Software Requirement Specification

For

Team 2

Andrew Hanson, Lorrin Smith, Samuel Asbury, Camryn Chamberlin, and Spencer Higgins

CSCI-C308-17760

October 2025

Table of Contents

1.0 Introduction

1.1 Goals and Objectives

The purpose of TheraBot is to provide a realistic and educational simulation platform where psychology students and mental-health trainees can practice conducting diagnostic interviews in a safe and interactive environment. Unlike traditional chatbots, which position the user as a client, TheraBot inverts the model by making the human the therapist and the AI the simulated patient.

The objectives are the following:

- Improve diagnostic reasoning and empathy-based communication skills.
- Offer accessible, web-based practice for clinical training programs.
- Ensure data privacy and maintain a non-clinical educational context.

1.2 Statement of Scope

TheraBot is a web application that allows a user to log in as a therapist and interact with an AI-driven client. The AI presents realistic emotional and behavioral responses. Users can engage in structured dialogue, note symptoms, and propose a preliminary diagnosis.

The system supports three core modules:

1. **Therapy Session Module:** Interactive dialogue between therapist and AI client.
2. **Scenario Builder Module:** Admins can design new cases, symptoms, and diagnostic conditions.
3. **Feedback and Session Review Module:** Provides summaries, learning outcomes, and stored history for analysis.

Inputs include therapist messages, dialogue selections, and diagnosis submissions. Processing involves AI-based dialogue branching, condition-matching, and data storage. Outputs include chat logs, performance summaries, and suggested diagnoses.

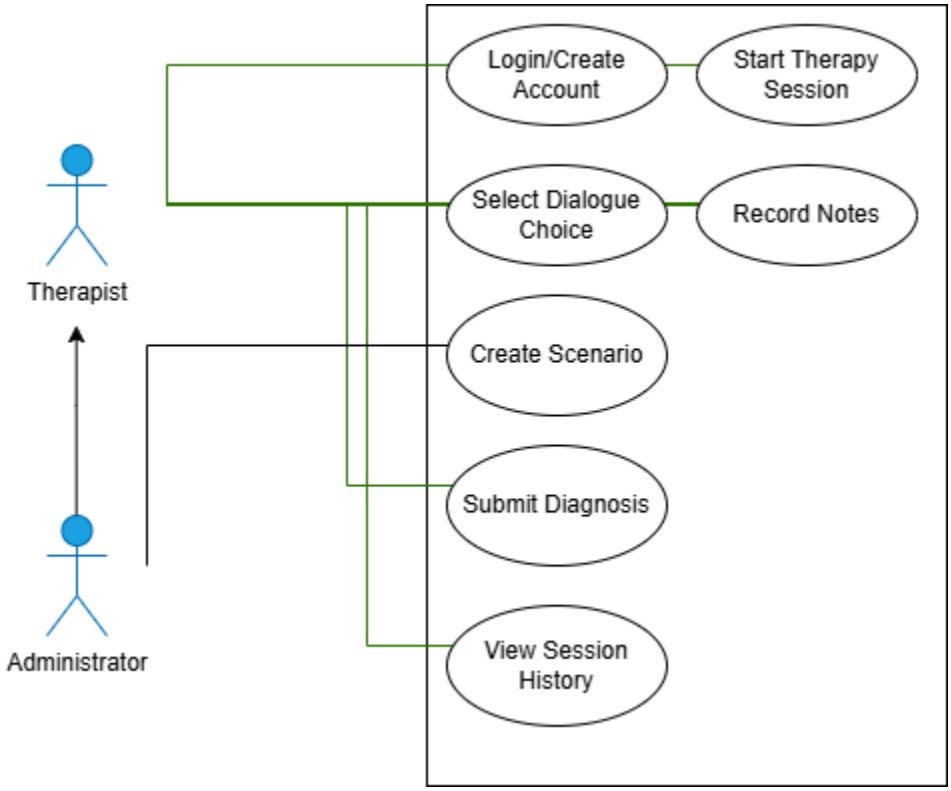
2.0 Function Description

2.1 User Profiles

TheraBot supports three user roles:

- **Therapist User**: Can log in, start sessions, view chat history, and propose diagnoses.
- **Administrator**: Can create and manage therapy scenarios, define dialogue trees, and update symptom databases.
- **Guest** (Future Feature): May access limited demo scenarios without data storage.

2.2 Use-Case Diagram



2.3 Use-Case Scenarios

2.3.1 Scenario 1: Login

This scenario is the first required step for interacting with TheraBot. A user navigates to the TheraBot web page and provides a valid username and password. Once authenticated, a session is created and a variable is set identifying the user's access level (Therapist or

Administrator). Successful login grants access to the system's dashboard where users can begin or manage therapy sessions.

2.3.2 Scenario 2: Start Therapy Session

This scenario allows a therapist to begin an interactive diagnostic session with the simulated AI client. After login, the therapist selects a case from the list of available scenarios. The system loads the chosen scenario and presents the client's opening dialogue. During the session, all messages are recorded in the database for future review. Each session is timestamped and linked to the therapist's user profile. Each therapist user can record notes throughout every session for documentation and review.

2.3.3 Scenario 3: Pick Dialogue Choice

When the simulated client presents a statement, the therapist must choose an appropriate response from a list of predefined options. Each choice corresponds to a new dialogue node, creating a branching interaction. The software records which response was chosen, updates the conversation state, and generates the client's next reply accordingly. This allows a realistic flow of conversation where decisions influence the direction and depth of the session.

2.3.4 Scenario 4: Create Scenario

This scenario is available only to users with Administrator privileges. Administrators can build new diagnostic simulations for educational purposes. They enter scenario details such as the client's background, presenting symptoms, and correct diagnosis. Dialogue nodes and therapist choices are added to shape the conversational flow. Once saved, the scenario becomes available to therapist users for practice sessions.

2.3.5 Scenario 5: Add Dialogue Node

Within the Scenario Builder, an Administrator can add new dialogue nodes to existing scenarios. Each node contains text that the AI client will say and a list of possible therapist responses. Administrators can link nodes together to form branching dialogue paths, creating more complex and realistic client interactions.

2.3.6 Scenario 6: Create Symptom

This scenario allows an Administrator to define a new symptom entity in the database. Information such as symptom name, severity, frequency, and duration are entered through a form. These symptoms can later be associated with conditions or diagnoses within the system to enrich the realism and educational accuracy of the AI simulations.

2.3.7 Scenario 7: Create Condition

Administrators can create new psychological conditions composed of one or more symptom entities. Each condition is named and described in detail, and linked to relevant symptoms. Conditions are used within diagnostic scenarios to represent potential or target diagnoses that therapist users must identify during sessions.

2.3.8 Scenario 8: Create Diagnosis

This scenario allows the Administrator to define a complete diagnosis record that combines one condition with its associated symptoms. Each diagnosis entity is linked to specific scenarios to serve as the “correct” outcome for the therapist to determine. When a therapist finishes a session, their chosen diagnosis is compared with this correct record to provide feedback.

2.3.9 Scenario 9: View Session Feedback

After a therapy session concludes, the system generates a feedback summary for the therapist. This includes session notes, message transcripts, and a comparison between the therapist’s submitted diagnosis and the expected result. The feedback assists in self-evaluation and helps instructors assess students’ diagnostic reasoning.

3.0 Data Description

The TheraBot system will rely on a layered data model. Entities represent persistent records in the database and repositories manage the low-level CRUD operations. Services contain the application’s business logic and connect entities and repositories. This modular approach ensures maintainability and scalability.

We have 3 types of data objects:

1. **Repository** - Repositories are used to interact with the database layer through the service type data objects. It contains the low-level functions that are needed to interact with the database. All repository classes will extend a generic Repository class given by an external ORM library that will give basic CRUD methods.
2. **Service** - Services are used to interact with the repository and entity data objects. It contains the business logic functions that allow more complex interactions between entities and other services.
3. **Entity** - Entities are strictly used to represent database records as data objects. The service and repository classes handle complex interactions that can’t be done. It contains attributes, their types, and their setter/getter functions.

3.1 Data Objects

- **CopingMechanismRepository**: Contains all database operations for CopingMechanismEntity. It's responsible for saving, retrieving, and deleting records of different coping strategies

- **CopingMechanismService**: Contains all business logic related to coping mechanisms. Uses CopingMechanismRepository to fetch data and perform operations, such as finding all coping mechanisms or a specific coping mechanism.

- **CopingMechanismEntity**: Represents a single coping mechanism in the database. It holds the data for a specific coping mechanism. An array of CopingMechanismEntities is used when making a SymptomEntity. Its data is only a name.

- **MoodRepository**: Contains all database operations for MoodEntity. It's responsible for saving, retrieving, and deleting records of different moods.

- **MoodService**: Contains all business logic related to moods. Uses MoodRepository to fetch data and perform operations, such as finding all moods or a specific mood.

- **MoodEntity**: Represents a single mood in the database. It holds the data for a specific mood. An array of MoodEntities is used when making a SymptomEntity. Its data is only a name.

- **TriggerRepository**: Contains all database operations for TriggerEntity. It's responsible for saving, retrieving, and deleting records of different triggers.

- **TriggerService**: Contains all business logic related to triggers. Uses TriggerRepository to fetch data and perform operations, such as finding all triggers or a specific trigger.

- **TriggerEntity**: Represents a single trigger in the database. It holds the data for a specific trigger. An array of TriggerEntities is used when making a SymptomEntity. Its data is only a name.

- **SymptomRepository**: Contains all database operations for SymptomEntity. It's responsible for saving, retrieving, and deleting records of different symptoms.

- **SymptomService**: Contains all business logic related to symptoms. Uses SymptomRepository to fetch data and perform operations, such as finding all symptoms or a specific symptom.

- **SymptomEntity**: Represents a single symptom in the database. It holds the data for a specific symptom. Each symptom entity contains data like the name, severity (on a scale of 1 to 10 with 1 being the lowest and 10 being the highest severity), frequency (on a scale of 1 to 5 with 1 being once a month and 5 being every day), duration (on a scale of 1 to 5 with 1 being a short duration of 30 - 60 minutes a day and 5 being a prolonged duration of a few hours a day), triggers, coping mechanisms, moods, and life impact (on a scale of 1 to 10 with 1 being the lowest and 10 being the highest). It's used when creating a specific DiagnosisEntity.

- **ConditionRepository**: Contains all database operations for ConditionEntity. It's responsible for saving, retrieving, and deleting records of different conditions.

- **ConditionService**: Contains all business logic related to conditions. Uses ConditionRepository to fetch data and perform operations, such as finding all conditions or a specific condition.

- **ConditionEntity**: Represents a single condition in the database. It holds the data for a specific condition. Each condition entity contains data like the name and an array of SymptomEntities. It's used when creating a specific DiagnosisEntity.

- **DiagnosisRepository**: Contains all database operations for DiagnosisEntity. It's responsible for saving, retrieving, and deleting records of different diagnoses.

- **DiagnosisService**: Contains all business logic related to diagnoses. Uses DiagnosisRepository to fetch data and perform operations, such as finding all diagnoses, a specific diagnosis, or comparing a therapist diagnosis to a correct diagnosis.

- **DiagnosisEntity**: Represents a single diagnosis in the database. It holds the data for a specific diagnosis. Each diagnosis entity contains a specific ConditionEntity and an array of SymptomEntities. It's used when creating scenarios in the ScenarioService to indicate what the correct diagnosis is for a given scenario.

- **TherapistRepository**: Contains all database operations for TherapistEntity. It's responsible for saving, retrieving, and deleting records of different therapist users.

- **TherapistService**: Contains all business logic related to therapist users. Uses TherapistRepository to fetch data and perform operations, such as finding a therapist by an id, username, or email. Handles the user's registration, login, logout, and hashing of their password when creating their account.

- **TherapistEntity**: Represents a single therapist in the database. It holds the data for a specific therapist user. Each therapist entity contains a username, a password hash, whether they're an admin, and any therapy sessions they have ongoing or have finished previously.

- **ScenarioRepository**: Contains all database operations for ScenarioEntity. It's responsible for saving, retrieving, and deleting records of different scenarios.

- **ScenarioService**: Contains all business logic related to creating ScenarioEntities as an administrator. Allows administrators to CRUD ScenarioEntities including the specific dialogue nodes in a scenario. Uses ScenarioRepository to fetch data and perform operations, such as finding all scenarios, or a specific scenario.

- **ScenarioEntity**: Represents a single scenario in the database. It holds the data for a specific scenario. Each scenario entity contains a name, description, correct diagnosis entity, and a root dialogue node.

- **DialogueNodeEntity**: Represents a single dialogue node in the database. It holds the data for a specific dialogue node. Each dialogue node entity contains the bot's text for the node and the different therapist choices that a therapist can respond to the bot with.

- **TherapistChoiceEntity**: Represents a single therapist choice in the database. It holds the data for a specific therapist choice. Each therapist choice entity contains the text for the therapist choice, a source dialogue node which is basically the dialogue node that led to this therapist choice, and the next dialogue node which is basically the next dialogue node that this therapist choice leads to. Each choice can branch to a new dialogue node or an existing dialogue node, creating a tree-type structure of nodes.

- **MessageRepository**: Contains all database operations for MessageEntity. It's responsible for saving messages in a TherapySessionEntity via the TherapySessionService.

- **TherapistChoiceRepository**: Contains all database operations for TherapistChoiceEntity. It's responsible for saving the therapist's choices that were made in a TherapySessionEntity via the TherapySessionService.

- **DialogueNodeRepository**: Contains all database operations for DialogueNodeEntity. It's responsible for saving the different dialogue nodes that were picked by a Therapist in a scenario via the TherapySessionService.

- **TherapySessionRepository**:Contains all database operations for TherapySessionEntity. It's responsible for saving a therapy session that a Therapist is in via the TherapySessionService.

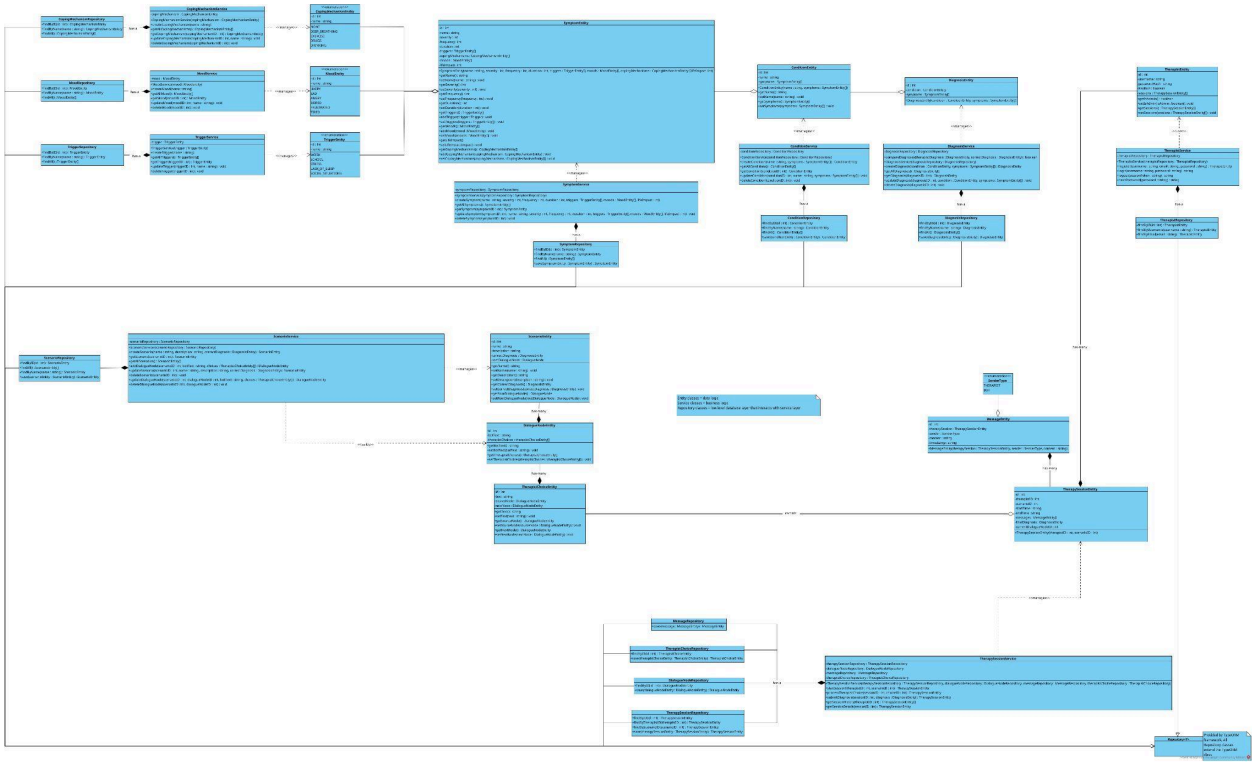
- **TherapySessionService**: Contains all business logic related to therapy sessions. Uses MessageRepository, TherapistChoiceRepository, DialogueNodeRepository, and TherapySessionRepository to fetch data and perform operations related to the TherapySessionEntity. It's used to start therapy sessions, process the therapist's choices, submit a final diagnosis, get session history, and get specific session details.

- **TherapySessionEntity**: Represents a single therapy session in the database. It holds the data for a specific therapy session. Each therapy session entity contains the therapist id, scenario id, start time, end time, the messages that were sent between the therapist and bot(only the strings, not the entire node ids are saved), the final diagnosis(if there is one yet), and the current dialogue node id so that they can start where they left off if they leave a therapy session early.

- **MessageEntity**: Represents a single message in a therapy session in the database. It holds the data for a specific message in a specific therapy session. Each message entity contains the therapy session id, who sent it, the contents, and the timestamp. This allows the therapist to view previous session history as all of its interactions and timestamps are saved.

- **SenderType**: Represents the different types of senders, with the sender either being a THERAPIST or BOT.

3.2 Data Relationships



4.0 Software Interface Description

4.1 External Machine Interfaces

TheraBot requires a web server environment for deployment. It interfaces with hosting services such as AWS or Heroku and connects to a MySQL or Firebase database.

4.2 External System Interfaces

Optional integration with open-source NLP or chatbot frameworks (e.g., TensorFlow, HuggingFace) may be supported in later versions to improve response realism.

4.3 Human Interface

The user interface follows modern web standards with accessibility compliance (WCAG). Navigation uses a sidebar for session control, and the main view displays interactive dialogue, note panels, and feedback areas.

5.0 Description for Software Behavior

TheraBot's behavior is defined by the interactions between the therapist user, the AI-driven client, and the system's backend services. The following overview describes how the system operates during a typical user session and how internal processes maintain functionality and data integrity.

5.1 Overall Behavior

The system follows a cyclical behavioral flow of authentication, interaction, and feedback.

1. **User Authentication**: A therapist or administrator logs in through the secure web interface. Credentials are verified using the TherapistService, and a session token is created for continued access.
2. **Scenario Initialization**: Once logged in, the user selects a scenario from the available library. The selected ScenarioEntity and its root DialogueNodeEntity are loaded from the database.
3. **Interactive Session**: The system alternates between therapist input and AI responses. Each therapist choice triggers the TherapySessionService to update the current node and retrieve the corresponding DialogueNodeEntity for the AI's next message.
4. **Session Recording**: All exchanges are recorded in the TherapySessionEntity, which stores message history, timestamps, and current progress. This allows the therapist to exit and later resume the session without losing context.
5. **Feedback Generation**: When the session concludes, the system compares the therapist's diagnosis with the expected DiagnosisEntity defined in the scenario. A feedback summary is generated that includes diagnostic accuracy, suggested improvements, and overall performance.
6. **Administrator Management**: Administrators can create, edit, or delete scenarios, conditions, and symptoms. They can also review therapist session logs to assess learning outcomes and ensure the system remains accurate and up to date.

5.2 Behavioral Logic

Each system action triggers a service-repository-entity chain:

- **Repositories** handle raw data storage and retrieval.
- **Services** manage business logic, validation, and scenario branching.
- **Entities** represent stored data and ensure structural consistency across features.

Error handling ensures users receive friendly feedback only if invalid inputs occur (such as a failed login, missing scenario data, or network loss). All behavioral logic gets executed

through asynchronous server calls, ensuring smooth user experience without full-page reloads.

5.3 Concurrency and Data Flow

Multiple therapists can interact with TheraBot simultaneously. Each user's actions are sandboxed within their own session to prevent data overlap. Session data is periodically saved to the database to protect against data loss during unexpected disconnections.

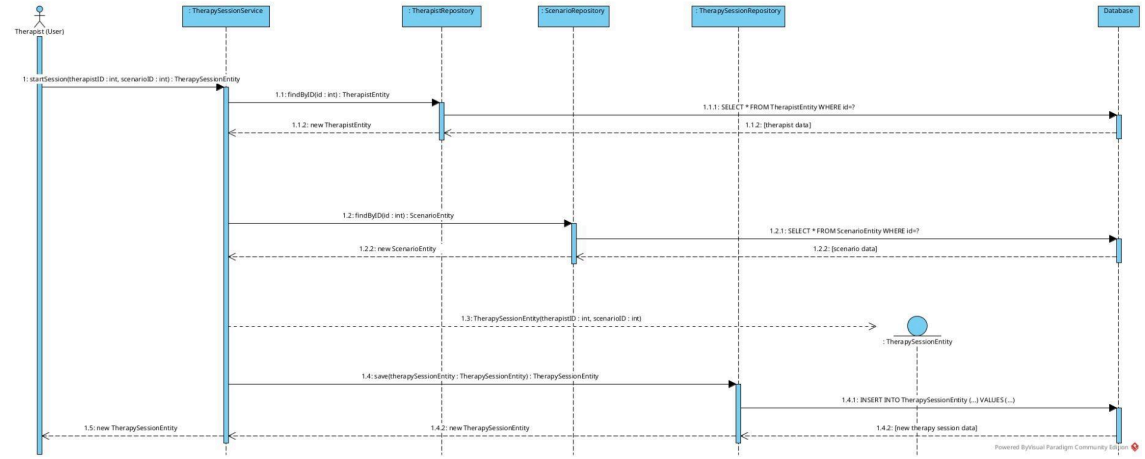
5.4 Termination and Data Persistence

When the user logs out or completes a session:

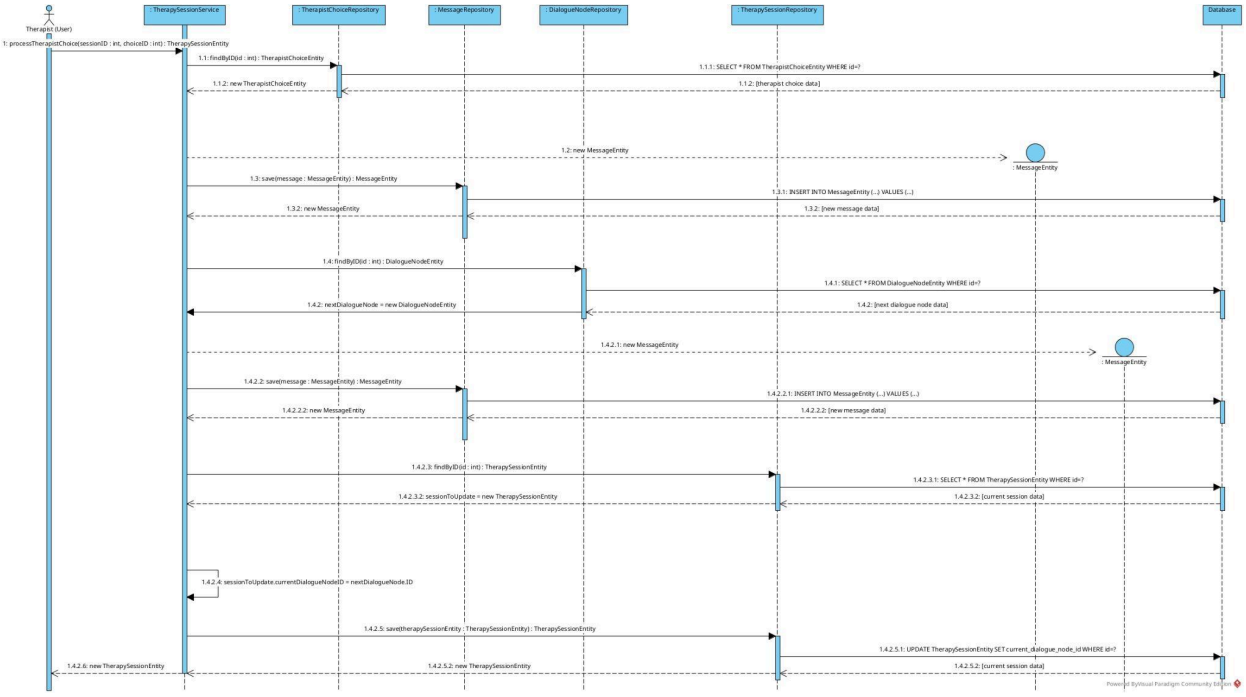
- The session state is stored permanently for later retrieval.
- Feedback results are compiled for academic review.
- The user is redirected to the dashboard or login screen depending on their role.

6.0 Use-Case Realization

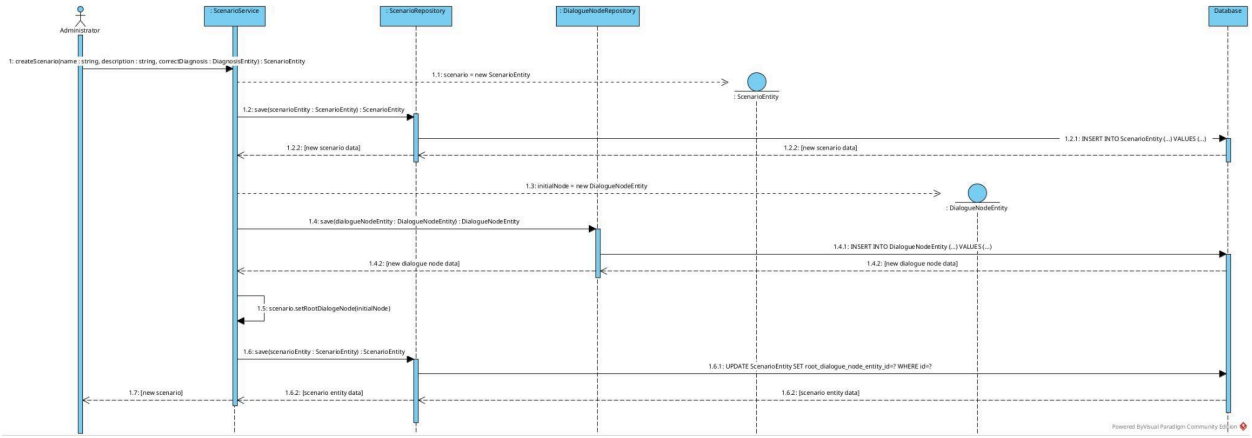
6.1 Realization 1 - Start Therapy Session



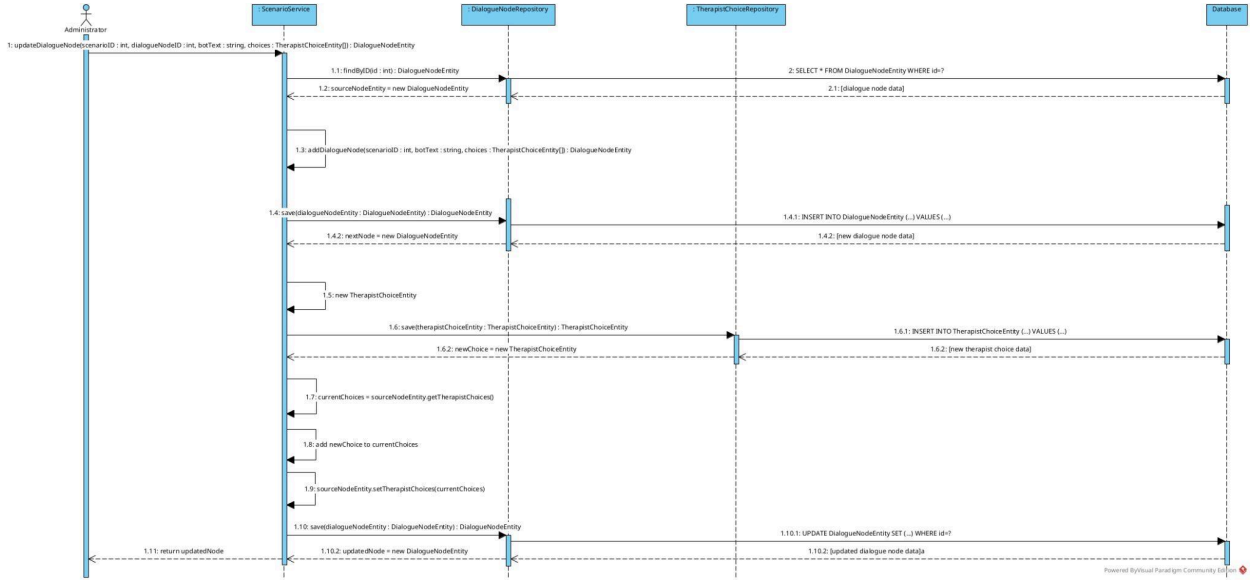
6.2 Realization 2 - Pick Dialogue Choice in Therapy Session



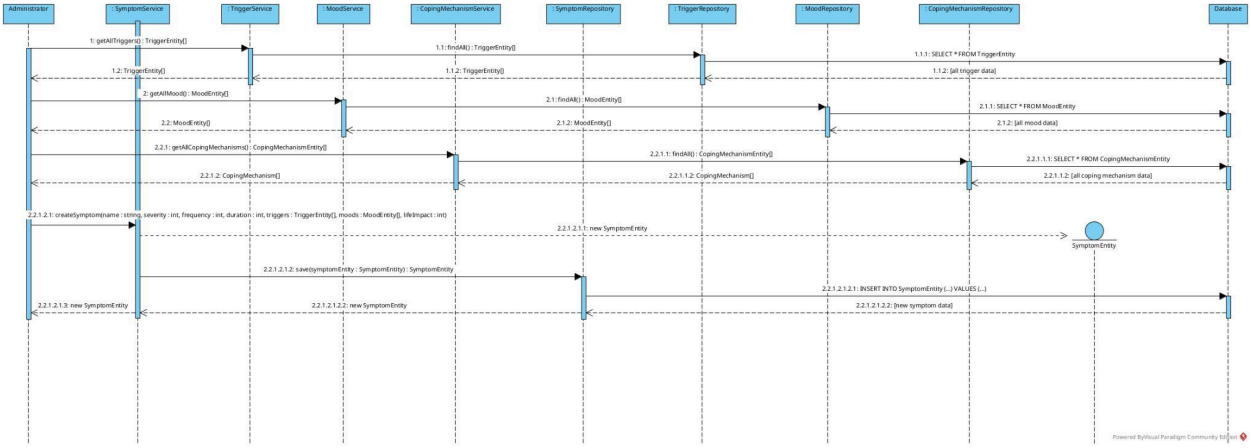
6.3 Realization 3 - Scenario Creation



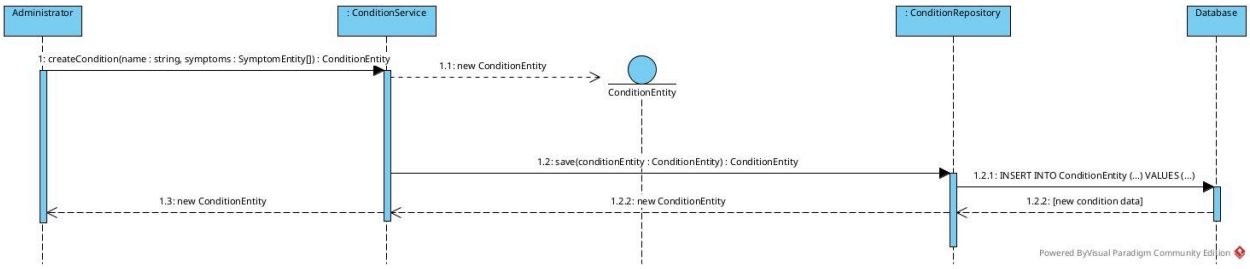
6.4 Realization 4 - Add Dialogue Node to Scenario



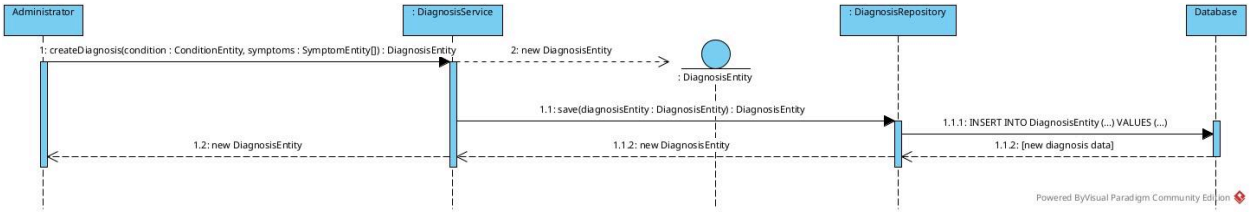
6.5 Realization 5 - Create Symptom



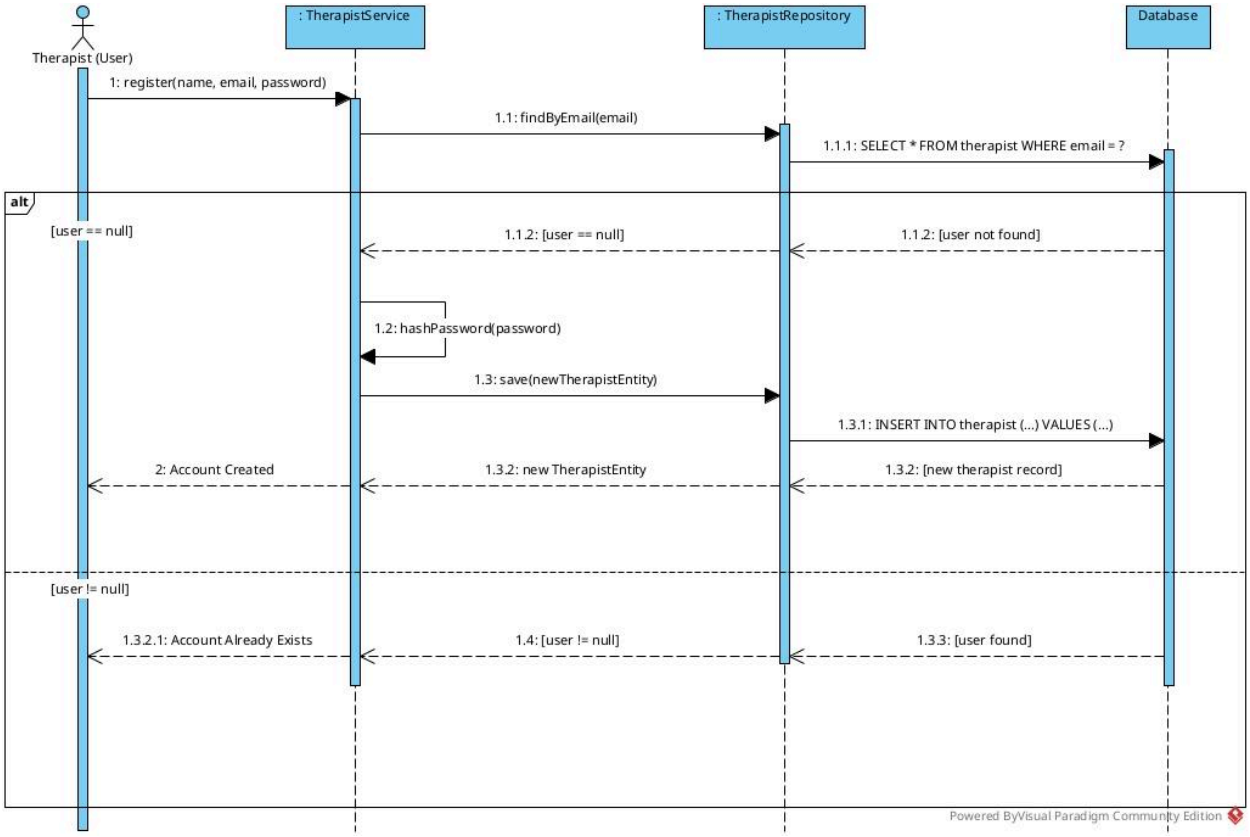
6.6 Realization 6 - Create Condition



6.7 Realization 7 - Create Diagnosis



6.8 Realization 8 - Account Creation



7.0 Restrictions, Limitations, and Other Issues

7.1 Restrictions

TheraBot is for **educational** use only, not for real therapeutic application.
AI responses are **limited** to pre-defined dialogue trees, with deep NLP understanding being part of a future enhancement.

7.2 Limitations

Requires internet access and a modern browser.
Performance may depend on server response times and database load.
Data stored is **non-sensitive** and limited to session logs for academic feedback.

7.3 Other

No other known issues at this time.

8.0 Appendices

Appendix A - Technical Stack

- **Frontend:** HTML, Tailwind CSS, JavaScript/TypeScript
- **Backend:** Node.js or Python Flask/Django
- **Database:** MySQL or Firebase for storing sessions and scenarios
- **Bot Engine:** Rule-based scripts + optional NLP integration

Appendix B - Future Enhancements

- Natural Language Processing (NLP): Integrate advanced language models to enable more adaptive and context-aware responses from the AI client.
- Analytics Dashboard: Develop a visual feedback panel that provides therapists with performance metrics, session summaries, and learning insights.
- Multi-User Support: Allow instructors to review multiple therapist accounts and evaluate session progress from a central portal.
- Session Playback: Introduce a feature to replay past sessions for self-assessment and training review.
- Mobile and Tablet Accessibility: Optimize the interface for mobile devices to support on-the-go training and practice.
- Security Enhancements: Add role-based access controls and encryption for user data to ensure confidentiality and compliance with FERPA or HIPAA standards.