DREAM TEAM

TECHNOLOGY SOLUTIONS

CogniOpen Software Application

Test Plan

Project Name: CogniOpen Software Application
Document: Test Plan (TP)
Version: 3.0
Date: November 7, 2023
Prepared By: Team A

Team Lead / Project Manager:
Vincent Galeano
Dream Team Technology Solutions LLC (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	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
02/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/Oct/2023	3.0	Milestone 4 version	DTTS

Table of Contents

Figures & Tables

1 Introduction

This document defines the software testing plan as well as full set of unit, functional, and non-functional test cases for the CogniOpen Software Application (App). All test cases must be specific, measurable, attainable, realistic, and testable (SMART).

1.1 Purpose

The purpose of this Test Plan document is to provide a comprehensive testing framework that defines a detailed testing plan as well as the unit, functional and non-functional test cases to be executed. It aims to establish a shared understanding amongst the stakeholders (including the development team, project manager, clients, and end-users), about how the project's software and coding quality is being managed. For high-level details on the project Software Quality Assurance (QA) Plan for software and code please reference the Project Plan; Sections 9.2.2 and 9.2.3.

Automated continuous integration (CI) testing for all test cases will ensure new software features are functioning as desired and to detect any defects early on in the development lifecycle. This minimizes the chances of the client finding bugs and reduces the Mean Time to Detect (MTTD) defects. A separate Test Report document will be created to capture the test results.

1.2 Scope

The following sections define what types of testing and functionality will be included within this document.

1.2.1 In Scope

This document will define all unit, functional and non-functional test cases required to test all Use-Case Reports listed in the SRS Section 3.1, as well as the Non-Functional Requirements listed in the SRS Section 4.0. The act-like-a-customer (ALAC) method will be followed to create tests with the mindset that all software is developed with bugs and that by defining a wide range of tests to cover all the ways a client will use it, there will be a better chance of uncovering defects during testing.

1.2.2 Out of Scope

This document will not define any functional or non-functional test cases which are defined within the SRS of Team B. For specifics, please see the Project Plan; Section 3.2.2, which defines out of scope functional requirements for DTTS.

In addition, there will be no testing defined for external web or data services. These services are assumed to have their own quality standards and to be tested by the organizations who provide the services. These services include:

- OpenAI Chat Generative Pre-trained Transformer (ChatGPT)
- Amazon Web Services (AWS) Rekognition
- AWS Transcribe
- SQLite
- AWS S3

1.3 Constraints

In the testing process for the CogniOpen application, several constraints need to be considered. These constraints may impact various aspects of the testing effort and require careful management to ensure the successful execution of the test plan.

Resource: The testing process operates within predefined financial boundaries, which affects the allocation of resources for acquiring testing tools, hardware, and environments. Given these limitations, it's essential to employ cost-effective testing strategies and prioritize resource usage effectively.

Time: Delays in development or other project phases can lead to a compressed testing schedule, limiting the available time for comprehensive testing activities. Release deadlines also impose time constraints on testing cycles, emphasizing the need to complete testing within stipulated timeframes to meet project milestones and ensure timely product delivery.

Technical: The CogniOpen application relies on external AI services like ChatGPT and AWS Rekognition for specific functionalities. The availability, reliability, and performance of these external services can impose technical constraints, particularly during integration testing scenarios. Device and platform variability represent additional technical constraints. Testing across a broad spectrum of devices, operating systems, and platforms may be limited due to budget and resource constraints. For example, developing iOS applications requires all developers needing Apple OS in order to code/test.

Scope: Decisions regarding which features are included or excluded from the current release determine the testing scope. Features deemed out of scope may not undergo testing in the current test cycle.

Compliance and Regulatory: The testing process must adhere to legal and regulatory requirements, including data privacy laws and accessibility standards. Security and privacy constraints also come into play, necessitating rigorous testing for vulnerabilities and data protection measures. Compliance with security standards and regulations is essential but may add complexity to testing.

1.4 Assumptions & Dependencies

The testing process relies on several critical assumptions and dependencies that must be acknowledged and managed throughout the testing lifecycle.

1.4.1 Assumptions

1. It is assumed that the testing team will have access to adequately configured test environments that mirror the real-world conditions in which the CogniOpen application will be used. These environments should accurately represent the target devices, operating systems, and network configurations.

2. The testing scenarios and test data assume an accurate representation of real-world usage patterns. Test cases should simulate user interactions, data inputs, and environmental conditions as closely as possible to ensure comprehensive testing.

3. In the event of critical defects identified during testing, it is assumed that the development team will respond promptly to address and resolve these issues. Effective communication and collaboration between the testing and development teams are crucial for timely issue resolution.

1.4.2 Dependencies

1. The testing process depends on the availability and reliability of external AI services, including ChatGPT and AWS Rekognition. These services are integral to the functionality of the CogniOpen application, and any disruptions or unavailability may impact testing activities, particularly integration testing.

2. The testing team relies on the development team to provide timely and stable software builds for testing purposes. Frequent and consistent software builds are essential for test execution and validation.

3. Effective collaboration with project stakeholders, including project managers and developers of DTTS and Team B, and end-users, is a dependency for issue resolution and validation of user requirements. Timely feedback and clear communication channels are crucial for addressing issues and ensuring alignment with project goals.

Acknowledging and managing these assumptions and dependencies are essential to mitigate potential risks and challenges during the testing process. Continuous monitoring and communication between testing and development teams, as well as regular updates to testing documentation, help ensure that testing activities proceed smoothly and effectively.

1.5 Document Conventions & Organization

This document is broken up into two main sections; Section 3.0 Testing Overview and Section 4.0 Test Cases. The Testing Overview Section provides details on who will be

involved in the testing, how the testing will be implemented, when the testing will be performed, what types of testing will be performed, as well as testing deliverables and how defects will be categorized and handled. The Test Cases Section defines the three main types of test cases which will be executed throughout the testing process: Unit, Functional, and Non-functional.

1.6 Intended Audience & Reading Suggestions

This document aims to establish a shared understanding amongst the stakeholders (including the development team, project manager, clients, and end-users), about how the project's software and coding quality is being managed through unit, integration and regression testing. For a full report on the output of all test cases, including a general application status, please see the Test Report.

1.7 Project Document Suite

This Test Plan 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

Bharath. (2023). *Flutter unit testing — the beginner’s guide*. Medium. https://betterprogramming.pub/flutter-unit-testing-the-beginners-guide-35105164722e

Flutter. (n.d.). *Testing Flutter apps*. https://docs.flutter.dev/testing

Kindacode. (2023). *Android Studio system requirements (2023)*. https://www.kindacode.com/article/android-studio-system-requirements

Microsoft. (2023a). *Define, capture, triage, and manage software bugs in Azure Boards*. https://learn.microsoft.com/en-us/azure/devops/boards/backlogs/manage-bugs?view=azure-devops

Microsoft. (2023b). *Query by rank and picklist value in Azure DevOps and Azure Boards*. https://learn.microsoft.com/en-us/azure/devops/boards/queries/planning-ranking-priorities?view=azure-devops

UMGC (2020). *Capstone Project Guide.* SWEN 670: Software Engineering Capstone, University of Maryland Global Campus (UMGC). https://learn.umgc.edu/d2l/le/content/920456/viewContent/31091351/view

UMGC (2023). *Syllabus.* SWEN 670: Software Engineering Capstone, University of Maryland Global Campus (UMGC). https://umgc.campusconcourse.com/view_syllabus?course_id=255524

1.9 Terms, Abbreviations, & Acronyms

This section will provide a glossary of terms, abbreviations, and acronyms used throughout the Test Plan document. It aims to ensure a shared understanding of terminology among all stakeholders involved in the project.

Term	Definition
AI	Artificial Intelligence
ALAC	Act Like a Customer
App	Application
AQL	Acceptable Quality Level
AWS	Amazon Web Services
ChatGPT	Chat Generative Pre-trained Transformer
CI	Continuous Integration
DTTS	Dream Team Technology Solutions
GPS	Global Positioning System
GUI	Graphical User Interface
IDE	Integrated Development Environment
MTTD	Mean Time to Detect
OS	Operating System
PP	Project Plan
QA	Quality Assurance
QE	Quality Engineering
SDLC	Software Development Life Cycle
SMART	Specific, measurable, attainable, realistic, testable
SRS	Software Requirements Specification
TP	Test Plan
TR	Test Report
UAT	User Acceptance Testing
UMGC	University of Maryland, Global Campus

Table 2: Terms, Abbreviations, & Acronyms

2 System Overview

The CogniOpen software application will have two main actors on the application: the memory-impaired individual and the caregiver/family member. The CogniOpen software application will provide a guided tour and help center to assist all actors in accomplishing tasks within the application. In addition, a settings module to toggle location services will also be available to provide expanded user functionality. The main functionality can be broken down into four separate high-level modules: the assistant module, conversation module, video module and gallery module.

Within the assistant module, actors will be able to ask the assistant a question about a particular list, conversation, or video. Actors will also be able to ask the assistant where they are when location services have been enabled. In all cases, the assistant will respond with the appropriate response or location.

Within the conversation module, actors will be able to record a conversation, search conversation history, edit a conversation which was erroneously transcribed, and enable a passive conversation recording mode.

Within the video module, actors will be able to record a video, search video history, edit video history which was stored erroneously, enable passive video recording mode, and manage significant items. Significant items are anything that can be captured by a video, and which are commonly used by the actor.

Within the gallery module, actors will be able to view previously captured photographs and videos. Viewing items within the gallery will also provide the actor with associated data.

3 Testing Overview

The following section will define the testing approach including who, what, when and how test results and defects will be identified and handled.

3.1 Testing Roles & Responsibilities

These roles and responsibilities converge to form a unified and organized testing process of CogniOpen. Together, they detect and report defects, validate functionality, and ensure the software conforms to business objectives and end-user requirements. This collaborative effort ensures that the software is not only devoid of critical flaws, but also of high standards and user satisfaction.

Number	Team Member	Role	Responsibilities
1.	Laura Hamann	Lead Test Engineer	▪ Defines the overall strategic direction for the testing process and sets project objectives. ▪ Identifies and describes test tools and techniques to be used in the testing process. ▪ Verifies and assesses the overall test approach, ensuring it aligns with project objectives.
2.	Zach Bowman Richard Tsang Vivek Goel Benjamin Sutter	Test Engineers	▪ Execute the defined test cases and scenarios. ▪ Log and maintains comprehensive records of test results. ▪ Report any identified defects or bugs, providing detailed information for resolution.
3.	Vincent Galeano	Team Lead / Project Manger	▪ Acquires the necessary resources, including personnel, tools, and infrastructure, to support testing activities.
4.	David Bright Benjamin Sutter Richard Tsang Andrea Pellot Selam Biru	Software Developers	▪ Implement test cases, test programs, and test suites based on the provided test specifications. ▪ Collaborate closely with the Test Lead to ensure that the testing process aligns with development efforts.
5.	Zach Bowman Vivek Goel Selam Biru	Business Analyst	▪ Collaborate with the Lead Test Engineer to provide business context, requirements, and use cases for testing. ▪ Ensure that testing efforts align with business goals and user expectations.

Table 3: Quality Specialists

3.2 Testing Resource Estimate

This Resource estimation involves identifying and quantifying the resources required to complete testing tasks for CogniOpen. It estimates the resources necessary for creating the Test Report, which includes a time estimate in hours.

Number	Resource Type	Estimated Quantity	Estimated Effort
1	Lead Test Engineer	1	30 hours
2	Test Engineers	4	30 hours
3	Team Lead / Project Manager	1	5 hours
4	Software Developers	5	10 hours
5	Business Analysts	3	10 hours

Table 4: Time Commitment

3.3 Testing Method

This section will provide the overall testing approach including details on how it will fit in with the development and project lifecycle. Each type of testing will detail what will be tested, when it will occur, who will be doing it, if the testing is manual or automated, and the outcomes from each type of test. Below is an overall diagram depicting the process.

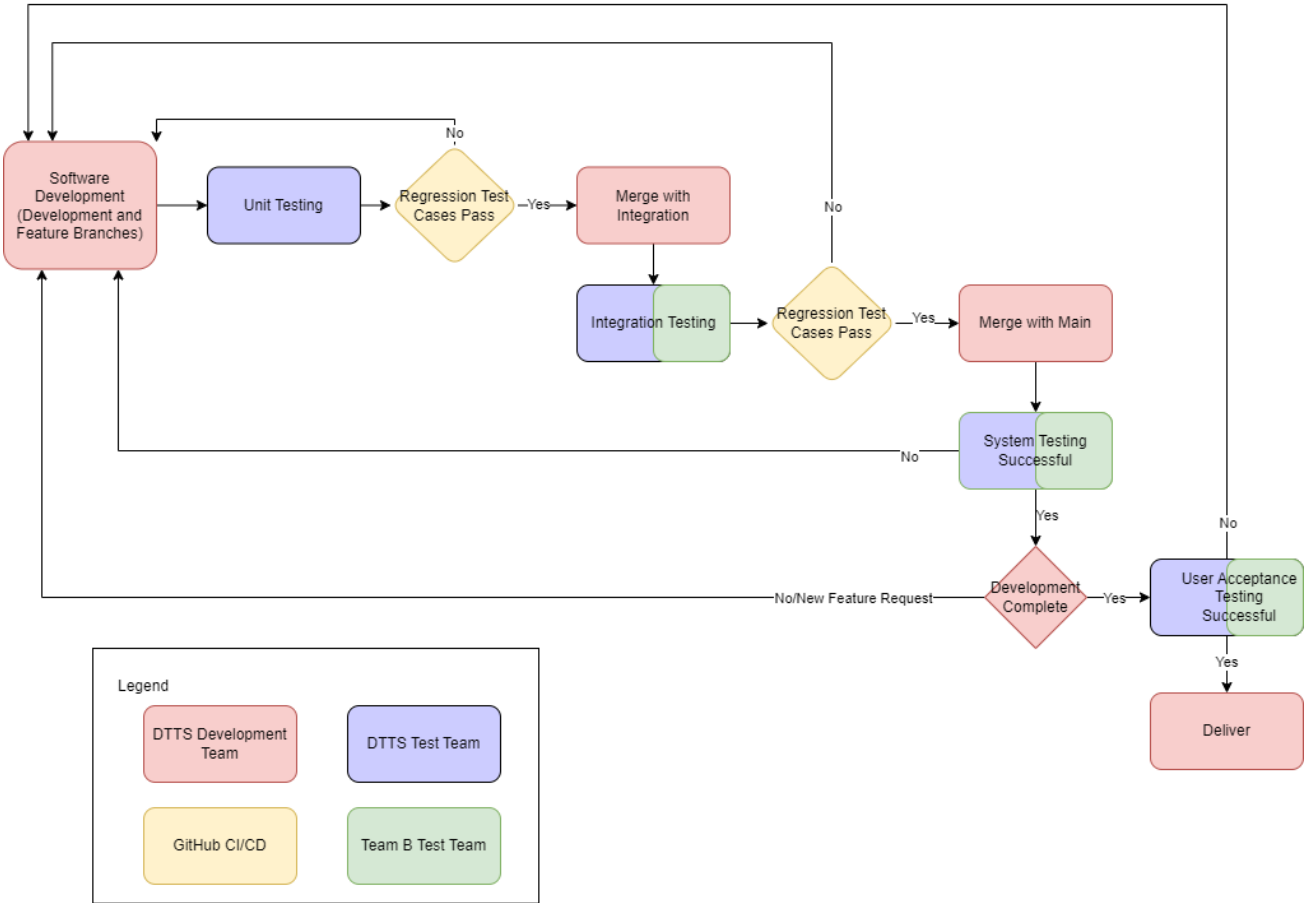

Figure 1: Testing Lifecycle

3.3.1 Unit Testing

Unit testing will be part of the automated testing process. The individual test cases will be documented in Section 4.1: Unit Tests, as they are developed. This type of testing will be used to verify the correctness of small units of code/functions/widgets within CogniOpen. Unit tests will be written and run by the testing team after development on a particular piece of code is complete. As new branches of code become available, test engineers will be assigned to test specific functions. The test engineers will check out the appropriate branch, create test cases, update the necessary sections within Section 4.0 of this document, and verify that all tests are successfully passing. Upon successful completion of the unit tests, the code is then deemed ready to be integrated. Below is a diagram depicting how the automated unit tests fit in with the development cycle.

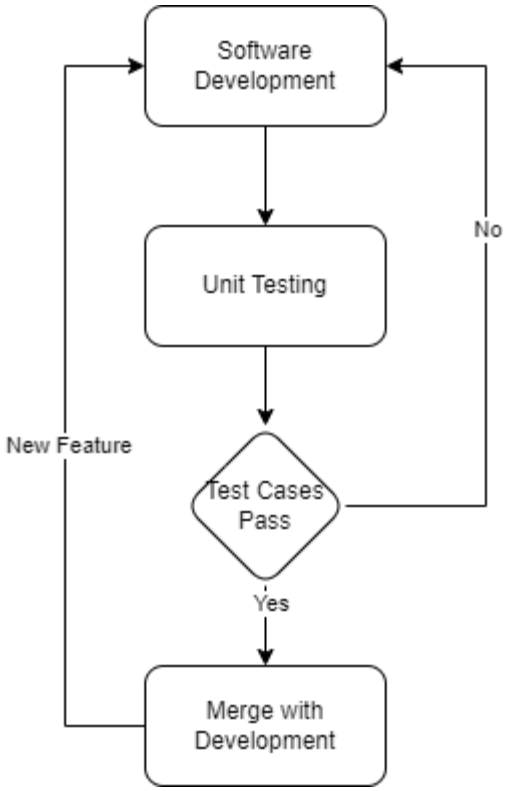

Figure 2: Unit Testing Process

3.3.2 Integration Testing

Integration testing will be part of the automated testing process. The individual integration test cases will be documented in Section 4.2: Functional Tests. This type of testing will be used to verify new components of CogniOpen as they are developed. The integration tests will be written and run by the testing team once the software development team deems a new component of CogniOpen complete. Upon successful completion of the integration tests, the new component is deemed ready to be integrated. Below is a diagram depicting how the automated integration tests fit in with the development cycle.

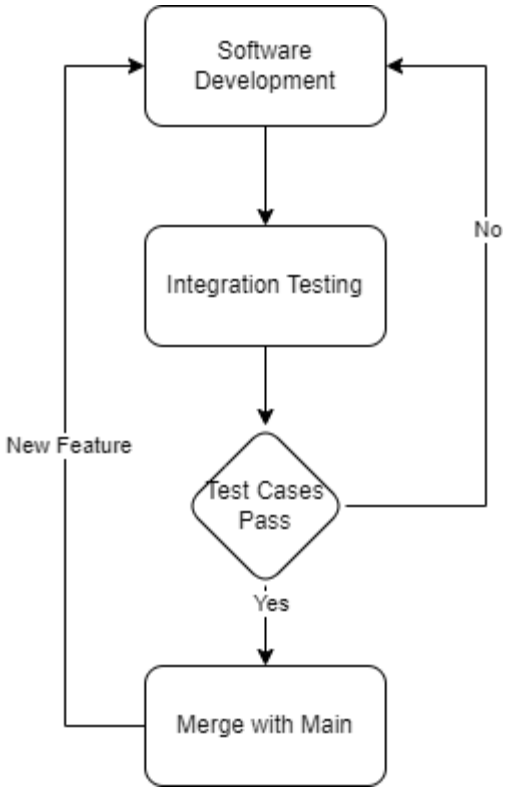

Figure 3: Integration Testing Process

3.3.3 Regression Testing

Regression testing is an automated process and will be performed as part of the continuous integration (CI) process. This testing will ensure existing functionality has not been altered or broken due to new functionality. Automated regression testing will be triggered by a pull request and a merge with both the development and main branches.

3.3.4 4System Testing

System Testing will be a manual process performed by the testing team for DTTS as well as the testing team for Team B. This testing will be performed once the application functions have been completed. The testing team does not need to wait until the full application is completed to perform this testing, however they do need to wait until chunks of functions are completed in their entirety to test. This will verify that the system is behaving as expected prior to entering into User Acceptance Testing (UAT) with the client.

3.3.5 User Acceptance Testing

User Acceptance Testing will be a manual testing process performed by the testing team and client. This testing will be performed once all software development is complete, and the product is ready to be delivered. This testing's intended purpose is to validate all functional and non-functional requirements as promised to the client. Upon

successful completion of this testing, CogniOpen will be delivered to the client along with the test results.

3.4 Testing Automation

All test cases written as part of the development process will be executed by the test team. Using GitHub CI/CD pipeline, a pull request and merge with both the development branch and main will trigger automatic testing of all unit and integration tests. The results will be displayed to the user under the pull request and can also be found user the Actions tab of the GitHub project page. Any failures will be assigned an appropriate priority level as defined in section 3.11 below and be entered into the Microsoft Azure board as a new bug.

3.5 Testing Deliverables

When delivering testing results and findings for CogniOpen, the client should receive the following documents:

Test Plan: is the blueprint for the testing tasks. It specifies the goals, methodology, tools, and timeline for testing CogniOpen. It lays out an organized plan for how and where tests will be performed.

Test Report: is an essential piece of documentation that compiles and analyzes the results of all testing activities. This also includes details on any defects, issues, or bugs found during the testing procedure. Details like the defect's severity, priority, description, and status (open, working or fixed) should all be recorded here.

3.6 Testing Tools

Name	Use
Android Studio Giraffe 2022.3.1 Patch 1 with Android SDK v34.0.0	Integrated Development Environment (IDE) used to test CogniOpen on Android
Dart Testing Library (test 1.24.6)	Unit testing of the business layer of CogniOpen
Flutter SDK Testing Package (flutter_test)	Unit testing of the presentation layer of CogniOpen
Flutter SDK Integration Testing Package (integration_test)	Integration testing of CogniOpen
Dart Mockito Library (mockito 5.4.2)	Used to mockup data required for unit and integration testing including data from live web services and or databases.
Git	Versioning control system used to track developed test cases.
GitHub CI/CD	Used for continuous integration testing. Will trigger unit and integrations tests using event hooks with GitHub.
Microsoft Excel	Used to track testing details such as who, when, what and results.

Table 5: Software Tools

3.7 Testing Environment

The tests can be developed and executed in a Windows, macOS or Linux environment. The following table lists the corresponding system requirements needed to successfully use the tools identified in Section 3.6: Testing Tools.

Operating System	Requirements
Windows	▪ OS: Windows 10/11 64-bit ▪ CPU: Intel Core i5-8400 3.0 GHz or better ▪ Memory: 16 GB RAM ▪ Free storage: 30 GB (SSD is strongly recommended) ▪ Screen resolution: 1920 x 1080
macOS	▪ OS: macOS 10.15 (Catalina) ▪ CPU: Intel Core i5-8400 3.0 GHz or better ▪ Memory: 8 GB RAM ▪ Free storage: 30 GB (SSD is strongly recommended) ▪ Screen resolution: 1920 x 1080
Linux	▪ OS: Any 64-bit Linux distribution that supports Gnome, KDE, or Unity DE ▪ CPU: Intel Core i5-8400 or better ▪ Memory: 8 GB ▪ Free storage: 20 GB SSD ▪ Free resolution: 1920 x 1080 ▪ GNU C Library (glibc) 2.19 or later

Table 6: Hardware Specifications

3.8 Test Data Sources

The data used for automated unit and integration testing will come from one of the following sources:

- Pre-recorded images and videos
- Mocked up database and web services using Mockito

3.9 Test Schedule

This Task Schedule outlines the timeline and essential tasks involved in the testing process of CogniOpen, from initial planning to test closure. It ensures that testing activities are executed systematically and in accordance with project goals and deadlines.

Task Schedule	Task Name	Methods
08/16/2023	Test Planning and Strategy	Test planning, strategy development, resource allocation
09/23/2023	Test Case Design	Test case creation, design, and review of testing tools
09/23/2023	GitHub CI/CD Integration	Setup CI/CD integration to allow automated regression testing
09/30/2023	Unit Test Start	Unit test case creation, execution, test automation tools
09/30/2023	Defect Identification and Reporting	Defect tracking, reporting
10/07/2023	Integration Test Start	Integration test case creation, execution, test automation tools
10/30/2023	System Testing Start	System testing execution, test report generation
11/07/2023	User Acceptance Testing Start	UAT planning, execution
11/07/2023	Test Reporting and Documentation Complete	Test result documentation, test report, defect report

Table 7: Schedule

3.10 Test Result Review

The purpose of conducting QA is to establish that software is fit to be released to potential customers. It thwarts potential embarrassment if a below-average product is delivered to end users. All intended CogniOpen features shall be checked for accuracy and effectiveness in accomplishing their stated tasks.

Any glitches shall be immediately articulated by test engineers. They are then passed along to the development team for remediation. The programmers express their coding changes back to QE. Together, the two organizations analyze the results. This process repeats itself until a satisfactory conclusion is reached.

Immediacy and impact of a problem are equally important. Those qualities, “Priority” and “Severity”, respectively, are assigned an intensity level. “Low”, “Medium”, and “High” are possible values. The figure below depicts how the two variables are connected.

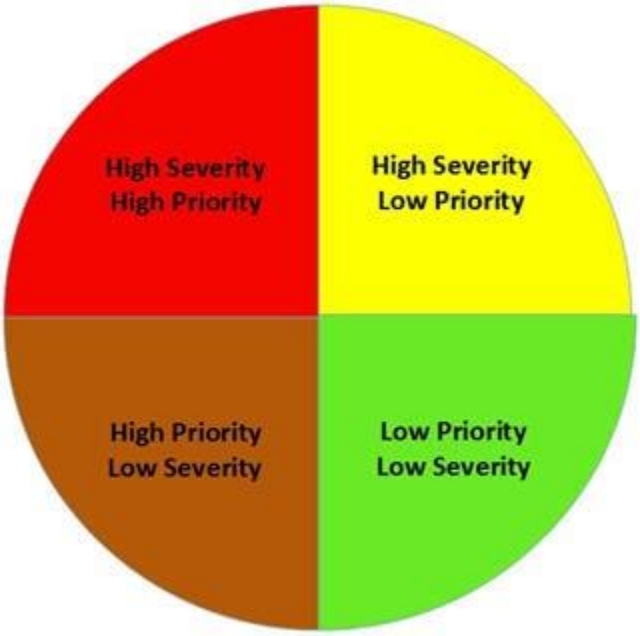

Figure 5: Quality Quadrant

It is most important to fix the errors on the upper left. That is followed by bugs in the upper right and lower left. The team shall address issues on the lower right as time allows.

3.11 Test Results & Defects

Priority and severity are not necessarily related. The former considers time. It defines the schedule when an issue ought to be rectified. Some problems, which are seemingly minor, must be addressed forthwith. An example is a spelling error on the home screen. A different error might cause CogniOpen to occasionally crash. However, it might not be realistic to attempt a repair shortly before a deadline.

The following *priority* levels are available in Microsoft Azure DevOps, and shall be employed by the team.

Priority: 1
The most elevated designation. It is of utmost importance that the component or repair occurs immediately. It is unreasonable to expect that CogniOpen is delivered in such a fulsome state.

Priority: 2
The second level means substantial, but not truly tectonic. The issue must be redressed before the end of the CogniOpen Software Development Life Cycle (SDLC). However, other tasks may supersede it temporarily.

Priority: 3
A bug marked as this level ought to be fixed. Whether it occurs is dependent on availability of the development team and outstanding test assignments.

Priority: 4
Software glitches at this level are purely stylistic in nature. They do not affect the product's ability to accomplish its designated computing chores. Rather, possible abatement would clarify ambiguities and streamline CogniOpen.

Severity is a qualitative designation of the effect of a coding mistake on the application as a whole. Microsoft Azure DevOps defines four *severity* levels, ranging from cataclysmic to trifling.

Severity: 1 – Critical
It is imperative to repair these problems. These types of errors may yield premature termination of CogniOpen. They could ruinate data stores and expose severe security vulnerabilities. There is no viable workaround to be discovered in other buttons or commands.

Severity: 2 – High
This bug is painful for users to endure, but not a showstopper. It represents a noteworthy weakness in CogniOpen. But with guidance and/or technical support, the user may employ an equivalent course of action.

Severity: 3 – Medium
This is a significant blotch. It may cause incorrect output and confusion. The operator could, however, find a comparable method to obtain the desired information.

Severity: 4 – Low
Definitely a weakness. But it is readily ignored or understood for its intended purpose.

3.12 Test Entry & Exit Criteria

An experiment may begin once all parties involved are synchronized in their expectations of what it might produce.

1. Test case particulars are fully detailed and checked for accuracy.
2. Management/customers have endorsed the test case.
3. Developers have acquiesced that a software feature is set for evaluation.
4. CogniOpen has been installed on a device with access to remote services.
5. Physical environment is staged—for example, an object to be recognized is in view.

An experiment ends once all steps are taken, and mitigation has been proffered for the outstanding issues. This could include a written plan of action or merely a full account of the problem.

3.13 Test Suspension & Resumption Criteria

On occasion, an experiment may be temporarily put on hold. This occurs if a situation presents itself that necessitates in-depth discourse between QA staff and the development team. Merely adding notes to digitized web forms is insufficient.

1. Catastrophic failure of CogniOpen, the underlying operating system (OS), or hardware.
2. Product definition/feature set mutates and renders the test case moot.
3. External distractions/responsibilities forestall a team member from completing his/her/their assignment.

The trial recommences once the obstacle is overcome.

4 Test Cases

Each test case will define the associated SRS requirement being tested, identified by the requirement number. In addition, the expected output or result will be defined. As the tests are executed if the expected output or result is achieved the test is said to be Passing. If the expected output or result is not achieved the test is said to be Failing. This portion of the document will be updated as needed to incorporate new unit tests as well as any changes in functional or non-functional requirements as documented within the SRS.

Prior to delivery of the application to the customer, all test cases are expected to be Passing. Results of the test cases can be found within the Test Report.

4.1 Unit Tests

Quality Engineering (QE) is one of the most essential facets of software development. It builds confidence within an organization for a product under construction. QE emboldens sales personnel to aggressively champion their company's labor. And customers prefer to pay for something that has been tested.

Unit tests establish baseline code functionality. The team writing them is knowledgeable of the internal workings of the focus program. Tests of this class circumvent Flutter and Dart graphical controls. Instead, the modules invoke business logic routines directly. Focusing on the littlest external interfaces facilitates pinpoint identification of trouble areas. That, in turn, enables efficient and effective debugging by computer scientists. Continuous unit testing discovers if new features ruinate frangible existing code. Those errors may then be expeditiously corrected.

This portion of the document will be populated with test cases as the application is developed. Unit tests shall be documented with the following template.

<Test Case #>

Test case name:
Method being tested:
Short Description:
Input Data to constructor and/or method you are testing:
Expected Results:

4.1.1 Address Class

4.1.1.1 Test Case U-1-1

Test case name: Obtain Current Address
Method being tested: whereIAm()
Short Description: Function that determines the address of the phone operator at the moment.

Input Data to constructor and/or method you are testing: During actual runs, the phone's Global Positioning System (GPS) receiver interacts with a remote web service to obtain the address. Those capabilities are imitated as such.

```
Position(
  latitude: -77.1517459,
  longitude: 39.0903002,
  timestamp: DateTime.fromMillisecondsSinceEpoch(0).toUtc(),
  altitude: 0.0,
  altitudeAccuracy: 0.0,
  accuracy: 0.0,
  heading: 0.0,
  headingAccuracy: 0.0,
  speed: 0.0,
  speedAccuracy: 0.0
);
Placemark(
  administrativeArea: 'Maryland',
  country: 'United States',
  isoCountryCode: 'US',
  locality: 'Rockville',
  name: '501',
  postalCode: '20850',
  street: '501 Hungerford Dr',
  subAdministrativeArea: 'Montgomery County',
  subLocality: '',
  subThoroughfare: '501',
  thoroughfare: 'Hungerford Drive'
);
```

Expected Results: Function returns "501 Hungerford Dr, Rockville, 20850, United States"

4.1.2 Audio Class

The following values were used when referenced below :

```
const int id = 1;
const String title = 'Test Title';
const String description = 'Test Description';
const List<String> tags = ['Tag1', 'Tag2'];
final DateTime timestamp = DateTime.now();
const int storageSize = 1000;
const bool isFavorited = true;
const String audioFileName = 'test_audio.mp3';
const String summary = 'Test Summary';
```

4.1.2.1 Test Case U-2-1

Test case name: Create an Audio object with all parameters

Method being tested: Audio() constructor
Short Description: Constructor used to create an Audio object
Input Data to constructor and/or method you are testing:
final Audio audio = Audio(
id: id,
title: title,
description: description,
tags: tags,
timestamp: timestamp,
storageSize: storageSize,
isFavorited: isFavorited,
audioFileName: audioFileName,
summary: summary,
);
Expected Results: Audio object created and values initialized correctly:
expect(audio.id, id);
expect(audio.mediaType, MediaType.audio);
expect(audio.title, title);
expect(audio.description, description);
expect(audio.tags, tags);
expect(audio.timestamp, timestamp);
expect(audio.storageSize, storageSize);
expect(audio.isFavorited, isFavorited);
expect(audio.audioFileName, audioFileName);
expect(audio.summary, summary);

4.1.2.2 Test Case U-2-2

Test case name: Create an Audio object with only required parameters
Method being tested: Audio() constructor
Short Description: Constructor used to create an Audio object
Input Data to constructor and/or method you are testing:
final Audio audio = Audio(
id: id,
title: title,
timestamp: timestamp,
storageSize: storageSize,
isFavorited: isFavorited,
audioFileName: audioFileName,
);
Expected Results: Audio object created and values initialized correctly:
expect(audio.id, id);
expect(audio.mediaType, MediaType.audio);
expect(audio.title, title);
expect(audio.description, isNull);

expect(audio.tags, isNull);
expect(audio.timestamp, timestamp);
expect(audio.storageSize, storageSize);
expect(audio.isFavorited, isFavorited);
expect(audio.audioFileName, audioFileName);
expect(audio.summary, isNull);

4.1.2.3 Test Case U-2-3

Test case name: Create an Audio object from JSON with all parameters
Method being tested: fromJson()
Short Description: method used to create an Audio object from JSON
Input Data to constructor and/or method you are testing:
final Map<String, Object?> json = {
MediaFields.id: id,
MediaFields.title: title,
MediaFields.description: description,
MediaFields.tags: tags.join(','),
MediaFields.timestamp: timestamp.toUtc().millisecondsSinceEpoch,
MediaFields.storageSize: storageSize,
MediaFields.isFavorited: 1,
AudioFields.audioFileName: audioFileName,
AudioFields.summary: summary,
};
Expected Results: Audio object created and values initialized correctly:
expect(audio.id, id);
expect(audio.mediaType, MediaType.audio);
expect(audio.title, title);
expect(audio.description, description);
expect(audio.tags, tags);
expect(
audio.timestamp,
DateTime.fromMillisecondsSinceEpoch(
timestamp.toUtc().millisecondsSinceEpoch,
isUtc: true));
expect(audio.storageSize, storageSize);
expect(audio.isFavorited, isFavorited);
expect(audio.audioFileName, audioFileName);
expect(audio.summary, summary);

4.1.2.4 Test Case U-2-4

Test case name: Create an Audio object from JSON with non-nullable fields
Method being tested: fromJson()
Short Description: method used to create an Audio object from JSON

Input Data to constructor and/or method you are testing:
final Map<String, Object?> json = {
MediaFields.id: id,
MediaFields.title: title,
MediaFields.timestamp: timestamp.toUtc().millisecondsSinceEpoch,
MediaFields.storageSize: storageSize,
MediaFields.isFavorited: 1,
AudioFields.audioFileName: audioFileName,
};
Expected Results: Audio object created and values initialized correctly:
expect(audio.id, id);
expect(audio.mediaType, MediaType.audio);
expect(audio.title, title);
expect(audio.description, isNull);
expect(audio.tags, isNull);
expect(
audio.timestamp,
DateTime.fromMillisecondsSinceEpoch(
timestamp.toUtc().millisecondsSinceEpoch,
isUtc: true));
expect(audio.storageSize, storageSize);
expect(audio.isFavorited, isFavorited);
expect(audio.audioFileName, audioFileName);
expect(audio.summary, isNull);

4.1.2.5 Test Case U-2-5

Test case name: fromJSON should throw a FormatException when given invalid JSON
Method being tested: fromJson()
Short Description: method used to create an Audio object from JSON
Input Data to constructor and/or method you are testing: empty JSON
Expected Results: FormatException thrown

4.1.2.6 Test Case U-2-6

Test case name: Audio.toJson should serialize an Audio object with all field values
Method being tested: toJson()
Short Description: method used to create JSON from an Audio object
Input Data to constructor and/or method you are testing:
final Audio audio = Audio(
id: id,
title: title,
description: description,
tags: tags,
timestamp: timestamp,

storageSize: storageSize,
isFavorited: isFavorited,
audioFileName: audioFileName,
summary: summary,
);
Expected Results: JSON created and values set correctly:
expect(json, {
MediaFields.id: id,
MediaFields.title: title,
MediaFields.description: description,
MediaFields.tags: tags.join(','),
MediaFields.timestamp: timestamp.toUtc().millisecondsSinceEpoch,
MediaFields.storageSize: storageSize,
MediaFields.isFavorited: 1,
AudioFields.audioFileName: audioFileName,
AudioFields.summary: summary,
});

4.1.2.7 Test Case U-2-7

Test case name: Audio.toJson should serialize an Audio object with only non-nullable field values
Method being tested: toJson()
Short Description: method used to create JSON from an Audio object
Input Data to constructor and/or method you are testing:
final Audio audio = Audio(
id: id,
title: title,
timestamp: timestamp,
storageSize: storageSize,
isFavorited: isFavorited,
audioFileName: audioFileName,
);
Expected Results: JSON created and values set correctly:
expect(json, {
MediaFields.id: id,
MediaFields.title: title,
MediaFields.description: null,
MediaFields.tags: null,
MediaFields.timestamp: timestamp.toUtc().millisecondsSinceEpoch,
MediaFields.storageSize: storageSize,
MediaFields.isFavorited: 1,
AudioFields.audioFileName: audioFileName,
AudioFields.summary: null,
});

4.1.3 CameraManager Class

4.1.3.1 Test Case U-3-1

Test case name: when initializeCameras gets called, then camera and controller get initialized
Method being tested: initializeCamera()
Short Description: used to initialize camera(s) and controller
Input Data to constructor and/or method you are testing: N/A
Expected Results: one camera is created and the controller is also instantiated (not null)

4.1.3.1 Test Case U-3-2

Test case name: when parseEnvironment gets called, then CameraManager values get set
Method being tested: pareEnvironmentSettings()
Short Description: used to initialize autoRecordingInterval and autoRecording
Input Data to constructor and/or method you are testing: N/A
Expected Results: autoRecordingInterval = 60, isAutoRecording = false

4.1.4 Media Class

4.1.4.1 Test Case U-4-1

Test case name: MediaFields.values should contain the correct field names
Method being tested: values
Short Description: list of strings associated with every Media object
Input Data to constructor and/or method you are testing: MediaFields object
Expected Results: contains the correct fields as defined below:

```
expect(MediaFields.values, isA<List<String>>());
expect(MediaFields.values, hasLength(7));
expect(MediaFields.values, contains(MediaFields.id));
expect(MediaFields.values, contains(MediaFields.title));
expect(MediaFields.values, contains(MediaFields.description));
expect(MediaFields.values, contains(MediaFields.tags));
expect(MediaFields.values, contains(MediaFields.timestamp));
expect(MediaFields.values, contains(MediaFields.storageSize));
expect(MediaFields.values, contains(MediaFields.isFavorited));
```

4.1.4.2 Test Case U-4-2

Test case name: MediaFields values should be the correct field name
Method being tested: N/A
Short Description: values populated in MediaFields should have the correct string name
Input Data to constructor and/or method you are testing: MediaFiefield names are set correctly as shown below:

```
expect(MediaFields.id, '_id');
```

expect(MediaFields.title, 'title');
expect(MediaFields.description, 'description');
expect(MediaFields.tags, 'tags');
expect(MediaFields.timestamp, 'timestamp');
expect(MediaFields.storageSize, 'storage_size');
expect(MediaFields.isFavorited, 'is_favorited');

4.1.5 Photo Class

The following variables are defined and referenced in the test cases below:

```
const int id = 1;
const String title = 'Test Title';
const String description = 'Test Description';
const List<String> tags = ['Tag1', 'Tag2'];
final DateTime timestamp = DateTime.now();
const int storageSize = 1000;
const bool isFavorited = true;
const String photoFileName = 'test_photo.jpg';
```

4.1.5.1 Test Case U-5-1

Test case name: Create a photo, with all parameters
Method being tested: Photo()
Short Description: Constructor used to create a Photo which extends the Media class
Input Data to constructor and/or method you are testing:

```
final Photo photo = Photo(
  id: id,
  title: title,
  description: description,
  tags: tags,
  timestamp: timestamp,
  storageSize: storageSize,
  isFavorited: isFavorited,
  photoFileName: photoFileName,
 );
```

Expected Results: photo object is created and has all values initialized correctly

```
expect(photo.id, id);
expect(photo.mediaType, MediaType.photo);
expect(photo.title, title);
expect(photo.description, description);
expect(photo.tags, tags);
expect(photo.timestamp, timestamp);
expect(photo.storageSize, storageSize);
expect(photo.isFavorited, isFavorited);
expect(photo.photoFileName, photoFileName);
```

4.1.5.2 Test Case U-5-2

Test case name: Create a photo, with only required parameters
Method being tested: Photo()
Short Description: Constructor used to create a Photo which extends the Media class
Input Data to constructor and/or method you are testing:

```
final Photo photo = Photo(
   id: id,
   title: title,
   timestamp: timestamp,
   storageSize: storageSize,
   isFavorited: isFavorited,
   photoFileName: photoFileName,
  );
```

Expected Results: photo object is created and has all values initialized correctly

```
        expect(photo.id, id);
          expect(photo.mediaType, MediaType.photo);
          expect(photo.title, title);
          expect(photo.description, isNull);
          expect(photo.tags, isNull);
          expect(photo.timestamp, timestamp);
          expect(photo.storageSize, storageSize);
          expect(photo.isFavorited, isFavorited);
          expect(photo.photoFileName, photoFileName);
```

4.1.5.3 Test Case U-5-3

Test case name: Create a photo from JSON, with all field values
Method being tested: fromJson()
Short Description: used to create a Photo object from JSON
Input Data to constructor and/or method you are testing:

```
final Map<String, Object?> json = {
   MediaFields.id: id,
   MediaFields.title: title,
   MediaFields.description: description,
   MediaFields.tags: tags.join(','),
   MediaFields.timestamp: timestamp.toUtc().millisecondsSinceEpoch,
   MediaFields.storageSize: storageSize,
   MediaFields.isFavorited: 1,
   PhotoFields.photoFileName: photoFileName,
  };
```

Expected Results: photo object is created and has all values initialized correctly

```
          expect(photo.id, id);
          expect(photo.mediaType, MediaType.photo);
```

```
expect(photo.title, title);
expect(photo.description, description);
expect(photo.tags, tags);
expect(
  photo.timestamp,
  DateTime.fromMillisecondsSinceEpoch(
    timestamp.toUtc().millisecondsSinceEpoch,
    isUtc: true));
expect(photo.storageSize, storageSize);
expect(photo.isFavorited, isFavorited);
expect(photo.photoFileName, photoFileName);
```

4.1.5.4 Test Case U-5-4

Test case name: Create a photo from JSON, with only non-nullable field values
Method being tested: fromJson()
Short Description: used to create a Photo object from JSON
Input Data to constructor and/or method you are testing:

```
final Map<String, Object?> json = {
  MediaFields.id: id,
  MediaFields.title: title,
  MediaFields.timestamp: timestamp.toUtc().millisecondsSinceEpoch,
  MediaFields.storageSize: storageSize,
  MediaFields.isFavorited: 1,
  PhotoFields.photoFileName: photoFileName,
 };
```

Expected Results: photo object is created and has all values initialized correctly

```
expect(photo.id, id);
expect(photo.mediaType, MediaType.photo);
expect(photo.title, title);
expect(photo.description, isNull);
expect(photo.tags, isNull);
expect(
  photo.timestamp,
  DateTime.fromMillisecondsSinceEpoch(
    timestamp.toUtc().millisecondsSinceEpoch,
    isUtc: true));
expect(photo.storageSize, storageSize);
expect(photo.isFavorited, isFavorited);
expect(photo.photoFileName, photoFileName);
```

4.1.5.5 Test Case U-5-5

Test case name: throw FormatException if trying to create a photo object with invalid JSON
Method being tested: fromJson()
Short Description: used to create a Photo object from JSON
Input Data to constructor and/or method you are testing: empty JSON
Expected Results: FormatException is thrown

4.1.5.6 Test Case U-5-6

Test case name: serialize a photo object to JSON, with all field values
Method being tested: toJson()
Short Description: used to create JSON from a photo object
Input Data to constructor and/or method you are testing:

```
final Photo photo = Photo(
   id: id,
   title: title,
   description: description,
   tags: tags,
   timestamp: timestamp,
   storageSize: storageSize,
   isFavorited: isFavorited,
   photoFileName: photoFileName,
  );
```

Expected Results: JSON created and values set correctly:

```
            expect(json, {
            MediaFields.id: id,
            MediaFields.title: title,
            MediaFields.description: description,
            MediaFields.tags: tags.join(','),
            MediaFields.timestamp: timestamp.toUtc().millisecondsSinceEpoch,
            MediaFields.storageSize: storageSize,
            MediaFields.isFavorited: 1,
            PhotoFields.photoFileName: photoFileName,
           })
```

4.1.5.7 Test Case U-5-7

Test case name: serialize a photo object to JSON, with only non-nullable field values
Method being tested: toJson()
Short Description: used to create JSON from a photo object
Input Data to constructor and/or method you are testing:

```
 final Photo photo = Photo(
```

id: id,
title: title,
timestamp: timestamp,
storageSize: storageSize,
isFavorited: isFavorited,
photoFileName: photoFileName,
);

Expected Results: JSON created and values set correctly:

expect(json, {
MediaFields.id: id,
MediaFields.title: title,
MediaFields.description: null,
MediaFields.tags: null,
MediaFields.timestamp: timestamp.toUtc().millisecondsSinceEpoch,
MediaFields.storageSize: storageSize,
MediaFields.isFavorited: 1,
PhotoFields.photoFileName: photoFileName,
});

4.1.6 S3 Connection Class

4.1.6.1 Test Case U-6-1

Test case name: add audio to S3 bucket
Method being tested: addAudioToS3
Short Description: test ability to addAudioToS3 and return string of audio title
Input Data to constructor and/or method you are testing: String audioName = 'testAudio', String pathToFile= '/assets/test_images/Sea waves.mp4'
Expected Results: the method should return the string 'testAudio'

4.1.6.2 Test Case U-6-2

Test case name: add video to S3 bucket
Method being tested: addVideoToS3
Short Description: test ability to addVideoToS3 and return string of video title
Input Data to constructor and/or method you are testing: String videoName = 'testVideo', String pathToFile= '/assets/test_images/1MinuteSampleVideo.mp4'
Expected Results: the method should return the string 'testVideo'

4.1.7 Settings Class

4.1.7.1 Test Case U-7-1

Test case name: testing Settings
Method being tested: Settings() constructor
Short Description: testing constructor for Settings

Input Data to constructor and/or method you are testing: N/A
Expected Results: Settings class locationEnabled is set to false

4.1.7.2 Test Case U-7-2

Test case name: testing Notification
Method being tested: Notification() constructor
Short Description: testing constructor for Notification
Input Data to constructor and/or method you are testing: String notificationID, String name and boolean isDisplayed as displayed below
String notificationID = '1'
String name = 'Notification1'
Bool isDisplayed = true
Expected Results: Notification created with notificationID = '1', name = 'Notification1', and isDisplayed = true

4.1.8 Significant Object Class

4.1.8.1 Test Case U-8-1

Test case name: testing deleteAlternateName
Method being tested: deleteAlternateName
Short Description: testing the ability to delete an alternate name from a SignificantObject
Input Data to constructor and/or method you are testing: String alternateName = 'black glasses'
Expected Results: alternateNames.length is 2 before the method is called, afterwards the length is 1

4.1.8.2 Test Case U-8-2

Test case name: testing addAlternateName
Method being tested: addAlternateName
Short Description: testing the ability to add an alternate name from a SignificantObject
Input Data to constructor and/or method you are testing: String alternateName = 'black glasses'
Expected Results: alternateNames.length is 1 before the method is called, afterwards the length is 2

4.1.9 Video Class

The following constants were defined and used when referenced below in the test cases:

```
const int id = 1;
const String title = 'Test Title';
const String description = 'Test Description';
const List<String> tags = ['Tag1', 'Tag2'];
final DateTime timestamp = DateTime.now();
```

```
const int storageSize = 1000;
const bool isFavorited = true;
const String videoFileName = 'test_video.mp4';
const String thumbnailFileName = 'test_thumbnail.jpg';
const String duration = '00:02:30';
```

4.1.9.1 Test Case U-9-1

Test case name: Create a video with all parameters provided
Method being tested: Video()
Short Description: Constructor used to create a Video which extends the Media class
Input Data to constructor and/or method you are testing:

```
Video video = Video(
  id: id,
  title: title,
  description: description,
  tags: tags,
  timestamp: timestamp,
  storageSize: storageSize,
  isFavorited: isFavorited,
  videoFileName: videoFileName,
  thumbnailFileName: thumbnailFileName,
  duration: duration,
 );
```

Expected Results: Video object created and values initialized correctly.

4.1.9.2 Test Case U-9-2

Test case name: Create a video with only required parameters provided
Method being tested: Video
Short Description: Constructor used to create a Video which extends the Media class
Input Data to constructor and/or method you are testing:

```
Video(
        id: id,
        title: title,
         timestamp: timestamp,
         storageSize: storageSize,
        isFavorited: isFavorited,
       videoFileName: videoFileName,
        duration: duration,
 );
```

Expected Results: Video object created and values initialized correctly.

4.1.9.3 Test Case U-9-3

Test case name: Video.fromJson should correctly create a Video object from JSON with all fields
Method being tested: fromJSON()
Short Description: Used to create a Video which extends the Media class
Input Data to constructor and/or method you are testing: json string, see below

```
        json = {
   MediaFields.id: id,
   MediaFields.title: title,
   MediaFields.description: description,
   MediaFields.tags: tags.join(','),
   MediaFields.timestamp: timestamp.toUtc().millisecondsSinceEpoch,
   MediaFields.storageSize: storageSize,
   MediaFields.isFavorited: 1,
   VideoFields.videoFileName: videoFileName,
   VideoFields.thumbnailFileName: thumbnailFileName,
   VideoFields.duration: duration,
  };
```

Expected Results: Video object created and values initialized correctly.

4.1.9.4 Test Case U-9-4

Test case name: Video.fromJson should correctly create a Video object from JSON with non-nullable field values only
Method being tested: fromJSON()
Short Description: Used to create a Video which extends the Media class
Input Data to constructor and/or method you are testing: json string, see below

```
        json = {
    MediaFields.id: id,
    MediaFields.title: title,
    MediaFields.timestamp: timestamp.toUtc().millisecondsSinceEpoch,
    MediaFields.storageSize: storageSize,
    MediaFields.isFavorited: 1,
    VideoFields.videoFileName: videoFileName,
    VideoFields.duration: duration,
   };
```

Expected Results: Video object created and values initialized correctly.

4.1.9.5 Test Case U-9-5

Test case name: Video.fromJson should throw a FormatException when given invalid JSON
Method being tested: fromJSON()

Short Description: Used to create a Video which extends the Media class
Input Data to constructor and/or method you are testing: empty JSON string
Expected Results: FormatException thrown

4.1.9.6 Test Case U-9-6

Test case name: Video.toJson should correctly serialize a Video object with all field values
Method being tested: toJSON()
Short Description: Used to serialize a Video object
Input Data to constructor and/or method you are testing:

```
final Video video = Video(
   id: id,
   title: title,
   description: description,
   tags: tags,
   timestamp: timestamp,
   storageSize: storageSize,
   isFavorited: isFavorited,
   videoFileName: videoFileName,
   thumbnailFileName: thumbnailFileName,
   duration: duration,
  );
```

Expected Results: json object created with all fields populated correctly

```
   MediaFields.id: id,
   MediaFields.title: title,
   MediaFields.description: description,
   MediaFields.tags: tags.join(','),
   MediaFields.timestamp: timestamp.toUtc().millisecondsSinceEpoch,
   MediaFields.storageSize: storageSize,
   MediaFields.isFavorited: 1,
   VideoFields.videoFileName: videoFileName,
   VideoFields.thumbnailFileName: thumbnailFileName,
   VideoFields.duration: duration,
```

4.1.9.7 Test Case U-9-7

Test case name: Video.toJson should correctly serialize a Video object with only non-nullable field values
Method being tested: toJSON()
Short Description: Used to serialize a Video object
Input Data to constructor and/or method you are testing:

```
final Video video = Video(
   id: id,
```

title: title,
timestamp: timestamp,
storageSize: storageSize,
isFavorited: isFavorited,
videoFileName: videoFileName,
duration: duration,
);
Expected Results: json object created with fields populated correctly
expect(json, {
MediaFields.id: id,
MediaFields.title: title,
MediaFields.description: null,
MediaFields.tags: null,
MediaFields.timestamp: timestamp.toUtc().millisecondsSinceEpoch,
MediaFields.storageSize: storageSize,
MediaFields.isFavorited: 1,
VideoFields.videoFileName: videoFileName,
VideoFields.thumbnailFileName: null,
VideoFields.duration: duration,
});

4.1.10 LocationDataModel Class

4.1.10.1 Test Case U-10-1

Test case name: Location constructor create a location object
Method being tested: Constructor of LocationDataModel
Short Description: Test that the constructor for the LocationDataModel class creates a LocationDataModel class with the correct values.
Input Data to constructor and/or method you are testing:
const int id = 1;
const double latitude = 90.51;
const double longitude = 100.35;
const String address = "123 Smith Lane Greenville, CT 12345";
final DateTime timestamp = DateTime.utc(2023, 10, 23);
Expected Results: LocationDataModel created with the following values:
expect(location.id, id);
expect(location.latitude, latitude);
expect(location.longitude, longitude);
expect(location.address, address);
expect(location.timestamp, timestamp);

4.1.10.2 Test Case U-10-2

Test case name: Create map from location
Method being tested: toMap()

Short Description: Test that a LocationDataModel object can be turned into a String, Object Map.
Input Data to constructor and/or method you are testing:
Location DataModel(const int id = 1;
const double latitude = 90.51;
const double longitude = 100.35;
const String address = "123 Smith Lane Greenville, CT 12345";
final DateTime timestamp = DateTime.utc(2023, 10, 23);)
Expected Results: Map created with the following values:
expect(json, {
'id': id,
'latitude': latitude,
'longitude': longitude,
'address': address,
'timestamp': '2023-10-23T00:00:00.000Z',
});

4.1.10.3 Test Case U-10-3

Test case name: Create a LocationDataModel from a map
Method being tested: fromMap()
Short Description: Test that a LocationDataModel object can be create from a String, Object Map.
Input Data to constructor and/or method you are testing: String, Object map as follows:
{
'id': id,
'latitude': latitude,
'longitude': longitude,
'address': address,
'timestamp': '2023-10-23T00:00:00.000Z',
}
Expected Results: LocationDataModel created with the following values:
const int id = 1;
const double latitude = 90.51;
const double longitude = 100.35;
const String address = "123 Smith Lane Greenville, CT 12345";
final DateTime timestamp = DateTime.utc(2023, 10, 23);

4.1.11 SignificantObject Data Model Class

4.1.11.1 Test Case U-11-1

Test case name: SignificantObject constructor create a significant object model
Method being tested: SignificantObject constructor
Short Description: Test that the constructor for the SignificantObject class creates a SignificantObject object with the correct values.

Input Data to constructor and/or method you are testing:

```
const int id = 1;
const String objectLabel = 'cat';
const String customLabel = 'my cat';
final int timestamp = DateTime.utc(2023, 10, 23).millisecondsSinceEpoch;
const String imageFileName = 'cat.png';
const double left = 10.0;
const double top = 11.0;
const double width = 150.5;
const double height = 300.0;
```

Expected Results: object created with the following values:

```
expect(so.id, id);
expect(so.objectLabel, objectLabel);
expect(so.timestamp, timestamp);
expect(so.imageFileName, imageFileName);
expect(so.left, left);
expect(so.top, top);
expect(so.width, width);
expect(so.height, height);
```

4.1.11.2 Test Case U-11-2

Test case name: Create a SignificantObject object from JSON
Method being tested: fromJson()
Short Description: Test that a SignificantObject object can be created from JSON
Input Data to constructor and/or method you are testing:

```
  {
  SignificantObjectFields.id: id,
  SignificantObjectFields.objectLabel: objectLabel,
  SignificantObjectFields.timestamp: timestamp,
  SignificantObjectFields.imageFileName: imageFileName,
  SignificantObjectFields.left: left,
  SignificantObjectFields.top: top,
  SignificantObjectFields.width: width,
  SignificantObjectFields.height: height,
 };
```

Expected Results: object created with the following values:

```
expect(so.id, id);
expect(so.objectLabel, objectLabel);
expect(so.timestamp, timestamp);
expect(so.imageFileName, imageFileName);
expect(so.left, left);
expect(so.top, top);
expect(so.width, width);
expect(so.height, height);
```

4.1.11.3 Test Case U-11-3

Test case name: Create JSON from a SignificantObject object
Method being tested: toJson()
Short Description: Test that a SignificantObject object can be used to create JSON
Input Data to constructor and/or method you are testing:
Existing SignificantObject with the following values:
const int id = 1;
const String objectLabel = 'cat';
const String customLabel = 'my cat';
final int timestamp = DateTime.utc(2023, 10, 23).millisecondsSinceEpoch;
const String imageFileName = 'cat.png';
const double left = 10.0;
const double top = 11.0;
const double width = 150.5;
const double height = 300.0;
Expected Results: JSON created:
{
SignificantObjectFields.id: id,
SignificantObjectFields.objectLabel: objectLabel,
SignificantObjectFields.timestamp: timestamp,
SignificantObjectFields.imageFileName: imageFileName,
SignificantObjectFields.left: left,
SignificantObjectFields.top: top,
SignificantObjectFields.width: width,
SignificantObjectFields.height: height,
};

4.1.13 App Database Class

4.1.13.1 Test Case U-13-1

Test case name: Adding Audio to DB
Method being tested: AudioRepository.create(Audio)
Short Description: Creates a database record of an audio taping
Input Data to constructor and/or method you are testing:
Audio(
title: "Unit Test Audio",
description: 'Unit Test Audio Description',
tags: ['almond', 'cashew', 'raisin'],
timestamp: DateTime(2023, 10, 20, 8, 26),
fileName: 'unit_test_audio.mp4',
storageSize: 1000000,
isFavorited: false,
summary: "Unit Test Audio Summary"
);
Expected Results: Row created in ‘audios’ table

4.1.13.2 Test Case U-13-2

Test case name: Adding Photo to DB
Method being tested: PhotoRepository.create(Photo)
Short Description: Creates a database record of a photograph
Input Data to constructor and/or method you are testing:

```
Photo(
  title: "Unit Test Photo",
  description: 'Unit Test Photo Description',
  tags: ['sun', 'moon', 'star'],
  timestamp: DateTime(2023, 10, 20, 8, 26),
  fileName: 'unit_test_photo.png',
  storageSize: 1000000,
  isFavorited: false
);
```

Expected Results: Row created in 'photos' table

4.1.13.3 Test Case U-13-3

Test case name: Adding Video to DB
Method being tested: VideoRepository.create(Video)
Short Description: Creates database record of a video
Input Data to constructor and/or method you are testing:

```
Video(
  title: "Unit Test Video",
  description: 'Unit Test Video Description',
  tags: ['orange', 'banana', 'bear'],
  timestamp: DateTime(2023, 10, 20, 8, 26),
  fileName: 'unit_test_video.mp4',
  storageSize: 1000000,
  isFavorited: false,
  duration: "1:00",
  thumbnail: "unit_test_thumbnail.png"
);
```

Expected Results: Row created in 'videos' table

4.1.15 DirectoryManager Class

4.1.15.1 Test Case U-15-1

Test case name: verify all directories were created
Method being tested: initializeDirectories
Short Description: method which is called when an instance of DirectoryManager is created, which initializes all local data directories
Input Data to constructor and/or method you are testing: N/A
Expected Results: all directories should exist, and be non-null

```
expect(DirectoryManager.instance.rootDirectory, isNot(null));
```

expect(DirectoryManager.instance.photosDirectory, isNot(null));
expect(DirectoryManager.instance.videosDirectory, isNot(null));
expect(DirectoryManager.instance.audiosDirectory, isNot(null));
expect(DirectoryManager.instance.videoStillsDirectory, isNot(null));
expect(DirectoryManager.instance.videoThumbnailsDirectory, isNot(null));
expect(DirectoryManager.instance.tmpDirectory, isNot(null));

4.1.17 FormatUtils Class

4.1.17.1 Test Case U-17-1

Test case name: testing getStorageSizeString KB
Method being tested: getStorageSizeString()
Short Description: method to convert storage size(as an integer) to a readable string format, testing conversion to KB
Input Data to constructor and/or method you are testing: 1024
Expected Results: '1.00 KB'

4.1.17.2 Test Case U-17-2

Test case name: testing getStorageSizeString Bytes
Method being tested: getStorageSizeString()
Short Description: method to convert storage size(as an integer) to a readable string format, testing conversion to Bytes
Input Data to constructor and/or method you are testing: 512
Expected Results: Media storage size is converted to string '512.00 Bytes'

4.1.17.3 Test Case U-17-3

Test case name: testing getStorageSizeString MB
Method being tested: getStorageSizeString()
Short Description: method to convert storage size(as an integer) to a readable string format, testing conversion to MB
Input Data to constructor and/or method you are testing: 10000000
Expected Results: Media storage size is converted to string '9.54 MB'

4.1.17.4 Test Case U-17-4

Test case name: testing getStorageSizeString GB
Method being tested: getStorageSizeString()
Short Description: method to convert storage size(as an integer) to a readable string format, testing conversion to GB
Input Data to constructor and/or method you are testing: 10000000000
Expected Results: Media storage size is converted to string '9.31 GB'

4.1.17.5 Test Case U-17-5

Test case name: testing getDateTimeString valid DateTime

Method being tested: getDateTimeString()
Short Description: method to convert DateTime to a readable string format, testing conversion with a non-null DateTime
Input Data to constructor and/or method you are testing: DateTime of October 5 2023
Expected Results: '2023-10-05 00:00:00'

4.1.17.6 Test Case U-17-6

Test case name: testing getDateTimeString null DateTime
Method being tested: getDateTimeString()
Short Description: method to convert DateTime to a readable string format, testing conversion with a null DateTime
Input Data to constructor and/or method you are testing: DateTime of null
Expected Results: 'N/A'

4.1.17.7 Test Case U-17-7

Test case name: testing getDateString with a valid DateTime
Method being tested: getDateString()
Short Description: method to convert DateTime to a readable string format, testing conversion with a valid DateTime
Input Data to constructor and/or method you are testing: DateTime of October 5, 2023
Expected Results: 'October 5th, 2023 12:00 AM'

4.1.17.8 Test Case U-17-8

Test case name: testing getDateString with a null DateTime
Method being tested: getDateString()
Short Description: method to convert DateTime attribute to a readable string format, testing conversion with a null DateTime
Input Data to constructor and/or method you are testing: DateTime of null
Expected Results: 'N/A'

4.1.17.9 Test Case U-17-9

Test case name: testing calculateDifference using today
Method being tested: caculateDifference
Short Description: method to calculate the difference in full days between the provided date and today
Input Data to constructor and/or method you are testing: DateTime of now()
Expected Results: 0

4.1.18 MediaType Class

4.1.18.1 Test Case U-18-1

Test case name: testing that the correct string and name is populated for an Audio MediaType
Method being tested: MediaType.audio
Short Description: creates a MediaType object of the specified type
Input Data to constructor and/or method you are testing: audio type
Expected Results: string name returns 'MediaType.audio' and name returns 'audio'

4.1.18.2 Test Case U-18-2

Test case name: testing that the correct string and name is populated for a photo MediaType
Method being tested: MediaType.photo
Short Description: creates a MediaType object of the specified type
Input Data to constructor and/or method you are testing: photo type
Expected Results: string name returns 'MediaType.photo' and name returns 'photo'

4.1.18.3 Test Case U-18-3

Test case name: testing that the correct string and name is populated for a video MediaType
Method being tested: MediaType.video
Short Description: creates a MediaType object of the specified type
Input Data to constructor and/or method you are testing: video type
Expected Results: string name returns 'MediaType.video' and name returns 'video'

4.2 Widget Tests

This section of the document provides details on widget testing done for each screen of the application. It looks for buttons, titles and menu items that are expected to always be visible.

4.2.1 Home Screen

4.2.1.1 Test Case W-1

Tests that the applications home page loads correctly, the following widgets are expected to be visible and are tested for:

- Virtual Assistant Elevated Button
- Gallery Elevated Button
- Record Video Button
- Record Audio Button
- Significant Objects Button
- Tour Guide Button

4.2.2 Gallery Screen

4.2.1.1 Test Case W-2

Tests that the applications gallery page loads correctly, the following widgets are expected to be visible and are tested for:

- Back Icon
- Search Icon
- Favorites Icon
- Photo Filter Icon
- Video Filter Icon
- Conversation Filter Icon
- Gallery Menu
- Grid Slider

4.2.3 Video Screen

4.2.1.1 Test Case W-3

Tests that the applications gallery page loads correctly, the following widgets are expected to be visible and are tested for:

- Camera Icon
- Camera Text

4.2.4 Significant Object Screen

4.2.1.1 Test Case W-4

Tests that the applications significant object page loads correctly, the following widgets are expected to be visible and are tested for:

- Camera
- Upload Image
- Significant Object Text

4.2.5 Location History Screen

4.2.1.1 Test Case W-5

Tests that the applications location history page loads correctly, the following widgets are expected to be visible and are tested for:

- Location History Text

4.3 Functional Tests

The following test cases are defined to cover all functional use-cases as defined within Section 3.1 of the SRS document.

4.3.1 Test Case F-01: Decline Camera Permissions

Description: Allows user to disallow the cogniopen application to take pictures and videos.
SRS Requirement: SRS-1
Requirements: The system shall gain consent from users before engaging in private interactions.
Prerequisites: Application has been installed on the smartphone.
Steps:

1. Tap “CogniOpen” icon on device’s home screen
2. Tap “Don’t allow” on the camera permissions dialog

Expected output: Application will save camera permission preference.
Assumptions: User has not already accepted or declined camera permissions.

4.3.2 Test Case F-02: Accept Camera Permissions

Description: Allows user to grant permission to the cogniopen application to take pictures and videos.
SRS Requirement: SRS-1
Requirements: The system shall gain consent from users before engaging in private interactions.
Prerequisites: Application has been installed on the smartphone.
Steps:

1. Tap “CogniOpen” icon on device’s home screen
2. Tap “While using this app” on camera permissions dialog

Expected output: Application will save camera permission preferences.
Assumptions: User has no already accepted camera permissions.

4.3.3 Test Case F-04: Create a Significant Object – Upload a Picture

Description: Creates a significant object that can be used to improve recognition performance.
SRS Requirement: SRS-6
Requirements: Photographs of significant objects are persisted by the system.
Prerequisites: The application’s home page is loaded.
Steps:

1. Tap “Significant Objects” tile
2. Click “Upload Image”
3. Select the image of a pair of glasses.

Expected output: New picture entry for glasses created in Significant Object screen.
Assumptions: CogniOpen has access to the smartphone’s camera roll.

4.3.4 Test Case F-05: Photograph Fire Extinguisher

Description: Takes a new photograph of a fire extinguisher’s location in the house.
SRS Requirement: SRS-6

Requirements: The system shall allow photographs to be taken and referenced by CogniOpen.
Prerequisites: The application's home page is loaded. A fire extinguisher is stored in its usual place.
Steps:
1. Tap "Significant Objects" tile
2. Select "Camera"
3. Point camera at fire extinguisher
4. Click camera icon to take the picture
5. Click the checkmark to accept the picture.

Expected output: New picture entry in Significant Object screen created for fire extinguisher.
Assumptions: CogniOpen granted access to the smartphone's camera.

4.3.5 Test Case F-07: Create Video of Kitchen Contents

Description: Creates a video of a mostly stocked kitchen.
SRS Requirement: SRS-8
Requirements: The system is able to capture and store videography.
Prerequisites: The application is open. The user has walked into the kitchen.
Steps:
1. Pan the camera around the kitchen

Expected output: New video entries in "Gallery" are created.
Assumptions: CogniOpen has been granted access to the phone's camera.

4.3.6 Test Case F-11: Recall Location of Frequently Used Reading Glasses

Description: Shows where the frequently used glasses are kept in the house.
SRS Requirement: SRS-11
Requirements: The system is able to match voice commands to saved imagery.
Prerequisites: The application's home page is loaded. A video showing the actor's routine stored in "My Gallery". Actor using the red reading glasses frequently.
Steps:
1. Tap "Object Search" tile
2. Type 'reading glasses'

Expected output: A screenshot of the video with the red reading glasses last seen is displayed on the screen. An option will pop up if the actor wants this item to be a "Significant Object".
Assumptions: CogniOpen has been set up with AWS to perform object detection.

4.3.7 Test Case F-12: Recap of Actor's Location

Description: The application aids memory-impaired individuals with recollecting events and locations previously visited.
SRS Requirement: SRS-7
Requirements: The system is able to track location data and parse visits.

Prerequisites: The actor has location services enabled and requests the assistant to recap any visits made on a specific date.
Steps:
1. The actor visits the library for 2 hours
2. The actor then goes to a doctor's appointment for 1 hour
3. The actor stops at a restaurant for 45 minutes
4. The actor asks, "What did I do today"? or "Where did I go today"?

Expected output: The application gives a detailed recap of the actor's whereabouts for the date in question.
Assumptions: Location services were enabled for the user during each stop at each location.

4.3.8 Test Case F-15: Relive Previous Discussion with Female

Description: Previously occurring conversation with a woman are saved.
SRS Requirement: SRS-4
Requirements: The system stores audio recording details across sessions.
Prerequisites: The application's home page is loaded. A conversation with a female test subject has been saved to CogniOpen. In it, the woman declares her favorite color.
Steps:
1. Tap "Gallery" tile
2. System displays a list of audio, video and photo recordings.
3. User taps the audio item of interest.

Expected output: System displays a conversation summary. The contents include, "Pratibha", "Favorite Color", and "Blue".
Assumptions: Conversation with Pratibha has not been deleted.

4.3.9 Test Case F-16: Relive Previous Discussion with Male

Description: Previously occurring conversation with a man are stored.
SRS Requirement: SRS-4
Requirements: The system stores audio recording details across sessions.
Prerequisites: The application's home page is loaded. A conversation with a male test subject has been saved to CogniOpen. In it, the man declares his favorite color.
Steps:
1. Tap "Gallery" tile.
2. System displays a list of audio, video and photo recordings
3. User taps the audio item of interest.

Expected output: System displays a conversation summary. The contents include, "Raj", "Favorite Color", and "Green".
Assumptions: Raj's conversation has not been removed.

4.3.10 Test Case F-17: Add Middle Name to Individual

Description: Adds a person's middle name to conversation information.
SRS Requirement: SRS-5

Requirements: System shall allow conversation metadata to be changed.
Prerequisites: The application's home page is loaded. A conversation with a male test subject has been saved to CogniOpen. In it, the man has identified himself by his first name, only.
Steps:

1. Tap "Gallery" tile
2. System displays a list of audio, video and photo recordings
3. Tap second list entry
4. Change the subject's name from "Raj" to "Raj Pal"
5. Tap "Close"

Expected output: Conversation summary has modified subject's name to "Raj Pal".
Assumptions: CogniOpen has been granted access to the device's microphone.

4.3.11 Test Case F-22: Pause Video Recording

Description: Verifies the application's ability to pause video recording.
SRS Requirements: SRS-9
Requirements: The system shall allow users to pause video recording.
Prerequisites: The application's home page is loaded.
Steps:

1. Tap the recording icon on bottom of the home page.
2. Click the red pause button.

Expected output: The application says "Video Recording paused" and stops recording.
Assumptions: CogniOpen has been granted access to the device's camera and passive video recording mode is initially disabled.

4.3.12 Test Case F-23: Un-Pause Video Recording

Description: Verifies the application's ability to un pause video recording.
SRS Requirements: SRS-9
Requirements: The system shall allow users to un pause video recording mode.
Prerequisites: The application's home page is loaded. Video recording is currently paused.
Steps:

1. Tap the recording icon on the bottom of the home page
2. Click the red recording button.

Expected output: The application says "Video recording resumed", a red square is displayed around the video and video recording is resumed.
Assumptions: CogniOpen has been granted access to the device's camera and passive video recording mode is initially enabled.

4.3.13 Test Case F-30: Locate Item (Ambiguous Item)

Description: Verify that the system can distinguish ambiguous items from each other.
SRS Requirement: SRS-11

Requirements: The system shall all the user to view all items they are trying to locate when there are multiple similar objects.
Prerequisites: The application's home page is loaded. The user has recorded video and pictures involving multiple of the same item.
Steps:
1. Tap "Object Search" tile
2. Type 'glasses'
3. The system displays eye glasses and water glass separately.
4. The user selects the appropriate item (eye glasses).

Expected output: A screenshot of the video with the red reading glasses last seen is displayed on the screen.
Assumptions: CogniOpen has been granted access to the device's microphone.

4.3.14 Test Case F- 35: View Picture for existing Media Gallery Item

Description: Verify that the system allows the user to view an existing picture within the Media Gallery.
SRS Requirement: SRS-14
Requirements: The system shall allow the user to view existing pictures in the Media Gallery.
Prerequisites: The application's home page is loaded.
Steps:
1. Tap "Gallery".
2. Select picture to view.

Expected output: The picture and details are visible to the user.
Assumptions: The user has uploaded a picture to the media gallery already with details to view.

4.3.15 Test Case F- 36: View Video for existing Media Gallery Item

Description: Verify that the system allows the user to view an existing video within the Media Gallery.
SRS Requirement: SRS-14
Requirements: The system shall allow the user to view existing videos in the Media Gallery.
Prerequisites: The application's home page is loaded.
Steps:
1. Tap "Gallery".
2. Select video to view.

Expected output: The video and details are visible to the user.
Assumptions: The user has uploaded a video to the media gallery already with details to view.

4.3.16 Test Case F-37: Find Single Instance of Significant Object

Description: Shows where fire extinguisher is in the house.

SRS Requirement: SRS-11
Requirements: The system is able to match voice commands to saved imagery.
Prerequisites: The application's home page is loaded. A photograph showing the fire extinguisher in its normal location was added to "Significant Objects". One video has been saved with the fire extinguisher.
Steps:
1. Tap "Object Search" tile
2. Type 'fire extinguisher?'

Expected output: One fire extinguisher image is shown.
Assumptions: CogniOpen has been granted access to the device's microphone.

4.3.17 Test Case F-38: Provide Error Message if Searching for a Non-added Significant Object

Description: Tests that CogniOpen does not misidentify significant objects.
SRS Requirement: SRS-11
Requirements: The system is able to match voice commands to saved imagery.
Prerequisites: The application's home page is loaded. A photograph showing the fire extinguisher in its normal location was added to "Significant Objects".
Steps:
1. Tap "Ask Question" tile
2. Speak 'Where is my hammer?'

Expected output: Message saying the system does not have that information.
Assumptions: CogniOpen has been granted access to the device's microphone.

4.3.18 Test Case F-39: Find Multiple Instances of a Significant Object

Description: Tests that CogniOpen displays the most recent video containing a significant object.
SRS Requirement: SRS-11
Requirements: The system is able to match voice commands to saved imagery.
Prerequisites: The application's home page is loaded. A photograph showing the fire extinguisher in its normal location was added to "Significant Objects". Two videos with the fire extinguisher have been saved.
Steps:
1. Tap "Ask Question" tile
2. Speak 'Where is my fire extinguisher?'

Expected output: An image of the fire extinguisher in its most recent location is shown.
Assumptions: CogniOpen has been granted access to the device's microphone.

4.3.19 Test Case F-40: Add Photo to the Gallery

Description: Tests that CogniOpen allows the user to add a photo to the Gallery.
SRS Requirement: SRS-13
Requirements: The system is able to add a photo to the Gallery.
Prerequisites: The application's home page is loaded.

Steps:
1. Tap “Gallery” tile
2. Tap the Camera icon.
3. Tap the Camera icon again to capture a picture of some eye glasses.
4. Tap the checkmark

Expected output: An image of the eye glasses is populated in the Gallery.
Assumptions: CogniOpen has been granted access to the device’s camera.

4.4 Non-Functional Tests

This section outlines all the non-functional tests based on the non-functional requirements defined (section 4 of the SRS document) for the software system. This section will highlight tests that are in-scope and out-of-scope. Non-functional requirements defined as in-scope will be the responsibility of DTTS to create test cases for. Out-of-scope non-functional requirements will only be documented here for awareness. The test cases related to those non-functional requirements are omitted from this document.

In-Scope (DTTS)

Performance**:**

4.4.1 Test Case NFR- 01: Start-up Time

Description: The application should load the login screen within 5 seconds of starting the app.
SRS Requirement: NFR-Perf-1
Requirements: The application should load within 10 seconds of it being opened.
Prerequisites: The user has installed the application on his/her device in trace startup and profile mode.
Steps:
1. The user opens the application from their mobile device.
2. The user observes the login screen

Expected output: Total load time is less than 5 seconds.
Assumptions: The user does not have any other apps running on his/her phone.

4.4.2 Test Case NFR- 02: Response Time Navigation Video

Description: The application should respond to user interactions within 2 seconds for navigation to the video recording.
SRS Requirement: NFR-Perf-2
Requirements: The system shall respond within 2 seconds to user interaction when it comes to initiating video recording.
Prerequisites: The user accesses the application from their mobile device.
Steps:
1. The user opens the application from their mobile device.
2. The system starts the video recording functionality.

Expected output: The system initiates the video recording.

Assumptions: The user already has an account setup.

4.4.3 Test Case NFR- 03: Response Time Navigation Gallery

Description: The application should respond to user interactions within 2 seconds for navigation to the gallery screen.
SRS Requirement: NFR-Perf-2
Requirements: The system shall respond within 2 seconds to user interaction when it comes to navigating to the gallery screen.
Prerequisites: The user is authenticated into the system and is on the home screen.
Steps:

1. The user is on the home screen, and they click the gallery option.
2. The system loads the gallery screen within 2 seconds.

Expected output: The system loads the gallery screen.
Assumptions: The user already has an account setup.

4.4.4 Test Case NFR- 04: Response Time Navigation Profile

Description: The application should respond to user interactions within 2 seconds for navigation to the profile screen.
SRS Requirement: NFR-Perf-2
Requirements: The system shall respond within 2 seconds to user interaction when it comes to navigating to the profile screen.
Prerequisites: The user is authenticated into the system and is on the home screen.
Steps:

1. The user is on the home screen, and they click the profile option.
2. The system loads the profile screen within 2 seconds.

Expected output: The system loads the profile screen.
Assumptions: The user already has an account setup.

4.4.5 Test Case NFR- 05: Response Time Navigation Virtual Assistant

Description: The application should respond to user interactions within 2 seconds for navigation to the virtual assistant screen.
SRS Requirement: NFR-Perf-2
Requirements: The system shall respond within 2 seconds to user interaction when it comes to navigating to the virtual assistant screen.
Prerequisites: The user is authenticated into the system and is on the home screen.
Steps:

1. The user is on the home screen, and they click the virtual assistant option.
2. The system loads the virtual assistant screen within 2 seconds.

Expected output: The system loads the virtual assistant screen.
Assumptions: The user already has an account setup.

4.4.6 Test Case NFR- 06: Response Time Significant Objects

Description: The application should respond to user interactions within 2 seconds for navigation to the significant objects screen.
SRS Requirement: NFR-Perf-6
Requirements: The system shall respond within 2 seconds to user interaction when it comes to navigating to the significant objects screen.
Prerequisites: The user is authenticated into the system and is on the home screen.
Steps:

1. The user is on the home screen, and they click the significant objects option.
2. The system loads the significant objects screen within 2 seconds.

Expected output: The system loads the significant objects screen.
Assumptions: The user already has an account setup.

Reliability:

4.4.7 Test Case NFR- 07: Uptime and Availability

Description: The application should have a minimum uptime of 99% to ensure consistent access for users.
SRS Requirement: NFR-Rel-1
Requirements: The system shall be available to all kinds of users 99% of the time.
Prerequisites: User has registered account.
Steps:

1. Monitoring service polls external services.
2. App remains open for 24 hours.

Expected output: The system displays the high availability of external services.
Assumptions: The external web services are available according to their Service Level Agreements (SLAs). The application will have real-time monitoring.

4.4.8 Test Case NFR- 08: Backup and Recovery

Description: Regular automated backups should be performed to facilitate data recovery.
SRS Requirement: NFR-Rel-2
Requirements: The system shall perform backup and retrieval procedures.
Prerequisites: The user has local data stored on their device. An automated backup has occurred.
Steps:

1. The user deletes some local data.
1. The user navigates to the local backup in account settings.
2. The user initiates backup recovery.

Expected output: The local data is restored.
Assumptions: User’s device has enough storage to store the backup.

4.4.9 Test Case NFR- 09: Offline Use Data Access

Description: Local data retrieval functionality still works properly in circumstances of low network connectivity or no network connection at all.
SRS Requirement: NFR-Rel-3
Requirements: The system shall still be able to retrieve data stored in the local database when there is low or no network connection.
Prerequisites: The network connection to the device is low or non-existent.
Steps:

1. User turns off the Wi-Fi or cellular data connection to their device.
2. User navigates to the gallery.
3.. User clicks an image file.
4. The system displays the image file.

Expected output: The user is able to view the image file.
Assumptions: An image file already exists in the gallery.

4.4.10 Test Case NFR- 10: Offline Use Recording

Description: Recording functionality still works properly in circumstances of low network connectivity or no network connection at all.
SRS Requirement: NFR-Rel-4
Requirements: The system shall still be able to record when there is low or no network connection.
Prerequisites: The network connection to the device is low or non-existent.
Steps:

1. The user navigates to the recording feature in the system.
2. User clicks the record button to start recording.
3. User clicks the record button to stop the recording.
4. The system stores the recording.

Expected output: The user records video or audio and that media is saved when the recording has concluded.
Assumptions: The audio and camera work on the user's device.

Usability:

4.4.11 Test Case NFR- 11: Speech Processing

Description: Speech-to-text for audio/video processing displays a message within 2 seconds, after completion of external processing.
SRS Requirement: NFR-Usa-1
Requirements: The system shall display a message after processing speech/text requests, within 2 seconds, after external processing has been completed.
Prerequisites: The speech/text processing functionality is working properly.
Steps:

1. The user records audio or video.
2. The system processes the audio/video file.
3. Within 2 seconds, the system displays a message indicating the processing has been completed.

Expected output: The system displays a message when processing has been completed.
Assumptions: The mobile device has storage to store the audio/video file.

4.4.12 Test Case NFR- 12: Video Processing

Description: Video processing displays a message within 2 seconds, after completion.
SRS Requirement: NFR-Usa-2
Requirements: The system shall display a message after processing video requests, within 2 seconds after external processing has been completed.
Prerequisites: A video was recorded.
Steps:
1. The user records a video.
2. The system processes the video file.
3. Within 2 seconds, the system displays a message indicating the processing has been completed.

Expected output: The system displays a message when processing has been completed.
Assumptions: The mobile device has storage to store the video.

4.4.13 Test Case NFR- 13: Status Messages

Description: Processing requests will display status message during and after process completion.
SRS Requirement: NFR-Usa-3
Requirements: The system shall display a message during and after a process is completed.
Prerequisites: A process is initiated.
Steps:
1. The user starts to record a video.
2. The user ends the recording.
3. The system displays a message indicating the processing is in progress.
4. The processing is complete, and the system displays a message indicating that processing has been completed.

Expected output: The system displays a status message during and after a process is completed.
Assumptions: The mobile device has access to networking connection.

4.4.14 Test Case NFR- 14: User Guidance

Description: Help documentation for users accessing/navigating core functionalities.
SRS Requirement: NFR-Usa-4
Requirements: The system will provide documentation for all core functionality of the system to guide users.
Prerequisites: User understands the language of the documentation.
Steps:

1. The user clicks on the documentation option.
2. The system displays the documentation page.
3. The user navigates to the section that aligns with their respective issue.

Expected output: The system displays the documentation page.
Assumptions: User understands the documentation that was written.

Out-of-scope (Team B)

Security:

- NFR-Sec-1: Data Transfer - All site data transferred to external services must be conducted via encrypted tunnels (i.e., HTTPS).
- NFR-Sec- 2: Data Encryption - User data, including personal and medical information, should be encrypted both during transmission and storage to prevent unauthorized access.
- NFR-Sec-3: Authentication - Robust user authentication mechanisms, including biometric options, should be implemented to ensure only authorized users can access the app.
- NFR-Sec-4: Authorization Levels - Different levels of authorization should be established to manage access to sensitive features and data, such as caregiver accounts.
- NFR-Sec-5: Privacy Compliance - The app should adhere to relevant data protection regulations, such as GDPR or HIPAA, depending on the jurisdiction and data processed.
- NFR-Sec-6: Secure Development - Adhere to secure coding practices to minimize vulnerabilities and potential entry points for cyberattacks.

Scalability:

- NFR-Sca-1: User Growth - The app should handle a minimum of 10,000 concurrent users without significant degradation in performance
- NFR-Sca-2: Server Scalability - The backend infrastructure should be designed to scale horizontally, adding additional resources as demand increases.
- NFR-Sca-3: Database Scalability - The database should be capable of handling increased data load and user interactions without compromising performance.

Appendix A: Unit Test Template

Template used for unit tests:

<Test Case #>

Test case name:
Method being tested:
Short Description:
Input Data to constructor and/or method you are testing:
Expected Results: