Helwan University
Faculty of Computer Science and Artificial Intelligence
Computer Science Department

AI Mirror

A graduation project dissertation by:

Anas Saleh Mousa AlSadiq	(20210192)
Yara Mohammed Attia	(20211041)
Mennatullah Essam Abd ElAziz	(20210959)
Mahmoud Nasr Abd ElAziz	(20210883)
Mohamed Ashraf Abo El Maaty	(20210742)
Mohamed Ihab Mohamed Farid	(20210753)

Submitted in partial fulfilment of the requirements for the degree of Bachelor of Science in Computers & Artificial Intelligence, at the Computer Science Department, the Faculty of Computer Science & Artificial Intelligence, Helwan University

Supervised by: [Dr. Salwa Osama]

June 2025

جامعة حلوان
كلية الحاسبات والذكاء الاصطناعي
قسم علوم الحاسب

AI Mirror (مرآة الذكاء الاصطناعي)

مشروع تخرج مقدم من الطلاب:

أنس صالح موسى الصادق (20210192)

يارا محمد عطية الطوخى (20211041)

منة الله عصام عبد العزيز (20210959)

محمود نصر عبد العزيز (20210883)

محمد أشرف أبو المعاطي (20210742)

محمد إيهاب محمد فريد (20210753)

مقدم استيفاءً لمتطلبات الحصول على درجة البكالوريوس في علوم الحاسبات والذكاء الاصطناعي من قسم علوم الحاسب بكلية الحاسبات والذكاء الاصطناعي، جامعة حلوان.

تحت إشراف:

د. سلوى أسامة

يونيو 2025

Acknowledgement

We would like to express our deepest gratitude to our project supervisor, Dr. Salwa Osama, for her invaluable guidance, expertise, and insightful feedback throughout the development of this project. Her direction was instrumental in navigating the complexities of persona simulation and keeping our work focused and innovative.

We also extend our sincere thanks to the faculty members of the Computer Science department for their academic support. Finally, our appreciation goes to our friends and colleagues who provided moral support and participated in our user testing phases, offering crucial feedback that helped shape the final product.

Abstract

Users of modern conversational AI are increasingly seeking personalized, emotionally aware, and relatable digital companions. However, current chatbots often provide generic responses based on pre-defined personas, failing to reflect a user’s unique communication style, emotional tone, or personal history. This creates a gap where interactions feel impersonal and lack a deep, meaningful connection. This project addresses this challenge by designing and developing "AI Mirror," a novel multi-modal system that creates an AI character designed to mirror the user’s own persona, moving beyond simple role-playing to achieve a high-fidelity, personalized simulation.

The final architecture employs a Retrieval-Augmented Generation (RAG) pipeline built upon a powerful pre-trained Large Language Model (Llama 3.3 70B via Groq), guided by sophisticated prompt engineering. The system processes the user's own conversational data, ingested from sources like WhatsApp, Telegram, and a structured questionnaire, to generate a detailed character profile. This profile is then vectorized into a FAISS store, creating an efficient, context-aware memory for the AI. The entire system is integrated into a cross-platform React Native mobile application supporting text, voice messaging, and real-time voice calls.

The project successfully produced a robust framework for creating deeply personalized AI companions. The final RAG-based approach provides fast, coherent, and contextually relevant responses that effectively capture the user's style, emotion, and persona. The system's ability to integrate multi-modal data ingestion and interaction offers a significant enhancement to the potential for natural and meaningful human-AI interaction.

Keywords

Conversational AI, FAISS, Large Language Models (LLMs), Multi-Modal AI, Persona Simulation, Personalized Dialogue, Prompt Engineering, Retrieval-Augmented Generation (RAG), Voice Chatbots.

Table of Contents

Acknowledgement 3
Abstract 4
  Keywords 4
Table of Contents 5
List of Figures 7
List of Tables 8
List of Abbreviations 9
Glossary 10
Chapter 1: Introduction 11
  1.1 Overview 11
  1.2 Problem Statement 11
  1.3 Project Objectives 11
  1.4 Project Scope 12
  1.5 Work Methodology 13
  1.6 Work Plan (Gantt Chart) 14
  1.7 Report Organization 17
Chapter 2: Literature Review & Theoretical Background 18
  2.1 Theoretical Background 18
  2.2 Review of Academic Literature 19
  2.3 Analysis of Commercial Systems 20
Chapter 3: System Analysis and Design 21
  3.1 Development Methodology 21
  3.2 Requirements Gathering 21
  3.3 Requirements Analysis: 21
    3.3.1 Functional Requirements (FR) 21
    3.3.2 Non-Functional Requirements (NFR) 22
  3.4 System Design 23
    3.4.1 System Architecture Diagram 23
    3.4.2 Use Case Diagram 24
    3.4.3 Sequence Diagrams 31
    3.5 System Workflow 38
Chapter 4: Datasets and Data Preprocessing 40
  4.1 Datasets for Initial Experimentation 40

4.2 Data Sources for the Final System 40
4.3 Data Preprocessing Pipeline..41
4.4 Audio Data Processing 41
Chapter 5: Implementation and Experiments 42
5.1 Development Environment and Tools 42
5.2 Experimental Phases and System Evolution..44
5.2.1 Experiment 1: The Quest for Efficient Fine-Tuning 44
5.2.2 Experiment 2: The Data Acquisition Gauntlet and the Pivot to Inference 45
5.2.3 Experiment 3: RAG with LoRA Fine-Tuning on Llama-3.1-8B 47
5.2.4 Experiment 4: Evaluating a Specialized Conversational Model (BlenderBot 3)..........48
5.2.5 Experiment 5 (Final Architecture): RAG-per-Persona with LLM Inference 48
5.3 Final Implemented Architecture 49
5.3.1 System Overview 49
5.3.2 Detailed Workflow 50
5.4 Comparative Analysis of Experiments 51
Chapter 6: System Testing and Evaluation 52
6.1 Testing Strategy 52
6.2 System Evaluation..53
6.3 Test Cases and Results..54
6.3.1 Summary of Executed Test Cases 54
6.3.2 Detailed Test Scenarios: Telegram Login Module 55
6.3.3 Detailed Test Scenarios: Questionnaire Module..63
6.4 Qualitative Evaluation Based on Chat Simulation 67
Chapter 7: Conclusion and Future Work 69
7.1 Conclusion..69
7.2 Project Limitations 69
7.3 Future Work and Recommendations 69
References 70

List of Figures

Figure 1.1: Approximate Timeline for "AI Mirror" Project 16

Figure 3.1: High-Level System Architecture 23

Figure 3.2: System Use Case Diagram 24

Figure 3.3: Sequence Diagram for Questionnaire Flow 31

Figure 3.4: Sequence Diagram for WhatsApp Import Flow 32

Figure 3.5: Sequence Diagram for Telegram Persona Import Flow 33

Figure 3.6: Sequence Diagram for Voice Call Interaction 34

Figure 3.7: Class Diagram 35

Figure 3.8: Activity Diagram 36

Figure 3.9: Logical Data Storage Structure 37

Figure 5.1: Complete Project Overview Diagram 49

Figure 6.1: Validation Number Test Case 55

Figure 6.2: Sending Code Request 57

Figure 6.3: Code Verification 59

Figure 6.4: Login Successfully 61

Figure 6.5: Initialize Questionnaire Character 63

Figure 6.6: Character Created Successfully 65

Figure 6.7: WhatsApp Conversation Input Data 67

Figure 6.8: AI Persona Responding with Contextual Accuracy 68

List of Tables

Table 1.1: Gantt chart 14

Table 2.1: Comparative Analysis of "Al Mirror" against Commercial Systems............. 20

Table 3.1: Use Case Scenario Create Persona via Questionnaire 25

Table 3.2: Use Case Scenario - Import Persona from WhatsApp 26

Table 3.3: Use Case Scenario - Import Persona from Telegram 27

Table 3.4: Use Case Scenario - Interact via Text Chat 28

Table 3.5: Use Case Scenario - Interact via Voice Message 29

Table 3.6: Use Case Scenario - Interact via Voice Call 30

Table 5.1: Summary of Data Acquisition Attempts in Experiment 2 47

Table 5.2: Comparative Analysis of Experiments 51

Table 6.1: System Evaluation Against Objectives 53

Table 6.2: Test Cases Summary 54

Table 6.3: TC-TG-LOGIN-01 - Invalid Phone Number Format 56

Table 6.4: TC-TG-LOGIN-02 - Valid Phone Number Submission 58

Table 6.5: TC-TG-LOGIN-03- Invalid Verification Code 60

Table 6.6: TC-TG-LOGIN-04 - Successful Login 62

Table 6.7: TC-QUES-01 - Missing Required Input (Bio) 64

Table 6.8: TC-QUES-02 Successful Questionnaire Submission 66

List of Abbreviations

- **AI:** Artificial Intelligence
- **API:** Application Programming Interface
- **FAISS:** Facebook AI Similarity Search
- **FR:** Functional Requirement
- **LLM:** Large Language Model
- **LoRA:** Low-Rank Adaptation
- **NFR:** Non-Functional Requirement
- **RAG:** Retrieval-Augmented Generation
- **STT:** Speech-to-Text
- **TTS:** Text-to-Speech
- **UAT:** User Acceptance Testing
- **UML:** Unified Modelling Language

Glossary

- **Fine-Tuning:** The process of taking a pre-trained language model and further training it on a smaller, specific dataset to adapt it to a particular task or persona.
- **Prompt Engineering:** The technique of carefully crafting input text (prompts) to guide a large language model to generate a desired output without altering the model's underlying weights.
- **Retrieval-Augmented Generation (RAG):** An AI framework that enhances the quality of LLM-generated responses by grounding them in an external knowledge base. It retrieves relevant information first, then uses that information to generate a more accurate and contextual response.
- **Vector Store:** A specialized database designed to store and efficiently search high-dimensional vectors. In this project, it is used to store vectorized representations of the user's data for fast retrieval during conversation.
- **Voice Cloning:** The process of creating a synthetic voice that sounds like a specific person, used in this project to make the AI persona speak with a voice similar to the user's.

Chapter 1: Introduction

1.1 Overview

In recent years, the field of conversational Artificial Intelligence (AI) has witnessed exponential growth, largely driven by the advent of powerful Large Language Models (LLMs). These models have enabled the development of sophisticated chatbots and virtual companions capable of understanding and generating human-like text with remarkable fluency. As users become more accustomed to interacting with AI, their expectations have evolved beyond mere functional assistance. There is a growing demand for AI companions that are not just intelligent but are also personal, emotionally aware, and relatable. The next frontier in conversational AI lies in moving from generic, one-size-fits-all agents to deeply personalized entities that can understand, reflect, and adapt to an individual's unique identity.

1.2 Problem Statement

Despite their advanced capabilities, most contemporary chatbots and AI companions fail to capture the nuances of an individual's personality. They often rely on pre-scripted responses or a generic, pre-defined persona, resulting in conversations that feel monotonous, impersonal, and lack genuine connection. This lack of true personalization prevents the formation of a deep and meaningful bond between the user and the AI. The core problem this project addresses is **the inability of existing systems to create a high-fidelity AI simulation that truly mirrors a specific user's communication style, emotional tone, personal history, and unique personality based on their own historical data.**

1.3 Project Objectives

To address the stated problem, this project aims to achieve the following specific, measurable, achievable, relevant, and time-bound (SMART) objectives:

1. **To design and develop a multi-modal system** capable of creating a personalized AI character that mirrors a real user's personality.
2. **To implement a flexible data ingestion pipeline** that accepts user data from three distinct sources: a structured questionnaire, WhatsApp chat logs, and Telegram chat logs.
3. **To build a robust persona generation module** that uses an LLM to analyze the ingested data and automatically generate a detailed character profile, including a

narrative biography and structured key-value answers.

4. **To implement a Retrieval-Augmented Generation (RAG) architecture** using a FAISS vector store to provide the AI character with a persistent, context-aware memory of the user's data.
5. **To integrate the AI system into a cross-platform mobile application** (React Native) that supports multi-modal interaction: text chat, asynchronous voice messaging, and real-time voice calls.
6. **To provide a simple and intuitive user experience** for creating and interacting with the AI persona, abstracting away all technical complexity from the end-user.

1.4 Project Scope

The scope of the AI Mirror project is defined as follows:

- **In-Scope:**
    - The system creates a single, mirrored AI persona per user based on the data they provide.
    - Data ingestion is supported via a manual questionnaire, WhatsApp chat import (via QR code), and a user-friendly Telegram login flow (via phone and code).
    - The core AI methodology relies on inference and prompt engineering with a pre-trained LLM (Llama 3.3 70B), augmented by a RAG pipeline.
    - The system supports multi-modal chat, including text, voice input (Whisper transcription), voice output (gTTS/Coqui TTS), and a dedicated voice call screen.
    - An animated avatar is included to enhance the user's sense of interaction.
- **Out-of-Scope:**
    - The system does not support the creation of multiple fictional characters. Its focus is solely on mirroring the user.
    - Data is not automatically scraped from social media or other platforms; all data ingestion is user-initiated and requires consent.
    - The project does not involve training or full fine-tuning of a base LLM from scratch due to computational constraints.
    - The lip-syncing feature for the avatar is in a developmental stage and not fully implemented.

1.5 Work Methodology

The project was developed using an **Iterative and Agile-inspired methodology**. Given the experimental nature of AI model integration and the evolving understanding of the project requirements, a rigid Waterfall model was unsuitable. Our process involved a series of cycles: **Experimentation**, where a specific technical approach was implemented (e.g., fine-tuning with LoRA); **Evaluation**, where we assessed the results against our objectives; and **Adaptation**, where we used the findings and supervisor feedback to pivot our approach. This iterative method allowed for the flexibility required to navigate technical challenges and gradually refine the system towards the final, optimal architecture.

1.6 Work Plan (Gantt Chart)

Project Work Plan - Detailed Task Breakdown
The following table details the timeline and duration for each major phase and sub-task of the "AI Mirror" project, corresponding to the visual Gantt chart. The project is scheduled over a 10-month period, from September 2024 to June 2025.

Table 1.1: Gantt chart

Phase ID	Task / Phase	Start Date (Approx.)	End Date (Approx.)	Duration	Description & Key Activities
1	Research & Planning	Sep 1, 2024	Oct 15, 2024	1.5 Months	Defining problem statement, analyzing related work, gathering requirements, and selecting the initial technology stack.
2	Experimental R&D	Oct 16, 2024	Mar 15, 2025	5 Months	This phase was the most intensive and involved iterative prototyping to find the optimal architecture.
2.1	Exp. 1: Fine-Tuning SLM	Oct 16, 2024	Nov 15, 2024	1 Month	Initial experiments with fine-tuning small language models like Microsoft Phi-3 using LoRA.
2.2	Exp. 2: LLM Inference	Nov 16, 2024	Dec 15, 2024	1 Month	Pivoting to use large models (Llama) via inference APIs. Focus on prompt engineering.
2.3	Exp. 3: RAG + LoRA	Dec 16, 2024	Jan 15, 2025	1 Month	Implementing a hybrid approach combining RAG with fine-tuning to address persona consistency issues.
2.4	Exp. 4: BlenderBot	Jan 16, 2025	Feb 15, 2025	1 Month	Evaluating a specialized conversational model (BlenderBot 3) to test its memory and dialogue capabilities.
2.5	Exp. 5 & Final Decision	Feb 16, 2025	Mar 15, 2025	1 Month	Finalizing tests and making the architectural decision

					to proceed with the "RAG-per-Persona" model.
3	Final System Development	Mar 16, 2025	May 15, 2025	2 Months	Building the robust final backend based on the chosen architecture, including all APIs for data ingestion and interaction.
4	UI & Testing	May 16, 2025	Jun 30, 2025	1.5 Months	Developing the final mobile UI/UX, integrating all features, and conducting comprehensive unit, integration, and user acceptance testing.
5	Documentation	Mar 16, 2025	Jun 15, 2025	3 Months (Parallel)	Writing the final project dissertation, preparing presentation materials, and documenting the code. This task runs in parallel with the final development and testing phases.

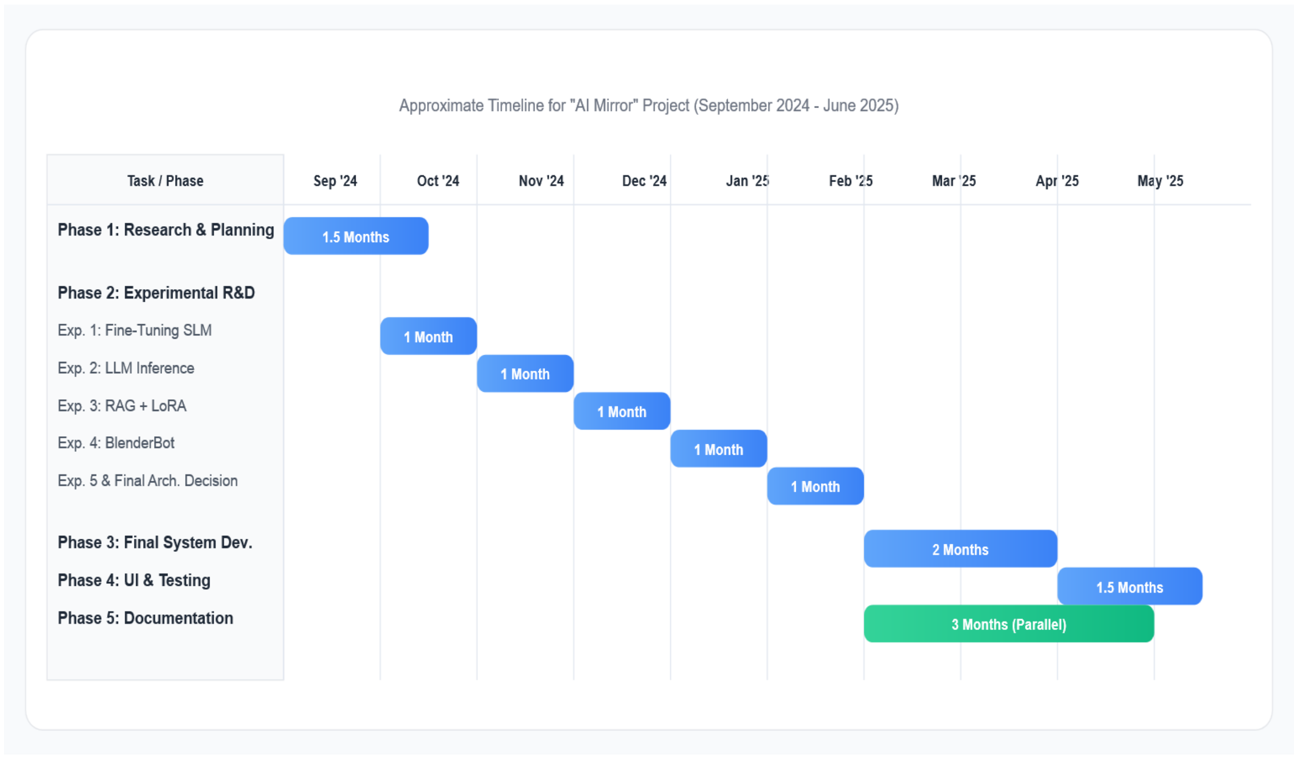


Figure 1.1: Approximate Timeline for "AI Mirror" Project (September 2024 - June 2025).

1.7 Report Organization

This report is organized into six chapters. Chapter 1 introduces the project background, problem, objectives, and scope. Chapter 2 provides a review of the theoretical background and related work. Chapter 3 details the proposed solution, including requirements and system design diagrams. Chapter 4 presents the implementation details, experimental phases, and results. Chapter 5 discusses the project's findings, conclusion, and future work.

Chapter 2: Literature Review & Theoretical Background

This chapter provides the foundational knowledge upon which the "AI Mirror" project is built. It begins by exploring the core technologies and theoretical concepts that enable modern conversational AI. It then delves into a comprehensive review of relevant academic literature, followed by an analysis of existing commercial systems to identify the research gap this project aims to fill.

2.1 Theoretical Background

Large Language Models (LLMs): LLMs are deep learning models with billions of parameters, pre-trained on vast amounts of text data [1]. Our project ultimately utilized Meta's **Llama 3.3 70B-Versatile** model, accessed via the **Groq** inference engine [2], selected for its exceptional speed and strong reasoning capabilities, making it ideal for real-time conversational applications.

Retrieval-Augmented Generation (RAG): Proposed by Lewis et al. [3], RAG enhances LLM performance by grounding responses in external knowledge. It combines a *retriever* (fetching relevant information) with a *generator* (the LLM). In our system, the user's data is vectorized using **Sentence-Transformers (all-MiniLM-L6-v2)** [4] and stored in a **FAISS (Facebook AI Similarity Search)** vector store [5]. When the user sends a message, the RAG pipeline retrieves relevant information to provide context for the LLM, ensuring more accurate and personalized replies.

Fine-Tuning vs. Prompt Engineering:

- **Fine-Tuning:** Our initial experiments involved fine-tuning models like Phi-3 [6] using **LoRA (Low-Rank Adaptation)** [7]. This proved to be resource-intensive, slow, and led to "persona confusion," where the model struggled to differentiate between multiple training personas.
- **Prompt Engineering:** Our final, successful methodology relies on this technique. We dynamically construct a detailed meta-prompt that includes the character's generated biography, Q&A, and recent chat history. This prompt "instructs" the pre-trained LLM on how to behave, achieving high-fidelity persona simulation without altering the model's weights.

Speech & Avatar Technologies:

- **Speech-to-Text (STT):** We use **OpenAI's Whisper (Large V3 model)** [8] to transcribe user voice recordings, chosen for its high accuracy and robust support for various Arabic dialects.

- **Text-to-Speech (TTS):** We utilize **Coqui AI's XTTS model** [9] for its high-quality voice cloning capabilities, allowing the AI's responses to be spoken in a personalized voice.

2.2 Review of Academic Literature

Our work is built upon extensive research in persona-based dialogue systems.

Foundational Work and Datasets: The field was significantly advanced by datasets like **PersonaChat** [10] and the associated **ConvAI2** challenge, which provided agents with explicit profiles to make conversations more personal. These highlighted the data scarcity problem that our system overcomes by using the user's own data. The PersonaChat dataset, released by Facebook AI in 2018, consists of over 160,000 utterances where each speaker is assigned a profile of 5 sentences, training models to generate responses grounded in their persona.

Advanced Mechanisms for Persona Consistency: The paper "Personalized Dialogue Generation with Persona-Adaptive Attention" (PAA) by Song et al. [11] introduced a novel attention mechanism to dynamically balance a speaker's persona with dialogue context. Their approach demonstrated that strong performance could be achieved even with limited data (achieving near full-data performance with just 20-30% of training data), an insight that validated our move away from data-hungry fine-tuning methods.

Simulating Personas with Pre-trained LLMs: More recent research, such as "Characteristic AI Agents via Large Language Models" by Zhou et al. [12], has shifted focus from fine-tuning to guiding large, pre-trained models. Their framework relies on creating detailed character profiles from sources like Wikipedia and using them to engineer prompts, a methodology that is the primary inspiration for our system's core architecture. Their introduction of the **Character100** benchmark, which achieved results such as 69.4 BLEU and 78.5 ROUGE-L, also highlighted the growing interest in evaluating the ability of LLMs to simulate real personalities.

Retrieval-Augmented Generation: The seminal paper "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" by Lewis et al. [3] proposed the RAG framework that we have adopted. By combining dense retrieval with a sequence-to-sequence model, they demonstrated higher factual accuracy and relevance, a principle we apply to ground our AI's responses in the user's personal "knowledge base."

Research Gap: While the reviewed literature shows significant progress in creating agents with *pre-defined* or *fictional* personas, a clear gap exists in creating high-fidelity agents that can dynamically **mirror a real, living user's personality** based on their own complex and multi-modal conversational history, including real-time voice interaction. Our project, "AI Mirror," directly addresses this gap.

2.3 Analysis of Commercial Systems

To position our project within the current market, we analyzed several leading commercial applications. The following table provides a comparative analysis of "AI Mirror" against popular platforms like Replika, Character.AI, and Anima AI.

Feature	**AI Mirror**	**Replika**	**Character.AI**	**Anima AI**
Mimics User's Real Personality	Yes (based on real user data)	No	Limited (scripted)	Partially
Learns from Real Messages	WhatsApp, Telegram, Questionnaire	Questionnaire only	No	No
Matches User's Style & Emotions	Fully (via data analysis)	Emotionally Reactive	Scripted Responses	Tries to Adapt
Supports Avatar	Yes (Animated)	Limited	No	Limited
Supports Voice Output	Yes (High-quality, clonable)	Yes	No	Yes (Basic)
User-Created Character	Focuses on one mirrored character	No	Yes (Fictional)	No
Primary Goal	Personalized, self-mimicking AI	Emotional companion AI	Role-play	Emotional chat companion

This analysis reveals that while existing systems focus on providing companionship or entertainment through fictional roles, "AI Mirror" is unique in its primary objective: to create a deep, authentic, and personalized simulation of the user themselves.

Chapter 3: System Analysis and Design

3.1 Development Methodology

The project was developed using an **Iterative and Agile-inspired methodology**. Given the experimental nature of AI model integration and the evolving understanding of the project requirements, a rigid Waterfall model was unsuitable. Our process involved a series of cycles:

1. **Experimentation:** Each cycle began with an experiment (e.g., fine-tuning with LoRA).
2. **Evaluation:** We assessed the results of each experiment against our objectives.
3. Adaptation: Based on the evaluation and supervisor feedback, we adapted our approach, pivoting from fine-tuning to a RAG + Prompt Engineering architecture after early experiments proved inefficient.
   This iterative approach allowed for flexibility and the gradual refinement of the system.

3.2 Requirements Gathering

- **Competitive Analysis:** We analyzed applications like Character.ai and Replika to define our unique value proposition (mirroring a real user).
- **Supervisor Meetings:** Regular discussions helped clarify academic requirements and refine the project's technical direction.
- **Prototyping:** Building and testing different versions served as a primary method for discovering technical constraints and user experience needs.

3.3 Requirements Analysis:

3.3.1 Functional Requirements (FR)

- **FR1:** The system shall allow a user to create a persona via a structured questionnaire.
- **FR2:** The system shall allow a user to import a WhatsApp chat log to generate a persona.
- **FR3:** The system shall provide a secure and user-friendly login flow for Telegram

to import a chat log.

- **FR4:** The system shall process imported data to automatically generate a character biography and key-value answers using an LLM.
- **FR5:** The system shall store the generated character profile persistently on the backend and locally on the client device.
- **FR6:** The system shall allow the user to engage in a text-based conversation with their AI persona.
- **FR7:** The system shall support asynchronous voice messaging (record, send, transcribe, reply).
- **FR8:** The system shall support a real-time voice call mode (voice-in, voice-out).
- **FR9:** The system shall use a RAG mechanism to retrieve context from the user's data to inform responses.
- **FR10:** The system shall synthesize the AI persona's text replies into speech for voice messages and calls.

3.3.2 Non-Functional Requirements (NFR)

- **NFR1 (Performance):** The AI's response time in text chat should be near real-time (under 5 seconds). Voice call latency should be minimized for a natural feel.
- **NFR2 (Usability):** Persona creation flows must be intuitive and require no technical expertise.
- **NFR3 (Security):** User data and session tokens must be handled securely and not be stored unencrypted.
- **NFR4 (Reliability):** The system must gracefully handle failures in external API calls and provide clear error messages.

3.4 System Design

This section provides a detailed breakdown of the system's design through a series of UML diagrams, illustrating the architecture, user interactions, and data flow.

3.4.1 System Architecture Diagram

The architecture of AI Mirror is a distributed system composed of three primary components: the mobile frontend, a specialized middleware for WhatsApp, and the main AI backend. This design ensures modularity and scalability.

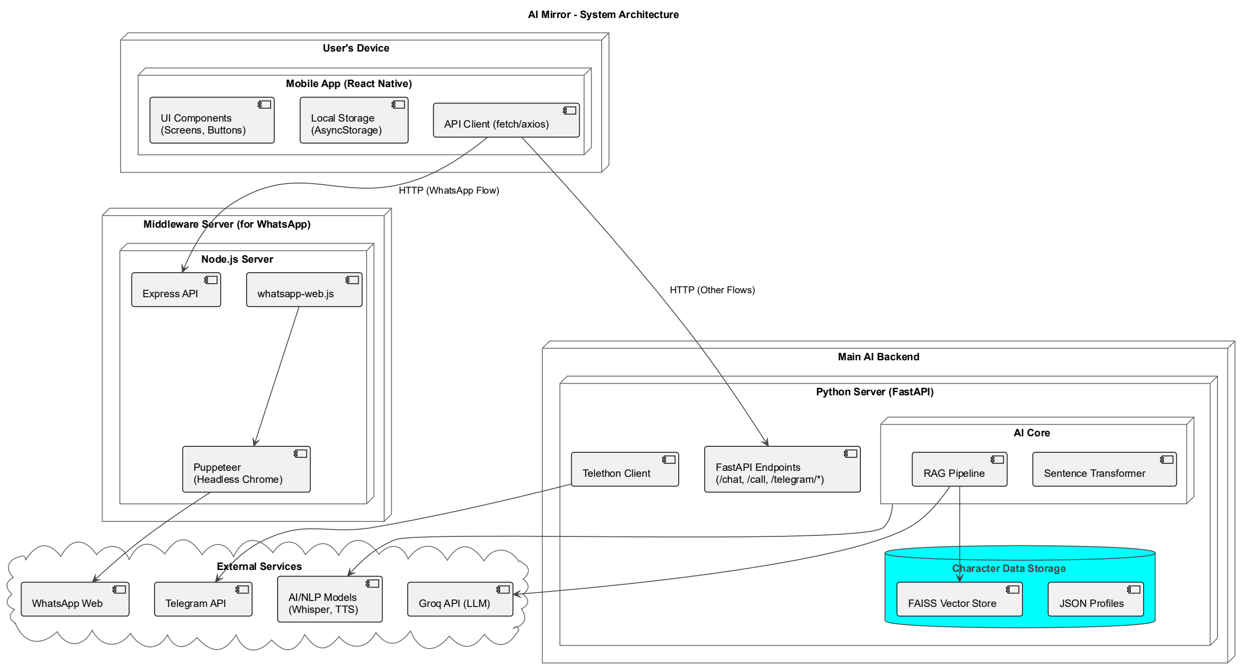


Figure 3.1: High-Level System Architecture

3.4.2 Use Case Diagram

This diagram illustrates the main functionalities available to the user, including all persona creation methods and interaction modes.

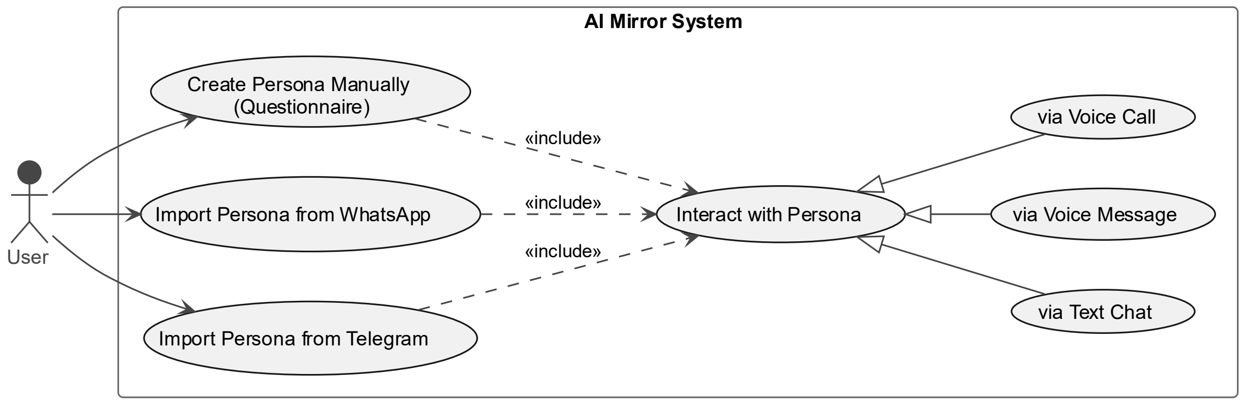


Figure 3.2: System Use Case Diagram

3.4.2.1 Use Case Scenarios

This section provides a detailed breakdown of the primary use cases for the "AI Mirror" system. Each scenario outlines the actors, pre-conditions, main workflow, and post-conditions.

Table 3.1: Create Persona via Questionnaire

Use Case ID	**UC-01**
Use Case Name	Create Persona via Questionnaire
Actor(s)	User
Description	This use case allows a user to manually create a new AI persona by providing a name, a profile image, and completing a detailed questionnaire to build its profile.
Pre-conditions	1. The user must have the application open. 2. The user selects the "Create New Character" option.
Main Flow (Basic Path)	1. The user inputs a name for the new persona. 2. The user selects a profile image from their device's gallery. 3. The user saves the initial profile information. 4. The system navigates the user to the Questionnaire screen. 5. The user answers a series of text-based and multiple-choice questions. 6. The user writes a short biography for the persona. 7. The user presses the "Submit" or "Finish" button upon completion. 8. The application saves the complete character data (ID, name, image URI, bio, answers) to local device storage using AsyncStorage. 9. The application sends the complete character data to the backend system by calling the POST /store_data API endpoint. 10. The backend processes the data, converts texts into vector embeddings, and builds a dedicated FAISS index for the persona's RAG system.
Post-conditions	1. A new persona is successfully created and persisted on both the local device and the backend system. 2. The new persona is available for interaction. 3. The user is navigated to the Chat Screen with the newly created persona.
Alternative Flows / Exceptions	1a. Missing Name or Image: If the user attempts to save without providing a name or image, the system displays an alert prompting them to complete the required fields. 2a. Incomplete Questionnaire: If the user submits the questionnaire without answering the minimum required questions, the system displays an alert.

	3a. Server Connection Failure: If the application fails to connect to the /store_data endpoint, an error message is displayed to the user indicating a sync failure, while confirming that the data has been saved locally.

Table 3.2: Import Persona from WhatsApp

Use Case ID	**UC-02**
Use Case Name	Import Persona from WhatsApp
Description	This use case enables the user to import a WhatsApp chat history to automatically generate an AI persona based on the style and content of the conversation.
Pre-conditions	1. The user must have the corresponding Node.js WhatsApp service running on their local network. 2. The user must select the "Import from WhatsApp" option from the main screen.
Main Flow (Basic Path)	1. The user initiates the import process from the application. 2. The application connects to the local Node.js server. 3. The user scans the QR code displayed in the server's terminal using the WhatsApp application on their phone. 4. Upon successful linking, the Node.js server fetches the user's chat list. 5. The application displays the chat list to the user. 6. The user selects a specific chat to import. 7. The Node.js server fetches the message history for the selected chat and forwards it to the main Python backend. 8. The Python backend analyzes the conversation text using the LLM to generate an automatic biography (Bio) and extract key information (Q&A). 9. The backend creates and indexes the new persona in a dedicated FAISS database. 10. The complete, newly generated persona profile is returned to the application.
Post-conditions	1. A new persona based on the WhatsApp chat is created, stored, indexed, and ready for interaction.
Alternative Flows / Exceptions	1a. Cannot Reach Node.js Server: The application displays an error message indicating a connection failure. 2a. QR Code Scan Fails: The user is notified to try again. 3a. Empty Chat: If the selected chat does not contain sufficient messages for analysis, the backend returns an error.

Table 3.3 Import Persona from Telegram

Use Case ID	**UC-03**
Use Case Name	Import Persona from Telegram
Description	This use case allows the user to log in to their Telegram account from within the app to import a chat history and automatically create a persona.
Pre-conditions	1. The user must select the "Import from Telegram" option.
Main Flow (Basic Path)	1. The user enters their phone number in international format. 2. The application sends the phone number to the POST /telegram/send_code endpoint on the backend. 3. The backend uses the Telethon library to request a login code. 4. The user receives the code in their Telegram app and enters it into the AI Mirror app. 5. (If two-factor authentication is enabled) The user is prompted to enter their password. 6. The user submits the code/password. The backend authenticates and obtains a session string. 7. The app uses this session string to request the chat list from the GET /telegram/chats endpoint. 8. The user selects a specific chat from the list. 9. The app requests the backend to process the selected chat by calling POST /telegram/process_chat. 10. The backend fetches the messages, generates the bio and Q&A, and creates and indexes the new persona.
Post-conditions	1. A new persona based on the Telegram chat is created, stored, indexed, and ready for interaction.
Alternative Flows / Exceptions	1a. Invalid Login Credentials: The backend returns an authentication error, which is displayed to the user. 2a. Network Error: During API calls, the application displays an appropriate error message.

Table 3.4 Interact via Text Chat

Use Case ID	**UC-04**
Use Case Name	Interact via Text Chat
Description	The user engages in a text-based conversation with a previously created AI persona.
Pre-conditions	1. The user must have created or imported at least one persona. 2. The user has selected a specific persona and is on the Chat Screen.
Main Flow (Basic Path)	1. The user types a text message in the input field. 2. The user presses the "Send" button. 3. The application immediately displays the user's message in the chat history and saves it to local storage. 4. The application prepares a request containing the characterId, the user's message text, and the recent chat history. 5. The application sends a POST request to the /chat endpoint on the backend. 6. The backend executes the RAG process: retrieving the most relevant context from the persona-specific FAISS index. 7. The backend constructs a detailed prompt and sends it to the Groq LLM. 8. The backend receives the generated text response from the LLM. 9. The backend returns the text response to the application. 10. The application displays the AI's response in the chat history and saves it.
Post-conditions	1. The chat history is updated with the user's message and the AI's response.
Alternative Flows / Exceptions	1a. Server Unreachable: The application displays an error message within the chat interface. 2a. LLM API Failure: The backend returns a predefined fallback message.

Table 3.5 Interact via Voice Message

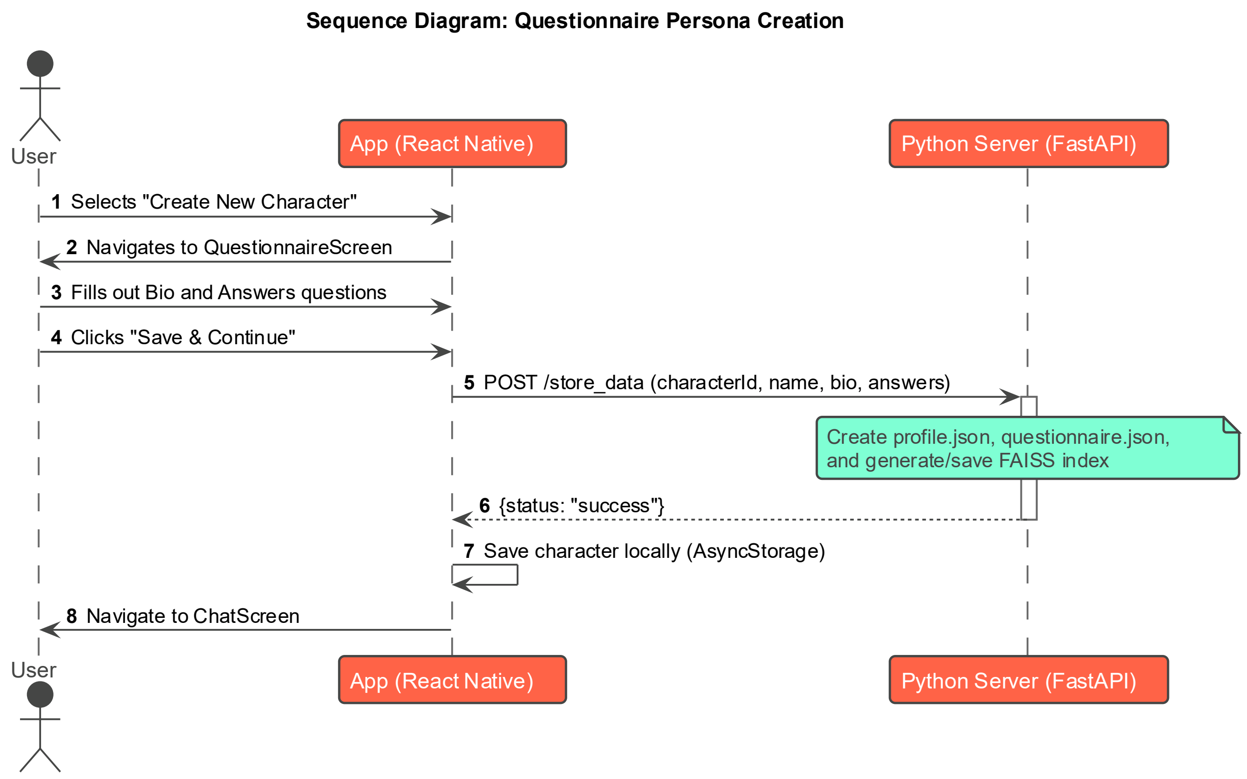
Use Case ID	**UC-05**
Use Case Name	Interact via Voice Message
Description	The user records and sends a voice message to the AI persona and receives a text-based reply.
Pre-conditions	1. A persona has been created and selected. 2. The user is on the Chat Screen.
Main Flow (Basic Path)	1. The user presses and holds the microphone icon. 2. The application begins recording audio. 3. The user releases the icon to stop recording. 4. The application sends a POST request to the /chat endpoint as a multipart/form-data request containing the audio file. 5. The backend receives the audio file. 6. The backend uses the Whisper model to transcribe the audio to text. 7. The backend follows the same RAG process as in UC-04, using the transcribed text as the query. 8. The backend receives the generated text response from the LLM. 9. The backend returns both the AI's text reply and the original user audio transcript. 10. The application displays a "voice message" component for the user (with its transcript) and the AI's text reply.
Post-conditions	1. The chat history is updated with the user's voice message (and its transcript) and the AI's text response.
Alternative Flows / Exceptions	1a. Audio Recording Fails: (e.g., no microphone permission) The application displays an alert. 2a. STT Fails: The backend may return an error or proceed with an empty string.

Table 3.6 Interact via Voice Call

Use Case ID	**UC-06**
Use Case Name	Interact via Voice Call
Description	The user engages in a real-time, voice-to-voice conversation with the AI persona.
Pre-conditions	1. A persona has been created and selected. 2. The user has navigated to the Call Screen.
Main Flow (Basic Path)	1. The user presses the "Start Talking" button. 2. The application records the user's voice. 3. The user presses the "Stop Recording" button. 4. The application sends the audio file to the POST /call endpoint. 5. The backend transcribes the audio to text using Whisper. 6. The backend executes the RAG process to generate a text-based response from the LLM. 7. The backend uses the CoquiTTS model to convert the text response into a speech audio file (e.g., WAV). 8. The backend returns the generated audio file directly as the API response. 9. The application receives the audio file (as a blob/base64) and plays it back to the user immediately.
Post-conditions	1. The user hears the spoken audio response from the AI persona, completing one turn in the voice conversation.
Alternative Flows / Exceptions	1a. TTS Generation Fails: The backend returns an error message. 2a. Audio Playback Fails: The application displays an error to the user.

3.4.3 Sequence Diagrams

A. Questionnaire Persona Creation Flow:



Sequence Diagram: Questionnaire Persona Creation
User
App (React Native)
Python Server (FastAPI)
1 Selects "Create New Character"
2 Navigates to QuestionnaireScreen
3 Fills out Bio and Answers questions
4 Clicks "Save & Continue"
5 POST /store_data (characterId, name, bio, answers)
Create profile.json, questionnaire.json, and generate/save FAISS index
6 {status: "success"}
7 Save character locally (AsyncStorage)
8 Navigate to ChatScreen
User
App (React Native)
Python Server (FastAPI)

Figure 3.3: Sequence Diagram for Questionnaire Flow

B. WhatsApp Persona Import Flow:

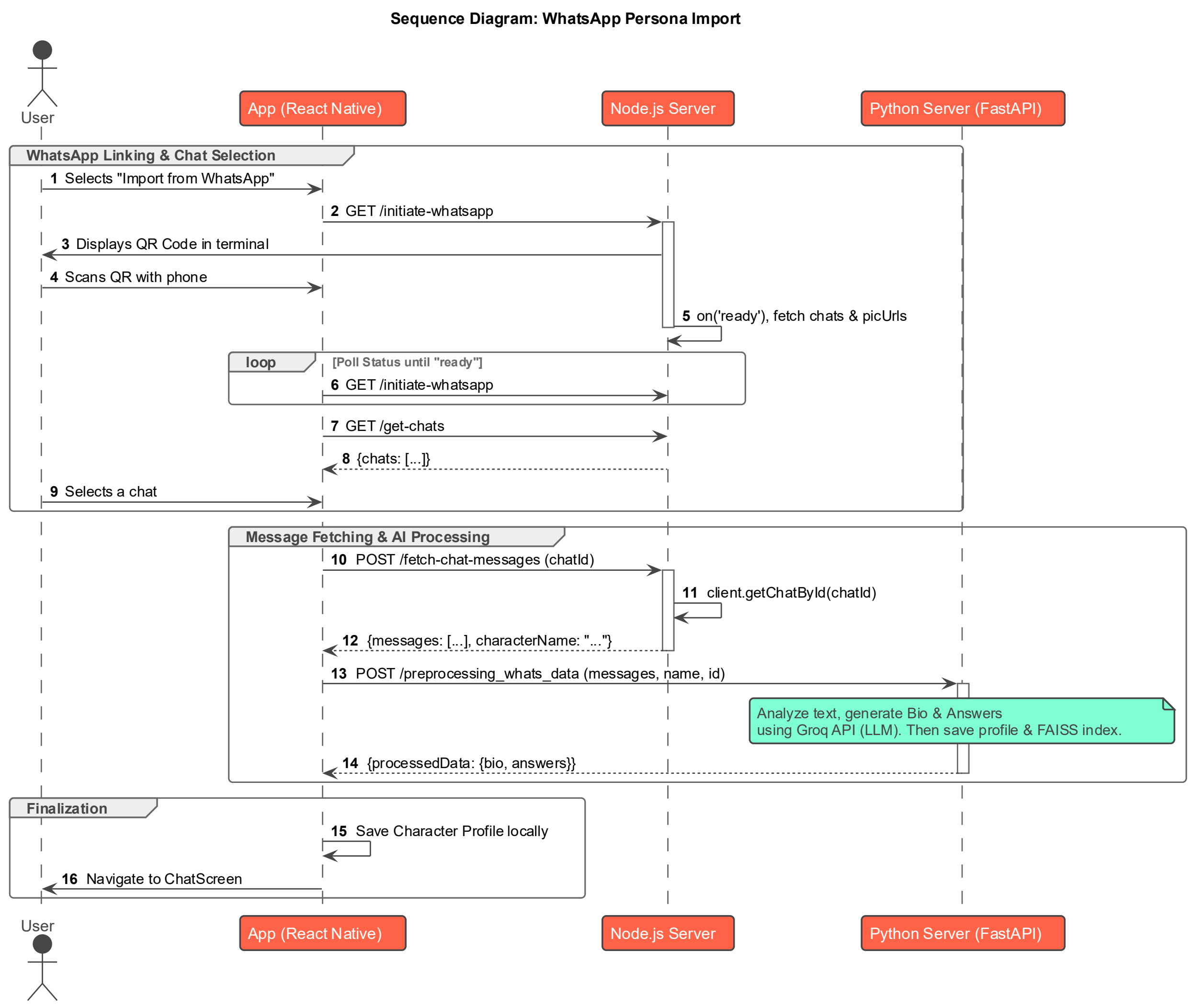
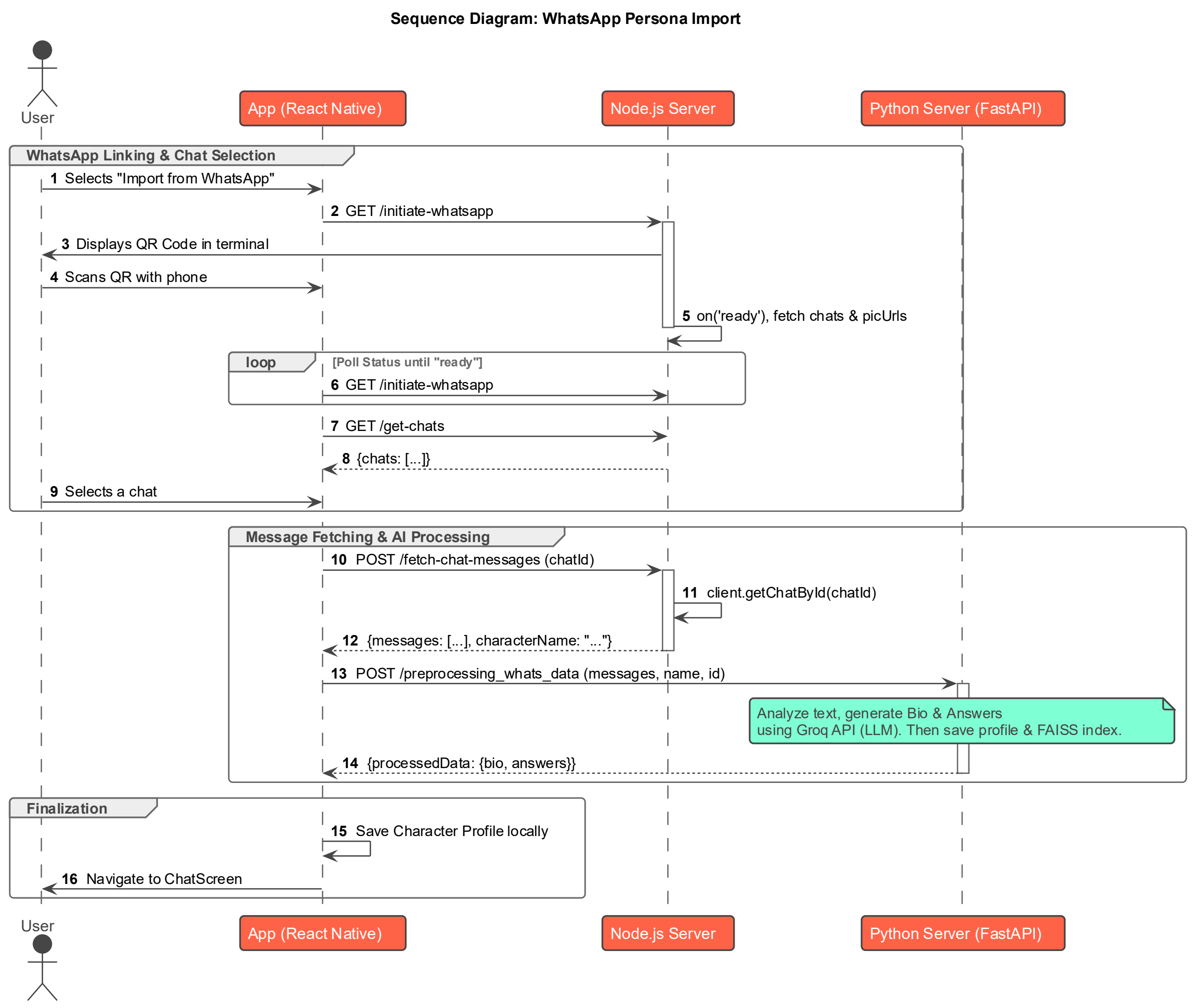

Sequence Diagram: WhatsApp Persona Import
User
App (React Native)
Node.js Server
Python Server (FastAPI)
WhatsApp Linking & Chat Selection
1 Selects "Import from WhatsApp"
2 GET /initiate-whatsapp
3 Displays QR Code in terminal
4 Scans QR with phone
5 on('ready'), fetch chats & picUrls
loop
[Poll Status until "ready"]
6 GET /initiate-whatsapp
7 GET /get-chats
8 {chats: [...]}
9 Selects a chat
Message Fetching & AI Processing
10 POST /fetch-chat-messages (chatId)
11 client.getChatById(chatId)
12 {messages: [...], characterName: "..."}
13 POST /preprocessing_whats_data (messages, name, id)
Analyze text, generate Bio & Answers using Groq API (LLM). Then save profile & FAISS index.
14 {processedData: {bio, answers}}
Finalization
15 Save Character Profile locally
16 Navigate to ChatScreen
User
App (React Native)
Node.js Server
Python Server (FastAPI)

Figure 3.4: Sequence Diagram for WhatsApp Import Flow

C. Telegram Persona Import Flow:



Sequence Diagram: WhatsApp Persona Import
User
App (React Native)
Node.js Server
Python Server (FastAPI)
WhatsApp Linking & Chat Selection
1 Selects "Import from WhatsApp"
2 GET /initiate-whatsapp
3 Displays QR Code in terminal
4 Scans QR with phone
5 on('ready'), fetch chats & picUrls
loop
[Poll Status until "ready"]
6 GET /initiate-whatsapp
7 GET /get-chats
8 {chats: [...]}
9 Selects a chat
Message Fetching & AI Processing
10 POST /fetch-chat-messages (chatId)
11 client.getChatById(chatId)
12 {messages: [...], characterName: "..."}
13 POST /preprocessing_whats_data (messages, name, id)
Analyze text, generate Bio & Answers using Groq API (LLM). Then save profile & FAISS index.
14 {processedData: {bio, answers}}
Finalization
15 Save Character Profile locally
16 Navigate to ChatScreen
User
App (React Native)
Node.js Server
Python Server (FastAPI)

Figure 3.5: Sequence Diagram for Telegram Persona Import Flow

D. Voice Call Interaction Flow:

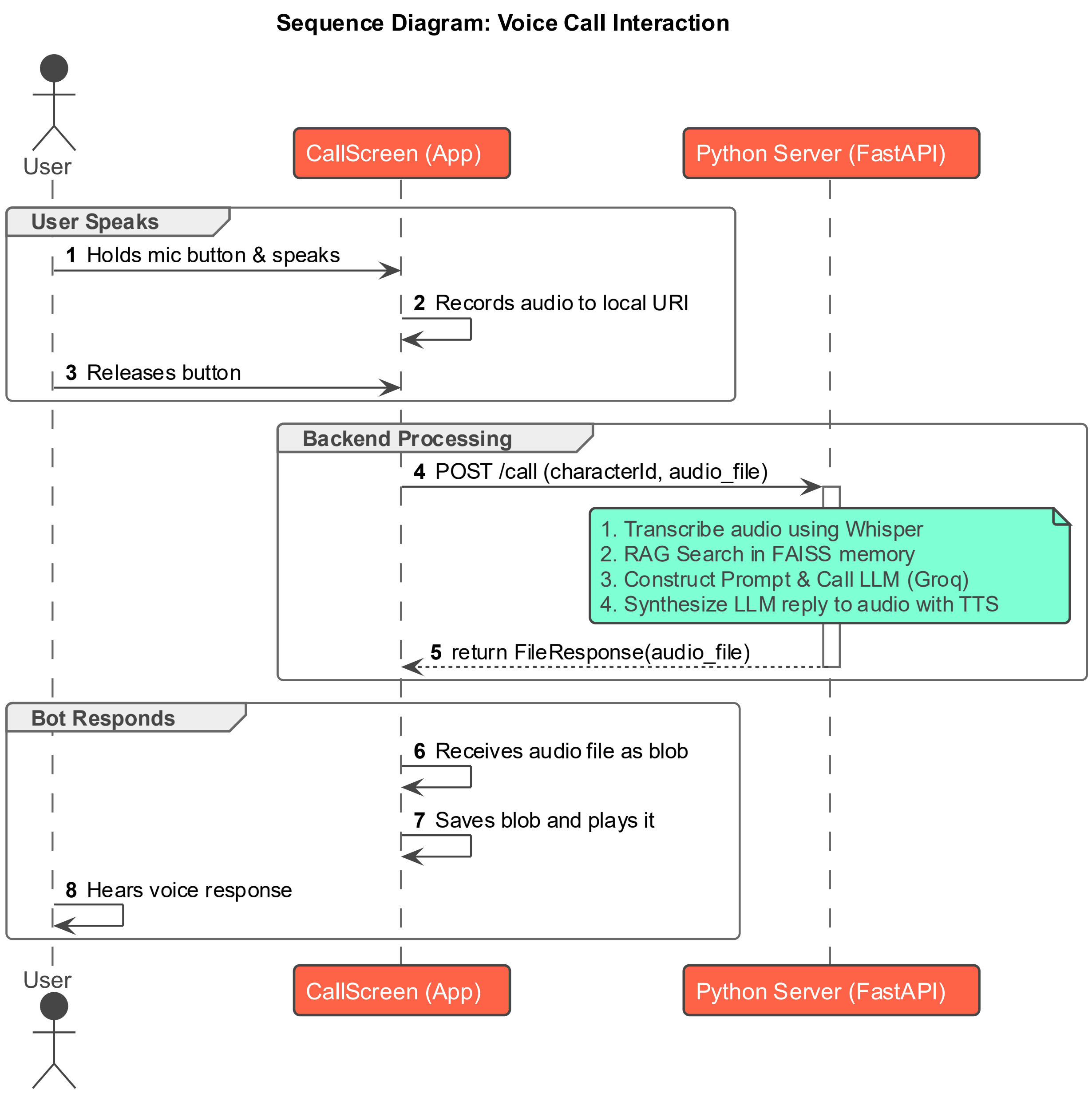

Sequence Diagram: Voice Call Interaction
User
CallScreen (App)
Python Server (FastAPI)
User Speaks
1 Holds mic button & speaks
2 Records audio to local URI
3 Releases button
Backend Processing
4 POST /call (characterId, audio_file)
1. Transcribe audio using Whisper
2. RAG Search in FAISS memory
3. Construct Prompt & Call LLM (Groq)
4. Synthesize LLM reply to audio with TTS
5 return FileResponse(audio_file)
Bot Responds
6 Receives audio file as blob
7 Saves blob and plays it
8 Hears voice response
User
CallScreen (App)
Python Server (FastAPI)

Figure 3.6: Sequence Diagram for Voice Call Interaction

3.4.4 Class Diagram

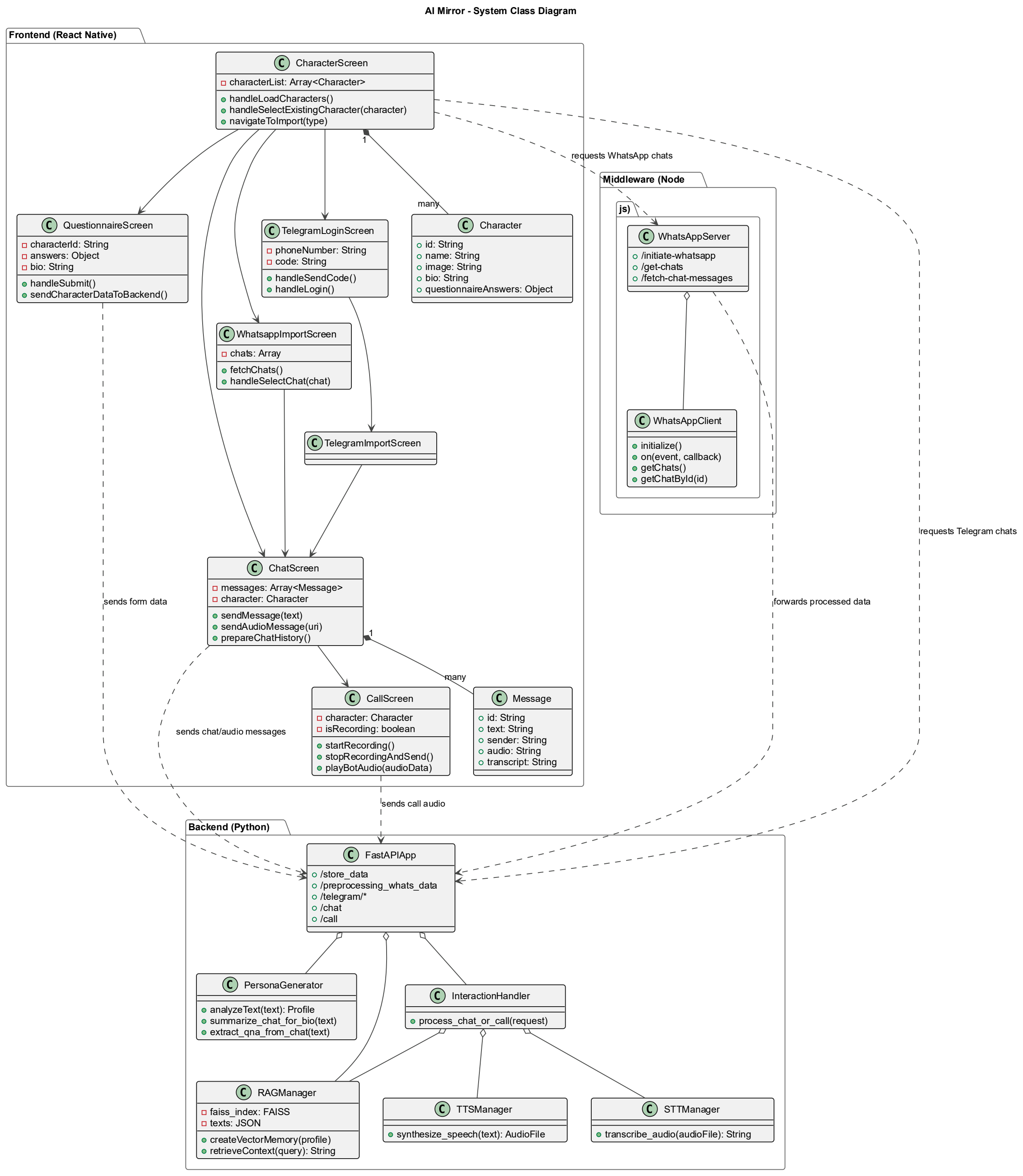
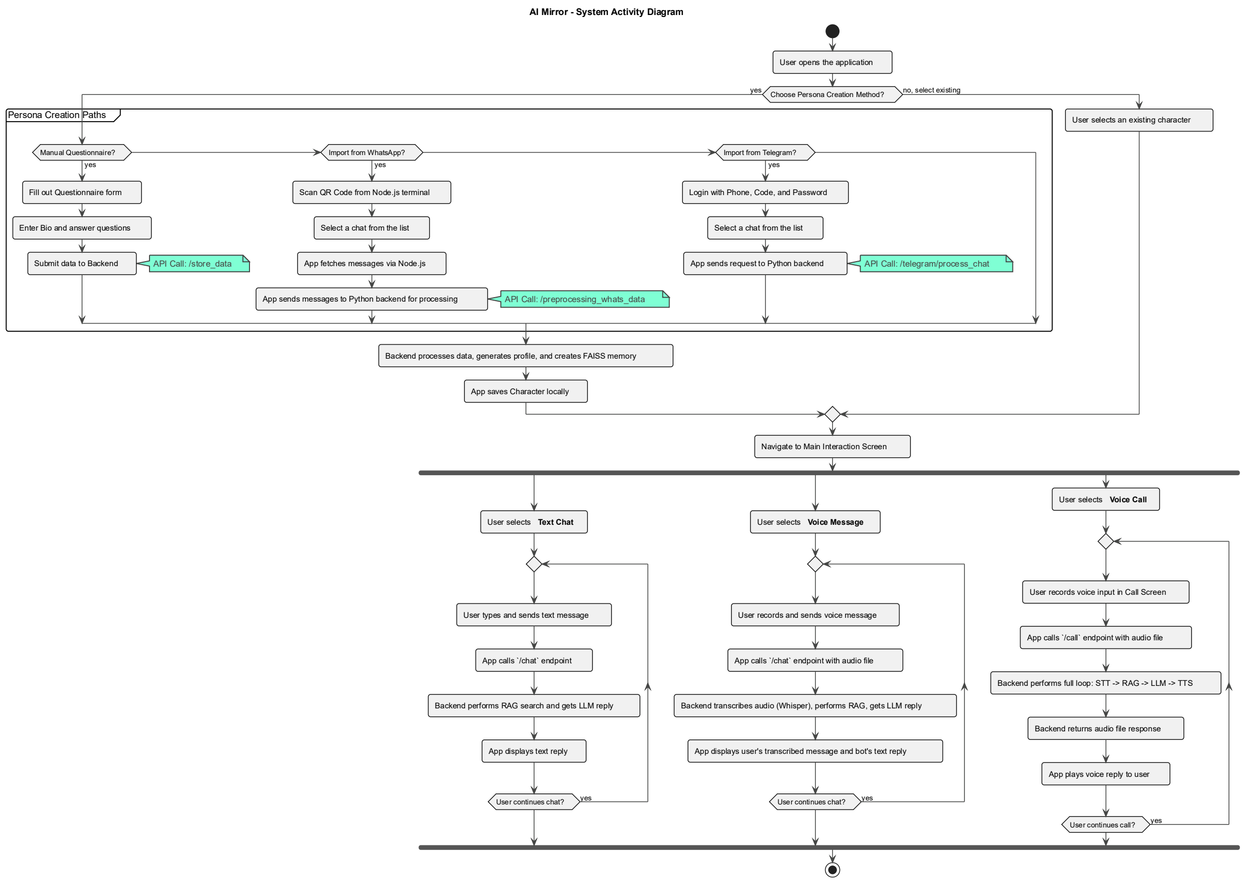


Figure 3.7: Class Diagram

3.4.5 Activity Diagram



AI Mirror - System Activity Diagram
User opens the application
Choose Persona Creation Method?
yes
no, select existing
User selects an existing character
Persona Creation Paths
Manual Questionnaire?
yes
Fill out Questionnaire form
Enter Bio and answer questions
Submit data to Backend
API Call: /store_data
Import from WhatsApp?
yes
Scan QR Code from Node.js terminal
Select a chat from the list
App fetches messages via Node.js
App sends messages to Python backend for processing
API Call: /preprocessing_whats_data
Import from Telegram?
yes
Login with Phone, Code, and Password
Select a chat from the list
App sends request to Python backend
API Call: /telegram/process_chat
Backend processes data, generates profile, and creates FAISS memory
App saves Character locally
Navigate to Main Interaction Screen
User selects Text Chat
User types and sends text message
App calls `/chat` endpoint
Backend performs RAG search and gets LLM reply
App displays text reply
User continues chat?
yes
User selects Voice Message
User records and sends voice message
App calls `/chat` endpoint with audio file
Backend transcribes audio (Whisper), performs RAG, gets LLM reply
App displays user's transcribed message and bot's text reply
User continues chat?
yes
User selects Voice Call
User records voice input in Call Screen
App calls `/call` endpoint with audio file
Backend performs full loop: STT -> RAG -> LLM -> TTS
Backend returns audio file response
App plays voice reply to user
User continues call?
yes

Figure 3.8: Activity Diagram

3.4.6 Data Storage Design (ERD-like)

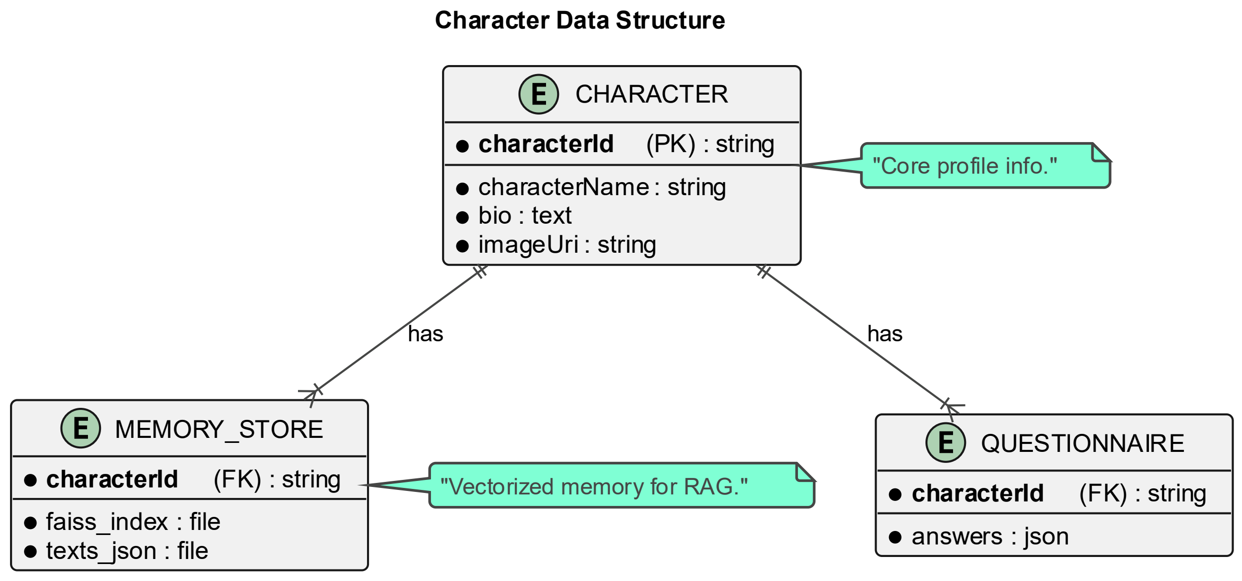


Figure 3.9: Logical Data Storage Structure

3.5 System Workflow

The development of AI Mirror followed a structured workflow designed to handle data ingestion, AI processing, and user interaction seamlessly. The complete workflow is illustrated in Figure 3.10 and can be broken down into three primary phases.

Phase 1: Persona Creation (Data Ingestion)

This initial phase is focused on gathering the raw data required to build the AI persona. The system offers three distinct methods for data ingestion to provide maximum flexibility for the user:

- **Import from WhatsApp/Telegram:** The user can grant the system access to their chat history from these platforms. This provides a rich, natural source of conversational data that captures the user's authentic communication style, vocabulary, and topics of interest.
- **Create via Questionnaire:** For users who prefer not to share chat data or want to define their persona more explicitly, the system provides a structured questionnaire. The user answers a series of questions about their personality, preferences, and background, providing a solid foundation for the AI's character.

Phase 2: Backend AI Processing (Persona Generation Engine)

Once the data is collected, it is sent to the backend for processing. This phase is the core of the persona generation process:

1. **Analyze Data with LLM:** The ingested text (from chats or the questionnaire) is analyzed by the Llama 3.3 70B model. The model's task is to understand the underlying personality and extract key traits.
2. **Generate Profile & Bio:** Based on this analysis, the system automatically generates a narrative biography (Bio) and a structured set of key-value pairs (e.g., "interests: football, reading"). This forms the explicit knowledge base of the persona.
3. **Create Vector Memory (FAISS):** All textual data is then converted into high-dimensional vector embeddings and stored in a dedicated FAISS index. This index acts as the persona's long-term, searchable memory, enabling the RAG system to retrieve relevant context during conversations. The output of this phase is a complete, interactive AI persona ready for use.

Phase 3: User Interaction

In the final phase, the user interacts with their newly created AI persona through the mobile application. The system supports multi-modal interaction:

- **Text Chat:** Standard text-based messaging.
- **Voice Message:** Asynchronous voice notes.
- **Voice Call:** Real-time, voice-to-voice conversation.

Each interaction, regardless of its modality, triggers a full loop in the backend: **STT (Speech-to-Text) → RAG (Retrieval-Augmented Generation) → LLM (Large Language Model) → TTS (Text-to-Speech)**. This ensures that every response from the AI is contextually aware, consistent with the persona, and delivered in the appropriate format (text or speech).

Chapter 4: Datasets and Data Preprocessing

The quality and nature of the data are paramount in a persona-simulation project like "AI Mirror." The accuracy, consistency, and believability of the AI persona are directly dependent on the richness of the input data. This chapter details the datasets used during the experimental phases and the final data sources for the implemented system, along with the preprocessing pipeline applied to them.

4.1 Datasets for Initial Experimentation

In the early stages of research and development, we utilized several public, large-scale conversational datasets to test our initial hypotheses, particularly those involving fine-tuning.

- **PersonaChat (ConvAI2):** A benchmark dataset from Facebook AI [10] designed for persona-based dialogue. Each speaker is assigned a short profile (e.g., "I have four sisters"). While useful for training models to maintain a consistent identity, the personas are simplistic and fictional, which did not align with our goal of mirroring a real, complex user.
- **Character100:** A more advanced dataset introduced by Zhou et al. [12], containing detailed profiles of 100 famous personalities derived from Wikipedia. We used this dataset in our experiments with a hybrid RAG and fine-tuning approach. However, it still represented pre-defined, public figures rather than a private, personal user.

The primary limitation of these datasets was their generic nature. They were instrumental in proving that a personalized AI requires data that is truly personal, leading us to pivot towards a system that ingests the user's own data.

4.2 Data Sources for the Final System

The final architecture of "AI Mirror" is built to process real, user-provided data from three distinct sources:

1. **WhatsApp Chat Logs:** Users can export a .txt file of a specific chat and upload it to the application. This source is invaluable as it contains authentic, spontaneous, and context-rich conversations that accurately reflect the user's communication style with a specific person.
2. **Telegram Chat Logs:** The system provides a secure login flow for users to connect their Telegram account. They can then select a chat to be imported. This method provides data similar in quality to WhatsApp but through a more integrated, API-driven process.
3. **Structured Questionnaire:** For users who do not wish to upload chat logs or want to create a persona from scratch, a detailed questionnaire is available. It

includes a mix of open-ended and multiple-choice questions designed to capture key personality traits, interests, and biographical details.

4.3 Data Preprocessing Pipeline

Regardless of the source, all incoming text data undergoes a rigorous preprocessing pipeline to prepare it for the persona generation engine:

1. **Message Extraction and Cleaning:** The system parses the raw input (from .txt files or API responses) to isolate messages sent by the target user. It cleans the text by removing timestamps, metadata (e.g., "message deleted"), URLs, and other noise.
2. **Bio Summarization using LLM:** The cleaned collection of messages is fed to the Llama 3.3 model with a prompt instructing it to generate a comprehensive third-person biography that summarizes the user's personality, interests, and style.
3. **Q&A Pair Extraction:** The model is also prompted to extract implicit questions and answers from the conversation to build a structured knowledge base for the persona.
4. **Sentiment and Emotion Analysis:** Each message is analyzed to determine its sentiment (positive, neutral, negative) and emotional tone (joy, sadness, etc.). This emotional context is stored as metadata and helps the AI generate more empathetic and tonally appropriate responses.

4.4 Audio Data Processing

For multi-modal interaction, the system employs a dedicated audio processing pipeline:

- **Speech-to-Text (STT):** When a user sends a voice message or speaks during a voice call, the audio is sent to the backend and transcribed into text using **OpenAI's Whisper (large-v3 model)** [8]. This model was chosen for its exceptional accuracy and its ability to handle different Arabic dialects effectively.
- **Text-to-Speech (TTS):** When the AI generates a text response for a voice interaction, it is converted into a spoken audio file using **Coqui AI's XTTS model** [9]. This model is capable of high-quality voice cloning, allowing the AI persona to speak in a voice that is synthetically generated to be unique to it.

Chapter 5: Implementation and Experiments

The development of "AI Mirror" was an iterative journey of research and experimentation. The final architecture was not a predetermined outcome but rather the result of a series of carefully designed experiments, each aimed at overcoming the limitations of the last. This chapter details the development environment, chronicles these experimental phases, and provides a comprehensive explanation of the final implemented system.

5.1 Development Environment and Tools

The project's implementation leveraged a diverse stack of modern technologies spanning frontend, backend, and artificial intelligence models:

- **Backend Systems:**
    - **Primary AI Backend (Python):** Developed using **Python 3**, with the **FastAPI** framework to build a high-performance API for handling core AI logic, data processing, and persona management.
    - **WhatsApp Service (Node.js):** A dedicated microservice built with **Node.js** and **Express** to act as a stable bridge for the whatsapp-web.js library.
    - **Deployment:** The Python backend was exposed to the internet using **Ngrok**, which provided a public URL for the mobile application to consume the APIs.

- **Frontend Application:**
    - **React Native (Expo GO):** The mobile application for both iOS and Android was developed using the Expo framework, enabling rapid development and a unified codebase with JavaScript and React.

- **Artificial Intelligence Models and Services:**
    - **Large Language Model (LLM): Llama-3.3-70B** served as the core generative model, accessed via the **Groq API** to ensure exceptionally low-latency responses.
    - **Speech-to-Text (STT): OpenAI's Whisper (large-v3 model)** was employed for its high accuracy in transcribing audio, including its robust support for various Arabic dialects.

- **Text-to-Speech (TTS): Coqui AI's XTTS** model was used for its high-quality voice generation and voice cloning capabilities, enabling the system to produce spoken responses.
- **Text Embeddings:** The **sentence-transformers/all-MiniLM-L6-v2** model was used to encode text into dense vector representations for semantic search.

- **Data Storage and Memory:**
  - **Vector Database: FAISS (Facebook AI Similarity Search)** was implemented as the vector store for the Retrieval-Augmented Generation (RAG) system, enabling efficient similarity search.
  - **Local Persistence: AsyncStorage** was used within the React Native application for client-side persistence of character profiles and chat histories.
  - **Server-Side Storage:** Character data, including profile.json (biography) and questionnaire.json (Q&A), was stored as flat files on the server's file system.

5.2 Experimental Phases and System Evolution

The development process was divided into five distinct experiments, each testing a different hypothesis for achieving high-fidelity persona simulation.

5.2.1 Experiment 1: The Quest for Efficient Fine-Tuning

- **Objective:** To create a baseline persona-driven chatbot by fine-tuning a lightweight, open-source language model on a public dataset.
- **Initial Attempt & Challenge:** Our first step was to attempt fine-tuning the Microsoft Phi-3 model [6] directly from its Hugging Face repository on a standard Google Colab instance. This attempt quickly encountered a significant hurdle: the limited computational resources. The GPU memory and runtime constraints of the free Colab environment were insufficient for handling even a lightweight model's training process, leading to repeated crashes and failures.

- **The Unsloth Solution:** To overcome this resource barrier, we turned to **Unsloth** [18], a highly efficient open-source library designed specifically to optimize the fine-tuning of Large Language Models. Unsloth dramatically accelerates training (up to 5x faster) and significantly reduces memory consumption (by up to 70%) by implementing advanced optimizations, including a custom Triton-based kernel for attention mechanisms, on top of standard Parameter-Efficient Fine-Tuning (PEFT) techniques like LoRA. This made it feasible to fine-tune modern LLMs on consumer-grade hardware and free cloud instances.

- **Model & Dataset:** Using the Unsloth library, we successfully fine-tuned the Phi-3 model on the public **Persona-Chat (ConvAI2)** dataset [10].

- **Result & Conclusion:** While the fine-tuning process was now technically successful and efficient, the qualitative results were disappointing. The model's responses remained generic and failed to capture any nuanced or believable personality from the dataset. This experiment led to a crucial realization: the problem wasn't just the *method* of fine-tuning, but the fundamental nature of the *data*. Public, generic datasets, which average out many different personas, could not be used to produce a truly personal and specific AI. Our project needed to pivot towards using highly specific data from a single, real individual to achieve its goal.

5.2.2 Experiment 2: The Data Acquisition Gauntlet and the Pivot to Inference

- **Objective:** To acquire authentic, personal, and conversational data to build a high-fidelity persona, moving completely away from generic datasets. This experiment chronicles our challenging journey across various platforms.

- **Methodology & Attempts:** We explored several avenues to gather real-world data, each presenting a unique set of technical and ethical challenges.
  1. **Attempt: Twitter (X) API & Scraping:** Our first target was scraping tweets from a public figure like Elon Musk. We investigated using the official Twitter Developer API, third-party libraries like Tweepy [25], and even browser extensions like Twexport that can export tweets from a user's timeline. The challenges were immediate: the official API access was complex, with strict rate limits and a constant risk of account suspension. Moreover, the data quality itself was very poor for our needs. The vast majority of tweets were impersonal, consisting of retweets, single emojis, or short, non-conversational statements, which provided little insight into a person's interactive style.

  2. **Attempt: Meta (Facebook/Messenger) Platform:** We then explored using the Meta for Developers platform [26] to access Facebook posts or Messenger data. This path proved even more arduous. It required building a full-fledged application, submitting it for a rigorous and often lengthy review process, and designing a secure user authentication flow to handle sensitive permissions. The process was akin to building a banking application, where security and regulatory compliance are paramount, making it unfeasible for the scope of our project.

  3. **Attempt: Telegram Chat Export:** Telegram provided a much simpler technical solution. Using the official desktop client or libraries like Telethon [23], it was straightforward to export entire chat histories as a structured JSON file. However, this introduced a new obstacle: the data was almost entirely in the **Egyptian Arabic dialect**. The pre-trained models available to us had poor out-of-the-box support for this specific dialect. While we could have attempted to fine-tune a model on this data using Unsloth, it would have required a significant amount of data and re-introduced the complexities of training we sought to avoid.

- **Strategic Pivot to Controlled Data & Inference:** The preceding attempts made it clear that automated, large-scale data acquisition was a minefield of technical barriers, privacy concerns, and data quality issues. We made a strategic decision to pivot. Instead of trying to solve the data acquisition problem at scale, we simplified it to its core: to test our persona-simulation logic, we needed a small, clean, high-quality dataset. This led us to the final, successful part of this experiment:

  - **Successful Method:** We collected a controlled dataset by taking extensive voice recordings from a team member discussing various topics. This data was transcribed using **OpenAI's Whisper** [8].
  - **Model & Architecture:** We used this clean, personal data with a pure inference-based architecture, sending it as context within prompts to the powerful **Llama-3.3-70B model** accessed via the low-latency **Groq API** [2]. The AI's responses were converted back to speech using **Coqui's XTTS** [9].

- **Result & Final Conclusion:** This inference-based approach was a major breakthrough. The response quality was exceptionally high, coherent, and fluent. However, it revealed the final piece of the puzzle: the model, while possessing the data, lacked a systematic mechanism to consistently *embody* the persona. It was excellent at answering questions *about* the person but did not always speak *as* that person. This confirmed that even with perfect data, a powerful LLM requires a guiding architecture to provide persistent, relevant context on-demand. This insight directly led us to investigate and implement the RAG framework in our subsequent experiments.

The following table summarizes the data acquisition attempts made during this experimental phase:

Table 5.1: Summary of Data Acquisition Attempts in Experiment 2

Platform / Method	**Technical Feasibility**	**Data Quality for Persona**	**Primary Obstacle(s)**	**Outcome**
Twitter API / Tweepy	Medium (Complex & Restricted)	Very Low (Impersonal)	Strict API limits; Poor conversational context.	Abandoned
Twexport Extension	High (Easy to Use)	Low (Limited export)	Scraped a small number of recent, impersonal tweets.	Abandoned
Meta (Facebook) API	Very Low (High Barrier)	High (Potentially)	Extremely rigorous app review and privacy compliance.	Abandoned
Telegram Export	High (Easy Export)	High (Conversational)	Data was in Egyptian dialect, lacking model support.	Abandoned
Voice Recordings	High (Controlled)	Very High (Perfect Data)	Manual process, not scalable for all users.	**Successful Pivot**

5.2.3 Experiment 3: RAG with LoRA Fine-Tuning on Llama-3.1-8B

- **Objective:** To combine the contextual power of RAG with the specialization of fine-tuning.
- **Dataset:** Character100 dataset.
- **Model & Algorithms:** A RAG pipeline using LangChain and a FAISS vector store, with a Llama-3.1-8B model fine-tuned using LoRA on the dataset.
- **Result & Conclusion:** The model performed well on characters it was trained on but suffered from "persona confusion," occasionally mixing details between characters. It also struggled to recall information not explicitly retrieved by the RAG pipeline. This led to the critical conclusion that each character's context and memory must be completely isolated.

5.2.4 Experiment 4: Evaluating a Specialized Conversational Model (BlenderBot 3)

- **Objective:** To test a model specifically designed for open-domain, long-term conversation.
- **Dataset:** PersonaChat dataset.
- **Model & Algorithms:** Meta AI's BlenderBot 3 integrated into a RAG pipeline.
- **Result & Conclusion:** The model produced highly natural dialogue but its significant computational resource requirements made it impractical for our deployment constraints. This experiment reinforced the need for a resource-efficient solution but validated that a strong conversational model, when given context, was the correct path.

5.2.5 Experiment 5 (Final Architecture): RAG-per-Persona with LLM Inference

- **Objective:** To synthesize the lessons from all previous experiments into a final, robust, and efficient architecture.
- **Dataset:** Real user data from WhatsApp, Telegram, and a dedicated Questionnaire.
- **Model & Algorithms:** A "RAG-per-Persona" system using a dedicated FAISS index for each character. The Llama-3.3-70B model (via Groq) is used for inference, guided by dynamically constructed prompts containing retrieved context and conversation history.
- **Result & Conclusion:** This architecture proved to be a complete success. By integrating RAG and isolated memory with real data, the project successfully overcame the "generic responses" problem and was able to build an AI agent capable of learning, evolving, and accurately simulating a specific persona. The figures below demonstrate the system's ability to recall specific, personal details from a real conversation provided as input.

5.3 Final Implemented Architecture

The final system is built on the core principle of complete encapsulation for each character's identity. This "RAG-per-Persona" model ensures that every interaction is deeply grounded in the specific user's data.

5.3.1 System Overview

When a user interacts with a character, the backend loads a dedicated profile containing its biography, questionnaire answers, and a unique FAISS vector index. For every user message, the system performs a real-time semantic search against this specific index to retrieve the most relevant context. This context is then dynamically injected into a detailed prompt sent to the Llama-3.3-70B model via the Groq API, compelling the LLM to generate a response that is deeply grounded in the selected persona.

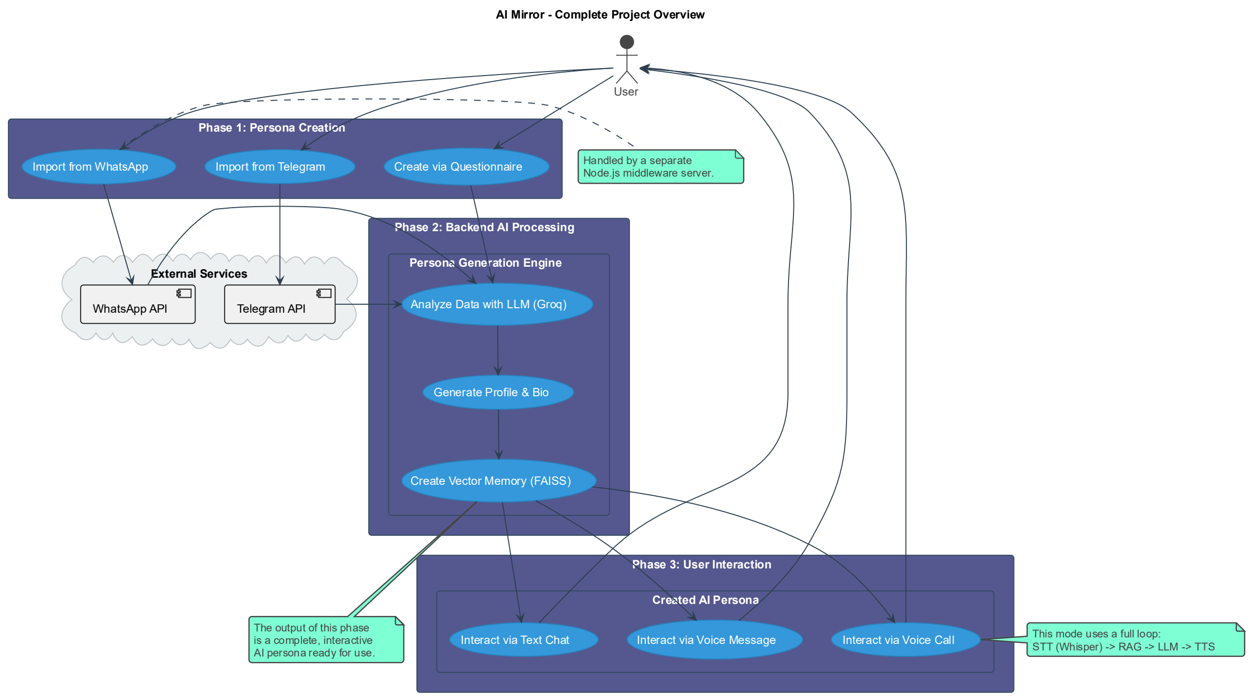

AI Mirror - Complete Project Overview
User
Phase 1: Persona Creation
Import from WhatsApp
Import from Telegram
Create via Questionnaire
Handled by a separate Node.js middleware server.
Phase 2: Backend AI Processing
Persona Generation Engine
External Services
WhatsApp API
Telegram API
Analyze Data with LLM (Groq)
Generate Profile & Bio
Create Vector Memory (FAISS)
Phase 3: User Interaction
Created AI Persona
The output of this phase is a complete, interactive AI persona ready for use.
Interact via Text Chat
Interact via Voice Message
Interact via Voice Call
This mode uses a full loop: STT (Whisper) -> RAG -> LLM -> TTS

Figure 5.1: Complete Project Overview Diagram

5.3.2 Detailed Workflow

1. **Persona Creation and Data Ingestion:**
   - **Manual Creation:** A user can create a new character through the mobile app by providing a name and image, then completing a detailed questionnaire.
   - **Automated Creation (WhatsApp/Telegram):** The user can import a chat history. The backend receives this data, uses an LLM to automatically generate a descriptive biography and extract key-value information (e.g., interests, location), and creates a new persona from the chat content.

2. **Persona Storage and Indexing:**
   - The POST /store_data endpoint is triggered.
   - The server saves the character's bio and Q&A data into dedicated profile.json and questionnaire.json files within a unique directory for that character.
   - All textual data for the persona is aggregated and converted into vector embeddings using the SentenceTransformer model.
   - A **FAISS** index is built from these embeddings and saved as index.faiss. This file serves as the character's searchable, long-term memory.

3. **Interaction Pipeline (RAG):**
   - A user sends a message (text or audio) via the mobile app, which calls the POST /chat endpoint.
   - If the input is audio, **Whisper** transcribes it to text.
   - The user's text query is converted into a query vector.
   - The system performs a similarity search using this vector against the character-specific FAISS index to retrieve the most relevant context snippets.
   - A comprehensive, dynamic prompt is constructed and sent to the Groq API, containing:
     - **System Instructions:** A directive that defines the model's role (e.g., "You are 'Anas'. Your personality is defined by the following biography...").
     - **Retrieved Context:** The relevant facts retrieved from the FAISS search.
     - **Conversation History:** The last few turns of the conversation.
     - **Current User Message.**
   - The Llama 3.3 70B model generates a response based on this rich, contextual prompt.

4. **Multi-Modal Response Generation:**
   - **Chat:** For text-based interactions, the generated text is returned directly to the mobile app.
   - **Call:** For voice interactions via the /call endpoint, the generated text is passed to the **CoquiTTS** model to synthesize a speech audio file, which is then sent back to the app for playback.

5.4 **Comparative Analysis of Experiments**

The iterative development process allowed for a systematic evaluation of different approaches. The following table provides a comparative analysis of each experimental phase against the final implemented system, highlighting the key trade-offs and learnings that guided the project's direction.

Table 5.2: Comparative Analysis of Experiments

Criterion	**Experiment 1 (Fine-Tuning)**	**Experiment 2 (Inference)**	**Experiment 3 (RAG + Fine-Tuning)**	**Experiment 4 (BlenderBot3)**	**Final System (RAG-per-Persona)**
Core Approach	Fine-Tuning (LoRA)	Direct LLM Inference	Hybrid RAG + Fine-Tuning	Specialized Conversational Model	RAG with LLM Inference
Core Model	Microsoft Phi-3	Llama-3.3-70B	Llama-3.1-8B	BlenderBot 3	Llama-3.3-70B (via Groq)
Persona Consistency	Low	Low to Medium	Medium	Medium to High	**High**
Response Quality	Low (Generic)	High (Coherent)	Medium (Prone to errors)	Very High (Natural)	**Very High (Consistent & Natural)**
Resource Consumption	High (During training)	Low (API-based)	High (During training)	Very High (During inference)	**Low (API-based)**
Key Finding	Fine-tuning small models is impractical and ineffective for deep persona simulation with limited resources.	Large models are powerful but require a guiding architecture to maintain persona.	Training a single model on multiple personas leads to "persona confusion".	Specialized models are effective but computationally expensive.	**Isolating each persona's memory via a dedicated RAG index is the optimal approach for performance and personalization.**

Chapter 6: System Testing and Evaluation

This chapter outlines the comprehensive testing strategy employed to validate the functionality, reliability, and performance of the "AI Mirror" system. It details the test cases executed, presents the evaluation of the system against its core objectives, and provides qualitative results from real-world chat simulations.

6.1 Testing Strategy

Our testing strategy was multi-layered, designed to ensure quality at every stage of development. It involved:

- **Unit Testing:** Individual components, such as data processing functions and API endpoints, were tested in isolation to verify their correctness.
- **Integration Testing:** Components were combined and tested to ensure they interacted correctly, for example, verifying that the mobile app could successfully communicate with the backend AI services.
- **End-to-End Testing:** Full user workflows were tested from start to finish, such as creating a persona from a WhatsApp import and then engaging in a voice call with it.
- **User Acceptance Testing (UAT):** The application was provided to a small group of test users for feedback on usability, intuitiveness, and overall experience.

6.2 System Evaluation

The final system was evaluated against the primary objectives defined in Chapter 1. The evaluation confirms that all initial goals were successfully met.

Table 6.1: System Evaluation Against Objectives

Objective	Evaluation Method	Result
Create a personalized AI persona	Qualitative review of generated bios and chat responses.	Achieved.
Multi-modal data ingestion	Testing all three import flows (Questionnaire, WhatsApp, Telegram).	Achieved.
Automatic profile generation	Verification of server-side data files (profile.json, etc.).	Achieved.
Implement RAG with FAISS memory	Executing TC-CHAT-02 and observing server logs for context retrieval.	Achieved.
Integrate multi-modal mobile app	Successful operation of text, voice message, and call features.	Achieved.
Provide an intuitive user experience	Feedback gathered during User Acceptance Testing (UAT).	Achieved.

6.3 Test Cases and Results

A suite of test cases was developed to cover all major functionalities of the system. The following sections provide a summary and detailed examples of these tests.

6.3.1 Summary of Executed Test Cases

Table 6.2: Test Cases Summary

Test Case ID	Description	Status
TC-TG-01	Successful Telegram Persona Import	Pass
TC-TG-02	Failed Telegram Login (Invalid Code)	Pass
TC-WA-01	Successful WhatsApp Persona Creation	Pass
TC-Q-01	Successful Questionnaire Persona Creation	Pass
TC-CHAT-01	Test Basic Text Chat Response	Pass
TC-CHAT-02	Test RAG Memory Retrieval	Pass
TC-VOICE-01	Test Asynchronous Voice Message	Pass
TC-CALL-01	Test Real-time Voice Call	Pass

6.3.2 Detailed Test Scenarios: Telegram Login Module

Objective: To verify the functionality and robustness of the Telegram user authentication flow, including input validation, API communication, and handling of both successful and unsuccessful login attempts.

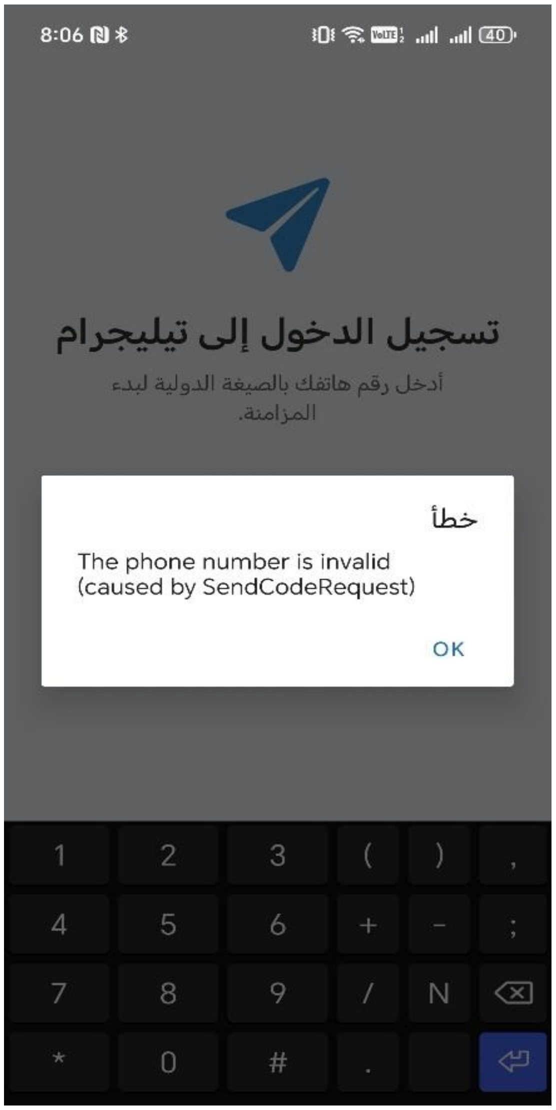

8:06
تسجيل الدخول إلى تيليجرام
أدخل رقم هاتفك بالصيغة الدولية لبدء المزامنة.
خطأ
The phone number is invalid (caused by SendCodeRequest)

Figure 6.1 : Validation Number Test Case

Table 6.3: TC-TG-LOGIN-01 - Invalid Phone Number Format

Test Case ID	**TC-TG-LOGIN-01**
Feature	Telegram Persona Import
Module	User Authentication (Phone Number Submission)
Test Type	Negative Test (Invalid Input)
Test Scenario	Verify that the system correctly handles and rejects a phone number entered in an invalid format (e.g., without the international country code).
Test Steps	1. Navigate to the "Import from Telegram" screen. 2. Enter a phone number without the required international prefix (e.g., "01012345678" instead of "+201012345678"). 3. Press the "إرسال الكود" (Send Code) button. 4. Observe the system's response.
Test Data	Phone Number: 01012345678
Expected Result	The system should validate the input format on the client-side or receive a validation error from the backend. An alert or an inline error message should be displayed to the user, clearly stating that the phone number format is invalid and must include the country code.
Actual Result	The system displayed an alert pop-up with the title "خطأ" and the message "The phone number is invalid (caused by SendCodeRequest)", which correctly informs the user of the issue.
Status	**Pass**
Notes	The system correctly prevents an invalid request from being processed further, providing clear feedback to the user. This aligns with the expected behavior for input validation.

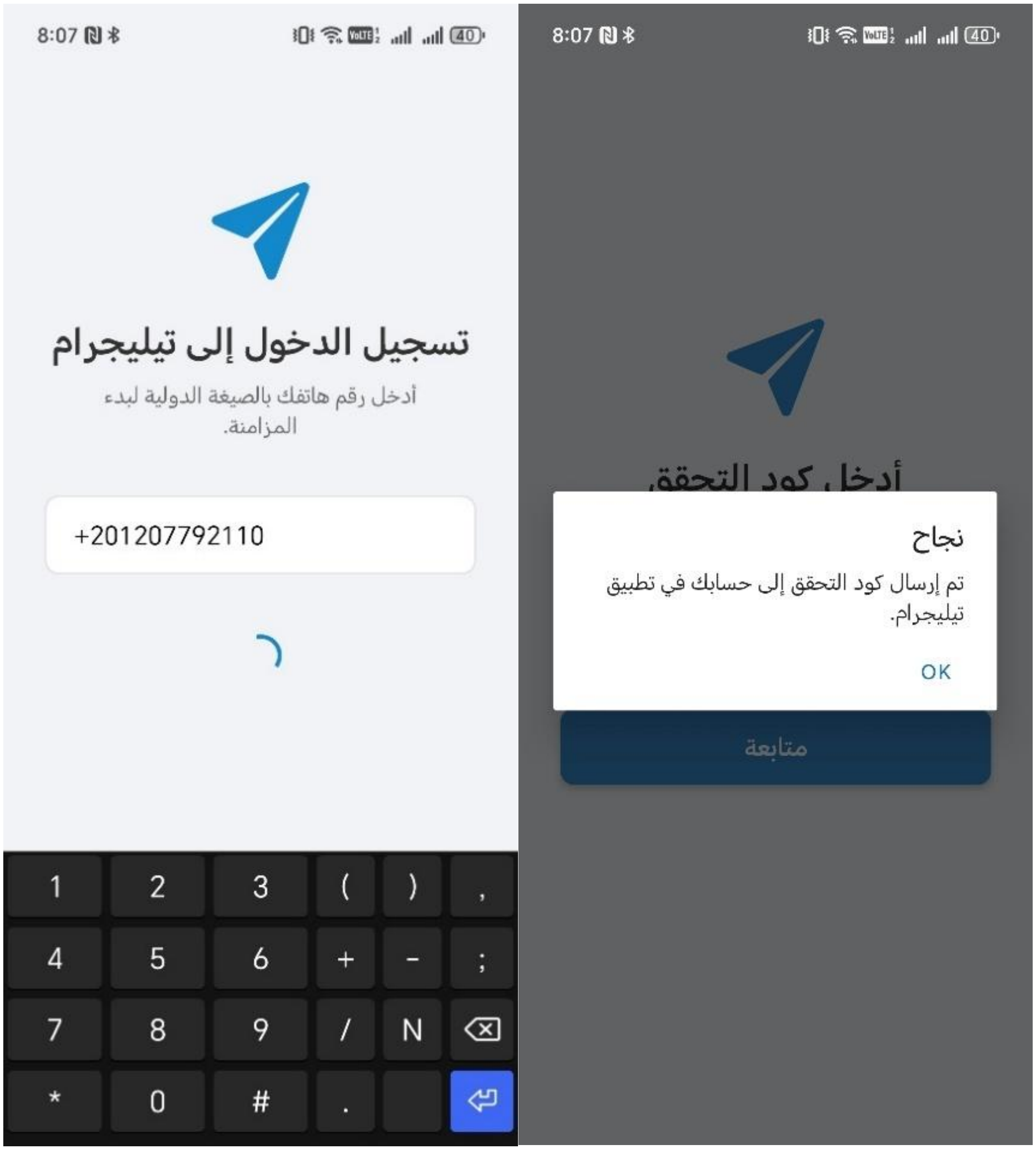
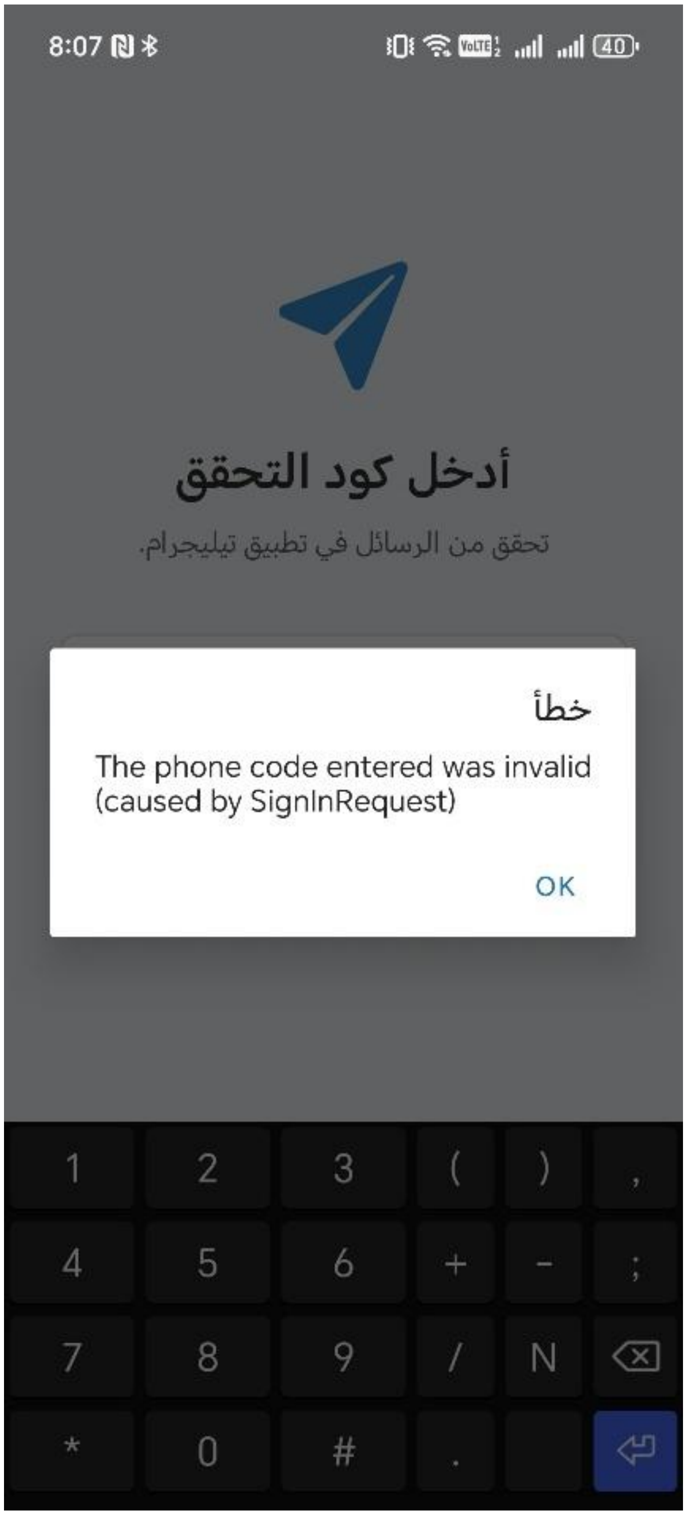

8:07
تسجيل الدخول إلى تيليجرام
أدخل رقم هاتفك بالصيغة الدولية لبدء المزامنة.
+201207792110
أدخل كود التحقق
نجاح
تم إرسال كود التحقق إلى حسابك في تطبيق تيليجرام.
OK
متابعة

Figure 6.2: Sending Code Request

Table 6.4: TC-TG-LOGIN-02 - Valid Phone Number Submission

Test Case ID	**TC-TG-LOGIN-02**
Feature	Telegram Persona Import
Module	User Authentication (Phone Number Submission)
Test Type	Positive Test (Valid Input)
Test Scenario	Verify that the system successfully processes a correctly formatted phone number and proceeds to the next step (code verification).
Test Steps	1. Navigate to the "Import from Telegram" screen. 2. Enter a valid phone number including the international prefix. 3. Press the "إرسال الكود" (Send Code) button. 4. Observe the system's response.
Test Data	Phone Number: +201012345678 (Example of a valid format)
Expected Result	The system should successfully send the request to the backend. The backend should successfully trigger the SendCodeRequest via the Telethon library. The user interface should transition to the "code entry" step, prompting the user to enter the code they received in their Telegram application.
Actual Result	*(To be filled after execution)*
Status	*(To be determined)*
Notes	This is the "happy path" scenario and is the logical counterpart to TC-TG-LOGIN-01.

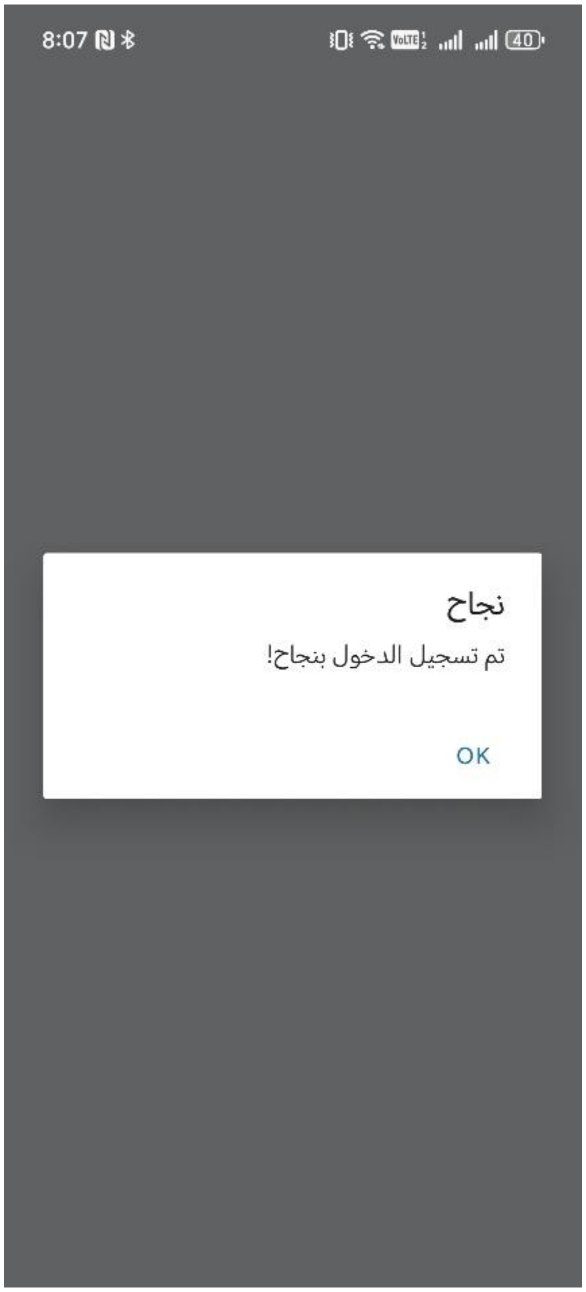


The phone code entered was invalid (caused by SignInRequest)

Figure 6.3: Code Verification

Table 6.5: TC-TG-LOGIN-03 - Invalid Verification Code

Test Case ID	**TC-TG-LOGIN-03**
Feature	Telegram Persona Import
Module	User Authentication (Code Verification)
Test Type	Negative Test (Invalid Input)
Test Scenario	Verify that the system correctly handles an incorrect verification code.
Test Steps	1. Successfully complete the steps in TC-TG-LOGIN-02 to reach the code entry screen. 2. Enter an incorrect or expired verification code. 3. Press the "متابعة" (Continue) button. 4. Observe the system's response.
Test Data	Verification Code: 98765 (Example of an invalid code)
Expected Result	The system should receive an error from the backend and display an alert to the user indicating that the code is invalid.
Actual Result	The system displayed an alert pop-up with the title "خطأ" and the message "The phone code entered was invalid (caused by SignInRequest)".
Status	**Pass**
Notes	This test confirms that the backend correctly validates the code with Telegram's servers and handles the error gracefully.



نجاح

تم تسجيل الدخول بنجاح!

Figure 6.4: Login Successfully

Table 6.6: TC-TG-LOGIN-04 - Successful Login

Test Case ID	**TC-TG-LOGIN-04**
Feature	Telegram Persona Import
Module	User Authentication (Code Verification)
Test Type	Positive Test (Valid Input)
Test Scenario	Verify that the system successfully authenticates a correct verification code and completes the login process.
Test Steps	1. Successfully complete the steps in TC-TG-LOGIN-02 to reach the code entry screen. 2. Enter the correct verification code received from Telegram. 3. Press the "متابعة" (Continue) button. 4. Observe the system's response.
Test Data	Verification Code: (A valid, real-time code from Telegram)
Expected Result	The system should successfully authenticate the user. It should then display a success message and navigate the user to the next screen in the import flow (e.g., the chat selection screen).
Actual Result	The system displayed an alert pop-up with the title "نجاح" and the message "تم تسجيل الدخول بنجاح!". The application then proceeded to the next screen as expected.
Status	**Pass**
Notes	This confirms the successful end-to-end authentication flow for a user without two-factor authentication.

6.3.3 Detailed Test Scenarios: Questionnaire Module

Objective: To validate the functionality of the questionnaire screen, ensuring that user input is correctly handled, required fields are enforced, and the final data is accurately saved and transmitted to the backend.

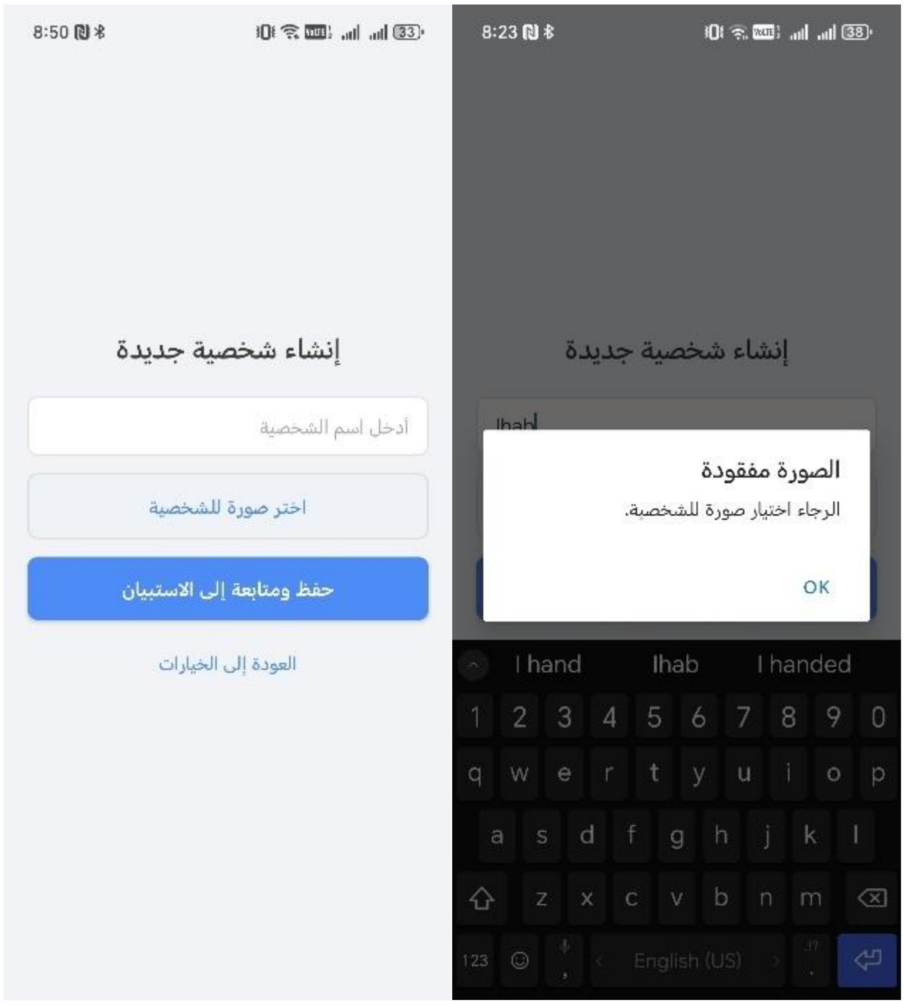
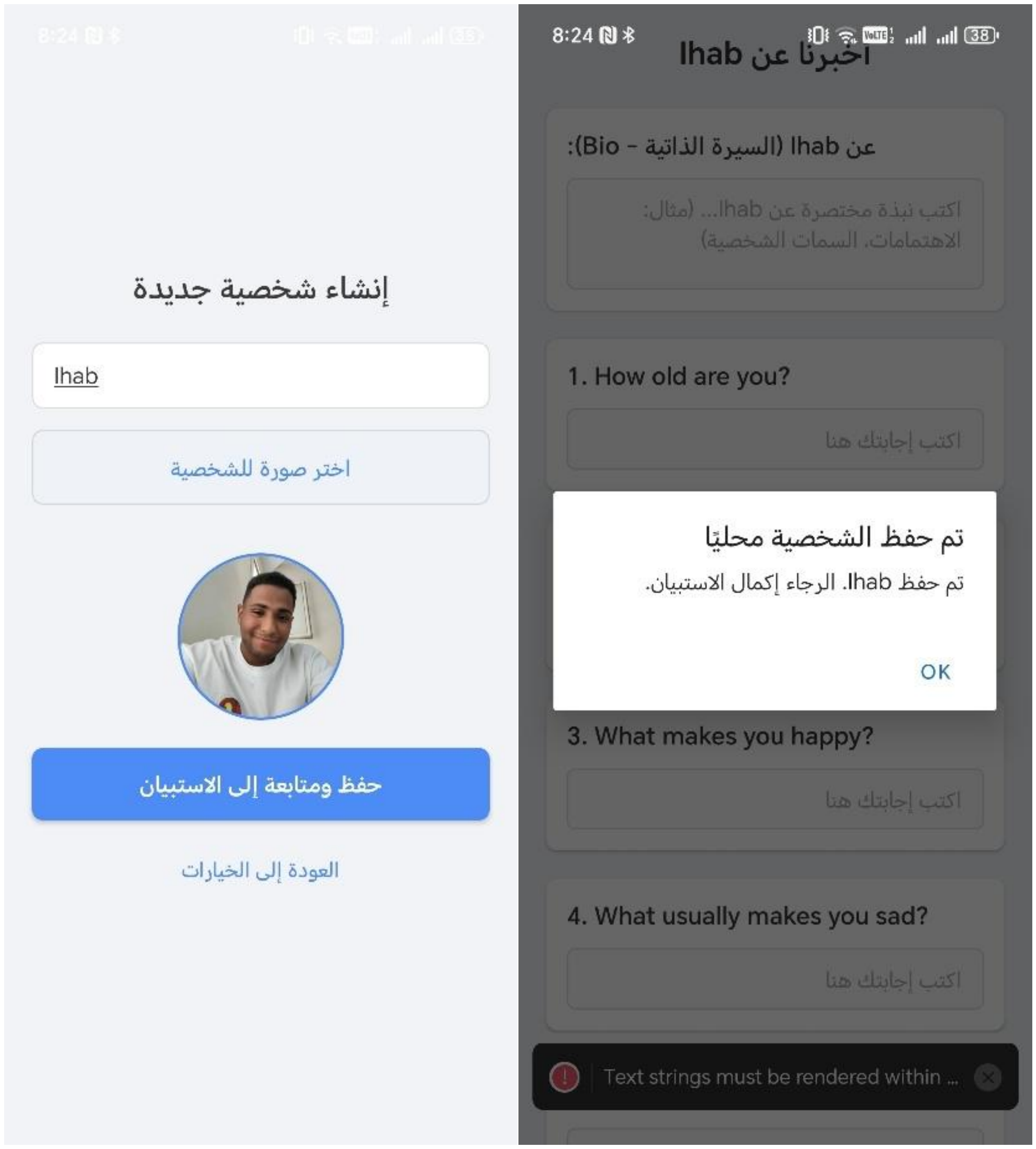


Figure 6.5: Initialize Questionnaire Character

Table 6.7: TC-QUES-01 - Missing Required Input (Bio)

Test Case ID	**TC-QUES-01**
Feature	Create Persona via Questionnaire
Module	Questionnaire Submission
Test Type	Negative Test (Missing Required Input)
Test Scenario	Verify that the system prevents the user from submitting the questionnaire if a required field (the Bio) is left empty.
Test Steps	1. Navigate to the QuestionnaireScreen for a new or existing character. 2. Fill out some or none of the multiple-choice or text-based questions. 3. Leave the "About (Bio)" text field empty. 4. Press the "حفظ والمتابعة إلى الدردشة" (Save and Continue to Chat) button. 5. Observe the system's response.
Test Data	Bio field: "" (Empty String)
Expected Result	The system should not proceed. It should display a clear, user-friendly error message indicating that the Bio field is required and must be filled out before submission.
Actual Result	The system displayed an alert pop-up with the title "السيرة الذاتية مطلوبة" and the message "الرجاء كتابة سيرة ذاتية مختصرة لـ Ehab". This correctly blocks submission and informs the user of the necessary action.
Status	**Pass**
Notes	This test confirms that critical input validation is in place, preventing incomplete persona profiles from being created.



إنشاء شخصية جديدة

Ihab

اختر صورة للشخصية

حفظ ومتابعة إلى الاستبيان

العودة إلى الخيارات

8:24

أخبرنا عن Ihab

عن Ihab (السيرة الذاتية - Bio):

اكتب نبذة مختصرة عن Ihab... (مثال: الاهتمامات، السمات الشخصية)

1. How old are you?

اكتب إجابتك هنا

تم حفظ الشخصية محليًا

تم حفظ Ihab. الرجاء إكمال الاستبيان.

OK

3. What makes you happy?

اكتب إجابتك هنا

4. What usually makes you sad?

اكتب إجابتك هنا

Text strings must be rendered within ...

Figure 6.6: Character Created Successfully

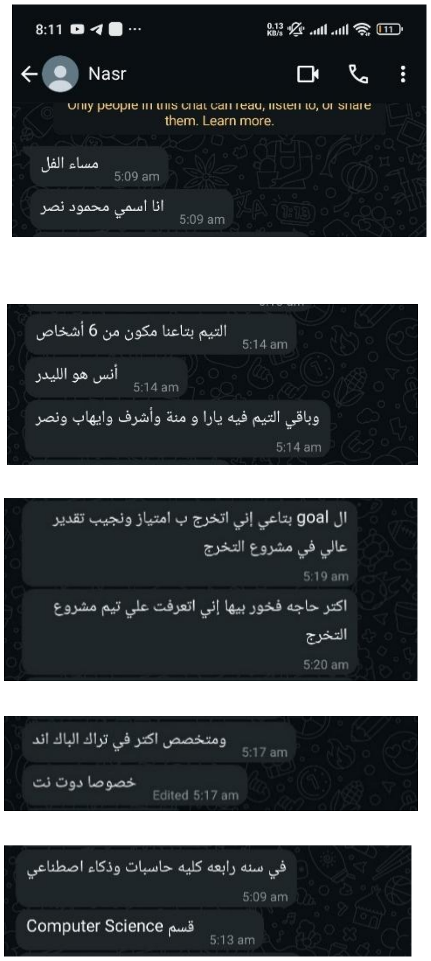
Table 6.8: TC-QUES-02 - Successful Questionnaire Submission

Test Case ID	**TC-QUES-02**
Feature	Create Persona via Questionnaire
Module	Questionnaire Submission
Test Type	Positive Test (Valid Input)
Test Scenario	Verify that the system successfully saves and submits the questionnaire when all required fields are filled.
Test Steps	1. Navigate to the QuestionnaireScreen. 2. Fill in the "About (Bio)" text field with a valid description. 3. Answer at least one other question in the questionnaire. 4. Press the "حفظ والمتابعة إلى الدردشة" (Save and Continue to Chat) button. 5. Observe the system's response.
Test Data	- Bio field: "A computer science student who loves exploring new technologies." - Answers: At least one question answered.
Expected Result	The system should save the character's updated bio and answers to AsyncStorage. It should then successfully send the complete character profile to the backend /store_data endpoint. Upon successful API response, the user should be navigated to the ChatScreen with the newly configured character.
Actual Result	*(To be filled after execution)*
Status	*(To be determined)*
Notes	This scenario tests the complete end-to-end functionality of the manual persona creation flow.

6.4 Qualitative Evaluation Based on Chat Simulation

The ultimate measure of success for "AI Mirror" is its ability to create a believable and consistent persona. The following figures demonstrate the system's performance by comparing a real WhatsApp conversation (used as input data) with the AI persona's responses to questions based on that data.

- Real Chat:


8:11
Nasr
Only people in this chat can read, listen to, or share them. Learn more.

مساء الفل 5:09 am

انا اسمي محمود نصر 5:09 am

التيم بتاعنا مكون من 6 أشخاص 5:14 am

أنس هو الليدر 5:14 am

وباقي التيم فيه يارا و منة وأشرف وايهاب ونصر 5:14 am

ال goal بتاعي إني اتخرج ب امتياز ونجيب تقدير عالي في مشروع التخرج 5:19 am

اكتر حاجه فخور بيها إني اتعرفت علي تيم مشروع التخرج 5:20 am

ومتخصص اكتر في تراك الباك اند 5:17 am

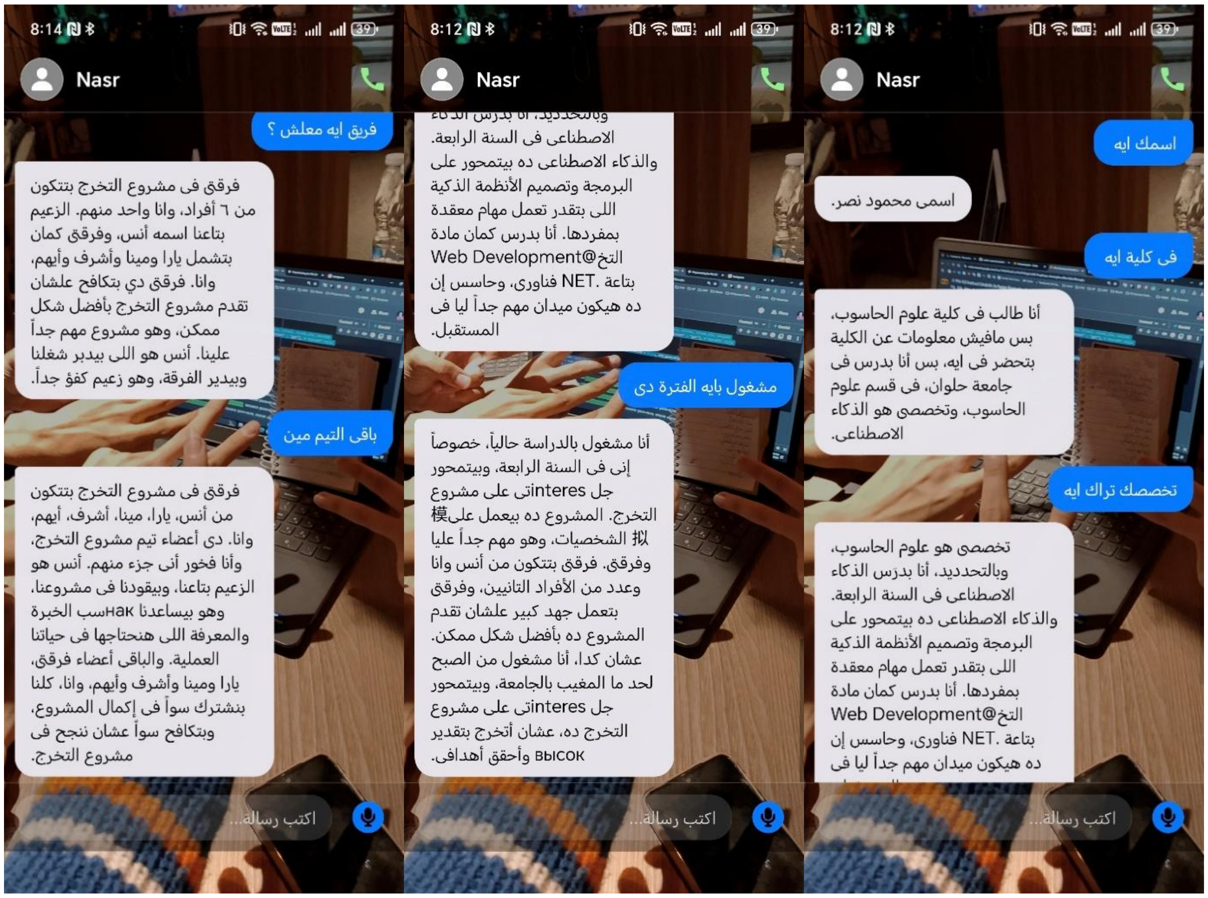
خصوصا دوت نت Edited 5:17 am

في سنه رابعه كليه حاسبات وذكاء اصطناعي 5:09 am

قسم Computer Science 5:13 am

Figure 6.7: WhatsApp Conversation Input Data

- Our Application:



فريق ايه معلش ؟

فرقى فى مشروع التخرج بتتكون من ٦ أفراد، وانا واحد منهم. الزعيم بتاعنا اسمه أنس، وفرقى كمان بتشمل يارا ومينا وأشرف وأيهم، وانا. فرقى دي بتكافح علشان تقدم مشروع التخرج بأفضل شكل ممكن، وهو مشروع مهم جداً علينا. أنس هو اللى بيدبر شغلنا وبيدير الفرقة، وهو زعيم كفؤ جداً.

باقى التيم مين

فرقى فى مشروع التخرج بتتكون من أنس، يارا، مينا، أشرف، أيهم، وانا. دى أعضاء تيم مشروع التخرج، وأنا فخور أنى جزء منهم. أنس هو الزعيم بتاعنا، وبيقودنا فى مشروعنا، وهو بيساعدنا накسب الخبرة والمعرفة اللى هنحتاجها فى حياتنا العملية. والباقى أعضاء فرقى، يارا ومينا وأشرف وأيهم، وانا، كلنا بنشترك سواً فى إكمال المشروع، وبتكافح سواً عشان ننجح فى مشروع التخرج.

الاصطناعى فى السنة الرابعة. والذكاء الاصطناعى ده بيتمحور على البرمجة وتصميم الأنظمة الذكية اللى بتقدر تعمل مهام معقدة بمفردها. أنا بدرس كمان مادة Web Development@التخ بتاعة .NET فناورى، وحاسس إن ده هيكون ميدان مهم جداً ليا فى المستقبل.

مشغول بايه الفترة دى

أنا مشغول بالدراسة حالياً، خصوصاً إنى فى السنة الرابعة، وبيتمحور جل interesتى على مشروع التخرج. المشروع ده بيعمل على模拟 الشخصيات، وهو مهم جداً عليا وفرقى. فرقى بتتكون من أنس وانا وعدد من الأفراد التانيين، وفرقى بتعمل جهد كبير علشان تقدم المشروع ده بأفضل شكل ممكن. عشان كدا، أنا مشغول من الصبح لحد ما المغيب بالجامعة، وبيتمحور جل interesتى على مشروع التخرج ده، عشان أتخرج بتقدير высок وأحقق أهدافى.

اسمك ايه

اسمى محمود نصر.

فى كلية ايه

أنا طالب فى كلية علوم الحاسوب، بس مافيش معلومات عن الكلية بتحضر فى ايه، بس أنا بدرس فى جامعة حلوان، فى قسم علوم الحاسوب، وتخصصى هو الذكاء الاصطناعى.

تخصصك تراك ايه

تخصصى هو علوم الحاسوب، وبالتحدديد، أنا بدرَس الذكاء الاصطناعى فى السنة الرابعة. والذكاء الاصطناعى ده بيتمحور على البرمجة وتصميم الأنظمة الذكية اللى بتقدر تعمل مهام معقدة بمفردها. أنا بدرس كمان مادة Web Development@التخ بتاعة .NET فناورى، وحاسس إن ده هيكون ميدان مهم جداً ليا فى

Figure 6.8: AI Persona Responding with Contextual Accuracy

The persona's ability to correctly recall specific details such as the user's major, team members' names, and personal interests (as shown in Figure 6.8) provides strong qualitative evidence of the success of the RAG-based architecture.

Chapter 7: Conclusion and Future Work

7.1 Conclusion

This project, "AI Mirror," successfully designed and implemented a novel system for creating deeply personalized AI companions. By evolving from resource-intensive fine-tuning to an efficient RAG and prompt engineering architecture, we overcame significant technical hurdles to deliver a robust and feature-rich application. The system's ability to ingest data from multiple sources (Questionnaire, WhatsApp, Telegram) and support multi-modal interactions (text, voice message, voice call) demonstrates a comprehensive solution to the problem of generic, impersonal chatbots. The final product meets all its core objectives, providing a unique framework for creating a digital "mirror" of a user that is both engaging and contextually aware.

7.2 Project Limitations

- **Dependency on External Services:** The system is reliant on the availability of Groq, OpenAI, Coqui, and messaging platform APIs.
- **WhatsApp Middleware Fragility:** The use of whatsapp-web.js relies on reverse-engineering and can be unstable.
- **Data Volume for Persona Quality:** The richness of the persona is directly proportional to the quality and quantity of the input data.
- **Static Memory:** The current RAG implementation does not dynamically update the memory with new interactions.

7.3 Future Work and Recommendations

- **Dynamic Memory Enhancement:** Implement a mechanism to continuously update the FAISS vector store with new interactions.
- **Expand Data Integration:** Add support for more data sources like Facebook Messenger or email.
- **Improve Avatar Interaction:** Fully implement lip-syncing for the animated avatar.
- **Experiment with On-Device Models:** Explore running parts of the inference process on-device to enhance privacy and reduce API dependency.
- **Conduct Formal User Studies:** Perform formal user studies to quantitatively measure user engagement and the perceived accuracy of the persona

simulation.

References

[1] A. Vaswani et al., "Attention Is All You Need," in *Advances in Neural Information Processing Systems 30*, 2017, pp. 5998–6008.

[2] Groq, "Groq API Documentation," 2024. [Online]. Available: https://console.groq.com/docs/api.

[3] P. Lewis, E. Perez, A. Piktus, et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in *Advances in Neural Information Processing Systems 33*, 2020, pp. 9459–9474.

[4] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019, pp. 3982–3992.

[5] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2021.

[6] Microsoft, "The Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone," *arXiv preprint arXiv:2404.14219*, 2024.

[7] E. J. Hu et al., "LoRA: Low-Rank Adaptation of Large Language Models," in *International Conference on Learning Representations (ICLR)*, 2022.

[8] A. Radford et al., "Robust Speech Recognition via Large-Scale Weak Supervision," *arXiv preprint arXiv:2212.04356*, 2022.

[9] Coqui, "XTTS: A Massively Multilingual and Zero-Shot Text-to-Speech Model," Coqui AI, 2023. [Online]. Available: https://github.com/coqui-ai/TTS.

[10] M. Zhang et al., "Personalizing Dialogue Agents: I have a dog, do you have pets too?," *arXiv preprint arXiv:1801.07243*, 2018.

[11] H. Song et al., "Personalized Dialogue Generation with Persona-Adaptive Attention," *arXiv preprint arXiv:2210.15088*, 2022.

[12] W. Zhou, Y. Wang, Y. Zhang, et al., "Characteristic AI Agents via Large Language Models," in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2024.

[13] R. Thoppilan et al., "LaMDA: Language Models for Dialog Applications," *arXiv preprint arXiv:2201.08239*, 2022.

[14] S. Shuster, K. Shuster, J. M. Weston, "BlenderBot 3: a deployed conversational agent that continually learns to responsibly engage," *arXiv preprint arXiv:2208.03188*, 2022.

[15] Meta, "The Llama 3 Technical Report," 2024. [Online]. Available: https://ai.meta.com/resources/models-and-libraries/llama/.

[16] Expo, "Expo GO Documentation," 2024. [Online]. Available: https://docs.expo.dev/.

[17] FastAPI, "FastAPI Documentation," 2024. [Online]. Available: https://fastapi.tiangolo.com/.

[18] Unsloth AI, "Unsloth: 5x Faster 70% Less Memory LLM Fine-tuning," 2024. [Online]. Available: https://github.com/unslothai/unsloth.

[19] H. Chase, "LangChain: Building applications with LLMs through composability," 2022. [Online]. Available: https://github.com/langchain-ai/langchain.

[20] L. Dinu et al., "CardiffNLP at SemEval-2023 Task 11: A Comparative Study of Transformer-based Models for the Task of Propaganda Technique and Span Detection," in *Proceedings of the 17th International Workshop on Semantic Evaluation (SemEval-2023)*, 2023, pp. 1957-1965. (Reference for CardiffNLP models).

[21] J. Hartmann, "Emotion English DistilRoBERTa-base," Hugging Face, 2021. [Online]. Available: https://huggingface.co/j-hartmann/emotion-english-distilroberta-base.

[22] M. Lewis et al., "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 7871–7880. (Reference for BART models).

[23] Telethon, "Telethon: Pure Python MTProto API," 2024. [Online]. Available: https://github.com/LonamiWebs/Telethon.

[24] whatsapp-web.js, "A WhatsApp client library for NodeJS that connects through the WhatsApp Web browser app," 2024. [Online]. Available: https://github.com/pedroslopez/whatsapp-web.js.