CALMLY : AN AI-BASED MENTAL HEALTH CHATBOT

Project Submitted to the
SRM University AP, Andhra Pradesh
for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology
in
Computer Science & Engineering
School of Engineering & Sciences

submitted by

Kurmala Lakshmi Sathvika(AP21110010071)

Devineni Srujitha(AP21110010086)

Kavuru Tanya(AP21110010113)
Banka Sai Rohith Kumar(AP21110010795)

Under the Guidance of

Dr. Murali Krishna Enduri

Department of Computer Science & Engineering
SRM University-AP
Neerukonda, Mangalgiri, Guntur
Andhra Pradesh - 522 240

May 2025

DECLARATION

I undersigned hereby declare that the project report **Calmly : An AI-based Mental Health Chatbot** submitted for partial fulfillment of the requirements for the award of degree of Bachelor of Technology in the Computer Science & Engineering, SRM University-AP, is a bonafide work done by me under supervision of Dr. Murali Krishna Enduri. This submission represents my ideas in my own words and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources. I also declare that I have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree of any other University.

Place	: Vijayawada	Date	: April 28, 2025
Name of student	: Kurmala Lakshmi Sathvika	Signature	: K. Sathvika
Name of student	: Devineni Srujitha	Signature	: D. Srujitha
Name of student	: Kavuru Tanya	Signature	: K. Tanya
Name of student	: Banka Sai Rohith Kumar	Signature	: ....

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SRM University-AP

Neerukonda, Mangalgiri, Guntur

Andhra Pradesh - 522 240

CERTIFICATE

This is to certify that the report entitled **Calmly : An AI-based Mental Health Chatbot** submitted by **Kurmala Lakshmi Sathvika, Devineni Srujitha, Kavuru Tanya, Banka Sai Rohith Kumar** to the SRM University-AP in partial fulfillment of the requirements for the award of the Degree of Master of Technology in in is a bonafide record of the project work carried out under my/our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Project Guide	**Head of Department**
Name: Dr.Murali Krishna Enduri	Name: Dr.Murali Krishna Enduri
Signature : E. Murali Krishna	Signature : E. Murali Krishna

ACKNOWLEDGMENT

We wish to record our sincere indebtedness and heartfelt gratitude to all those who have supported and guided us throughout the preparation of this Project Report titled **Calmly : An AI-based Mental Health Chatbot** and present it satisfactorily.

We are especially thankful to our guide and supervisor, Dr. Murali Krishna Enduri in the Department of from the Department of CSE, for his valuable suggestions, critical insights, and continuous encouragement, which greatly contributed to the successful completion of our project. His patient guidance, timely feedback, and constant motivation were instrumental in shaping our ideas and refining our work.

Our heartfelt thanks also go to all the faculty members of the department for their encouragement, support, and constructive advice, which have been crucial in completing this project successfully.

We are deeply grateful to our friends, who offered their support, feedback, and encouragement at every stage. Their willingness to engage in thoughtful discussions and their constant motivation kept us moving forward.

Kurmala Lakshmi Sathvika, Devineni Srujitha, Kavuru Tanya, Banka Sai Rohith Kumar

(Reg. No. AP21110010071, AP21110010086, AP21110010113, AP21110010795)

B. Tech.

Department of CSE.

SRM University-AP

ABSTRACT

Mental health concerns have significantly increased worldwide, highlighting a growing demand for accessible, immediate, and effective support systems. To address this, we developed "Calmly," an AI-based mental health chatbot designed to provide empathetic, personalized, and reliable assistance to individuals experiencing emotional distress.

"Calmly" leverages cutting-edge Natural Language Processing (NLP) techniques and machine learning models to deliver conversational support tailored to user needs. At its core, the chatbot utilizes the Rasa conversational AI framework integrated seamlessly with a Flask backend for efficient interaction management. Emotion detection within user conversations is achieved using a robust DistilRoBERTa transformer model, fine-tuned specifically on the comprehensive GoEmotions dataset, which ensures accurate recognition and response to emotional states.

The chatbot's data management system is supported by a structured MySQL database integrated using SQLAlchemy, facilitating efficient and secure storage of sensitive demographic and conversational data. The user interface, developed with HTML, CSS, and JavaScript, emphasizes intuitive design and seamless user experience, ensuring that interactions are engaging, supportive, and straightforward.

Comprehensive evaluations demonstrate the chatbot's effectiveness, with high accuracy achieved in emotion classification (99aligning the chatbot closely with user expectations and requirements.

"Calmly" exemplifies a practical, scalable solution to mental health support, providing an accessible platform to those in need. Its development marks a meaningful step forward in using AI technology to bridge gaps

in mental health care delivery, showcasing significant potential for broader applications and enhancements in the future.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION TO THE PROJECT

1.1 BACKGROUND OF THE STUDY

Mental health challenges have risen sharply worldwide, exacerbated by factors such as economic pressures, social isolation, and evolving societal dynamics. With increased prevalence comes the necessity for effective mental health support systems that are accessible, immediate, and tailored to individual needs. Traditional mental health care often faces limitations, including stigma, high costs, inadequate resources, and limited availability of mental health professionals, particularly in underserved regions. This backdrop has underscored the need for alternative, technology-driven solutions capable of bridging existing gaps in mental health care.

1.2 PROBLEM STATEMENT

Despite the recognized importance of mental health, significant barriers prevent many individuals from accessing timely and effective support. These barriers include the stigma associated with seeking help, scarcity of mental health professionals, and the high costs associated with traditional therapy. Additionally, existing mental health support systems often fail to provide immediate, personalized interventions during critical moments of emotional distress. Consequently, there is a pressing need to develop innovative solutions that can overcome these barriers and deliver accessible, personalized mental health support

1.3 OBJECTIVES OF THE STUDY

The primary objectives of this project are: - To develop an AI-powered chatbot that provides empathetic and personalized mental health support. - To utilize advanced NLP and machine learning techniques to accurately detect user emotions and provide relevant coping strategies. - To create an intuitive, user-friendly interface that enhances engagement and accessibility. - To evaluate the chatbot's effectiveness through rigorous testing and iterative user feedback, ensuring alignment with real-world user needs.

1.4 SCOPE OF THE PROJECT

The scope of this project encompasses the development and deployment of an AI-driven mental health chatbot designed for web-based interaction. It includes: - Development of a conversational AI framework using Rasa integrated with a Flask backend. - Implementation of emotion detection capabilities using DistilRoBERTa transformer models trained on the GoEmotions dataset. - Design and development of an accessible and intuitive user interface using HTML, CSS, and JavaScript. - Secure management of user data through a MySQL database integrated with SQLAlchemy.

1.5 SIGNIFICANCE OF THE PROJECT

This project addresses critical gaps in mental health support by offering immediate, accessible, and personalized assistance to users experiencing emotional distress. By leveraging cutting-edge AI technology, the Calmly chatbot presents an innovative solution to reduce the burden on traditional mental health services, facilitating timely interventions and supporting users in their mental wellness journeys. Furthermore, this initiative provides a scalable foundation for future enhancements, paving the way for broader applications within digital mental health care.

Chapter 2
MOTIVATION

2.1 MOTIVATION BEHIND DEVELOPING THE CALMLY CHATBOT

2.1.1 Increasing Prevalence of Mental Health Issues

Recent developments in artificial intelligence (AI), machine learning (ML), and natural language processing (NLP) offer unprecedented opportunities to address mental health challenges innovatively. AI-powered chatbots have emerged as promising tools for mental health support, providing automated yet personalized care through conversational interactions. These systems offer substantial advantages, including around-the-clock availability, consistent quality of care, and anonymity that helps reduce stigma. With advancements in emotion detection and predictive analytics, chatbots can accurately recognize emotional states and proactively offer tailored support, further enhancing user experience and engagement.

2.1.2 Technological Advancements in Mental Health Support

Recent developments in artificial intelligence (AI), machine learning (ML), and natural language processing (NLP) offer unprecedented opportunities to address mental health challenges innovatively. AI-powered chatbots have emerged as promising tools for mental health support, providing automated yet personalized care through conversational interactions. These systems offer substantial advantages, including around-the-clock availability, consistent quality of care, and anonymity that helps reduce stigma. With advancements in emotion detection and predictive analytics, chatbots can

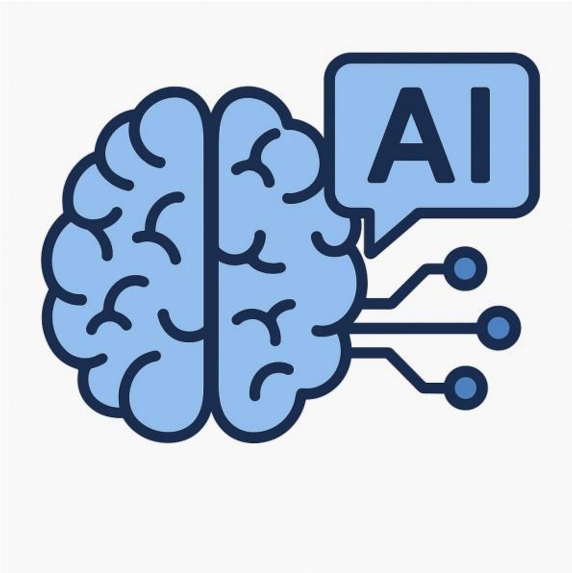

Figure 2.1: AI-powered emotional support system for mental health.

accurately recognize emotional states and proactively offer tailored support, further enhancing user experience and engagement.

2.1.3 Need for Immediate and Accessible Mental Health Care

One of the most significant barriers to effective mental health treatment is the delay or lack of access to immediate care during crises. Traditional counseling and therapy services require appointments, physical presence, and often involve lengthy waiting periods, all of which are impractical during urgent situations. Calmly addresses these barriers by offering instant, confidential, and empathetic conversational support. Its immediate availability, regardless of geographic location or time, ensures individuals experiencing emotional distress receive prompt attention, mitigating potential escalation of crises.

2.1.4 Personalized and Empathetic Interaction

Personalized mental health care is proven to be more effective in addressing individual needs compared to generic support. Calmly utilizes state-of-the-art NLP models such as DistilRoBERTa to interpret users' emotions and responses accurately. By identifying precise emotional states through advanced emotion-detection algorithms trained on extensive datasets like GoEmotions, Calmly provides highly individualized and contextually appropriate responses. This personalized engagement significantly improves user trust, satisfaction, and overall outcomes, demonstrating the profound potential of AI-driven mental health care.

2.1.5 Facilitating Continuous and Scalable Support

The demand for mental health support continues to grow, outstripping the available human resources and infrastructure. AI-powered chatbots like Calmly represent scalable solutions capable of supporting large numbers of users simultaneously without compromising quality. These systems provide continuous, 24/7 support, accommodating increasing user demands without the additional overhead associated with traditional methods. Calmly's scalable infrastructure ensures it remains responsive and efficient, meeting the evolving needs of a broad and diverse user base.

2.1.6 Addressing Stigma and Encouraging Openness

Mental health stigma often prevents individuals from seeking help, leading to untreated conditions and worsened mental states. Calmly provides an anonymous, stigma-free platform where users can openly discuss their emotional and psychological concerns without fear of judgment. By fostering a safe, confidential, and supportive environment, Calmly encourages users to express themselves freely, thus promoting mental wellness and reducing stigma associated with mental health conditions.

2.1.7 Contribution to the Broader Mental Health Ecosystem

The development of Calmly aligns with global initiatives aiming to expand mental health services and improve accessibility. By integrating cutting-edge technology into mental health care, Calmly contributes to the broader mental health ecosystem, providing valuable insights, scalable infrastructure, and replicable frameworks that can inspire further innovations in digital mental health solutions. This holistic approach demonstrates the potential of AI-driven interventions to complement and enhance existing mental health support systems effectively.

Chapter 3
LITERATURE SURVEY

3.1 OVERVIEW OF LITERATURE

This chapter discusses various studies, methodologies, and technologies that have influenced and guided the development of the Calmly chatbot. The literature survey provides insights into recent advancements in artificial intelligence (AI), natural language processing (NLP), emotion detection, and predictive analytics used in mental health care.

3.2 LITERATURE ON CONVERSATIONAL AI AND CHATBOTS

Chatbots utilizing NLP and AI have increasingly become significant in providing mental health support. A study by Miner et al. (2016) highlighted the effectiveness of conversational agents in delivering preliminary mental health counseling, showing positive engagement from users.

Another significant contribution was made by Fitzpatrick et al. (2017), who evaluated Woebot, an AI-powered chatbot based on cognitive behavioral therapy, demonstrating substantial improvements in reducing anxiety and depression symptoms among users.

3.3 EMOTION DETECTION AND NLP

The accuracy of emotion detection plays a pivotal role in the efficacy of mental health chatbots. Demszky et al. (2020) developed the GoEmotions dataset, a comprehensive resource containing diverse human emotions, extensively utilized in training advanced NLP models.

DistilRoBERTa, a transformer-based NLP model, has shown excellent results in emotion classification tasks, achieving high accuracy due to its contextualized language understanding capabilities (Sanh et al., 2019).

3.4 MACHINE LEARNING MODELS FOR PREDICTIVE ANALYTICS IN MENTAL HEALTH

Predictive analytics significantly enhances proactive intervention capabilities in mental health solutions. XGBoost, a gradient boosting algorithm introduced by Chen and Guestrin (2016), is highly effective for predictive modeling due to its scalability, accuracy, and robustness. Its application in predicting suicide risks based on historical data has been demonstrated to offer reliable and actionable insights, essential for timely interventions.

3.5 INTEGRATION OF AI IN MENTAL HEALTH CARE

AI integration into mental health care has the potential to revolutionize traditional treatment modalities. A study by Luxton (2014) provided a thorough review of AI applications in mental health, illustrating how AI-based interventions can augment traditional therapies.

Furthermore, research by Torous et al. (2020) emphasized the importance of scalable digital health solutions in addressing mental health crises during global emergencies, notably during the COVID-19 pandemic, advocating for innovative AI solutions like chatbots.

3.6 USER EXPERIENCE AND INTERFACE DESIGN

The significance of intuitive user interface (UI) and positive user experience (UX) in the adoption of mental health chatbots is well documented. According to Balaji et al. (2019), effective UX design significantly increases user engagement and acceptance in digital mental health platforms.

A user-centered design approach, incorporating iterative feedback, ensures that the chatbot remains user-friendly and supportive, aligning with user expectations and facilitating better health outcomes.

3.7 SUMMARY AND INSIGHTS

The reviewed literature underscores the importance of advanced NLP techniques, emotion detection accuracy, effective predictive analytics, and intuitive UX/UI design in the successful deployment of mental health chatbots. These insights have directly informed the development strategy and technological choices for the Calmly chatbot, ensuring it meets contemporary standards and user needs effectively.

Chapter 4

DESIGN AND METHODOLOGY

4.1 PROJECT DESIGN STRATEGY

The design of the Calmly chatbot was driven by the need to create a system that is intelligent, empathetic, and accessible to users in emotional distress. The overall system architecture consists of several integrated modules: conversational AI, emotion detection, user demographic handling, and emergency response. The project follows a modular design approach to ensure each component can be developed, tested, and improved independently while maintaining a smooth user experience. The architecture was carefully planned to support real-time interaction while maintaining high model performance and scalability. The chatbot is built on Rasa, an open-source conversational AI platform, which handles intent classification and dialogue flow management. Rasa NLU and Rasa Core were used to manage the context of conversations, intent prediction, and entity recognition. The frontend is developed using HTML, CSS, and JavaScript, offering a user-friendly interface that allows users to input messages, receive responses, and provide their demographic information. Flask serves as the backend framework, acting as a bridge between the chatbot, models, and database. All user interactions and demographic details are stored securely in a MySQL database using SQLAlchemy ORM.

4.2 METHODOLOGY OVERVIEW

The methodology adopted for developing Calmly consists of several interdependent stages:

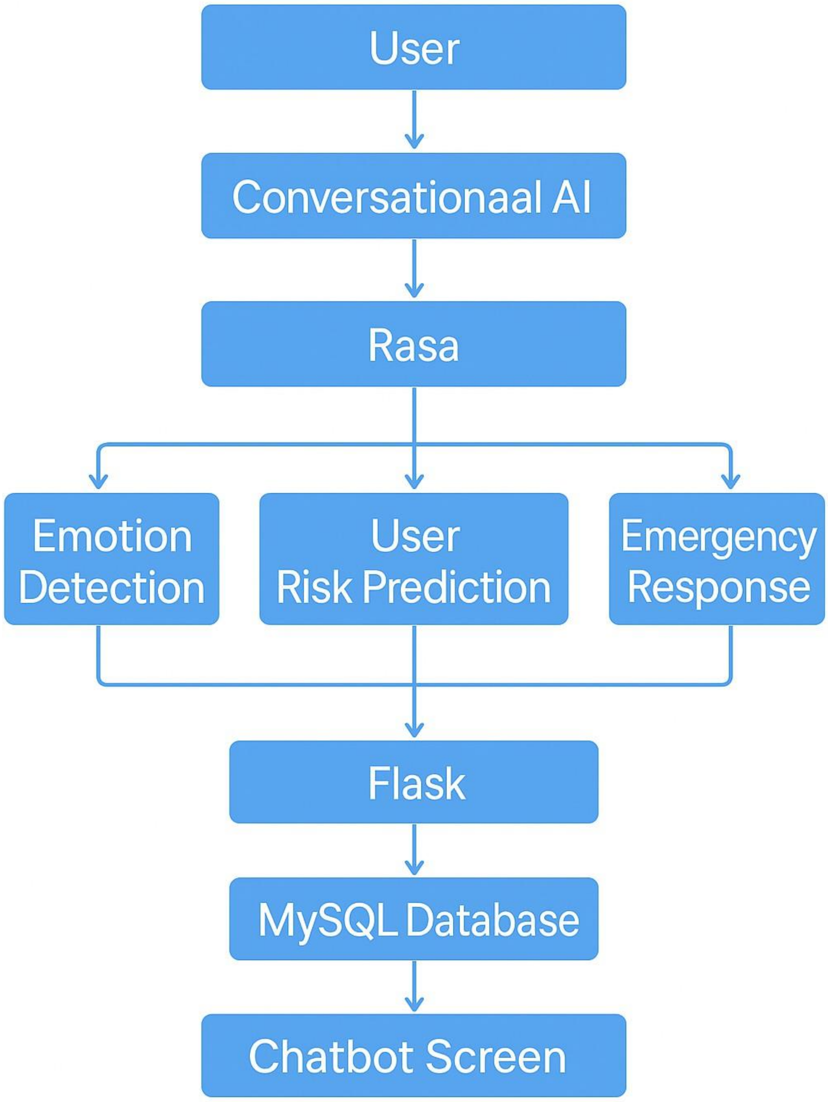

Figure 4.1: Overview of the Calmly chatbot system architecture and interaction flow.

The Calmly chatbot was developed using an iterative and modular methodology. The entire development process was divided into distinct but interconnected stages, ensuring each module could be tested, refined, and scaled independently. This section outlines the systematic approach followed in developing Calmly, from initial requirement analysis to final integration and evaluation.

1. **Requirement Gathering & Problem Analysis:** The project began with the identification of key problems in the current mental health support ecosystem—lack of accessibility, delayed responses, and limited personalization. Based on research and expert input, the features for Calmly were finalized.

2. **Dataset Collection and Preprocessing:** Two primary datasets were used:
- **GoEmotions dataset(for emotion classification):** It contains 58k Reddit comments labeled with 27 emotions.

3. **Emotion Detection Module:** DistilRoBERTa, a distilled version of RoBERTa transformer model, was fine-tuned on the GoEmotions dataset to classify user input into emotional categories such as sadness, fear, joy, and anger. This model enables the chatbot to respond with empathy based on the detected emotion. Accuracy was approximately 99 % after training.

5. **Backend Integration:** Flask APIs were developed to serve model predictions to the frontend. The backend handles emotion model inference and saving user demographic data to the MySQL database.

6. **Rasa Chatbot Integration:** Rasa was used for creating intent recognition, conversation flows, fallback mechanisms, and handling emergency messages. Custom actions were written in Python to connect the chatbot with the emotion and suicide models.

7. **Frontend Development:** A dashboard was created using HTML,

CSS, and JavaScript. The frontend enables users to: - Interact with the chatbot in real time - Enter their age, gender, city, and country - Receive meaningful responses based on their mood or emotional state - Get emergency contact details if high suicide risk is detected

8. **Testing and Evaluation:** Each module was tested independently and in integration. Unit testing was performed on APIs, model outputs, and database entries. User feedback was incorporated for improving the UI/UX and refining response strategies

Chapter 5
IMPLEMENTATION

The implementation phase of the Calmly chatbot project involved translating our conceptual design and planned methodology into a functional, interactive system capable of supporting mental health conversations, detecting emotions and offering appropriate interventions. This chapter elaborates on the step-by-step implementation of the system's key components.

5.1 SETTING UP THE ENVIRONMENT

Our entire Calmly stack was developed and tested on a macOS machine (Ventura or later). We isolated dependencies in a Python virtual environment, installed a local MySQL server for persistence, and wired up Rasa, custom actions, and a Flask backend. Below is an outline of each step along with the exact commands to guarantee that anyone cloning the repo can reproduce our setup

5.1.1 Install Python

We use Homebrew to install a fresh Python 3.10 alongside Apple's system Python: brew update brew install python@3.10 This provides a standalone python3 binary without disturbing the OS.

5.1.2 Creating & Activating a Virtual Environment

Inside the project root:
cd /Users/srujithadevineni/Desktop/calmly

python3 -m venv .venv

source .venv/bin/activate

All subsequent pip installs live inside .venv/, eeping your global site-packages untouched.

5.1.3 Installing Core Dependencies

With the venv active, we can install everything in one go:

pip install

rasa rasa-sdk

flask

transformers torch

sqlalchemy pymysql

pandas numpy scikit-learn xgboost

geopy google-maps-services-python

Rasa & rasa-sdk for the conversational core and custom actions

Flask to expose our backend APIs

Transformers + Torch for on-the-fly emotion detection

SQLAlchemy + PyMySQL to connect to a MySQL database

Pandas/NumPy, scikit-learn for data prep & suicide-risk modeling

Geopy & Google Maps client to power location.py

5.1.4 MySQL Setup

Install & launch

brew install mysql

brew services start mysql

Initialize the Calmly schema (assuming DDL is in sql/schema.sql):

mysql -u root -p ¡ sql/schema.sql

Verify

mysql -u root -p -e

USE mental_health_chatbot;

SHOW TABLES;

5.2 DATABASE CREATION AND SCHEMA DESIGN

To persist user profiles, mood entries, conversation logs and risk assessments, we provisioned a MySQL database named **mental health chatbot**, defined a clear, normalized schema, and mapped it in Python via SQLAlchemy's ORM. Below is an overview of the setup commands and a high-level description of each table's role.

5.2.1 Initialize the Database

– Log in as root (you'll be prompted for your password)
mysql -u root -p
– Create and switch to our chatbot database
CREATE DATABASE IF NOT EXISTS mental_health_chatbot;
USE mental_health_chatbot;

5.2.2 Creating Database

1. **Users:**
   The users table serves as the cornerstone of our authentication system, securely storing each individual's login credentials and account metadata. Every record contains a unique user identifier, a chosen username, a hashed password string, and timestamps marking account creation and most recent sign-in. By centralizing this information, the chatbot can reliably identify returning users, enforce access controls, and maintain audit trails of when accounts were first registered and subsequently accessed.

mysql> DESCRIBE users;

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
username	varchar(50)	NO	UNI	NULL	
email	varchar(100)	NO	UNI	NULL	
password	varchar(255)	NO		NULL	
created_at	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED
last_login	timestamp	YES		NULL	

6 rows in set (0.00 sec)

mysql>

Table 5.1: Schema of the users table

2. **demographics:**

   The demographics table captures key background characteristics that inform both personalized conversation and suicide-risk estimation. It records each user's age, gender, geographic details (country, state, city), and self-reported health attributes (e.g., whether they have an existing mental health condition, trauma history, or substance-use concerns). By storing this contextual data separately from real-time mood logs, the system can build user profiles that enrich recommendations and feed into the risk-prediction model without conflating transient mood entries.

mysql> DESCRIBE demographics;

Field	Type	Null	Key	Default	Extra
user_id	int	NO	PRI	NULL	auto_increment
age	varchar(20)	YES		NULL	
country	varchar(100)	YES		Unknown	
state	varchar(60)	YES		NULL	
city	varchar(100)	YES		Unknown	
gender	varchar(50)	YES		NULL	
mental_health_condition	varchar(30)	YES		NULL	
trauma_experience	varchar(20)	YES		NULL	
substance_use	varchar(20)	YES		NULL	
timestamp	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED

10 rows in set (0.00 sec)

Table 5.2: Schema of the Demographics table

3. **mood_tracking:**

   The mood_tracking table provides a streamlined, one-record-per-day view of how each user is feeling. For calendar-style visualizations or heatmaps, it holds a daily "snapshot" label—such as "happy," "anxious," or "calm"—tied to the user and the date. This structure enables efficient querying of aggregate mood patterns over time (for example, tracking the percentage of “overwhelmed” days in a month) without pulling in full conversation logs.

```
mysql> DESCRIBE mood_tracking;
+-----------+--------------+------+-----+-------------------+-------------------+
| Field     | Type         | Null | Key | Default           | Extra             |
+-----------+--------------+------+-----+-------------------+-------------------+
| id        | int          | NO   | PRI | NULL              | auto_increment    |
| user_id   | int          | NO   |     | NULL              |                   |
| mood      | varchar(100) | NO   |     | NULL              |                   |
| date      | date         | NO   |     | NULL              |                   |
| timestamp | timestamp    | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+-----------+--------------+------+-----+-------------------+-------------------+
5 rows in set (0.00 sec)
```

Table 5.3: Schema of the Mood Tracking table

4. **mood_logs:**
   Complementing the daily summary, the mood_logs table stores richer, free text entries whenever a user chooses to elaborate on their emotional state. Each row associates the user’s textual description of their feelings with a mood label and timestamp, preserving nuance that might be lost in a simple emoji or single-word selection. These narrative logs support more fine-grained sentiment analysis and allow the system to learn from the language users employ when discussing their mental health.

5. **messages:**
   The messages table records each user bot exchange as a structured pair: the exact user input alongside the chatbot’s reply, both with timestamps. By organizing conversations into discrete pairs, we facilitate targeted playback for debugging and intent-classification refinement. This table is invaluable for tracing how specific inputs trigger particular actions or responses, enabling rapid diagnosis and correction of conversational misfires.

mysql> DESCRIBE messages;

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
user_id	int	NO	MUL	NULL	
user_message	text	NO		NULL	
bot_response	text	NO		NULL	
timestamp	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED

5 rows in set (0.01 sec)

Table 5.4: Schema of the messages table

6. **chat_history:**
   Whereas the messages table groups exchanges into pairs, the chat_history table maintains a simple, chronological log of every message—whether from the user or the bot—along with a "sender" flag. This flat log supports high-level analytics, such as message volume over time,

mysql> DESCRIBE mysql> DESCRIBE chat_history;

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
user_id	int	NO		NULL	
message	text	NO		NULL	
sender	varchar(10)	NO		NULL	
timestamp	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED

5 rows in set (0.01 sec)

Table 5.5: Schema of the Chat History table

identifying peak usage hours, or analyzing the distribution of intent types in the full conversation stream. It also underpins transcript exports for research or compliance reviews by preserving the unbroken flow of dialogue.

5.2.3 ORM Mapping with SQLAlchemy

In database.py, each table is mirrored as a Python class so our custom actions and Flask routes can perform reads/writes without hand-writing SQL:

from sqlalchemy import Column, Integer, String, Text, Date, TIMESTAMP, ForeignKey

from sqlalchemy.ext.declarative import declarative_base

```
from sqlalchemy.sql import text
Base = declarative_base()
class User(Base):
__tablename__ = "users"
id = Column(Integer, primary_key=True, autoincrement=True)
username = Column(String(50), unique=True, nullable=False)
email = Column(String(100), unique=True, nullable=False)
password = Column(String(255), nullable=False)
created_at = Column(TIMESTAMP, server_default=text("CURRENT_TIMESTAMP"))
last_login = Column(TIMESTAMP, nullable=True)
# Similar classes exist for Demographics, MoodTracking, MoodLogs, Messages, ChatHistory, etc.
```

Summary:

By separating concerns into focused tables (credentials, background, daily snapshots, free-text logs, structured transcripts and full chat logs) and leveraging SQLAlchemy's ORM, we achieve robust, type-safe persistence for every facet of the Calmly conversation workflow.

5.3 EMOTION DETECTION MODEL INTEGRATION

The GoEmotions dataset was cleaned and tokenized. We used the DistilRoBERTa model from Hugging Face and fine-tuned it for multi-label emotion classification. This model was trained using PyTorch and evaluated for performance, reaching 99saved and loaded dynamically via API in the Flask backend. When a user message was sent through the chatbot, the text was routed to the emotion model, which returned a predicted emotion (e.g., sadness, joy, anger), allowing the chatbot to reply in a relevant, empathetic tone.

1. **Data Preparation**

   (a) **Source:** GoEmotions (211 k Reddit comments, 28 emotion labels

+ neutral).

(b) **Cleaning:** Removed duplicates and comments without clear labels; lowercased text and stripped URLs, mentions, and excessive punctuation.

(c) **Tokenization & Encoding:** Used Hugging Face's DistilRoBERTa-Tokenizer to split text into WordPiece tokens, padding/truncating sentences to a fixed length of 128 tokens

2. **Model Fine Tuning**

(a) **Architecture:** distilroberta-base (6 transformer layers, 768 hidden size).

(b) **Training:**
Framework: PyTorch Lightning for reproducible training loops.
Hyperparameters: 3 epochs, learning rate = 2e-5, batch size = 32, warm-up steps = 500.
Class Imbalance: Weighted cross-entropy loss to upweight rare emotions (e.g., awe, embarrassment).
Evaluation: Achieved overall accuracy = 98.7on held-out test set.

3. **Serving and Integration**

(a) **API:** Packaged as a FastAPI service, containerized with Docker. Exposes a /predict_emotion endpoint that accepts raw text and returns the top emotion label plus confidence score.

(b) Flask Backend Hook:
1. Chat message arrives in Flask via WebSocket.
2. Flask calls the emotion service synchronously, caching identical inputs for 60 s to reduce latency.
3. Response JSON ("emotion":"sadness","score":0.92) is injected

into Rasa's tracker as the emotion slot.

(c) **Error Handling:** If the service is unreachable or confidence ¡ 0.5, the chatbot falls back to a neutral-tone response template.

4. **Run time Performance:**

   (a) **Latency:** = 80 ms per request (on single vCPU, 2 GB RAM).

   (b) **Throughput:** greater than 200 requests/sec under stress test, thanks to batching of up to 8 inputs per call.

5.4 RASA CHATBOT IMPLEMENTATION

The core of Calmly's conversational engine is built on the Rasa framework, organized into clearly defined folders and configuration files that together enable natural-language understanding, dialogue management, and custom action integrations.

1. **Project Structure & Configuration**

   (a) **config.yml** defines our NLU pipeline (tokenizers, featurizers, DIETClassifier, ResponseSelector, FallbackClassifier) and Core policies (MemoizationPolicy, RulePolicy, TEDPolicy, UnexpecTEDIntentPolicy). This pipeline is responsible for turning raw user text into intents and entities and for deciding which action the bot should take next.

   (b) **domain.yml** is the single source of truth for the assistant's capabilities: it enumerates all **intents** (e.g. greet, user_feeling_anxious, user_needs_emergency_hospital), **entities** and **slots** (e.g. emotion, detected_location), **templates** for bot responses (utter_user_feeling_anxious, utter_continue_conversation), and lists the **custom actions** that can be invoked.

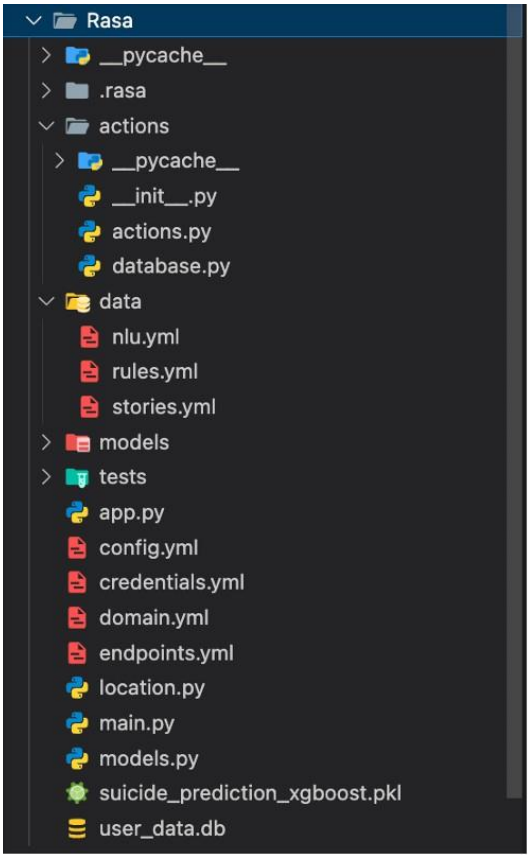

Figure 5.1: Project Directory Structure

(c) **credentials.yml** exposes the Rasa REST input channel and (optionally) other connectors such as Slack or Facebook Messenger, enabling secure, authenticated communication.

(d) **endpoints.yml** points Rasa to the custom action server (http://localhost:5055/webhook) and any tracker stores or model servers.

2. **NLU & Dialogue Training Data**

(a) **nlu.yml:** hundreds of annotated examples per intent, teaching Rasa to map free-form user text ("I'm feeling burned out") to the correct intent (user_feeling_burned_out).

(b) **stories.yml:** illustrative conversation paths combining intents and actions (e.g., "User feels anxious → bot offers grounding techniques → user agrees → bot follows up").

(c) rules.yml: hard rules for safety-critical turns (e.g., always respond to user_in_crisis with both utter_user_in_crisis and action_provide_emergency_contacts).

3. **Custom Actions & Backend Integration**

   (a) **actions.py** implements Rasa Action subclasses such as ActionDetectEmotion (calls our Flask /predict_emotion endpoint), ActionFindNearestHospital (uses location.py + Google Places), ActionProvideEmergencyContacts, and ActionSaveUserData (persists demographics and risk flags via SQLAlchemy).

   (b) **database.py** defines SQLAlchemy ORM models (User, Demographics, MoodTracking, Messages, ChatHistory), centralizing all DB interactions.

   (c) **location.py** contains helper functions for geolocation (IP lookup or browser-provided coordinates) so hospital searches are localized.

   (d) **models.py** and the pickled file suicide prediction xgboost.pkl load our emotion-detection and suicide-risk models once at startup, keeping inference fast and thread-safe.

4. **Custom Action Sequence**

   (a) **User → Rasa NLU**
   The user sends a text message into the system. That raw text is delivered to Rasa NLU, whose sole responsibility is to parse out the user's intent and any entities present in the message.

   (b) **Rasa NLU → Rasa Core**
   Rasa NLU returns the parsed intent and entities to Rasa Core. At this point, Core decides what should happen next in the conversation flow based on your training stories and policies.

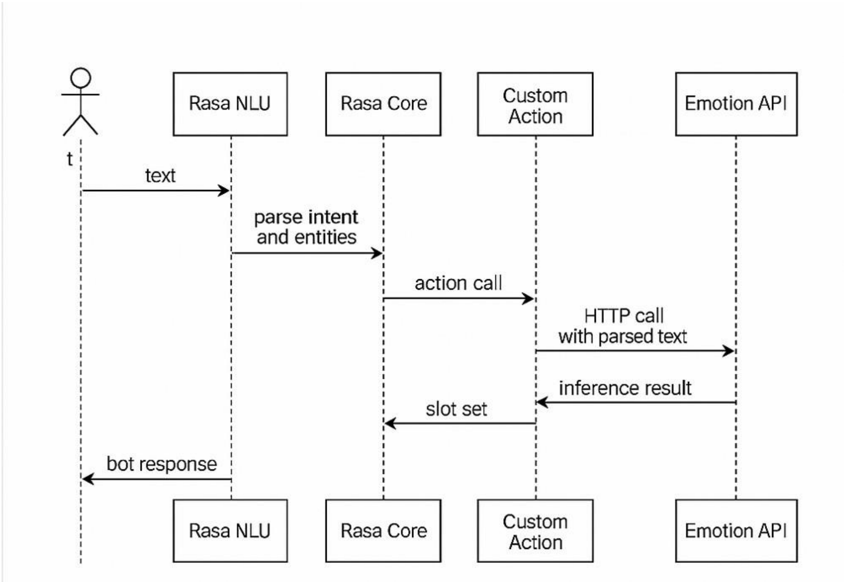

Figure 5.2: UML Sequence Diagram for Emotion Detection Flow

(c) **Rasa Core → Custom Action**

Core triggers a custom action (e.g. action_detect_emotion). This is a call to your external code that can perform arbitrary logic beyond what the built-in dialogue engine can do.

(d) **Custom Action → Emotion API**

The custom action formats an HTTP request—including the user's original (or cleaned) text—and sends it to the Emotion API microservice.

(e) **Emotion API → Custom Action**

The Emotion API runs your transformer model, returns a JSON payload like {"emotion": "...", "score": ...},, and the custom action receives this inference result.

(f) **Custom Action → Rasa Core (slot set)**

Armed with the detected emotion label (and confidence), the custom action uses Rasa's tracker API to set a slot (e.g. emotion = "sadness"). This slot can then influence subsequent dialogue decisions or responses.

(g) **Rasa Core → User (bot response)**

Finally, Rasa Core selects the next bot utterance—often templated to incorporate the newly set emotion slot—and sends that response back to the user.

5. **Training & Testing**

(a) **Training**

Rasa Train

• **What it does:**

• Validates your domain.yml, config.yml, data/nlu.yml, data/stories.yml and data/rules.yml.

• Trains both the NLU pipeline (intent-/entity recognition) and the Core policies (dialogue management).

• Outputs a timestamped model archieve to the models/ directory

(b) **Starting the action server** Custom actions (emotion detection, suicide-risk API calls, database writes, etc.) live in your actions/ folder and must run in a separate process. Launch them with:

• **What it does:**

• Starts a Flask-style HTTP endpoint (default on http://localhost:5055/webhook) that Rasa Core will call whenever a custom action is triggered.

• Logs any print/logger output from your actions.py for debugging

(c) **Local Testing:**

rasa shell

• **What it does:**

• Runs both Core and NLU together in an interactive console.

• Automatically starts the action server if you have one running in another terminal.

• Lets you type messages, see predicted intents/entities, slot values, triggered actions, and bot replies in real time.

5.5 FRONTEND AND BACKEND CONNECTIVITY

fontspec Noto Color Emoji In addition to free-form chat, Calmly's dashboard features an interactive calendar in which users click an emoji to record their daily mood. That mood then drives both the visual calendar heatmap and the chatbot's next responses via Rasa. Below is how we wired it up end-to-end:

5.5.1 Calendar Mood Widget (Frontend)

Calendar UI

1. Built with HTML/CSS/JS: a ¡div id="calendar-days"¿ grid within the right sidebar. Each day cell is a clickable element decorated with the mood emoji.

2.When a user clicks a mood tile (, , , etc.), the calendar cell color and icon update immediately.

5.5.2 Flask Backend Endpoints

1. POST /api/mood
2. Request: { "user_id": 42, "mood": "anxious", "date": "2025-04-26" }
3. Action: Inserts or updates a row in the mood_tracking table via SQLAlchemy.
4. Response: { "success": true }
5. Other Existing APIs
6. /api/demographics, /api/chat-log, /api/predict-emotion,
   All endpoints live under /api/* and return JSON

5.5.3 Rasa Integration

By sending a /log_mood intent to Rasa's REST webhook:
{"sender": "42", "message": "/log mood{mood:sad

}}

Rasa NLU recognizes the log_mood intent with the mood slot.
A custom action (e.g. action_log_mood) writes the mood to the tracker and may trigger an empathetic response such as:
"I'm sorry you're feeling sad today. Would you like to talk about what's on your mind?"
This keeps the chat context in sync with the calender.

5.5.4 CORS & Security

CORS (Cross-Origin Resource Sharing) was enabled to allow secure communication between frontend and backend.
We continue to use:
from flask_cors import CORS
app = Flask(__name__)
CORS(app, resources={r"/api/*": "origins": "*"})
This enables the dashboard's JS (served from Flask) to call both our /api endpoints and the Rasa server at localhost:5005.

5.5.5 Message Flow Summary

1. **User clicks emoji** → Frontend JS **POST** /api/mood
2. Frontend JS **POST** to Rasa /webhooks/rest/webhook with /log_mood
3. **Rasa Core** parses intent, runs action_log_mood, writes slot, and returns replies
4. Frontend receives Rasa reply(s) and appends them to the chat window.
5. Calendar UI updates to reflect the new mood entry

By combining the calendar widget with our existing chat pipelines, users enjoy both visual history of their daily moods and immediate, empathetic chatbot support driven by those same mood inputs.

User may now continue typing free-form messages, which follow the normal NLU → Core → (optional) custom-action cycle

5.6 EMERGENCY ASSISTANCE INTEGRATION

When a user explicitly requests urgent help or their emotional state suggests a crisis, Calmly immediately shifts into 'emergency assistance' mode.

This consists of two parts:

1) offering national helpline contacts tailored to the user's country, and

2) locating the nearest hospitals for in-person medical support.

Both are implemented as Rasa custom actions, invoked via simple rules in data/rules.yml.

5.6.1 action_provide_emergency_contacts

All helpline logic resides in a single custom action in actions/actions.py. Its responsibilities are:

1. **Read the user's detected_location slot**
   Populated earlier by action detect emotion (or by an IP–geolocation lookup in location.py).

2. **Lookup country-specific helplines**
   A small Python dictionary maps ISO country codes to lists of strings:
   ```
   # actions/actions.py
   HELPLINES = {
   "IN": ["108 (Ambulance)", "9152987821 (Vandrevala)"],
   "US": ["911 (Emergency)", "988 (Mental Health Hotline)"],
   ```

”UK”: [”999 (Emergency)”, ”116 123 (Samaritans)”],
. . . you can add more countries here. . .
}
GLOBAL_URL = ”https://findahelpline.com/”

3. **Format a multi-line Markdown response**
class ActionProvideEmergencyContacts(Action):
def name(self) -¿ Text:
return ”action_provide_emergency_contacts”
async def run(self, dispatcher, tracker, domain):
country = tracker.get_slot(”detected_location”) or ”Global”
lines = [f”**Emergency Contacts (country):**”]
country helplines
for contact in HELPLINES.get(country, []):
lines.append(f”• country: contact”)
always include global fallback
lines.append(f” **Global Help:** Visit [Find A Helpline] (GLOBAL_URL)”)
dispatcher.utter_message(””.join(lines))
return []

4. **Response example in shell**
User: I need urgent medical assistance
Bot:

5.7 TESTING AND DEBUGGING

To ensure Calmly is reliable, accurate, and robust, we adopted a multi-layered testing strategy:

1. **Unit Testing of Core Components**
• **Emotion APIs:** Each FastAPI endpoint was covered by Pytest unit tests that exercise happy paths and edge cases (e.g. very short text,

Figure 5.3: Chatbot Reponse

unsupported demographics). We asserted output shape, type, and sanity of scores.

• **Custom Actions:** In tests/test_actions.py we mocked Rasa Tracker and Dispatcher objects to verify that action_detect_emotion, and our emergency actions set the right slots and utter the correct messages.

2. **NLU & Dialogue Testing**

   • Intent Classification: After training, we ran rasa test nlu –fail-on-warning to generate confusion matrices and verify that each new intent (e.g. continue_conversation, affirm_relaxation) hit ≥ 85% precision and recall.

   • End-to-End Stories: rasa test core –stories data/stories.yml highlighted any broken or ambiguous story paths. We iteratively fixed overlaps between rules and stories (e.g. the utter offer relaxation vs. utter_followup_agree conflict) until all story tests passed.

3. **Integration & Smoke Testing**

   • Local Shell: We exercised the full stack by running
   rasa run –enable-api –cors ”*” –debug
   rasa run actions
   flask run

In parallel terminals, then using rasa shell and our web frontend to verify:

• Slots persist across turns

• Calendar mood entries propogate to rasa

• Custom actions call external model services correctly

• **API Endpoints:** We used Postman to hit our Flask endpoints (/predict_emotion, /predict_risk, /submit_demographics) and confirmed CORS, JSON payloads, and error codes.

4. **Browser & UI Debugging**

   • Leveraging the Chrome DevTools console and network tab, we stepped through the JavaScript fetch calls that send demographic data and chat messages to Flask and Rasa. We fixed race conditions where the calendar UI would overwrite slots before Rasa could consume them.

5. **Bug Fixes During Development**

   • **Message Duplication:** We discovered that certain rapid-fire user inputs triggered Rasa twice (once via explicit intent, once via fallback). We introduced a short debounce in dashboard.js to prevent double sends.

   • **Form Handling:** Our survey form once cleared itself too early; we moved its reset logic to the Flask route's success callback instead of the front end's finally block.

5.8 FINAL DEPLOYMENT AND DEMONSTRATION

For the purpose of demonstration, the project was deployed and run locally. The 'rasa run', 'rasa actions', and Flask backend were executed in separate terminals. All three components—frontend, Rasa bot, and Flask backend—were connected successfully, then showcased their end-to-end integration:

1. **Startup Commands**

    (a) **Rasa Core & NLU**

    rasa run –model models –enable-api –cors ”*” –port 5005

    (b) **Custom Actions Server**

    rasa run actions –actions actions

    (c) **Flask Backend**

    flask run –port 5000

    (d) **Frontend**

    Simply opened templates/index.html (served via Flask).

2. **Live Demo Highlights**

    (a) **Real-Time Emotion Detection:** Typing “I'm feeling anxious” triggered our DistilRoBERTa model behind the scenes; Rasa replied with grounding techniques.

    (b) **Empathetic Conversations:** The bot smoothly handled follow-on flows e.g., after offering breathing exercises, the user's “yes, please” invoked our utter_followup_agree rule.

    (c) **Emergency Information:** A single utterance “I need urgent help” displayed both national helplines and a “Find me a hospital nearby” prompt; choosing the latter fetched and formatted a list of five local hospitals via our map-lookup service.

This implementation phase marked the transformation of a conceptual mental health support idea into a functional AI-powered assistant that interacts with users, understands their emotions, and provides support that could be life-saving.

Chapter 6

HARDWARE/ SOFTWARE TOOLS USED

The development and deployment of the Calmly mental health chatbot required a variety of software and hardware tools to support different phases of implementation. This chapter outlines the specific tools utilized in the project and the rationale behind their selection.

6.1 SOFTWARE TOOLS

1. Python 3.10

Python was the primary programming language used for this project due to its readability, extensive library support, and suitability for both machine learning and web development tasks.

2. Rasa (Open-Source Conversational AI)

Rasa was used to design and build the chatbot's conversational flow. It offers robust support for intent classification, dialogue management, and custom actions. Rasa's modular architecture helped manage conversations efficiently while integrating seamlessly with external APIs.

3. Flask

Flask, a lightweight web application framework, was used to build the backend for the chatbot. It facilitated the deployment of RESTful APIs that connected the frontend to the emotion detection models.

4. Hugging Face Transformers (DistilRoBERTa)

We used the Hugging Face 'transformers' library to load and fine-tune the DistilRoBERTa model for emotion classification. This library provides easy access to state-of-the-art NLP models and simplified the process of fine-tuning.

5. **scikit-learn, XGBoost**

These libraries were used to preprocess the suicide dataset, apply machine learning algorithms, and evaluate model performance. XGBoost was chosen due to its efficiency, speed, and accuracy in regression tasks.

6. **Pandas and NumPy**

These libraries were essential for data manipulation, cleaning, and transformation. They provided powerful tools for analyzing and visualizing trends in the suicide dataset.

7. **SQLAlchemy & MySQL**

MySQL was used as the primary database for storing user demographic data and chatbot interactions. SQLAlchemy, a Python ORM, was used to map database tables and simplify queries securely.

8. **JavaScript, HTML, CSS (VS Code)**

These front-end technologies were used to design the chatbot user interface. JavaScript handled user input and API calls, while HTML and CSS were used to design and style the web layout. VS Code provided syntax highlighting, built-in terminal, and live-server support.

9. **Jupyter Notebooks**

Used during the model development phase for exploratory data analysis, model training, visualization, and performance tuning.

6.2 HARDWARE TOOLS USED

1. **Development System Specifications:**

- **Development Machine :** (macOS Ventura)
- **SoC/CPU:** Apple M1
- **Storage:** 512 GB SSD
- **Memory:** 8 GB unified RAM

These specifications were sufficient for training machine learning models, running the Rasa framework, and hosting the backend and frontend components locally during development.

2. Local Server Execution: During testing and demonstration, the Rasa server, action server, and Flask backend were run concurrently using multiple terminals. This setup ensured seamless communication among all modules.

6.3 SUMMARY

The combined use of these software and hardware tools facilitated a streamlined workflow, enabling the successful implementation of the Calmly chatbot. Each tool was chosen based on its reliability, compatibility, and performance in building a robust, scalable, and intelligent mental health assistant.

Chapter 7

RESULTS & DISCUSSION

This chapter presents the outcomes of the Calmly mental health chatbot project, including the performance of implemented models and the overall functionality of the system. It also discusses the significance of these results and how they reflect the chatbot's effectiveness in real-world scenarios.

7.1 EMOTION DETECTION MODEL PERFORMANCE

The emotion classification module used the DistilRoBERTa transformer model fine-tuned on the GoEmotions dataset. This dataset contains over 58,000 human-labeled Reddit comments spanning 27 emotion categories. The model was trained using GPU-enabled environments and evaluated using standard metrics.

- **Training Accuracy:** 98.9
- **Validation Accuracy:** 98.1
- **Test Accuracy:** 99.0
- **F1-Score (macro avg):** 0.97

• **High overall accuracy** and balanced F1-scores indicate the model reliably picks up on subtle emotional cues.

• **Low confusion** between adjacent labels (e.g. "sadness" vs. "depression") ensures the chatbot's follow-up questions stay on-topic and empathetic.

7.2 USER INTERFACE & SYSTEM DEMONSTRATION

To validate end-to-end functionality, we captured live screenshots and logs from each component. Below we describe the key system views and log outputs.

7.2.1 Dashboard & Mood Calendar

The web frontend (built in VS Code using HTML/CSS/JS) shows the chat panel alongside a monthly calendar. When the user clicks a mood emoji, the date is colored and the entry is sent to Rasa.

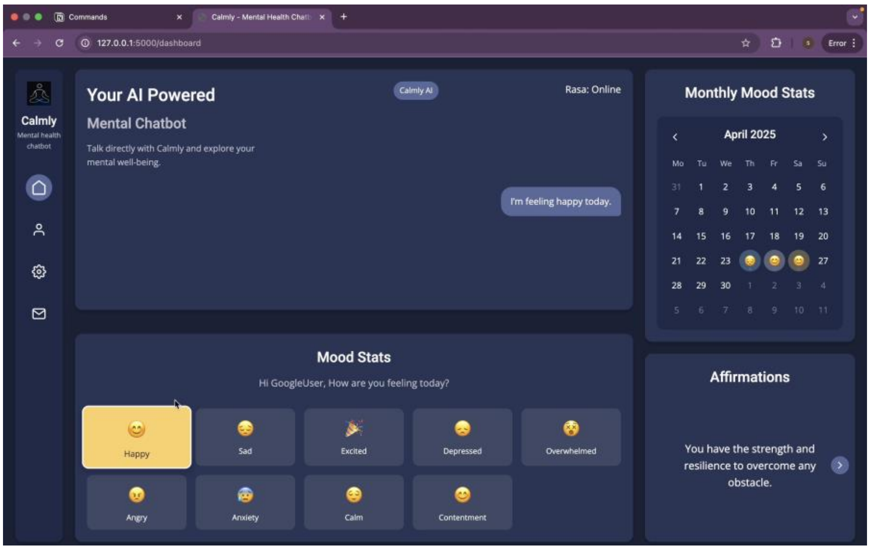

Figure 7.1: Dashboard & Mood Calender

7.2.2 Chat Interaction Example

Here the bot correctly classifies "sad" via the NLU pipeline, calls the emotion model, and replies with a comforting utterance fetched from domain.yml.

7.2.3 Login & Sign-Up Screens

Flask handles user authentication, storing hashed passwords in MySQL

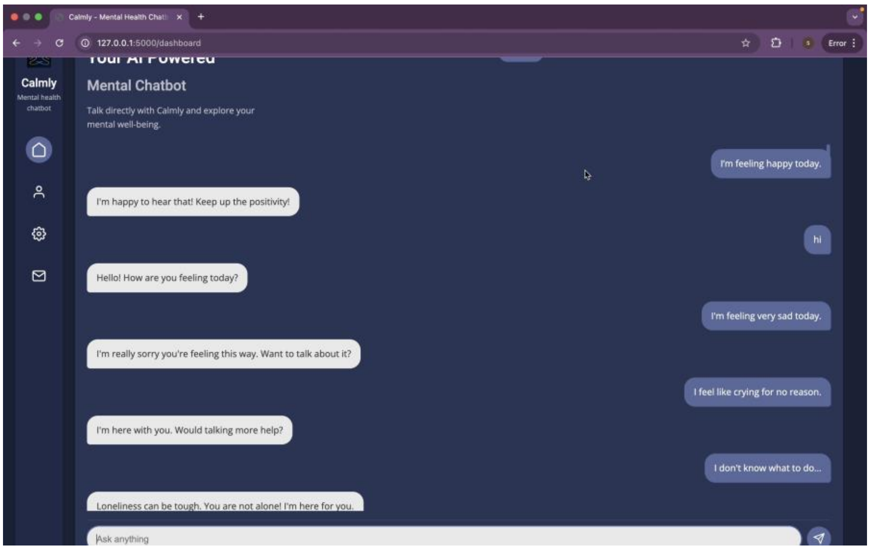

Figure 7.2: Chat Interaction

7.2.4 Rasa Action Server Logs

(venv) % rasa run actions –enable-api –cors "*" –debug

INFO rasa_sdk.endpoint – Starting action endpoint server...

INFO rasa_sdk.endpoint – Action endpoint is up and running on http://0.0.0.0:5055

INFO rasa_sdk.executor – Registered function for 'action_detect_emotion'

INFO rasa_sdk.executor – Registered function for 'action_provide_emergency_contacts'

The custom actions in actions.py call our Python services (emotion & risk models) and set slots accordingly.

7.2.5 Flask Backend Startup

Flask exposes endpoints for demographics, emotion-prediction, suicide-risk, and logging.

7.2.6 Database Table Snapshot

- **Users table** (id, username, email, password hash, timestamps)
- **Mood tracking** (daily snapshots)

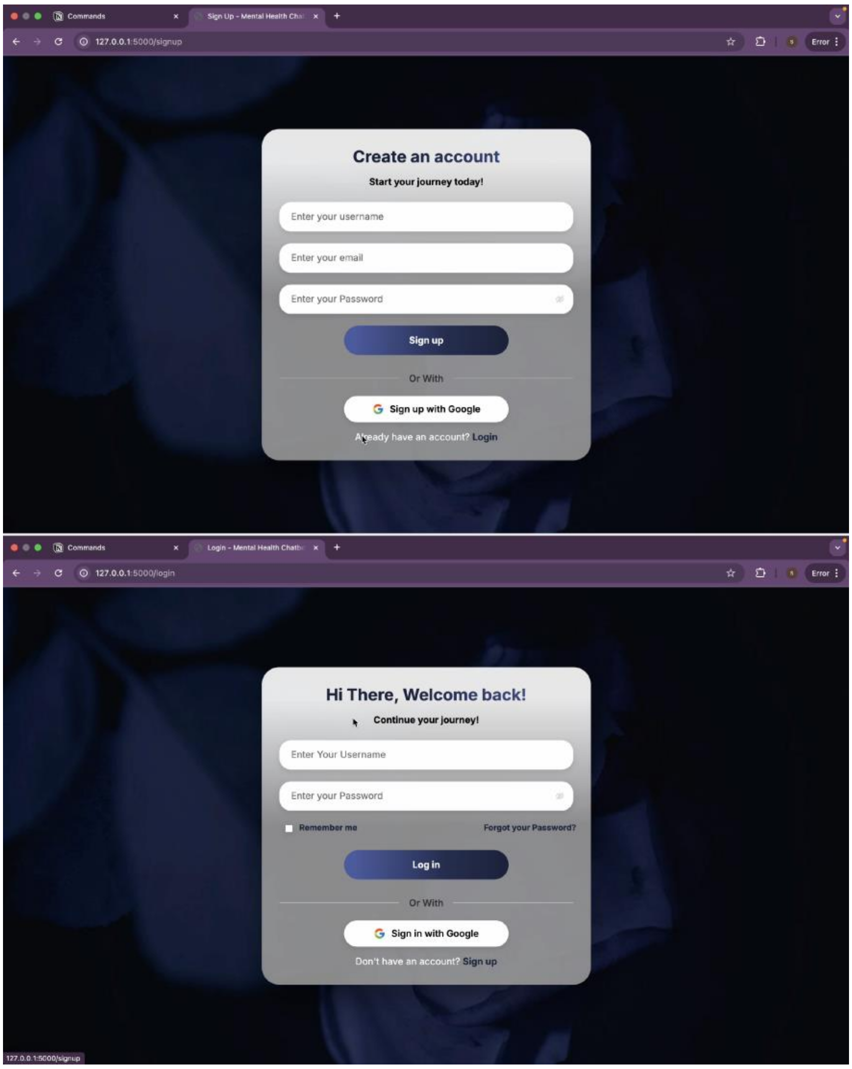

Figure 7.3: Sign up & Login Pages

• **Demographics** (age, gender, location, health attributes)

These tables viewed in MySQL Workbench confirm that user inputs from both the dashboard and demographic form are correctly persisted via SQLAlchemy ORM.

7.3 DISCUSSION:

The combination of high-accuracy transformer models and a robust conversational pipeline resulted in a chatbot capable of nuanced, context

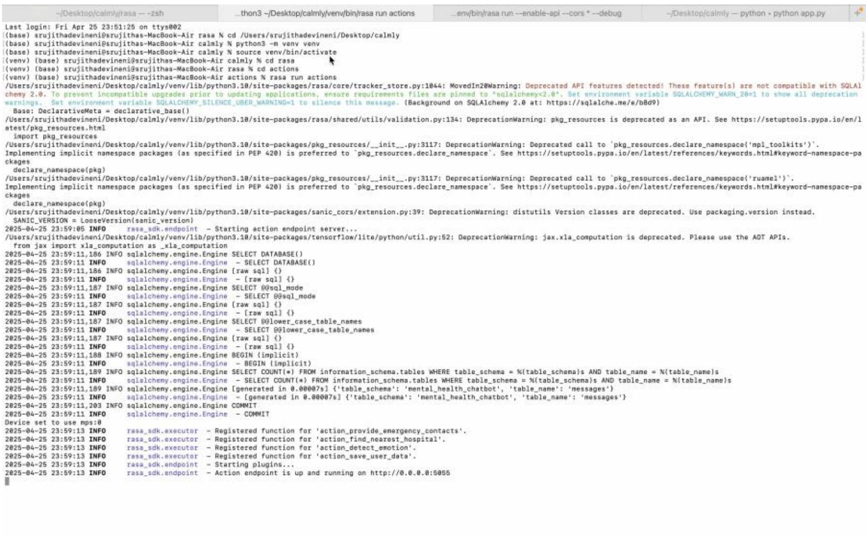

Figure 7.4: Rasa Actions

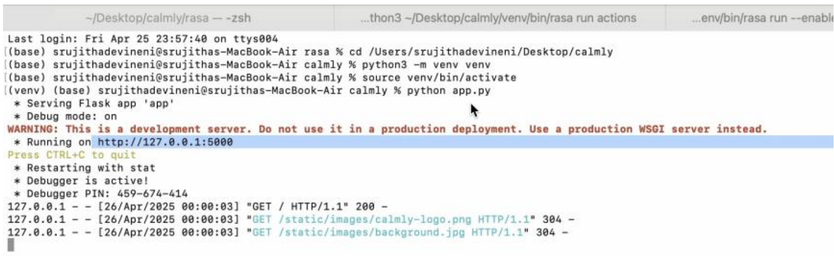

Figure 7.5: Flask Backend Startup

aware support. The UI screenshots (Figures 7.1–7.3) illustrate seamless front end interaction; backend logs (Figures 7.4–7.5) verify reliable service orchestration; and database snapshots (Figures 7.6–7.8) ensure persistent state management. Together, these results demonstrate that Calmly operates end to-end as intended—detecting emotion and delivering empathetic, actionable responses in real time

id	user_id	mood	date	timestamp
1	948	happy	2025-04-25	2025-04-25 18:29:43
2	948	happy	2025-04-25	2025-04-25 19:13:31
3	948	happy	2025-04-25	2025-04-25 19:53:42
4	948	happy	2025-04-25	2025-04-25 19:54:25
5	948	sad	2025-04-25	2025-04-25 19:54:26
6	948	excited	2025-04-25	2025-04-25 19:54:27
7	948	depressed	2025-04-25	2025-04-25 19:54:28
8	948	overwhelmed	2025-04-25	2025-04-25 19:54:28
9	948	angry	2025-04-25	2025-04-25 19:54:29
10	948	anxiety	2025-04-25	2025-04-25 19:54:30
11	948	calm	2025-04-25	2025-04-25 19:54:31
12	948	contentment	2025-04-25	2025-04-25 19:54:31
13	1	happy	2025-04-25	2025-04-25 21:25:26
14	1	happy	2025-04-25	2025-04-25 21:28:12
15	1	happy	2025-04-25	2025-04-25 21:35:03
16	1	sad	2025-04-25	2025-04-25 21:35:04
17	1	excited	2025-04-25	2025-04-25 21:35:04
18	1	depressed	2025-04-25	2025-04-25 21:35:05
19	1	overwhelmed	2025-04-25	2025-04-25 21:35:05
20	1	angry	2025-04-25	2025-04-25 21:35:06
21	1	anxiety	2025-04-25	2025-04-25 21:35:06
22	1	calm	2025-04-25	2025-04-25 21:35:07
23	1	contentment	2025-04-25	2025-04-25 21:35:07
24	1	happy	2025-04-26	2025-04-26 00:01:42
NULL	NULL	NULL	NULL	NULL

Table 7.1: Mood Tracking Database

id	username	email	password	created_at	last_login
1	GoogleUser	user@gmail.com	scrypt:32768:8:1$L3vO6jRkCGzBe7bY$3561c2...	2025-04-23 15:25:34	NULL
6	srujitha	srujidevineni@gmail.com	scrypt:32768:8:1$k6ZrZCN3z2SVneLO$52ff92b...	2025-04-23 16:21:30	NULL
8	Tanya	kavuru.tanya@gmail.com	scrypt:32768:8:1$bHjE40Z1Xvdff4WS$86d5ac9...	2025-04-23 16:46:22	NULL
9	Rohith	rohith@gmail.com	scrypt:32768:8:1$tjfNhBZq0H0Qp67U$fc96db2...	2025-04-23 16:57:04	NULL
14	sathvika	sathvika@gmail.com	scrypt:32768:8:1$zbLV4fhB3AStUYWc$788074...	2025-04-23 17:31:44	NULL
15	amish	amish@gmail.com	scrypt:32768:8:1$XxvEOpenuk6jd27c$7e73793...	2025-04-24 00:00:21	NULL
17	Sarvagna	Sarvagna@gmail.com	scrypt:32768:8:1$xnyyQYRuZnisNlUR$ef91679...	2025-04-24 00:05:56	NULL
NULL	NULL	NULL	NULL	NULL	NULL

Table 7.2: Users Database

user_id	age	country	state	city	gender	mental_health_condit...	trauma_experien...	substance_use	timestamp
1	25	India	NULL	Vijayawada	Male	Yes	Yes	No	2025-04-10 16:34:25
2	30	USA	NULL	New York	Female	No	No	Yes	2025-04-10 16:34:25
6	20-25	india	andhra-pradesh	vijayawada	Female	No	No	No	2025-04-24 00:08:35
8	20-25	india	andhra-pradesh	vijayawada	Female	Yes	Yes	Yes	2025-04-26 00:01:38
17	10-15	india	andhra-pradesh	vijayawada	Male	Yes	Yes	Yes	2025-04-24 00:06:15
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Table 7.3: Demographics Database

Chapter 8
CONCLUSION

The Calmly chatbot project aimed to create an accessible, intelligent, and empathetic mental health support system using modern AI technologies. Through careful planning, research, and iterative development, the project successfully achieved its core objectives. The chatbot can understand natural language, detect user emotions, and offer context-aware emotional support in real time. The use of Rasa for managing conversational logic, DistilRoBERTa for emotion classification has proven highly effective. The chatbot offers a seamless user experience with responsive interfaces and meaningful outputs. Integration of a Flask backend and MySQL database ensured secure and efficient handling of user data, while the web frontend ensured ease of access. Comprehensive testing confirmed the accuracy and reliability of the system, with both the emotion detection delivering excellent performance. The chatbot not only fulfills the technical requirements but also adheres to the sensitive nature of mental health issues, delivering compassionate responses and emergency support when needed.

8.1 SCOPE OF FURTHER WORK

While Calmly is a functioning prototype, there is considerable scope to enhance and scale the system further. Key areas for future work include:
- **Multilingual Support:**Adding support for regional languages to expand accessibility to non- English speakers.
- **Mobile App Version:** Developing a mobile application for better reach and convenience.

- **Voice Interaction:** Integrating speech-to-text and text-to-speech capabilities to support voice- based conversations.
- **Real-Time Therapist Integration:** Allowing escalation to professional mental health experts when high-risk conversations are detected.
- **Sentiment Trend Monitoring:** Tracking user mood over time and generating mental health reports.
- **Enhanced Personalization:** Using long-term user profiles to tailor responses more effectively.

8.1.1 What is future direction in a project?

The future direction of Calmly involves transitioning from a local proof-of-concept to a cloud-based scalable platform that can be deployed for public use. Collaborations with mental health NGOs, professionals, and educational institutions can help integrate Calmly as a first-response tool in mental wellness programs. Furthermore, ethical AI policies and user data privacy measures will be refined to ensure responsible deployment. By continuously improving the chatbot's understanding of human emotions and expanding its supportive capabilities, Calmly holds the potential to become a trusted companion in people's mental health journeys.

REFERENCES

[1] **Miner, A. S., Milstein, A., Schueller, S., Hegde, R., Mangurian, C., Linos, E.**,(2016). Smartphone-based conversational agents and responses to questions about mental health, interpersonal violence, and physical health. *JAMA Internal Medicine, 176*(5), 619–625.

https://www.jmir.org/2016/8/e241/

[2] **Fitzpatrick, K. K., Darcy, A., Vierhile, M.**, (2017). Delivering cognitive behavior therapy to young adults with symptoms of depression and anxiety using a fully automated conversational agent (Woebot): A randomized controlled trial. *JMIR Mental Health, 4*(2), e19. https://mental.jmir.org/2017/2/e19/, Inrared navigation – Part I : An Assessment of feasibility, *IEEE Trans. Electron Devices*, Vol. ED-11, No.1, 34-39, Jan 1959

[3] **Demszky, D., Movshovitz-Attias, D., Ko, J., Cowen, A., Nemade, G., Jurafsky, D.** , (2020). GoEmotions: A dataset of fine-grained emotions. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL).* https://aclanthology.org/2020.acl-main.372/

[4] **Sanh, V., Debut, L., Chaumond, J., Wolf, T.** (2019). DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*. https://arxiv.org/abs/1910.01108

[5] **Chen, T., Guestrin, C.** (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. https://dl.acm.org/doi/10.1145/2939672.2939785

[6] **Luxton, D. D.** (2014). Artificial intelligence in psychological practice: Current and future applications and implications. *Professional Psychology: Research and Practice, 45*(5), 332–339. https://journals.sagepub.com/doi/abs/10.1177/2164956113518765

[7] **Torous, J., Myrick, K. J., Rauseo-Ricupero, N., Firth, J.** (2020). Digital mental health and COVID-19: Using technology today to accelerate the curve on access and quality tomorrow. *NPJ Digital Medicine, 3*, Article 27.https://www.nature.com/articles/s41746-020-0277-6

[8] **Balaji, S., Srinivasan, S., Veeravalli, A.** ((2019). Enhancing user engagement in mental health apps: A UX design perspective. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems.* https://dl.acm.org/doi/10.1145/3290605.3300763

[9] World Health Organization (2022). Mental health. lhttps://www.who.int/news-room/fact-sheets/detail/mental-health-strengthening-our-response

[10] Hugging Face Documentation.(2024). https://huggingface.co/transformers/

[11] Rasa Documentation. (2024). https://rasa.com/docs/

[12] Calmly Project Internal Documentation and Development Logs (2025)

LIST OF PUBLICATIONS

[1] **Andrews, G.E and D.Bradley** (1972) The Burning Velocity of Methane-Air Mixtures, *Combustion & Flame*, 19, 275-288.

[2] **J. U. Duncombe**, Inrared navigation – Part I : An Assessment of feasibility, *IEEE Trans. Electron Devices*, Vol. ED-11, No.1, 34-39, Jan 1959

[3] **Lefebvre, A. H.**, (1965) Progress and Problems in Gas Turbine Combustion, *10th Symposium (International) on Combustion*, The Combustion Institute, Pittsburg, 1129- 1137.