DREAM TEAM

TECHNOLOGY SOLUTIONS

CogniOpen Software Application

Technical Design Document

Project Name: CogniOpen Software Application
Document: Technical Design Document (TDD)
Version: 3.0
Date: November 7, 2023
Prepared By: Team A

Team Lead / Project Manager:
Vincent Galeano
Dream Team Technology Solutions (DTTS)
vgaleano1@student.umgc.edu

Client:
Dr. Mir Assadullah
University of Maryland Global Campus (UMGC)
mir.assadullah@faculty.umgc.edu

Contributors: Vincent Galeano, Kavon Johnson, Zach Bowman, David Bright, Juan Torres-Chardon, Laura Hamann, Vivek Goel, Selam Biru, Benjamin Sutter, Richard Tsang, & Andrea Pellot

Leads Sign-off Sheet

Key Reviewer	Version	Date	Signature
Vincent Galeano Team Lead / Project Manger	1.0	23/Sep/2023	Vincent Galeano
Kavon Johnson Lead Technical Writer	1.0	23/Sep/2023	Kavon Johnson
Zach Bowman Lead Business Analyst	1.0	23/Sep/2023	Zach Bowman
David Bright Architect / Lead Software Developer	1.0	23/Sep/2023	David Bright
Juan Torres-Chardon Lead UI/UX Designer	1.0	23/Sep/2023	Juan Torres-Chardon
Laura Hamann Lead Test Engineer	1.0	23/Sep/2023	Laura Hamann
Vincent Galeano Team Lead / Project Manger	2.0	28/Oct/2023	Vincent Galeano
Kavon Johnson Lead Technical Writer	2.0	28/Oct/2023	Kavon Johnson
Zach Bowman Lead Business Analyst	2.0	28/Oct/2023	Zach Bowman
David Bright Architect / Lead Software Developer	2.0	28/Oct/2023	David Bright
Juan Torres-Chardon Lead UI/UX Designer	2.0	28/Oct/2023	Juan Torres-Chardon
Laura Hamann Lead Test Engineer	2.0	28/Oct/2023	Laura Hamann
Vincent Galeano Team Lead / Project Manger	3.0	07/Nov/2023	Vincent Galeano
Kavon Johnson Lead Technical Writer	3.0	07/Nov/2023	Kavon Johnson
Zach Bowman Lead Business Analyst	3.0	07/Nov/2023	Zach Bowman
David Bright Architect / Lead Software Developer	3.0	07/Nov/2023	David Bright
Juan Torres-Chardon Lead UI/UX Designer	3.0	07/Nov/2023	Juan Torres-Chardon
Laura Hamann Lead Test Engineer	3.0	07/Nov/2023	Laura Hamann

Revision History

Date	Version	Description	Author
03/Sep/2023	0.1	Document created	DTTS
23/Sep/2023	1.0	Milestone 2 version	DTTS
28/Oct/2023	2.0	Milestone 3 version	DTTS
07/Nov/2023	3.0	Milestone 4 version	DTTS

Table of Contents

List of Figures & Tables

1 Introduction

The Technical Design Document (TDD) for the CogniOpen software application provides an in-depth technical perspective on the development and architecture of the CogniOpen software application. This document serves as a critical resource for the project team, stakeholders, and developers involved in building and maintaining the software application.

1.1 Purpose

The primary purpose of this TDD is to:

1.1.1 Provide Technical Clarity

This document aims to elucidate the technical aspects of the CogniOpen software application. It delineates the architectural design, data management, class elements, user interface considerations, and other essential technical details.

1.1.2 Guide Development

It serves as a guiding blueprint for developers, helping them understand the intricacies of the system's design and how different components interact. This knowledge is invaluable for efficient development and future enhancements.

1.1.3 Ensure Consistency

Consistency in design and development practices is crucial for delivering a cohesive and reliable software application. This TDD establishes design principles and technical standards to maintain uniformity across the project.

1.1.4 Facilitate Maintenance

A well-documented technical design simplifies maintenance and troubleshooting. Developers and support teams can refer to this document to comprehend the system's purpose, structure, and functionality.

1.1.5 Assist Stakeholders

Stakeholders, including project managers, clients, and quality assurance teams, can use this document to gain insight into the technical underpinnings of the application, aiding in project management and quality assurance efforts.

1.2 Scope

The scope of this TDD encompasses all technical aspects of the CogniOpen software application. It provides comprehensive insights into architecture, data management, user interface design, and other significant technical considerations.

1.2.1 In Scope

The following technical aspects are within the scope of this document:

- System Architecture: Detailed descriptions of the system's architectural components and their interactions.
- Data Design: Entity Relationship Diagrams (ERDs), data dictionaries, and data flow diagrams.
- Class Design: Detailed information on the design of critical software components.
- Human Interface Design: User interface design and functionality.
- Development, Test, Deployment, & Operations: Considerations for the development, testing, deployment, and operational phases.
- Issues & Risks: Identification of potential technical issues and risks.

1.2.2 Out of Scope

This document does not cover non-technical aspects such as project management, marketing, or financial considerations. Additionally, it does not provide detailed coding instructions but rather focuses on high-level design and technical decisions.

This document does not cover the technical design details for Team B’s implementation of their project deliverables for the application product. The features covered by Team B (core module, assistant module, and conversation module features) are not covered in this technical design document.

1.3 Constraints

The development of the CogniOpen Software Application operates within certain constraints:

- Timeline: The project must adhere to the agreed-upon timeline for development and delivery.
- Budget: Financial constraints are in place to manage project costs effectively.
- Technological: The application must function within the technical constraints of the selected technologies and platforms.
- Regulatory: Compliance with relevant data protection and privacy regulations is mandatory.

1.4 Assumptions & Dependencies

This document is based on the following assumptions and dependencies:

- The project team has access to the required hardware and software resources.
- Developers have a fundamental understanding of the chosen technologies.
- Dependencies on external services or components are documented and managed correctly.

1.5 Document Conventions & Organization

To ensure clarity and consistency, this document follows specific conventions:

- Section Numbering: Sections are numbered hierarchically, following a format like "1.1" or "1.2.1."

- References: Citations and references follow the project's standard reference style, which will be detailed in the "References" section.
- Terminology: Industry-standard terms and abbreviations are used, and a glossary of terms is provided in Section 1.9.

1.6 Intended Audience & Reading Suggestions

This document is intended for a diverse audience, including:

- Developers: For in-depth technical guidance and insights.
- Project Managers: To understand the technical intricacies of the project.
- Quality Assurance Teams: To aid in test planning and execution.
- Stakeholders: To gain a high-level understanding of the technical aspects of the application.

Reading suggestions:

- Developers and technical team members may focus on sections related to system architecture and component (database, class, UI) design.
- Project managers and stakeholders may prioritize the system overview and scope sections.
- Quality assurance teams can refer to sections related to testing, issues & risks, and requirements matrix.

1.7 Project Document Suite

This TDD is part of a suite of project documents that collectively provide comprehensive project documentation. The suite includes:

Document	Version	Date
Project Plan (PP)	4.0	07/Nov/2023
Software Requirements Specification (SRS)	4.0	07/Nov/2023
Technical Design Document (TDD)	3.0	07/Nov/2023
Test Plan (TP)	3.0	07/Nov/2023
Programmer Guide (PG)	2.0	07/Nov/2023
Deployment and Operations Guide (Runbook)	2.0	07/Nov/2023
User Guide (UG)	1.0	07/Nov/2023
Test Report (TR)	1.0	07/Nov/2023

Table 1: Project Document Suite

Additional project documents, including the UG and TR, will be developed as the project progresses.

1.8 References

James, D. (n.d). *User Interface Design Template*. SWEN 661: User Interface Implementation, University of Maryland Global Campus (UMGC). https://learn.umgc.edu/d2l/le/content/920760/viewContent/31091164/View

UMGC. (2023). *Previous projects.* SWEN 670: Software Engineering Capstone, University of Maryland Global Campus (UMGC). https://umgc-cappms.azurewebsites.net/previousprojects

1.9 Terms, Abbreviations, & Acronyms

Term	Definition
AI	Artificial Intelligence
ALAC	Act-like-a-customer
API	Application Programming Interface
Application	A subsystem within the software system that implements all modules and is divided into the presentation and business layers
AWS	Amazon Web Services
CI/CD	Continuous Integration, Continuous Deployment
ChatGPT	Chat Generative Pre-trained Transformer
Class Element	Internally developed functionalities grouped together that serve as the building blocks for realizing use-cases
Component	Externally sourced functionalities grouped together that serve as the building blocks for realizing use-cases
Data System	A subsystem within the software system which includes services that are responsible for local and non-local data storage and retrieval
DFD	Data Flow Diagram
DTTS	Dream Team Technology Solutions
ERD	Entity Relationship Diagram
Functionality	A discrete operation or capability the software system can perform
KPIs	Key Performance Indicators
LLM	Large Language Model
Module	Logical collections of user-facing use-cases
PP	Project Plan
REST	Representational State Transfer
S3	Simple Storage Service
SDK	Software Development Kit
SQL	Structured Query Language
SRS	Software Requirements Specification
STML	Short Term Memory Loss
STP	Software Test Plan
STT	Speech to Text

Term	Definition
Subsystem	An abstract container that groups class elements and components within the system
System	Software system; overarching abstract container that encapsulates and orchestrates all modular components and data
TDD	Technical Design Document
UI	User Interface
UMGC	University of Maryland Global Campus
Use-case	System features; specific scenarios that describe how a user, or an external system, interacts with the software system to achieve a particular goal or outcome
UX	User Experience
Web Services	A subsystem within the software system which includes services that are responsible for data processing tasks for text, image, audio, and video

Table 2: Terms, abbreviations, and acronyms

2 System Overview

2.1 Summary

The CogniOpen software application is designed to assist anyone affected by short term memory loss (STML). The application's purpose is to aid in recalling events, objects, and obligations that occur during daily life. Users share video and audio recordings with the application, and the application will return lost items, transcribe conversations, and remind the user of past conversations. Through these features, the CogniOpen software application will ease some of the burden from caregivers of those affected by STML.

2.2 Technical Approach

While the application will support multiple features and functionalities, not all these features will be developed and implemented by the Dream Team Technical Solutions firm (DTTS). DTTS will be implementing the video and gallery modules of the system; also, they will contribute towards the base code functionality and design as well as extra features, including the location integration. Team B will be producing the remaining features which are associated with the assistant module and the conversation module; they also will be implementing most of the core functionality (e.g., main menu, settings, navigation, help).

Nevertheless, both teams will operate under the same architectural overview and layered design with an application (presentation and business layers), webservices, and relational database persistence data services. Both teams will develop feature functionalities in tandem across teams' functional domains, focusing on the database design, then configuring the web services and backend business logic, and finally the application interface.

2.3 Technical Tools & Resources

Presentation UI (User Interface) and application services implemented in Dart and Flutter. Web services accessible via REST (Representational State Transfer) API (Application Programming Interface). Database stores in a local SQLite database and cloud Amazon Web Services (AWS) S3 buckets.

- GitHub
- Dart v3.1.0
- Flutter v3.13.1
- Pencil v3.1.0
- Android Studio Giraffe 2022.3.1 Patch 1 with Android SDK v34.0.0
- AWS S3
- AWS Transcriber
- AWS Rekognition
- OpenAI ChatGPT
- Sqflite 2.3.0

2.4 System Overview Diagram

The system is composed of several components. The user interacts with the system via the application loaded onto a mobile device. From the application, users can access the web services capable of transcribing speech to text (STT), analyzing text, and analyzing video. Additionally, the user's processed key data, calendar events, lists, and reminders are stored locally on the device, and they can be accessed through the application. The user's previous audio and video recordings are stored in AWS S3 (simple storage service) buckets which are accessible through the application as well. As seen in the following figure, the application is the central hub of the system.

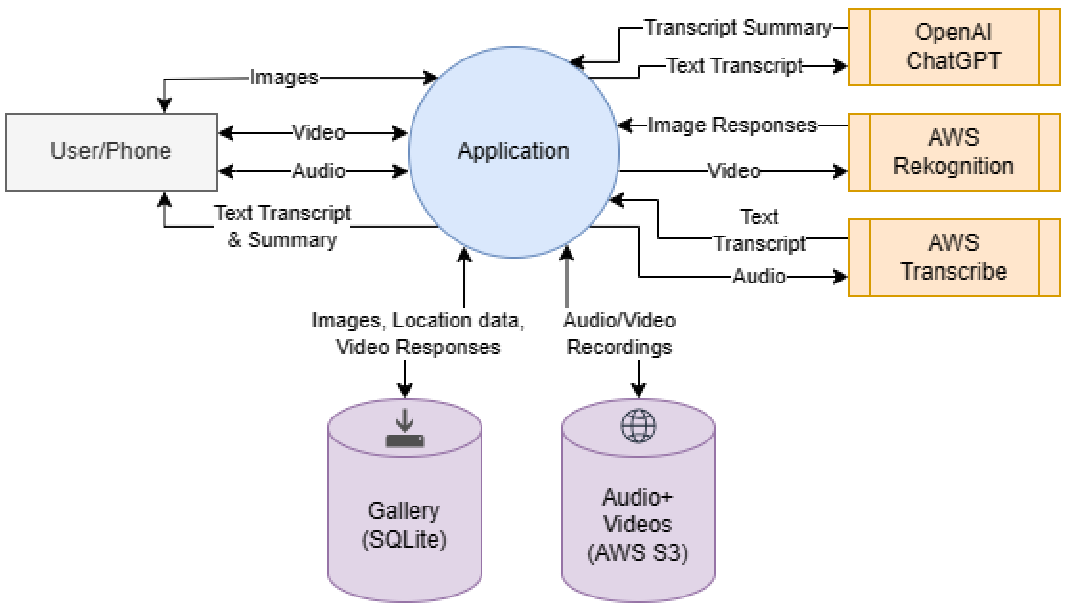


Figure 1: System Overview Diagram

2.5 Design Rationale

The rationale behind our system design is the utilization of existing solutions that meet our needs. For the functional requirements of processing audio conversations, we needed a STT solution; for speech diarization, we needed a large language model (LLM) AI solution; for video processing, we needed an object-detection AI solution. For data persistence, we analyze what data would be stored in the system and decided on a local database storage need for the smaller data demands (lists, reminders, and calendar events through the user's native app) and a remote database storage that could elastically meet the need of the larger data demands (videos and audio recordings). The chosen solutions were communicated with the peer development team to ensure a cohesive product design (and reduce risks associated with integrating the two teams' development efforts). The existing solutions that meet these needs are explained in the following degree:

- LLM AI – OpenAI ChatGPT was chosen for our speech diarization needs due in part to developer familiarity and the pricing structure. Our developers (across both teams) are familiar with this product, there exists abundant learning material online on the subject, and the basic pricing structure of the solution will enable our team to develop our product at little to no cost.

- STT – AWS Transcribe was chosen for our speech to text needs due to compatibility with other solutions we chose to utilize. The pricing structure of Amazon’s web services will enable us to utilize various solutions (STT, object-detection, and elastic storage) in their “Free tier”, keeping the implementation costs of this project to a minimum. This solution was also chosen for its breadth of functionality, supporting multiple languages, should this product be used by users outside of English speakers. Finally, previous project teams have used this solution, so our development teams have learning resources available to inspect implementation techniques.

- Object-detection – AWS Rekognition was chosen for our video processing needs due to compatibility with other solutions we chose to utilize. As mentioned above, Amazon's web services free tier allows us to minimize project costs. This solution was also chosen for its ability to recognize more objects than competitor solutions and in a more robust fashion (video files that may have been imported as rotated by 90°).

- Remote data persistence – AWS S3 was chosen for our elastic data demands due to compatibility with other solutions we chose to utilize. As mentioned above, Amazon's web services free tier allows us to minimize project costs. Our developers are familiar with this product, and there is abundant learning material online to assist in any implementation hurdles. Also, by utilizing a solution service, we reduce the complexity of setting up a database server ourselves and lower the maintenance of operating the system long-term.

- Local data persistence – SQLite was chosen for our smaller data demands to reduce the complexity of the system. The smaller demand of the location data, significant objects, and Rekognition responses user case functionalities allows the design team to keep these items local to the user’s device where they are accessible while offline. Additionally, keeping the data locally does not increase our remote data usage, keeping the utilization costs lower.

Of note in our design rationale is the insistence of the design team in keeping backend services on the device. As described in more detail in the following Section 3 “Architectural Design”, the application is composed of both a presentation layer and a business layer. This business layer is where the various solutions chosen as components of our system design will interact with our application interface. The design team could have put these backend request processing services on a remote web server but decided against this approach due to the additional complexity without ascertainable return on investment nor performance. Also, due to the limited time schedule of this project, a “combined” application design of both a presentation layer and business layer met the product needs.

3 Architectural Design

This section outlines the architectural design of the software system with respect to its organizational structure and constituent parts. In effort to ensure long-term success of the software system, including organized code and ease of implementation, testing, debugging, scalability, and use, the architectural design employs a modular approach. The software system makes use of various modular elements that can be composed of or integrated with others which creates a hierarchical structure that further enhances logical organization and flexibility.

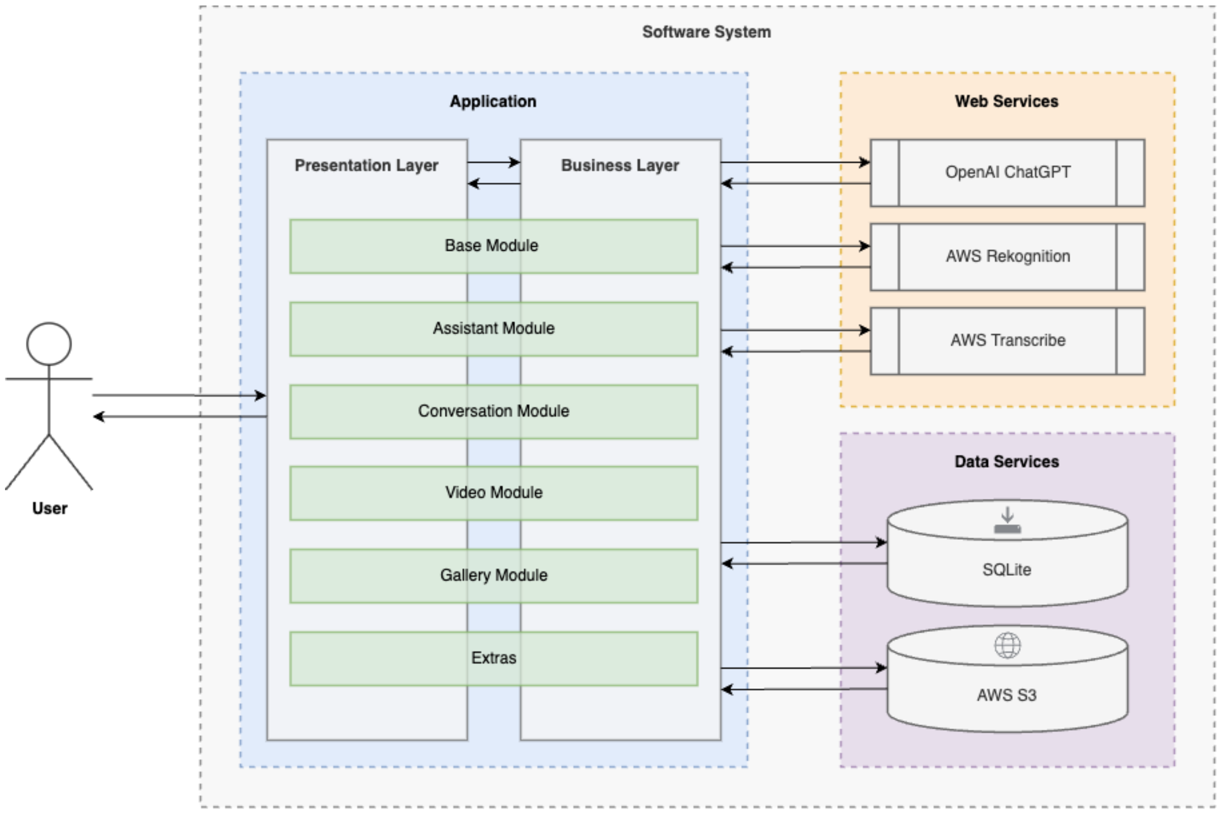


Figure 2: Architectural Design Diagram

The software system is the overarching abstract container that encapsulates and orchestrates all modular components and data to collectively achieve the goals and objectives of the project. Naturally, the overall technical design is primarily guided by system requirements and use-cases, which are detailed in the SRS document. Use-cases, also referred to as system features, are specific scenarios that describe how a user, or an external system, interacts with the software system to achieve a particular goal or outcome. Modules are defined as logical collections of user-facing use-cases. A functionality is a discrete operation or capability the software system can perform; functionalities are grouped together to form components and class elements that serve as the building blocks for realizing use-cases. Class elements are developed internally, while components are sourced externally.

The software system is comprised of subsystems, which are abstract containers that group class elements and components. Specifically, there is a subsystem called the application that is divided into the presentation and business layers that implement all modules. Alternatively, other subsystems are simply referred to as services and are the web and data services. The organization and hierarchy of the key modular elements are visually represented above in the Architectural Design Diagram, which extends the System Overview Diagram from Section 2.4.

The following subsections describe the specific subsystems, error, and exception handling, as well as provide the decomposition description for the software system.

3.1 Application

As mentioned above, the application is a subsystem within the software system that implements all modules and is divided into the presentation and business layers. The presentation layer is responsible for managing the user interface (UI) and user experience (UX). It interfaces directly with end-users and communicates with the business layer to retrieve, format, and display data or services in a user-friendly and effective manner. The business layer serves as the intermediary between the presentation layer and backend services to provide the required system features and data. It is responsible for executing business logic, rules, and workflows, and ensures that requests and data are processed appropriately to meet requirements.

The presentation layer will be developed using Flutter, and the business layer will be developed using Dart.

3.2 Web Services

Web services is a subsystem within the software system which includes services that are responsible for data processing tasks for text, image, audio, and video; it also incorporates artificial intelligence for enhanced functionality. These technologies include speech to text, large language models, and object detection machine learning algorithms that we would rather utilize than create for this solution. The business layer within the application manages all integrations with the web services.

Web services include the OpenAI ChatGPT, AWS Rekognition, and AWS Transcribe services.

3.3 Data Services

Data services is a subsystem within the software system which includes services that are responsible for local and non-local data storage and retrieval. The local database service will utilize an SQLite database for saving application data to a file on the user's device. The non-local database storage will be an AWS S3 saving the (larger) user's video and audio files on a cloud server. The business layer within the application manages all integrations with its services and instances.

Data services include SQLite and AWS S3.

3.4 Error and Exception Handling

Error and exception handling will ensure system stability and user satisfaction. User inputs will be validated on the front-end, providing immediate feedback on required fields and formats. Backend validation will prevent unacceptable inputs via API requests. In case of user generated errors, the user will receive an error notification or be guided to an error page. App crashes may be monitored by services such as Google Firebase's Crashlytics to diagnose and resolve issues promptly. Exception handling will be conducted to resolve common exceptions such as NullPointerException, FileNotFoundException, IOException, and InterruptedException.

3.5 Decomposition Description

This subsection provides a detailed decomposition description of the software system, breaking it down into its modules and use-case functionalities. Below, the individual modules are detailed, outlining their primary functions, interfaces, relationships, and interactions.

Note that Team A (DTTS) is primarily responsible for implementing the video and gallery modules, thus they are detailed in this document. Team B is primarily responsible for implementing the base, assistant, and conversation modules, so they are detailed in their complementary TDD.

3.5.1 Video Module

The video module is the collection of video recording-related use-cases implemented in the software system which are accessible to users as system features via the application. The video module includes the following use-case functionalities:

- Record a video
- Enable passive video recording mode

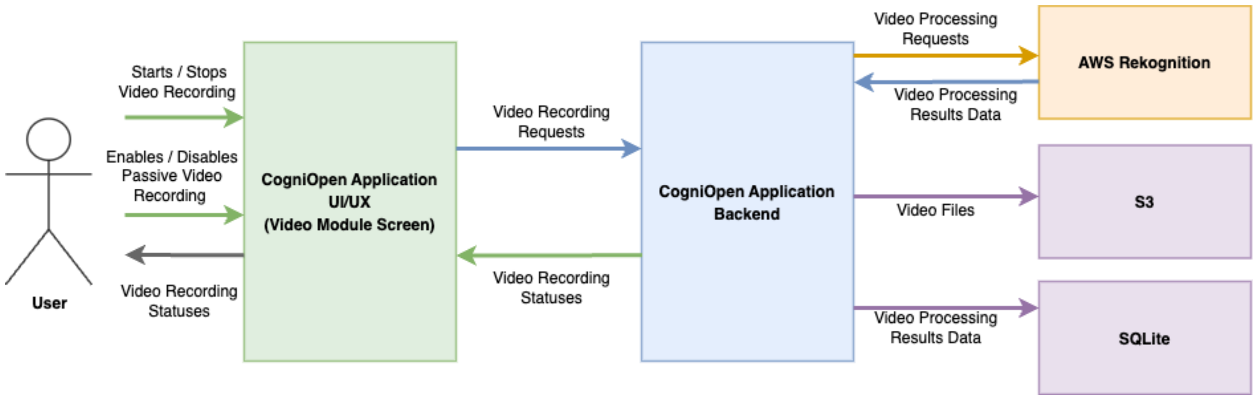

User
Starts / Stops Video Recording
Enables / Disables Passive Video Recording
Video Recording Statuses
CogniOpen Application UI/UX (Video Module Screen)
Video Recording Requests
Video Recording Statuses
CogniOpen Application Backend
Video Processing Requests
Video Processing Results Data
AWS Rekognition
Video Files
S3
Video Processing Results Data
SQLite

Figure 3: Video Module Diagram

3.5.2 Gallery Module

The gallery module is the collection of media data-related use-cases implemented in the software system which are accessible to users as system features via the application. The gallery module includes the following use-cases functionalities:

- Search, view, and edit media (and associated key data)
- Add additional media (including images)
- Search, view, and edit significant items (and associated key data)
- Set up new significant items

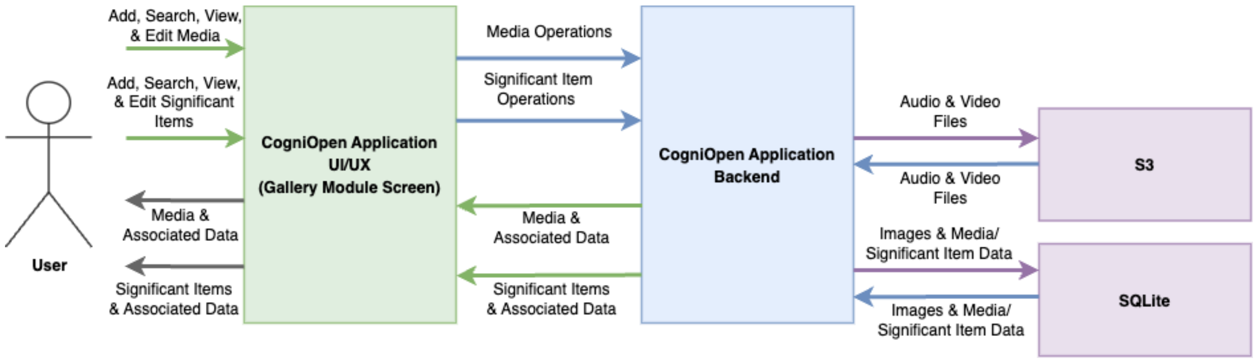

Add, Search, View, & Edit Media
Add, Search, View, & Edit Significant Items
Media & Associated Data
Significant Items & Associated Data
User
CogniOpen Application UI/UX (Gallery Module Screen)
Media Operations
Significant Item Operations
Media & Associated Data
Significant Items & Associated Data
CogniOpen Application Backend
Audio & Video Files
Audio & Video Files
Images & Media/ Significant Item Data
Images & Media/ Significant Item Data
S3
SQLite

Figure 4: Gallery Module Diagram

4 Data Design

This section will delve into the fundamental aspects of data design for the CogniOpen application. Proper data design is crucial to ensure efficient data management, storage, and retrieval, which are integral to the application's functionality.

4.1 Entity Relationship Diagram (ERD)

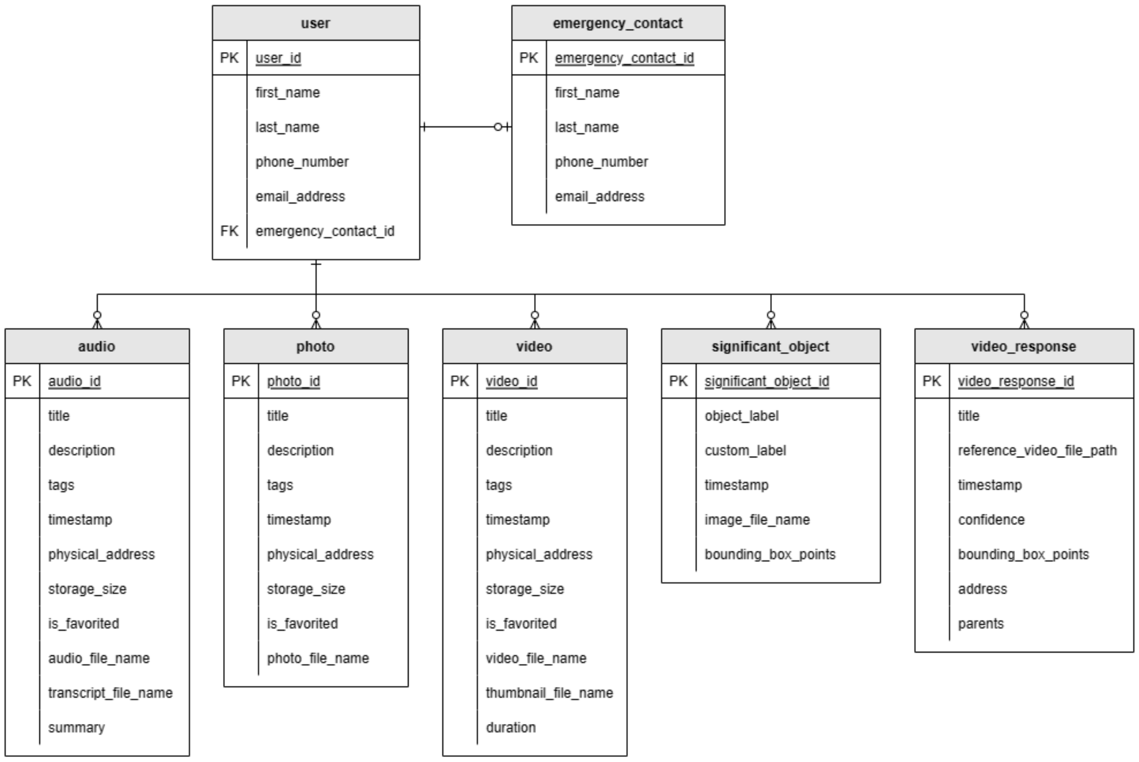


Figure 5: Entity Relationship Diagram

The ERD of the CogniOpen application is designed to capture the essential entities and their relationships, ensuring that data is structured logically and efficiently. The ERD encompasses the following key components:

User Entity: This entity represents the user of the CogniOpen application. It stores user-specific information, such as the user's phone number, email address, and emergency contact.

EmergencyContact Entity: This entity stores information about emergency contacts that users can list within the application for quick access during emergency situations. Each emergency contact record consists of a unique Contact_ID, the first name and last name of the contact, their email address, and phone number.

Audio Entity: This entity represents audio files that users may associate with various data points within the application, primarily with conversations. Each audio file is uniquely identified by an Audio_ID and includes attributes for its size, location, transcription file, and summary of the conversation.

Photo Entity: This entity represents images or photographs that users may associate with various data points within the application, primarily with SignificantObjects. Each photo is uniquely identified by a Photo_ID and includes attributes for file name, size, and location.

Video Entity: This entity represents videos that the users may associate with various data points within the application, primarily with Responses. Each video is uniquely identified by a Video_ID and includes attributes for its duration, size, and location.

SignificantObject Entity: Represents significant objects that users want to store information about, such as where they are kept or their significance. Each significant object is characterized by a unique Sig_Obj_ID, a descriptive SO_Name, references or images related to the object stored in the referencePhotos attribute, and potential alternative names or descriptions in the alternateNames field.

Video Response Entity: The Response entity embodies the system's generated responses to user queries. Responses have a unique identifier in "Video Response_ID". Pertinent information here is the “reference_video_file_path”, “timestamp”, and “bounding_box_points” which are all useful in recreating the stillframe image to show to the user the detected object. Title is the title of the object, confidence is how sure Rekognition is that the detected object is correct

4.1.1 Entity Descriptions

This subsection will provide detailed descriptions of each entity included in the ERD:

User Entity

- Attributes: User_ID (Primary Key), first_name, last_name, phone_number, emergency_contact_id (Foreign Key), email_address.
- Relationships:
    - User-Audio (One-to-Many): Each user can have multiple audio recordings, but each audio recording belongs to only one user. This relationship links users to their recorded conversation.
    - User-Photo (One-to-Many): Users can capture multiple photographs, but each photograph is associated with one user. This relationship connects users to their many photos.
    - User-Video (One-to-Many): Each user can create multiple videos of whatever they many be recording throughout their day to day lives; each video is associated with but one user. This relationship connects the users to their many videos.

- User-Significant_Object (One-to-Many): Users can store information about significant objects, and each object is associated with a user.
- User-Video_Response (One-to-Many): Users can store information about video responses, and each object is associated with a user.
- User-Emergency_Contact (One-to-Many): Each user can have multiple emergency contacts listed.

EmergencyContact Entity

- Attributes: Emergency_Contact_ID (Primary Key), User_ID (Foreign Key), First_Name, Last_Name, Phone_Number, Email_Address
- Relationships:
  - EmergencyContact-User (Many-to-One): Many emergency contacts can be associated with one user, indicating that users can have multiple emergency contacts.

Audio Entity

- Attributes: Audio_ID (Primary Key), Title, Description, Tags, Timestamp, Physical_Address, Storage_Size, Is_Favorited, Audio_File_Name, Transcript_File_Name, Summary
- Relationships:
  - Audio-User (Many-to-One): Many audio files can be associated with one user, but only one user.

Photo Entity

- Attributes: Photo_ID (Primary Key), Title, Description, Tags, Timestamp, Physical_Address, Storage_Size, Is_Favorited, Photo_File_Name
- Relationships:
  - Photo-User (Many-to-One): Many photos can be associated with one user, but only one user.

Video Entity

- Attributes: Video_ID (Primary Key), Title, Description, Tags, Timestamp, Physical_Address, Storage_Size, Is_Favorited, Video_File_Name, Thumbnail_File_Name, Duration

- Relationships:
    - Video-User (Many-to-One): Many video files can be associated with one user, but only one user.

SignificantObject Entity

- Attributes: Sig_Obj_ID (Primary Key), User_ID (Foreign Key), Object_Label, Custom_Label, Timestamp, Image_file_name, Bounding_Box_points
- Relationships:
    - SignificantObject-User (Many-to-One): Many significant objects can be associated with one user, indicating that users can store information about multiple significant objects.

Video_Response Entity

- Attributes: Video_Response_ID (Primary Key), Title, Reference_Video_File_Path, Timestamp, Confidence, Bounding_Box_Points, Address, Parents
- Relationships:
    - Response-User (Many-to-One): Multiple responses can be associated with a single usre. Each response is linked to one user. This reflects that a user can receive multiple responses, and all those responses are connected to the same user.

4.2 Data Dictionary

This section contains a detailed description of the database entities along with examples of each entry and its associated table within the database. This dictionary will be used as a reference for database implementation and development.

4.2.1 Emergency Contact Table

Field Name	Data Type	Data Format	Field Size	Description	Example
First Name	String	Text	60	The Emergency Contact’s first name.	Tom
Last Name	String	Text	60	The Emergency Contact’s last name.	Petty

Field Name	Data Type	Data Format	Field Size	Description	Example
Phone Number	String	Text	13	Phone number to contact	1-301-244-9486
Email Address	String	Text	120	Email address for application notifications	“tompetty4ever@ hotmail.com”

Table 3: Emergency Contact Table Data Dictionary

4.2.2 Audio Table

Field Name	Data Type	Data Format	Field Size	Description	Example
Title	String	Text	60	The title for the piece of media.	“My Cousin Richard’s Birthday Party”
Description	String	Text	500	A description of the media.	“70th birthday. This was taken at the Royal Caribbean Cruise.”
Tags	String[]	Comma-separated values	N/A	A list of tags related to the media for sorting/querying.	[“travel”, “cruise”]
Timestamp	DateTime	ISO 8601 DateTime format	N/A	The date and time associated with the media.	"2023-09-15T14:30:00Z"
Physical_address	String	Text	N/A	Street Address Location of where the audio file was recorded	123 Main Street, Big City, MD 21711
storageSize	Int	Megabytes (MB)	N/A	The size that the media is taking up on local storage.	28

Field Name	Data Type	Data Format	Field Size	Description	Example
isFavorited	Bool	True/False	N/A	Determines if the user has favorited the media. A favorite media will have an extra icon next to it and can be filtered.	True
Audio_File_Name	String	Text	N/A	File location for the audio file	//s3:birthday_party.mp4
Transcrip_File_Name	String	Text	N/A	File location for the file containing the audio recording transcription	/transcript/birthday_party.mp4
Summary	String	Text	N/A	LLM processed understanding of the audio transcription	"At your cousin Richard's birthday party, you thanked him for inviting you along with the cruise. You wished him a Happy Birthday, and he thanked for the kind words."

Table 4: Audio Table Data Dictionary

4.2.3 Photo Table

Field Name	Data Type	Data Format	Field Size	Description	Example
Title	String	Text	60	The title for the piece of media.	"My Cousin Richard's Birthday Party"
Description	String	Text	500	A description of the media.	"70th birthday. This was taken at the Royal Caribbean Cruise."

Field Name	Data Type	Data Format	Field Size	Description	Example
Tags	String[]	Comma-separated values	N/A	A list of tags related to the media for sorting/querying.	[“travel”, “cruise”]
Timestamp	DateTime	ISO 8601 DateTime format	N/A	The date and time associated with the media.	"2023-09-15T14:30:00Z"
Physical_address	String	Text	N/A	Street Address Location of where the audio file was recorded.	123 Main Street, Big City, MD 21711
storageSize	Int	Mega bytes (MB)	N/A	The size that the media is taking up on local storage.	28
isFavorited	Bool	True/ False	N/A	Determines if the user has favorited the media. A favorite media will have an extra icon next to it and can be filtered.	True
Photo_File_Name	String	Text	N/A	File location for the photo file	//s3:birthday_party.jpg

Table 5: Photo Table Data Dictionary

4.2.4 Video Table

Field Name	Data Type	Data Format	Field Size	Description	Example
Title	String	Text	60	The title for the piece of media.	“My Cousin Richard’s Birthday Party”
Description	String	Text	500	A description of the media.	“70th birthday. This was taken at the Royal Caribbean Cruise.”
Tags	String[]	Comma-separated values	N/A	A list of tags related to the media for sorting/querying.	[“travel”, “cruise”]
Timestamp	DateTime	ISO 8601 DateTime format	N/A	The date and time associated with the media.	"2023-09-15T14:30:00Z"
Physical_address	String	Text	N/A	Street Address Location of where the audio file was recorded.	123 Main Street, Big City, MD 21711
storageSize	Int	Megabytes (MB)	N/A	The size that the media is taking up on local storage.	28
isFavorited	Bool	True/False	N/A	Determines if the user has favorited the media. A favorite media will have an extra icon next to it and	True

Field Name	Data Type	Data Format	Field Size	Description	Example
				can be filtered.	
thumbnail_file_name	String	Text	N/A	File location for the video file.	//s3:birthday_party.mp4
Duration	String	Text	N/A	The duration of the video. String for human-legible format.	120

Table 6: Video Table Data Dictionary

4.2.5 SignificantObject Table

Field Name	Data Type	Data Format	Field Size	Description	Example
object_label	String	Text	60	The object title	“sunglasses”
custom_label	v	Text	60	User’s defined custom label	“my sunglasses”
timestamp	DateTime	ISO 8601 DateTime format	N/A	The date and time associated with the media.	"2023-09-15T14:30:00Z"
image_file_name	String	Text	N/A	File location for the image file.	//birthday_party.jpg
bounding_box_points	String	Numerical text	N/A	Coordinates for the starting corner of a bounding box, plus width and height of the box.	(0.10, 0.13, 0.25, 0.62)

Table 7: Significant Object Table Data Dictionary

4.2.6 VideoResponse Table

Field Name	Data Type	Data Format	Field Size	Description	Example
Title	String	Text	60	The object title	“sunglasses”
reference_ video_ file_path	String	Text	60	The path for the referenced video.	//s3:birthday_party video.mp4
timestamp	DateTime	ISO 8601 DateTime format	N/A	The date and time associated with the media.	"2023-09-15T14:30:00Z"
confidence	Double	Numbe	4	Rekognition value that object detected is the actual object	“86.75”
bounding_ box_points	String	Numerical text	N/A	Coordinates for the starting corner of a bounding box, plus width and height of the box	(0.10, 0.13, 0.25, 0.62)
address	String	Text	40	Street Address Location of where the audio file was recorded.	123 Main Street, Big City, MD 21711
parents	String	Text	120	List of object Parenets	“glasses, reading glasses, eyeglasses”

Table 8: Video Response Table Data Dictionary

4.3 Diagram 0 Data Flow Diagram (DFD)

The DFD serves as the top-level representation of the CogniOpen system's data flow. It identifies key external entities that interact with the application and processes that manipulate data. At this level, the DFD offers a simplified but comprehensive overview of the entire system.

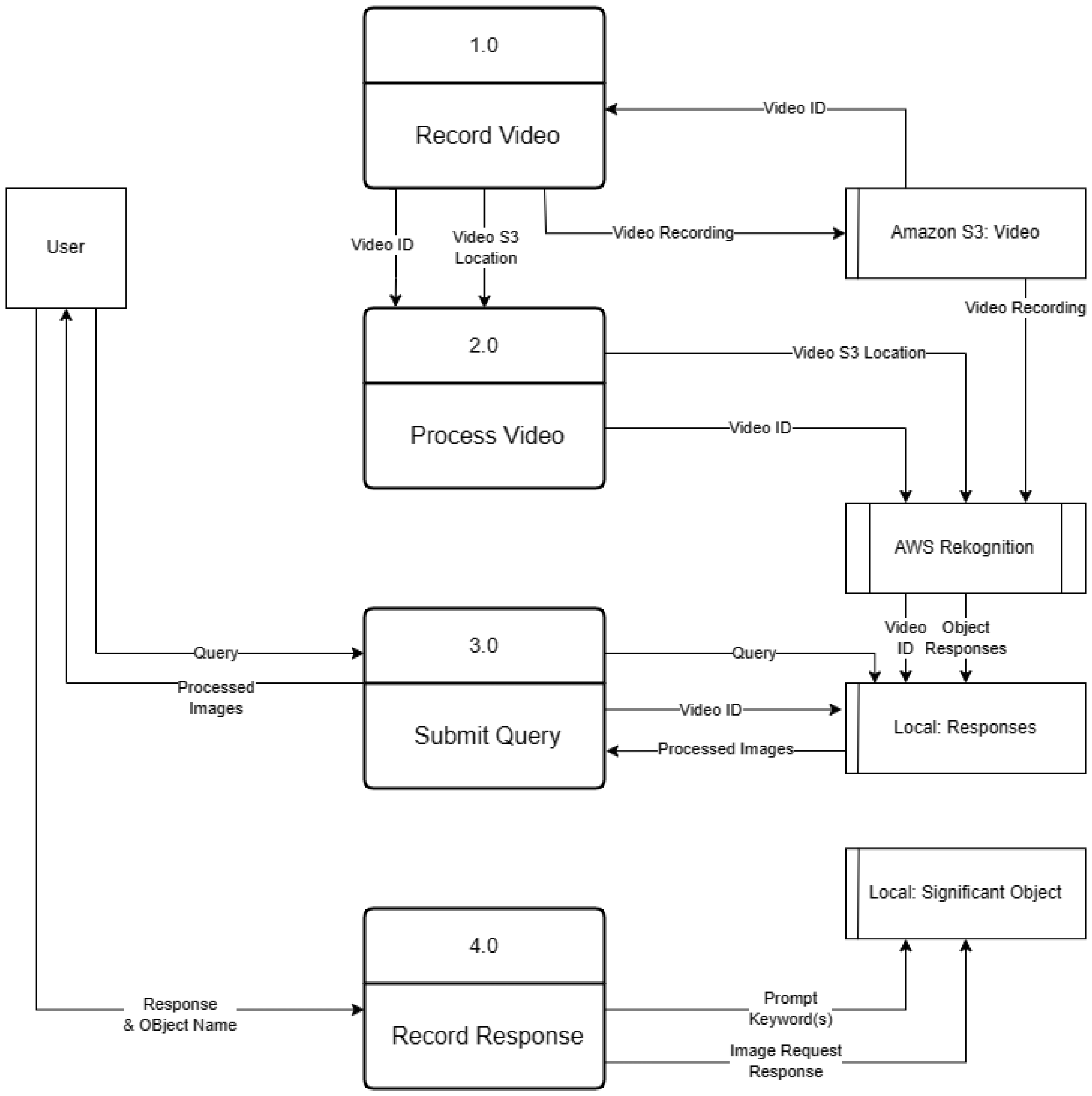


Figure 6: Data Flow Diagram (part I)

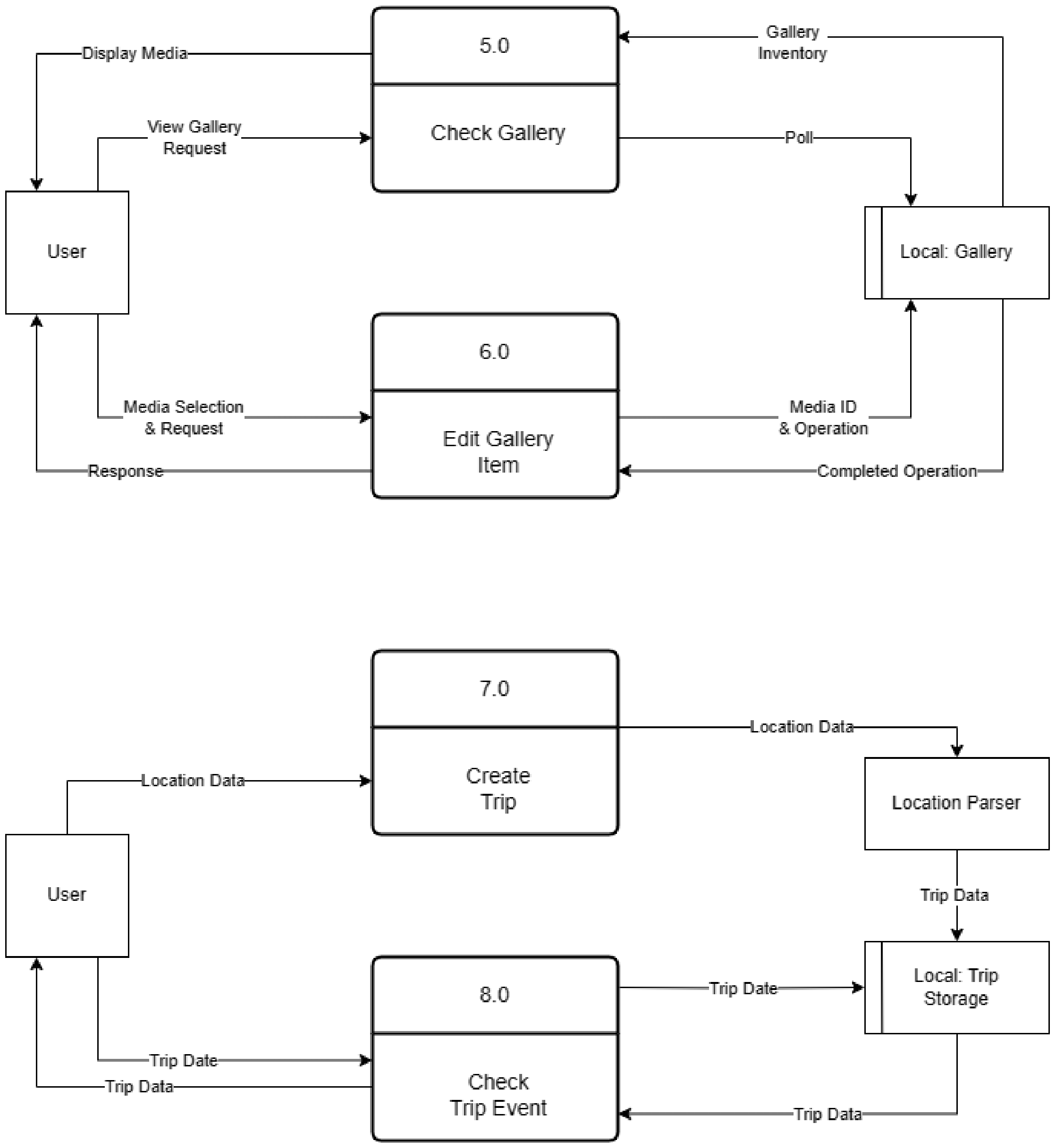


Figure 7: Data Flow Diagram (part II)

5 Class Element Design

Class Diagram: For a comprehensive view of the CogniOpen Software Application class diagram, please refer to the full-size image at class-diagram-full.svg. Additionally, for ease of reference, Appendix B includes the same class diagram divided into several segmented images, providing a more detailed look at each section of the diagram.

Aside from the menus, center, and other UI classes, most class elements covered in this section are data containers to help consolidate and organize data into a logical flow. Object inheritance is heavily utilized to allow for replication of repeated behaviors while also simplifying processes for the user and programming design. It is important to note that the CogniOpen software is quite vast with classes, attributes, and methods. This section will only cover the classes developed by team A of the project team.

5.1 Media (base class)

The media class is the parent class of all objects stored in the gallery view. It consolidates all the shared, common attributes and allows for easier containing and sorting amongst the various derived classes. The class has attributes for id, media type, title, description, tags, timestamp, physical address, storage size and favoritism.

5.2 Photo (extends Media class)

The photo is used as an object in the gallery to assist cognitive impaired individuals with memory recollection and details associated with that specific snapshot of time. Photo objects are also associated with identified items and significant objects to help the user and application visualize answers to any questions they may have. The class inherits the attributes and methods from the Media class but adds two more attributes, photo and file name. The class contains a method to load the photo.

5.3 Video (extends Media class)

The video class sets the properties and manipulates the components of the videos related to the application. Users can record videos (actively and passively) from their device that will be uploaded to the gallery. The system then parses through those video recordings to detect objects that the user is looking for. The class inherits all the attributes from the media class with the addition of storing others including video file name, thumbnail file name, thumbnail image and duration of the video.

5.4 Audio (extends Media class)

The application has some audio functionality that enhances the capabilities of the system. Users can record conversations that, when complete, are fully transcribed and displayed to the user. There is also the ability for users to interact with the virtual assistant, Cora, to ask questions and get feedback. Inheriting the attributes of media, it also has attributes for audio file name, transcription file name and conversation summary. It has similar methods to the other child classes of media with the addition of a method to load the transcription.

5.5 Location Data Model

The location data model class stores information related to a location gathered from the user’s device. Location is used to determine where objects are found in a recorded video. This helps the user locate their lost items compared to just showing them where they were last seen. The class has attributes for id, latitude, longitude, address and timestamp.

5.6 Significant Object

Significant objects are items that the user identifies as things to look for in video recordings. The user can upload photos to the application and train the AI to locate those objects in the videos. The accuracy of those objects found in videos is also displayed in percentage to the user. The class has attributes for id, object label, custom label, timestamp, image file name and dimensions of the bounding box (left, top, width and height).

5.7 Video Response

The video response class stores data related to the following: id, title, reference video file path, time stamp, confidence and a few dimensions. The confidence attribute is the AI’s confidence that the object identified is the significant object that the user is looking for. The dimensional attributes (left, top, width, and height) relate to the bounding box that hovers over the identified object in a video file.

5.8 Photo Controller

The photo controller class is responsible for many things related to a photo. The class adds gallery metadata to a photo: title, description, tags and the file of the photo. The class also calculates the size of the file. The class contains the functionality to update and remove a photo in the repository.

5.9 Significant Object Controller

The significant object controller class adds significant objects to the repository so that the user can query video files to find those items. The class contains methods to update and remove significant objects from the repository.

5.10 Video Controller

The video controller class adds gallery metadata to a video. This data includes title, description, tags, video file itself, the thumbnail and duration of the video. This is gallery data that the user will be able to use to query through their database of video files. The class calculates the size of the video file, creates a new video and adds that to the video repository. The class can also update and remove videos from the repository.

5.11 Video Response Controller

The video response controller class adds video responses to the system. A video response consists of a title, reference video file path, AI confidence accuracy, time stamp and dimensions for the bounding box that identifies an object in a video. The class also has methods that update and remove video responses from the video response repository.

5.12 Audio Controller

The audio controller class adds gallery metadata to an audio file. This data includes title, description, tags, audio file itself, the transcript file and summary of the audio recorded. This is gallery data that the user will be able to use to query through their database of audio files. The class calculates the size of the video file, gets the file extension, creates a new file that stores transcription and adds that to the transcription directory. The class can also update and remove videos from the repository. There are methods to add new audio, update an existing audio file and to remove an audio file from the repository.

5.13 Video Controller

The video controller class adds gallery metadata to a video. This data includes title, description, tags, video file itself, the thumbnail and duration of the video. This is gallery data that the user will be able to use to query through their database of video files. The class calculates the size of the video file, creates a new video and adds that to the video repository. The class can also update video metadata and remove videos from the repository.

5.14 Media Fields

The media fields class is an abstract class that the photo, video, and audio classes all inherit from. This abstract class maintains a list of string values. Those strings relate to id, title, description, tags, timestamp, storage size and favoritism. This is gallery metadata that the user will be able to use to query through their database of media files.

5.15 Photo Fields (extends Media Fields class)

The photo fields class inherits the attributes of the media fields class and adds one more, the photo file name. This data is essential for giving the user categories to query the galley with. It also plays a significant role in the backend functionality that deals with object recognition.

5.16 Significant Object Fields

The significant object fields class stores a list of strings related to significant objects, including id, label and image file name. This data is used to identify the objects that users upload to be able to locate within video files.

5.17 Video Fields (extends Media Fields class)

The photo fields class inherits the attributes of the media fields class and adds a few more. These extras include the video file name, the thumbnail file name and the duration of the video. This data is essential for giving the user categories to query the galley with. It also plays a significant role in the backend functionality that deals with identifying video captures of objects that are recognized and highlighted.

5.18 Video Response Fields

The video response fields class stores a list of strings related to a video response. That list includes the following: id, title, time stamp, confidence and bounding box dimensions and the reference video file path. label and image file name. This data is essential in the object recognition functionality of the software system.

5.19 Audio Fields (extends Media Fields class)

The audio fields class inherits the attributes of the media fields class and adds a few more. These extras include the audio file name, the transcription file name and the summary of the audio. This data is essential for giving the user categories to query the galley with. It also plays a significant role in displaying a transcription from a recorded audio to the user. The system trains the virtual assistant with the transcription so that the user can ask specific questions and get accurate feedback.

5.20 Location Fields (extends Media Fields class)

The location fields class inherits the attributes of the media fields class and adds a few more. These extras include the latitude, longitude, address and timestamp. This data is essential for enabling the location services in the application. Objects detected in recorded videos will be able to inform the user of where that item was last seen.

5.21 Audio Repository

The audio repository class is responsible for managing data in the audio table of the system’s database. The class hosts CRUD operations to interact with that particular database table.

5.22 Photo Repository

The photo repository class is responsible for managing data in the photo table of the system’s database. The class hosts CRUD operations to interact with that particular database table.

5.23 Significant Object Repository

The significant object repository class is responsible for managing data in the significant objects table of the system’s database. The class hosts CRUD operations to interact with that particular database table.

5.24 Video Repository

The video repository class is responsible for managing data in the video table of the system’s database. The class hosts CRUD operations to interact with that particular database table.

5.25 Video Response Repository

The video response repository class is responsible for managing data in the video response table of the system's database. The class hosts CRUD operations to interact with that particular database table.

5.26 Directory Manager

The directory manager class manages all the directories that are a part of the system. This class has a method to get a directory. It also will create a directory if it does not already exist. The different directories that can be created are the root, photos, videos, audio, transcripts, video thumbnails, video stills, video responses, significant objects and temporary directory.

5.27 File Manager

The file manager class manages the system's file system. The class holds two attributes: most recent video path and most recent video name. This class has many methods that can add files to the filesystem, remove files from the file system, update files in the file system, load and unload an asset file, calculate the size of a file, generate a file name, extract file extension from a file, get name of file without extension, load an image, load a file and list file names in the directory. There are also many accessor methods that can retrieve a file name, file time stamp, name of thumbnail file, thumbnail, most recent video and file name for AWS.

5.28 Format Utils

The format utils class is a unique class that formats the display of certain data that is outputted to the system's UI. The class has a method that can return the storage size of a file in many ways (bytes, kilobytes, megabytes and gigabytes). There are methods that can format the date and get the date from a timestamp. Ther are methods to format the time and calculation methods to get the difference of time in days and hours.

5.29 Permission Manager

The permission manager class manages the system permissions that the user will have to allow to get the CogniOpen fully functional on their device. When a user is going through the onboarding process, they must allow the system access to the device's camera, microphone, file storage and location services. The class also has methods to check if location services are active, check if camera permission was granted, check if file storage permission is granted and attempt to show video screen.

5.30 UI Utils

The UI utils class is responsible for some of the major UI components of the CogniOpen application. There is an accessor method that can retrieve the icons of different media types (audio, photo and video). There is another method that creates the bottom navigation bar which has the following items: home, virtual assistant, video and settings.

5.31 Address

The system stores addresses as gallery metadata to associate with different media files. This helps with locating a detected object from a video recording. There is also a location screen that tracks the location of a user's device with an associated timestamp. That is essentially the logic behind the class's single method: where I am.

5.32 AWS Video Response

The AWS video response class stores attributes for name, confidence, timestamp, bounding box, reference video file path, address and parents. The class also has a get and set method for the image of the video thumbnail.

5.33 Camera Manager

The camera manager class is responsible for handling the camera functionality of the application. Users grant permissions to their device's camera for the purpose of recording videos and taking photos that will be added to the gallery. The class contains objects that are unique to operating the camera as well as some Boolean attributes to determine if auto recording is active, to update to AWS Rekognition and initialization. There are methods to initialize the camera, parse environment settings, start auto-recording, stop recording, manually start and stop recording, start recording in the background, save media locally and to capture a photo.

5.34 Response Parser

The response parser class is responsible for parsing video responses gathered from the application. There is a method to get requested responses by video titles. There is another method to convert video responses to significant objects, locally. Lastly, there are some accessor methods to get a list of responses, get a thumbnail, get a timestamp from a video response and get hours from a video response.

5.35 S3 Bucket

The application uses AWS S3 buckets to store objects that connect with AWS Rekognition and Transcription services. There is a connection attribute that is used to set the connection to S3. There are methods to start the service, create a S3 bucket, add audio to S3, add an image to S3, add a file to S3, add a video to S3, add anything to S3 (which all add media methods use) and deleting something from S3 based on a passed in key value.

5.36 Video Display (extends stateful widget class)

Videos will need to be displayed in the gallery so that users can watch the playback of their video recordings. Videos are recognized by the thumbnail image that is associated with them. There will also be some metadata stored with videos. There is an attribute that stores the full file path of a video file locally.

5.37 Video Processor

With auto-recording enabled for the application, once the user stops the video recording, the system will have to process that video. There are many attributes in this class including confidence (for AWS Rekognition label detection service), Rekognition service, job id, available project, current project version, available models, active models, video title, address, video path, stopwatch and a list of excluded video responses. There is a method to get the elapsed time in seconds, a method to start the service, a method to automatically send to AWS Rekognition, a method to send Rekognition a video request to process the video and a method to create a response list. There are more methods to poll for complete video request, collect results for label detection, upload a video to S3, create a new AWS Rekognition custom label project, add a new model, poll for version description, poll for trained models, start and stop custom detection and find a matching model.

5.38 Data Service

The data service class is responsible for managing all the data associated with the various types of media stored in the gallery. There are data manipulations for videos, photos, audio files, video responses and significant objects. All the changes to the gallery metadata are controlled through the many methods that are a part of this class.

5.39 Gallery Screen

The gallery is a screen in the application that stores all the recorded content user captures. The gallery has audio files, photos and videos that are sorted by recency in a defaulted state. Users can change the view of the gallery screen based on different media filters like files that are favorited or not. The gallery view can also be manipulated but the display size of media files, which is controlled by a scroll bar. The actual metadata associated with media files can be updated in the gallery screen as well.

5.40 Location Screen

The location screen displays a log of all the locations that the user's mobile device was located at a given time (location history). Each log contains the physical address of where the phone was located, the date and a duration for how long the user was there. This will assist the user in tracking their whereabouts over a given time to help with finding loss items and helping them remember where they previously were. This functionality is also tied into the gallery module, associating media files with an address of the device when captured/recorded.

5.41 Object Search Screen

The object search screen displays some items that the AI model was able to detect based on recorded videos stored in the gallery. Objects are associated with a confidence percentage that reflects how sure the AI is with detecting a specific object. Users can look at the various detected objects and either confirm or deny if that is the item they were looking for. If it was, they then have the option to save the image as a significant object with a recognizable name. This leads to the AI model training functionality which enhances the user experience.

5.42 Home Screen

The home screen in the application is the central hub for all the user engagement activity within the application. The home screen is comprised of six buttons that manage the major functionality of the CogniOpen software. The virtual assistant button takes the user to a screen where they can chat with Cora, the AI assistant. The gallery button takes the user to a screen where all the recorded video, audio and photos are located. The object search button allows the user to look for specific items within the videos they have recorded. The record audio button will take the user to a screen where they can record conversations that will be transcribed. The location button takes the user to a screen where a log of addresses is displayed based on the location of the user's device at the time. Lastly the tour guide button takes the user through an application guide to show them how to use various system features.

6 Human Interface Design

The application User Interface (UI) will be developed using the latest version of the Flutter development framework and Dart programming language. Additionally, AWS S3 will be used for video storage and AWS Rekognition for object detection (video processing).

6.1 CogniOpen Interface

The following mock-up designs are the different modules and features found in the CogniOpen application. These include:

- Main Menu
- Video recording
- Audio recording
- Search Conversation History
- Significant Objects
- My Timeline
- Ask Question
- Tour Guide
- My Gallery
- Settings

6.1.1 Main Menu Screen

The following screen shall display the CogniOpen main menu items. Actors will be able to access all the application's features from this screen.

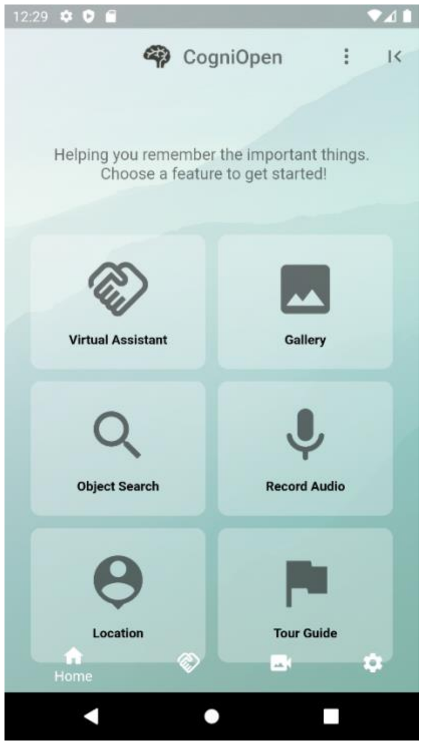


Figure 8: Main Menu screen

Internal UI functionality: When the actor selects the record button, the Record Menu screen shall open. When the actor selects the Search Conversation History button, the Conversation List screen shall open. When the actor selects the Significant Objects button, the Significant Objects screen shall open. When the actor selects the My Timeline button, the My Timeline screen shall open. When the actor selects the Ask Question button, the system shall begin listening to the user's audio query. When the actor selects the Tour Guide button, the CogniOpen Tour Screen shall open. When the actor selects the My Gallery button, the My Gallery screen shall open. When the actor selects the Settings button, the Settings Menu screen shall open.

External UI functionality: N/A

6.1.2 Video Recording Screen

The following screen shall display the video recording menu. The actor can select to start and stop an active video recording.

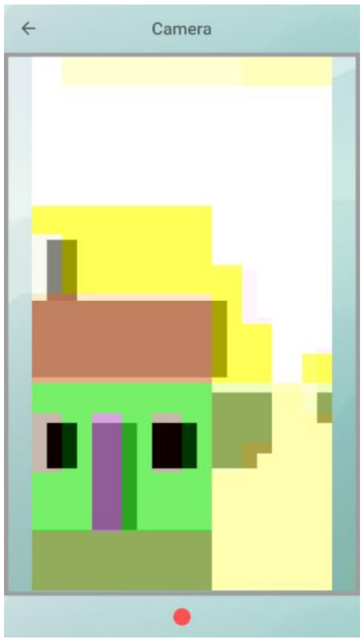


Figure 9: Video Recording screen

Internal UI functionality: When the actor first selects the record button, if there is no active recording in progress, the system shall start recording a video. If there is an active recording in progress, the system shall stop the recording.

External UI functionality: When the actor stops the video recording, the system shall save a copy of the video using AWS S3 service.

6.1.3 Significant Objects Screen

The following shall enable the actor to identify significant objects from images uploaded through the My Gallery screen.

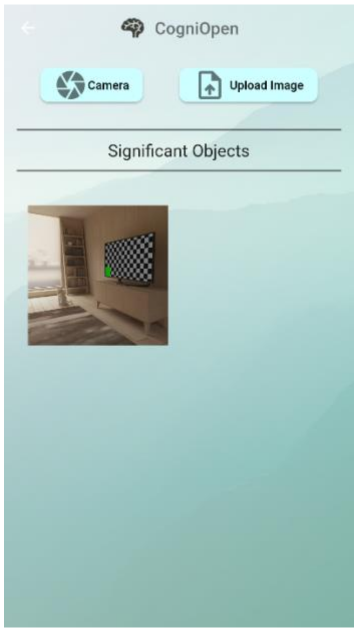


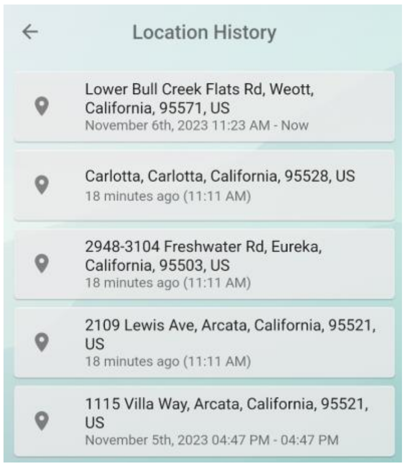
Figure 10: Significant Objects Screen

Internal UI functionality: The actor may select an image to declare as a significant object. After, the system shall open the Significant Object Details screen, where the actor may change and update the corresponding information.

External UI functionality: The system shall annotate the user's significant object and prioritize finding it over similar objects if a query was created.

6.1.4 Location Screen

The following shall display the actor's timeline in terms of locations visited with their corresponding duration.



Lower Bull Creek Flats Rd, Weott, California, 95571, US
November 6th, 2023 11:23 AM - Now

Carlotta, Carlotta, California, 95528, US
18 minutes ago (11:11 AM)

2948-3104 Freshwater Rd, Eureka, California, 95503, US
18 minutes ago (11:11 AM)

2109 Lewis Ave, Arcata, California, 95521, US
18 minutes ago (11:11 AM)

1115 Villa Way, Arcata, California, 95521, US
November 5th, 2023 04:47 PM - 04:47 PM

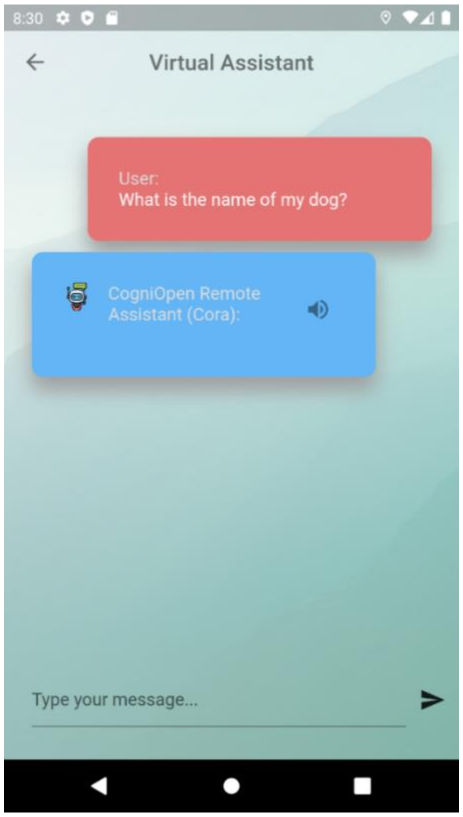
Figure 11: Location screen

Internal UI functionality: When the actor selects a timeline item, the system shall display the entry's corresponding information in the My Timeline Details screen.

External UI functionality: N/A

6.1.5 Virtual Assistant

This feature shall read the actor's request and display the corresponding information, if any.



User:
What is the name of my dog?

Figure 12: Virtual Assistant screen

Internal UI functionality: The system shall analyze the actor's audio request and display the corresponding information, if available.

External UI functionality: N/A

6.1.6 Tour Guide Screen

The following shall display a tour guide of CogniOpen's features for the actor.

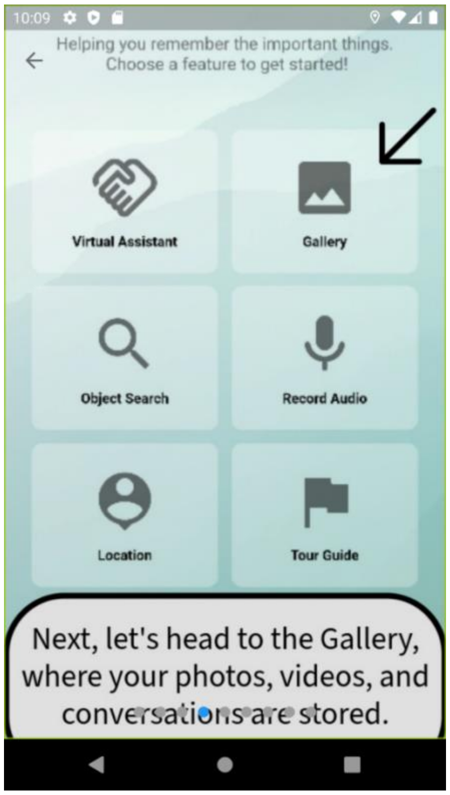

10:09
Helping you remember the important things.
Choose a feature to get started!
Virtual Assistant
Gallery
Object Search
Record Audio
Location
Tour Guide
Next, let's head to the Gallery, where your photos, videos, and conversations are stored.

Figure 13: Tour Guide screen

Internal UI functionality: When the actor selects the right arrow button, the system shall display information for the next feature of the application. When the actor selects the skip button, the actor shall be returned to the Main Menu screen.

External UI functionality: N/A

6.1.7 Gallery Screen

The following screen shall display the Gallery Menu. The actor may choose to view or play media

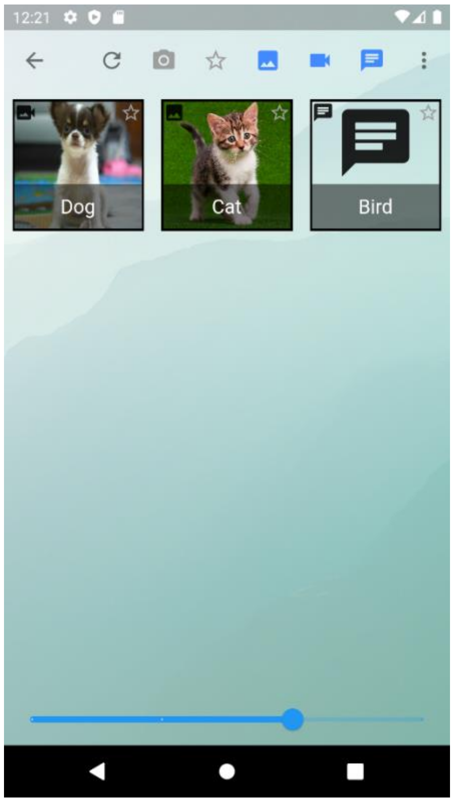
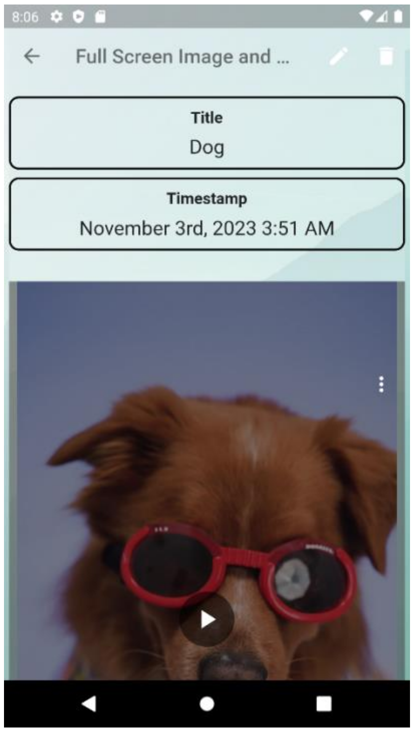


Figure 14: Gallery Screen



Title
Dog

Timestamp
November 3rd, 2023 3:51 AM

Figure 15: Gallery Detail Screen

Internal UI functionality: When the actor selects a gallery item, the system shall display options to view or play the media. When the actor selects to play the media, the system shall display the My Gallery Playback Screen and play the media.

External UI functionality: N/A

7 Development, Test, Deployment, & Operations

This section of the TDD reviews the strategies implemented in development, testing, deployment, and operations of the CogniOpen system. These outlined processes provide a detailed understanding of the elements to assist in the development of the system from a concept to an application.

7.1 Development

The Scrum methodology will be utilized for the development process of the project. It will include clear communication and collaboration with the stakeholders, iterative development, as well as adaptability with potentially changing requirements.

The application will be developed by a cross-functional team which consists of members with skill sets in varying areas of expertise such as developers, testers, designers, business analysts and technical writers crucial to the success of a high-quality product.

7.2 Testing

Testing will be conducted in the form of automated unit, integration, and regression testing. The act-like-a-customer (ALAC) method will be used in the test cases to ensure that the application aligns with the user abilities and requirements. This project will not test external service providers as they are assumed to be tested by their own organizations at their defined standards.

Clear communication between the development teams and the testing team will assist in delivering a well-developed product without defects on schedule and on budget. As development of new features is completed, coordination will be required between the Software Engineer who developed the new feature and the Test Engineer who is assigned to test the new feature. Testing will be automated to allow for early detection of defects in code changes and fast action to mitigate any bugs that may potentially be found by the end-user.

The following testing lifecycle and associated roles will be utilized:

1. DTTS Software Engineer completes work on a feature and commits to developer branch

2. DTTS Test Engineer creates the appropriate unit tests for the completed feature.

3. DTTS Test Engineer creates a pull request to merge the feature branch into development, this triggers step 4.

4. GitHub CI runs all tests and a Team B Test Engineer validates all are passing. If test(s) are passing, move to step 5, otherwise a bug report is created and development work continues.

5. Unit testing passes, DTTS Software Engineer merges the branch with development. This triggers GitHub CI automation testing again.
6. DTTS Software Engineer identifies major features which are complete and merges with the Integration branch.
7. DTTS Test Engineer creates integration tests, this triggers step 8.
8. GitHub CI runs all tests and a Team B Test Engineer validates all are passing. If test(s) are passing, move to step 6, otherwise a bug report is created and development work continues.
9. DTTS Software Engineer identifies stable functionality and merges with main branch.
10. System Testing is started as a manual process with both DTTS and Team B test teams. If test(s) are passing, move to step 11, otherwise a bug report is created and development work continues.
11. If all development is complete, move to step 12. Otherwise continue to develop new features.
12. User Acceptance Testing is started as a manual process with both DTTS and Team B test teams as well as with the client. If test(s) are passing, move to step 13, otherwise a bug report is created and development work continues.

13. Application is delivered to the client. Development and testing is complete. Deliver final test plan and test report to client.

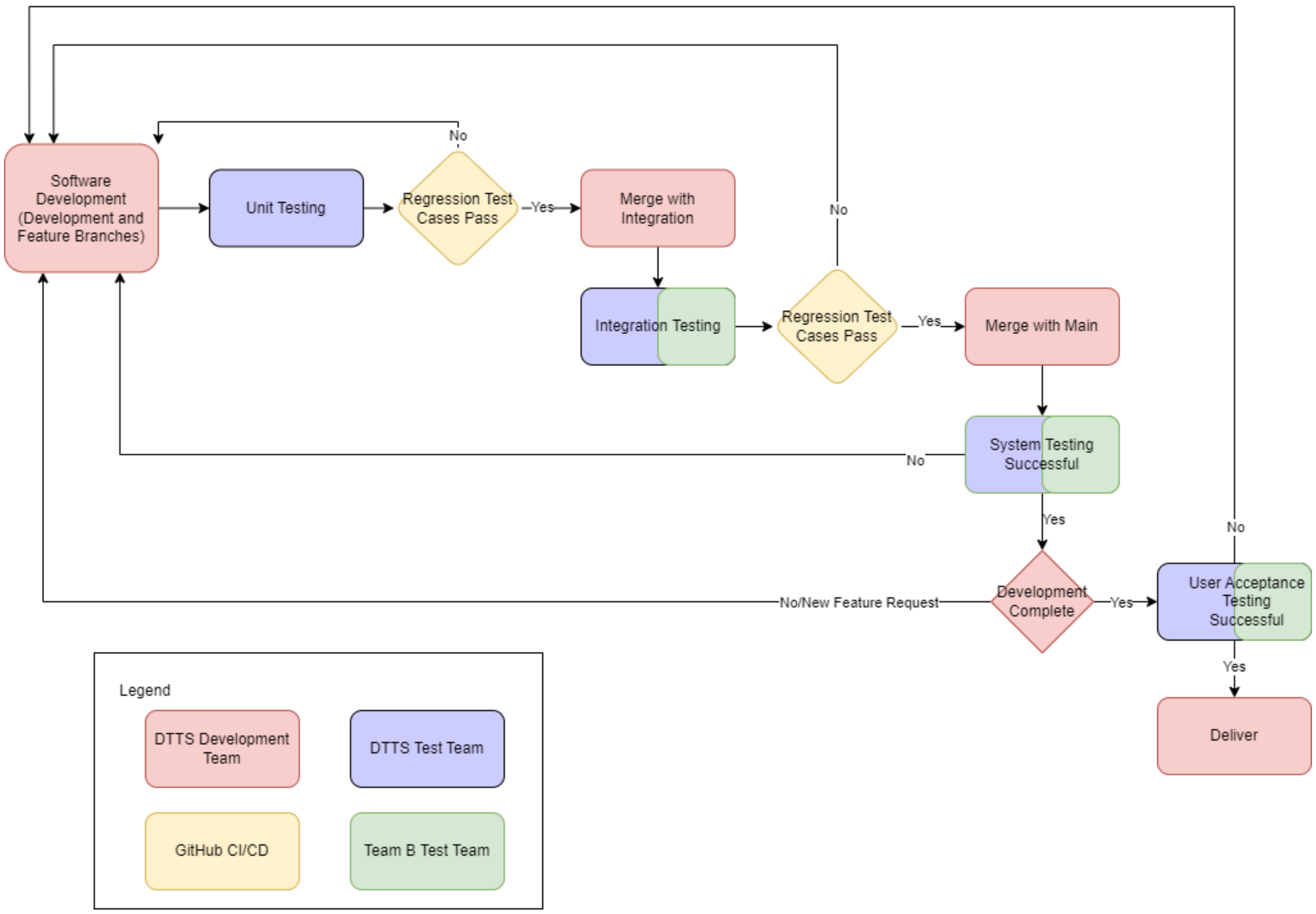


Figure 16: Test Plan Flowchart

The Test Plan document, which is comprised of the testing methodology and all the detailed test cases for the application, will be managed and updated according to any emergent changes made within the code.

7.3 Deployment

Continuous Integration and Continuous Deployment (CI/CD) will be implemented to help streamline the deployment of the application as well as newer versions of the code. Automation will help to shorten the time between development, and deployment will maintain code quality by highlighting deficiencies in the product sooner.

7.4 Operations

After deployment of the application, it will be monitored for performance and needed maintenance and support. Key Performance Indicators (KPIs) will be observed to measure the performance of the application. The operations team will be notified of any major issues from alerts and notifications by setting thresholds on certain performance metrics.

Documentation such as test plan, technical design document, software requirement specification, deployment and operations guide, and programmer’s guide will be kept to assist in the maintenance and understanding of the application.

The cross-functional team will undergo knowledge transfers meetings to ensure that all members of the team have a good understanding of all aspects of the application.

8 Issues & Risks

In addition to the technical challenges and considerations, it's essential to address non-technical aspects that can impact the technical design and project implementation. The following section outlines potential issues and risks that needs attention:

8.1 User Acceptance

Issue: One of the central concerns is ensuring that the CogniOpen application is highly intuitive and aligns with the unique needs of memory-impaired individuals and their caregivers.

Risk: If the user interface and functionality do not cater to the cognitive challenges of the target audience, it could result in low user adoption rates and user dissatisfaction.

Mitigation: DTTS will conduct extensive user testing involving the target audience throughout the development process. Gathering continuous feedback and involving healthcare professionals and caregivers in the design phase will be integral to making iterative improvements and ensuring high user acceptance.

8.2 Privacy & Data Security

Issue: The CogniOpen application handles sensitive information, including personal conversations and location data.

Risk: Inadequate security measures could lead to privacy breaches and data leaks. Such incidents not only harm user trust but may also result in legal consequences.

Mitigation: To mitigate this risk, DTTS will implement robust encryption mechanisms, stringent access controls, and strict compliance with data protection regulations. Regular security audits and monitoring will be incorporated into the development and operational processes to identify and address vulnerabilities promptly.

Transport Layer Security (TLS) will be used to ensure security for data transmitted between the application and external servers. Locally stored data on a user's device will be encrypted using the Advanced Encryption Standard (AES) with a 256 bits key length. These keys will be securely store in Android Keystore and the iOS Keychain. These practices demonstrate the commitment of CogniOpen to earn the trust of its users and secure their data.

8.3 Regulatory Compliance

Issue: The CogniOpen application may fall within the purview of regulatory frameworks related to healthcare and data privacy.

Risk: Non-compliance with these regulations could result in legal penalties and project delays.

Mitigation: DTTS will maintain regular consultations with legal experts to ensure ongoing compliance with all relevant regulatory requirements. Adherence to these regulations will be a central focus throughout both the development and deployment phases.

8.4 Caregiver Support

Issue: The CogniOpen application serves both memory-impaired individuals and their caregivers, who play a pivotal support role.

Risk: Overlooking caregiver needs may impede the application's effectiveness in delivering comprehensive support.

Mitigation: DTTS is committed to actively engaging with caregivers, understanding their specific requirements, and integrating features that assist caregivers in effectively using the application. Addressing caregiver needs enhances the overall user experience and the application's effectiveness.

9 Requirements Matrix

This Traceability Matrix aims to build a cross-reference table that defines the relationship between various system and data structure components and the requirements described in the Software Requirements Specification (SRS). The following table is provided as an illustration of the system components that conform to the SRS's Use-case requirements. Each requirement will be accompanied by the applicable part tasked with achieving that requirement, as stated in this document. This matrix demonstrates the quality of the product achieving the project’s needs.

SRS ID	Requirements Description	Element Number	Element Name
SRS-1	Initialize the Application	5.9	AccessCenter
SRS-3	Toggle Passive Audio Recording	5.13	SettingsMenu
SRS-5	Edit a Conversation.	5.20	GalleryMenu
SRS-6	Set up Significant Objects.	5.20	GalleryMenu
SRS-7	Search in Location History.	5.11	LocationMenu
SRS-8	Record a video.	5.16	VideoMenu
SRS-9	Toggle Passive Video Recording Mode	5.16	VideoMenu
SRS-11	Ask Assistant to Locate Item	5.16	VideoMenu
SRS-12	Toggle Location Services	5.13	SettingsMenu
SRS-13	Add Media to Gallery	5.20	GalleryMenu
SRS-14	View Media in Gallery	5.20	GalleryMenu
SRS-15	Application Tour Guide	5.9	AccessCenter
SRS-16	View Timeline	5.18	LocationMenu

Table 9: Requirements Traceability Matrix

SRS IDs 2 (Record a conversation), 4 (search conversation), and 10 (Ask assistant a question) were use cases originally developed with the whole application in mind, but based on the work breakdown the new work breakdown established by both teams, they are out of scope for DTTS development.

Appendix A: Considerations

In this appendix section, we included design ideas that we initially considered for implementation, but during development, we scraped or postponed for some future development effort.

These ideas include:

- integrating the user's calendar to search for upcoming events
- creating reminders from a user's conversation
- using Rekognition to detect custom labels (the user selected significant objects) from the recognized objects from the video recording
- send notifications to the user for upcoming events, parsed objects, and generally remind the user of what's going on with the application (e.g., video still recording; video has not resumed recording since being paused; custom model is available to turn on; custom model is still running).

System Overview Diagram

The previous system overview highlights were the calendars, lists, and reminders would interact inside the system. Data would be retrieved from the user through the application and stored in the local SQLite database instance. The application would process requests for adding and viewing calendar events, along with adding, viewing, and changing the lists and reminder items.

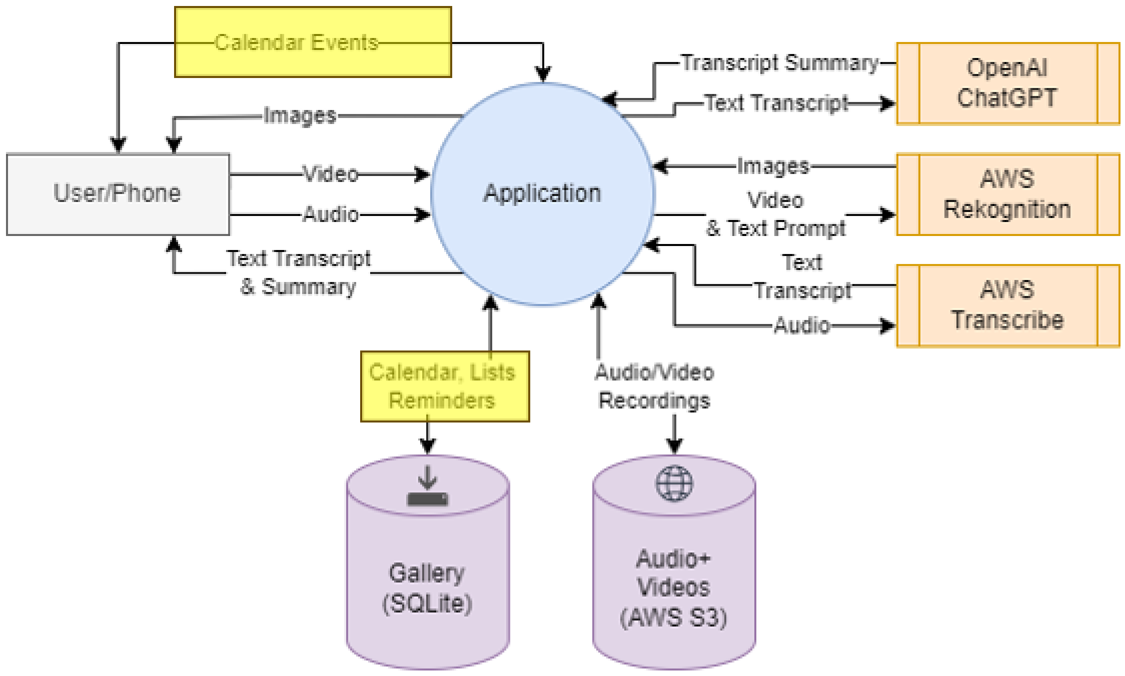


Figure A1: System Overview including Calendar, Lists, and Reminders

Entity Relationship Diagram

The previous design had an instance of the entity relationship for the event and reminder objects. These objects were removed from the ERD for the current TDD submission, but they are recorded here for posterity. A user could have one or many events or reminders, but any event or reminder would only exist for one user. These were the fields we expected to use through development, but are obviously subject to change during the implementation.

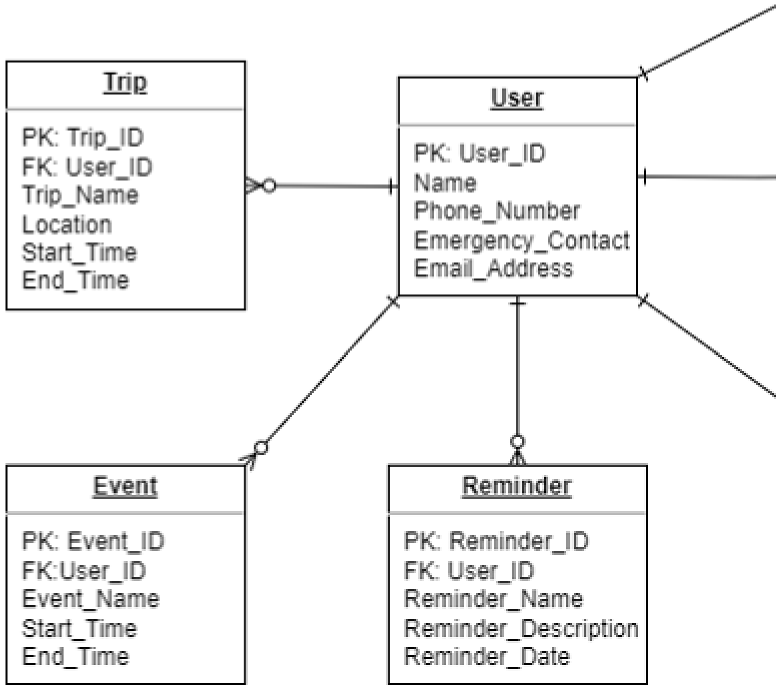


Figure A2: Event and Reminder ERD illustration

Entity Descriptions

Reminder Entity: This entity is crucial for users to set reminders for tasks or events. Each reminder is identified by a unique "Reminder_ID" and linked to the user who created it using "User_ID." The reminder's text description is recorded in "Reminder_Description," and the "Reminder_Date" specifies when the reminder is scheduled.

- Attributes: Reminder_ID (Primary Key), User_ID (Foreign Key), Reminder_Name, Reminder_Description, Reminder_Date
- Relationships:
    - Reminder-User (Many-to-One): Multiple reminders can belong to a single user. Each reminder is associated with one user. This means that a user can create and manage multiple reminders, and each reminder is linked back to the user who created it.

Event Entity: Represents events or activities planned or attended by users. Events can include meetings, appointments, or social gatherings. Each event is uniquely identified by an Event_ID and includes attributes such as the event's name or title, the start date and time, and the end date and time.

- Attributes: Event_ID (Primary Key), User_ID (Foreign Key), Event_Name, Start_Time, End_Time
- Relationships:
  - Event-User (Many-to-One): Many events can be associated with one user, indicating that users can plan and attend multiple events.

Data Flow Diagram

The data flow diagram for the calendar event data circulated around pulling and pushing information to the user’s native calendar application.

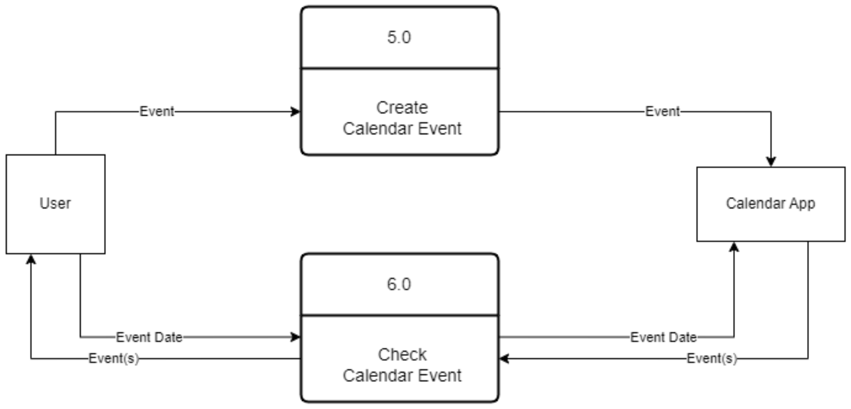


Figure A3: Calendar Data Flow Diagram

Class Element Design

We discovered that, compared to our initial design, what was available in flutter varied significantly from what we designed. Our paradigm was that of working with Java with declared getter and setter methods – these were not of important in declaration when working in Flutter. Nevertheless, here were our initial designs indicating the attributes we found pertinent based on our data design analysis. Likewise, we envisioned a settings screen for the user to track their personalized settings, because of the native device settings; if the user declined to allow the application to use the device’s location/audio/video, they can use the device system settings to re-enable the permissions.

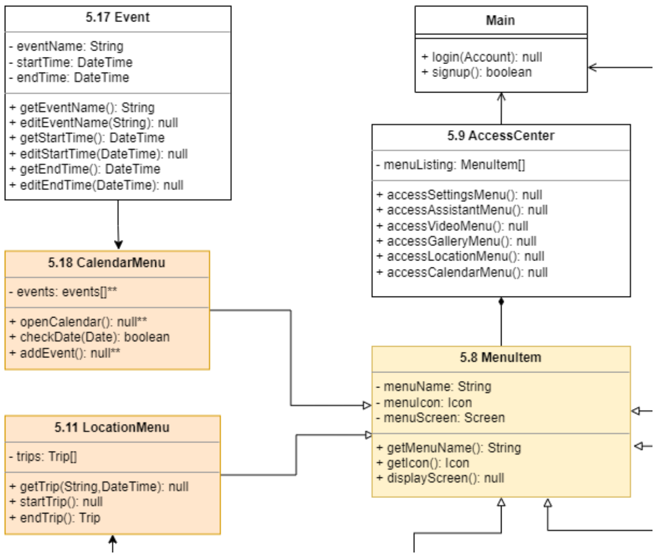


Figure A4: Calendar Event Class Diagram

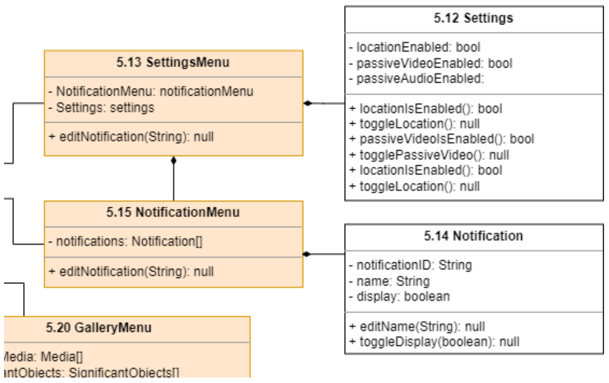

5.12 Settings
- locationEnabled: bool
- passiveVideoEnabled: bool
- passiveAudioEnabled:
+ locationIsEnabled(): bool
+ toggleLocation(): null
+ passiveVideoIsEnabled(): bool
+ togglePassiveVideo(): null
+ locationIsEnabled(): bool
+ toggleLocation(): null
5.13 SettingsMenu
- NotificationMenu: notificationMenu
- Settings: settings
+ editNotification(String): null
5.15 NotificationMenu
- notifications: Notification[]
+ editNotification(String): null
5.14 Notification
- notificationID: String
- name: String
- display: boolean
+ editName(String): null
+ toggleDisplay(boolean): null
5.20 GalleryMenu
ledia: Media[]
intObjects: SignificantObjects[]

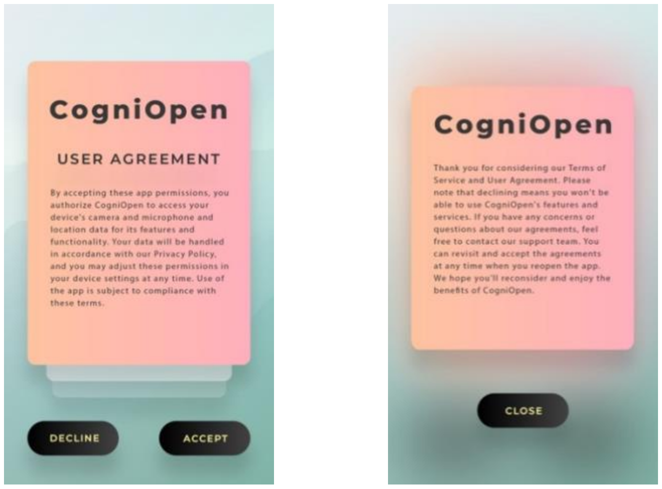
Figure A5: Notification Class Diagram

Human Interface Design

We initially expected to have a screen where the user would be able to accept a license agreement and allow all permissions to the application. We found during the implementation that we could pass through the device permissions handler to the user, so this screen was deprecated. Should the situation arise that we need the user to accept a “Terms of Service”, something to this design would need to come into effect.

App permissions Screen

The following screen shall display the CogniOpen user agreement and privacy policy needed to be accepted to access all features of the application. This screen shall give the actor options to agree or decline these agreements.


CogniOpen

USER AGREEMENT

By accepting these app permissions, you authorize CogniOpen to access your device's camera and microphone and location data for its features and functionality. Your data will be handled in accordance with our Privacy Policy, and you may adjust these permissions in your device settings at any time. Use of the app is subject to compliance with these terms.

DECLINE ACCEPT

CogniOpen

Thank you for considering our Terms of Service and User Agreement. Please note that declining means you won't be able to use CogniOpen's features and services. If you have any concerns or questions about our agreements, feel free to contact our support team. You can revisit and accept the agreements at any time when you reopen the app. We hope you'll reconsider and enjoy the benefits of CogniOpen.

CLOSE

Figure A6: Base user agreement, and User agreement declined

Internal UI functionality: When the actor selects the Accept button, the main menu screen shall open. If the actor selects the Decline button, the app shall inform them that accepting these policies is necessary before continuing.

External UI functionality: If the actor declines the user agreement and privacy policy, the app shall close.

Settings Menu Screen

The following screen shall allow the actor to change the settings and permissions within CogniOpen.

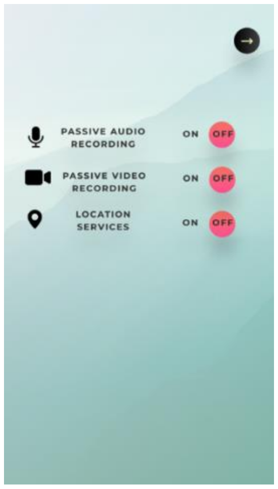


Figure A7: Settings Screen

Internal UI functionality: When the actor toggles the Passive Audio Recording option, the system shall enable or disable the feature. When the actor toggles the Video Audio Recording option, the system shall enable or disable the feature. When the actor toggles Video Audio Recording option, the system shall enable or disable the feature.

When the actor toggles the Location Services option, the system shall enable or disable the feature.

External UI functionality: The system shall enable or disable features corresponding to the actor’s actions.

Record Bar

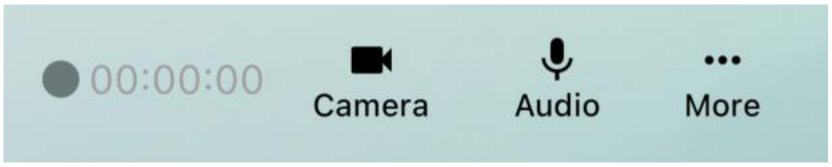


Figure A8: Record Bar

With the implementation of passive audio and video recording, we considered adding a record bar to all CogniOpen screens. This bar would allow the user to visually see whether an audio or video recording was currently in progress, would have a pause and stop button, as well as a timer for the current recording.

Application learning

A portion of this application that could be enhanced is the culmination of user’s significant objects, AWS Rekognition Custom Labels, and object detection. In Team A’s implementation, we developed a process to convert a user’s selected object into a significant object. We also used this user-select object response to train a custom label model for future detection of objects. We did not implement using this model for detecting custom objects (the user’s significant objects) in the list of identified object responses.

To implement such a feature, we envision that the user would record a video, search for an object such as their “glasses”, and then select the detect “glasses” object from the list of object responses. This object response would be saved as a significant object. Additionally, this object response would have image information and bounding box annotation to properly train a model for use in AWS Rekognition custom label machine learning training. We implemented this far. The additional images that were not selected as the identified object in the object response could be utilized as the “testing” set of data in the machine learning algorithm.

Once a model has been trained, it is available to be started. In our exploratory development sessions, we discovered that it would take about 10 minutes to train a model and then another 10 minutes to start the model. Once a model has started, it would take about another 1 minute to stop the model, when the user is done using it (ideally on application close, but we did not get that far). Of importance to note here is that a running model incurs an AWS charge for “inference usage” whether requests are being made to identify custom objects with the model or not.

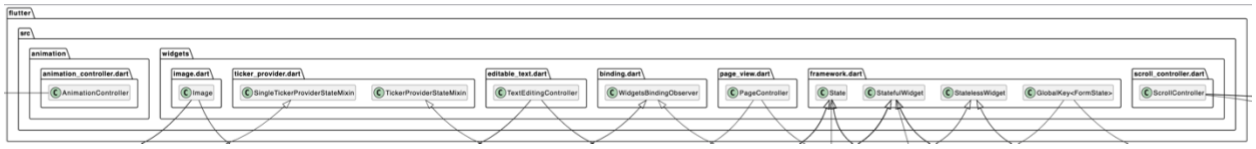
Once a model is started, the object responses from the most recent video would be requested to detect any custom objects in those images. It is important to note that while AWS Rekognition allows for object detection from video media, AWS Rekognition only detects custom labels from photo image media. So, based on the previous example, all “glasses” objects from the most recent video would be compiled and sent for Rekognition to detect if any apply to the custom label “my-glasses” trained from the user’s “my glasses” significant object.

While our team implemented creating a significant object (and custom label model) from the user’s selection of a Rekognition object response, we did not implement the same functionality for creating a model from all significant objects. In this regard, the user’s current ability to create significant objects from the settings screen is rather lackluster. Tying this capability into the Rekognition custom label training would allow the user (especially a caregiver) to “seed” the application with many of the user’s important objects from the start and not just when Rekognition detects them the first time.

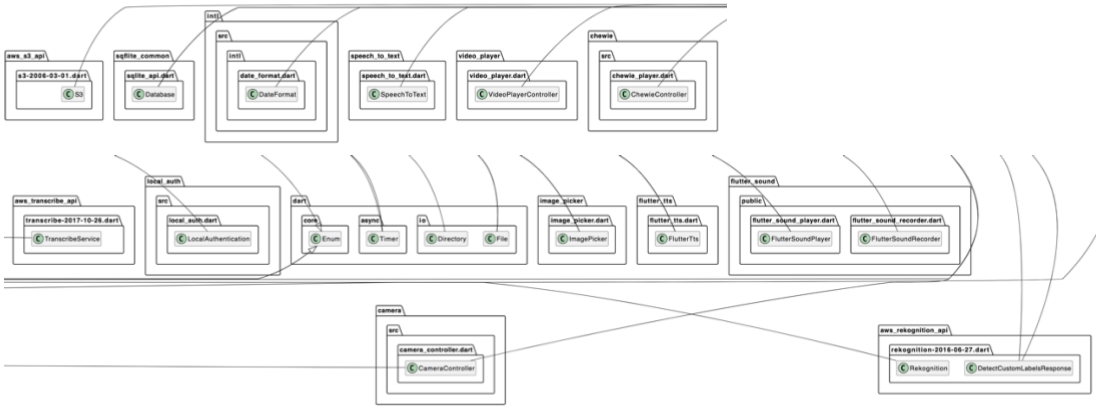
Appendix B: Class Diagram

As mentioned in Section 5, for a comprehensive view of the CogniOpen Software Application class diagram, please refer to the full-size image at class-diagram-full.svg. For ease of reference, the same class diagram divided into several segmented images providing a more detailed look at each section of the diagram is included below.

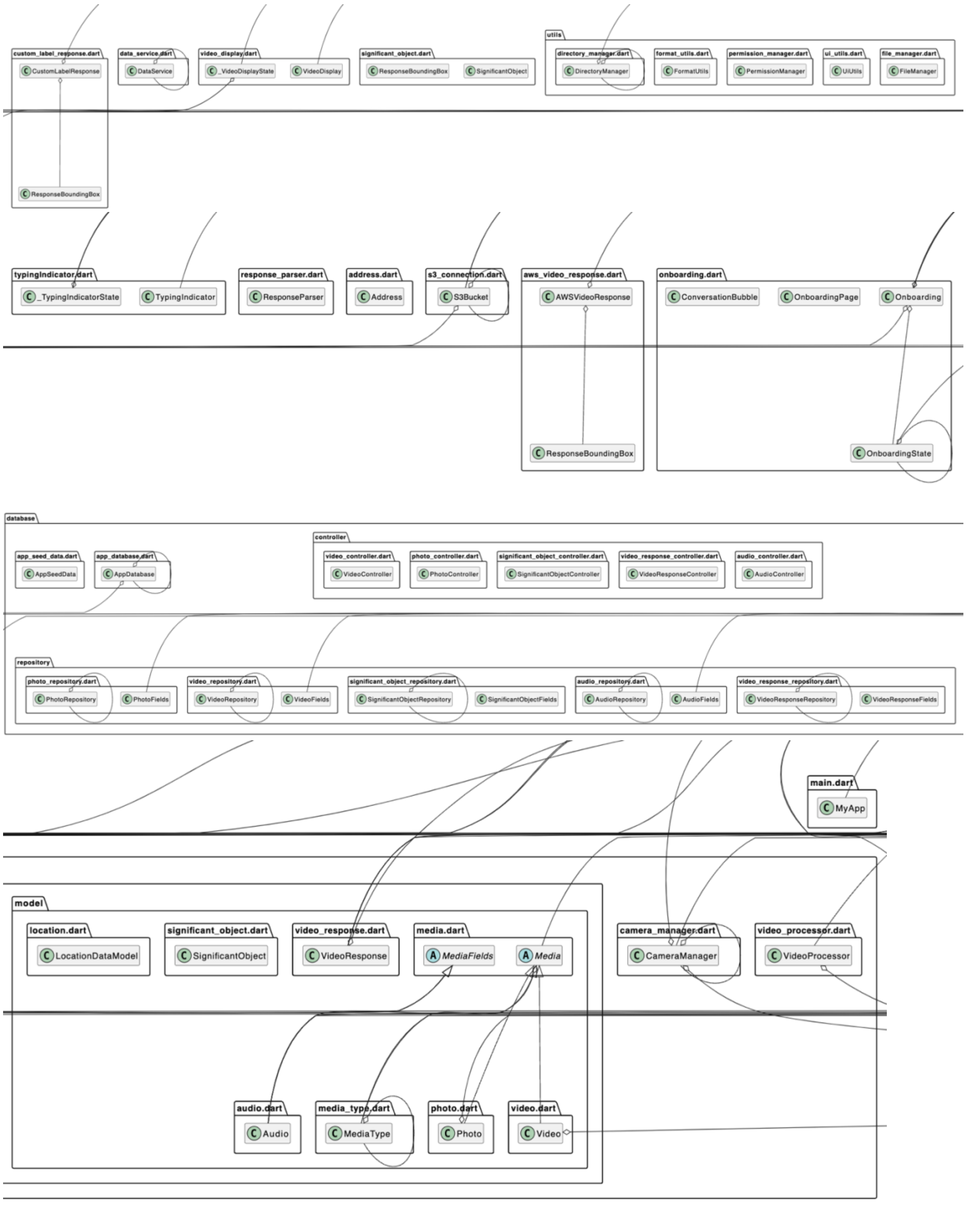
flutter:



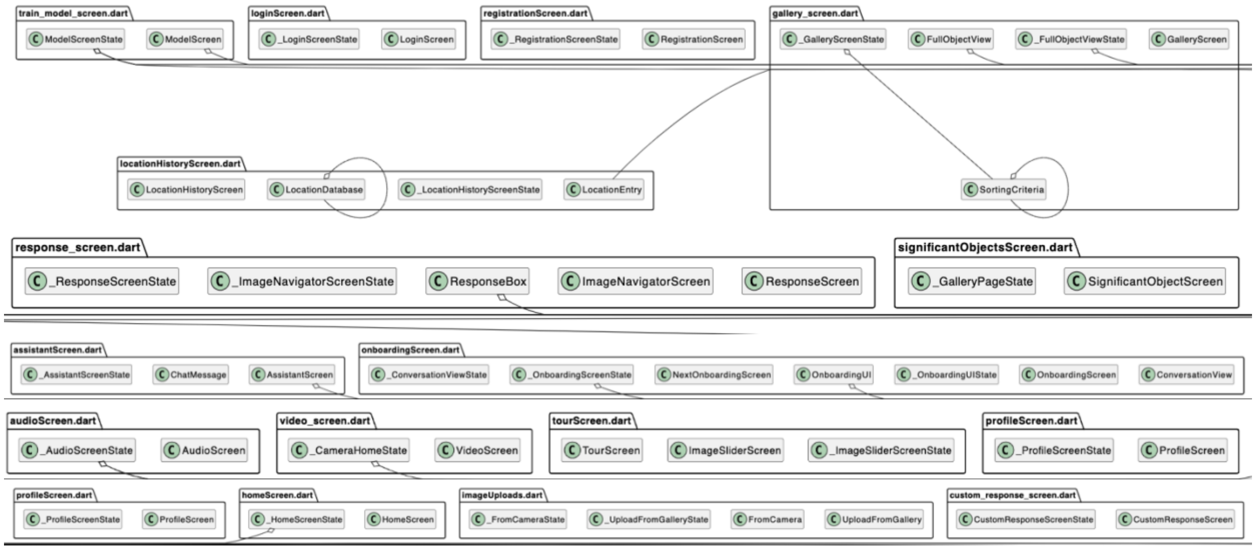
external:



src:


custom_label_response.dart
CustomLabelResponse
ResponseBoundingBox
data_service.dart
DataService
video_display.dart
_VideoDisplayState
VideoDisplay
significant_object.dart
ResponseBoundingBox
SignificantObject
utils
directory_manager.dart
DirectoryManager
format_utils.dart
FormatUtils
permission_manager.dart
PermissionManager
ui_utils.dart
UiUtils
file_manager.dart
FileManager
typingIndicator.dart
_TypingIndicatorState
TypingIndicator
response_parser.dart
ResponseParser
address.dart
Address
s3_connection.dart
S3Bucket
aws_video_response.dart
AWSVideoResponse
ResponseBoundingBox
onboarding.dart
ConversationBubble
OnboardingPage
Onboarding
OnboardingState
database
app_seed_data.dart
AppSeedData
app_database.dart
AppDatabase
controller
video_controller.dart
VideoController
photo_controller.dart
PhotoController
significant_object_controller.dart
SignificantObjectController
video_response_controller.dart
VideoResponseController
audio_controller.dart
AudioController
repository
photo_repository.dart
PhotoRepository
PhotoFields
video_repository.dart
VideoRepository
VideoFields
significant_object_repository.dart
SignificantObjectRepository
SignificantObjectFields
audio_repository.dart
AudioRepository
AudioFields
video_response_repository.dart
VideoResponseRepository
VideoResponseFields
main.dart
MyApp
model
location.dart
LocationDataModel
significant_object.dart
SignificantObject
video_response.dart
VideoResponse
media.dart
MediaFields
Media
camera_manager.dart
CameraManager
video_processor.dart
VideoProcessor
audio.dart
Audio
media_type.dart
MediaType
photo.dart
Photo
video.dart
Video

ui:


train_model_screen.dart
ModelScreenState
ModelScreen
loginScreen.dart
_LoginScreenState
LoginScreen
registrationScreen.dart
_RegistrationScreenState
RegistrationScreen
gallery_screen.dart
_GalleryScreenState
FullObjectView
_FullObjectViewState
GalleryScreen
SortingCriteria
locationHistoryScreen.dart
LocationHistoryScreen
LocationDatabase
_LocationHistoryScreenState
LocationEntry
response_screen.dart
_ResponseScreenState
_ImageNavigatorScreenState
ResponseBox
ImageNavigatorScreen
ResponseScreen
significantObjectsScreen.dart
_GalleryPageState
SignificantObjectScreen
assistantScreen.dart
_AssistantScreenState
ChatMessage
AssistantScreen
onboardingScreen.dart
_ConversationViewState
_OnboardingScreenState
NextOnboardingScreen
OnboardingUI
_OnboardingUIState
OnboardingScreen
ConversationView
audioScreen.dart
_AudioScreenState
AudioScreen
video_screen.dart
_CameraHomeState
VideoScreen
tourScreen.dart
TourScreen
ImageSliderScreen
_ImageSliderScreenState
profileScreen.dart
_ProfileScreenState
ProfileScreen
profileScreen.dart
_ProfileScreenState
ProfileScreen
homeScreen.dart
_HomeScreenState
HomeScreen
imageUploads.dart
_FromCameraState
_UploadFromGalleryState
FromCamera
UploadFromGallery
custom_response_screen.dart
CustomResponseScreenState
CustomResponseScreen