## Best Team

# STeMS Backend Services
# Software Requirements Specification

Version 4.0

**Contributors**
David Babers, Mohamed Ben Lakbir, Robson De Souza, Collin Hicks
Johnny Huynh, Benny Iko, Scott McCrillis, Jonathan Nagy, Amol Thomare

# Revision History

| Date | Version | Description | Author(s) |
|---|---|---|---|
| 06/02/2023 | 1.0 | Milestone 1 Version | Jonathan Nagy<br>Scott McCrillis<br>Johnny Huynh<br>Benny Iko |
| 6/7/2023 | 2.0 | Milestone 2 Version | Robson De Souza<br>Jonathan Nagy |
| 7/1/2023 | 2.1 | Updated Browser Extension API Sequence Diagrams | Jonathan Nagy |
| 7/9/2023 | 3.0 | Milestone 3 Version | Team B Members |
| 7/24/2023 | 4.0 | Milestone 4 Version | Team B Members |

The following project documentation will be included in the entire documentation package for this project/application:

| | Document | Version | Date |
|---|---|---|---|
| 1 | Project Management Plan (PMP) | 4.0 | 8/04/2023 |
| 2 | Software Requirements Specification (SRS) | 4.0 | 7/24/2023 |
| 3 | Technical Design Document (TDD) | 3.0 | 7/24/2023 |
| 4 | Programmers Guide (PG) | 2.0 | 7/24/2023 |
| 5 | Deployment and Operations (DevOps) | 2.0 | 8/04/2023 |
| 6 | User Guide (UG) | 1.0 | 8/05/2023 |
| 7 | Test Report (TR) | 1.0 | 8/05/2023 |

# Table of Contents

# Software Requirements Specification

## 1. Introduction

Short-term memory refers to primary or active memory involved in retention of pieces of information for a relatively short period of time, usually up to 30 seconds. This contrasts with long-term memory, which is stored indefinitely. Short-term memory loss (STML) refers to impairment of the ability of the mind to retain information in short-term memory. The most common cause of STML are neurodegenerative conditions, such as dementia and mild cognitive impairment (MCI)—the stage of cognitive decline in the spectrum between expected normal decline due to aging and Alzheimer disease. Approximately 15 to 20% of people aged 65 and older suffer from mild cognitive impairment.

The Short-Term Memory System (STeMS) software is a mobile application that hopes to alleviate some of the burden of STML, through utilization of a recording management platform along with advanced query and summarization features made possible by ChatGPT API and speaker diarization. This SRS is for the backend components of the STeMS system. Three identified use cases for this application include restaurant ordering, memory loss assistant, and forms filler through the browser extension. The STeMS software is accessed through the ConvoBuddy application and the Browser Extension. STeMS may include the Browser Extension Service (BESie), an intermediary between the Browser Extension and the STeMS system on the mobile device.

### 1.1. Purpose

This document outlines and describes the system's functional and nonfunctional requirements for the STeMS backend system. The product is an app that utilizes artificial intelligence (AI) functionality to help with STML. This document will highlight the various external behavior through use cases for this product along with assumptions, constraints, and dependencies. The product will be access by the front end of the STeMS, ConvoBuddy, and Browser Extension components to manage recordings and recording metadata, generate and manage transcripts, and use those transcripts along with ChatGPT API to extract contextual information for various tasks. The SRS document will outline the list of interactions and activities implemented by the STeMS backend system in different use case scenarios.

### 1.2. Scope

The product is an app developed in Google Flutter application development framework utilizing the Dart programming language. The ConvoBuddy app will acquire raw audio recording data and utilize voice recognition technology to produce transcripts. The app will communicate with OpenAI API (ChatGPT) which will be used as an external tool to help parse and summarize the raw data into various outputs depending on the use case. The main use cases that the SRS document will describe can be categorized into supporting service calls for recordings management, service calls for the form filling browser extension, and service calls for ChatGPT integration.

### 1.3. Definitions, Acronyms and Abbreviations

**Software Requirements Specification (SRS)** - A document that outlines the software requirements, features, and specifications.

**Artificial Intelligence (AI)** - Machine technology able to perceive, synthesize, and infer information.

**Automatic Speech Recognition (ASR) -** Functionality that converts spoken language into written text. It is also known as Speech-to-Text (STT) and computer speech recognition.

**ChatGPT** – A language model developed by OpenAI that uses deep learning techniques to generate human-like responses in natural language conversations

**ChatGPT API** – An interface provided by OpenAI for developers to integrate ChatGPT into their applications

**Dart** – A general-purpose programming language developed by Google object-oriented with strong static type system that compiles efficient native code for high-performance execution.

**Flutter** - Flutter is an open-source UI (User Interface) framework developed by Google, used for building high-performance cross-platform mobile applications that run on both Android and iOS devices, as well as web and desktop platforms, by utilizing a single codebase.

**Local Storage** – Storage provided on a user's device, sandboxed on mobile operating systems for security.

**Cloud Storage –** A means for storing data and files on the internet that allows access from any location with a connection to the internet.

**Short-Term Memory Loss (STML) -** A tendency to forget recent events or performed actions.

**Speaker Diarization** – The application of an algorithm to separate individual speakers in an audio stream, so that a speaker can be identified with each utterance in a textual transcript.

**Speech-to-Text (STT) -** Recognition and translation of spoken language into text. Also known as ASR or computer speech recognition.

**STeMS** – Short-Term Memory System is the application ecosystem that supports assisting those with short-term memory loss. This document describes the backend components of that system.

**BESie** – Browser Extension Service supports the web form filling browser extension that is part of the STeMS suite.

**App Instance Code** – A code generated by the STeMS backend system to identify ConvoBuddy application instances and used with the Browser Extension.

### 1.4. References

Amazon (n.d.). What is cloud storage? Retrieved May 27, 2023, from https://aws.amazon.com/what-is/cloud-storage/

Doty, C. (August 16, 2022). What is Speaker Diarization. Retrieved May 27, 2023, from https://blog.deepgram.com/what-is-speaker-diarization/

Flutter (n.d.). Flutter Development Documentation. Retrieved May 27, 2023, from https://docs.flutter.dev/

Cascella, M., Al Khlili, Y. (July 21, 2022). Short Term Memory Impairment. Retrieved May 27, 2023, from https://www.ncbi.nlm.nih.gov/books/NBK545136/

OpenAI (n.d.). API Overview. Retrieved May 27, 2023, from https://platform.openai.com/overview

### 1.5. Overview

The SRS document begins with providing an overview of the app by highlighting the purpose, scope, and overview. Also, the app's requirements, specifications, and scope are defined. Then, the SRS document contains requirement details with sections for outlining the functional requirements of the app through use-cases, triggers, and outputs. Lastly, supporting information is provided to provided reference to details throughout the document.

### 1.6. Overall Project Documents

This Software Requirements Specification document is part of a set of documents created to aid in developing the STeMS Backend Services and to provide artifacts with vital information for the application's ongoing support and operation throughout its full life cycle.

## 2. Overall Description

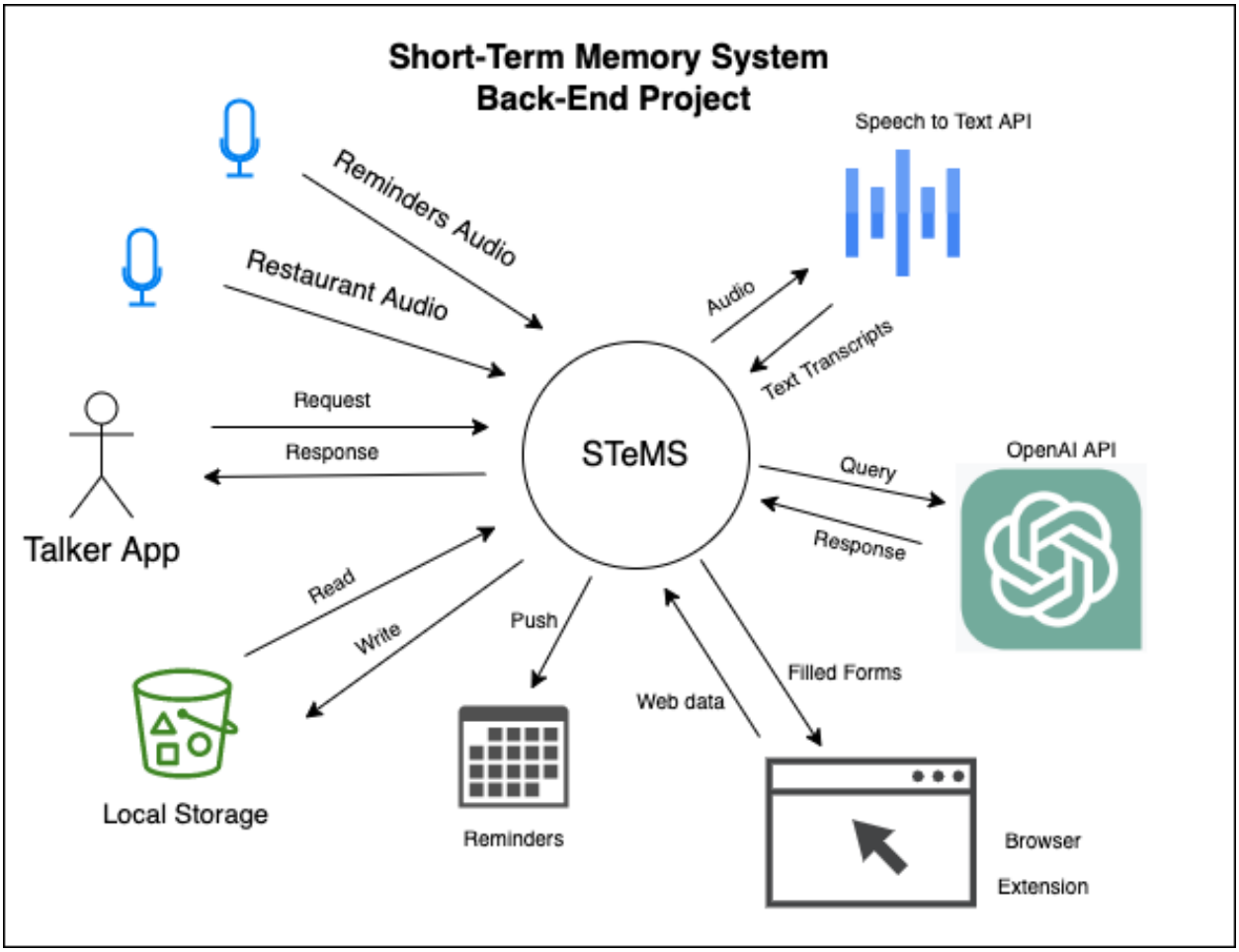
The STeMS backend system can be segmented into 3 areas: audio recording management API, ChatGPT Query API, and Browser Extension API.

The primary entity in the STeMS system is the audio recording. Much of the core functionality of the application is designed to make creating and managing recordings easier. The interface for viewing and maintaining recordings includes activities such as adding a recording, get a list of recordings, and sorting recordings.

The recording query functionality uses Speech-to-Text (STT), speaker diarization, and ChatGPT to generate rich transcripts and query ChatGPT with preprogrammed prompts to extract information from them. All recording metadata is combined to provide the free query interface, which allows the user to ask questions in the context of all recordings made to date.

The Browser Extension API facilitates the filling of forms on any webpage using information from previously recorded audio, a rich transcript based on the recording, and ChatGPT to extract the information that should go in each field of the form.

**Figure 1. STeMS Overview**




### 2.1. Use-Case Model Survey

There are 16 Use Cases that make up the requirements for the PTA system. The detailed Use Cases are in section 3.

| Use Case | Description |
|---|---|
| 1. Edit Recording Metadata | Edit metadata of audio recording. |
| 2. Get Recording Metadata | Get metadata of audio recording. |
| 3. Search Recordings | Local text-based search of audio recordings in local storage. |
| 4. List Recordings | List audio recordings in local storage. |
| 5. Delete Recording | Delete audio recording from local storage. |
| 6. Save Recording | Save audio recording to local storage. |
| 7. Convert Speech to Text | Send audio recording to speech to text API service. |
| 8. Generate App Instance Code | Create a short code for Browser Extension to identify ConvoBuddy application. |
| 9. Get App Instance Code | Returns the local app instance code from local storage. |

| | |
| --- | --- |
| 10. Extract Form Values | Takes a list of fields and sends query to ChatGPT to fill in form blanks based on recording transcript. |
| 11. Process Request for Food Orders | Send query to ChatGPT to generate a food order. |
| 12. Process Request for Reminders | Send query to ChatGPT to determine relevant reminders based on audio recording. |
| 13. Global Search | Send arbitrary query to ChatGPT to provide a response based upon many (all) audio transcripts. |
| 14. Get User Profile | Return user profile information to UI. |
| 15. Set User Profile | Set user profile information. |
| 16. Push Reminder Notification | Send reminders to app/OS based on time-based triggers. |

### 2.2. Assumptions and Dependencies

- The app must utilize ChatGPT.
- The app will be developed in Flutter and Dart.
- Data input will be dependent on audio recordings.

## 3. Specific Requirements

The Specific Requirements section will provide the Use Case Reports specifying the 16 Use Cases that make up this system.

### 3.1. Use-Case Reports

Functionality depicted in use cases show the Actor as either the ConvoBuddy front-end or the Browser Extension

#### 3.1.1 Use Case Name: Edit Recording Metadata

**Actor:** ConvoBuddy Application

**Summary:** A saved recording's attributes are updated.

**Preconditions**:

1. A recording must exist to edit its attributes.

**Triggers**: Request to edit recording metadata is executed.

**Basic course of events** (main scenario):

| ConvoBuddy Application | System |
| --- | --- |
| 1. ConvoBuddy Application launches functionality to allow metadata edits. | |
| | 2. Inputted metadata changes are updated for that recording. |

**Post-conditions**:

1. Changes to a recording's metadata are reflected.

#### 3.1.2 Use Case Name: Get Recording Metadata

**Actor:** ConvoBuddy Application

**Summary:** A saved recording's metadata is retrieved.

**Preconditions**:

1. Metadata for a recording must exist for it to be retrieved.

**Triggers**: A request for recording retrieval is executed.

**Basic course of events** (main scenario):

| ConvoBuddy Application | System |
|---|---|
| 1. ConvoBuddy Application launches request to retrieve a recording's metadata. | |
| | 2. The recording's metadata is sent back to the ConvoBuddy Application. |

**Post-conditions**:

1. Recording properties are displayed in the application.

### 3.1.3 Use Case Name: Search Recordings (Deleted)

Use case deleted as duplicate with Team A implemented functionality on August 7, 2023.

### 3.1.4 Use Case Name: List Recordings

**Actor:** ConvoBuddy Application

**Summary:** List of recordings are displayed based on search query results.

**Preconditions**:

1. Search query matches recording titles.

**Triggers**: Recording list has matching titles to search input.

**Basic course of events** (main scenario):

| ConvoBuddy Application | System |
|---|---|
| 1. Recording repository request a list of all recordings. | |
| | 2. All recording titles are returned for display. |

**Post-conditions**:

1. Recordings are displayed in the ConvoBuddy Application.

### 3.1.5 Use Case Name: Delete Recording

**Actor:** ConvoBuddy Application

**Summary:** A recording is deleted from the storage.

**Preconditions**:

1. A recording must exist.

**Triggers**: Delete request is received.

**Basic course of events** (main scenario):

| ConvoBuddy Application | System |
|---|---|
| 1. The ConvoBuddy Application sends a deletion request. | |
| | 2. The recording is removed from storage. |

**Post-conditions**:

1. Recording is no longer available for retrieval from ConvoBuddy Application.

### 3.1.6 Use Case Name: Save Recording

**Actor:** ConvoBuddy Application

**Summary:** Recording is stored into local storage.

**Preconditions**:

1. Recording is initiated.
2. Microphone usage is enabled for the application.

**Triggers**: Recording is stopped.

**Basic course of events** (main scenario):

| ConvoBuddy Application | System |
|---|---|
| 1. Recording is stopped and completed. | |
| | 2. The recording gets stored and saved into local storage. |

**Alternate Scenarios:**

A.

| ConvoBuddy Application | System |
|---|---|
| 1. Recording is stopped and completed. | |
| | 2. Local storage is full, and recording was unable to be saved. |

**Post-conditions**:

1. Recording becomes available for retrieval from ConvoBuddy Application.

### 3.1.7 Use Case Name: Convert Speech to Text

**Actor:** ConvoBuddy Application and Whisper API

**Summary:** API consumes audio recording and converts it into text for ChatGPT processing.

**Preconditions**:

1. A saved recording must exist.

**Triggers**: A request to invoke the Whisper API is received.

**Basic course of events** (main scenario):

| ConvoBuddy Application | System |
| --- | --- |
| 1. ConvoBuddy Application sends a request with the audio recording data to the Whisper API | |
| | 2. The Whisper API sends a response containing the formatted data into text information back to the ConvoBuddy Application. |

**Alternate Scenarios:**

B.

| ConvoBuddy Application | Whisper API |
| --- | --- |
| 1. Invalid request is sent from ConvoBuddy Application. | |
| | 2. Whisper API returns error response. |

**Post-conditions**:

1. The converted text is stored in application memory.

### 3.1.8 Use Case Name: Generate App instance Code

**Actor:** ConvoBuddy Application

**Summary:** A short code is generated on the backend for the Browser Extension to identify a ConvoBuddy application instance.

**Preconditions**:

1. The ConvoBuddy application is installed.

**Triggers**: The ConvoBuddy application is opened.

**Basic course of events** (main scenario):

| ConvoBuddy Application | System |
| --- | --- |
| 1. The ConvoBuddy Application establishes calls the STeMS backend service to get an app instance code. | |
| | 2. The system generates a code and returns it to the actor. The code is stored on local storage for reference later on form population requests. |

**Post-conditions**:

1. The browser extension has a one-time code it can use for the session and the code is stored in the local storage on the STeMS backend.

### 3.1.9 Use Case Name: Get App Instance Code

**Actor:** ConvoBuddy Application

**Summary:** Get the generated short code for the current ConvoBuddy app instance.

**Preconditions**:

1. The ConvoBuddy application is installed.
2. The ConvoBuddy application is open.
3. The ConvoBuddy application has generated an app instance code on startup.

**Triggers**: The ConvoBuddy application is opened.

**Basic course of events** (main scenario):

| ConvoBuddy Application | System |
| --- | --- |
| 1. The ConvoBuddy Application requests the current app instance code. | |
| | 2. The system reads the app instance code from local storage and returns it to the ConvoBuddy application. |

**Post-conditions**:

1. The ConvoBuddy application has the app instance code that was generated during the application's startup. It can be displayed for a user to read and input into the user's instance of Browser Extension.

### 3.1.10 Use Case Name: Extract Form Values

**Actor:** Browser Extension

**Summary:** This API method call takes a list of fields and a recording selected in the ConvoBuddy app and generates values for as many fields as possible using ChatGPT prompt with the recording transcript as context.

**Preconditions**:

1. A user is on a webpage with the Browser Extension loaded.
2. The browser extension has an app instance code generated by the browser user's ConvoBuddy app instance and input by the browser user.

**Triggers**: None

**Basic course of events** (main scenario):

| Browser Extension | System |
| --- | --- |
| 1. The actor sends their app instance code along with the form fields to populate in a machine-readable format. | |
| | 2. The system checks if the app instance code is active and the app instance identified by the code is connected. If the app instance is connected, the application is queried for selected recording and ChatGPT API is queried |

| | |
|---|---|
| | to fill in all fields specified in the browser extension request. |

**Alternate Scenarios:**

A. The app instance code is invalid.

| Actor | System |
|---|---|
| 1. The actor sends their app instance code along with the form fields to populate in a format to be determined later. | |
| | 2. The system checks if the application code is active and determines it is not. An error response is sent back to the browser extension. |

B. The app instance is no longer connected.

| Actor | System |
|---|---|
| 1. The actor sends their app instance code along with the form fields to populate in a format to be determined later. | |
| | 2. The system checks if the application code is active, but the app instance identified is not connected. An error response is sent back to the browser extension. |

**Post-conditions**:

1. The browser extension receives a payload indicating all fields that can be populated based on information in the selected recording, in JSON format.

### 3.1.11 Use Case Name: Process Request for Food Orders (Waiter)

**Actor:** ConvoBuddy Application

**Summary:** Send a request to ChatGPT based on restaurant audio transcript. The system will leverage OpenAI API to parse through audio transcript via natural language processing to produce a formatted food order.

**Preconditions**:

1. ConvoBuddy application is installed, logged in, user profile exists
2. Audio of waiter interaction has been recorded
3. Speech to text API call has completed, transcript of audio has been stored locally to ConvoBuddy application

**Triggers**: None

**Basic course of events** (main scenario):

| ConvoBuddy Application | System |
|---|---|

| | |
| --- | --- |
| 1. The actor submits job to send audio transcript for processing. | |
| | 2. The system validates the audio transcript. If the transcript is valid, the system concatenates a relevant ChatGPT restaurant-related prompt, the restaurant-related user profile data, and the full audio transcript. This text is sent to OpenAI via API. |
| | 3. The system polls OpenAI API to determine job status. |
| | 4. The system receives and parses text response from OpenAI and passes result back to the front-end requestor. |

**Alternate Scenarios:**

A. The audio transcript fails input validation (may be empty file).

| Actor | System |
| --- | --- |
| 1. The actor submits job to send audio transcript for processing. | |
| | 2. The system determines file is invalid. The system returns an error response to the actor. |

B. The OpenAI API service is unavailable (may be unreachable from device).

| Actor | System |
| --- | --- |
| 1. The actor submits job to send audio transcript for processing. | |
| | 2. The system returns an error response to the actor. |

C. The OpenAI API service returns error response or polling times out.

| Actor | System |
| --- | --- |
| 1. The actor submits job to send audio transcript for processing. | |
| | 2. The system returns an error response to the actor. |

**Post-conditions**:

1. The front-end UI processes text response resulting from the OpenAI query.

### 3.1.12 Use Case Name: Process Request for Reminders (STML)

**Actor:** ConvoBuddy Application

**Summary:** Send a request to ChatGPT based on short term memory loss audio transcript. The system will leverage OpenAI API to parse through audio transcript via natural language processing to produce useful summary

information.

**Preconditions:**

1. ConvoBuddy application is installed, logged in, user profile exists
2. Relevant audio of STML interaction has been recorded
3. Speech to text API call has completed, transcript of audio has been stored locally to ConvoBuddy application

**Triggers**: None

**Basic course of events** (main scenario):

| ConvoBuddy Application | System |
| --- | --- |
| 1. The actor submits job to send audio transcript for processing. | |
| | 2. The system validates the audio transcript. If the transcript is valid, the system concatenates a relevant ChatGPT STML-related prompt, the STML-related user profile data, and the full audio transcript. This text is sent to OpenAI via API. |
| | 3. The system polls OpenAI API to determine job status. |
| | 4. The system receives and parses text response from OpenAI and passes result back to the front-end requestor. |

**Alternate Scenarios:**

1. The audio transcript fails input validation (may be empty file).

| Actor | System |
| --- | --- |
| 1. The actor submits job to send audio transcript for processing. | |
| | 2. The system determines file is invalid. The system returns an error response to the actor. |

2. The OpenAI API service is unavailable (may be unreachable from device).

| Actor | System |
| --- | --- |
| 1. The actor submits job to send audio transcript for processing. | |
| | 2. The system returns an error response to the actor. |

3. The OpenAI API service returns error response or polling times out.

| Actor | System |
| --- | --- |
| 1. The actor submits job to send audio transcript for processing. | |

| | |
| --- | --- |
| | 2. The system returns an error response to the actor. |

**Post-conditions**:

1. The front-end UI processes text response resulting from the OpenAI query.

### 3.1.13 Use Case Name: Global Search

**Actor:** ConvoBuddy Application

**Summary:** Send a request to ChatGPT based on a specific query over a set of multiple (all) audio transcripts. The system will leverage OpenAI API to parse through audio transcript via natural language processing to produce useful information.

**Preconditions**:

1. ConvoBuddy application is installed, logged in, user profile exists
2. At least one recording has been made.
3. Speech to text API call has completed, transcript of audio has been stored locally to ConvoBuddy application

**Triggers**: None

**Basic course of events** (main scenario):

| Actor | System |
| --- | --- |
| 1. The actor submits job to send ChatGPT query for processing. | |
| | 2. The system validates the request and tests for 1 or more valid transcripts. If the transcript is valid, the system concatenates a relevant ChatGPT prompt with the user's query, the relevant user profile data, and the set of audio transcripts. This text is sent to OpenAI via API. |
| | 3. The system polls OpenAI API to determine job status. |
| | 4. The system receives and parses text response from OpenAI and passes result back to the front-end requestor. |

**Alternate Scenarios:**

1. The request or audio transcript fails input validation (may be empty request or no valid audio transcripts exist).

| Actor | System |
| --- | --- |
| 1. The actor submits job to request to ChatGPT for processing. | |

| | |
| --- | --- |
| | 2. The system determines request is invalid or there are no valid audio transcripts for query. The system returns an error response to the actor. |

2. The OpenAI API service is unavailable (may be unreachable from device).

| Actor | System |
| --- | --- |
| 1. The actor submits job to request to ChatGPT for processing. | |
| | 2. The system returns an error response to the actor. |

3. The OpenAI API service returns error response or polling times out.

| Actor | System |
| --- | --- |
| 1. The actor submits job to request to ChatGPT for processing. | |
| | 2. The system returns an error response to the actor. |

**Post-conditions**:

1. The front-end UI processes text response resulting from the OpenAI query.

### 3.1.14 Use Case Name: Get User Profile

**Actor:** ConvoBuddy Application

**Summary:** Return the user profile information to the UI. This profile will be stored will be stored as local text within the app. The profile will include relevant context for ChatGPT such as the name of the restaurant, menu items available, etc. for it to be able to contextualize the recorded audio. Specific to the short-term memory loss use case, it would be necessary to include the user's name, address, or other context for the OpenAI query to parse out relevant information for the user. Getter and setter methods will be made available to the front-end UI if this needs to be edited.

**Preconditions**:

1. ConvoBuddy application is installed, logged in, user profile exists

**Triggers**: None

**Basic course of events** (main scenario):

| Actor | System |
| --- | --- |
| 1. The actor sends request for the user profile. | |
| | 2. The system returns the text of the user profile from local storage. |

**Alternate Scenarios:**

A. User profile text file is unavailable

| Actor | System |
| --- | --- |
| 1. The actor sends request for the user profile. | |
| | 2. The system returns an error response to the actor. |

**Post-conditions**:

1. The front-end UI processes text response.

### 3.1.15 Use Case Name: Set User Profile

**Actor:** ConvoBuddy Application

**Summary:** Set the user profile information from the UI. This profile will be stored will be stored as local text within the app. The profile will include relevant context for ChatGPT such as the name of the restaurant, menu items available, etc. for it to be able to contextualize the recorded audio. Specific to the short-term memory loss use case, it would be necessary to include the user's name, address, or other context for the OpenAI query to parse out relevant information for the user. Getter and setter methods will be made available to the front-end UI if this needs to be edited.

**Preconditions**:

1. ConvoBuddy application is installed, logged in.

**Triggers**: None

**Basic course of events** (main scenario):

| Actor | System |
| --- | --- |
| 1. The actor sends user profile data. | |
| | 2. The system validates input. If the format is valid, the system overwrites the text of the user profile in a local file. |

**Alternate Scenarios:**

A. The request fails input validation (may be empty request or other defined invalid format).

| Actor | System |
| --- | --- |
| 1. The actor sends user profile data. | |
| | 2. The system validates input. If validation fails, the system returns an error response to the actor. |

B. The write operation fails.

| Actor | System |
| --- | --- |
| 1. The actor sends user profile data. | |
| | 3. The system validates input. The system |

| | |
| --- | --- |
| | tries to write to file. If the write operation fails, the system returns an error response to the actor. |

**Post-conditions**:

1. The front-end UI receives response of successful write operation.

### 3.1.16 Use Case Name: Push Reminder Notification

**Actor:** None

**Summary:** The system will, upon background detection of a reminder time being reached, show an OS rendered notification for the reminder.

**Preconditions**:

1. A reminder time has passed.

**Triggers**: None

**Basic course of events** (main scenario):

| N/A | System | Screen |
| --- | --- | --- |
| | 1. The system detects that a reminder time has passed, and the reminder has not been shown yet. The system submits the reminder details to the OS to display as a notification. | |

**Post-conditions**:

1. A reminder notification is shown, and the reminder metadata is updated to reflect that it has been displayed.

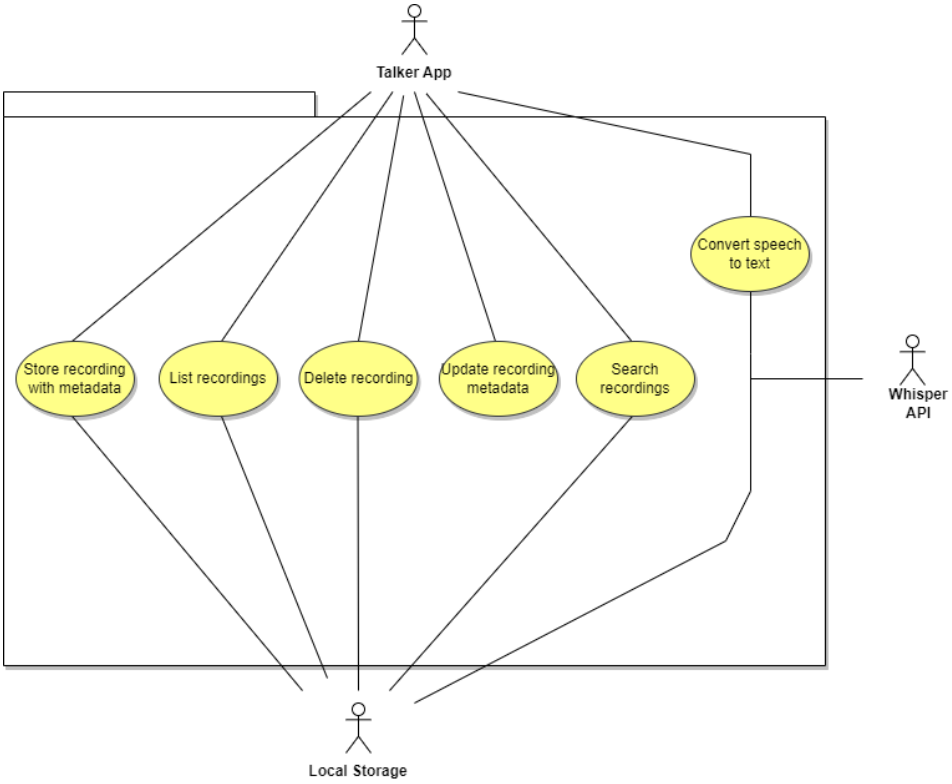
## 3.2. Use Case Diagrams

The following Use Case diagrams illustrate the interaction between the actor and system using standard UML Use Case diagram format.

### 3.2.1 Recordings API

The recordings API handles storing and retrieving conversations recorded through the ConvoBuddy app.

**Figure 2. Recordings API Diagram**




### 3.2.2 Browser Extension API

The Browser Extension API brokering a link between the browser extension and the mobile ConvoBuddy application, and passing form filling request to the ChatGPT API.

**Figure 3. Browser Extension API Diagram**

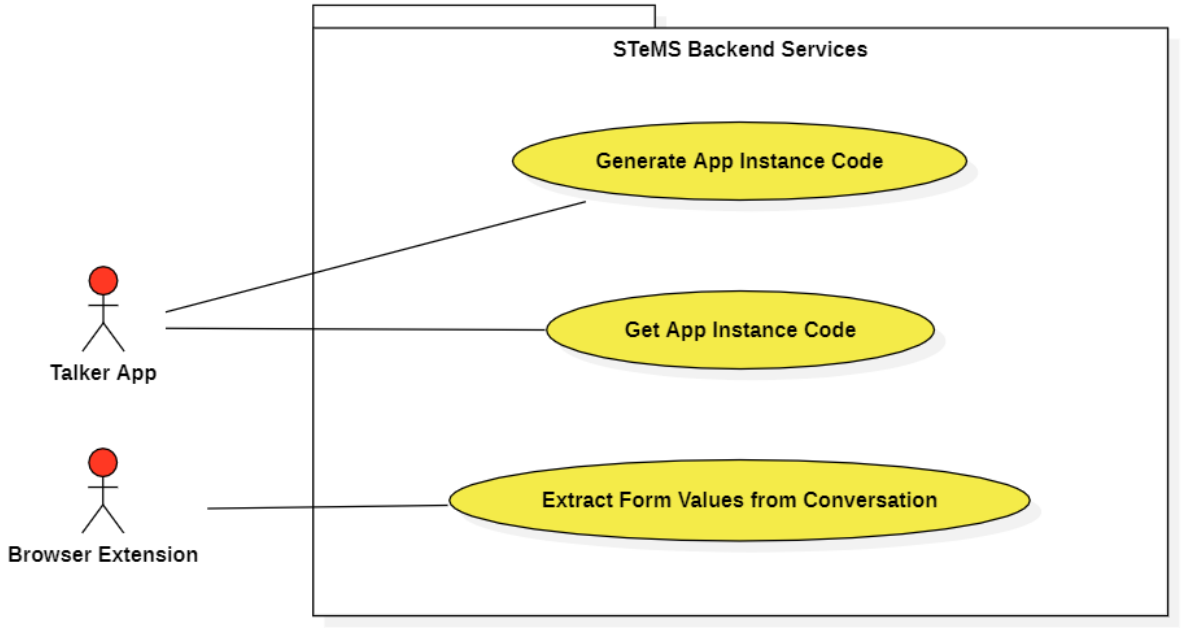



**Figure 4. Browser Extension API Generate App Instance Code Sequence Diagram**

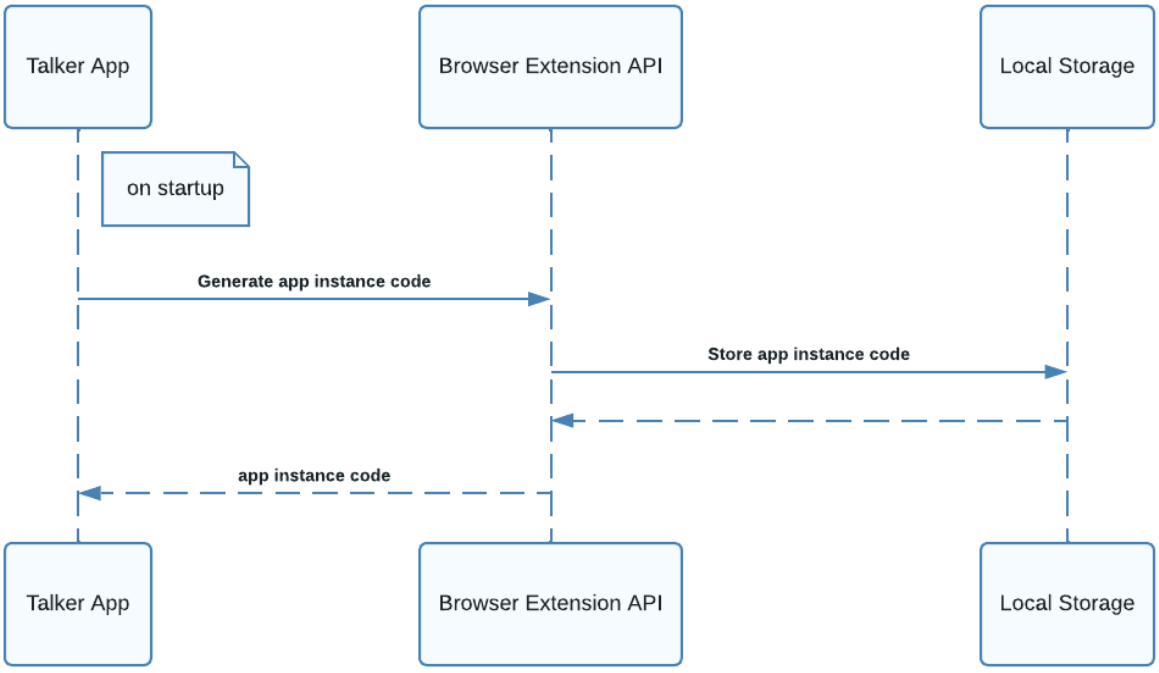


**Figure 5. Browser Extension API Fill Form Request Sequence Diagram**

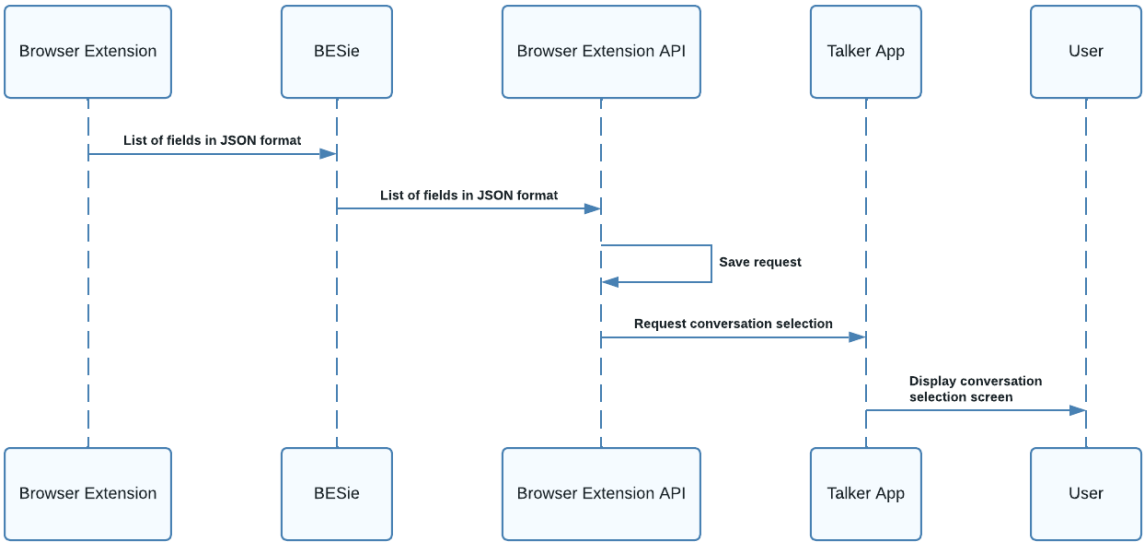
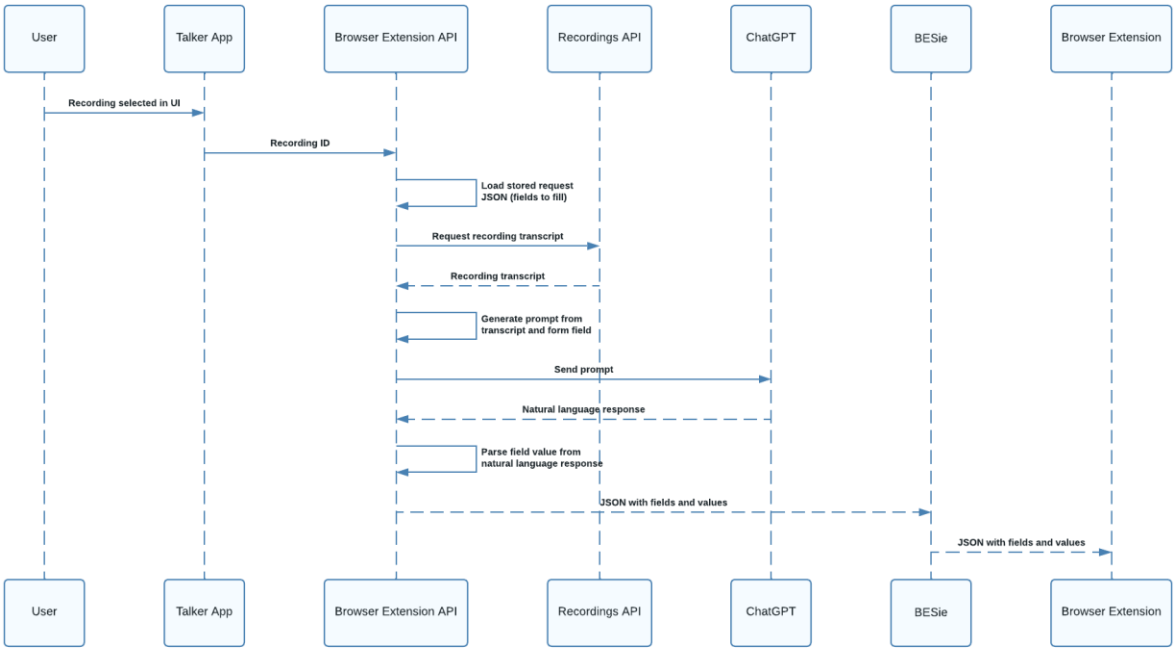



**Figure 6. Browser Extension API Fill Form Response Sequence Diagram**




### 3.2.3 ChatGPT Query API

The ChatGPT Query API allows the ConvoBuddy application and Browser Extension API to query ChatGPT for

contextual information given a textual transcript or set of transcripts. The ChatGPT Query API takes a basic request defined by it's API and generates a ChatGPT prompt using recording transcript information and

**Figure 7. ChatGPT Query API Diagram**

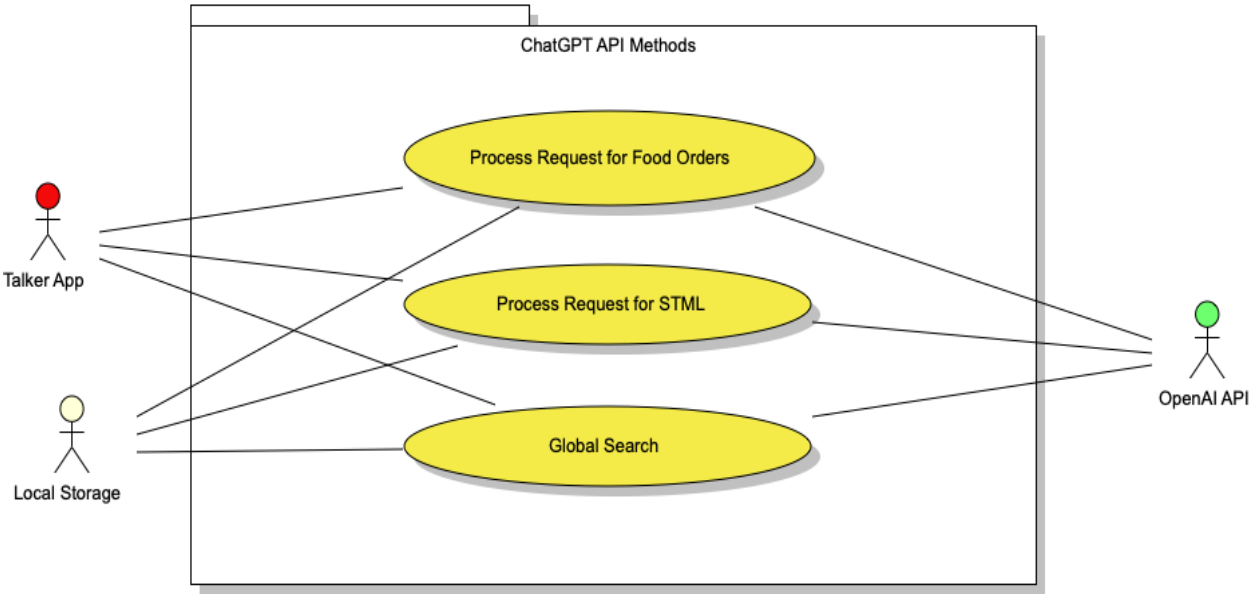
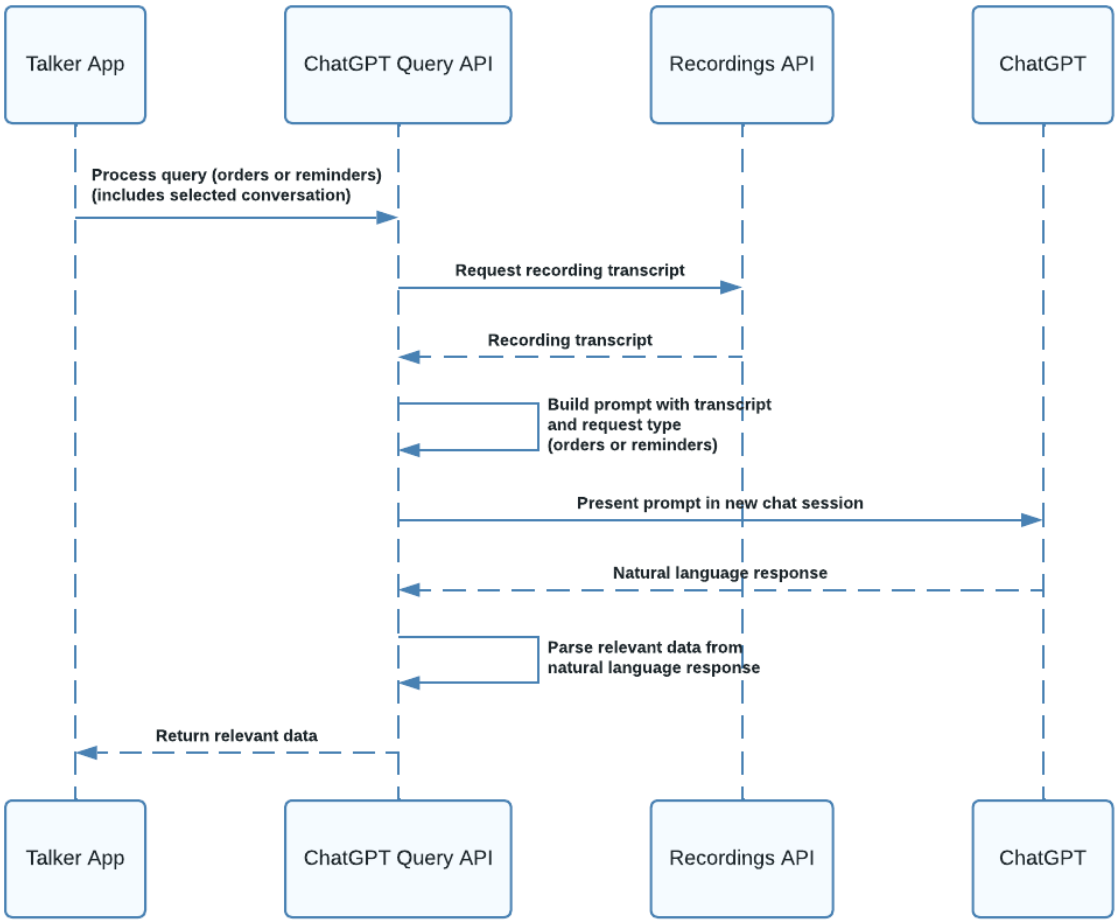


**Figure 8. ChatGPT Query API Sequence Diagram**




### 3.3. Supplementary Requirements

Supplementary requirements cover requirements applicable to the STeMS backend system not expressed in any use case.

SR1 - Local Storage: All recordings are to be stored in the device's local storage.
SR2 - Recording will continue in the background even while the application is not open.
SR3 - Only one recording can be recorded at a time.

## 4. Non-functional Requirements

Non-functional requirements are requirements that are not necessary for basic operation of the system, but help define aspects of the system that define a baseline of capability such as performance, security, reliability, and scalability.

### 4.1. Performance

- NF1 – All API endpoints not calling an external service should have a response time of 200

milliseconds or less for single user load.

### 4.2. Security

- NF2 - The API must use secure communication protocols (e.g., HTTPS) to protect data transmission.
- NF3 - Services exposed to the internet should implement API secret or user authentication to prevent external actor use or denial of service attacks.
- NF4 - API keys and other secrets must be sanitized from source code.

### 4.3. Reliability

- NF5 - The API servers should have a minimum uptime of 99% per month.
- NF6 - API endpoints should expose a common response base object that includes required status code and optional error message fields.
- NF7 - API endpoints should not throw an exception, but instead catch it to ensure the client can rely on either getting a successful response or a well formatted error response.

### 4.4. Scalability

- NF8 - The API should be designed to handle increased user load by scaling horizontally or vertically.

## 5. Documentation

The API must be thoroughly documented, including:

- Overview and purpose of the API
- Detailed descriptions of each functionality and their usage
- API endpoints, request parameters, and expected responses
- Examples of API requests and responses

## 6. Testing Requirements

The system must be testable to demonstrate establishment of requirements.

- The API must be thoroughly tested to ensure the correctness of its functionalities and adherence to the specified requirements.
- Test cases should be developed to cover positive and negative scenarios.
- Test data should be generated or used to simulate various scenarios.

## 7. Future Enhancements

Future enhancements may be used to add additional capability of the system if development time permits.

- Cloud Storage: Recordings may be backed up or offloaded to the cloud if the user configures a third-party cloud storage service.