels
    fig.update_layout(
        xaxis=dict(tickmode='array', tickvals=index, ticktext=dates),
        xaxis_tickangle=-45,
        title='Emotion Distribution',
        xaxis_title='Date',
        yaxis=dict(tickvals=list(range(max(max(counts.values())) + 1))),
        legend_title='Emotion',
        title_x=0.5
    )

    # Convert the plot to a base64-encoded image
    image_stream = BytesIO()
    pio.write_image(fig, image_stream, format='png')
    image_stream.seek(0)
    image_base64 = base64.b64encode(image_stream.getvalue()).decode('utf-8')

    return image_base64


import plotly.graph_objects as go

def plot_emotions_monthly_analysis(journal_entries):
    # Define emotions and their corresponding colors
    emotions_colors = {
        "anger": "#A1CCD1",
        "joy": "#C8E4B2",
        "surprise": "#FDFFAE",
        "sadness": "#EF9595",
        "love": "#FFC0D9",
        "fear": "#E5E0FF"
    }

    # Count occurrences of each emotion for each date
    date_counts = {}
    for entry in journal_entries:
```

```
        predicted_emotion_label = entry.predicted_emotion
        date = entry.created_at.date()

        if date in date_counts:
            date_counts[date][predicted_emotion_label] =
date_counts[date].get(predicted_emotion_label, 0) + 1
        else:
            date_counts[date] = {emotion: 0 for emotion in emotions_colors.keys()}
            date_counts[date][predicted_emotion_label] = 1

    # Count total occurrences of each emotion
    counts = {emotion: sum(date_counts[date].get(emotion, 0) for date in date_counts.keys()) for
emotion in emotions_colors.keys()}

    # Filter out emotions with zero occurrences
    emotions_detected = [emotion for emotion in emotions_colors.keys() if counts[emotion] > 0]
    counts_detected = {emotion: counts[emotion] for emotion in emotions_detected}

    # Create a pie chart of emotion distribution
    fig = go.Figure(data=[go.Pie(labels=emotions_detected, values=[counts_detected[emotion] for
emotion in emotions_detected])])
    fig.update_traces(marker=dict(colors=[emotions_colors[emotion] for emotion in emotions_detected]),
pull=[0.1] * len(emotions_detected), textinfo='percent+label')

    # Update layout with chart title
    fig.update_layout(
        title='Emotion Distribution',
        title_x=0.5
    )

    # Convert the plot to a base64-encoded image
    image_stream = BytesIO()
    pio.write_image(fig, image_stream, format='png')
    image_stream.seek(0)
    image_base64 = base64.b64encode(image_stream.getvalue()).decode('utf-8')

    return image_base64
```

Urls.py

```
from django.contrib import admin
from django.urls import path, include
from . import views

from django.contrib.auth.views import (
    PasswordResetView,
    PasswordResetDoneView,
    PasswordResetConfirmView,
    PasswordResetCompleteView
)
urlpatterns = [
path('', views.home,name="home"),
path('signup', views.signup, name="signup"),
path('signin', views.signin, name="signin"),
path('signout', views.signout, name="signout"),
path('activate/<uidb64>/<token>', views.activate, name="activate"),
path('contact_submit/', views.contact_submit, name='contact_submit'),
    path('password-reset/',
        PasswordResetView.as_view(
            template_name='SoulGlow/password_reset.html',
            html_email_template_name='SoulGlow/password_reset_email.html'
        ),
        name='password-reset'
    ),
    path('password-reset/done/',
PasswordResetDoneView.as_view(template_name='SoulGlow/password_reset_done.html'),name='password_reset_
done'),
    path('password-reset-confirm/<uidb64>/<token>/',
PasswordResetConfirmView.as_view(template_name='SoulGlow/password_reset_confirm.html'),name='password_
reset_confirm'),
    path('password-reset-
complete/',PasswordResetCompleteView.as_view(template_name='SoulGlow/password_reset_complete.html'),na
me='password_reset_complete'),
    path('journal/', views.journal, name="journal"),
    path('save_journal/', views.save_journal, name='save_journal'),
    path('profile/', views.profile, name='profile'),
    path('prevjournal/', views.prevjournal, name='prevjournal'),
    path('delete_journal/<int:entry_id>/', views.delete_journal_entry,
name='delete_journal_entry'),#****
    path('delete/<str:pk>/', views.delete_account, name='delete_account'),
    path('settings', views.settings, name='settings'),
    path('analyze_emotions/', views.analyze_emotions, name='analyze_emotions'),
    path('get_affirmation/', views.get_affirmation, name='get_affirmation'),
]
```

Models.py

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

class JournalEntry(models.Model):
    # ForeignKey to link each journal entry to a user
    user = models.ForeignKey(User, on_delete=models.CASCADE)

    # BinaryField to store the text content of the journal entry
    text = models.BinaryField()

    # CharField to store the mood associated with the journal entry
    mood = models.CharField(max_length=50, null=True, blank=True)

    # ImageField to upload and store images related to the journal entry
    image = models.ImageField(upload_to='journal_images/', null=True, blank=True)

    # DateTimeField to store the date and time when the journal entry was created
    created_at = models.DateTimeField(auto_now_add=True)

    # CharField to store the predicted emotion for the journal entry
    predicted_emotion = models.CharField(max_length=50, null=True, blank=True)

    # FloatField to store the probability of the predicted emotion
    probability = models.FloatField(null=True, blank=True)

    # FloatField to store the prob of depression associated with the journal entry
    depression = models.FloatField(null=True, blank=True)



    # Method to return a string representation of the journal entry
    def __str__(self):
        return f"{self.user.username}'s Journal Entry"
```

Utils.py

```
import re
import nltk
import wordninja
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk import pos_tag
from .models import JournalEntry

nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')

def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return 'a'  # Adjective
    elif tag.startswith('V'):
        return 'v'  # Verb
    elif tag.startswith('N'):
        return 'n'  # Noun
    elif tag.startswith('R'):
        return 'r'  # Adverb
    else:
        return 'n'  # Default to noun if not found

def preprocess_text(text):
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer()

    # Convert text to lowercase
    text = text.lower()

    # Remove characters and symbols
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    contractions = {
        # ... (your contractions dictionary)
    }

    # Replace contractions
    for contraction, expanded_form in contractions.items():
        text = text.replace(contraction, expanded_form)

    # Tokenize the text into words using wordninja
    tokens = wordninja.split(text)
```

```
    # Perform POS tagging
    pos_tags = pos_tag(tokens)

    # Lemmatization
    lemmatized_tokens = []
    for word, tag in pos_tags:
        pos = get_wordnet_pos(tag)
        lemma = lemmatizer.lemmatize(word, pos=pos)
        lemmatized_tokens.append(lemma)

    # Remove stop words
    filtered_tokens = [word for word in lemmatized_tokens if word.lower() not in stop_words]

    # Remove duplicate words
    unique_tokens = []
    for word in filtered_tokens:
        if word not in unique_tokens:
            unique_tokens.append(word)

    # Recreate text from unique tokens
    filtered_text = ' '.join(unique_tokens)

    return filtered_text

import joblib
from sklearn.svm import SVC
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import make_pipeline

# Load the SVM model
svm_model_file = open("SoulGlow\\depression_svm_model.pkl", "rb")
svm = joblib.load(svm_model_file)
svm_model_file.close()

# Load the TF-IDF vectorizer
tfidf_vectorizer_file = open("SoulGlow\\depression_tfidf_vectorizer.pkl", "rb")
tfidf_vectorizer = joblib.load(tfidf_vectorizer_file)
tfidf_vectorizer_file.close()

def predict_depression(input_text):
    preprocessed_input = preprocess_text(input_text)
    input_vectorized = tfidf_vectorizer.transform([preprocessed_input]).toarray()
    prediction = svm.predict(input_vectorized)
    probability = svm.predict_proba(input_vectorized)
```

```
    if prediction[0] == 1:
        result = "The input text indicates depression."
    else:
        result = "The input text does not indicate depression."

    return result, probability
```

Admin.py

```
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from django.contrib.auth.models import User
from .models import JournalEntry

# Define an inline admin for the JournalEntry model
class JournalEntryInline(admin.TabularInline):
    model = JournalEntry
    extra = 0  # Remove the ability to add new JournalEntry instances
    list_display = ('text', 'mood', 'predicted_emotion', 'probability', 'depression', 'display_id',
'created_at')
    readonly_fields = ('text', 'mood', 'predicted_emotion', 'probability', 'depression', 'display_id'
, 'created_at')  # Make fields read-only
    can_delete = False  # Remove the ability to delete JournalEntry instances

    # Define a method to display the journal ID
    def display_id(self, instance):
        return f"Journal ID: {instance.id}"

    # Define a method to display the creation date
    def created_at(self, instance):
        return instance.created_at.strftime('%Y-%m-%d %H:%M:%S')  # Format the date as desired

    # Short descriptions for display purposes
    display_id.short_description = 'Journal ID'
    created_at.short_description = 'Created At'
    # Remove image field from the inline
    exclude = ('image',)
    # Override to disable the ability to add new user journals
    def has_add_permission(self, request, obj=None):
        return False
# Define a custom UserAdmin class
class CustomUserAdmin(UserAdmin):
    inlines = [JournalEntryInline]  # Add the JournalEntryInline to the UserAdmin
# Unregister the default UserAdmin
admin.site.unregister(User)
# Register the custom UserAdmin
admin.site.register(User, CustomUserAdmin)
# Customize the admin site header with your logo
admin.site.site_header = 'SoulGlow'
# Set site URL to None
admin.site.site_url = None
```

Settings.py

```
"""
Django settings for my_app project.

Generated by 'django-admin startproject' using Django 5.0.1.

For more information on this file, see
https://docs.djangoproject.com/en/5.0/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/5.0/ref/settings/
"""

from pathlib import Path
from . info import *
import os




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

#MEDIA_URL = '/media/'
#MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')



#sitting emails
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_USE_TLS = True
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_HOST_USER = 'soulglow.website@gmail.com'  # Your Gmail address
EMAIL_HOST_PASSWORD =  'zzrq hagv aeby agxh' # Your Gmail password

#from cryptography.fernet import Fernet

# # Generate a key
# key = Fernet.generate_key()

# # Convert the key to a string for storage or transmission
# key_string = key.decode()
```

```
# # Print or store the key securely
# print("Generated Key:", key_string)

# Secret key settings
SECRET_KEY = os.environ.get('SECRET_KEY')


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
#SECRET_KEY = 'django-insecure-ekshtvv&!)+k29i!6_8pusfqt!9+ccmk6mz*wl$*ww#=#%-rgy'

# 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',
    'SoulGlow.apps.SoulglowConfig', #


]

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 = 'my_app.urls'
```

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        "DIRS": ["templates"],  # new
        '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 = 'my_app.wsgi.application'


# Database
# https://docs.djangoproject.com/en/5.0/ref/settings/#databases

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


# Password validation
# https://docs.djangoproject.com/en/5.0/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',
    },
```

```
]


# Internationalization
# https://docs.djangoproject.com/en/5.0/topics/i18n/

LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'Asia/Riyadh'
USE_I18N = True
USE_TZ = True



# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.0/howto/static-files/

STATIC_URL = 'static/'
STATICFILES_DIRS = [
   os.path.join(BASE_DIR, "static"),
   ]
# Default primary key field type
# https://docs.djangoproject.com/en/5.0/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

Tokens.py

```
from django.contrib.auth.tokens import PasswordResetTokenGenerator
from six import text_type

class TokenGenerator(PasswordResetTokenGenerator):
    def _make_hash_value(self, user, timestamp):
        return text_type(user.pk) + text_type(timestamp)


# Instantiate the TokenGenerator
generate_token = TokenGenerator()
```