



CogniOpen Software Application

Programmer Guide

Project Name: CogniOpen Software Application
Document: Programmer Guide (PG)
Version: 2.0
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

Leads Sign-off Sheet

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

Revision History

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

Table of Contents

<i>Leads Sign-off Sheet</i>	2
<i>Revision History</i>	3
<i>Table of Contents</i>	4
<i>List of Figures & Tables</i>	7
1 Introduction	8
1.1 Purpose	8
1.2 Intended Audience	8
1.3 Technical Project Stakeholders	8
1.4 Document Conventions & Organization	9
1.4.1 Document Conventions	9
1.4.1.1 Terminology	9
1.4.1.2 Formatting	9
1.4.1.3 Lists	9
1.4.1.4 Hyperlinks	9
1.4.2 Document Organization	9
1.4.2.1 Table of Contents	9
1.4.2.2 Section Structure	9
1.4.2.3 Appendices.....	10
1.4.2.4 Revision History	10
1.5 Project Document Suite	10
1.6 References	10
1.7 Terms, Abbreviations, & Acronyms	11
2 Operating Systems & Development Tool Versions	13
3 Development Environment	14
3.1 GitHub	14
3.2 Android Studio.....	14
3.3 Visual Studio Code.....	14
3.4 Communication	15
3.5 AWS Transcriber	15
3.6 AWS Rekognition	15
3.7 ChatGPT	16
3.8 Photoshop	16

4	Frameworks & Languages	17
4.1	Dart.....	17
4.2	Flutter	17
5	Development Process.....	18
5.1	Project Tracking	18
5.2	Branching Strategy	18
5.3	Code Review	21
5.4	System Overview	21
5.5	Architectural Design.....	22
5.6	Video Analysis	23
6	Test Process.....	26
6.1	Test Strategy	26
6.2	Unit Testing.....	27
6.3	Integration Testing.....	27
6.4	Regression Testing.....	27
7	Code Structure.....	29
7.1	Gallery Module	29
7.2	Video Module	30
8	Project File Structure	31
8.1	Project Documents.....	31
8.1.1	Root Level Folder Structure	31
8.1.2	Milestone Folder Structure	31
8.1.3	Milestone Documents	32
8.2	Source Code File Structure	33
8.2.1	Business Logic Layer	33
8.2.2	UI Layer.....	33
8.2.3	Database Layer	34
9	User Interface.....	35
9.1	Home Screen	35
9.1.1	Internal functionality.....	35
9.1.2	External functionality	35
9.2	Gallery Screen	36
9.2.1	Internal functionality.....	37
9.2.2	External functionality	37

9.3	Video Recording Screens	38
9.3.1	Internal functionality.....	38
9.3.2	External functionality	38
9.4	Significant Objects Screen.....	39
9.4.1	Internal Functionality	39
9.4.2	External Functionality	39
10	Data & Backend	41
10.1	Simple Storage Service	41
10.2	Rekognition	41
10.3	Sqflite.....	41
11	Publishing	43
<i>Appendix A.....</i>		<i>44</i>

List of Figures & Tables

Table 1: Technical Project Stakeholders	9
Table 2: Project Document Suite.....	10
Table 3: Terms, abbreviations, and acronyms	12
Table 4: Operating Systems & Development Tool Versions	13
Figure 1: Branching Strategy Option A	19
Figure 2: Branching Strategy Option B	20
Figure 3: System Overview	22
Figure 4: Architectural Design	23
Figure 5: Kinesis Rekognition Streaming vs Static Rekognition Detection	24
Figure 6: Testing Strategy	26
Figure 7: Gallery Module Diagram	29
Figure 8: Video Module Diagram	30
Figure 9: Project Document - Root Level Folder Structure	31
Figure 10: Project Document – Milestone Folder Structure	32
Figure 11: Project Document – Milestone Documents	33
Figure 12: Home Screen	35
Figure 13: Gallery Screen	36
Figure 14: Video Recording preview screen.....	38
Figure 15: Significant Objects Screen	39

1 Introduction

1.1 Purpose

CogniOpen Programmer Guide fosters cooperation between developers. It shares the rationale behind design decisions in the software architecture. On the server side, CogniOpen makes extensive use of third-party software utilities. Those products are built and maintained by experts in artificial intelligence and natural language processing (NLP). CogniOpen must conform to external application programming interfaces (APIs). This document explores the intricacies of CogniOpen's procedural invocation to remote servers, aka, the cloud.

On the client side, CogniOpen adheres to a strict mobile device graphical user interface (GUI). The programmer guide explains how the controls were placed and their intent.

1.2 Intended Audience

CogniOpen Programmer Guide informs and educates prospective development teams on their task. The knowledge-level of the reader is assumed to be relatively high. CogniOpen is a complex software product with several loosely bound components. This document explains the nature of the connections. Furthermore, it promulgates the purpose of the myriad design decisions. Curious parties and/or executives may peruse the document when deciding how to build related software or extend CogniOpen.

1.3 Technical Project Stakeholders

The successful development and deployment of the CogniOpen Software Application involves collaboration among various technical project stakeholders. The roles and responsibilities of these stakeholders are defined as follows:

Role	Responsibility
Client	Dr. Mir Assadullah, representing the client's interests and requirements for the CogniOpen Software Application.
Team Lead / Project Manager	Vincent Galeano, who oversees project management and team coordination.
Lead Technical Writer	Kavon Johnson, responsible for technical documentation, including this Programmer Guide.
Lead Business Analyst	Zach Bowman, involved in requirements analysis and business processes.
Architect / Lead Software Developer	David Bright, responsible for software architecture and development leadership.
Lead UI/UX Designer	Juan Torres-Chardon, focuses on user interface and user experience design.

Role	Responsibility
Lead Test Engineer	Laura Hamann, responsible for software testing and quality assurance.

Table 1: Technical Project Stakeholders

1.4 Document Conventions & Organization

Throughout this guide, we adhere to consistent terminology and naming conventions to enhance clarity and comprehension. Key terms and their definitions are provided in Section 1.7, "Terms, Abbreviations, & Acronyms."

1.4.1 Document Conventions

1.4.1.1 Terminology

Throughout this guide, we adhere to consistent terminology and naming conventions to enhance clarity and comprehension. Key terms and their definitions are provided in Section 1.7, "Terms, Abbreviations, & Acronyms."

1.4.1.2 Formatting

Text formatting is used consistently to convey information:

Bold Text: Used for emphasizing important points, section headings, and UI elements.

Italic Text: Employed for highlighting variable names, file paths, and placeholders.

Monospace Font: Indicates code snippets, file names, and inline code references.

1.4.1.3 Lists

Lists are presented in two formats:

Numbered Lists: Used for sequential steps or procedures.

Bulleted Lists: Used for items without a specific order.

1.4.1.4 Hyperlinks

Hyperlinks to external resources, documents, or web pages are provided for additional information and context. They are displayed in blue and are clickable when viewed digitally.

1.4.2 Document Organization

1.4.2.1 Table of Contents

The document begins with a Table of Contents listing all major sections and subsections. The Table of Contents allows quick navigation to specific topics.

1.4.2.2 Section Structure

Each section of this Programmer Guide follows a consistent structure:

Section Heading: Clearly identifies the section's topic.

Subsections: Sections may contain subsections to further organize and detail information.

Content: Provides comprehensive information, explanations, examples, and best practices related to the topic.

1.4.2.3 Appendices

Appendices (if included) are placed at the end of the document and provide supplementary information, references, or resources relevant to the content discussed in the main document.

1.4.2.4 Revision History

To track changes and updates, a revision history is included at the end of this document. It lists the version history, dates of revisions, and descriptions of changes made.

1.5 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 2: Project Document Suite

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

1.6 References

Amazon. (2014). *Amazon Simple Storage Service: Getting started guide*.
<http://s3.amazonaws.com/awsdocs/S3/latest/s3-gsg.pdf>

Amazon. (2021, August 5). *Overview of AWS Web Services*.
<https://d1.awsstatic.com/whitepapers/aws-overview.pdf>

Davis, N. (2022, April 22). *What is AWS S3?* Digital Cloud Training.
<https://digitalcloud.training/what-is-aws-s3/>

Dragos. (2023, February 21). *Everything you need to know about developer documentation*. <https://www.archbee.com/blog/developer-documentation-guide>

Google. (n.d.). *Dart overview*. <https://dart.dev/overview>

Google. (n.d.). *Flutter architectural overview*.
<https://docs.flutter.dev/resources/architectural-overview>

Soos, I. (2023). *aws_rekognition_api 2.0.0*.
https://pub.dev/packages/aws_rekognition_api

Soos, I. (2021). *aws_s3_api 2.0.0*. https://pub.dev/packages/aws_s3_api

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

Wilson, M. (2023, March 15). ChatGPT explained: everything you need to know about the AI chatbot. TechRadar. <https://www.techradar.com/news/chatgpt-explained>

1.7 Terms, Abbreviations, & Acronyms

Term	Definition
AI	Artificial Intelligence
AJAX	Asynchronous JavaScript and XML
API	Application Programmer Interface
AWS	Amazon Web Services
GPT	Generative Pre-trained Transformer
GUI	Graphical User Interface
IDE	Integrated Development Environment
IT	Information Technology
JSON	JavaScript Object Notation
MS	Microsoft
NLP	Natural Language Processing
OS	Operating System
PDF	Portable Document Format
S3	Simple Storage Service
UI	User Interface
URL	Uniform Resource Locator

Term	Definition
UX	User Experience
VS	Visual Studio

Table 3: Terms, abbreviations, and acronyms

2 Operating Systems & Development Tool Versions

Product Name	Version
ChatGPT	3.5
Photoshop	v2023.Q3
Android Studio	Giraffe 2022.3.1 with Android SDK 34
Visual Studio Code	2022 v1.81.1
Microsoft Teams	1.6.00.22378 (64-bit)
Microsoft Word + Excel	2307 (Build 16626.20170)
Dart	v3.1.0
Flutter	V3.1.0
MacOS	Sonoma 14.1
Windows OS	10 v22H2

Table 4: Operating Systems & Development Tool Versions

3 Development Environment

This section will highlight the elements of the project necessary to establish the development environment. The most important, the location of the code repository that will be a reference point for future development and project teams that will maintain the CogniOpen software. Developing a mobile application, we will need a way to validate that the software is functional, meets requirements and is visually appealing. And of course, the development team will need some editor to put the pieces of the CogniOpen application together. Many tools were leveraged to build the application that relates to the backend services, front-end development, and data storage of the software system.

3.1 GitHub

GitHub is being used to host the source code for the CogniOpen application. This was the selected location for the code repository based on the preferences of the stakeholders. Because of this, there were no alternative considerations for the location of the source code other than GitHub. However, this allows both development teams for the CogniOpen application to have a centralized repository that fosters continuous development and integration. All requests to merge code into the development and production branches are reviewed by members from both teams. Also, the test team is involved in this process, which allows them to perform the various levels of testing necessary (unit tests, integration tests, regression tests) to produce software that upholds code quality standards.

3.2 Android Studio

Android Studio is a development platform that allows users to design and develop Android applications. Specifically, Android Studio Giraffe with Android SDK 34 is being utilized in the CogniOpen software. Android Studio was built on the software of JetBrains' IntelliJ IDEA, which was another option for developing this software. IntelliJ IDEA itself was another consideration for developing this mobile application. Android Studio was chosen because of the familiarity that was had with the tool when it came to most of the developers on the team that had experience developing mobile applications.

Integrating Android Studio into the development environment allows the project team to test the Dart/Flutter code that is developed with the mobile emulators that can be provisioned. This helps the development team identify UI components and functionality that need to be further expanded, refined, and redesigned. This promotes collaboration between the frontend and backend developers/designers to build a functional and usable application that meets system requirements and stakeholder expectations.

3.3 Visual Studio Code

Visual Studio Code is a code editor that was used to develop the software for the CogniOpen software. There are many code editors out there but the main option that was considered was IntelliJ IDEA 2022.3.1. Visual Studio Code version 1.83.0 was chosen because the connection to the GitHub repository was simple to set up by just adding an extension in VS Code and cloning the repository onto your local machine. VS

Code has many more useful extensions that help with formatting and syntax resolution. It has an integrated terminal that simplifies building and running the software locally.

3.4 Communication

Microsoft Teams was chosen as the preferred platform for communications amongst both development teams because the accounts already existed for each team member (via UMGC emails). This is also already linked to each team member's UMGC email so any notifications on chat discussions/conversations, calendar alerts and invitations would be easily accessible. Other possible options were Zoom or the discussion thread within the class site. But for the reasons already stated, these considerations were not selected. MS Teams allows the development team to communicate, collaborate and set up meetings very easily with minimal risks. This also allows the development team to communicate with the various stakeholders to define and establish system requirements.

3.5 AWS Transcriber

Amazon Transcribe is an automatic speech recognition service that uses machine learning models to convert audio to text. Amazon Transcribe can be used as an independent transcription service or can add speech-to-text capabilities to an application. This was selected over Google Cloud Speech-to-Text because it integrates better with the Amazon S3 buckets that will store the media files that need to be transcribed. The system will take audio and video files and produce transcriptions that capture the dialogues and speakers in the recorded conversations.

3.6 AWS Rekognition

The system requirements state that the application be able to detect objects in a photo or video file. Therefore, we needed an object detection service. Rather than developing our own, we decided to pursue commercially available object detection services. In our research, we discovered two viable options: (1) Google Vision and (2) AWS Rekognition. We decided to pursue AWS Rekognition.

AWS Rekognition offers a larger free tier of 5,000 images and 1,000 mins of video every month whereas Google Vision only allows 1,000 images for free. This larger allowance in AWS reduces the risk of incurring a charge in development. AWS Rekognition also has a more robust library of objects being detected, with a greater accuracy of detecting objects, and more robust compared to Google Vision.

By using AWS Rekognition, we are leveraging our existing connection to AWS. This reduces the complexity of managing an additional account credential. Compared to Google Vision, AWS Rekognition does not allow for batch processing of images, so there is a slight performance drop in our AWS approach compared to using Google Vision.

3.7 ChatGPT

One of the significant components that the stakeholders stated that they wanted implemented into their application was ChatGPT. This was noted as a system requirement and no alternatives were considered. ChatGPT is an artificial intelligence chatbot that generates responses to text prompts from users interacting with the system. The knowledge base is quite vast, the chatbot has been fed an enormous amount of data and is constantly learning to improve.

In the context of the CogniOpen software, ChatGPT is being used to take video recordings from the application to find significant objects within them. The CogniOpen software is meant to be a virtual assistant that helps users that suffer from health issues that affect their memory. ChatGPT will get some prompt from the user, requesting the location of something they are having trouble finding. It will return where that object/item was last seen based on the video recordings taken on the app from the user's mobile device.

3.8 Photoshop

Photoshop v2023.Q3 was utilized as the tool to create UI designs related to the CogniOpen software. An alternative tool that was considered for designing the user interface of the application was Pencil. Pencil is a tool that most of the development team is familiar with using, however it is very limited in the designs you can generate. We found this to be quite basic compared to what Photoshop had to offer. Also, our lead UI/UX designer was more familiar with designing application screens with Photoshop. Implementing this tool allowed the team to develop a more modern application. With the target audience (application users) in mind, we were able to develop a UI model that is user friendly and easy to navigate.

4 Frameworks & Languages

This section will cover the frameworks and languages that were used to develop the CogniOpen software. This section will omit any options or alternative considerations as the project stakeholders were clear on the framework and programming language they expected to be utilized. CogniOpen is built using the Flutter development framework. Flutter is an extension of Dart, which is a programming language that is used to build user interfaces in web, mobile, server and desktop applications.

4.1 Dart

Dart is a programming language created by Google, LLC. It addresses deficiencies of legacy technologies in the mobile realm. Firstly, it is platform independent. Code written with Dart will work on Apple iPhone and Google Android. Secondly, it supports hot reload. This means that the coder does not need to rebuild the entire application every time a change is made. Rather, the modified section is pushed to the target device with a single keystroke.

Dart thwarts human error in software engineering by strictly enforcing how data is stored in memory. Variable assignment statements are checked by the compiler to ensure that only reasonable calculations are performed. This includes limiting null, or no data, declarations to the bare minimum.

Parallelism is intrinsic to Dart. Asynchronous JavaScript and XML (AJAX) have risen meteorically in the computer industry. This powerful web browsing technology enables quiet, yet dynamic, screen updates immediately after information becomes available. With Dart, the engineer may forestall subsequent steps if an asynchronous call is integral to multiple features. But if the results of a function are only superficial, they may begin a more complicated task in a virtualized pipeline.

4.2 Flutter

Flutter is packages that are imported into a Dart program. Flutter provides a unified interface for invoking basic smartphone functionality. At the same time, it abstracts distant web services and makes them easy to understand. Flutter is a unique and creative Graphical User Interface (GUI). The controls receive their cues from a person touching their screen.

Each page of a Flutter app is meticulously formulated by the developer. They apprise the Flutter runtime engine to where desired functionality ought to appear. An intricate hierarchy of classes expose distinctions in related pages. By including Flutter packages, handily organized in an online repository, one may program without much regard for the hardware or even paradigm (web, desktop, mobile) that will eventually host the app.

Flutter embraces and extends the pioneering React style. As the user interacts and engages the buttons, the system detects that the screen must refresh on its own. Flutter encapsulates less/more sensitive pages with a stateless/stateful designation, respectively.

5 Development Process

This section will reflect the rationale behind the aspects of the development process that were implemented into the development lifecycle for the CogniOpen application. Project tracking was a very important part of the project that both teams (Team A and B) needed to agree upon so that each team could easily track the progression of work items. Even more important, the branching strategy for all the software developed for the mobile application had to have buy-in from both teams to avoid as much technical conflict as possible. Paired with that is the code review process that would validate and approve all code attempting to be merged to the overall system. Lastly, we touch on a few high-level technical designs and decisions that guide the development of the major system features.

5.1 Project Tracking

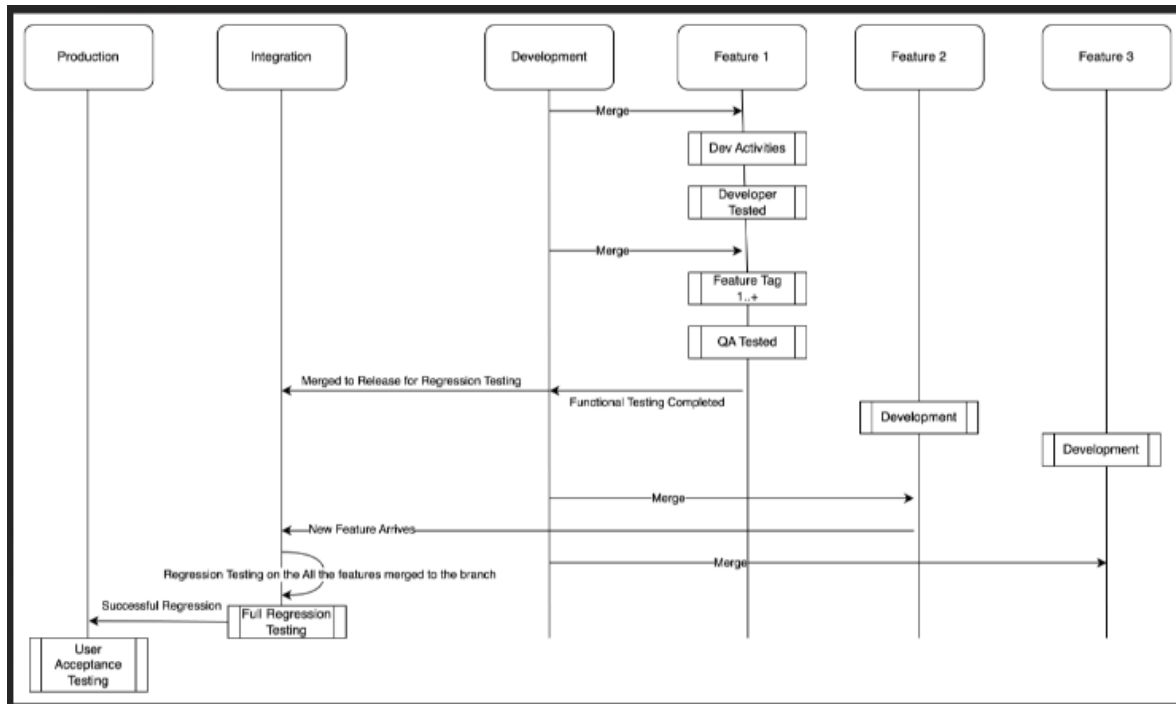
When it comes to determining what would be used to track progression of various tasks involved in the project file, there were two real options (Jira and Azure DevOps). There was a strong consideration to use a Jira board to track the project tasks. More people, than was expected, were familiar with Jira boards and had experience using them.

Ultimately, the Azure DevOps (ADO) board was chosen to be used as the project tracking tool because it is a way for the team to keep track of the progress of work items, associate story points with them and maintain notes that detail the work that was done by project team members. It was also something that was very familiar to those responsible for keeping up with the team tracking board and easy enough to learn for people who worked with similar ticketing systems. This allows the project team to manage and keep track of all work that needs to be completed for the development of the software. ADO also connects smoothly with Git, which is where the source code will be hosted. So, connecting Pull Requests (PRs) to various user stories or Product Backlog Items (PBIs) will help with linking source code implementations to work items on the ADO board.

5.2 Branching Strategy

To standardize a central branching strategy that would work for both teams, there were two approaches that were considered:

Option A:

*Figure 1: Branching Strategy Option A*

Option B:

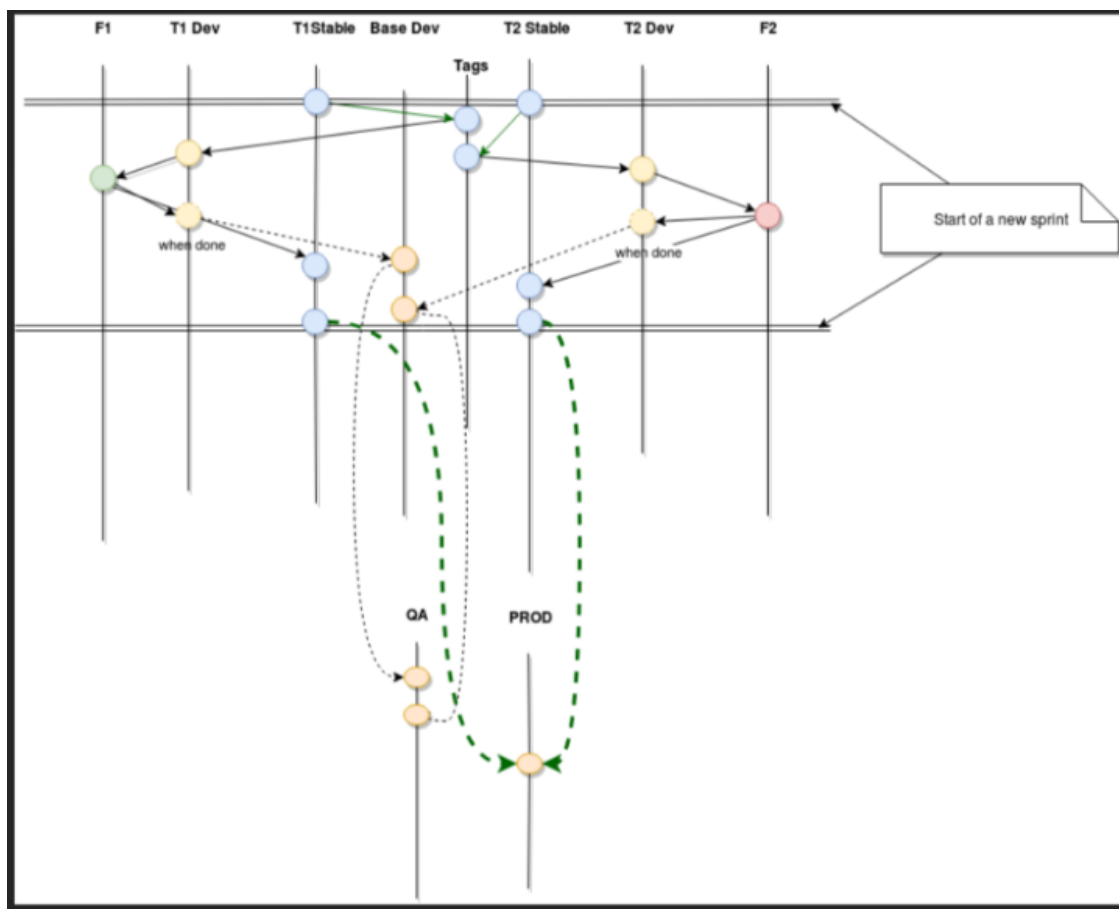


Figure 2: Branching Strategy Option B

The first option (option A) consisted of using feature branches to develop system features simultaneously. Consistently checking for code conflicts to ensure out-of-date code never makes it to the development branch. The development branch is used to flush out problems before things are pushed to the integration branch. The integration branch acts as a gateway to the production branch to ensure development has passed various stages of testing (Regression Testing, User Acceptance Testing). The development, integration and production branches can never be added to directly. They will require reviewers to approve the Pull Request.

The second option (option B) was a possible option that coordinated by having each team have a team 'Development' branch that does not mix with the other team's branch. The stable branches are copies of the development branches, but only contain stable, working code. At the start of the sprint, a tag is made of the corresponding stable branch as a release tracking mechanism. New feature branches are created from the tag (team A feature branches from the team A tag). Completed development is checked into the team 'Development' branch, then into the base development branch. The changes would go to a release branch (called QA in this illustration) for system testing. Whenever

the release branch is at a stable point and ready to release, changes would be merged back to main (labeled PROD in this illustration).

Both teams decided to choose option A because it allowed us to keep development of features continuous and simultaneous. It allows for both teams to develop (at a rapid rate) the various features within the system, without conflicting with other developers. Development on the primary branches is isolated by working on sub-branches for the most part. Testing is conducted at various stages to ensure source code is not being merged that is damaging the overall system. This branching strategy avoids/minimizes the following: (1) merging conflicts of source code, (2) bad code from making it to the production level and (3) new changes to negatively overwrite previous efforts made by either development team.

5.3 Code Review

The code review process aligns with the branching strategy. The two options of the branching strategy yielded different approaches for project code review. The branching strategy that isolates the two teams' development would have required approval from the internal team without any visibility from the other team. The branching strategy that was utilized led to an approach that was more appropriate for the situation. Merging code to the common (Team A and B) branches require an approval from at least one member from each team (two approvals total). This ensures that both teams have visibility to all new the functionality being integrated. It also requires a second pair of eyes, with a different perspective, to review the request. Both teams have complete transparency when it comes to the development of the application and makes working together that much simpler.

5.4 System Overview

Defining and standardizing a high-level system overview that encapsulates all the major components of the user interactions and all the services tied into the actual functionality of the application. The two main options in designing this crucial aspect of the project were: (1) business logic hosted (as backend services) on a web server and (2) business logic is executed by the application running on the user's device. The decision to have the business logic executed by application was made.

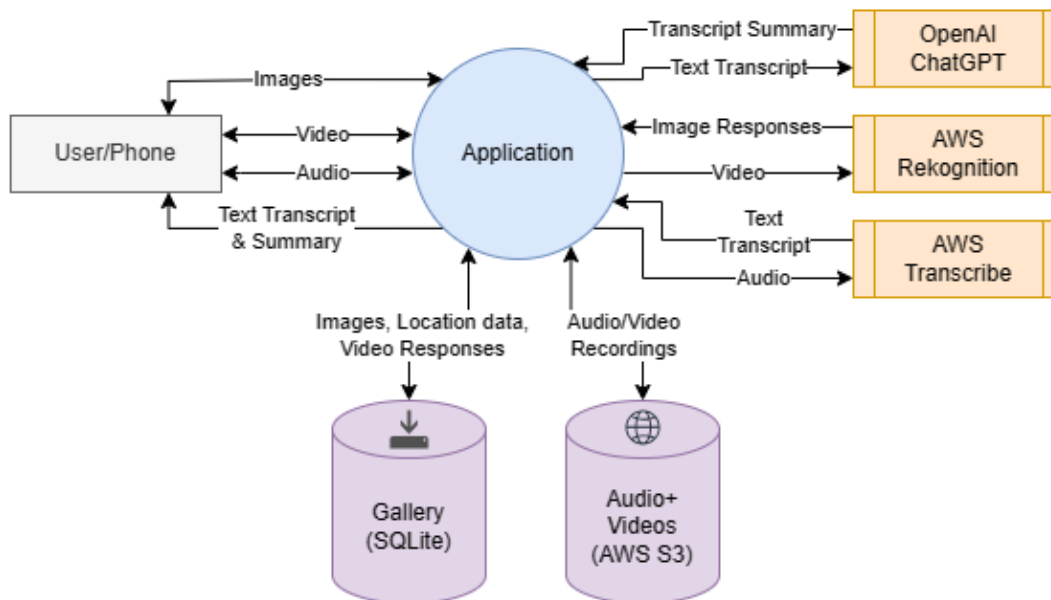


Figure 3: System Overview

That is the case primarily due to time constraints for implementing the backend server. Secondly due to reduce complexity of maintaining an additional component without much gain (YAGNI principle). Application-executed business logic in a “business layer” is also easier to scale because, as users adopt the application, additional server resources are not consumed on our end (if we had a backend server establish, we would need to establish elastic demand capabilities to scale with expansion in user base).

5.5 Architectural Design

With the system overview set in place, the next thing to agree upon was the architectural design for the software system. There were two main considerations for designing this. One was setting up a dedicated backend server to host backend services. Another was to have a backend services layer on the user device (a business layer) to handle requests from the UI (presentation layer). The decision was made to integrate backend services onto the user’s device.

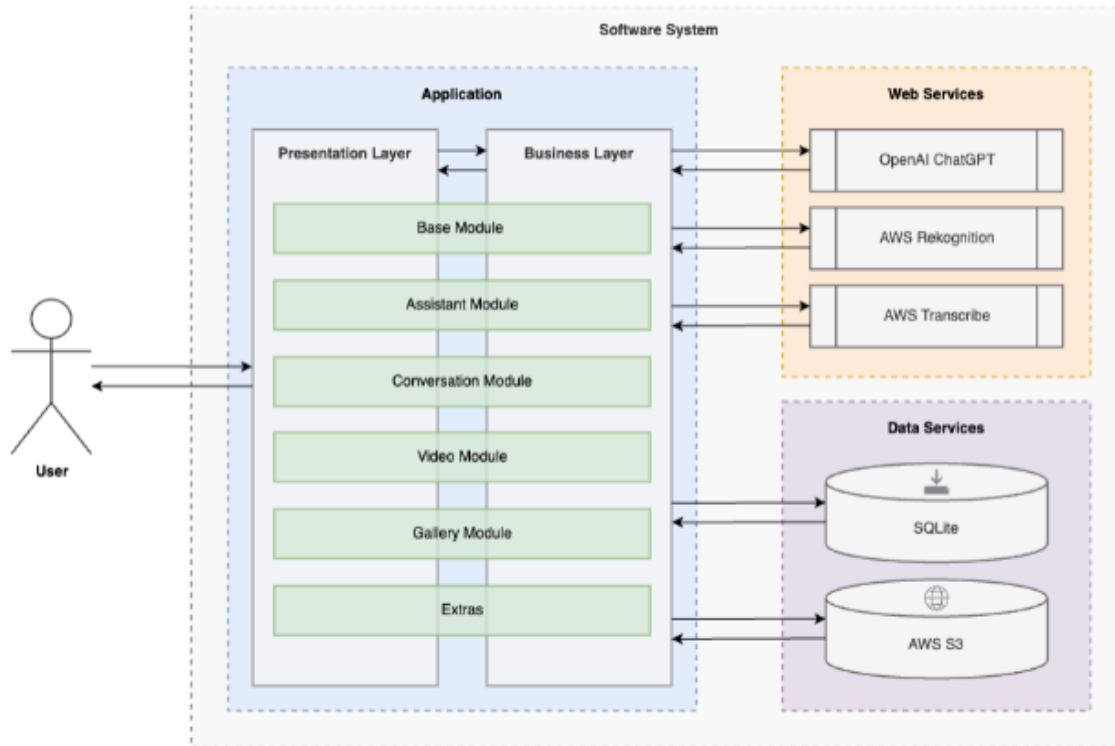


Figure 4: Architectural Design

Due to the limited project budget and timeline to instantiate this application, we decided to put the backend services on the user's device. Rather than provisioning a server to host our business rules and delegate requests/responses from the external web and data services, we crowdsourced the demand for processing resources into each user's phone. This does mean that the user's device must handle the processing of the presentation and business layers. Fortunately, we are not using the application for processing of the video or audio; external services are doing that.

5.6 Video Analysis

The video analyzation for the CogniOpen application had two possible options: (1) using AWS Kinesis for streaming live data into the AWS Rekognition service, or (2) uploading static video recording segments to the AWS Rekognition service. We went with the static video analysis due to complexity, time, and budgetary constraints.

For starters, we investigated AWS Kinesis streams as the mechanism to transport our user's camera feed into the Rekognition (object detection service). Kinesis was the offering recommended by AWS in the AWS Rekognition documentation, so that was the route that we explored. In theory, Kinesis would pick up from a real-time communication feed, created from the user's device, transport the data into a Rekognition consumer, and then Rekognition would save its response (detected objects in the feed) to a S3 bucket.

When it comes to complexity, comparing the programmatic process of converting the user's camera recordings into queried detected-object responses would be more complex in real-time (see image below). Using the real-time feed and Kinesis stream would involve additional technologies, system components, and project resources compared to static analysis.

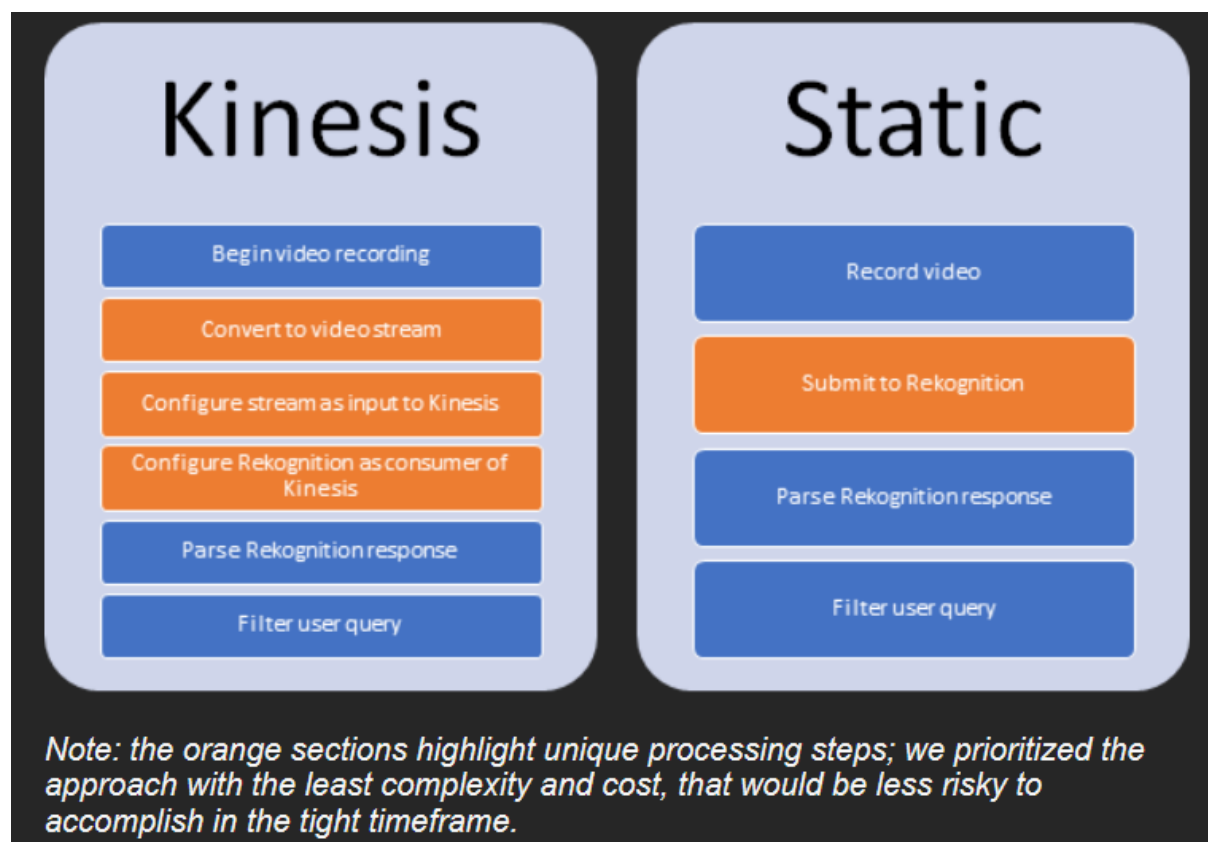


Figure 5: Kinesis Rekognition Streaming vs Static Rekognition Detection

Thinking of time constraints now, the additional system components would require time to research, configure, and implement. We are simply too late in the implementation process of what we have designed and started developing to implement this advanced functionality for this project.

The last constraint, but certainly the most impactful in our decision, deals with financials. In practice, running a provisioned Kinesis data stream for a brief (less than 20 mins) exploratory session incurred a financial cost of \$1.45 USD. A developer running for several hours followed by a tester running for even more hours would have incurred a financial cost that is not inside of the project budget. Therefore, further development of this functionality was not considered. In the future, this feature could be implemented as a "Premium" option for users. Static processing is free of charge because there is minimal cost incurred at this time. In real-time, streaming could be implemented, and then the cost relayed onto the user's subscription (example: \$25/month allows for 5 hours of real-time processing).

The impact of this decision is that there is latency within the system between video submission from the application to Rekognition. Also, responses can be queried from the application. We have made other modifications (chopping videos into increments, polling Rekognition jobs in the backend, and only displaying certain results) to try to hide as much of the latency in the backend. But our approach of static video analysis, while significantly lower in costs, does in fact impact the performance of the application.

6 Test Process

The testing process is an integral part of the development lifecycle because it is important to produce not only working code, but software that adheres to quality standards that also aligns with stakeholder expectations. This section will cover the testing approach for the development of the CogniOpen software. First, you will explore the testing strategy from the highest level to get a sense of how testing integrates with the development process. Then each type of test (unit, integration, and regression) is detailed to provide a synopsis of how various levels of testing were considered and eventually implemented into the project.

6.1 Test Strategy

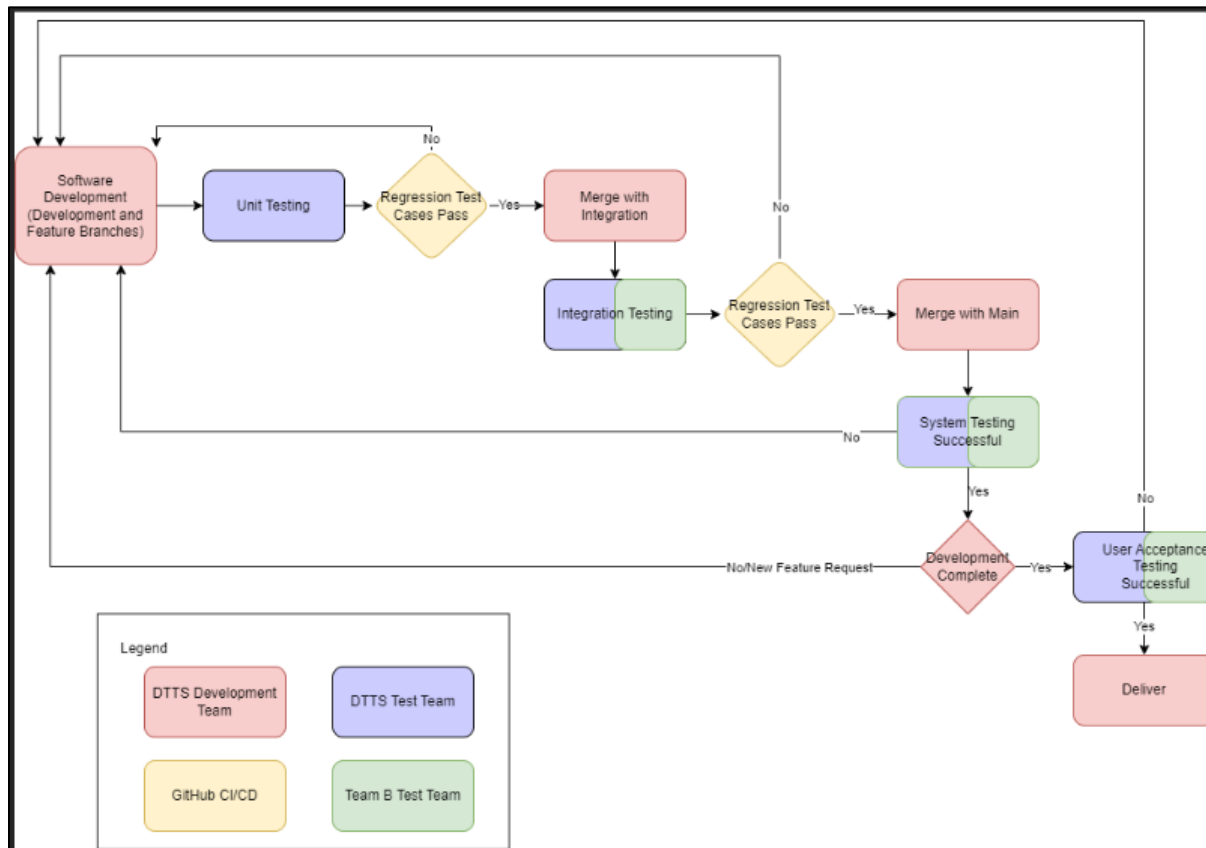


Figure 6: Testing Strategy

In the process of defining the high-level testing strategy for the CogniOpen application, there were a few things to consider. The first being whether we should perform unit and integration testing on the same branch. The second consideration, should testing automation be setup in the command line (using 'flutter test') or automated testing within GitHub? The last thing was deciding on the participation of each development team with the other team's testing.

To the first point, it was decided that unit testing and integration testing will be conducted on separate branches. Regarding test automation, it was decided to use GitHub CI pipeline. Lastly, each team will only be involved in validating the automated testing within GitHub before approving pull requests to be merged.

Integration testing would be risky on the development branch due to instability in the application. Allows testers to focus on unit testing, until the application is more stable. Then only perform integration tests on more stable branches. This approach was also taken because things are error prone, and results are visible to all users. Removing human error to have to remember to test the application. Automation tests are visible to the team via GitHub web actions. Lastly, it is too hard to coordinate with both teams, having their hands in the same test files and having to wait for the other team to create tests. This allows us to control the number of tests and how thorough the tests are that we are creating. This also allows us to get buy-in from team B to validate our tests.

6.2 Unit Testing

Unit testing is significant to the testing process of an application because it allows the team to isolate individual pieces of the code and determine if they work as expected. There were two possible thoughts on how to integrate unit testing into the development of the application. Those were to either perform unit testing on the shared (Team A and B) development branch or individual feature branches that were created by each developer for local development and testing purposes.

Unit testing ended up being performed on the shared development branch, rather than the individual feature branches. Due to the rapid development environment, code needed to be shared to development almost as soon as it was created. We did not have time to wait for tests to be created on each feature branch. However, it is important to note that code being pushed to the shared development branch and used before being tested can cause unexpected bugs.

6.3 Integration Testing

Integration testing expands on unit testing by taking a module, or a group of them, and tests the validity and functionality of that bulk of code. There were many possible choices for how integration would be conducted when it came to the development of the mobile application. Integration testing could be performed on the shared development branch, the main branch, or a dedicated integration branch. The latter was chosen, and integration testing was conducted on a dedicated integration branch. A branch specifically for integration testing, that was stable enough for testing and that minimized the risk of changes to the shared development branch affecting the existing integration tests, was deemed the most optimal solution.

6.4 Regression Testing

Regression testing is vital to the development of the CogniOpen software because it tests whether new code, attempting to be integrated into the overall system, will break the existing infrastructure, or introduce new problems. Because of this, it was only

realistic that regression testing be considered for one of the following branches: (1) shared development branch or (2) the main branch. This would take place anytime there was a push or submitted pull request to one of these branches.

Regression testing was then decided to be conducted on both the shared development branch and the main branch. Reason being, this helps the development team identify errors sooner because regression testing is done on a lower level, shared development branch, and on a higher level, main branch. In retrospect, this approach provides more thorough testing of any and all changes at any point during the development project lifecycle. There will be less defects that occur when the team is rapidly developing.

7 Code Structure

This section will serve as an overview of the code structure of the CogniOpen software as it relates to Team A. We will decipher the mind of the developers that built out the core functionalities of the system. These are categorized by system modules and provide a high-level illustration of these modules along with the rationale behind the development of these features.

7.1 Gallery Module

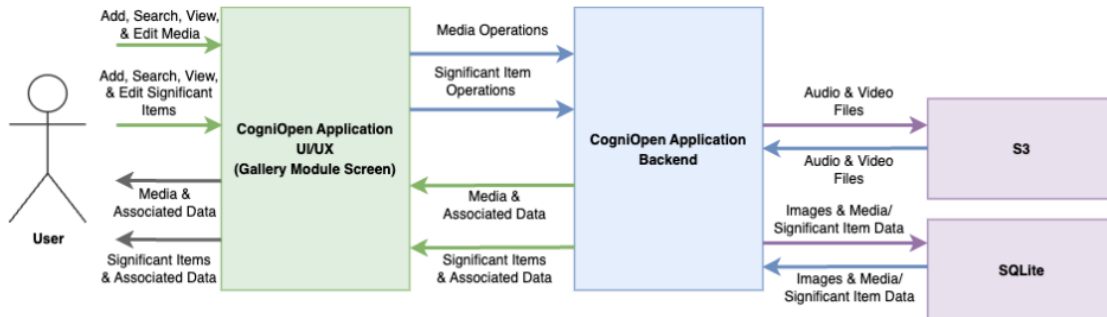


Figure 7: Gallery Module Diagram

The two most reasonable options were to store files locally or use S3 to store them online. The decision was made to utilize local storage instead of S3 for images and significant object tracking. While not shown to the user, recent video responses (from Rekognition) are also stored locally in the SQLite database. Videos would still be uploaded to S3 as they only counted as one call (and they were necessary to be loaded as an object in an S3 bucket for the call to Rekognition).

This decision was made to simplify the process and utilize local phone storage. Keeping with our presentation and business layer design, data objects would be stored locally in the SQLite database, the “src/data_service.dart” class serves as the backend service, and the “ui/galleryScreen.dart” displays the objects from the database to the user.

The impact on the system was less charge to the user because there are no calls to upload and download files from S3. This does mean that more storage is used as the application is used. The system is easier to maintain because changes to the user interface can be implemented separately from changes to the service processes or the database structure. While a change in one may span all three layers (adding a column in the database may affect the parsing completed on the backend, and the display on the frontend), many changes can be completed independently. A change to the UI to rearrange a button would not affect how the call to the database is formatted to get the information.

7.2 Video Module

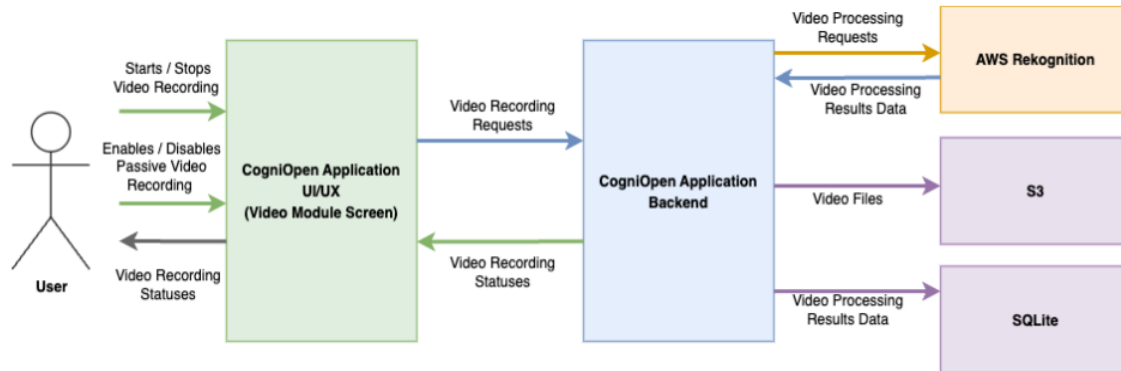


Figure 8: Video Module Diagram

We decided to keep our implementation in line with our design of the video module aspects of the application. A singleton object was created for each of the video processing, S3, and camera manager object class.

The design of the application featured an application frontend (the presentation layer to the user) and an application backend (the business layer, hosted on the user's device). This design enabled us to rapidly test and develop user features while logically organizing the structure of the code. This implementation is easier to maintain. For performance, singleton objects were used so that they could be utilized across the various screens without instantiating new objects nor losing current device state (i.e., the user's most recently completed processed video).

Video processing methods are stored in the `src/video_processor.dart` file; same with the other backend service for the S3 (`src/s3_connection.dart`) and camera management (the managing the user device recording) in the `camera_manager.dart` file. The interfacing with the web service and data service layers is all handled by these backend service "src" classes, leaving the presentation of the results of these methods to the "ui" classes.

The device is still processing all these requests, but the logic is decoupled from the user's experience. As such, when an element of the UI is altered, nothing on the "backend" needs to be changed. Conversely, additional methods can be added or modified on the backend without directly affecting the look and feel of the frontend. The overall experience may change, but the maintainability of the system is improved with this approach. The application does have to create and manage the services request/responses, meaning a higher performance demand on the user's phone.

8 Project File Structure

This section will document the project file structure for two significant areas. One being how project documents are organized and stored for Team A. The other reflects the source code file structure, as it will appear in the GitHub repository. The organization of official and draft documents is important for aligning milestone deliverables and demonstrating the continuous nature to revise previous documents based on the progression of the project. Just as important, the organization of source code files in the repository provides a hierarchical structure for development that will help maintain and support this application.

8.1 Project Documents

8.1.1 Root Level Folder Structure

The root level folders organize all the project documents in sub-directories that align with the major project efforts. A folder 'meetings' is maintained to store internal team weekly meetings, collaboration meetings with Team B, PM leads meetings and stakeholder meeting notes. The 'Misc' folder stores all miscellaneous documents that were used throughout the project lifecycle. The 'Sample & Reference Materials' folder hosts all documents shared with our teams from previous semesters. This helps with constructing our documents to ensure we address the necessary aspects of technical and project documents. You will also notice some folders specific to project documents that have a catalog of diagrams, outlines, and templates.










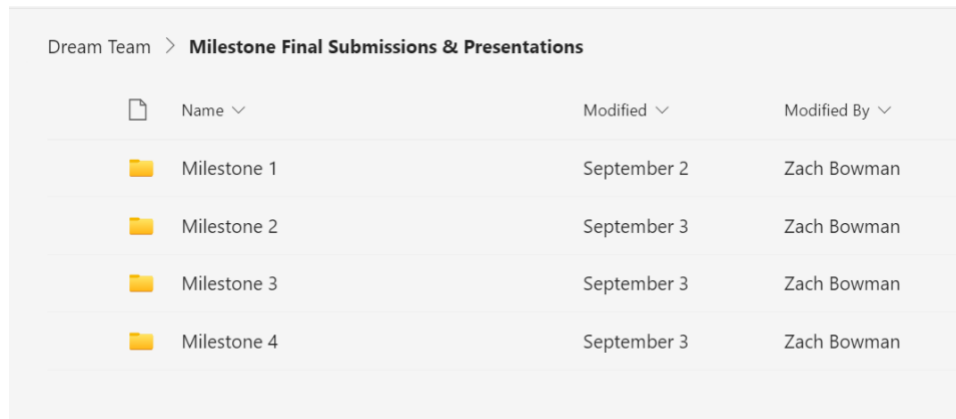
 Name ▾	Modified ▾	Modified By ▾
 Individual Review Templates	September 2	Benjamin Sutter
 Meetings	August 23	Laura Hamann
 Milestone Final Submissions & Presentations	September 3	Zach Bowman
 Misc	August 25	Zach Bowman
 Programmer's Guide	September 26	Kavon Johnson
 Sample & Reference Materials	September 3	Zach Bowman
 TDD Diagrams	September 5	David Bright
 Test Plan Diagrams	September 23	Laura Hamann

Figure 9: Project Document - Root Level Folder Structure

8.1.2 Milestone Folder Structure

One of the folders in the previous section is the 'Milestone Final Submissions & Presentations' at the root level. This folder is organized by each milestone (having a dedicated folder for all four). Each of these four folders has the actual documents that

were submitted to the project stakeholder. Diving deeper (which we will do in the next subsection) you will notice that documents from previous milestones will appear. That is because refinements were made to those documents and the next iterations of those documents were submitted with the next milestone's deliverables. This demonstrates the lifecycle of those documents and how they evolve at different stages of the project lifecycle.

The screenshot shows a web interface for a project named 'Dream Team'. The breadcrumb navigation indicates the current location is 'Milestone Final Submissions & Presentations'. Below this is a table with four columns: a file icon, 'Name', 'Modified', and 'Modified By'. The table lists four milestones, each represented by a folder icon. All milestones were modified on September 3 by Zach Bowman.






Dream Team > Milestone Final Submissions & Presentations			
	Name ▾	Modified ▾	Modified By ▾
	Milestone 1	September 2	Zach Bowman
	Milestone 2	September 3	Zach Bowman
	Milestone 3	September 3	Zach Bowman
	Milestone 4	September 3	Zach Bowman

Figure 10: Project Document – Milestone Folder Structure

8.1.3 Milestone Documents

This is one of the milestone folders (Milestone 2) with all the submitted documents to the professor and the drafts folder. Notice that every word document has a corresponding PDF file (optional), which is just for the benefit of the stakeholders to have viewing options. The word document is necessary as it serves to track changes from the original submission to the newer revisions. Also notice the Milestone 1, revised, documents are placed here for all the reasons mentioned in the previous subsection. And of course, all drafts before we got to the final deliverable, are added to the 'drafts' folder for accessibility within the team.

Dream Team > Milestone Final Submissions & Presentations > Milestone 2		
Name	Modified	Modified By
drafts	September 23	Zach Bowman
Team-A_Milestone-2-Presentation.pptx	September 23	Benjamin Sutter
Team-A_Project-Plan_v2.docx	September 25	David Bright
Team-A_Project-Plan_v2.pdf	September 23	Zach Bowman
Team-A_Software-Requirements-Specificati...	September 23	David Bright
Team-A_Software-Requirements-Specificati...	September 23	Zach Bowman
Team-A_Technical-Design-Document_v1.docx	September 23	Zach Bowman
Team-A_Technical-Design-Document_v1.pdf	September 23	Zach Bowman
Team-A_Test-Plan_v1.docx	September 23	Zach Bowman
Team-A_Test-Plan_v1.pdf	September 23	Zach Bowman

Figure 11: Project Document – Milestone Documents

8.2 Source Code File Structure

The source code for our application has been structured to uphold the principle of separation of concerns, ensuring scalability and ease of maintenance. The main divisions encompass Business Logic, UI, and Database layers.

8.2.1 Business Logic Layer

The Business Logic Layer encapsulates the core functionalities and logic of the application. Its independence from the UI and Database layers guarantees that modifications in those layers won't jeopardize the business logic's integrity.

Directories & Files:

- **controller**: Serves as the nerve center, bridging the UI and the database, orchestrating application logic.
- **repository**: This is the gateway to data operations, mediating communications with the database layer.
- **utils**: A collection of utility tools, featuring files like `file_manager.dart` and `format_utils.dart`, to bolster diverse operations.
- **s3_connection.dart**: An interface for AWS S3 storage operations, including file uploads, deletions, and bucket management.

8.2.2 UI Layer

The UI Layer is the application's face, embodying all elements the end-user interacts with, ranging from screens to widgets.

Directories & Files:

- **ui**: The visual epicenter, comprising major screens such as `homeScreen.dart`, which is the landing page; `loginScreen.dart`, facilitating user authentication; and `settingsScreen.dart` for customization preferences.

- **assets:** A vault for static elements like images, fonts, and other multimedia assets.

8.2.3 Database Layer

The Database Layer is the guardian of data-related tasks, standing as the pillar for data persistence, consistency, and integrity.

Directories & Files:

- **database:** This houses core database configurations, including the master setup file `app_database.dart`.
- **model:** This is the blueprint chamber, defining the structure and attributes of data entities before their journey to or from the database.

9 User Interface

This section will detail the UI screens developed by Team A for the CogniOpen software. The screens displayed below were gathered from an emulator that was successfully running the application. Each subsection in this section will provide the internal and external functionality of the screen. We will discuss the home screen, gallery screen, video recording screen, and significant objects screen.

9.1 Home Screen

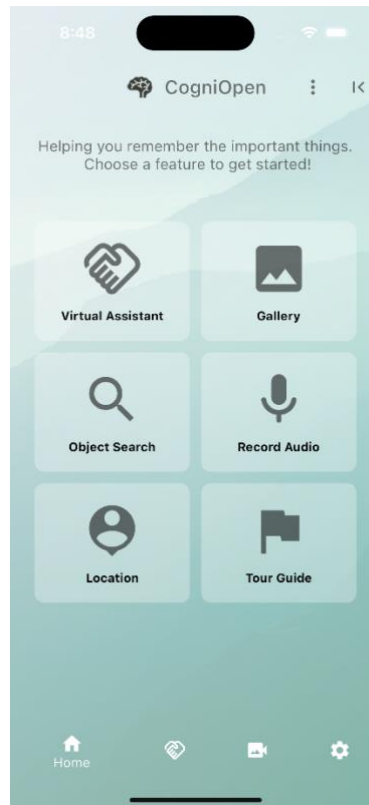


Figure 12: Home Screen

The home screen serves as the gateway to the internal features of CogniOpen. It provides the actor with a quick way to access any functionality in the system.

9.1.1 Internal functionality

The internal functionality of the home screen includes initializing databases and services associated with CogniOpen's features such as Amazon S3, and ChatGPT APIs.

9.1.2 External functionality

The external functionalities of the home screen include:

- Virtual Assistant button – Opens the Virtual Assistant screen, activating the ChatGPT-enabled virtual assistant
- Gallery button – Opens the Gallery screen
- Record Video button – Opens the Record Video screen
- Record Audio button – Opens the Record Audio screen
- Significant Object button – Opens the Significant Objects screen
- Tour Guide button – Opens the tour guide screen

9.2 Gallery Screen

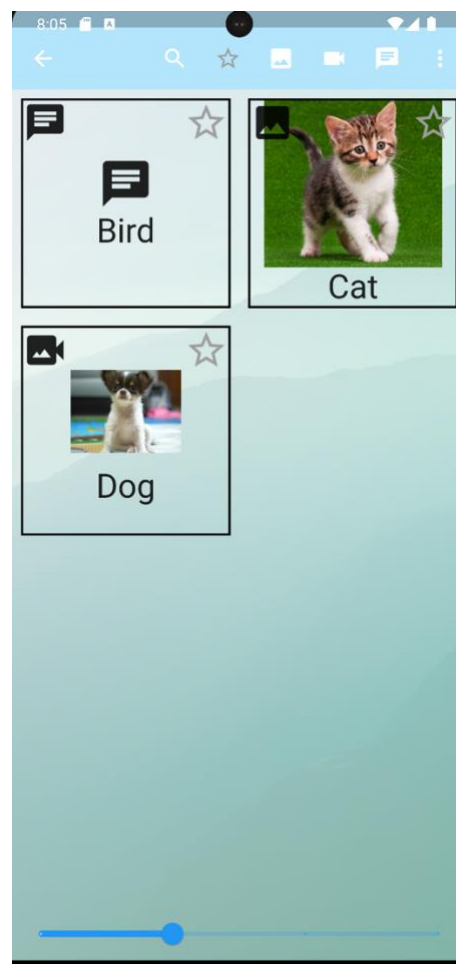


Figure 13: Gallery Screen

The gallery screen organizes and displays conversations, videos, and photos the actor has saved. It serves as a central location for the actor to access and manage their multimedia content.

9.2.1 Internal functionality

The internal functionality of the gallery screen includes the initialization of the database, which is queried to populate and display the content on the screen. Metadata associated with each item, such as automatically assigned and user-defined labels, thumbnails and timestamps are categorized and prepared to display with the content.

9.2.2 External functionality

The external functionality of the gallery screen includes:

- Search button – Open the search bar for the user to query the database
- Favorites button – Queries the database to populate favorited items
- Images button – Queries the database to display only images
- Videos button – Queries the database to display only videos
- Conversations button – Queries the database to display only conversations
- Gallery item button – Opens the Gallery item details screen, presenting the metadata associated with the item, such as timestamp, labels, and media type
- Grid layout bar – Changes grid layout size of the gallery screen

9.3 Video Recording Screens



Figure 14: Video Recording preview screen

The Record Video screen allows the actor to view the camera on what is currently being recorded.

9.3.1 Internal functionality

The internal functionality of the Record Video screen includes initializing the device's camera to record video and audio.

9.3.2 External functionality

The external functionality of the Record Video screen includes:

- Stop button – Stops the video and save the recorded video into the gallery
- Record button – Records the video

9.4 Significant Objects Screen

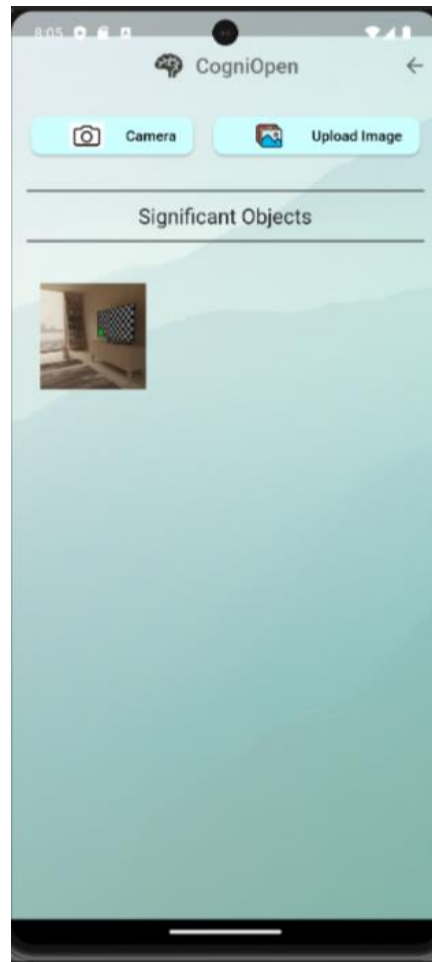


Figure 15: Significant Objects Screen

The Significant Objects screen allows the actor to view objects which they've determined to be significant to them.

9.4.1 Internal Functionality

The internal functionality of the Significant Objects screen includes the initialization of the database, which is queried to populate and display the content on the screen.

9.4.2 External Functionality

The external functionality of the Significant Objects screen includes:

- Camera button – Opens the device's camera to take a photo and assign a significant object.

- Upload Image button – Opens the gallery to assign a significant object.

10 Data & Backend

CogniOpen leverages Amazon Web Services (AWS) for data storage and artificial intelligence (AI) capabilities. AWS is a wide array of sophisticated computing instruments. In the past, customers were forced to purchase their own hardware to host servers. Those machines would often be underutilized and poorly maintained. AWS revolutionized the field of information technology (IT) by centralizing those resources and making them available widely over the Internet.

10.1 Simple Storage Service

AWS Simple Storage Service (S3) is an online directory structure. One creates a Bucket to demarcate file groups. One further subdivides Buckets with hierarchical custom folders. Bucket names must be unique within the region of AWS in which they reside. Folder names must only be unique intra-Bucket. Each object in the bucket receives a Uniform Resource Locator (URL), distinguished by its own key (filename). Videos, photographs, and audio recordings captured by a CogniOpen user's phone are immediately added to S3 by the system.

S3 has powerful administration tools that are available on the web console, command line, and within compiled code. In the first case, credentials are entered manually by the engineer. The second and third cases require a pre-configured access key. CogniOpen makes asynchronous procedure calls in its compiled statements to create a Bucket, populate it with objects, and remove entities if they are no longer needed.

10.2 Rekognition

AWS Rekognition identifies the contents of images and videos. CogniOpen users endeavor to parse the experiences of their lives for the most essential elements. To do this, they record and add raster pictures to their individual CogniOpen managed collection of information. Once those objects are stored in S3, Rekognition begins to process them.

Rekognition outputs JavaScript Object Notation (JSON). This is a series of nested curly braces and tag-to-value data pairs. It is text. Each image and video in CogniOpen will receive a list of labels. The data is ephemeral. Rekognition does not persist the associations, and neither does S3. CogniOpen wields the smartphone's local hard drive to do that. Specifically, the database row for the still/moving picture strips the extraneous punctuation from JSON. The labels are matched to voice commands in the cutting-edge audio intake mechanism.

10.3 Sqflite

Data storage is one of the most important components of the application and therefore needed to be developed in a way that was easy to integrate while still providing the value that was needed. Before making a final decision on this implementation, we tested and thought about the most optimal and efficient way to store the system's data. This included the possibilities of using text files and setting up a database server.

We decided to use this plugin, Sqflite, as it is the library for Flutter apps to use an SQLite database. We needed local data storage that would be easy to query, can load and store data relatively easily compared to storing data in individual text files. SQLite creates a persistent database file on the user's local device, as opposed to setting up a database or using raw JSON files. Taking this approach allows the application to store vast amounts of data locally and query through it much more efficiently when handling data requests and ChatGPT interactions.

11 Publishing

Please refer to the Deployment and Operations Guide for instructions on how the CogniOpen application is deployed and operated.

Appendix A

Flutter App Dependencies in GitHub -

<https://github.com/umgc/fall2023/commit/099595c630ebd599a5c7c5fd47ec1cd7a19744a1>