###### Department of Computer Science

University of Gujrat

***Chat Bot for Mental Health***

**Submitted By:**

|  |  |
| --- | --- |
| Anss Abrar | 20021519-053 |
| Ammara Khalid | 20021519-057 |
| Hizar Sajjad | 20021519-068 |

**Supervised By:**

**Dr. Naveed Anwar Butt**

****

STATEMENT OF SUBMISSION

This is certify that \_\_\_\_\_\_\_\_\_\_\_\_\_ Roll No. \_\_\_\_\_\_\_\_\_\_\_\_ and \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Roll No. \_\_\_\_\_\_\_\_\_\_\_\_\_\_ has successfully completed the final year project named as­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­\_\_\_\_\_\_\_\_\_\_\_\_\_ at the Department of Information Technology, University of Gujrat, to fulfill the requirement of the degree of \_\_\_\_\_\_\_ **in Computer Science.**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Project Supervisor Project Coordination Office

Faculty of C&IT -UOG

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Head of the Department

Acknowledgement

We truly acknowledge the cooperation and help make by \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, Chairman, Department of Software Engineering, University of Gujrat. He has been a constant source of guidance throughout the course of this project. We would also like to thank \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ for his help and guidance throughout this project. We are also thankful to our friends and families whose silent support led us to complete our project.

Date:

**Abstract**

Our project **"Chatbot for Mental Health"** system addresses the critical need for accessible and immediate mental health support in the face of escalating mental health challenges worldwide. With issues like depression, anxiety, overthinking, and suicidal thoughts affecting millions, this project aims to provide a versatile and user-friendly solution. Grounded in the realities of limited access to professional mental health services and persistent stigma, the chatbot offers individuals a confidential and empathetic space for discussing their concerns while providing tailored information, resources, and guidance. Key modules include the design and development of the chatbot, integration into web or mobile applications, and a robust user feedback evaluation mechanism. Utilizing the **Llama chat pre-trained model**, the chatbot engages users in meaningful conversations, offering support for specific mental health concerns. Prospects of enhanced access, reduced stigma, and scalability make this system promising for individuals grappling with these challenges. However, potential privacy concerns and limitations in handling complex issues must be considered. Our project aims to leverage technology as a lifeline, promoting well-being, and reducing barriers to vital mental health support. The **"Chatbot for Mental Health"** project emerges as a beacon of hope for those navigating the complexities of mental health challenges.

Table of Contents

[Chapter 1: Project Feasibility Report 8](#_Toc510009926)

[1.1. Introduction 9](#_Toc510009928)

[1.2. Project/Product Feasibility Report 9](#_Toc510009929)

[1.2.1. Technical Feasibility 9](#_Toc510009930)

[1.2.2. Operational Feasibility 9](#_Toc510009931)

[1.2.3. Economic Feasibility 9](#_Toc510009932)

[1.2.4. Schedule Feasibility 10](#_Toc510009933)

[1.2.5. Specification Feasibility 10](#_Toc510009934)

[1.2.6. Information Feasibility 10](#_Toc510009935)

[1.2.7. Motivational Feasibility 10](#_Toc510009936)

[1.2.8. Legal & Ethical Feasibility 10](#_Toc510009937)

[1.3. Project/Product Scope 10](#_Toc510009938)

[1.4. Project/Product Costing 10](#_Toc510009939)

[1.4.1. Project Cost Estimation By Function Point Analysis 11](#_Toc510009940)

[1.4.2. Project Cost Estimation by using COCOMO’81 (Constructive Cost Model) 13](#_Toc510009941)

[1.4.3. Activity Based Costing 14](#_Toc510009942)

[1.5. Task Dependency Table 14](#_Toc510009943)

[1.6. CPM - Critical Path Method 15](#_Toc510009944)

[1.7. Gantt chart 18](#_Toc510009945)

[1.8. Introduction to Team member and their skill set 18](#_Toc510009946)

[1.9. Task and Member Assignment Table 19](#_Toc510009947)

[1.10. Tools and Technology with reasoning 22](#_Toc510009948)

[1.11. Vision Document 22](#_Toc510009949)

[1.12. Risk List 23](#_Toc510009950)

[1.13. Product Features/ Product Decomposition 23](#_Toc510009951)

[Chapter 2: Software Requirement Specification (For Object Oriented Approach) 24](#_Toc510009952)

[2.1 Introduction: 25](#_Toc510009953)

[2.1.1 Systems Specifications 26](#_Toc510009954)

[3.1.2. Identifying External Entities 27](#_Toc510009955)

[2.1.3. Context Level Data Flow Diagram: 27](#_Toc510009956)

[2.1.4. Capture "shall" Statements: 27](#_Toc510009957)

[2.1.5. Allocate Requirements: 27](#_Toc510009958)

[2.1.6. Prioritize Requirements: 27](#_Toc510009959)

[2.1.7. Requirements Trace-ability Matrix: 28](#_Toc510009960)

[2.2. Example: 28](#_Toc510009961)

[2.2.1. Introduction 28](#_Toc510009962)

[2.2.2. Existing System 28](#_Toc510009963)

[2.2.3. Scope of the System 29](#_Toc510009964)

[2.2.4. Summary of Requirements:(Initial Requirements) 30](#_Toc510009965)

[2.2.5. Identifying External Entities: 31](#_Toc510009966)

[2.2.6. Capture "shall" Statements: 32](#_Toc510009967)

[3.2.7. Allocate Requirements: 32](#_Toc510009968)

[2.2.8. Priorities Requirements: 34](#_Toc510009969)

[2.2.9. Requirements Traceability Matrix: 37](#_Toc510009970)

[2.2.10. High Level Usecase Diagram: 39](#_Toc510009971)

[2.2.11. Analysis Level Usecase Diagram: 41](#_Toc510009972)

[2.2.12. Usecase Description 41](#_Toc510009973)

[Chapter 3: Design Document (For Object Oriented Approach) 43](#_Toc510009974)

[3.1. Introduction: 44](#_Toc510009976)

[3.2. Domain Model 44](#_Toc510009977)

[3.3. System Sequence Diagram 45](#_Toc510009978)

[3.4. Sequence Diagram 45](#_Toc510009979)

[3.4.1. Defining a Sequence diagram 46](#_Toc510009980)

[3.4.2. Basic Sequence Diagram Symbols and Notations 46](#_Toc510009981)

[3.4.3. Example 49](#_Toc510009982)

[3.4.4. Distributing Control Flow in Sequence Diagrams 49](#_Toc510009983)

[3.5. Collaboration Diagram 52](#_Toc510009984)

[3.5.1. Contents of Collaboration Diagrams 53](#_Toc510009985)

[3.5.2. Constructs of Collaboration Diagram: 54](#_Toc510009986)

[3.6. Operation Contracts 55](#_Toc510009987)

[3.7. Design Class Diagram 55](#_Toc510009988)

[3.7.1. Create Initial Design Classes 56](#_Toc510009989)

[3.7.2. Designing Boundary Classes 56](#_Toc510009990)

[3.7.3. Designing Entity Classes 56](#_Toc510009991)

[3.7.4. Designing Control Classes 56](#_Toc510009992)

[3.7.5. Identify Persistent Classes 57](#_Toc510009993)

[3.7.6. Define Class Visibility 58](#_Toc510009994)

[3.7.7. Design Class Relationships 62](#_Toc510009995)

[3.8. State chart diagram 66](#_Toc510009996)

[3.9. Data Model 67](#_Toc510009997)

**List of Figures**

Figure 1: Network Diagram………………………………………………………………………………………19

Figure 2: Critical Path…………………………………………………………………………………………….…19

Figure 3: Gantt Chart …………………………………………………………………………………….…………20

Figure 4: Organizational Chart…………………………………………………………

Figure 5: Data flow diagram………………………………………………………………

Figure 6: Use Case Diagram…………………………………………………………….

Figure 7: Level Use Case Diagram………………………………………………………

Figure 8: Sequence Diagram…………………………………………………………….

Figure 9: Collaboration Diagram……………………………………………………….

**List of Tables**

Table 1: unadjusted function points …………………………………………………………………………12

Table 2: Value adjustment factor …………………………………………………………………………..…14

Table 3: Modes of COCOMO Model………………………………………………………………………..…15

Table 4: Task dependency table…………………………………………………………………………………17

Table 5: Tasks…………………………………………………………………………………………………….………18

Table 6: Sequence of activities…………………………………………………………..………………………18

Table 7: Activity completion time………………………………………………………………………………19

Table 8: Team members, skill set and tasks………………………………………………….……………20

Table 9: Tasks Duration and Dependencies…………………………………………………..……………21

Table 10: Task Assignment table…………………………………………………………………………..……21

Table 11: Vision document…………………………………………………………………………………………23

# Chapter 1: Project Feasibility Report

## Introduction

The "Project Feasibility Report" chapter of our **Chatbot-for-mental-health** project document will provide a detailed analysis of the feasibility of the proposed system. This report will include a thorough evaluation of the technical, operational, economic and schedule feasibility of the project. It will determine whether the project is viable from a financial, technical and operational perspective.

Chatbot for Mental Health is an innovative AI-powered companion dedicated to fostering emotional well-being. Through the seamless integration of advanced technologies, including **emotion detection**, Chatbot for Mental Health provides personalized support for relaxation techniques and stress relief. Its empathetic approach, **available 24/7**, aims to address the challenges associated with expressing deep emotions, creating a supportive and accessible platform for individuals navigating mental health concerns.

## 1.2. Project/Product Feasibility Report

Our **"Chatbot for Mental Health"** project demonstrates strong feasibility on various fronts. From a technical standpoint, the use of the Llama chat pre-trained model, Flutter for mobile app development, Python for chatbot functionality, and Firebase for data storage ensures a robust and scalable solution. The economic feasibility is favorable with cost-effective tools and potential revenue streams. Operationally, the project aligns with student capabilities and offers compatibility with major platforms. Legal and ethical considerations are addressed through privacy measures and adherence to mental health guidelines. The well-defined schedule feasibility ensures a systematic project timeline. In conclusion, the comprehensive assessment indicates that the "Chatbot for Mental Health" project is feasible, promising, and aligned with its objectives for providing accessible mental health support. There are many types of feasibilities:

* Technical
* Operational
* Economic
* Schedule
* Specification
* Information
* Motivational
* Legal and Ethical

### 1.2.1. Technical Feasibility

The **"Chatbot for Mental Health"** project demonstrates strong technical feasibility through its thoughtful selection of tools and technologies, aligning with the complexities of developing a mental health support system. The utilization of the Llama chat pre-trained model, **Flutter** for cross-platform mobile app development, **Python** for chatbot functionality, and **Firebase** for data storage ensures a robust and scalable solution. The choice of Python, given its versatility and compatibility with the **Llama model**, enhances the chatbot's natural language processing capabilities. Firebase provides a cloud-based and user-friendly database solution, eliminating the need for extensive server maintenance. The project's three-tier architecture, encompassing presentation, application logic, and data storage layers, reflects modularity, scalability, and maintainability. The consideration of privacy constraints and ethical considerations underscores the project's commitment to user confidentiality. Overall, the technical feasibility of the "Chatbot for Mental Health" project is evident in its thoughtful integration of cutting-edge technologies to address the unique challenges of mental health support.

### 1.2.2. Operational Feasibility

Our project **"Chatbot for Mental Health"** exhibits strong operational feasibility, aligning well with the capabilities and resources available to the student development team. The chosen tools and technologies, including Flutter for cross-platform mobile app development, Python for chatbot functionality, and Firebase for data storage, are well-suited for a project of this scale. The compatibility with major operating systems ensures widespread accessibility. Collaboration is facilitated through version control using Git and an integrated development environment like Visual Studio Code. The operational feasibility is further enhanced by the project's adaptability to evolving needs in mental health support, providing a user-friendly and efficient system seamlessly integrated into both web and mobile platforms. The commitment to continuous improvement and monitoring aligns with operational best practices, ensuring the system's reliability and relevance over time. Overall, the operational feasibility of the project is evident in its strategic alignment with student capabilities and its responsiveness to the dynamic nature of mental health support requirements.

### 1.2.3. Economic Feasibility

Economically our project has many benefits. Economic feasibility has two types

* **Cost Estimates:** The acquisition cost for the development of Chattbot For Mental Health is estimated to be within a budget of 80 thousand.

**Maintenance and Operation Cost:** There will be ongoing costs for server

hosting, maintenance, and regular updates, estimated at 10 thousand per year.

* **Benefit Estimates:** Benefit estimates enclose tangible benefits and intangible benefits.
* **Tangible Benefits**: Most of the services are free and at start all services will be available freely for 3 month and then cost will be charged for some services.
* **Intangible Benefits**: We try to provide best services to the users of our system and provide correct and useful.

### 1.2.4. Schedule Feasibility

Effective project scheduling and planning play a crucial role in ensuring the timely completion of our project, "Chatbot for Mental Health." To address this, we employ **Microsoft Project** as a tool for meticulous task management before deadlines. Recognizing the inherent challenges, we have adopted a systematic approach by breaking down the project into tasks with well-defined dependencies and durations. This strategy ensures each task is clearly articulated, aligned with project goals, and the team comprehends their order and interdependence. Our utilization of the **Gantt chart** further enhances this systematic planning. By estimating task durations, we have constructed a realistic project schedule, setting clear expectations for completion timelines. The creation of a work breakdown structure aids in identifying dependencies in the proper order, aiming to conclude the project a week before the evaluation date. The remaining time will be dedicated to thorough testing, ensuring the quality of our deliverables. Our schedule is designed with flexibility to accommodate unforeseen challenges while staying focused on achieving milestones.

### 1.2.5. Specification Feasibility

Chatbot for Mental Health strives to deliver emotional support and mindfulness by leveraging technical capabilities such as emotion detection. Through meticulous attention to clear priorities and constraints, our requirement gathering process will be precise and grounded in realism. Our overarching objective is to create a user-friendly environment where individuals can interact comfortably, receiving the emotional support they need. We are committed to utilizing available resources and expertise to ensure the effective implementation of our plans.

### 1.2.6. Information Feasibility

### Chatbot For Mental Health prioritizes the acquisition of accurate information to optimize its functionality. Leveraging data on emotions, language, and user preferences, the system aims to deliver improved and personalized responses. Continuous adaptation and updates will be integral to ensuring the bot remains accurate and beneficial as technology advances. Vigilant monitoring of the data landscape underscores the system's commitment to providing ongoing assistance to individuals seeking support for their emotional well-being.

### 1.2.7. Motivational Feasibility

Our team is deeply committed to the development of **Chatbot For Mental Health**, an innovative system designed to address shortcomings observed in existing applications. With unique features focused on emotional support, such as advanced sentiment understanding and interactive communication, we are enthusiastic about the project's potential impact. Our collective belief in the feasibility of this initiative stems from its practicality and significance in aiding individuals with their emotional well-being. The motivation behind Chatbot For Mental Health lies in its potential to make a positive difference in people's lives by offering tailored support for their emotional challenges.

### 1.2.8. Legal & Ethical Feasibility

## Chatbot for Mental Health is in complete adherence to legal and ethical standards. We possess the requisite rights to utilize tools for each module within the system. The entire system is designed to comply with all existing rules and regulations, mitigating any potential risks or liabilities. We are committed to undertaking both legal and financial responsibility for any necessary services, thereby fostering user trust and demonstrating a responsible approach across all modules.

## 1.3. Project/Product Scope

## The project scope for Chatbot for Mental Health revolves around the creation of a responsive and emotionally intelligent chatbot. It encompasses pivotal elements, including emotion detection, emotion-driven dialogue, privacy management, and user registration. This chatbot is designed to provide users with real-time, human-like responses, delivering mindfulness exercises and techniques. The user registration module empowers individuals to customize their interactions, ensuring a personalized and private experience. While the initial phase of the project concentrates on these fundamental functionalities, it serves as a cornerstone for potential expansion, responding to the escalating demand for easily accessible emotional support facilitated by advanced AI technology. This project is unwavering in its commitment to deliver a comprehensive, emotionally supportive chatbot system tailored to the needs of individuals seeking effective stress management and emotional well-being.

## 1.4. Project Costing

The cost depends on how much computation power we use to train our model. Essentially, we will subscribe to the Google Colab plan, which costs $9.99 per month. If we require additional computation power, we will consider switching our subscription plan.

### 1.4.1. Project Cost Estimation By Function Point Analysis

Table 1: Unadjusted Functional Point

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type of Component** | **Complexity of Components** |  |  |  |
|  | **Low** | **Average** | **High** | **Total** |
| External Input | 0\*30=0 | 14\*4=56 | 0\*6=0 | 56 |
| External output | 0\*9=0 | 3\* 5=15 | 0\*7=0 | 15 |
| External Inquiries | 0\*9=0 | 8\*4=32 | 0\*6=0 | 32 |
| Internal Logical Files | 0\*5=0 | 10\*10=100 | 0\*15=0 | 100 |
| External interface files | 0\*1=0 | 2\*7=14 | 0\*10=0 | 14 |
|  |  | **Total Function point** |  | 217 |

Information domain values are defined in the following manner:

**Number of user inputs:**

**Authentication for user:**

Authentication for user involves several crucial user inputs as outlined in the proposal:

* Email: Required for user login and registration, serving as a unique identifier.
* Password: Essential for user login and registration to ensure secure access.
* Username: Specifically for user registration, creating a personalized account.
* Request to Reset Password: A user-initiated request, contributing to password recovery.

**Chat:**

Chatbot Interaction Features:

* Verbal expression of emotions
* Search commands for reviewing interaction history
* User input for ending and pausing conversations

**User Feedback Evaluation:**

* Users engage positively with empathetic responses, indicating the effectiveness of the chatbot in providing emotional support and understanding.
* Users actively participate by providing input for adding or removing favorites, showcasing an interactive and user-centric feature in line with the project's goal of tailored assistance and guidance.

**Preferences:**

Users have the autonomy to customize their interactions with the chatbot, setting preferences that cater to their unique needs and preferences. This feature allows individuals to tailor the chatbot's responses and engagement style, creating a more personalized and effective user experience.

**Number of user outputs**

* + Tailored empathetic conversations catering to individual user needs.
  + Mindfulness exercises and tips for relaxation to promote mental well-being.
  + Written transcriptions of user's verbal interactions, ensuring a comprehensive record of the conversation for user reflection and support analysis."

**Number of user inquiries:**

**Authentication for User:**

* + Incorrect email during user login
  + Incorrect password during user login

**Chat:**

* + Misinterpretation of user's expressed emotions
  + User dissatisfaction with chatbot responses
  + Issues with the tone or clarity of the synthesized speech

**Feedback:**

* Reporting any technical issues or glitches

**General Inquiries:**

* Inquiring about the capabilities of the Chatbot for Mental Health
* Seeking information on available features

**Number of internal files:**

* File for storing Registered users
* File for storing login users
* File for storing input messages
* File for storing detected emotions
* File for storing the history of user interactions
* File for strong context of user chat
* File for storing user feedback

**Number of external interface files:**

* Speech synthesis API
* Speech recognition API

Once these data have been collected, a complexity value is associated with each count. Organizations that use function point methods develop criteria for determining whether a particular entry is simple, average, or complex. Nonetheless, the determination of complexity is somewhat subjective.

To compute function points (FP), the following relationship is used:

FP est. = Count Total \* [ 0.65 + 0.01 \* (Fi)]

Where count total is the sum of all FP entries obtained from above figure and (Fi) is value adjustment factor (VAF) is based on 14 general system characteristics (GSC's) that rate the general functionality of the application being counted. Each characteristic has associated descriptions that help determine the degrees of influence of the characteristics. The degrees of influence range on a scale of zero to five, from no influence to strong influence.

Table 2: Value Adjustment Factor

|  |  |  |
| --- | --- | --- |
| **Sr.no** | **Value adjustment factor (VAF) Values** | **Values** |
| 1 | Data communication | 4 |
| 2 | Distributed data processing | 4 |
| 3 | Performance | 3 |
| 4 | Heavily used Configuration | 3 |
| 5 | Transaction rate | 4 |
| 6 | On-line data entry | 3 |
| 7 | End-user efficiency | 3 |
| 8 | On-line update | 3 |
| 9 | Complex processing | 4 |
| 10 | Reusability | 4 |
| 11 | Installation ease | 4 |
| 12 | Operational ease | 2 |
| 13 | Multiple sites | 3 |
| 14 | Facilitate change | 4 |
|  | Total | 48 |

**Total Unadjusted Function Points (UFP**) = 217

**Complexity Adjustment Factor (CAF)** = 0.65 + (0.01 \*48) = 1.13

**Total Adjusted Function Points (FP)** = UFP \* CAF = 217\*1.13= 245.21

**Language Factor (LF) for:**

**JavaScript =47 (avg)**

**Python= 22 (avg)**

**SQL = 21 (avg)**

**Source Lines of Code (SLOC) Using JavaScript**= FP \* LF = 245.21\* 47= 11,524.87

**Source Lines of Code (SLOC) Using Python**= FP \* LF = 245.21\* 22= 5,394.62

**Source Lines of Code (SLOC) Using SQL**= FP \* LF = 245.21\* 21= 5149.41

**Total SLOC** =11,524.87+ 5,394.62+ 5149.41=22,068.9

For our Project

Average Productivity = 3x6= 18 FP/PM (function points per person-month)

Labor Rate = 40,000 RS/Month

**Total estimated Effort** = FP est. / productivity

= 245.21/18 = 13.622 pm

**Cost/FP = Labor Rate/productivity**

= 40,000/18 = 2222.23 Rs/FP

= **2222 Rs/FP Approx.**

**Total Project Cost = FP est. \* (Cost/FP)**

= 245.21\* (2222.23)

= **544,913 Rs**

### 1.4.2. Project Cost Estimation by using COCOMO’81 (Constructive Cost Model)

Boehm's COCOMO model stands as a widely adopted commercial model for calculating software development effort and cost based on the estimated Lines of Code (LOC). To initiate cost estimation using the COCOMO model, the first step involves selecting the appropriate mode from the three available options:

1. Organic

2. Semi-Detached

3. Embedded

Upon careful consideration, our software project exhibits specifications and characteristics closely aligned with the organic mode of the COCOMO model. Therefore, for the purpose of cost evaluation, we will utilize the parameters associated with the COCOMO model, specifically tailored to the organic mode, as outlined in the table below.

**Modes of COCOMO Model**

Table 3: Modes of COCOMO Model

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameters** | **Organic** | **Semi-Detached** | **Embed** |
| Size | 2-50 KLOC | 50-300 KLOC | 300KLOC or above |
| Team | Small | Medium | Large |
| Developer Experience | Experienced Developers | Average | Little Experience |
| Environment | Familiar | Less familiar | Changed |
| Innovation | Little | Medium | Major Innovation |
| Deadline | Flexible | Medium | Tight deadline |

Basic COCOMO

Type Effort Schedule

Organic PM= 2.4 (KLOC)1.05 TD= 2.5(PM)0.38

Semi-Detached PM= 3.0 (KLOC)1.12 TD= 2.5(PM)0.35

Embedded PM= 2.4 (KLOC)1.20 TD= 2.5(PM)0.32

PM= person-month (effort)

KLOC= lines of code, in thousands

TD= number of months estimated for software development (duration)

KLOC= 22,068/1000= 22.068k

Our project falls under **Organic class:**

Now we will put values in the given formulas:

**Calculate Effort:**

PM = 2.4 \* (KLOC) ^ 1.05

PM = 2.4 \* ( 22.068k) ^ 1.05

PM = 2.4 \* 25. 7603

**PM = 61.8247**

**Calculate Schedule:**

TD = 2.5\* ( **61.8247** ) ^ 0.38

TD = 2.5 \* 4.7933

**TD = 11.9834 Number of months**

**Calculate People Required:**

People Required = Effort / Duration

People Required =  **61.8247** / **11.9834**

**People Required = 5.1591**

**Calculate Productivity:**

Productivity = FP/Effort

Productivity = 245.21/ **61.8247**

**Productivity = 3.9662**

### 1.4.3. Activity Based Costing

Basic Cost Drivers:

Table 4: Cost Drivers

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Cost Drivers** | **Very Low** | **Low** | **Nominal** | **High** | **Very High** |
| **Product Attributes** |  |  |  |  |  |
| Required Software Reliability | 0.75 | 0.88 | 1.00 | 1.15 | 1.40 |
| Size of Application Database |  | 0.94 | 1.00 | 1.08 | 1.16 |
| Complexity of the product | 0.70 | 0.85 | 1.00 | 1.15 | 1.30 |
| **Hardware Attributes** |  |  |  |  |  |
| Runtime Performance Constraints |  |  | 1.00 | 1.11 | 1.30 |
| Memory Constraints |  |  | 1.00 | 1.06 | 1.21 |
| Volatility of the Virtual Machine environment |  | 0.87 | 1.00 | 1.07 | 1.15 |
| Required Turnabout Time |  | 0.94 | 1.00 | 1.07 | 1.15 |
| **Personnel Attributes** |  |  |  |  |  |
| Analyst Capability | 1.46 | 1.19 | 1.00 | 0.86 | 0.71 |
| Application experience | 1.29 | 1.13 | 1.00 | 0.91 | 0.82 |
| Software engineer Capability | 1.42 | 1.17 | 1.00 | 0.86 | 0.70 |
| Virtual Machine experience | 1.21 | 1.10 | 1.00 | 0.90 |  |
| Programming language Experience | 1.14 | 1.07 | 1.00 | 0.95 |  |
| **Project Attributes** |  |  |  |  |  |
| Application Of Software Engineering Methods | 1.24 | 1.10 | 1.00 | 0.91 | 0.82 |
| Use Of Software Tools | 1.24 | 1.10 | 1.00 | 0.91 | 0.83 |
| Requirement Development Schedule | 1.23 | 1.08 | 1.00 | 1.04 | 1.10 |

COCOMO II

PM= 2.4(KLOC)^1.05 \* EAF

Where:

KLOC = 22.068k

Now ,

EAF= 1.115\*1.00\*1.00\*1.00\*1.00\*1.00\*1.00\*1.00\*1.00\*1.00\*1.00\*1.00\*1.11=1.2765

**Calculate Effort:**

PM = 2.4 \* (22.068k) ^ 1.05\*EAF

PM = 2.4 \* 25.7603\*1.2765

**PM = 78.9192**

**Calculate Schedule:**

TD = 2.5\* ( **78.9192**) ^ 0.38

TD = 2.5\*5. 2593

TD =13.1483

## 1.5. Task Dependency Table

Table 4: Task dependency table

|  |  |  |
| --- | --- | --- |
| **Task** | **Task Name** | **Dependency** |
| Task 1 | Research Analysis | None |
| Task 2 | Planning | Task1 |
| Task 3 | Requirement Gathering | Task2 |
| Task 4 | Requirement Specification | Task2, Task3 |
| Task 5 | Architectural Design | Task1, Task4 |
| Task 6 | Initial Prototype | Task4, Task5 |
| Task 7 | Test Prototype | Task6 |
| Task 8 | Design Database | Task4 |
| Task 9 | Classification | Task8 |
| Task 10 | Server-Side Implementation | Task8, Task9 |
| Task 11 | Module Development | Task7, Task5, Task10 |
| Task 12 | Unit Testing | Task11, Task8 |
| Task 13 | Integrating Module | Task11, Task12 |
| Task 14 | System Testing | Task4, Task12, Task13 |
| Task 15 | Deployment | Task14 |

## 1.6. CPM - Critical Path Method

**1. Specify the Individual Activities**

Table 5: Tasks

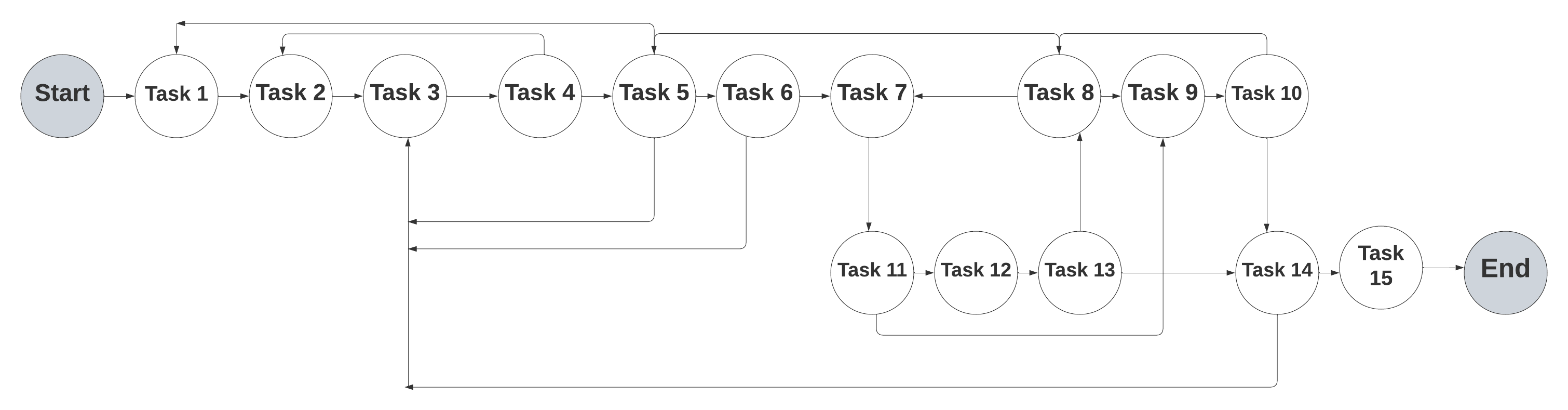
|  |  |
| --- | --- |
| **Activity** | **Activity Name** |
| Task 1 | Research Analysis |
| Task 2 | Planning |
| Task 3 | Requirement Gathering |
| Task 4 | Requirement Specification |
| Task 5 | Architectural Design |
| Task 6 | Initial Prototype |
| Task 7 | Test Prototype |
| Task 8 | Design Database |
| Task 9 | Classification |
| Task 10 | Server-Side Implementation |
| Task 11 | Module Development |
| Task 12 | Unit Testing |
| Task 13 | Integrating Module |
| Task 14 | System Testing |
| Task 15 | Deployment |

1. **Determine the Sequence of the Activities**

Table 6: Sequence of activities

|  |  |  |
| --- | --- | --- |
| **Activity** | **Predecessor** | **Duration in Days** |
| Task 1 | None | 5 days |
| Task 2 | Task1 | 7 days |
| Task 3 | Task2 | 10 days |
| Task 4 | Task2, Task3 | 7 days |
| Task 5 | Task1, Task4 | 14 days |
| Task 6 | Task4, Task5 | 10 days |
| Task 7 | Task6 | 7 days |
| Task 8 | Task4 | 8 days |
| Task 9 | Task5 | 5 days |
| Task 10 | Task5, Task9 | 10 days |
| Task 11 | Task7, Task5, Task10 | 15 days |
| Task 12 | Task11, Task8 | 10days |
| Task 13 | Task11, Task12 | 8 days |
| Task 14 | Task4, Task10, Task13 | 12 days |
| Task 15 | Nask14 | 10 days |

1. **Draw the Network Diagram**

****

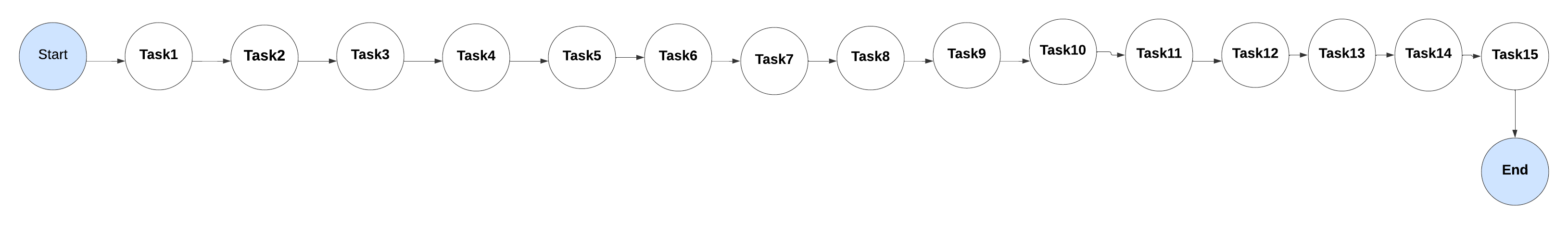
**Figure 1: Network Diagram**

1. **Estimate Activity Completion Time**

Table 7: Activity completion time

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Activity** | **Duration** | **ES** | **EF** | **LS** | **LF** | **TS** | **FS** |
| Task 1 | 5 days | 0 | 5 | 0 | 5 | 0 | 0 |
| Task 2 | 4 days | 5 | 12 | 5 | 12 | 0 | 0 |
| Task 3 | 2 days | 12 | 22 | 12 | 22 | 0 | 0 |
| Task 4 | 7 days | 22 | 29 | 22 | 29 | 0 | 0 |
| Task 5 | 6 days | 29 | 43 | 29 | 43 | 5 | 5 |
| Task 6 | 7 days | 43 | 53 | 43 | 53 | 5 | 5 |
| Task 7 | 6 days | 53 | 60 | 53 | 60 | 5 | 5 |
| Task 8 | 6 days | 60 | 68 | 60 | 68 | 0 | 0 |
| Task 9 | 7 days | 68 | 73 | 68 | 73 | 0 | 0 |
| Task 10 | 11 days | 73 | 83 | 73 | 83 | 0 | 0 |
| Task 11 | 6 days | 83 | 98 | 83 | 98 | 0 | 0 |
| Task 12 | 9 days | 98 | 108 | 98 | 108 | 0 | 0 |
| Task 13 | 6 days | 108 | 116 | 108 | 116 | 0 | 0 |
| Task 14 | 9 days | 116 | 128 | 116 | 128 | 0 | 0 |
| Task 15 | 10 days | 128 | 138 | 128 | 138 | 0 | 0 |

**5. Identify the Critical Path**

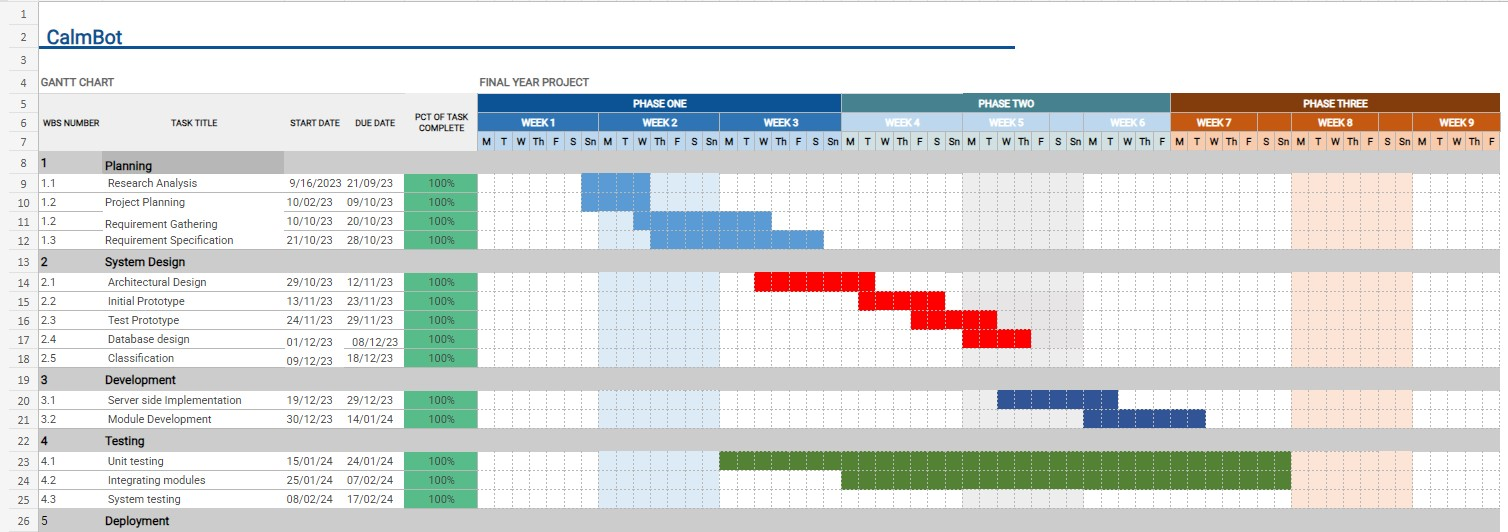
Figure 2: Critical Path

**Critical Path**: Task1 -> Task2 -> Task3 -> Task4 -> Task5 -> Task6 -> Task7 -> Task8

-> Task9 -> Task10 -> Task11 -> Task12 ->Task13 ->Task14 ->Task15->0

The sequence of activities outlined here constitutes the critical path for the "Chatbot For Mental Health" project, with a cumulative duration of 138 days. Any deviations or delays in the execution of activities along this path would directly impact the overall timeline for project completion.

## 1.7. Gantt chart

Figure 3: Gantt Chart

## 1.8. Introduction to Team member and their skill set

## Table 8: Team members, skill set and tasks

|  |  |  |
| --- | --- | --- |
| **Name of Members** | **Skill-set** | **Tasks** |
| **Anss Abrar**  **20021519-053** | Frontend Development, UI designing, Backend, Testing, Integration Documentation. | User interface, Database designing, Backend development, Collaboration in documentation. |
| **Ammara Khalid**  **20021519-057** | Frontend-Backend Integration, Testing, Documentation. | Requirements Gathering, database designing, writing documentation, Integrating frontend. |
| **Hizar Sajjad**  **20021519-068** | Documentation, SRS, Testing, Backend Development | Requirement Specification, writing documentation, Complete Black-box testing, Backend Development, Formatting document. Dataset collection and cleaning. |

## 1.9. Task and Member Assignment Table

Table 9:Tasks Duration and Dependencies

|  |  |  |
| --- | --- | --- |
| **Tasks** | **Dependencies** | **Duration(weeks)** |
| Task 1 | None | 5 days |
| Task 2 | Task 1 | 7 days |
| Task 3 | Task 2 | 10 days |
| Task 4 | Task2, Task3 | 7 days |
| Task 5 | Task1, Task4 | 14 days |
| Task 6 | Task4, Task5 | 10 days |
| Task 7 | Task6 | 7 days |
| Task 8 | Task4 | 8 days |
| Task 9 | Task8 | 5 days |
| Task 10 | Task8, Task9 | 10 days |
| Task 11 | Task7, Task5, Task10 | 3 days |
| Task 12 | Task11, Task8 | 7 days |
| Task 13 | Task11, Task12 | 5 days |
| Task 14 | Task4, Task12, Task13 | 9 days |
| Task 15 | Task 14 | 5 days |

**Anss Abrar = M1**

**Ammara Khalid = M2**

**Hizar Sajjad =M3**

## 1.10. Tools and Technology with reasoning

## Front-End:

## 1. Mobile App Development (Flutter):

## Reasoning: Flutter is chosen for its cross-platform development capabilities, allowing you to create a single codebase for both Android and iOS. This choice is cost-effectivand efficient for students. Flutter's rich set of pre-built widgets simplifies UI development.

## 2. UI/UX Design (Figma):

## Reasoning: Figma is a collaborative and user-friendly design tool. Its real-time collaboration and prototyping features make it suitable for small teams and students. It supports efficient design iterations and is cost-effective.

## Back-End:

## Programming Language (Python)

## Reasoning: Python is selected for chatbot development, especially when incorporating the Llama pre-trained model. Python's readability, versatility, extensive library support make it an optimal choice for natural language processing tasks.

## 

## 4. Dataset (Mental Health-related Subreddits):

## Reasoning: The choice of a dataset from Mental Health-related Subreddits is explained. Real talks from Reddit provide diverse experiences, and the privacy rules are followed, ensuring a good understanding for the chatbot.

## 5. Database (Firebase):

## Reasoning: Firebase is chosen for its scalable and user-friendly NoSQL database. Its cloud-based nature eliminates server maintenance, making it integration with the Python-based solution. Real-time updates and straightforward integration enhance efficiency.

## 6. Version Control (Git):

## Reasoning: Git is a standard version control system essential for collaborative development. Its free nature and wide industry usage ensure reliability and support for the project.

## 7. Integrated Development Environment (IDE - Visual Studio Code):

## Reasoning: Visual Studio Code is a free, lightweight, and highly extensible IDE. It has excellent support for both Python and Flutter development, making it a cost-effective choice for students.

## Tool Support Needs and Constraints:

## Development Platform:

## The tools are compatible with major platforms, including macOS, Windows, and Linux.

## 

## Target Platform:

## Flutter allows targeting both Android and iOS platforms.

## Programming Language:

## Python is versatile and aligns well with the project's requirements.

## Existing Tools:

## The selected tools are popular and widely used in the industry, ensuring reliability and support.

## Distribution of Development Organization:

## Tools support distributed teams effectively through version control and collaboration features.

## Size of Development Effort:

## The selected tools are suitable for small to medium-sized development efforts, making them cost-effective for students.

## Budget and Time Constraints:

## Most tools are open-source or offer free tiers, aligning well with budget constraints.

## Additional Considerations:

## Ethical Concerns:

## Consider addressing potential ethical concerns explicitly, especially related to data privacy and confidentiality.

## Risk Management:

## Consider including a section on risk management, identifying potential risks and mitigation strategies.

## Project Team Members:

## Include a section on project team members and their roles for a clearer understanding of project responsibilities.

## 1.11. Vision Document

Chatbot for Mental Health envisions a world where individuals have easy access to personalized mindfulness resources, creating a community that supports emotional well-being. By addressing the lack of awareness and limited access to emotional support, our chatbot strives to promote mindful living practices, empowering users to lead healthier and more fulfilling lives.

|  |  |
| --- | --- |
| Problems | Solutions |
| * Lack of Accessible Mental Health Support | Develop a versatile chatbot accessible via web and mobile platforms, providing immediate and confidential support for individuals facing mental health challenges such as depression, anxiety, overthinking, and suicidal thoughts. |
| * Stigma Surrounding Mental Health | Implement a user-friendly chatbot that reduces stigma by offering empathetic and personalized responses, creating a safe space for individuals to discuss their mental health concerns without fear of judgment. |
| * Privacy Concerns | Address privacy concerns by implementing robust data security practices, ensuring that user interactions and information are stored securely in a cloud-based database while maintaining confidentiality. |
| * Lack of Mindfulness Resources | Offer a comprehensive solution by integrating mindfulness resources such as relaxation techniques and meditation guidance into the chatbot, making them easily accessible to users at their fingertips. |
| * Limited Awareness about Mental Health Issues | Promote mental health awareness by utilizing the chatbot to provide information and resources about various mental health topics, understanding and reducing the overall lack of awareness. |
| * Difficulty in Navigating Mental Health Challenges | Provide a user-friendly interface and tailored guidance through the chatbot, helping individuals navigate mental health challenges more effectively by offering information, resources, and support. |
| * Lack of Immediate Support | Address the need for immediate support by designing the chatbot to engage users in meaningful conversations, offering timely assistance and creating a virtual lifeline for individuals in crisis. |

## 1.12. Risk List

* Delays in project timelines due to unforeseen circumstances.
* Delays caused by dependencies on external services or third-party APIs.
* Unforeseen expenses leading to a budget overrun.
* Compatibility issues with certain browsers or devices.

## 1.13. Product Features/ Product Decomposition

**Features:**

* Tailoring responses based on user voice tones.
* Understanding user queries for personalized interactions.
* Learning from behavior for tailored recommendations.
* Providing comfort with empathetic dialogue.

# 

# Chapter 2: Software Requirement Specification (For Object Oriented Approach)

## 2.1 Introduction:

In an era marked by escalating stressors, lifestyle pressures, and increasing awareness of mental health, the necessity for accessible and immediate support is more apparent than ever. Mental health issues, such as depression, anxiety, overthinking, and suicidal thoughts, affect millions worldwide, yet seeking professional assistance often remains hampered by limited access, the stigma surrounding mental health, and the prohibitive costs of therapy. To address this growing crisis, our project, titled "**Chatbot for Mental Health**," seeks to innovate by developing a versatile and user-friendly solution.

Grounded in the backdrop of these challenges, our project aims to create a chatbot that offers individuals an empathetic and confidential space to discuss their mental health concerns. We understand that navigating issues like depression and anxiety can be daunting, and for those contemplating suicide, timely support can be lifesaving. This project seeks to bridge the gap, providing accessible, stigma-free, and immediate assistance.

The Chatbot for Mental Health project will involve designing and developing a chatbot powered by natural language processing and machine learning technologies. This chatbot will serve as a scalable and adaptable solution, with the integration decision to be made at a later stage - whether it be within a web application or an Android app, depending on the project's evolving requirements. Through this chatbot, users can seek information, resources, and guidance tailored to their specific mental health concerns. Moreover, user feedback will be meticulously collected and analyzed to enhance the chatbot's capabilities continuously.

By offering an innovative approach to mental health support, this project aspires to reduce the stigma associated with seeking help, increase access to much-needed resources, and ultimately improve the well-being of those facing mental health challenges. The potential of technology to serve as a lifeline for individuals battling depression, anxiety, overthinking, and suicidal thoughts is both promising and transformative. This project endeavors to harness this potential, promising hope and assistance to those in need.

• Identify external interfaces

• Development of context diagram

• Capture “shall statements

• Allocate requirements

• Prioritize requirements

• Development of requirements traceability matrix

### 2.1.1 Systems Specifications

The following are the clauses that must be included while describing the system specifications.

**Introduction**

The Chatbot for Mental Health project will involve designing and developing a chatbot powered by natural language processing and machine learning technologies. This chatbot will serve as a scalable and adaptable solution, with the integration decision to be made at a later stage - whether it be within a web application or an Android app, depending on the project's evolving requirements. Through this chatbot, users can seek information, resources, and guidance tailored to their specific mental health concerns. Moreover, user feedback will be meticulously collected and analyzed to enhance the chatbot's capabilities continuously.

**Existing System**

**Woebot:**

The focal point of the Woebot project is the development of an interactive and emotionally intelligent chatbot. The project includes essential components like emotion detection, emotion-driven dialogue, privacy management, and user registration. Woebot is engineered to provide users with real-time, human-like responses, offering mindfulness exercises and techniques for emotional well-being. The user registration module enables individuals to personalize their interactions, emphasizing privacy.

**Replica:**

The project scope for Replica is centered on the development of a responsive and emotionally intelligent chatbot. It includes crucial components such as emotion detection, emotion-driven dialogue, privacy management, and user registration. The primary goal is to create a chatbot that offers real-time, human-like responses while delivering mindfulness exercises and techniques to users. The user registration module provides individuals with the ability to personalize their interactions, ensuring a private and tailored experience.

**Organizational Chart**

**Scope of the System**

Our "Chatbot for Mental Health" system is specifically tailored to cater to individuals in search of immediate emotional support, relaxation techniques, and stress relief. It aims to create a confidential and empathetic space for users dealing with mental health challenges such as depression, anxiety, overthinking, and suicidal thoughts. The platform offers a conversational AI companion, incorporating advanced features like speech recognition and emotion detection to provide personalized assistance. Privacy measures are implemented to ensure user confidentiality during sensitive interactions about their mental well-being. The scope of the system extends to fostering a supportive environment accessible through both web and mobile platforms, offering tailored guidance and resources to enhance users' mental health and well-being.

**Summary of Requirements: (Initial Requirements)**

**Functional Requirements:**

Users are expected to undergo a comprehensive and empathetic interaction with the Chatbot for Mental Health. The registration process involves email registration, account creation, and login using registered credentials. Users are empowered to express emotions, receive personalized support. Additionally, users can save, mark, manage, and delete favorites. The system offers access to interaction history, emotional wellness progress, and a variety of mindfulness resources. To enhance user experience, the Chatbot for Mental Health processes text inputs, incorporates noise reduction, utilizes sentiment analysis and emotion identification, and delivers emotionally supportive, personalized, and natural-sounding speech responses through advanced Natural Language Processing (NLP) techniques and context management.

**Non-Functional Requirements:**

1. **Usability:** System should be intuitive and user-friendly, ensuring ease of navigation and a positive user experience.

2. **Reliability:** System should be reliable and available at all times.

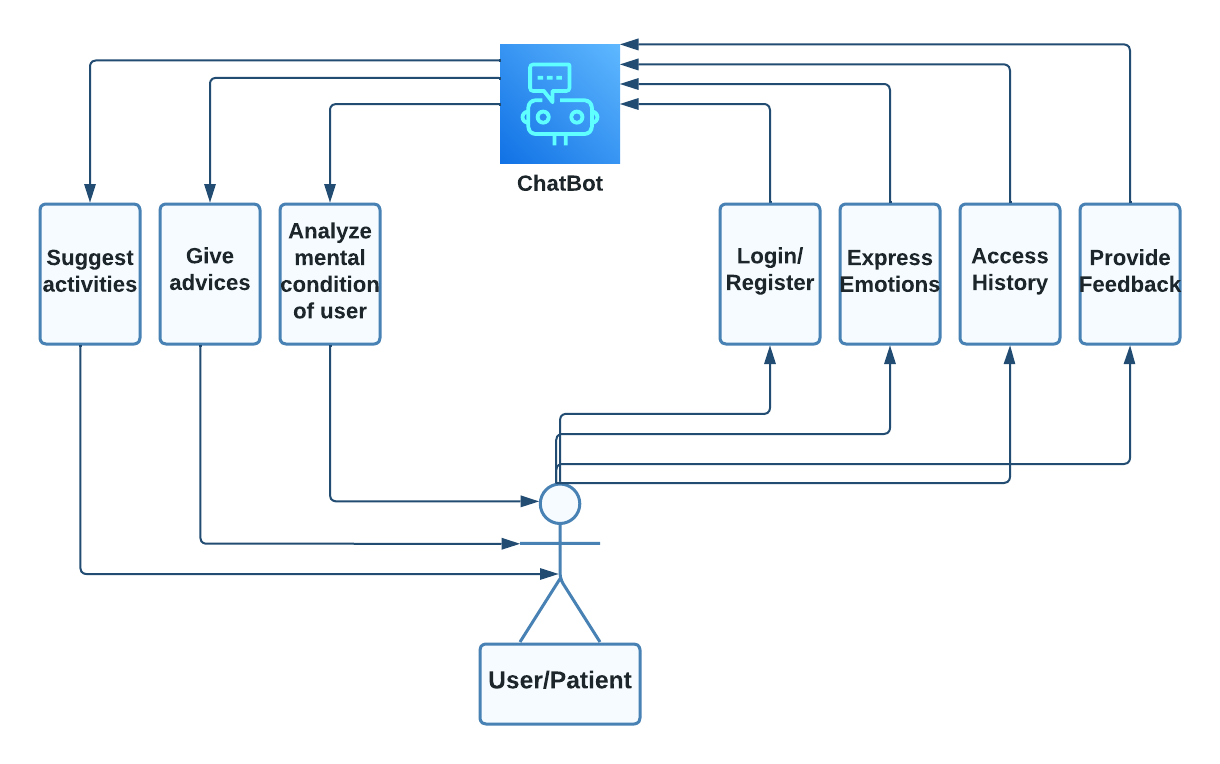
3. **Security:** System should have secure authentication and authorization mechanisms to protect user data.

4. **Performance:** System should be able to handle large amounts of data and respond quickly to user requests.

5. **Scalability:** System should be able to scale up or down depending on user demand.

6. **Compatibility:** System should be compatible with multiple -platforms and devices.

### 2.1.3. Context Level Data Flow Diagram:



### 2.1.4. Capture "shall" Statements:

Identify “shall” statements, as they would be all functional requirements.

1. Represents user

2. Represents chatbot

Table 13.Functional requirements

|  |  |
| --- | --- |
| **Sr#** | **Functional Requirements** |
| 1.1 | User "shall" have options to register through email. |
| 1.2 | User “shall” create accounts with unique identifiers |
| 1.3 | User "shall" log in to the system using their registered credentials |
| 1.4 | User "shall" receive an email confirmation upon successful registration for account verification. |
| 1.5 | User "shall" update their profile by entering general details. |
| 1.6 | User "shall" have the ability to upload and manage a profile picture. |
| 1.7 | User "shall" express emotions and seek personalized support from the chatbot. |
| 1.8 | User "shall" receive empathetic and tailored responses based on their emotional expressions |
| 1.9 | User “shall" sends a text/message. |
| 1.10 | User "shall" have access to delete message. |
| 1.11 | User "shall" be empowered to save or mark certain responses or exercises as favorites |
| 1.13 | User "shall" have access to their interaction history and personalized emotional wellness progress |
| 1.14 | User "shall" receive a range of mindfulness techniques and emotional well-being resources |
| 2.0 | Chatbot "shall" process user message inputs. |
| s2.2 | Chatbot "shall" implement sentiment analysis to determine the emotional tone of user input. |
| 2.3 | Chatbot "shall" identify and classify user emotions. |
| 2.4 | Chatbot "shall" deliver emotionally supportive responses. |
| 2.5 | Chatbot "shall" deliver personalized responses to users' emotional states. |
| 2.6 | Chatbot "shall" synthesize text responses into natural-sounding speech for user interaction |
| 2.7 | Chatbot "shall" utilize advanced NLP techniques, including sentiment analysis and context management |
| 2.8 | Chatbot "shall" manage and maintain contextual information for personalized and context-aware responses |

### 2.1.5. Allocate Requirements:

Table 14. Allocate Requirements

|  |  |  |
| --- | --- | --- |
| **Sr#** | **Functional Requirements** | **Use-case** |
| 1.1 | The system “shall” provide a user registration process. | UC\_Register\_Account |
| 1.2 | User “shall” create accounts with unique identifiers | UC\_Register\_Account |
| 1.3 | User "shall" log in to the system using their registered credentials | UC\_User\_Login |
| 1.4 | User "shall" receive an email confirmation upon successful registration for account verification. | UC\_Register\_Account |
| 1.5 | User "shall" update their profile by entering general details. | UC\_Engage\_with\_Bot |
| 1.6 | User "shall" have the ability to upload and manage a profile picture. | UC\_Engage\_with\_Bot |
| 1.7 | User "shall" express emotions and seek personalized support from the chatbot. | UC\_Engage\_with\_Bot |
| 1.8 | User "shall" receive empathetic and tailored responses based on their emotional expressions | UC\_Generate\_  Personalized\_  responses |
| 1.9 | User "shall" engage in one-on-one conversations with the chatbot for emotional guidance and support. | UC\_Engage\_with\_Bot |
| 1.10 | User "shall" be empowered to save or mark certain responses or exercises as favorites | UC\_Engage\_with\_Bot |
| 1.11 | User "shall" manage and delete their saved favorites as needed | UC\_Engage\_with\_Bot |
| 1.12 | User "shall" have access to their interaction history and personalized emotional wellness progress | UC\_ Engage\_with\_Bot |
| 1.13 | User "shall" receive a range of mindfulness techniques and emotional well-being resources | UC\_Engage\_with\_Bot |
| 2.0 | Chatbot "shall" implement sentiment analysis to determine the emotional tone of user input. | UC\_Analyze\_text\_Emotions |
| 2.1 | Chatbot "shall" identify and classify user emotions. | UC\_Analyze\_text\_Emotions |
| 2.2 | Chatbot "shall" deliver emotionally supportive responses. | UC\_Generate\_Personalized\_  responses |
| 2.3 | Chatbot "shall" deliver personalized responses to users' emotional states. | UC\_Generate\_Personalized\_responses |
| 2.4 | Chatbot "shall" synthesize text responses into natural-sounding speech for user interaction | UC\_Synthesize\_speech |
| 2.5 | Chatbot "shall" utilize advanced NLP techniques, including sentiment analysis and context management | UC\_Context\_management |

### 2.1.6. Prioritize Requirements:

Table 15.Prioritize Requirements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr#** | **Rank** | **Functional Requirements** | **UC\_ID** | **Use-case** |
| 1.1 | Highest | User "shall" have options to register through email. | UC-1 | UC\_Register\_Account |
| 1.2 | Highest | User “shall” create accounts with unique identifiers | UC-1 | UC\_Register\_Account |
| 1.3 | Medium | User "shall" log in to the system using their registered credentials | UC-2 | UC\_User\_Login |
| 1.4 | Highest | User "shall" receive an email confirmation upon successful registration for account verification. | UC-1 | UC\_Register\_Account |
| 1.5 | Highest | User "shall" update their profile by entering general details. | UC-4 | UC\_Engage\_with\_Bot |
| 1.6 | Highest | User "shall" have the ability to upload and manage a profile picture. | UC-4 | UC\_Engage\_with\_Bot |
| 1.7 | Highest | User "shall" express emotions and seek personalized support from the chatbot. | UC-4 | UC\_Engage\_with\_Bot |
| 1.8 | Highest | User "shall" receive empathetic and tailored responses based on their emotional expressions | UC-10 | UC\_Generate\_  Personalized\_  responses |
| 1.9 | Highest | User "shall" engage in one-on-one conversations with the chatbot for emotional guidance and support. | UC-4 | UC\_Engage\_with\_Bot |
| 2.0 | Highest | User "shall" be empowered to save or mark certain responses or exercises as favorites | UC-4 | UC\_Engage\_with\_Bot |
| 2.1 | Highest | User "shall" manage and delete their saved favorites as needed | UC-4 | UC\_Engage\_with\_Bot |
| 2.2 | Medium | User "shall" have access to their interaction history and personalized emotional wellness progress | UC-12 | UC\_ Engage\_with\_Bot |
| 2.3 | Highest | User "shall" receive a range of mindfulness techniques and emotional well-being resources | UC-4 | UC\_Engage\_with\_Bot |
| 2.4 | Highest | Chatbot "shall" identify and classify user emotions. | UC-7 | UC\_Analyze\_text\_Emotions |
| 2.5 | Highest | Chatbot "shall" deliver emotionally supportive responses. | UC-10 | UC\_Generate\_Personalized\_  responses |
| 2.6 | Highest | Chatbot "shall" deliver personalized responses to users' emotional states. | UC-10 | UC\_Generate\_Personalized\_  responses |
| 2.7 | Highest | Chatbot "shall" synthesize text responses into natural-sounding speech for user interaction | UC-11 | UC\_Synthesize\_speech |
| 2.8 | Highest | Chatbot "shall" utilize advanced NLP techniques, including sentiment analysis and context management | UC-9 | UC\_Context\_management |

### 

### 2.1.7. Requirements Trace-ability Matrix:

Table 16. Requirements Trace-ability Matrix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr#** | **Functional Requirements** | **Build** | **Use-case** | **Category** |
| 1 | User "shall" have options to register through email. | B1 | UC\_Register\_Account | Business |
| 2 | User “shall” create accounts with unique identifiers | B1 | UC\_Register\_Account | Business |
| 3 | User "shall" log in to the system using their registered credentials | B1 | UC\_User\_Login | Business |
| 4 | User "shall" receive an email confirmation upon successful registration for account verification. | B1 | UC\_Register\_Account | Business |
| 5 | User "shall" update their profile by entering general details. | B1 | UC\_Engage\_with\_Bot | User |
| 6 | User "shall" have the ability to upload and manage a profile picture. | B1 | UC\_Engage\_with\_Bot | User |
| 7 | User "shall" express emotions and seek personalized support from the chatbot. | B1 | UC\_Engage\_with\_Bot | User |
| 8 | User "shall" receive empathetic and tailored responses based on their emotional expressions | B1 | UC\_Generate\_  Personalized\_  responses | User |
| 9 | User "shall" engage in one-on-one conversations with the chatbot for emotional guidance and support. | B1 | UC\_Engage\_with\_Bot | User |
| 10 | User "shall" be empowered to save or mark certain responses or exercises as favorites | B1 | UC\_Engage\_with\_Bot | User |
| 11 | User "shall" manage and delete their saved favorites as needed | B1 | UC\_Engage\_with\_Bot | User |
| 12 | User "shall" have access to their interaction history and personalized emotional wellness progress | B1 | UC\_ Engage\_with\_Bot | User |
| 13 | User "shall" receive a range of mindfulness techniques and emotional well-being resources | B1 | UC\_Engage\_with\_Bot | User |
| 14 | Chatbot "shall" implement sentiment analysis to determine the emotional tone of user input. | B1 | UC\_Analyze\_text\_Emotions | User |
| 15 | Chatbot "shall" identify and classify user emotions. | B1 | UC\_Analyze\_text\_Emotions | User |
| 16 | Chatbot "shall" deliver emotionally supportive responses. | B1 | UC\_Generate\_Personalized\_  responses | User |
| 17 | Chatbot "shall" deliver personalized responses to users' emotional states. | B1 | UC\_Generate\_Personalized\_  responses | User |
| 18 | Chatbot "shall" synthesize text responses into natural-sounding speech for user interaction | B1 | UC\_Synthesize\_speech | User |
| 19 | Chatbot "shall" utilize advanced NLP techniques, including sentiment analysis and context management | B1 | UC\_Context\_management | User |

### 2.2.10. High Level Use-case Diagram:



Figure 6: Use case Diagram

### 2.2.11. Analysis Level Use-case Diagram:

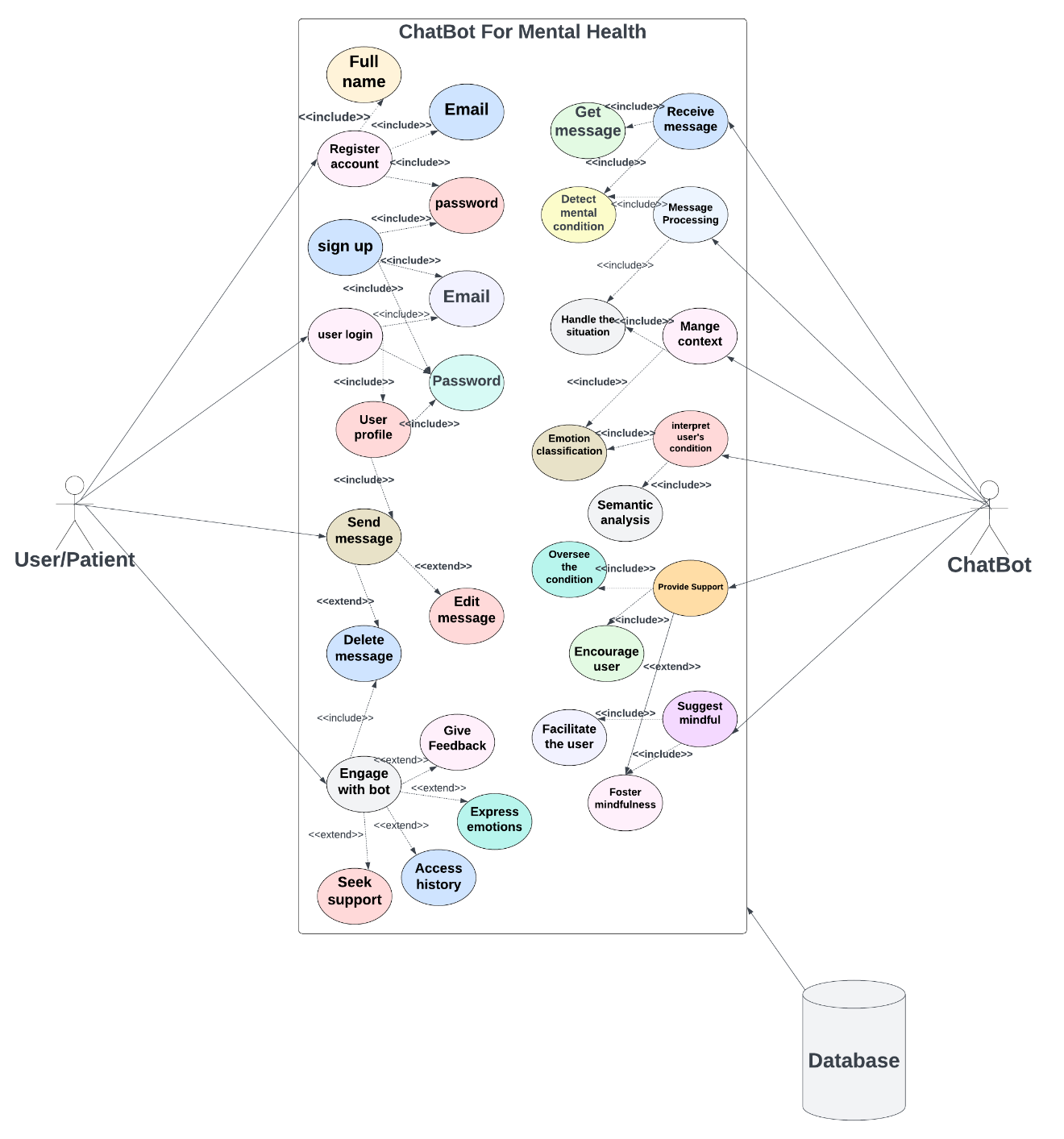


Figure 7: Analysis Level use-case diagram

### 2.2.12. Use-case Description

**Registration:**

|  |
| --- |
| **Use case Name**: UC\_ Registration **Use-case ID:** UC\_1  **Actors**: User **Created By**: Anss Abrar, Ammara  Khalid, Hizar Sajjad  **Date**: 24/12/2023 |
| **Description:**  Users can register for a Chatbot For Mental Health account to access personalized features and content |
| **Pre-conditions:**  The user should have a internet connection |
| **Basic Flow:**   |  |  | | --- | --- | | **User** | **System** | | 1. User enters required personal details and presses the "Register" button. | 1. Chatbot For Mental Health successfully captures user details and registers the account. | | 1. User receives a confirmation message of successful registration. | 1. The user can now access the Chatbot For Mental Health app with a registered account. | |
| **Alternative Flow:**  In point 3, the user attempts to register without entering all required data. An error message prompts the user to add the necessary details.  In point 2, the user navigates away without completing the registration. Chatbot For Mental Health acknowledges the exit, and the use case ends. |
| **Post-Conditions:**  The user has successfully registered an account and can access Chatbot For Mental Health's features with their login credentials. |

**Login:**

|  |
| --- |
| **Use case Name**: UC\_ Login **Use case ID:** UC\_2  **Actors**: User **Created By**: Anss Abrar, Ammara  Khalid, Hizar Sajjad  **Date**: 24/12/2023 |
| **Description:**  Users log in to the Chatbot For Mental Health system to access personalized features and continue their interactions |
| **Pre-conditions:**  The user has a registered account with Chatbot For Mental Health. |
| **Basic Flow:**   |  |  | | --- | --- | | **User** | **System** | | 1. User enters their username and password and clicks the "Login" button. | 1. Chatbot For Mental Health verifies the entered credentials against the registered user database. | |  | 1. Chatbot For Mental Health grants access and acknowledges successful login. | | 1. The user gains access to personalized features and continues their interaction within the Chatbot For Mental Health system |  | |
| **Alternative Flow:**  In point 3, if the user enters incorrect credentials, Chatbot For Mental Health informs the user of the error and may provide an option for password recovery or account assistance. |
| **Post-Conditions:**  The user successfully logs in to the Chatbot For Mental Health system, gaining access to personalized features |

**Send Message:**

|  |
| --- |
| **Use case Name**: UC\_ User\_ Send\_ Message **Use case ID:** UC\_3  **Actors**: User **Created By**: Anss Abrar, Ammara  Khalid, Hizar Sajjad  **Date**: 24/12/2023 |
| **Description:**  Users can send a message to Chatbot For Mental Health for more personalized interactions. |
| **Pre-conditions:**  The user should have a device with a keyword and internet access. |
| **Basic Flow:**   |  |  | | --- | --- | | **User** | **System** | | 1. Users write a message and presses the "Send" button | 1. System processes the message input. | |  |  | |
| **Alternative Flow:**  If the user does not press the send button, Chatbot For Mental Health acknowledges the cancellation, and the use case ends. |
| **Post-Conditions:**  The user successfully sends a message. |

**Analyse Text Emotions:**

|  |
| --- |
| **Use case Name**: UC\_ Analyze\_text\_emotions **Use case ID:** UC\_4  **Actors**: Chatbot **Created By**: Anss Abrar, Ammara  Khalid, Hizar Sajjad  **Date**: 24/12/2023 |
| **Description:**  Chatbot For Mental Health analyses textual input from the user to detect and understand emotional content and sentiment. |
| **Pre-conditions:** Chatbot For Mental Health has received a text message from the user. |
| **Basic Flow:**   |  |  | | --- | --- | | **User** | **System** | | 1.Chattbot\_For\_Mental\_Health analyses the text for emotional content and sentiment. |  | |  | 2.The system classifies the emotional state conveyed in the text (e.g., happy, sad, calm). | |  |  | |
| **Alternative Flow:**  In point 1, if Chatbot For Mental Health encounters challenges in analyzing the text for emotions, it may inform the user and ask for clarification |
| **Post-Conditions:**  Chatbot For Mental Health successfully analyzes the text for emotions, classifies emotional states, generates insights. |

**Context Management:**

|  |
| --- |
| **Use case Name**: UC\_Context management **Use case ID:** UC\_5  **Actors**: Chatbot **Created By**: Ans Abrar, Ammara  Khalid, Hizar Sajjad  **Date**: 24/12/2023 |
| **Description:**  Chatbot For Mental Health manages and maintains context information to provide personalized and context-aware responses to the user. |
| **Pre-conditions:**  Chatbot For Mental Health is actively engaged in a conversation with the user, and context information is available. |
| **Basic Flow:**   |  |  | | --- | --- | | **User** | **System** | |  | 1. Chatbot For Mental Health receives user inputs and maintains a conversation context. | | 2. The system stores and manages relevant information from the ongoing conversation. |  | |  | 1. Chatbot For Mental Health utilizes the stored context to tailor responses based on the user's history and preference | |
| **Alternative Flow:**  In point 1, should Chatbot for Mental Health experience a loss of connection or session interruption, it will promptly notify the user and make efforts to re-establish the connection, aiming to seamlessly resume the ongoing context upon successful reconnection. |
| **Post-Conditions:**  Chatbot For Mental Health successfully manages and maintains the context of the ongoing conversation, providing personalized and context-aware responses to the user. |

**Response Generation:**

|  |
| --- |
| **Use case Name**: UC\_ Generate\_personalized\_responses **Use case ID:** UC\_6  **Actors**: Chatbot **Created By**: Ans Abrar, Ammara  Khalid, Hizar Sajjad  **Date**: 24/12/2023 |
| **Description:**  Chatbot For Mental Health generates context-aware and personalized responses based on the maintained conversation context. |
| **Pre-conditions:**  Chatbot For Mental Health is actively engaged in a conversation with the user, and relevant context information is available. |
| **Basic Flow:**   |  |  | | --- | --- | | **User** | **System** | |  | 1. Chatbot For Mental Health receives user inputs and accesses the stored conversation context | | 1. The system analyses the context to understand the user's history, preferences, and ongoing conversation details. |  | |  | 1. Chatbot For Mental Health generates a response tailored to the user's individual needs | |
| **Alternative Flow:**  If, during response generation, Chatbot for Mental Health encounters difficulties or lacks sufficient context, it informs the user about the limitation and seeks clarification or additional information to enhance response accuracy |
| **Post-Conditions:**  Chatbot for Mental Health successfully generates context-aware and personalized responses. |

# Chapter 3: Design Document (For Object Oriented Approach)

## 3.1. Introduction:

In an era marked by escalating stressors, lifestyle pressures, and increasing awareness of mental health, the necessity for accessible and immediate support is more apparent than ever. Mental health issues, such as depression, anxiety, overthinking, and suicidal thoughts, affect millions worldwide, yet seeking professional assistance often remains hampered by limited access, the stigma surrounding mental health, and the prohibitive costs of therapy. To address this growing crisis, our project, titled "**Chatbot for Mental Health**," seeks to innovate by developing a versatile and user-friendly solution.

Grounded in the backdrop of these challenges, our project aims to create a chatbot that offers individuals an empathetic and confidential space to discuss their mental health concerns. We understand that navigating issues like depression and anxiety can be daunting, and for those contemplating suicide, timely support can be lifesaving. This project seeks to bridge the gap, providing accessible, stigma-free, and immediate assistance.

## 3.2. Domain Model

Domain models represent the set of requirements that are common to systems within a product line. There may be many domains, or areas of expertise, represented in a single product line and a single domain may span multiple product lines. The requirements represented in a domain model include:

* Definition of scope for the domain
* Information or objects
* Features or use cases, including factors that lead to variation
* Operational/behavioral characteristics

A product line definition will describe the domains necessary to build systems in the product line.

What is domain modeling?

## In the "Chatbot for Mental Health" project, domain modeling involves creating a representation of the key concepts, entities, and relationships within the mental opponents of the system. Here's how domain modeling could be approached for this project:

## 1. Identifying Entities:

## User: Represents individuals seeking mental health support through the chatbot.

## Chatbot: The core system that interacts with users, providing support and resources.

## Mental Health Topics: Entities related to different aspects of mental health, including depression, anxiety, overthinking, and suicidal thoughts.

## Chat Interactions: Represents the conversations between users and the chatbot.

## Feedback: User feedback collected for system improvement.

## Geographical Areas: Entities representing locations to address disparities in access.

## 2. Defining Attributes:

## User Attributes: Name, age, location, current emotional state.

## Chatbot Attributes: Pre-trained model, conversation logs, crisis intervention capabilities.

## Mental Health Topics Attributes: Information, resources, coping mechanisms.

## Chat Interaction Attributes: Timestamps, user responses, chatbot responses.

## Feedback Attributes: Ratings, comments, suggestions.

## 3. Establishing Relationships:

## User-Chatbot Relationship: Users engage in conversations with the chatbot for mental health support.

## User-Feedback Relationship: Users provide feedback to the system.

## Chatbot-Mental Health Topics Relationship: The chatbot provides information and resources related to mental health topics.

## User-Geographical Areas Relationship: Considering geographical disparities, users from different areas may have specific needs.

## 4. Modeling Behavior:

## User Behavior: Initiates conversations, seeks information, provides feedback.

## Chatbot Behavior: Engages in conversations, provides information, identifies and responds to crises.

## System Behavior: Collects and analyzes user feedback, continuously improves chatbot capabilities.

## 5. Considering Constraints:

## Privacy Constraints: Ensuring user data and conversations are handled confidentially.

## Ethical Constraints: Adhering to ethical guidelines in providing mental health support.

## Data Security Constraints: Implementing measures to secure user data.

## 6. Future Extensions:

## Integration Entities: Representing potential future integration with other mental health services.

## Collaboration Entities: Entities related to collaborations with healthcare providers or workplace wellness programs.

## 3.3. System Sequence Diagram

The System Sequence Diagram (SSD) in our "Chatbot for Mental Health" project illustrates the sequential input of events from external sources to the system. This diagram serves to define the system events and operations, offering a timeline depiction of an expanded use case. The events are chronologically arranged, with the top events occurring first. Notably, system events, which trigger a system response, are highlighted as crucial elements. Optionally, use case text can be positioned on the left side of the SSD, ensuring alignment with the corresponding events in the diagram. The system may involve multiple actors, which could be individuals interacting with the chatbot, and possibly external automated systems with which the chatbot communicates. Automated actors or robots are represented with a line horizontally through the head in the diagram, distinguishing them from human actors.

## 3.4. Sequence Diagram

### A Sequence Diagram serves as a visual representation illustrating the chronological sequence of actions within a system. It captures the invocation of methods in each object and their order of occurrence, providing a valuable tool to depict the dynamic behavior of the "Chatbot for Mental Health" project proposed in this document. The two-dimensional nature of the Sequence Diagram is utilized to showcase the object's lifespan on the horizontal axis and the sequential creation or invocation of these objects on the vertical axis. Utilizing class and object name references, the Sequence Diagram becomes instrumental in elucidating and detailing the dynamic design, as well as the sequence and origin of object invocations within the context of the "Chatbot for Mental Health." Therefore, the Sequence Diagram stands out as a fundamental and widely used dynamic diagram in UML for representing the dynamic interactions within the proposed system.

### 3.4.1. Defining a Sequence diagram

A Sequence Diagram comprises entities and messages, with entities depicted in the familiar UML format—rectangles featuring the underlined class name. In line with UML conventions, objects are represented in this manner across all diagrams. The primary function of Sequence diagrams is to articulate interactions between classes through a temporal exchange of messages. These diagrams effectively illustrate how various components or classes communicate with each other over a specified period, providing a clear depiction of the dynamic relationships within the system.

### 3.4.2. Basic Sequence Diagram Symbols and Notations

**Class roles**

In our project **"Chatbot for Mental Health"**, class roles are depicted using UML object symbols to illustrate the behavior of objects such as the **Chatbot** and **User**.These symbols serve to represent the functionality and responsibilities of each class without explicitly listing object attributes. The focus is on conveying the roles and interactions of the classes within the proposed mental health chatbot system.

Activation

Activation boxes in the sequence diagram illustrate the temporal duration during which an object, such as the 'Chatbot' or 'User,' is actively engaged in executing a specific task within the "Chatbot for Mental Health" system. These boxes provide a visual representation of the time taken by an object to complete a particular operation, whether it be processing an authentication request, generating a chatbot response, or handling user input related to mental health concerns. The duration of these activation boxes captures the dynamic nature of interactions and task execution, aiding in the comprehension of the temporal aspects of the system's behavior.

Messages

In our project "Chatbot for Mental Health" , messages are depicted as arrows symbolizing communication between objects, such as the **Chatbot** and the **User**.The sequence diagram employs half-arrowed lines to specifically illustrate asynchronous messages. These asynchronous messages signify communication initiated by an object without waiting for an immediate response from the receiver, enabling the sender to proceed with its tasks concurrently. This asynchronous communication approach aligns with the dynamic and real-time nature of interactions between the chatbot and users seeking mental health support.

Lifelines

Lifelines are depicted as vertical dashed lines, symbolizing the existence of objects such as the **Chatbot** and **User**. over the course of the interaction. These lines visually represent the temporal presence of the respective objects throughout the sequence diagram, illustrating their engagement and participation in the mental health support conversation.

Destroying Objects

In our project Chatbot for Mental Health, the termination of objects within the sequence diagram is symbolized by an arrow labeled "<<destroy>>." This notation signifies the early termination or cessation of an object's existence in the context of the mental health chatbot interaction. This may occur, for instance, when the user session concludes or when certain conditions, specified within the sequence diagram, necessitate the premature destruction of an object, ensuring clarity and coherence in representing the lifecycle of objects during the interactive processes.

Loops

Loops within a sequence diagram are visually represented as rectangles. The condition for exiting the loop is placed at the bottom left corner inside square brackets [ ]. This notation is employed to illustrate repetitive interactions or processes within the sequence diagram, providing a clear depiction of the conditions under which the loop will be terminated. When the user enters wrong password, he will go to the login screen until he enters the right password.

Objects

In our "Chatbot for Mental Health" project, objects are depicted as vertical dashed lines, known as lifelines, which symbolize the existence of an object over time. At the head of each lifeline, an object symbol is drawn, displaying the name of the object and its class, separated by a colon in the format `object name: class name`. These objects can be utilized in sequence diagrams in various ways. Typically, a lifeline represents all objects of a specific class, showcasing both class and object behavior. In cases where the object's class is unspecified during the initial diagram creation, classes can be specified later in the process. Objects can be unnamed, but naming them becomes essential to distinguish between different objects of the same class. Additionally, multiple lifelines within the same diagram can represent distinct objects of the same class, provided each object is named for discrimination. Furthermore, a lifeline representing a class can coexist in parallel with lifelines representing individual objects of that class, with the object name of the class set to the name of the class itself.

Actors

Normally an actor instance is represented by the first (left-most) lifeline in the sequence diagram, as the invoker of the interaction. If you have several actor instances in the same diagram, try keeping them either at the left-most, or the right-most lifelines.

Messages

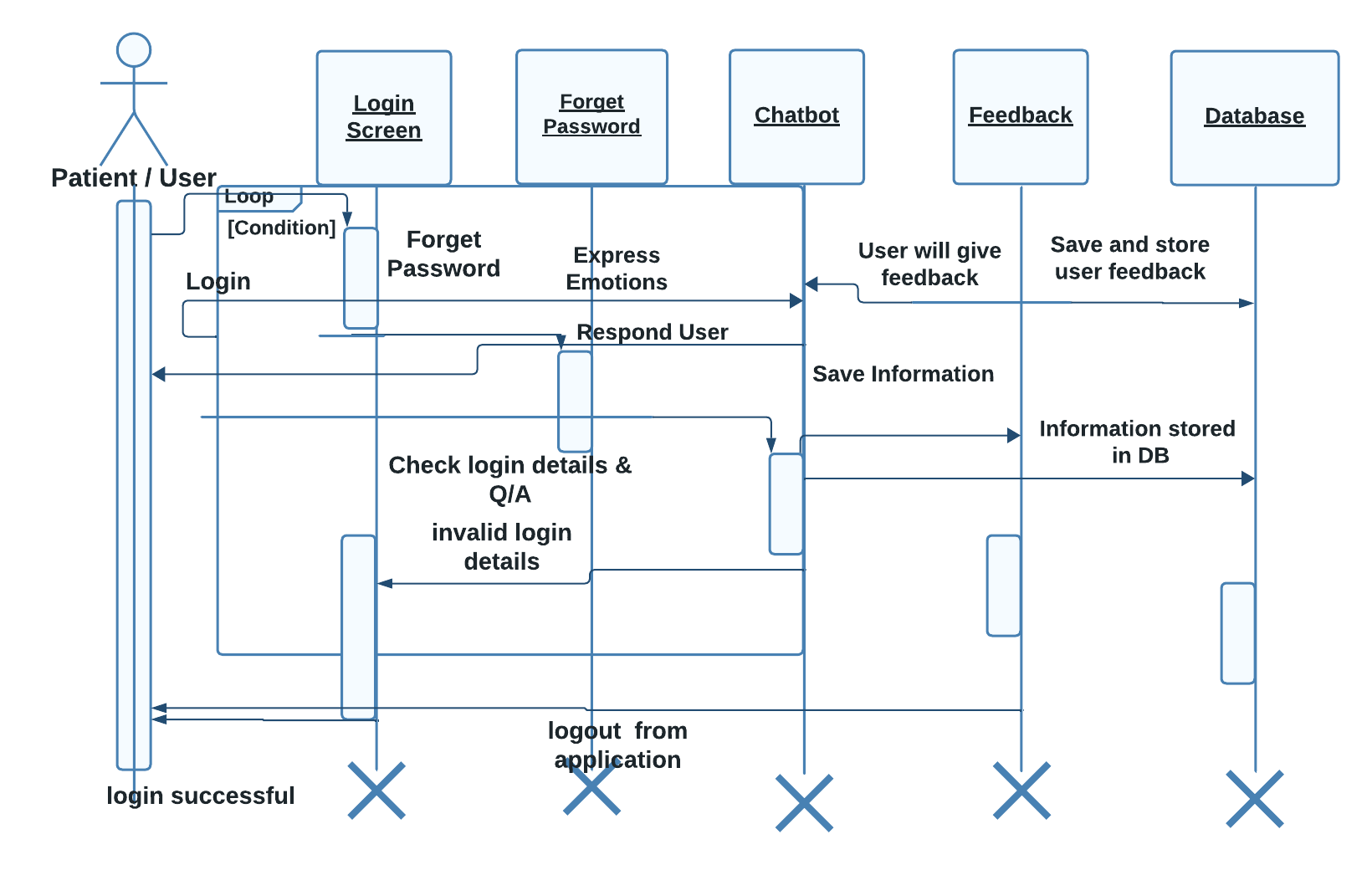
A message is a communication between objects that conveys information with the expectation that activity will ensue; in sequence diagrams, a message is shown as a horizontal solid arrow from the lifeline of one object to the lifeline of another object. In the case of a message from an object to itself, the arrow may start and finish on the same lifeline. The arrow is labeled with the name of the message, and its parameters. The arrow may also be labeled with a sequence number to show the sequence of the message in the overall interaction. Sequence numbers are often omitted in sequence diagrams, in which the physical location of the arrow shows the relative sequence.

A message can be unassigned, meaning that its name is a temporary string that describes the overall meaning of the message and is not the name of an operation of the receiving object. You can later assign the message by specifying the operation of the message's destination object. The specified operation will then replace the name of the message.

Scripts

Scripts describe the flow of events textually in a sequence diagram.

You should position the scripts to the left of the lifelines so that you can read the complete flow from top to bottom (see figure above). You can attach scripts to a certain message, thus ensuring that the script moves with the message.



Sequence diagram

### 3.4.4. Distributing Control Flow in Sequence Diagrams

**Centralized control** of a flow of events or part of the flow of events means that a few objects steer the flow by sending messages to, and receiving messages from other objects. These controlling objects decide the order in which other objects will be activated in the use case. Interaction among the rest of the objects is very minor or does not exist.

Example

In the Recycling-Machine System, the use case Print Daily Report keeps track of - among other things - the number and type of returned objects, and writes the tally on a receipt. The Report Generator control object decides the order in which the sums will be extracted and written.

The behavior structure of the use case Print Daily Report is centralized in the Report Generator control object.

This is an example of centralized behavior. The control structure is centralized primarily because the different sub-event phases of the flow of events are not dependent on each other. The main advantage of this approach is that each object does not have to keep track of the next object's tally. To change the order of the sub-event phases, you merely make the change in the control object. You can also easily add still another sub-event phase if, for example, a new type of return item is included. Another advantage to this structure is that you can easily reuse the various sub-event phases in other use cases because the order of behavior is not built into the objects.

Decentralized control arises when the participating objects communicate directly with one another, not through one or more controlling objects.

Example

In the use case Send Letter someone mails a letter to another country through a post office. The letter is first sent to the country of the addressee. In the country, the letter is sent to a specific city. The city, in turn, sends the letter to the home of the addressee.

The behavior structure of the use case **Send Letter** is decentralized.

The use case behavior is a decentralized flow of events. The sub-event phases belong together. The sender of the letter speaks of "sending a letter to someone." He neither needs nor wants to know the details of how letters are forwarded in countries or cities. (Probably, if someone were mailing a letter within the same country, not all these actions would occur.)

The type of control used depends on the application. In general, you should try to achieve independent objects, that is, to delegate various tasks to the objects most naturally suited to perform them.

A flow of events with centralized control will have a "fork-shaped" sequence diagram. On the other hand, a "stairway-shaped" sequence diagram illustrates that the control-structure is decentralized for the participating objects.

A centralized control structure in a flow of events produces a "fork-shaped" sequence diagram. A decentralized control structure produces a "stairway-shaped" sequence diagram.

The behavior structure of a use-case realization most often consists of a mix of centralized and decentralized behavior.

A decentralized structure is appropriate:

* If the sub-event phases are tightly coupled. This will be the case if the participating objects:
* Form a part-of or consists-of hierarchy, such as Country - State - City;
* Form an information hierarchy, such as CEO - Division Manager - Section Manager;
* Represent a fixed chronological progression (the sequence of sub-event phases will always be performed in the same order), such as Advertisement - Order - Invoice -Delivery - Payment; or
* Form a conceptual inheritance hierarchy, such as Animal - Mammal - Cat.
* If you want to encapsulate, and thereby make abstractions of, functionality. This is good for someone who always wants to use the whole functionality, because the functionality can become unnecessarily hard to grasp if the behavior structure is centralized.
* A centralized structure is appropriate:
* If the order in which the sub-event phases will be performed is likely to change.
* If you expect to insert new sub-event phases.
* If you want to keep parts of the functionality reusable as separate pieces.

## 3.5. Collaboration Diagram

A collaboration diagram for the Chatbot for Mental Health project describes the pattern of interaction among key system components. It illustrates how objects, in this context representing various modules and functionalities, participate in the interaction by depicting their links and the messages exchanged during the execution of specific use cases.

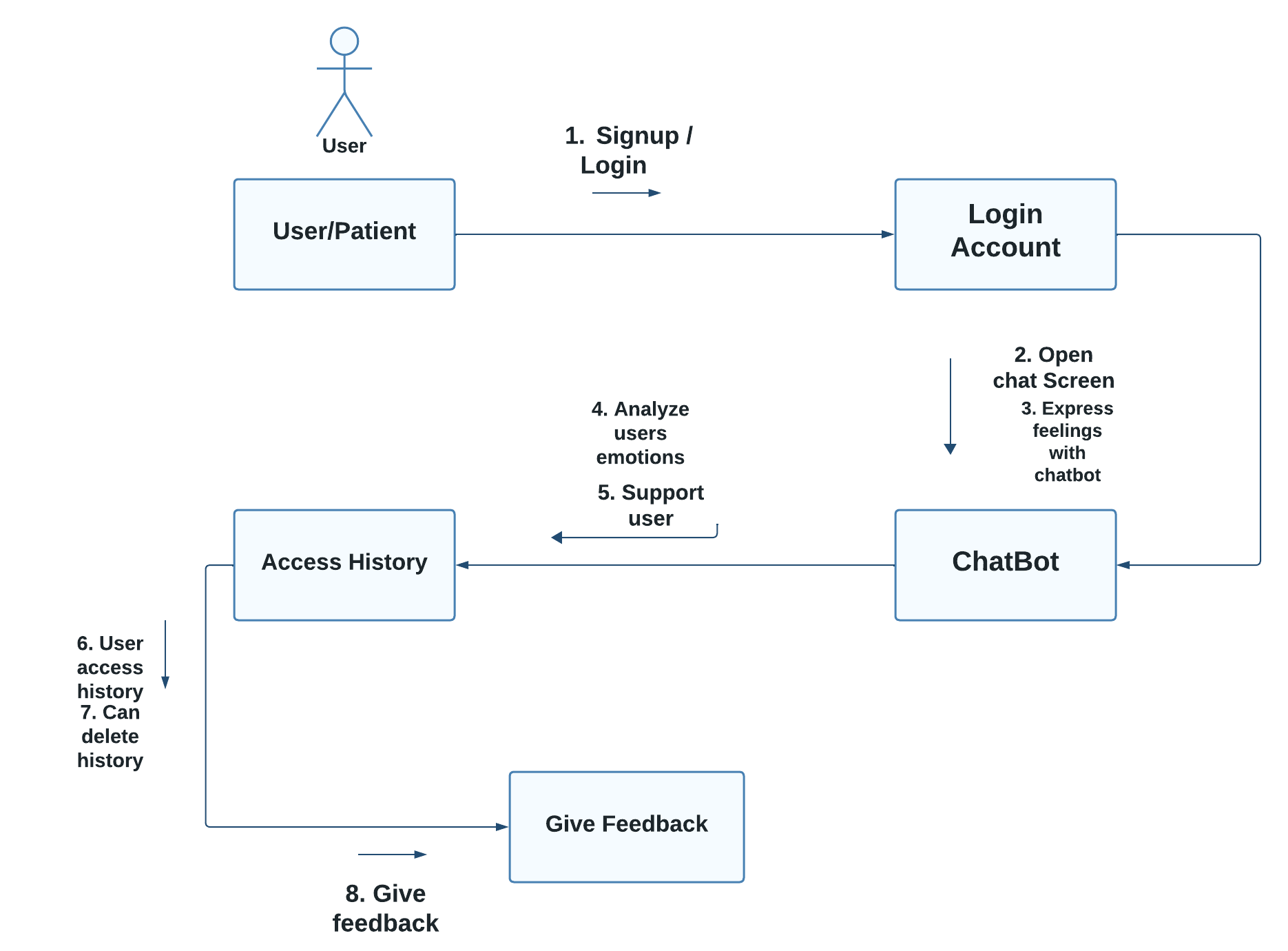
Collaboration diagrams play a crucial role in showcasing how different elements within the system collaborate to fulfill the behavior defined in a particular use case or a subset of it. Alongside sequence diagrams, collaborations serve as valuable tools for designers to elucidate the roles played by objects in executing a specific flow of events within a use case. These diagrams are instrumental in determining class responsibilities and interfaces during the design phase.

In comparison to sequence diagrams, collaboration diagrams focus on displaying the relationships among objects, making them particularly effective for comprehending the overall impact on a specific object and aiding in procedural design. While both sequence and collaboration diagrams convey similar information, collaboration diagrams excel in showcasing object relationships.

The Chatbot for Mental Health collaboration diagram, aligned with the project proposal, is especially valuable during analysis activities. It excels in illustrating simpler interactions involving a smaller number of objects, which is essential for understanding the core functionality of the chatbot. However, as the number of objects and messages increases, the diagram's readability may diminish. It is important to note that collaboration diagrams may encounter challenges in representing additional descriptive information such as timing, decision points, or other unstructured details that can be easily incorporated into the notes section of a sequence diagram.

### 3.5.1. Contents of Collaboration Diagrams

You can have objects and actor instances in collaboration diagrams, together with links and messages describing how they are related and how they interact. The diagram describes what takes place in the participating objects, in terms of how the objects communicate by sending messages to one another. You can make a collaboration diagram for each variant of a use case's flow of events.



**Collaboration Diagram**

### 3.5.2. Constructs of Collaboration Diagram:

Objects

In the context of the 'Chatbot for Mental Health' project, objects are depicted using object symbols. Each object symbol includes the name of the object and its class, presented with the object name followed by a colon and the underlined class name:

objectname: classname

These objects play a crucial role in the collaboration diagram, showcasing the interactions and relationships among system components. The class of an object may be unspecified initially, allowing for flexibility in the design process. Objects can also be unnamed, although providing distinct names is recommended for differentiation, especially when multiple objects belong to the same class. Additionally, if an object's class actively participates in the collaboration, it can be represented within the collaboration diagram.

The use of objects in collaboration diagrams is particularly beneficial during the initial stages of system design, allowing designers to visualize and analyze the interaction patterns before specifying the detailed class information. This aligns with the project's emphasis on developing a versatile and user-friendly chatbot for mental health support.

Actors

Actors play a pivotal role in the collaboration diagram for the Chatbot for Mental Health project. Actor instances are represented as the initiators or participants in the interaction, reflecting their involvement in the system's functionalities. In alignment with the project proposal, if multiple actor instances are present in the collaboration diagram, it is advisable to position them at the periphery, ensuring clarity and a focused representation of their roles within the context of the mental health chatbot system. This arrangement helps in emphasizing the centrality of the collaborative interactions among key system components, highlighting the essential role of actors in invoking and participating in these interactions.

Links

Links within the collaboration diagram are integral components that define relationships among key system elements, facilitating the exchange of messages. In the context of our project, a link is visually represented as a solid line connecting two objects, symbolizing the communication channel between them.

Objects, depicting various modules and functionalities, engage in interactions and navigations through these links. The connections established between objects enable the seamless exchange of messages crucial for the collaborative execution of specific use cases within the Chatbot for Mental Health system.

Each link may represent either an instance of an association, establishing a specific relationship between objects, or it can remain anonymous, signifying an unspecified association. This flexibility allows for a nuanced representation of object relationships within the collaboration diagram.

Messages

In our project Chatbot for Mental Health collaboration diagram, a message represents a communication exchange between various system components, symbolizing the flow of information with the anticipation that specific activities will follow. These messages are crucial for showcasing how different objects collaborate to achieve the desired behavior within the system.

In the collaboration diagrams designed for this project, a message is visually represented as a labeled arrow positioned in proximity to a link, indicating that the link serves as the conduit for delivering the message to the intended object. The arrow aligns with the direction of the target object, which is the recipient of the message. The label on the arrow includes the name of the message along with its relevant parameters. Additionally, a sequence number may be incorporated on the arrow to illustrate the chronological order of the message within the overall interaction. Sequence numbers play a vital role in these collaboration diagrams, providing a clear depiction of the relative sequencing of messages.

## 3.6. Operation Contracts

Operation contracts in the context of the "Chatbot for Mental Health" project serve to articulate the system state changes resulting from specific operations, as identified in the system sequence diagrams. These contracts define the expected behavior of each system operation and play a vital role in generating a comprehensive understanding of the system's functionality.

**Operation Contract Syntax:**

Name: Appropriate Name

Responsibilities: Execute a specific function

Cross References: System functions and Use Cases

Exceptions: None

Preconditions: Verify the existence of certain elements or relationships

Postconditions: Confirm changes in the system state, including the formation of associations

When formulating an operation contract, it is essential to consider the system's state both before and after the operation. This involves creating snapshots of the system at these two points in time. The operation contract should clearly outline the conditions preceding the action (preconditions) and the conditions following the action (postconditions). It's crucial to focus on describing the state of the system rather than detailing the actions or processes involved.

Typical postcondition changes may include:

* Modification of object attributes.
* Creation of an instance of an object.
* Formation or dissolution of an association.

Postconditions are articulated in the past tense, emphasizing the system's state changes resulting from the operation. To complete an operation contract, follow these steps: fill in the name, outline the responsibilities, and specify the postconditions.

**Example Operation Contract:**

Name: UserLogin

Responsibilities: Authenticate user credentials and grant access

Cross References: User authentication function, Login Use Case

Exceptions: None

Preconditions: User account exists in the system

Postconditions: User is granted access, session token is generated, and user status is updated to 'logged in.'

Operation contracts, aligned with the principles outlined in the project proposal, contribute to a clearer understanding of the Chatbot for Mental Health system's behavior and facilitate the development of a robust and effective mental health support mechanism.

## 3.7. Design Class Diagram

### In the design of the Chatbot for Mental Health system, classes play a pivotal role as the primary entities responsible for carrying out the essential functionalities. The design elements, including subsystems, packages, and collaborations, serve to illustrate the grouping of classes and their interoperation within the system.

### In the context of real-time systems, capsules, which are stereotyped classes, are employed to represent concurrent threads of execution. In our project, these 'capsules' are reflective of the active threads within the chatbot system, managing real-time interactions. Concurrent behavior is achieved through a combination of 'active' classes and 'passive' classes. While 'passive' classes are utilized within the execution context provided by the 'active' capsules, the design approach doesn't necessarily rely on capsules, offering flexibility in modeling concurrent behavior.

### Active classes within the "Chatbot for Mental Health" system are those that orchestrate and drive the behavior of the passive classes. An 'active' class, in this context, represents a class whose instances are deemed active objects, each possessing its own thread of control. These active objects collaborate to deliver immediate mental health support, responding to user inputs, engaging in conversations, and providing tailored assistance based on the identified mental health concerns.

### The design class diagram serves as a visual representation of the structure and relationships among classes within the system. It encapsulates the fundamental building blocks and their interactions, guiding the software architect and designer in formulating a coherent and effective system architecture. This diagram is instrumental in depicting how the 'Chatbot for Mental Health' system is organized, facilitating a clear understanding of the roles, responsibilities, and interactions among various classes.

### 3.7.1. Create Initial Design Classes

Commence the design phase by pinpointing one or more (initial) design classes derived from the domain model and establish trace dependencies. These design classes, formulated at this stage, will undergo refinement, adjustment, and potential merging or splitting in subsequent steps as they are endowed with various design attributes, including operations, methods, and a state machine, detailing the design of the analysis class. The creation of initial design classes is guided by the type of analysis class – whether it is a boundary, entity, or control class. Specific strategies tailored to each type are employed to generate these foundational design classes.

In the context of Chatbot for Mental Health, this process involves the identification of classes that represent crucial components within the system. These may include classes responsible for user interactions (boundary classes), core functionalities or data manipulation (entity classes), and control logic governing the flow of the chatbot's operations (control classes).

### 3.7.2. Designing Boundary Classes

### During the analysis phase, a fundamental principle is to establish one boundary class for each window or form within the user interface. This principle leads to boundary classes carrying responsibilities at a relatively high level, requiring further refinement and detailing in the subsequent design step.

### The design of boundary classes is contingent upon the available user interface development tools or GUI frameworks employed by the project. In contemporary technology, it is common for the user interface to be visually constructed directly within the development tool. This process automatically generates user interface classes that must be correlated with the design of control and/or entity classes. If the GUI development environment autonomously generates the necessary supporting classes for implementing the user interface, there is no need to explicitly consider them in the design; focus on designing elements not automatically created by the development environment.

### Supplementary input for this task can be derived from sketches or screen captures of an executable user interface prototype. Such artifacts may have been generated to further specify the requirements imposed on the boundary classes.

### Boundary classes representing interfaces to existing systems are often modeled as subsystems, given their potential for complex internal behavior. In instances where the interface behavior is straightforward, such as serving as a pass-through to an existing API, it may be appropriate to represent the interface using one or more design classes. If this approach is chosen, adhere to a single design class per protocol, interface, or API. Include any special requirements related to standards or protocols in the class's special requirements documentation.

### "3.7.3. Designing Entity Classes

### In the analysis phase, entity classes serve as the fundamental units of information manipulation, and the corresponding entity objects are typically passive and persistent. These entity classes, identified and linked with the analysis mechanism for persistence during the analysis phase, may undergo modifications during the design phase due to performance considerations. Such changes to the design model necessitate collaborative discussions between the Database Designer and the Designer.

### As outlined in the provided proposal, the design process involves careful consideration of the persistent classes, and adjustments may be required to enhance performance. The Database Designer and the Designer collaborate to analyze and refine the entity classes, ensuring that they meet both functional and performance-related requirements.

### This collaborative effort is vital for aligning the design model with the evolving needs of the project. As changes are introduced, it is essential to maintain synchronization between the design of entity classes and their representation in the database. This iterative process ensures that the final design is not only functionally robust but also optimized for performance.

### By fostering communication between the Database Designer and the Designer, the design phase becomes a dynamic and responsive stage in the software development life cycle. This approach allows for the incorporation of performance-driven adjustments to entity classes, ultimately contributing to the creation of a well-optimized and effective system.

### 3.7.4. Designing Control Classes

A control object is responsible for managing the flow of a use case and thus coordinates most of its actions; control objects encapsulate logic that is not particularly related to user interface issues (boundary objects), or to data engineering issues (entity objects). This logic is sometimes called application logic, or business logic.

Given this, at least the following issues need to be taken into consideration when control classes are designed:

Complexity:

Simple controlling or coordinating behavior can be handled by boundary and/or entity classes. As the complexity of the application grows, however, significant drawbacks to this approach surface:

* The use case coordinating behavior becomes imbedded in the UI, making it more difficult to change the system.
* The same UI cannot be used in different use case realizations without difficulty.
* The UI becomes burdened with additional functionality, degrading its performance.
* The entity objects may become burdened with use-case specific behavior, reducing their generality.

To avoid these problems, control classes are introduced to provide behavior related to coordinating flows-of-events

Change probability

If the probability of changing flows of events is low, or the cost is negligible, the extra expense and complexity of additional control classes may not be justified.

Distribution and performance

The need to run parts of the application on different nodes or in different process spaces introduces the need for specialization of design model elements. This specialization is often accomplished by adding control objects and distributing behavior from the boundary and entity classes onto the control classes. In doing this, the boundary classes migrate toward providing purely UI services, and the entity classes toward providing purely data services, with the control classes providing the rest.

Transaction management:

Managing transactions is a classic coordination activity. Absent a framework to handle transaction management, one would have one or more transaction manager classes which would interact to ensure that the integrity of transactions is maintained.

Note that in the latter two cases, if the control class represents a separate thread of control it may be more appropriate to use an active class to model the thread of control.

### 3.7.5. Identify Persistent Classes

Classes which need to be able to store their state on a permanent medium are referred to as 'persistent'. The need to store their state may be for permanent recording of class information, for back-up in case of system failure, or for exchange of information. A persistent class may have both persistent and transient instances; labeling a class 'persistent' means merely that some instances of the class may need to be persistent.

Identifying persistent classes serves to notify the Database Designer that the class requires special attention to its physical storage characteristics. It also notifies the Software Architect that the class needs to be persistent, and the Designer responsible for the persistence mechanism that instances of the class need to be made persistent.

Because of the need for a coordinated persistence strategy, the Database Designer is responsible for mapping persistent classes into the database, using a persistence framework. If the project is developing a persistence framework, the framework developer will also be responsible for understanding the persistence requirements of design classes. To provide these people with the information they need, it is sufficient at this point to simply indicate that the class (or more precisely, instances of the class) are persistent. Also incorporate any design mechanisms corresponding to persistency mechanisms found during analysis.

Example

The analysis mechanism for persistency might be realized by one of the following design mechanisms:

* In-memory storage
* Flash card
* Binary file
* Database Management System (DBMS)

depending on what is required by the class.

Note that persistent objects may not only be derived from entity classes; persistent objects may also be needed to handle non-functional requirements in general. Examples are persistent objects needed to maintain information relevant to process control, or to maintain state information between transactions.

### 3.7.6. Define Class Visibility

For each class, determine the class visibility within the package in which it resides. A 'public' class may be referenced outside the containing package. A 'private' class (or one whose visibility is 'implementation') may only be referenced by classes within the same package.

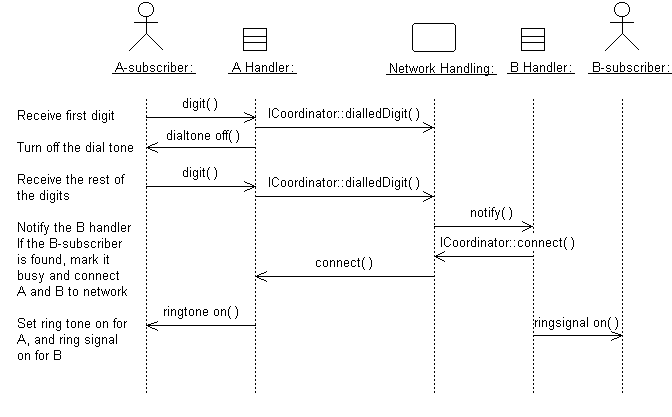
* Define Operations
* Identify Operations
* Name and Describe the Operations
* Define Operation Visibility
* Define Class Operations

Identify Operations

To identify Operations on design classes:

* Study the responsibilities of each corresponding analysis class, creating an operation for each responsibility. Use the description of the responsibility as the initial description of the operation.
* Study the use-case realizations in the class participates to see how the operations are used by the use-case realizations. Extend the operations, one use-case realization at the time, refining the operations, their descriptions, return types and parameters. Each use-case realization's requirements as regards classes are textually described in the Flow of Events of the use-case realization.
* Study the use case Special Requirements, to make sure that you do not miss implicit requirements on the operation that might be stated there.

Operations are required to support the messages that appear on sequence diagrams because scripts; messages (temporary message specifications) which have not yet been assigned to operations describe the behavior the class is expected to perform. An example sequence diagram is shown below:



Messages form the basis for identifying operations.

Do not define operations, which merely get and set the values of public attributes. These are generally generated by code generation facilities and do not need to be explicitly defined.

Name and Describe the Operations

The naming conventions of the implementation language should be used when naming operations, return types, and parameters and their types.

For each operation, you should define the following:

Operation name:

The name should be short and descriptive of the result the operation achieves.

The names of operations should follow the syntax of the implementation language. Example: find\_location would be acceptable for C++ or Visual Basic, but not for Smalltalk (in which underscores are not used); a better name for all would be findLocation.

Avoid names that imply how the operation is performed (example: Employee.wages() is better than Employee.calculateWages(), since the latter implies a calculation is performed. The operation may simply return a value in a database).

The name of an operation should clearly show its purpose. Avoid unspecific names, such as getData, that are not descriptive about the result they return. Use a name that shows exactly what is expected, such as getAddress. Better yet, simply let the operation name be the name of the property which is returned or set; if it has a parameter, it sets the property, if it has no parameter it gets the property. Example: the operation address returns the address of a Customer, while address(aString) sets or changes the address of the Customer. The 'get' and 'set' nature of the operation are implicit from the signature of the operation.

Operations that are conceptually the same should have the same name even if different classes define them, they are implemented in entirely different ways, or they have a different number of parameters. An operation that creates an object, for example, should have the same name in all classes.

If operations in several classes have the same signature, the operation must return the same kind of result, appropriate for the receiver object. This is an example of the concept of polymorphism, which says that different objects should respond to the same message in similar ways. Example: the operation name should return the name of the object, regardless how the name is stored or derived. Following this principle makes the model easier to understand.

The return type:

The return type should be the class of object that is returned by the operation.

A short description:

As meaningful as we try to make it, the name of the operation is often only vaguely useful in trying to understand what the operation does. Give the operation a short description consisting of a couple of sentences, written from the operation user's perspective.

The parameters. For each parameter, create a short descriptive name, decide on its class, and give it a brief description. As you specify parameters, remember that fewer parameters mean better reusability. A small number of parameters makes the operation easier to understand and hence there is a higher likelihood of finding similar operations. You may need to divide an operation with many parameters into several operations. The operation must be understandable to those who want to use it. The brief description should include the following:

* The meaning of the parameters (if not apparent from their names).
* Whether the parameter is passed by value or by reference
* Parameters which must have values supplied
* Parameters which can be optional, and their default values if no value is provided
* Valid ranges for parameters (if applicable)
* What is done in the operation.
* Which by reference parameters are changed by the operation.

Once you have defined the operations, complete the sequence diagrams with information about which operations are invoked for each message.

Define Operation Visibility

For each operation, identify the export visibility of the operation. The following choices exist:

* Public: the operation is visible to model elements other than the class itself.
* Implementation: the operation is visible only within to the class itself.
* Protected: the operation is visible only to the class itself, to its subclasses, or to friends of the class (language dependent)
* Private: the operation is only visible to the class itself and to friends of the class

Choose the most restricted visibility possible which can still accomplish the objectives of the operation. In order to do this, look at the sequence diagrams, and for each message determine whether the message is coming from a class outside the receiver's package (requires public visibility), from inside the package (requires implementation visibility), from a subclass (requires protected visibility) or from the class itself or a friend (requires private visibility).

Define Class Operations

For the most part, operations are 'instance' operations, that is, they are performed on instances of the class. In some cases, however, an operation applies to all instances of the class, and thus is a class-scope operation. The 'class' operation receiver is actually an instance of a metaclass, the description of the class itself, rather than any specific instance of the class. Examples of class operations include messages, which create (instantiate) new instances, which return all instances of a class, and so on.

To denote a class-scope operation, the operation string is underlined.

A method specifies the implementation of an operation. In many cases, methods are implemented directly in the programming language, in cases where the behavior required by the operation is sufficiently defined by the operation name, description and parameters. Where the implementation of an operation requires use of a specific algorithm, or requires more information than is presented in the operation's description, a separate method description is required. The method describes how the operation works, not just what it does.

* The method, if described, should discuss:
* How operations are to be implemented.
* How attributes are to be implemented and used to implement operations.
* How relationships are to be implemented and used to implement operations.

The requirements will naturally vary from case to case. However, the method specifications for a class should always state:

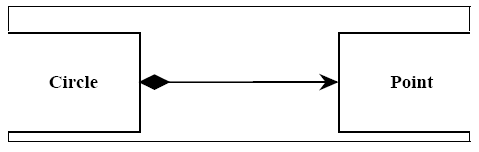
* What is to be done according to the requirements?
* What other objects and their operations are to be used?
* More specific requirements may concern:
* How parameters are to be implemented.
* Any special algorithms to be used.

Sequence diagrams are an important source for this. From these it is clear what operations are used in other objects when an operation is performed. A specification of what operations are to be used in other objects is necessary for the full implementation of an operation. The production of a complete method specification thus requires that you identify the operations for the objects involved and inspect the corresponding sequence diagrams.

### 3.7.7. Design Class Relationships

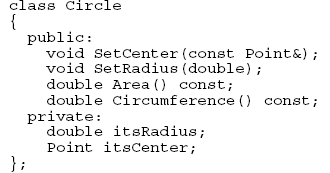
Composition Relationship

Each instance of type Circle seems to contain an instance of type Point. This is a relationship known as composition. It can be depicted in UML using a class relationship. Figure shows the composition relationship.



The black diamond represents composition. It is placed on the Circle class because it is the Circle that is composed of a Point. The arrowhead on the other end of the relationship denotes that the relationship is navigable in only one direction. That is, Point does not know about Circle. In UML relationships are presumed to be bidirectional unless the arrowhead is present to restrict them. Had I omitted the arrowhead, it would have meant that Point knew about Circle. At the code level, this would imply a #include “circle.h” within point.h. For this reason, I tend to use a lot of arrowheads. Composition relationships are a strong form of containment or aggregation. Aggregation is a whole/part relationship. In this case, Circle is the whole, and Point is part of Circle. How-ever, composition is more than just aggregation. Composition also indicates that the lifetime of Point is dependent upon Circle. This means that if Circle is destroyed, Point will be destroyed with it. For those of you who are familiar with the Booch-94 notation, this is the Has-by-value relationship.

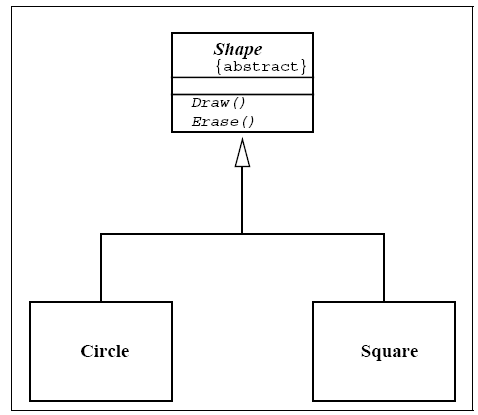
In C++ we would represent this as shown in Listing 1. In this case we have represented the composition relationship as a member variable. We could also have used a pointer so long as the destructor of Circle deleted the pointer.



**Inheritance Relationship**

A peculiar triangular arrowhead depicts the inheritance relationship in UML. This arrowhead, that looks rather like a slice of pizza, points to the base class. One or more lines proceed from the base of the arrowhead connecting it to the derived classes. Figure shows the form of the inheritance relationship. In this diagram we see that Circle and

Square both derive from Shape. Note that the name of class Shape is shown in italics. This indicates that Shape is an abstract class. Note also that the operations, Draw () and Erase () are also shown in italics. This indicates that they are pure virtual.



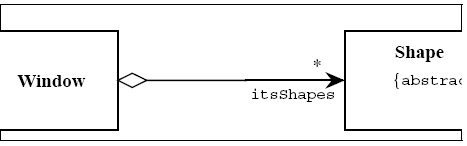
Italics are not always very easy to see. Therefore, as shown in Figure, an abstract class can also be marked with the {abstract} property. What’s more, though it is not a standard part of UML, I will often write Draw()=0 in the operations compartment to denote a pure virtual function.

**Aggregation / Association**

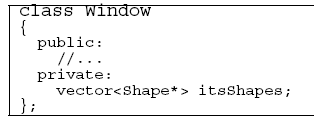
The weak form of aggregation is denoted with an open diamond. This relationship denotes that the aggregate class (the class with the white diamond touching it) is in some way the “whole”, and the other class in the relationship is somehow “part” of that whole.

Figure shows an aggregation relationship. In this case, the Window class contains many

Shape instances. In UML the ends of a relationship are referred to as its “roles”. Notice that the role at the Shape end of the aggregation is marked with a “ \*”. This indicates that the Window contains many Shape instances. Notice also that the role has been named. This is the name that Window knows its Shape instances by. i.e. it is the name of the instance variable within Window that holds all the Shapes.



Above figure might be implemented in C++ code as under:



There are other forms of containment that do not have whole / part implications. For example, each window refers back to its parent Frame. This is not aggregation since it is not reasonable to consider a parent Frame to be part of a child Window. We use the association relationship to depict this.

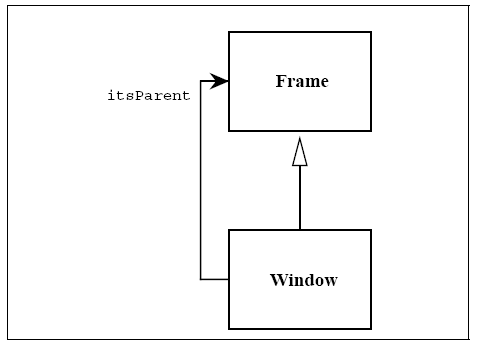


Figure shows how we draw an association. An association is nothing but a line drawn between the participating classes. In Figure 6 the association has an arrowhead to denote that Frame does not know anything about Window. Once again note the name on the role. This relationship will almost certainly be implemented with a pointer of some kind.

What is the difference between an aggregation and an association? The difference is one of implication. Aggregation denotes whole/part relationships whereas associations do not. However, there is not likely to be much difference in the way that the two relationships are implemented. That is, it would be very difficult to look at the code and determine whether a particular relationship ought to be aggregation or association. For this reason, it is pretty safe to ignore the aggregation relationship altogether. As the amigos said in the UML 0.8 document: “...if you don’t understand [aggregation] don’t use it.” Aggregation and Association both correspond to the Has-by-reference relationship from the Booch-94 notation.

Dependency

Sometimes the relationship between a two classes is very weak. They are not implemented with member variables at all. Rather they might be implemented as member function arguments. Consider, for example, the Draw function of the Shape class. Suppose that this function takes an argument of type Drawing Context.

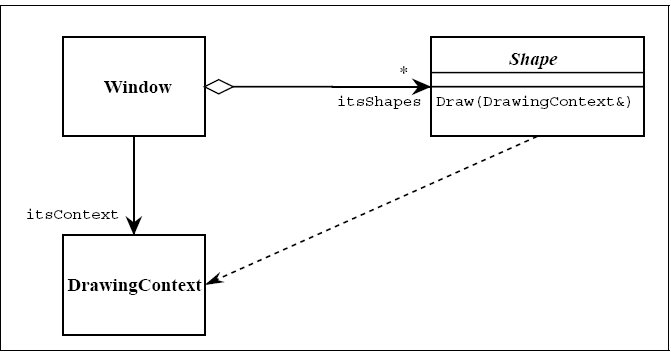
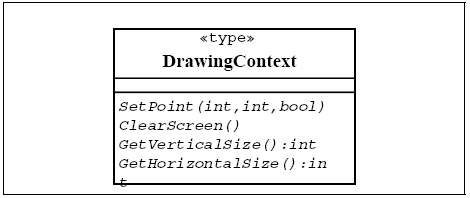
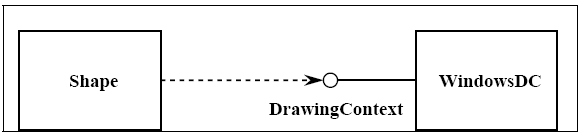


Figure shows a dashed arrow between the Shape class and the DrawingContext class. This is the dependency relationship. In Booch94 this was called a ‘using’ relationship. This relationship simply means that Shape somehow depends upon DrawingContext. In C++ this almost always results in a #include.

**Interfaces**

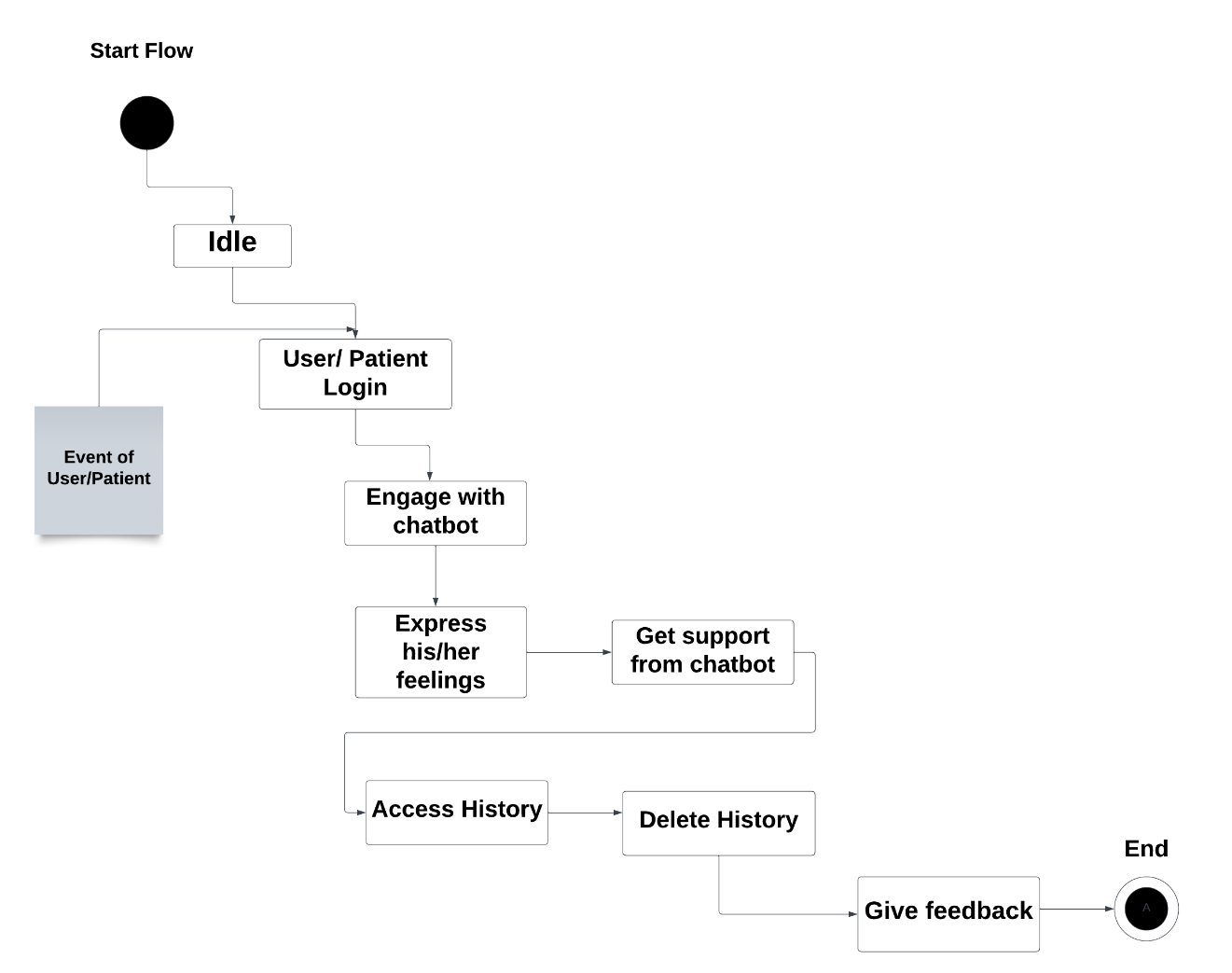
There are classes that have nothing but pure virtual functions. In Java such entities are not classes at all; they are a special language element called an interface. UML has followed the Java example and has created some special syntactic elements for such entities. The primary icon for an interface is just like a class except that it has a special denotation called a stereotype. Figure shows this icon. Note the «type» string at the top of the class. The two surrounding characters “«»” are called guillemots (pronounced Gee-**may**). A word or phrase surrounded by guillemots is called a “stereotype”. Stereotypes are one of the mechanisms that can be used to extend UML. When a stereotype is used above the name of a class it indicates that this class is a special kind of class that conforms to a rather rigid specification. The «type» stereotype indicates that the class is an interface. This means that it has no member variables, and that all of its member functions are pure virtual. UML supplies a shortcut for «type» classes. Figure 9 shows how the “lollypop” notation can be used to represent an interface. Notice that the dependency between Shape and DrawingContext is shown as usual. The class WindowsDC is derived from, or conforms to, the Drawingcontext interface. This is a shorthand notation for an inheritance relationship between





## 3.8. State chart diagram

For some operations, the behavior of the operation depends upon the state the receiver object is in. A state machine is a tool for describing the states the object can assume and the events that cause the object to move from one state to another. State machines are most useful for describing active classes. The use of state machines is particularly important for defining the behavior. An example of a simple state machine is shown below:



Each state transition event can be associated with an operation. Depending on the object's state, the operation may have a different behavior; the transition events describe how this occurs.

The method description for the associated operation should be updated with the state-specific information, indicating, for each relevant state, what the operation should do. States are often represented using attributes; the state chart diagrams serve as input into the attribute identification step.

## 3.9. Data Model

The data model is a subset of the implementation model, which describes the logical and physical representation of persistent data in the system.

**The Relational Data Model**

The relational model is composed of entities and relations. An entity may be a physical table or a logical projection of several tables also known as a view. The figure below illustrates LINEITEM and PRODUCT tables and the various relationships between them.

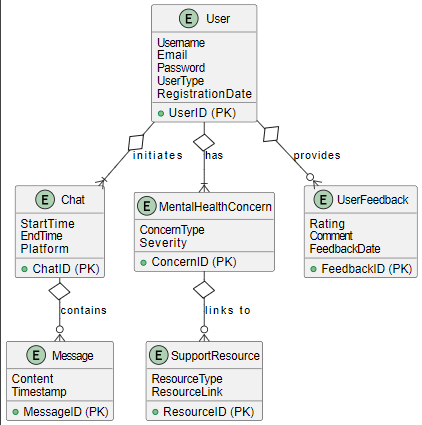
A relational model has the following elements:

An entity has columns. A name and a type identify each column. In the figure above, the LINEITEM entity has the columns LineItem\_Id (the primary key), Description, Price, Quantity, Product\_Id and Order\_Id (the latter two are foreign keys that link the LINEITEM entity to the ORDER and PRODUCT entities).

An entity has records or rows. Each row represents a unique set of information, which typically represents an object's persistent data. Each entity has one or more primary keys. The primary keys uniquely identify each record (for example, Id is the primary key for LINEITEM table).

Support for relations is vendor specific. The example illustrates the logical model and the relation between the PRODUCT and LINEITEM tables. In the physical model relations are typically implemented using foreign key / primary key references. If one entity relates to another, it will contain columns, which are foreign keys. Foreign key columns contain data, which can relate specific records in the entity to the related entity.

Relations have multiplicity (also known as cardinality). Common cardinalities are one to one (1:1), one to many (1:m), many to one (m:1), and many to many (m:n). In the example, LINEITEM has a 1:1 relationship with PRODUCT and PRODUCT has a 0:m relationship with LINEITEM.



**ERD diagram**

The "User" entity, a cornerstone of the system, initiates meaningful interactions by engaging in Chat sessions. Each chat session, denoted by the "Chat" entity, contains crucial messages that form the backbone of communication, establishing a dynamic flow of information. Meanwhile, the "User" entity, driven by a commitment to provide valuable insights, provides critical User Feedback that serves as a compass for system improvement. At the heart of the user experience, the "User" entity holds significant Mental Health Concerns, emphasizing the system's commitment to addressing diverse and challenging emotional states. These concerns, captured by the "Mental Health Concern" entity, act as a catalyst for the system to link to essential Support Resources. This linkage enhances the accessibility of resources and fortifies the system's arsenal against mental health challenges.

**Dataset:**