Team B Best Team

STeMS

Technical Design Document

Version 3.0

Document Control

Date	Version	Description	Author
06/12/2023	**1.0**	**Initial Revision of the TDD**	**Amol Thomare, Benny Iko, David Babers, Collin Hicks, Scott McCrillis**
07/22/2023	**2.0**	**Milestone 3**	**Team B Members**
07/24/2023	**3.0**	**Milestone 4**	**Team B Members**

Table of Contents

1 INTRODUCTION

1.1 Purpose

The purpose of this document is to provide an overview of the ConvoBuddy application to the client. This document explains the architecture of the software and the approach Team B is using to create the backend software used to support Team AlphaSoft's frontend code.

1.2 Scope

The scope of the project is the backend software for the ConvoBuddy mobile application. This document will go over the architecture, design, and requirements involved for the backend, which includes:

- How the data is being stored locally
- How transcriptions are created through the recording of vocal inputs and sending those recordings to OpenAI Whisper.
- How queries are being created through vocal inputs recording and transcriptions are created through sending those recordings to OpenAI Whisper.
- Sending both transcriptions and queries to ChatGPT to receive responses that are passed back to the frontend

1.3 Overview

This TDD consists of the following parts:

- Introduction
- System Overview
- System Architecture
- Data Design
- Component Design
- Human Interface Design (Done by Team AlphaSoft)
- Requirements Matrixes
- Appendices

1.4 Project Documentation

1.4.1 Project Suite of Documents

	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

1.5 Definitions and Acronyms

Acronym	Full Name
TDD	Technical Design Document
API	Application Programming Interface
STeMS	Short-Term Memory System
BESie	Browser Extension Interface
STML	Short-Term Memory Loss
PMP	Project Management Plan
SRS	Software Requirements Specification
STP	Software Test Plan
PG	Programmers Guide
DevOps	Deployment and Operations
UG	User Guide
TR	Test Report
OO	Object Oriented
ASR	Automatic Speech Recognition
HTTP	Hypertext Transfer Protocol
TCP	Transmission Control Protocol
STT	Speech To Text
REST	Representational State Transfer
NPM	Node Package Manager
YAGNI	You Aren’t Gonna Need It
YAML	Yet Another Markup Language

2 SYSTEM OVERVIEW

The ConvoBuddy Application uses a diarization API and ChatGPT to take in vocal inputs from users, converts them into transcriptions, and sends them to ChatGPT to generate responses based on the transcription and the queries asked about the transcription. The data of the recordings and the transcriptions is stored locally on the user device and the technology stack should look like the following:

- Presentation: UI of the mobile application implemented by Team A within Flutter and Dart as well as the BESie (Browser Extension Service)
- Application/Domain: The services implemented by the backend which include

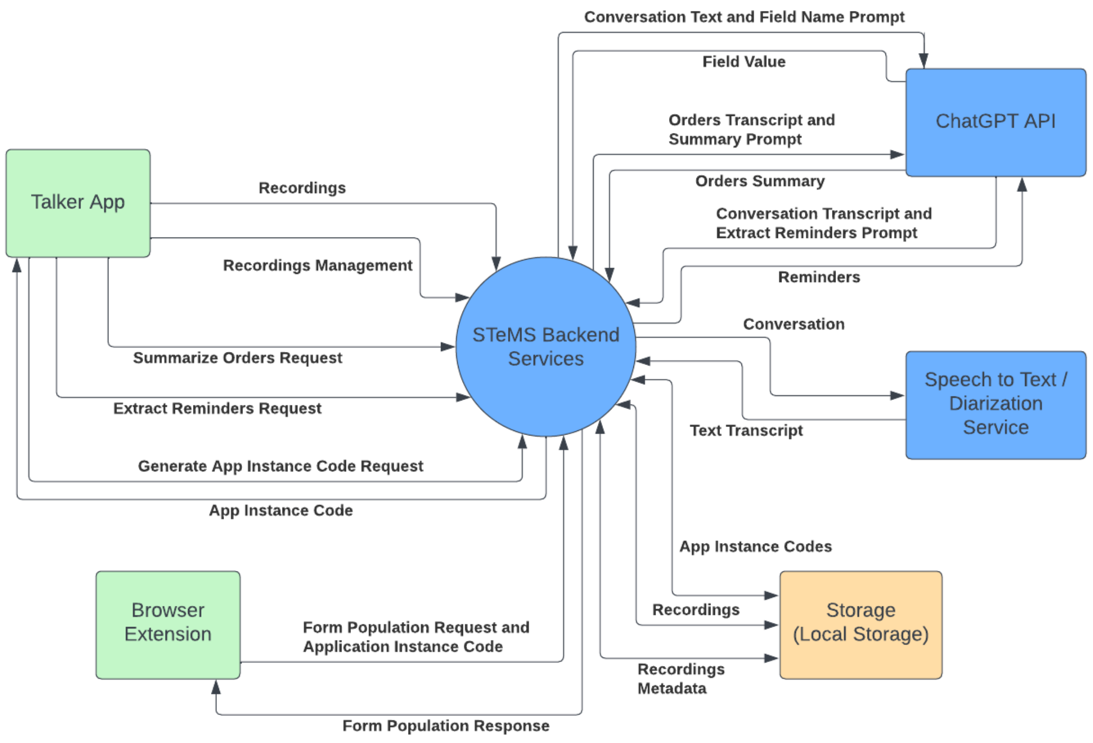
    - ChatGPT API to process transcriptions and queries
    - Speech to Text diarization software
    - WebSocket API between Mobile application and browser extension
- Data Storage: The data will be stored within the local storage of the device. This is to ensure that the user data is kept secure.

3 SYSTEM ARCHITECTURE

3.1 Architectural Design

This document will detail the architectural design of the STeMS system and its associated services. STeMS are comprised of three main services: the ChatGPT API Service, the Speech to Text Service, and the Local Storage service. Each service is responsible for sending, transforming, and responding to data with the associated front-end application, ConvoBuddy App. Each respective services and its associated data flows can be viewed in the below figure:

Figure 1. System Overview

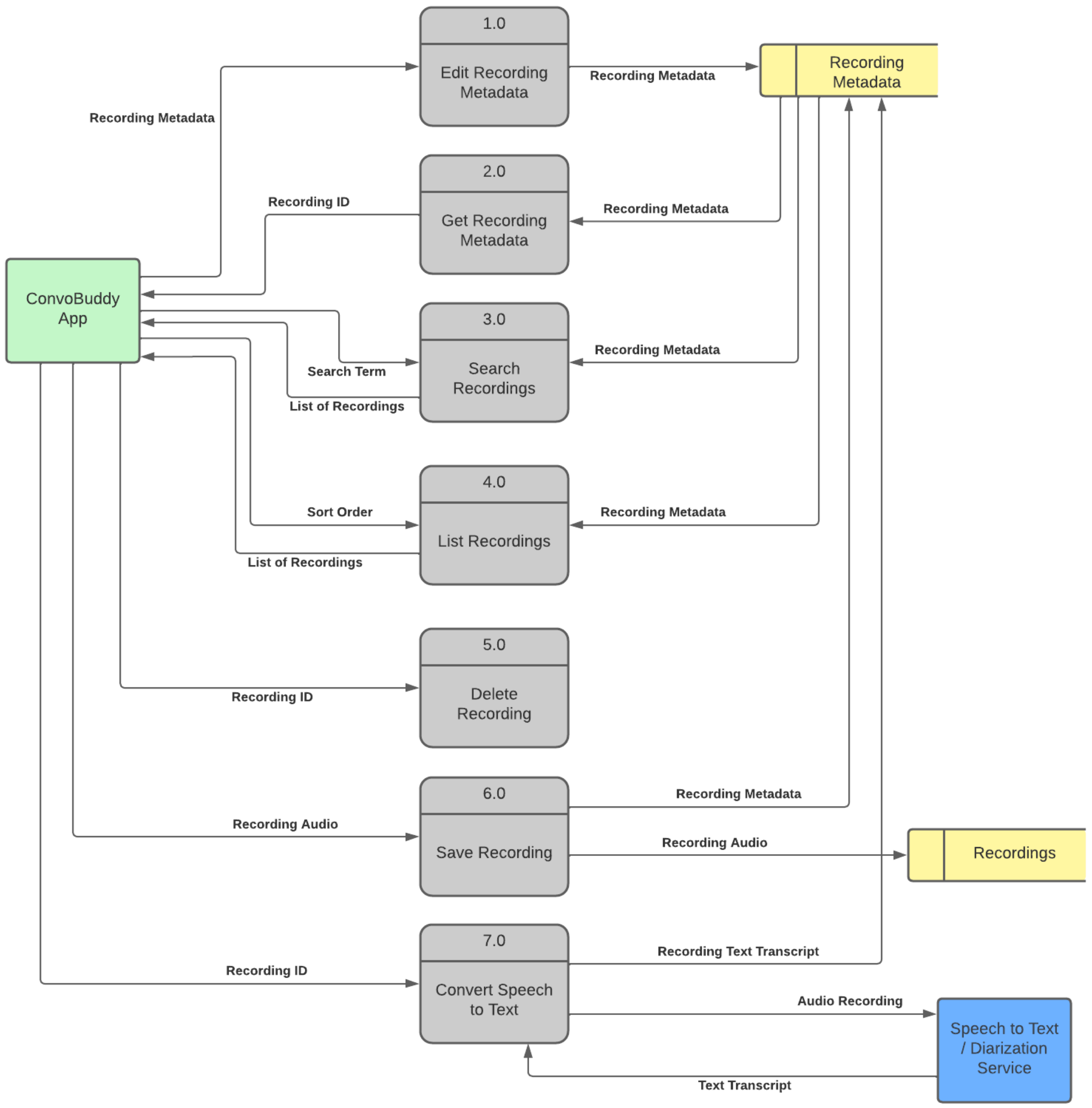


3.2 Decomposition Description

The following sections will show the flow of data through each of the three services, and their relation to specific methods, functions, and use cases.

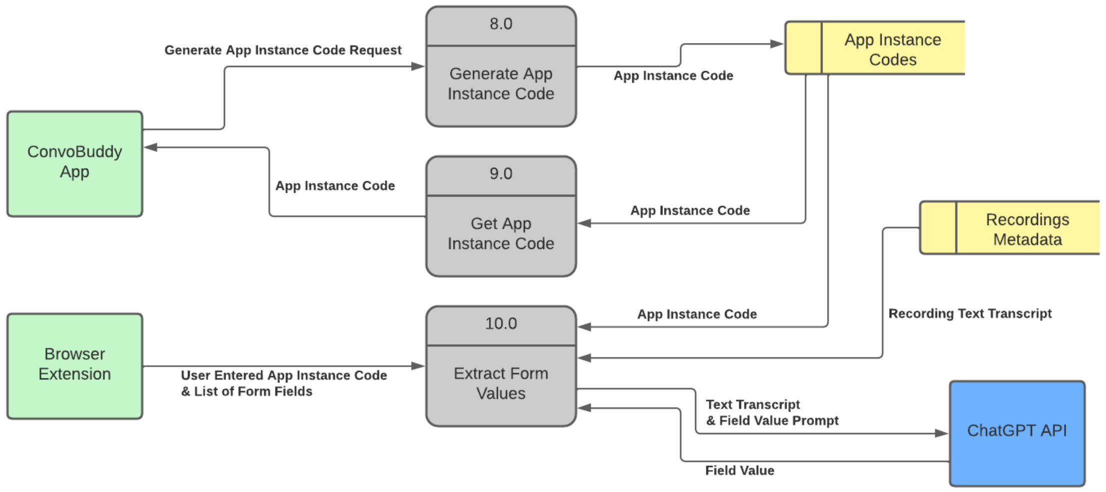
3.2.1 Recording Services (ChatGPT API, Speech Service, Local Storage)

Figure 2. Recording Services Data Flow



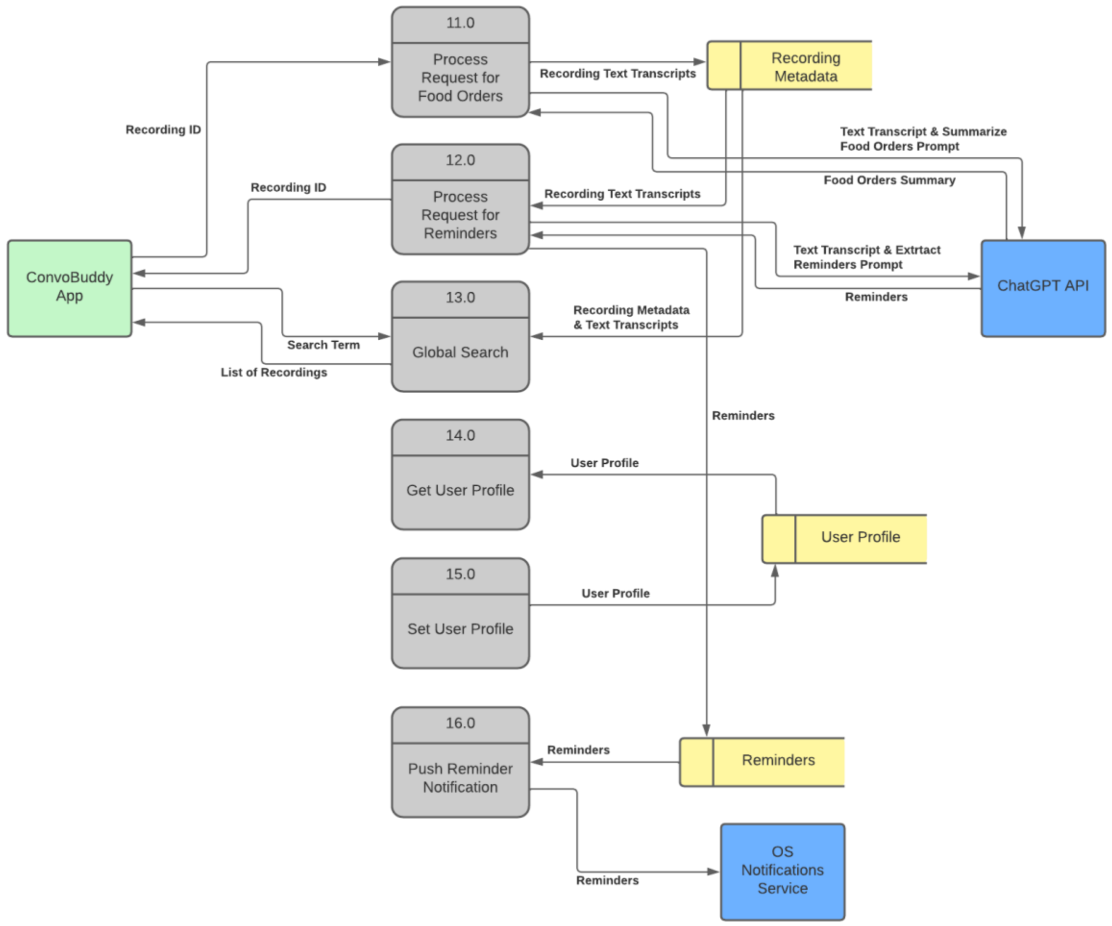
3.2.2 Browser Extension API (ChatGPT API, Local Storage Service)

Figure 3. Browser Extension API Data Flow



3.2.3 ChatGPT Query Service (ChatGPT API, Local Storage Service)

Figure 4. ChatGPT Query Service Data Flow



3.3 Exception Handling

Briefly summarize each user defined exception to include what data must be passed in the constructor(s), what message(s) will be generated, and which class and methods will throw the exceptions.

Exception	Description
InvalidAppInstanceCodeException	Used when App instance code argument is incorrect.
InvalidTranscriptException	Used when Recording transcript fails input validation prior to API call.
ApiUnreachableException	Used when API call returns unreachable.
ApiErrorException	Used when API call returns error or times out.
SocketUnreachableException	Used when push to Browser Extension returns unreachable.
FailedWriteOperationException	Used when local write operation fails.
FailedReadOperationException	Used when local read operation fails.

3.4 Design Rationale

Several factors, decisions, and requirements lead to the design choices reflected in section 3.1. First, the requirements and time frame for this project have necessitated that the application be designed in a monolithic architecture in which each "service" is a modular package within the main branch of code. Each backend service within STeMS must reside within the main application and be readily available for front end services to consume. To ensure better collaboration between the members of Team B and the Front-End development team, the following decisions were made in regards to design.

First, the backend should be composed of modular and discrete services. Each service is to be loosely coupled from the others so that one does not act a single point of failure in the application. The additional benefit of having multiple supporting services within the backend is that development and engineering can be better matched to the requirements matrix, properly scoped, and better covered with specific unit tests.

Second, the backend services are architected so that they can be consumed easily throughout the entire stack of the STeMS backend and ConvoBuddy App. Each service is designed to be available at any point within the application. Since each service has responsibility localized to only its functionality and does not share responsibilities with the other services, each can be invoked and instantiated without fear of duplication of effort, responsibilities, or the confusion.

Finally, the requirements have necessitated that the final product, both front and backend, be delivered as a single monolithic service. To ensure that the entire ConvoBuddy App could be run with one common repository, and to reduce the number of third-party services needed, the STeMS services were architected to accommodate these constraints. As a result, each service only requires access to either the local device, or the ChatGPT/Whisper API. It does necessitate the need for additional infrastructure, access to disparate code bases, and reduces the likelihood of service failures and application downtime.

4 DATA DESIGN

4.1 Data Description

This section explains how the information domain of the STeMS backend services is transformed into data structures. Major data or system entities are discussed including how they are stored, processed and organized.

Data is stored locally within the device filesystem. Reminders are stored in JSON format via path: “reminders.json”

Example file format:

```
[
{
   "reminderId": 1,
   "createTimestamp": 1686672543,
   "notifyTimestamp": 1686800000,
   "reminderDescription": "Pick up Eggs from the grocery store",
   "userId": "user"
},
{
   "reminderId": 2,
   "createTimestamp": 1686672543,
   "notifyTimestamp": 1687300000,
   "reminderDescription": "Dental Appointment",
   "userId": "user"
},
{
   "reminderId": 3,
   "createTimestamp": 1686672543,
   "notifyTimestamp": 1686900000,
   "reminderDescription": "Mow the lawn",
   "userId": "user"
},
{
   "reminderId": 4,
   "createTimestamp": 1686672543,
   "notifyTimestamp": 1686692543,
   "reminderDescription": "Take medication",
   "userId": "user"
},
{
   "reminderId": 5,
   "createTimestamp": 1686672543,
   "notifyTimestamp": 1688933343,
   "reminderDescription": "Call your sister - Mary's birthday",
   "userId": "user"
}
]
```

All recording metadata is stored in a single JSON files via path: “conversations.json”. The guid is a 128-bit primary key that identifies the recording. Example file format:

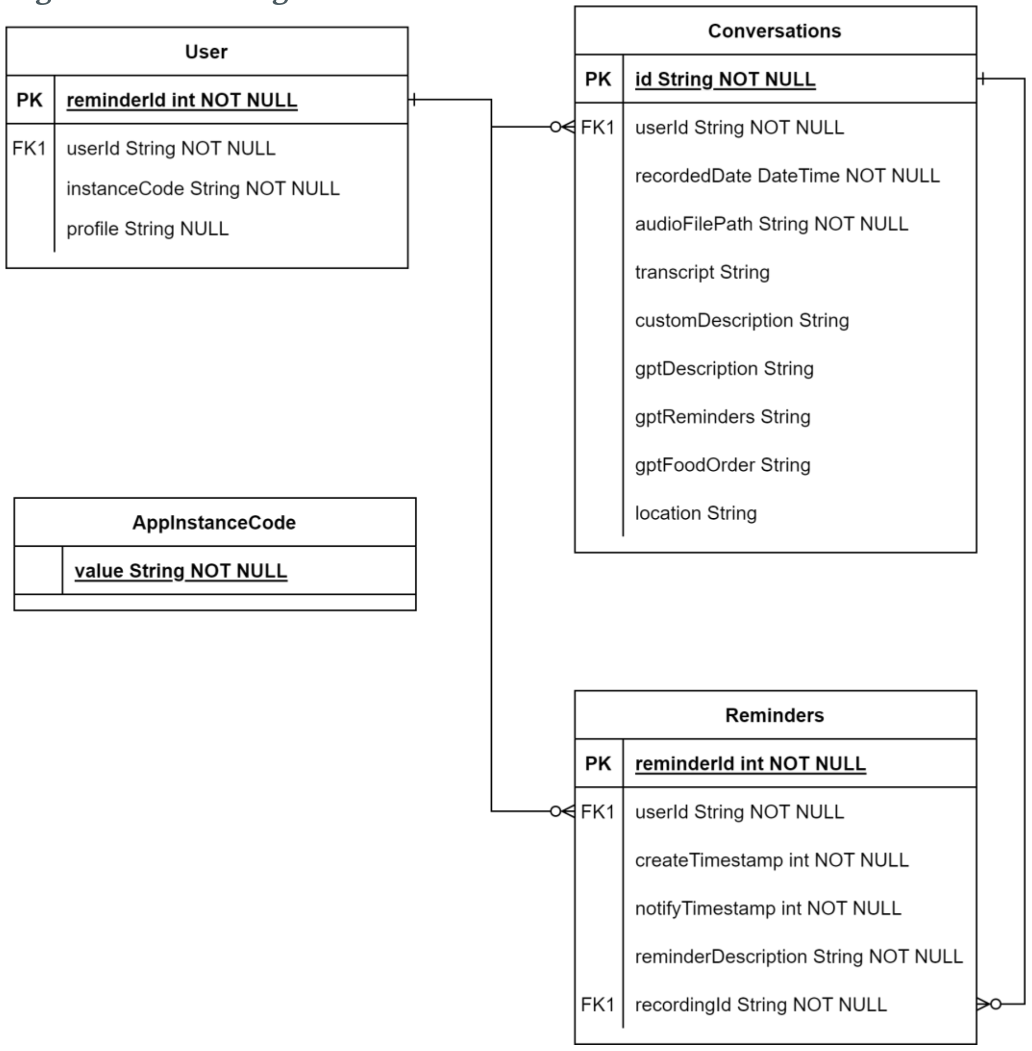
```
[
 {
   "id": "abc37428-e345-492f-8a32-bbbb183d763f",
   "audioFilePath":
"/data/user/0/com.example.talker_mobile_app/app_flutter/abc37428-e345-
492f-8a32-bbbb183d763f",
   "duration": "35000",
   "recordedDate": "2023-07-04T03:35:16.427603",
   "transcript": "Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip
ex ea commodo consequat. Duis aute irure dolor in reprehenderit in
voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur
sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt
mollit anim id est laborum.",
   "customDescription": "Lunch with Murphy",
   "gptDescription": "Charles had lunch with his coworker Murphy.  They
discussed horses, the weather, and next week's golf outing.",
   "gptReminders": "Reminder 1: 2023-07-22T21:34, Record a message on
Combo Buddy\nReminder 2: 2023-07-23T21:34, Listen to the recorded message
on Combo Buddy.",
   "gptFoodOrder": "Seat 1: Hamburger with no pickles, waffle fries, and a
medium Fanta.",
   "userId": "user",
   "location": "[{uuid: \"550e8400-e29b-41d4-a716-446655440000\",date:
\"2010-04-30T16:50:00Z\",location: {position: {latitude: 45.516,longitude:
-122.636}}}]"
 }
]
```

An input variable used for sorting recordings is enum SortingType. This is constrained to a small number of possible values for the sort operation. An example declaration of this variable is below:

```
enum SortingType {
  dateNewToOld,
  dateOldToNew,
  titleAToZ,
  titleZToA,
  durationShortToLong,
  durationLongToShort
}
```

The following Entity Relational Diagram reflects the data relationships in the STeMS backend.

Figure 5. ERD Diagram


User
PK reminderId int NOT NULL
FK1 userId String NOT NULL
instanceCode String NOT NULL
profile String NULL
Conversations
PK id String NOT NULL
FK1 userId String NOT NULL
recordedDate DateTime NOT NULL
audioFilePath String NOT NULL
transcript String
customDescription String
gptDescription String
gptReminders String
gptFoodOrder String
location String
AppInstanceCode
value String NOT NULL
Reminders
PK reminderId int NOT NULL
FK1 userId String NOT NULL
createTimestamp int NOT NULL
notifyTimestamp int NOT NULL
reminderDescription String NOT NULL
FK1 recordingId String NOT NULL

4.2 Data Dictionary

Alphabetically list the system entities or major data along with their types and descriptions. For OO design this consists of the list the objects and their attributes, methods and method parameters.

Field Name	Type
addConversation(Conversation conversation)	returns void
addReminder(Reminder newReminder)	returns void
Agent	Object

audioFilePath	String
convertSpeechToText(String guid)	returns void
createRecording(String filename)	returns String
createTimestamp	int
customDescription	String
deleteAudioFile(String id)	returns void
deleteRecordings(String guid)	returns void
deleteReminder(int reminderId)	returns void
ReceiveFormValuesRequest(BERequest request)	returns void
extractFormValues(String recordingGuid)	returns void
filterConversations(String searchText)	return null
generateInstanceCodeIfNone ()	returns String
Get conversations	returns Array of Objects
getConversationsFromJsonFile()	returns Array of Objects
getInstanceCode()	returns String
getProfile()	returns String
getReminders()	returns Array of Objects
globalSearch(String searchTerm)	returns Array of Objects
gptDescription	String
Guid	String
instanceCode	String

Location	String
notifyTimestamp	int
processFoodOrder(String guid)	returns String
processReminders(String guid)	returns String
Profile	String
Recording	Object
recordingList	Array of Objects
removeAllConversations()	returns void
removeConversation(Conversation conversation)	returns void
Reminder	Object
reminderDescription	String
reminderId	int
reminderList	Array of Objects
saveToFile()	returns void
searchRecordings(String searchTerm)	returns Array of Objects
setSelectedConversation(Conversation conversation)	returns void
setProfile()	returns void
timestamp	int
toJson	returns String
transcript	String
updateConversationTitle(String id, String newTitle)	returns void
updateConversationTranscript(String id, String newTranscript)	returns void
updateCustomDescription(String id, String newCustomDescription)	returns void
updateGptDescription(String id, String newGptDescription)	returns void
updateGptReminders(String id, string newGptReminders)	returns void

updateGptFoodOrder(String id, String newGptFoodOrder)		returns void
userId		String
writeConversationsToJsonFile()	return void	Writes a list of recordings to local file storage.

5 COMPONENT DESIGN

The STeMS backend is divided into classes defining reminder objects, recording objects, and an agent object that provides methods that enable the front end to invoke back-end functionality.

5.1 Components

Class Name: Agent
Class Description/Purpose: Provides the ability to interact with recordings and reminders. Enables interaction with external APIs.
Class Modifiers: Public
Class Inheritance: none
Class Attributes:

- Private userId: String
- Private _logger: Logger
- Private profile: String?
- Private conversationProvider: ConversationProvider
- Private reminderList: List<Reminder>
- Private _beService: BEService
- Private numAppInstanceCodeDigits: const int
- Private _webSocketClient: WebSocketClient?
- Private _recordingSelectionActivator: RecordingSelectionActivator?

Exceptions Thrown:

- Public invalidAppInstanceCodeException()
- Public invalidTranscriptException()
- Public apiUnreachableException()
- Public apiErrorException()
- Public socketUnreachableException()
- Public failedWriteOperationException()
- Public failedReadOperationException()

Class Constructors: Agent (String userId, Directory appDirectory)
Class Methods:

- Public initialize(RecordingSelectionActivator recordingSelectionActivator)

- Public setBeStorageService(JsonStorage storageService)
- Public shutdown()
- Public getProfile(): String
- Public setProfile()
- Public getInstanceCode(): String
- Public generateInstanceCodeIfNone()
- Public setRecordingSelector(RecordingSelectionActivator recordingSelectionActivator)
- Public extractFormValues(String instanceCode, List formFields)
- Public getOpenAiSummary(String guid): String?
- Public getOpenAiReminders(String guid): String?
- Public getOpenAiFoodOrder(String guid): String?
- Public processFoodOrder(String guid): String
- Public processReminders(String guid): String
- Public globalSearch(String searchTerm): List<Recording>
- Public getReminders(): List<Reminder>
- Public deleteReminder(int reminderId)
- Public addReminder(Reminder newReminder)
- Public writeRemindersToFile()
- Public readRemindersFileJSON()
- Private _recieveTranscript(StompFrame frame)

Class Name: ConversationProvider
Class Description/Purpose: Object for storing and managing the list of recordings.
Class Modifiers: Public
Class Inheritance: mixin with ChangeNotifier
Class Attributes:

- Private _conversations: List<Conversation>
- Private _appDirectory: Directory
- Private _conversationsJsonPath: String
- Private _sortingType: SortingType
- Private _selectedConversation: Conversation?

Exceptions Thrown:
Class Constructors: ConversationProvider(Directory appDirectory)
Class Methods:

- Public get conversations: List<Conversation>
- Public get selectedConversation: Conversation?
- Public get sortingType: SortingType
- Public get appDirectory: Directory
- Public get conversationJsonPath: String
- Public addConversation(Conversation conversation): void
- Public removeConversation(Conversation conversation): void
- Public updateConversationTitle(String id, String newTitle): void
- Public updateConversationTranscript(String id, String newTranscript): void

- Public updateCustomDescription(String id, String newCustomDescription): void
- Public updateGptDescription(String id, String newGptDescription): void
- Public updateGptReminders(String id, String newGptReminders): void
- Public updateGptFoodOrder(String id, String newGptFoodOrder): void
- Public filterConversations(String searchText): List<Conversation>
- Public setSortingType(SortingType sortingType): void
- Public getConversationsFromJsonFile: List<Conversation>
- Public writeConversationsToJsonFile(): void
- Public deleteAudioFile(String id): void

Class Name: Conversation
Class Description/Purpose: Stores all data regarding an audio recording made by the app.
Class Modifiers: Public
Class Inheritance: none
Class Attributes:

- Public id: String
- Public recoredDate: DateTime
- Public audioFilePath: String
- Public duration: Duration
- Public transcript: String
- Public customDescription: String
- Public gptDescription: String
- Public gptReminders: String
- Public gptFoodOrder: String
- Public userId: String
- Public location: String

Exceptions Thrown:

- failedWriteOperationException()

Class Constructors: Conversation(String id, String audioFilePath, DateTime recordedDate, Duration duration, String title, String transcript, String customDescription, String gptDescription, String gptReminders, String gptFoodOrder)
Class Methods:

- Public toJson(): String
- Public saveToFile()

Class Name: Reminder
Class Description/Purpose: Store of information pertaining to an individual reminder event.
Class Modifiers: Public
Class Inheritance: none
Class Attributes:

- Public reminderId: int
- Public createTimestamp: int
- Public notifyTimestamp: int
- Public reminderDescription: String
- Public userId: String
- Public guid: String

Exceptions Thrown:
Class Constructors: Reminder (): No parameter required.
Class Methods:

- Public toJson(): String

Class Name: GptCalls
Class Description/Purpose: Contains methods for calling ChatGPT
Class Modifiers: Public
Class Inheritance: none
Class Attributes:

- Private _openAIApiKey: String
- Private _logger: Logger
- Private resturantPrompt: String
- Private descriptionPrompt: String
- Private summaryPrompt: String
- Private reminderPrompt: String

Exceptions Thrown:
Class Constructors: GptCalls(String openAIApiKey)
Class Methods:

- Public getOpenAiSummary(): String
- Public extractFormValuesFromTranscript(): String
- Public getRestaurantOrder(): String
- Public getReminders(): String

Class Name: GptReminder
Class Description/Purpose: Contains method for calling ChatGPT to get a list of reminders
Class Modifiers: Public
Class Inheritance: none
Class Attributes:

- Private _openAIApiKey: String
- Private _logger: Logger
- Private reminderPrompt: String

Exceptions Thrown:
Class Constructors: GptReminder(String openAIApiKey)
Class Methods:

- Public getOpenAiReminderList (): String

Method: audioFileUpload(File audioFile): String
Method Description/Purpose: Uploads a file to API endpoint on AWS and returns a status message.

Class Name: BEService
Class Description/Purpose: Contains methods for interacting with the browser extension service (BESie)
Class Modifiers: Public
Class Inheritance: none
Class Attributes:

- Private _storage: JsonStorage
- Private _appInstanceCodeFileName: const String
- Private _logger: Logger
- Private _request: BERequest?

Exceptions Thrown:
Class Constructors: BEService(JsonStorage storage)
Class Methods:

- Public handleRequestFrame(): void
- Public storeRequest(BERequest request): void
- Public getLastRequest(): BERequest?
- Public parseRequestFromFrame(StompFrame frame): BERequest?
- Public saveAppInstanceCode(String instanceCode): void
- Public loadAppInstanceCode(): String

Class Name: LocalStorageService
Class Description/Purpose: Contains method for loading and saving objects a JSON formatted files in a single directory.
Class Modifiers: Public
Class Inheritance: JsonStorage
Class Attributes:

- Private basePath: String

Exceptions Thrown:
Class Constructors: LocalStorageService (String basePath)
Class Methods:

- Public doesExist(String filename): bool
- Public loadFromPath(String filename): bool
- Public saveToPath(String filename): bool

Class Name: WebSocketClient
Class Description/Purpose: Contains methods for interacting with a WebSocket server
Class Modifiers: Public
Class Inheritance: none
Class Attributes:

- Private _wsUrl: String

- Private _wsConnectionTimeoutMs: int
- Private _listeners: List<WebSocketListener>
- Private _logger: Logger
- Private _isConnected: bool = false

Exceptions Thrown:
Class Constructors: WebSocketClient(String wsUrl, int wsConnectionTimeoutMs, List<WebSocketListener> listeners)
Class Methods:

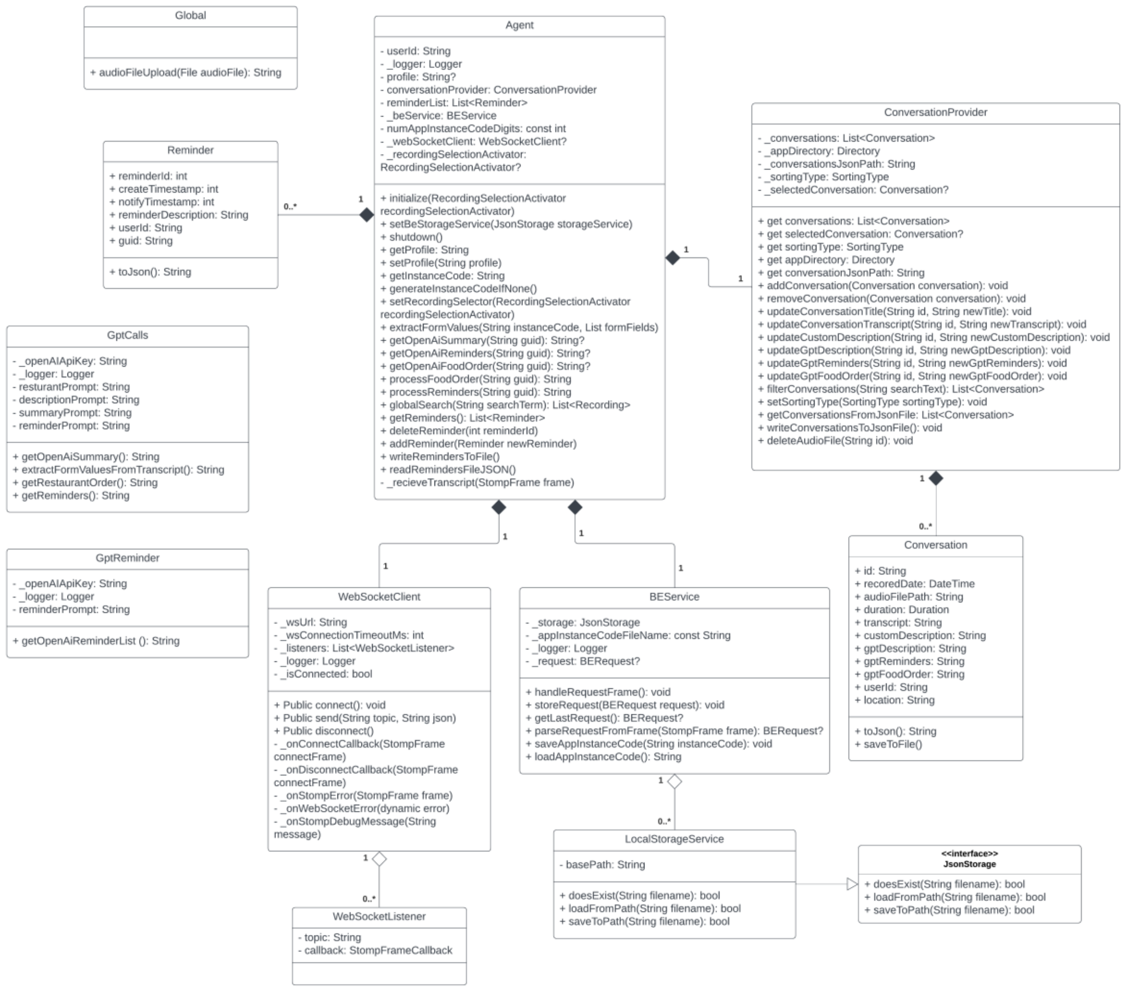
- Public connect(): void
- Public send(String topic, String json)
- Public disconnect()
- Private _onConnectCallback(StompFrame connectFrame)
- Private _onDisconnectCallback(StompFrame connectFrame)
- Private _onStompError(StompFrame frame)
- Private _onWebSocketError(dynamic error)
- Private _onStompDebugMessage(String message)

Class Name: WebSocketListener
Class Description/Purpose: Contains a callback method for parsing messages received on a WebSocket connection
Class Modifiers: Public
Class Inheritance: none
Class Attributes:

- Private topic: String
- Private callback: StompFrameCallback

Exceptions Thrown:
Class Constructors: WebSocketListener(String topic, StompFrameCallback callback)
Class Methods:

Figure 6. STeMS Backend Services Class Diagram


Global
+ audioFileUpload(File audioFile): String
Reminder
+ reminderId: int
+ createTimestamp: int
+ notifyTimestamp: int
+ reminderDescription: String
+ userId: String
+ guid: String
+ toJson(): String
0..*
1
GptCalls
- _openAIApiKey: String
- _logger: Logger
- resturantPrompt: String
- descriptionPrompt: String
- summaryPrompt: String
- reminderPrompt: String
+ getOpenAiSummary(): String
+ extractFormValuesFromTranscript(): String
+ getRestaurantOrder(): String
+ getReminders(): String
GptReminder
- _openAIApiKey: String
- _logger: Logger
- reminderPrompt: String
+ getOpenAiReminderList (): String
Agent
- userId: String
- _logger: Logger
- profile: String?
- conversationProvider: ConversationProvider
- reminderList: List<Reminder>
- _beService: BEService
- numAppInstanceCodeDigits: const int
- _webSocketClient: WebSocketClient?
- _recordingSelectionActivator: RecordingSelectionActivator?
+ initialize(RecordingSelectionActivator recordingSelectionActivator)
+ setBeStorageService(JsonStorage storageService)
+ shutdown()
+ getProfile: String
+ setProfile(String profile)
+ getInstanceCode: String
+ generateInstanceCodeIfNone()
+ setRecordingSelector(RecordingSelectionActivator recordingSelectionActivator)
+ extractFormValues(String instanceCode, List formFields)
+ getOpenAiSummary(String guid): String?
+ getOpenAiReminders(String guid): String?
+ getOpenAiFoodOrder(String guid): String?
+ processFoodOrder(String guid): String
+ processReminders(String guid): String
+ globalSearch(String searchTerm): List<Recording>
+ getReminders(): List<Reminder>
+ deleteReminder(int reminderId)
+ addReminder(Reminder newReminder)
+ writeRemindersToFile()
+ readRemindersFileJSON()
- _recieveTranscript(StompFrame frame)
1
1
ConversationProvider
- _conversations: List<Conversation>
- _appDirectory: Directory
- _conversationsJsonPath: String
- _sortingType: SortingType
- _selectedConversation: Conversation?
+ get conversations: List<Conversation>
+ get selectedConversation: Conversation?
+ get sortingType: SortingType
+ get appDirectory: Directory
+ get conversationJsonPath: String
+ addConversation(Conversation conversation): void
+ removeConversation(Conversation conversation): void
+ updateConversationTitle(String id, String newTitle): void
+ updateConversationTranscript(String id, String newTranscript): void
+ updateCustomDescription(String id, String newCustomDescription): void
+ updateGptDescription(String id, String newGptDescription): void
+ updateGptReminders(String id, String newGptReminders): void
+ updateGptFoodOrder(String id, String newGptFoodOrder): void
+ filterConversations(String searchText): List<Conversation>
+ setSortingType(SortingType sortingType): void
+ getConversationsFromJsonFile: List<Conversation>
+ writeConversationsToJsonFile(): void
+ deleteAudioFile(String id): void
1
0..*
Conversation
+ id: String
+ recoredDate: DateTime
+ audioFilePath: String
+ duration: Duration
+ transcript: String
+ customDescription: String
+ gptDescription: String
+ gptReminders: String
+ gptFoodOrder: String
+ userId: String
+ location: String
+ toJson(): String
+ saveToFile()
1
1
1
1
WebSocketClient
- _wsUrl: String
- _wsConnectionTimeoutMs: int
- _listeners: List<WebSocketListener>
- _logger: Logger
- _isConnected: bool
+ Public connect(): void
+ Public send(String topic, String json)
+ Public disconnect()
- _onConnectCallback(StompFrame connectFrame)
- _onDisconnectCallback(StompFrame connectFrame)
- _onStompError(StompFrame frame)
- _onWebSocketError(dynamic error)
- _onStompDebugMessage(String message)
1
0..*
WebSocketListener
- topic: String
- callback: StompFrameCallback
BEService
- _storage: JsonStorage
- _appInstanceCodeFileName: const String
- _logger: Logger
- _request: BERequest?
+ handleRequestFrame(): void
+ storeRequest(BERequest request): void
+ getLastRequest(): BERequest?
+ parseRequestFromFrame(StompFrame frame): BERequest?
+ saveAppInstanceCode(String instanceCode): void
+ loadAppInstanceCode(): String
1
0..*
LocalStorageService
- basePath: String
+ doesExist(String filename): bool
+ loadFromPath(String filename): bool
+ saveToPath(String filename): bool
<<interface>>
JsonStorage
+ doesExist(String filename): bool
+ loadFromPath(String filename): bool
+ saveToPath(String filename): bool

6 HUMAN INTERFACE DESIGN

This is covered by the scope of Team A.

7 REQUIREMENTS MATRIX

SRS – Requirement Number	Requirements Description	Component Number	Component Name
3.1.1	Edit Recording Metadata	5.1	Agent
3.1.2	Get Recording Metadata	5.1	Agent
3.1.3	Search Recordings	5.1	Agent
3.1.4	List Recordings	5.1	Agent
3.1.5	Delete Recording	5.1	Agent

3.1.6	Save Recording	5.1	Recording
3.1.7	Convert Speech to Text	5.1	Agent
3.1.8	Generate App Instance Code	5.1	Agent
3.1.9	Get App Instance Code	5.1	Agent
3.1.10	Extract Form Values	5.1	Agent
3.1.11	Process Request for Food Orders (Waiter)	5.1	Agent
3.1.12	Process Request for Reminders (STML)	5.1	Agent
3.1.13	Global Search	5.1	Agent
3.1.14	Get User Profile	5.1	Agent
3.1.15	Set User Profile	5.1	Agent
3.1.16	Push Reminder Notifications	5.1	Reminder

8 APPENDICES

The following appendices provide additional information about the approaches used or analysis of different technologies that could be used to provide development solutions.

8.1 Appendix A: Speech to Text Comparison

Speech-to-text technology, also known as automatic speech recognition (ASR) or computer speech recognition has many applications, including transcription services, and accessibility for people with impairments. This Appendix includes a comparison of some of the most popular free speech-to-text APIs for convenience. The APIs listed are free to use but may have limitations and/or usage restrictions. The accuracy of STT conversions may vary depending on multiple factors including the specific API used, and the audio input quality. It's recommended to test the APIs and evaluate their performance before using them in a production environment. This appendix section is not to be considered part of the requirements.

Name	Pros	Cons
Amazon Transcribe https://aws.amazon.com/getting-started/hands-on/create-audio-transcript-transcribe/	• Can differentiate between maximum of 10 unique speakers. • Uses deep learning models to recognize speech.	• Free tier includes monthly time limit (60 min). • Asynchronous transcription audio files must be stored in S3 bucket.

	• Able to support multiple languages. • Free tier with 12 hours of transcribing time per month. • Impressive client list including: Netflix, Airbnb, and Dow Jones	
Google Cloud Speech-to-Text API https://cloud.google.com/speech-to-text/docs/how-to	• Provided by Google Cloud. • Uses deep learning models to recognize speech. • Able to support multiple languages. • Support for diarization: recognition of multiple speakers available • Free tier allows for 60 min of transcribing time per month. • Impressive client list including: Spotify, Snapchat, and HSBC.	• Free tier includes monthly time limit (60 min). • Considerably less usage available in free tier when compared to Amazon Transcribe and Microsoft Azure. • Asynchronous transcription audio files must be stored in Google Cloud Storage Bucket
Microsoft Azure https://learn.microsoft.com/en-us/azure/cognitive-services/speech-service/batch-transcription	• Provided by Microsoft. • Uses deep learning models to recognize speech. • Able to support multiple languages. • Free tier allows for 5 hrs. of transcribing time per month.	• Free tier includes monthly time limit. • Considerably less usage available in free tier when compared to Amazon Transcribe. • Asynchronous transcription audio files must be stored in Azure Blob

		Storage or URL
Houndify	• Provided by SoundHound Inc. • Uses deep learning models to recognize speech. • Able to support multiple languages. • Free tier available allowing for 100 requests per month. • No mention of time limits for free.	• Free tier includes monthly transaction limit. • Free tier limited to only 100 requests per month.
OpenVINO	• Provided by Intel. • Open-source. • Widely used in research community and in the development of open-source projects. • Able to support multiple languages. • Not a web-based API, but instead an offline speech recognition toolkit.	
AssemblyAI https://www.assemblyai.com/why-assemblyai	• Several hours per month of free-tier Speech to Text • Support for Diarization, Punctuation and Casing, text summarization	• Asynchronous transcription is limited to 10 hours, 5GB. Local file uploads are capped at 2.2GB.

8.2 Appendix B: Analysis of WebSocket Server Hosting Options

The following section describes various hosting options for the browser extension service (BESie).

What are WebSockets?
WebSockets are bi-directional, full-duplex communication on a single TCP connection, commonly used between a browser and a server. They remove the need for polling and almost every browser supports WebSocket (Caniuse.com, n.d.). They are typically used for chat applications, transmitting real-time data, and interactive experiences. Some tests have found that WebSockets tend to be more performant when high loads are present than REST services (Gamage, 2017).

How do WebSockets work?
The web part of WebSocket is the initial request for a WebSocket connection along with a handshake between client and server. Uses a WebSocket key to negotiate a connection and identify concurrent clients. A somewhat obfuscated handshake is used to avoid allowing a WebSocket client from transmitting data to a basic HTTP service and possibly exposing private data (Mozilla, n.d.). Likewise, all elements of the client WebSocket request must be understood by the server to avoid possibly compromising client data by not providing a client required extension.

WebSocket APIs
WebSocket APIs allow for the creation of real-time, bi-directional APIs. They allow messages to be sent to a server and event-driven responses to be sent back without having to constantly reconnect or use more unreliable methods like long polling. Large cloud computing companies like Amazon and Microsoft offer WebSocket API gateways or similar offerings that provide a hosted, scalable WebSocket server in the cloud (Amazon, n.d.a). Other smaller companies offer more niche services that tend toward a freemium model, one where some initial offering is free but to add scale or capability, a monthly subscription must be paid for higher tier service.

What about Serverless?
WebSocket servers and serverless are a bit of a contradiction. WebSocket allows clients to connect and to maintain that connection, knowing the WebSocket server is on the other end of the line waiting for the next message (Netlify, n.d.). However, cloud computing companies have figured out how to provide the full time WebSocket server, and then allow you to decouple your code from the WebSocket server, either as a separate server for processing messages or serverless functions.

Hosting Options
There are three hosting methods to be considered when choosing how to host a WebSocket server. They are managed WebSocket API service, hosted web server, and self-hosted web server. Each approach has pros and cons associated.

Managed WebSocket API Service

Pros	Low or no code, don't have to create or manage hosting environment, ability to test multi-user scenarios, unlikely to reach free tier limits in single user scenario, security is built in.
Cons	Less control over the code/how the service works, shared environment could mean slower response times and capacity limitations

Hosted Web Server

Pros	Complete control over the code, no need to install and configure a hosting platform such as docker desktop.
Cons	Limited processing power, still have to provide all the code.

Self-Hosted

Pros	Complete control over the code, can fully utilize local resources (CPU/IO) with considerably higher performance than a free tier VM, easier to debug/troubleshoot than a remote service.
Cons	Requires creating and maintaining a hosting platform on each developer machine, borrows resources from the development computer which will usually be running a mobile emulator as well, more issues to troubleshoot managing the hosting platform as well as the hosted container.

Table 1. Managed Services

	Free Tier	Paid Tier
AWS WebSocket API Gateway	one million messages and 750,000 connection minutes for WebSocket APIs per month for up to 12 months (Amazon, n.d.b)	$1 for first billion messages $0.25 per million connection minutes (Amazon, n.d.b)
Azure Web PubSub	20 concurrent connections 20,000 messages per day (Microsoft, n.d.)	1,000 concurrent connections unlimited messages (free for first 1M) $1.61 per day $1 per million messages (Microsoft, n.d.)
Netlify	100 GB bandwidth 300 build minutes 1M edge function invocations (Netlify, n.d.)	$19/month Pro 1TB/month bandwidth 2m/month edge function invocations (Netlify, n.d.)
PieSocket	Tester and Free Channel 200,000 messages/day 200 concurrent connections (PieSocket, 2021)	$29/month 1,000,000 messages/day 1,000 concurrent connections (PieSocket, n.d.)

Rapid Development and Deployment

Cloud hosting companies offer web service hosting, typically using a Linux platform and free web stacks such as express and node.js. These services can be used to host a small 24/7 virtual machine appropriate for a WebSocket server, such as Render, Heroku, and Digital

Ocean.

Digital Ocean offers tutorials to get a WebSocket server off the ground (Digital Ocean, 2022), as well Source repository on GitHub that you can fork (Normore, 2020). Google cloud documentation includes a walkthrough of building and deploying a WebSocket server on their cloud platform using their own deployment tool, "gcloud" (Google Cloud, 2023). Heroku offers a video tutorial on how to host a free WebSocket server on their platform (Blankensmith, 2022).

Table 2. Web Service Hosting Options

Host	Length of Free Period	RAM & CPU in free or near free tier with unlimited usage	Cost of free or near free tier with unlimited usage
Google Cloud	3 months	2 vCPUs, 1 GB RAM	Free
Amazon Web Services	12 months	1 vCPU, 1 GB RAM	Free
Microsoft Azure	12 months	1 vCPU, 1 GB RAM	Free
Render	Permanent	0.5 vCPU	$7/month
Heroku	N/a	1 vCPU	$7/month
Digital Ocean	Permanent	1 vCPU, 500 GiB Transfer	$4/month

(Jones, 2023)

Docker
Docker expands the set of possibilities for hosting further. Docker brings virtual-machine like creation and management to the developer's desktop through the use of containers. Containers are lightweight packages of software with all the pieces necessary to create a runtime environment. Containers run on top of a standardized virtualized environment, providing the flexibility to run in different hosts operating systems without change, but without a full operating system themselves like a virtual machine image (Netapp, n.d.). You can host Docker locally with Docker Desktop or use container hosting services provided by Amazon (EC2 Container Service) and Microsoft (Azure Container Instances) (Docker, n.d.c). Soketi gives you a working WebSocket server with a single command using docker desktop. There is also the option to host your Soketi server with Cloudflare, though you have to set up a domain and it costs about $12/month (Soketi, n.d.).

Self-Hosting
Self-hosting options are by their nature at no additional cost, beyond the development environment that is already available. The downside is that the development environment may not be set up consistently between developers and additional platform tools will likely have to be installed, such as Docker Desktop or npm/node.js.

Docker Compose
Using the YAML configuration file, a single command will spin up a docker instance with

everything configured to run a service in their development environment (Docker, n.d.b). Docker Compose is particularly useful when orchestrating multiple servers, but still provides flexibility and ease of use for a single instance. It also simplifies deployment to multiple environments, such as "production, staging, development, testing, as well as CI workflows" (Docker, n.d.b). Creating an environment from scratch sounds resource intensive, but containers are only rebuilt on change and caching of resources means base images only must be downloaded once on a developer's machine.

Local Server

Given the single user scenario, a simple local instance of node.js or similar service would suffice. An instance of the service could be started from the command line. This has the advantage of avoiding the Docker technology stack but may slow conversion to a multiuser environment in the future. A counter argument would be the YAGNI principle, "you aren't gonna need it", which says that investing time in future development often doesn't pay off because either the future feature is never needed or is provided in a manner different than original conceived (Fowler, 2016).

8.3 Appendix C: References

Amazon (n.d.a) About WebSocket APIs in API Gateway. https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-websocket-api-overview.html

Amazon (n.d.b) Amazon API Gateway pricing. https://aws.amazon.com/api-gateway/pricing/

Blankensmith, T. (October 2, 2022). Tutorial 2 - Host a free WebSocket server on Heroku. control TD with a website using WebSockets. https://www.youtube.com/watch?v=m5k8d4bmoUU

Caniuse.com. (n.d.). Websockets. https://caniuse.com/websockets

Digital Ocean (August 11, 2022). Deploy an app using WebSockets to app platform. https://docs.digitalocean.com/tutorials/app-deploy-websockets/

Docker (n.d.a). Docker Desktop. https://docs.docker.com/desktop/

Docker (n.d.b). Docker Compose. https://docs.docker.com/compose/

Docker (n.d.c). Deploy your app. https://docs.docker.com/language/java/deploy/

Fowler, M. (May 26, 2016). YAGNI. https://martinfowler.com/bliki/Yagni.html

Gamage, T. A. (November 19, 2017). HTTP and WebSockets: understanding the capabilities of today's web communication technologies. https://medium.com/platform-engineer/web-api-design-35df8167460

Google Cloud (June 6, 2023). Building a WebSocket chat service for cloud run tutorial. https://cloud.google.com/run/docs/tutorials/websockets

Jones, R. (May 28, 2023). 8 Best Free Cloud Hosting Services: Entirely Risk-Free in 2023. https://www.websiteplanet.com/blog/best-free-cloud-hosting-services/

Mozilla (n.d.). Writing WebSocket servers. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_servers

NetApp (n.d.). What are containers? https://www.netapp.com/devops-solutions/what-are-containers/

Netlify (November 15, 2022). WebSockets in a serverless world. https://www.netlify.com/blog/web-sockets-in-a-serverless-world/

Netlify (n.d.). Netlify pricing. https://www.netlify.com/pricing/

Normore, S. (October 12, 2020). Snormore/sample-websocket. https://github.com/snormore/sample-websocket

PieSocket (December 30, 2021). How to use PieSocket's WebSocket server for free. https://www.piesocket.com/blog/free-websocket-server

PieSocket (n.d.). PieSocket pricing. https://www.piesocket.com/pricing

Soketi (n.d.) Soketi. https://soketi.app

University of Maryland Global Campus. (n.d.). *Previous projects.* https://umgc-cappms.azurewebsites.net/previousprojects