Assistive Vision

Design Document

Version 4
03/31/2025

VERSION HISTORY

Version #	Implemented By	Completion Date	Reason
1.0	Venkat Yenduri Pierre Tawfik Sukrut Nadigotti Sharefa Alshaary	*02/16/2025*	Initial Design Document draft.
2.0	Venkat Yenduri Pierre Tawfik Sukrut Nadigotti Sharefa Alshaary	*02/23/2025*	Revisions after group review.
3.0	Venkat Yenduri Pierre Tawfik Sukrut Nadigotti Sharefa Alshaary	*02/26/2025*	Modifications after feedback from GTA & professor.
4.0	Venkat Yenduri Pierre Tawfik Sukrut Nadigotti Sharefa Alshaary	*03/31/2025*	Updated design document to meet updated project features.

TABLE OF CONTENTS

1 Introduction **4**
1.1 Purpose of The Design Document 5
1.2 Intended Audience 5
2 General Overview and Design Guidelines/Approach **5**
2.1 Overview 5
2.2 Design Guidelines 6
2.3 Assumptions / Constraints / Standards 6
2.3.1 Assumptions 6
2.3.2 Constraints 7
2.3.3 Standards 8
3 Architecture Design **8**
3.1 Hardware Architecture 8
3.2 Software Architecture 9
3.3 Security Architecture 10
3.4 Communication Architecture 12
3.5 Performance 13
4 System Design **15**
4.1 Use-Cases 15
4.1.1 Launch Application 15
4.1.2 Object Detection in Continuous Mode 16
4.1.3 Object Detection in On-Demand Mode 17
4.1.4 Adjust Object Detection Settings 17
4.1.5 Scene Exploration Mode 18
4.1.6 Enable Battery Saver Mode 19
4.1.7 Review Recent Object Detections 20
4.1.8 Configure Audio & Haptic Feedback 21
4.1.9 Enable Scene Navigation Mode 21
4.1.10 Identify Specific Object in View 22
4.1.11 Detect Obstacles While Walking 23
4.1.12 Camera Initialization 24
4.1.13 Provide Audio Feedback 25
4.1.14 Voice Command Interface 26
4.1.15 Hazard Alert Customization 27
4.1.16 Siri Shortcut 28
4.2 Use Case Diagram 28
Actors in the System: 29
Key Use Cases: 29

4.3 Sequence Diagrams 30
4.3.1 Launch Application 32
4.3.2 Object Detection Mode 32
4.3.3 Object Detection in on-Demand Mode 33
4.3.4 Adjust Object Detection Settings 33
4.3.5 Scene Exploration Mode 34
4.3.6 Enable Battery Saver Mode 34
4.3.7 Review Recent Object Detections 35
4.3.8 Configure Audio & Haptic Feedback 35
4.3.9 Enable Scene Navigation Mode 36
4.3.10 Identify Specific Object in View 36
4.3.11 Detect Obstacles While Walking 37
4.3.12 Camera Initialization 37
4.3.13 Provide Audio Feedback 38
4.3.14 Voice Command Interface 39
4.3.15 Hazard Alert Customization 39
4.3.16 Siri Shortcuts 40
4.4 Data Flow Diagram 41
4.5 Database Design 42
Overview of the Database Diagram 42
4.6 Class Diagram 43
4.7 Application Program Interfaces 46
4.8 User Interface Design 47
4.8.1 Launch Screen 48
4.8.2 Main Interface 49
4.8.3 Detection Mode 50
4.8.4 Detection History 51
4.8.5 FAQ 52
4.8.6 Tutorial 53
4.8.7 Settings 54
APPENDIX A: KEY TERMS 55
References 58

1 Introduction

1.1 Purpose of The Design Document

The purpose of this design document is to explain the architecture and system design of the Assistive Vision iOS application. It is meant for developers, testers, and anyone who wants to understand how the system works. This document outlines key technical details, including how the software is structured, how different components interact, and how the system handles real-time object detection and audio feedback.

It provides diagrams, design decisions, and system workflows to guide the development process. The goal is to ensure the Assistive Vision app is built correctly, meets its objectives, and remains easy to use for visually impaired individuals. By following this document, the development team can create a well organized and efficient system that enhances user independence and accessibility.

1.2 Intended Audience

The Assistive Vision Design Document is intended for multiple stakeholders involved in the development and implementation of the Assistive Vision iOS application. Primarily, it serves as a technical guide for the development team, providing a structured framework for software architecture, system components, and implementation strategies. Developers will use this document to ensure that the application meets functional and non-functional requirements while adhering to best practices in software engineering and accessibility design. By outlining system specifications, design decisions, and testing methodologies, this document helps developers maintain consistency and efficiency throughout the project lifecycle.

Additionally, this document is designed for future open-source use and academic instructors at Wayne State University, who will assess the project's technical depth, completeness, and adherence to best practices. It provides insights into design choices, ensuring that the project aligns with academic objectives and industry standards. Future developers or researchers interested in accessibility solutions may also reference this document for insights into designing assistive applications. By serving as a comprehensive resource, this document facilitates collaboration, knowledge transfer, and long-term project development.

2 General Overview and Design Guidelines/Approach

2.1 Overview

This document explains the design and structure of the Assistive Vision iOS application. It provides a clear breakdown of how the system works, including its architecture, features,

and technical details. The goal is to help developers, testers, and other stakeholders understand how different parts of the application interact and function.

The document covers key areas such as design guidelines, hardware and software architecture, security measures, and system interactions. It also includes detailed use cases, diagrams, and database design to show how the app processes information and responds to user input. Additionally, it outlines the main frameworks and technologies used, such as CoreML for object detection and AVFoundation for audio feedback.

2.2 Design Guidelines

The Assistive Vision iOS application follows a user-centered design philosophy focused on accessibility, efficiency, and real-time responsiveness. The interface is designed for simplicity and ease of use, allowing visually impaired users to interact through voice commands and real-time audio feedback. Adhering to Apple's Human Interface Guidelines (HIG) for accessibility, the application integrates with VoiceOver, Siri Shortcuts, and Dynamic Type to enhance usability. Its minimalistic design ensures intuitive navigation, reducing complexity for users. The development approach follows a modular and scalable architecture, using the Model-View-Controller (MVC) pattern to separate data processing, system logic, and user interface components. On-device AI processing with CoreML and Apple's Neural Engine eliminates reliance on cloud-based services, improving speed, security, and privacy while ensuring low-latency responses, efficient memory management, and adaptive power-saving features. Additionally, robust error handling and security measures safeguard the application from failures and unauthorized access. By combining accessibility, performance, and security, Assistive Vision provides a fast, private, and adaptive assistive tool, empowering visually impaired users with greater independence while allowing for future scalability and enhancements.

2.3 Assumptions / Constraints / Standards

2.3.1 Assumptions

It is assumed that during the development of the Assistive Vision iOS application, users will interact with the system using voice commands. The application is designed primarily for visually impaired individuals, so accessibility features such as VoiceOver and Shortcuts will be essential for navigation.

Additionally, it is expected that the development team will follow all Wayne State University guidelines throughout the design, implementation, and testing phases. The application will be developed with a focus on privacy, efficiency, and accessibility, ensuring that it meets the needs of its users while complying with ethical and technical standards.

2.3.2 CONSTRAINTS

The project must be completed within one academic semester (12 weeks), requiring the team to focus on essential features such as real-time object detection, audio feedback, and accessibility integration. Advanced features or optimizations may be scheduled for later iterations.

The development team consists of four undergraduate students with varying levels of experience in iOS development, machine learning, and accessibility-focused design. As a learning project, progress may depend on the team's familiarity with technologies such as CoreML, AVFoundation, and SwiftUI.

With no allocated budget, all development will be conducted using open-source tools and Apple's built-in frameworks. On-device AI processing is required to ensure user privacy and efficiency, eliminating reliance on external cloud-based services.

The application is designed exclusively for iOS devices, leveraging Apple's A-series/M-series chips and the Apple Neural Engine for on-device processing. The project requires a minimum device specification—supporting iOS 17 or later—to meet performance and functionality targets.

Development is constrained by the Apple ecosystem, using the latest stable versions of Xcode and Swift (for example, Xcode 15 and Swift 5.8) alongside native frameworks such as SwiftUI, AVFoundation, and CoreML to ensure compatibility, security, and optimal performance.

Finally, the app must adhere to Wayne State University's project guidelines, Apple's Human Interface Guidelines, and WCAG accessibility standards to ensure that it remains secure, efficient, and accessible for visually impaired users.

2.3.3 STANDARDS

The Assistive Vision iOS application will be designed to comply with Wayne State University's project guidelines while maintaining a user-friendly and highly accessible experience. The interface will be developed with simplicity and ease of navigation in mind, ensuring that visually impaired users can interact with the application efficiently. Voice commands, tactile gestures, and Siri integration will be the primary methods of user interaction, reducing the need for complex manual inputs.

The design will follow WCAG (Web Content Accessibility Guidelines) for accessibility, ensuring compatibility with VoiceOver, Dynamic Type, and other iOS accessibility features. The user interface will be uncluttered, responsive, and intuitive, providing clear

audio feedback with minimal delay. Additionally, the system's text-to-speech output will be optimized for clarity, allowing users to receive precise and contextually relevant information about their surroundings.

To maintain performance, the application will use on-device processing with CoreML to ensure fast and private object detection. The app will also be optimized for low latency, battery efficiency, and real-time processing, making it a reliable tool for visually impaired individuals in various environments.

3 Architecture Design

3.1 Hardware Architecture

Devices:

iPhone (Primary Device)

Bluetooth-compatible audio devices (Headphones, Hearing Aids, Smart Speakers)

IPhone Hardware Components:

Camera: Rear-facing/Front-facing camera used for real-time object detection

Processor: Apple A-series/M-series chip with Neural Engine for on-device AI processing

RAM: Optimized memory management for real-time inference

Storage Capacity: Utilizes local storage for user settings and detection history

Architecture: 64-bit ARM-based processor

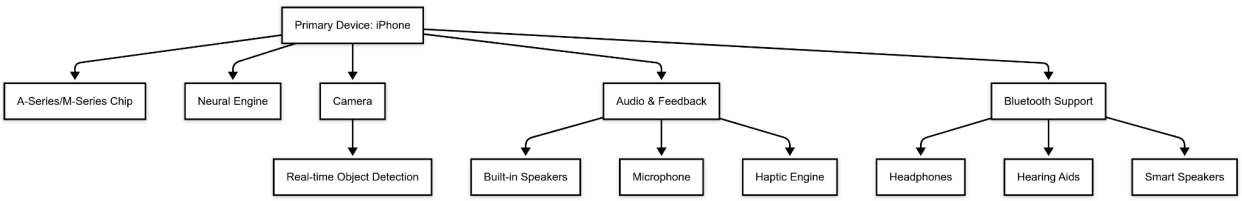
Audio Feedback Components:

Speakers: Built-in stereo speakers for real-time audio feedback

Microphone: Captures voice commands for Siri Shortcuts and voice-based navigation

Haptic Engine: Provides vibration feedback for alerts and notifications

By leveraging the iPhone's built-in hardware, the Assistive Vision application ensures fast, efficient, and privacy-focused real-time object detection, delivering an intuitive and seamless assistive experience for visually impaired users.



3.2 Software Architecture

This section outlines the updated high-level software architecture of the Assistive Vision iOS application, incorporating all recent modifications and enhancements to the system.

The architecture comprises the following components:

- **User (Visually Impaired)** – Interacts primarily through voice commands, audio feedback, and optional haptic feedback. The system supports seamless integration with Siri Shortcuts, enabling quick hands-free access and interaction.
- **View (UI Layer)** – Provides real-time visualizations of object detection results using high-contrast bounding boxes and simplified UI elements optimized for visually impaired users. It integrates VoiceOver for accessible navigation, Siri Shortcuts for voice activation, and Dynamic Type for adjustable text sizes. Additionally, the View layer includes distinct screens for detection mode selection (Continuous or On-Demand), detection history logs, FAQs, tutorials, settings customization, and scene exploration summaries.
- **Controller (Logic Layer)** – Coordinates user interactions, manages communication between the View and Model layers, and processes voice commands via Apple's Speech framework. It interfaces with AVFoundation for real-time camera access, audio processing, and speech synthesis. The Controller dynamically adjusts settings like detection sensitivity, battery-saving measures, and hazard alerts according to user preferences and current device status.
- **Model (Data Processing)** – Incorporates the YOLOv11m CoreML object detection model, utilizing Apple's Neural Engine for efficient on-device processing. The Model layer stores and manages user preferences, detection settings (such as confidence thresholds, Intersection over Union (IoU) values, and maximum detected objects), and historical detection data. It ensures that all object detection computations are performed locally, maintaining privacy and minimizing latency.
- **iOS Frameworks** – Includes native frameworks such as CoreML for machine learning inference, Vision for computer vision tasks, AVFoundation for multimedia capture and playback, UIKit for interface construction, CoreMedia and CoreVideo for efficient media processing, and Speech for speech recognition. The architecture relies solely on these native frameworks to ensure compatibility, performance optimization, and security.

This refined architecture strictly adheres to the Model-View-Controller (MVC) design pattern, emphasizing modularity, scalability, and maintainability. The design ensures rapid responsiveness, robust error handling, and adaptive power management, accommodating future enhancements such as expanded detection models, advanced accessibility options, and personalized user experiences.

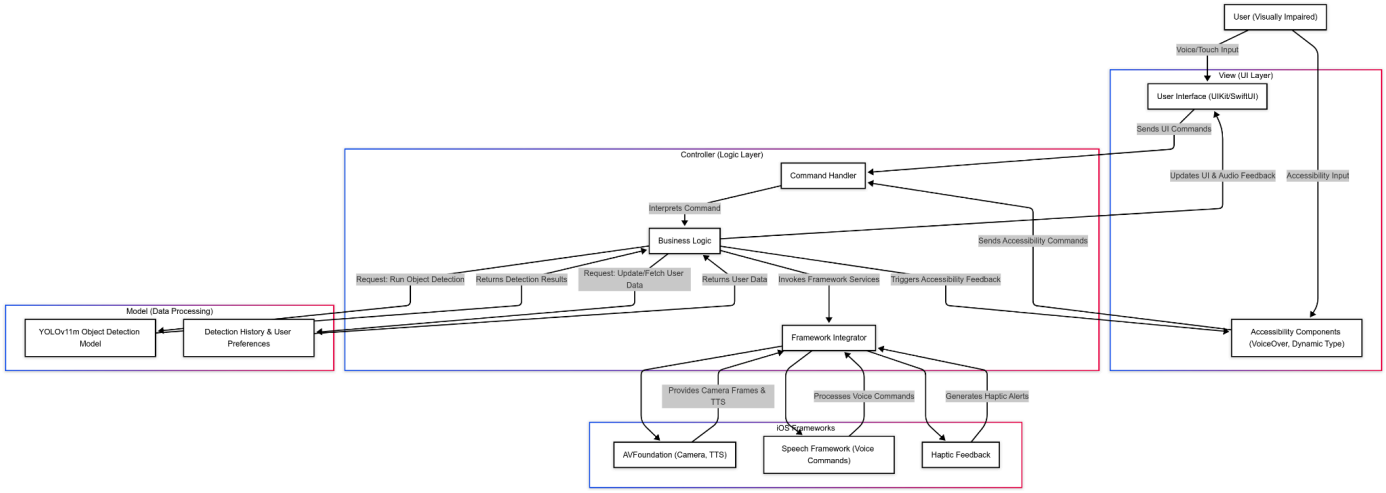


Figure 1

3.3 Security Architecture

The Assistive Vision iOS application prioritizes security and privacy, safeguarding user data, managing permissions carefully, and maintaining system integrity against unauthorized access and vulnerabilities. Real-time object detection occurs entirely on-device, significantly enhancing privacy by ensuring no personal data or detection history is transmitted or stored externally. This security architecture follows iOS best practices, including strict sandboxing, careful permission management, and encrypted local storage to secure user information.

Key Security Components:

Access Control & Authentication

- The application requests only essential permissions, such as access to the camera, microphone, and speech recognition services, strictly when required.
- Users can manage these permissions via iOS settings, with clear guidance provided by the app on enabling or revoking permissions.

Data Protection & Privacy

- All object detection processing occurs locally using CoreML and Apple's Neural Engine, eliminating dependence on cloud-based AI services and thus safeguarding user privacy.
- User settings and preferences are securely stored locally using Apple’s Secure Storage APIs, such as encrypted NSUserDefaults for sensitive data.
- The application does not store camera feed recordings or retain visual data beyond immediate detection.

Network & Communication Security

- The only instance of external data transmission occurs through Google's Gemini third-party API for advanced voice-based interactions, strictly adhering to HTTPS/TLS 1.2+ protocols for secure communication.
- Apart from this controlled interaction, the application abstains from any external data transmission, third-party analytics, or tracking mechanisms, reinforcing privacy and autonomy.

Application Security & Reliability

- Secure coding standards are rigorously enforced to mitigate vulnerabilities, including buffer overflows, unauthorized API access, and input validation issues.
- Robust error handling and logging mechanisms ensure stability and safeguard sensitive information from unauthorized exposure.
- Security notifications, including haptic feedback and voice alerts, keep users informed about critical actions or permissions.

By strictly adhering to these security measures, Assistive Vision ensures a private, secure, and dependable experience, enabling visually impaired users to safely navigate their environment without risking personal data exposure or system integrity compromise. Continuous review and updates of the security framework address emerging threats and evolving user requirements.

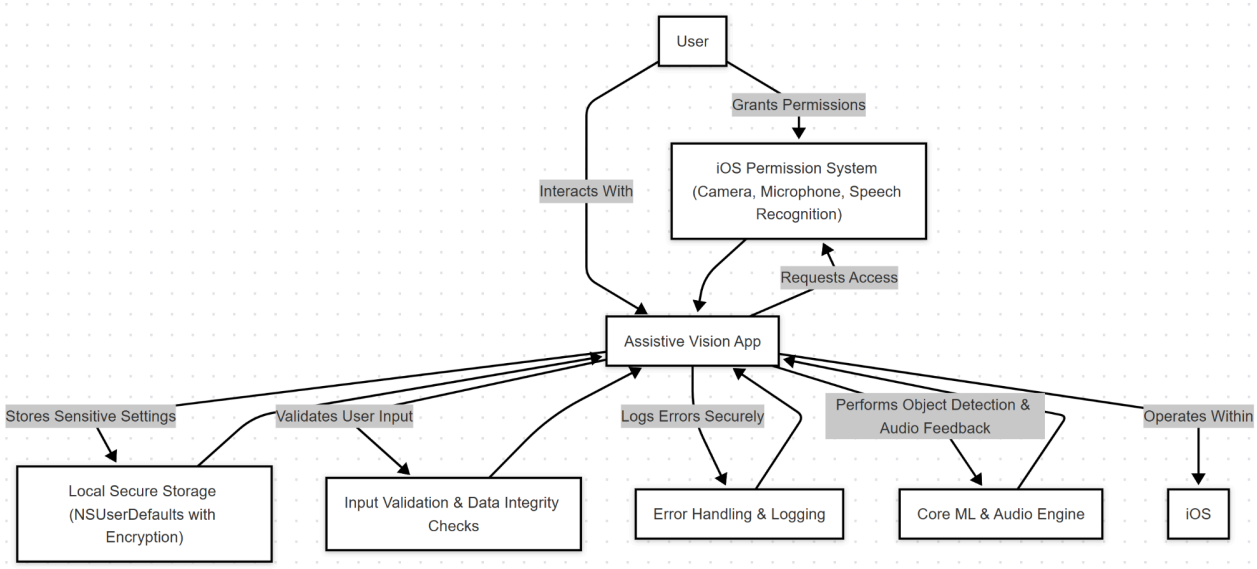


Figure 2

3.4 Communication Architecture

The Assistive Vision iOS application follows a localized and self-contained communication architecture, designed to operate without internet dependency, ensuring privacy, security, and reliability for visually impaired users. The system primarily facilitates internal communication

within the device, leveraging iOS-native frameworks to enable seamless interactions between components while minimizing latency and power consumption.

Key Communication Components

1. Internal Component Communication
    - The camera module captures real-time video frames and transmits them to the custom YOLOv11m object detection model via CoreML, which processes the images and identifies objects.
    - Once an object is detected, the information is passed to the audio feedback system, which generates real-time spoken descriptions using Apple's Text-to-Speech (TTS) API.
    - User preferences and detection settings are stored locally using NSUserDefaults and Core Data, allowing the app to dynamically adapt detection behaviors.
2. Voice Command & Accessibility Integration
    - The app supports hands-free control via in-app voice commands and iOS Speech Recognition APIs, enabling users to activate object detection, adjust settings, or request environmental descriptions using voice commands.
    - Haptic feedback and VoiceOver provide alternative communication methods, ensuring accessibility for users who rely on tactile feedback or screen readers.
3. No External Data Transmission
    - Unlike cloud-based AI systems, Assistive Vision operates entirely offline, ensuring that all object detection and user interactions remain private within the device.
    - There are no API calls, external database dependencies, or network-based communication protocols, reducing potential security risks such as data leaks or third-party tracking.
    - The app allows users to connect Bluetooth audio devices (e.g., hearing aids, wireless headphones) to enhance the audio feedback experience via iOS Bluetooth communication protocols.
4. Efficiency & Low-Latency Processing
    - The communication flow between the camera, detection model, and audio engine is optimized to ensure near-instantaneous feedback, allowing visually impaired users to receive real-time object descriptions without delays.
    - Dynamic power management ensures that frame processing rates are adjusted based on battery status, preventing unnecessary energy consumption while maintaining responsiveness.

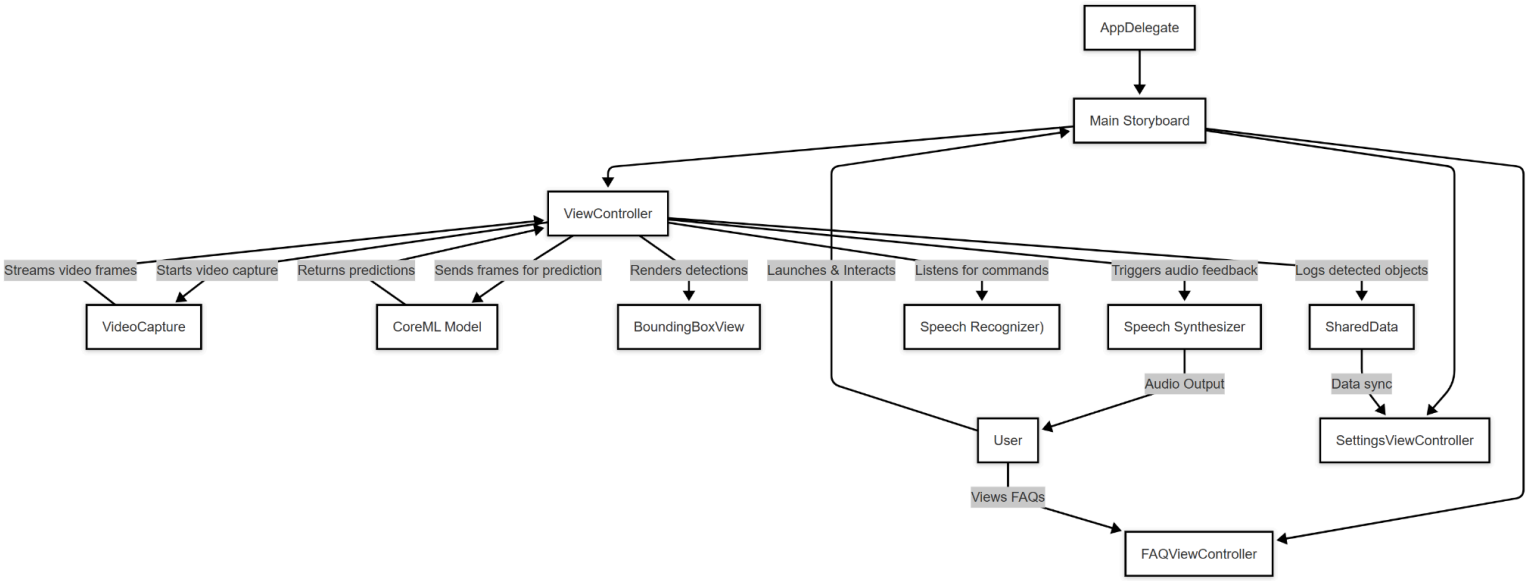


Figure 3

3.5 Performance

The Assistive Vision iOS application is engineered to optimize responsiveness, efficiency, security, reliability, and user experience, ensuring seamless real time object detection and audio feedback for visually impaired users. The system is designed to be lightweight and energy efficient while providing near instantaneous responses to user interactions. For example, the camera must initialize within one second after launch so that core modules such as the camera, object detection, and audio feedback become operational promptly. In this way the application announces "Assistive Vision activated" and displays the home screen without delay.

Real time object detection is central to the application’s performance goals. The custom YOLOv11m model running on Apple's Neural Engine via CoreML processes every frame within 1000 milliseconds. This guarantees that objects entering the camera's field of view are detected and announced swiftly, preserving the critical low latency feedback necessary for safe navigation. In addition, when users activate scene exploration mode the system is designed to capture and analyze a high resolution snapshot and deliver a comprehensive spoken scene summary within two seconds. These quantifiable targets complement the original emphasis on consistent frame rates and optimized on device machine learning that eliminates delays caused by external processing.

Scalability remains a key focus as the machine learning pipeline is optimized to balance computational efficiency with detection accuracy, maintaining an accuracy rate of at least 90 percent across diverse environments such as indoor, outdoor, and crowded spaces. As iOS devices evolve the application is structured to leverage new hardware capabilities, ensuring long term compatibility and improved performance on future models.

Security is inherent in the system design because all processing is conducted offline. This approach not only minimizes processing delays but also ensures that no user data is transmitted externally, protecting privacy through iOS's sandboxing and strict permission management. By performing object detection on device the system avoids the complexities associated with external database encryption and remote authentication.

Reliability is maintained through robust error handling and power management features. The application incorporates a Battery Saver mode that activates automatically when the device's battery level drops below 15 percent. In this mode the frame rate is reduced, for instance from 30 frames per second to 15 frames per second, and the detection frequency is adjusted to conserve power without sacrificing essential performance. In addition dynamic sensitivity adjustments based on environmental conditions help ensure accurate detection even under varying lighting and movement scenarios.

Finally the user experience is prioritized through intuitive voice commands, tactile interactions, and an accessibility focused design that leverages VoiceOver, haptic feedback, and Siri integration. The interface remains minimal yet responsive, delivering clear spoken descriptions and simple navigation controls that are tailored to the needs of visually impaired users. By combining precise quantifiable metrics such as startup times, frame processing targets, and scene summary response times with broad goals of scalability, security, reliability, and accessibility the Assistive Vision application delivers a high performance, dependable tool that enhances independence and situational awareness in daily life.

4 System Design

4.1 Use-Cases

4.1.1 Launch Application

ID	UC-01
Name	Launch Application
Actors	User
Description	When opening the application, the user can launch it using a voice command (e.g., Siri) or by tapping the app icon. The app initializes the camera, object detection model, and audio engine while providing an audio confirmation that states "Assistive Vision activated".
Assumptions	The user has installed the app and granted the required camera and microphone permissions.

Trigger	The user says "Hey Siri, open Assistive Vision" or taps the app icon.
Preconditions	The app must be installed, and necessary permissions must be granted.
Normal Flow	1. The user launches the application. 2. The system initializes the camera, object detection model, and audio engine. 3. The system confirms successful activation with an audio message. 4. The app becomes fully operational within 2 seconds of launch.
Postconditions	The app is fully initialized and ready for object detection and audio feedback.
Alternative Flow	If the camera is already in use by another application, the system provides an option to proceed with limited functionality (e.g., using preloaded data) or suggests closing conflicting apps.
Exceptions	If camera access is denied, the app informs the user and provides instructions to enable permissions in settings. - If microphone access is unavailable, the app notifies the user and suggests enabling it for full functionality.

4.1.2 Object Detection in Continuous Mode

ID	**UC-02**
Name	Object Detection in Continuous Mode
Actors	User
Description	The user enables Continuous Mode, allowing the app to automatically scan and provide real-time descriptions of detected objects

	without requiring repeated manual input. The system continuously processes the camera feed, detecting objects, and providing ongoing audio feedback throughout its operation.
Assumptions	The user has granted camera access, and the CoreML model is initialized.
Trigger	The user says "Enable continuous scanning" or toggles the Continuous Mode option in the settings.
Preconditions	The app must be open, and the camera must be active.
Normal Flow	1. The user enables Continuous Mode. 2. The app continuously captures frames, detects objects, and provides real-time audio descriptions. 3. The system continues detecting objects until the user disables Continuous Mode. 4. The app provides both audio and optional haptic feedback when objects are detected.
Postconditions	The app must be open, and the camera must be active.
Alternative Flow	If battery levels are low, the app dynamically adjusts processing settings to reduce power consumption while maintaining detection functionality.
Exceptions	- If camera access is lost mid-session, the app notifies the user and provides instructions to re-enable permissions. - If processing resources are insufficient, the app suggests switching to On-Demand Mode to conserve system performance.

4.1.3 Object Detection in On-Demand Mode

ID	UC-03
Name	Object Detection in On-Demand Mode
Actors	User
Description	The user manually triggers object detection using a voice command or tap gesture to receive real-time feedback on detected objects. The app captures a single frame, processes object detection, and provides immediate audio feedback before waiting for the next command.
Assumptions	The app is installed, and the camera is accessible.
Trigger	The user says "Scan now."
Preconditions	The app must be running, and the camera must be initialized.
Normal Flow	1. The user issues a scan command. 2. The app captures a frame and runs object detection. 3. The system provides an audio announcement describing detected objects and their confidence levels. 4. The app returns to a ready state, awaiting the next command.
Postconditions	The app delivers detected object information only when explicitly requested.
Alternative Flow	If no objects are detected, the app suggests rescanning with better lighting or a different angle rather than simply stating "No objects detected".
Exceptions	- If camera access is denied, the app notifies the user and provides instructions to re-enable permissions. - If object detection fails due to technical

	issues, the app suggests restarting the application or checking for updates.

4.1.4 Adjust Object Detection Settings

ID	UC-04
Name	Adjust Object Detection Settings
Actors	User
Description	The user customizes object detection settings, including confidence thresholds, IoU thresholds, max objects, prioritized objects, and audio preferences to enhance usability. These settings can be modified via the settings screen or through voice commands.
Assumptions	The app supports customization and retains user preferences for future use.
Trigger	The user opens settings or uses voice commands such as "Prioritize stairs and vehicles" or "Set confidence threshold to 0.5".
Preconditions	The app must be running, and the settings menu must be accessible.
Normal Flow	1. The user accesses detection settings via the settings screen or voice command. 2. The app updates detection filters based on the user's preferences. 3. The system confirms successful customization with audio feedback. 4. Changes persist across app restarts.
Postconditions	The updated detection settings take effect immediately and are stored for future use.
Alternative Flow	If an object type is not available in the detection model, the app suggests the closest recognized object type or provides a list of

	detectable categories.
Exceptions	- If an unsupported object type is requested, the system notifies the user and provides alternative detection options. - If the settings cannot be applied due to a system error, the app offers troubleshooting steps.

4.1.5 Scene Exploration Mode

ID	UC-05
Name	Scene Exploration Mode
Actors	User
Description	The user pauses real-time detection to explore the surroundings and receive a structured scene analysis, enhancing spatial awareness. The app captures a snapshot and provides a comprehensive spoken summary of objects and their spatial relationships.
Assumptions	The app supports scene analysis and can provide structured reports.
Trigger	The user says "Explore scene."
Preconditions	The app must have camera access enabled and a clear field of view.
Normal Flow	1. The user activates Scene Exploration Mode. 2. The app captures a snapshot and detects objects in the surroundings. 3. The system generates an audio summary describing object locations and distances. 4. The user receives structured feedback for navigation. 5. The summary is delivered within 2 seconds of activation.

Postconditions	The app delivers a structured audio report on the surrounding environment.
Alternative Flow	If the camera cannot capture a clear image, the app enhances object detection using depth-sensing technology or suggests adjusting the device position for better visibility.
Exceptions	- If the scene is unclear due to low visibility or obstructions, the app notifies the user and provides guidance to improve detection. - If scene analysis encounters an error, the app offers a basic environmental summary using last-known object data.

4.1.6 Enable Battery Saver Mode

ID	**UC-06**
Name	Enable Battery Saver Mode
Actors	User
Description	The user activates Battery Saver Mode to reduce frame processing and optimize power consumption, thereby extending battery life. This mode adjusts frame rate and processing intensity to conserve power while maintaining essential functionality.
Assumptions	The device is low on battery, or the user wants to conserve power.
Trigger	The user says "Enable Battery Saver Mode" or the battery level drops below 15%.
Preconditions	The app must be open, and object detection must be running.
Normal Flow	1. The user enables Battery Saver Mode or the system activates it automatically at low battery levels. 2. The app reduces the frame processing rate

	(e.g., from 30fps to 15fps) and optimizes resource consumption. 3. The system confirms the activation of Battery Saver Mode. 4. The app continues to function with reduced power demands.
Postconditions	The app operates in a power-efficient mode to extend battery life.
Alternative Flow	If Battery Saver Mode is already active or power settings are optimized, the app dynamically adjusts additional settings, such as disabling background processes, to further reduce power consumption.
Exceptions	- If the battery level is critically low, the app alerts the user and suggests plugging in the device. - If Battery Saver Mode cannot be activated due to system restrictions, the app provides troubleshooting guidance.

4.1.7 Request Detection Summary

ID	**UC-07**
Name	Request Detection Summary
Actors	User
Description	The user requests a summary of the 10 most recently detected objects. The app retrieves this data from the log file, where detected objects and their timestamps are stored. The system then announces the detections in a structured format to provide context about the recent environment.
Assumptions	The app has logged previously detected objects.
Trigger	The user says "What did you detect recently?"

Preconditions	The app is actively running, listening for voice commands, and object detection is enabled. At least one object has been detected and stored in the log file.
Normal Flow	1. The user requests a recent detection history. 2. The app retrieves the last 10 detected objects with their timestamps. 3. The system announces them in chronological order with confidence levels.
Postconditions	The user hears a list of recent objects detected by the app.
Alternative Flow	N/A
Exceptions	Error: No Data Available – If there are no recent detections, the app provides a relevant message instead of remaining silent, such as "No objects have been detected yet".

4.1.8 Configure Audio & Haptic Feedback

ID	**UC-08**
Name	Configure Audio & Haptic Feedback
Actors	User
Description	Adjust audio volume, voice type, speech rate, and haptic feedback intensity.
Assumptions	Feedback settings customizable.
Trigger	Voice commands, segmented controls, or settings menu.
Preconditions	App running, settings accessible.
Normal Flow	1. User selects the feedback setting. 2. Preferences updated. 3. Changes confirmed audibly. 4. Settings persist.

Postconditions	Updated feedback settings active.
Alternative Flow	Suggest alternatives if options are unavailable.
Exceptions	Notify clearly of invalid requests.

4.1.9 Enable Scene Navigation Mode

ID	**UC-09**
Name	Enable Scene Navigation Mode
Actors	User
Description	The user activates Scene Navigation Mode, allowing the app to provide a structured spatial overview of objects in the environment. This mode offers detailed guidance for movement through spaces, highlighting potential obstacles and pathways.
Assumptions	The user requires detailed environmental awareness to assist with navigation.
Trigger	The user says "Enable Scene Navigation."
Preconditions	The app must be running, and the camera must have an unobstructed view.
Normal Flow	1. The user enables Scene Navigation Mode. 2. The app captures a scene snapshot. 3. The app analyzes the scene and generates an audio summary describing object locations and distances. 4. The user receives structured feedback to assist with navigation, including directions and proximity warnings.
Postconditions	The user gains a comprehensive spatial overview of nearby objects and potential pathways.

Alternative Flow	If the camera view is obstructed or lighting conditions are poor, the app enhances detection using depth-sensing techniques or prompts the user to adjust the device position.
Exceptions	- If objects are not detected due to environmental factors, the app notifies the user and suggests repositioning or improving lighting. - If the scene analysis encounters an error, the app provides fallback guidance using basic obstacle warnings.

4.1.10 Identify Specific Object in View

ID	**UC-10**
Name	Identify Specific Object in View
Actors	User
Description	The user requests the app to identify a specific object within the camera's field of view. The app processes the request and provides real-time feedback about the object's presence and location, helping users locate items of interest.
Assumptions	The object detection model can recognize predefined object categories.
Trigger	The user says "Find a door" or "Is there a person ahead?"
Preconditions	The app must be running, and the camera must be active.
Normal Flow	1. The user requests identification of a specific object. 2. The app scans the camera feed, focusing on detecting the requested object type. 3. If the object is found, the app provides an

	audio alert indicating the object's presence and distance. 4. The app can continue tracking the object if requested.
Postconditions	The user receives real-time confirmation about the presence and location of the requested object.
Alternative Flow	If the object is not found, the app suggests scanning the surroundings again or adjusting the camera angle for better detection.
Exceptions	- If the object is not recognized because it is outside the detection model's scope, the system informs the user and provides a list of detectable object categories. - If lighting conditions interfere with detection, the app notifies the user and suggests improving visibility.

4.1.11 Detect Obstacles While Walking

ID	**UC-11**
Name	Detect Obstacles While Walking
Actors	User
Description	The app provides real-time warnings about obstacles in the user's walking path to help prevent collisions. The system uses continuous detection to identify potential hazards and alerts the user with appropriate audio and haptic feedback.
Assumptions	The user is moving forward, and the camera is properly aligned to detect obstacles.
Trigger	The app automatically detects an obstacle or the user asks "Are there any obstacles?"
Preconditions	The app must be active, and object detection must be enabled.

Normal Flow	1. The app continuously scans the user's surroundings for obstacles. 2. If an obstacle is detected, the app provides an audio alert with distance information. 3. As the user continues moving, the app updates proximity warnings dynamically. 4. Haptic feedback intensity increases as obstacles get closer.
Postconditions	The user receives real-time obstacle alerts to assist with navigation.
Alternative Flow	If visibility conditions are poor, the app enhances detection using infrared or LiDAR-based depth sensing to improve obstacle recognition.
Exceptions	- If lighting conditions are inadequate for object detection, the system warns the user and suggests activating a flashlight or alternative sensing methods. - If the app cannot detect obstacles due to obstruction or camera failure, it notifies the user and recommends troubleshooting steps.

4.1.12 Camera Initialization

ID	**UC-12**
Name	Camera Initialization
Actors	User
Description	When the app launches, it automatically initializes the camera for real-time object detection. If permissions are missing, the app guides the user through enabling them.
Assumptions	The device has a rear camera, and the user has granted camera permissions.
Trigger	The user launches the application via voice command or by tapping the app icon.

Preconditions	The app must have camera access enabled in iOS settings.
Normal Flow	1. The app checks for camera permissions. 2. If granted, the system initializes the camera and sets the appropriate resolution and frame rate. 3. The system confirms successful initialization and enables object detection. 4. Camera initialization completes within 1 second of app launch.
Postconditions	The camera is ready for object detection, and the app is fully operational.
Alternative Flow	If camera access is restricted, the app provides an option to use stored images for object recognition instead of real-time detection.
Exceptions	- If another app is using the camera, the system notifies the user and suggests closing conflicting applications. - If the camera encounters a hardware failure, the system provides diagnostic steps or suggests using an alternative input method.

4.1.13 Provide Audio Feedback

ID	**UC-13**
Name	Provide Audio Feedback
Actors	User
Description	The app provides spoken descriptions of detected objects using the AVFoundation TTS engine, helping users understand their surroundings. Audio feedback is clear, concise, and configurable to user preferences.
Assumptions	The app has detected an object with a confidence level above the user's threshold.

Trigger	The system detects an object, or the user asks "What's in front of me?"
Preconditions	The Text-to-Speech (TTS) engine must be functional.
Normal Flow	1. The app detects an object. 2. The system generates an audio description based on object type, confidence level, and distance. 3. The description is played through the built-in speaker or a connected Bluetooth headset. 4. The speech is generated within 1 second of detection.
Postconditions	The user hears real-time audio descriptions of detected objects.
Alternative Flow	If the TTS engine is unavailable, the system provides an alternative method, such as haptic feedback or a pre-recorded sound indicating an object's presence.
Exceptions	If the TTS engine encounters an error, the app attempts to restart the audio system or notifies the user of the issue.

4.1.14 Voice Command Interface

ID	**UC-14**
Name	Voice Command Interface
Actors	User
Description	Comprehensive app control through voice commands (detection control, navigation, querying AI, closing app).
Assumptions	Voice control enabled, microphone accessible.
Trigger	Voice commands ("Start detection", "Open

	settings", "Ask [question]", etc.).
Preconditions	Speech API functional, microphone permissions granted.
Normal Flow	1. User issues a voice command. 2. Command matched and executed. 3. Confirmation via voice.
Postconditions	User-requested actions completed.
Alternative Flow	Request repetition if unclear command detected.
Exceptions	Notify users if microphone permissions disabled.

4.1.15 Hazard Alert Customization

ID	**UC-15**
Name	Hazard Alert Customization
Actors	User
Description	The user customizes hazard detection alerts to enhance safety. This includes prioritizing objects such as stairs, curbs, or vehicles and assigning unique alert tones for each. The system can be configured to provide different intensities of alerts based on the type of hazard.
Assumptions	The object detection system supports custom hazard alerts.
Trigger	The user says "Enable stairs alert" or "Set a loud tone for obstacles."
Preconditions	The app must be running, and hazard detection must be enabled.

Normal Flow	1. The user configures hazard detection settings via voice command or settings menu. 2. The app updates the hazard alert database. 3. The system confirms: "Hazard alerts updated successfully." 4. The app immediately applies the new alert preferences.
Postconditions	The new hazard settings take effect immediately and persist across app launches.
Alternative Flow	If a hazard category is already enabled, the app suggests modifying the existing settings instead of re-enabling them.
Exceptions	If an unrecognized hazard type is requested, the system notifies the user that the hazard is not supported and provides guidance on valid options.

4.1.16 Siri Shortcut

ID	**UC-16**
Name	Siri Shortcut
Actors	User
Description	The user can open the app hands-free via Siri, enabling quick access to the Assistive Vision app. Integration with Siri Shortcuts provides seamless system-level voice control.
Assumptions	The user has granted Siri the necessary permissions.
Trigger	The user says "Hey Siri, open Assistive Vision"
Preconditions	Siri Shortcuts must be enabled in the iOS settings.

Normal Flow	1. The user issues a Siri command. 2. Siri processes the command and opens the app using iOS Shortcuts. 3. The app confirms execution through audio feedback. 4. The process completes within 3 seconds of the voice command.
Postconditions	The app opens successfully, and the user receives confirmation.
Alternative Flow	If Siri permissions are not enabled, the system prompts the user to enable shortcut permissions before proceeding.
Exceptions	If Siri fails to interpret the command, the system notifies the user and prompts them to retry.

4.1.17 Tutorial Mode with Gemini AI Integration

ID	**UC-17**
Name	Tutorial Mode with Gemini AI Integration
Actors	User
Description	Interactive tutorial mode powered by Google Gemini AI providing personalized, voice-guided instructions.
Assumptions	Gemini API accessible, internet connectivity active.
Trigger	User says "Open Tutorial" or taps the Tutorial button.
Preconditions	App running, microphone access granted.
Normal Flow	1. User activates Tutorial mode. 2. Gemini processes voice queries. 3. Real-time interactive guidance delivered via audio and UI.

Postconditions	Users receive personalized guidance.
Alternative Flow	Offline tutorial guidance if Gemini is unavailable.
Exceptions	Notify users of connectivity or API issues.

4.1.18 Query AI Assistant (Gemini Integration)

ID	**UC-18**
Name	Query AI Assistant (Gemini Integration)
Actors	User
Description	Users ask environment-specific questions answered by Gemini AI.
Assumptions	Gemini API accessible, internet connectivity active.
Trigger	Voice command: "Ask [question]".
Preconditions	App active, microphone permissions granted.
Normal Flow	1. User issues "Ask [question]". 2. Query processed via Gemini AI. 3. App provides verbal responses.
Postconditions	User receives context-specific verbal feedback.
Alternative Flow	Notification if Gemini unavailable.
Exceptions	Notify the user clearly if processing fails.

4.1.19 Switch Camera (Front/Back)

ID	**UC-19**
Name	Switch Camera (Front/Back)
Actors	User
Description	User toggles between front and back camera views.
Assumptions	Both cameras are functional.

Trigger	Tap the "Switch Camera" button.
Preconditions	Camera permissions granted.
Normal Flow	1. Tap the switch button. 2. Immediate camera toggle.
Postconditions	Camera switched successfully.
Alternative Flow	Notify the user if the camera is unavailable.
Exceptions	Notify clearly if unable to switch cameras.

4.2 Use Case Diagram

The Use Case Diagram for the Object Detection Application provides a high-level visual representation of how different users and external systems interact with the application. This diagram helps in understanding the key functionalities, user interactions, and system boundaries.

The system is designed primarily for users who rely on real-time object detection, audio feedback, and customization options to enhance their experience. Additionally, the application integrates with Siri for voice commands and iCloud for backup and settings synchronization.

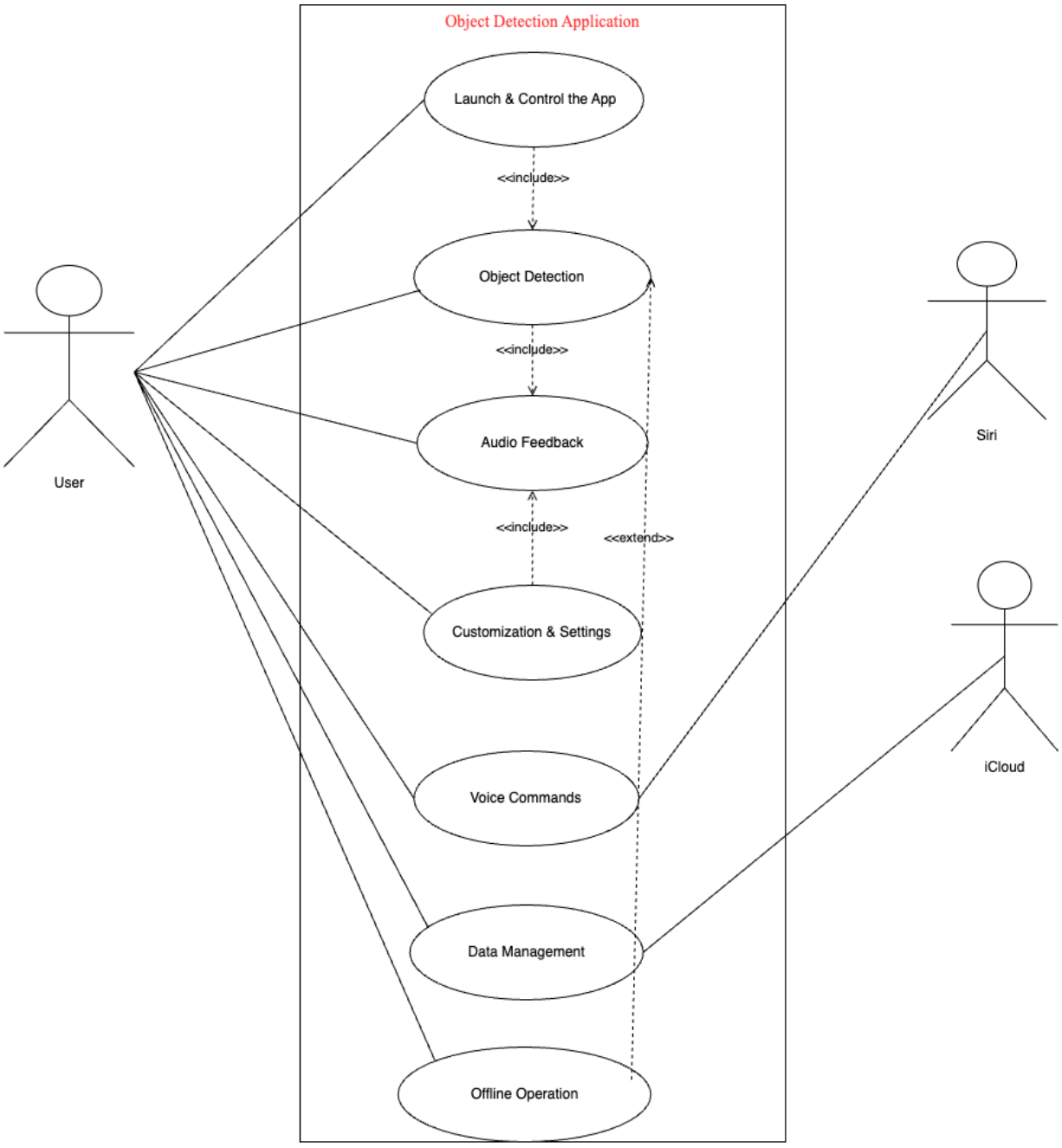
Actors in the System:

- User – The primary actor who interacts with the app to detect objects, customize settings, and use advanced features like scene exploration and haptic feedback.
- Siri – An external system that allows users to control the application through voice commands.
- iCloud – A cloud storage service used to back up user settings and preferences.

Key Use Cases:

The application offers various functionalities, including:

- Core Features: Launching and deactivating the app, initializing the camera, real-time object detection, and providing audio feedback.
- Customization: Users can configure detection settings, adjust audio feedback, enable battery optimization, and modify hazard alerts.
- Advanced Functionalities: Features like haptic feedback for close objects, scene exploration mode, and camera alignment assistance enhance user experience.
- Integration & Data Management: The app supports Siri integration for voice commands and iCloud backup for storing user preferences.



4.3 Sequence Diagrams

A sequence diagram is a vital part of system design as it provides a detailed view of the sequential interaction between actors and system components. It helps visualize the processes

involved in a user-driven feature, ensuring that every interaction is understood and accounted for during development.

This sequence diagram illustrates the interaction and flow of messages between various components of the Object Detection Application. It represents a specific use case where a User launches the app and utilizes the Object Detection feature to identify objects in their environment. The diagram highlights how the app communicates with different internal modules, such as the Object Detection Module, Audio Feedback, and Display/Results, to deliver real-time results to the user.

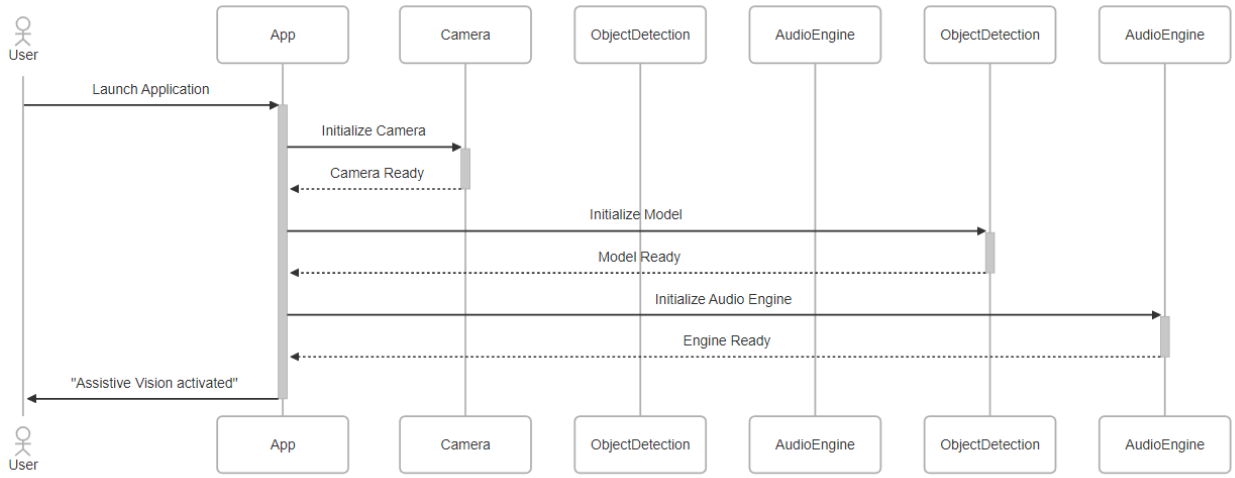
In this sequence diagram, the following main actors and objects are involved:

- Actors:
    - User: Initiates the sequence by launching the app and interacting with the app features.
    - Siri (optional): If voice commands are used, it allows the User to control the app via voice.
    - iCloud (optional): Used for synchronizing or storing data.
- Objects:
    - Object Detection App: The main application that the User interacts with.
    - Object Detection Module: The internal module is responsible for processing and detecting objects from the camera feed.
    - Audio Feedback: Provides real-time audio feedback to the User upon detection of objects.
    - UI/Display: Displays the detection results and informs the User visually.

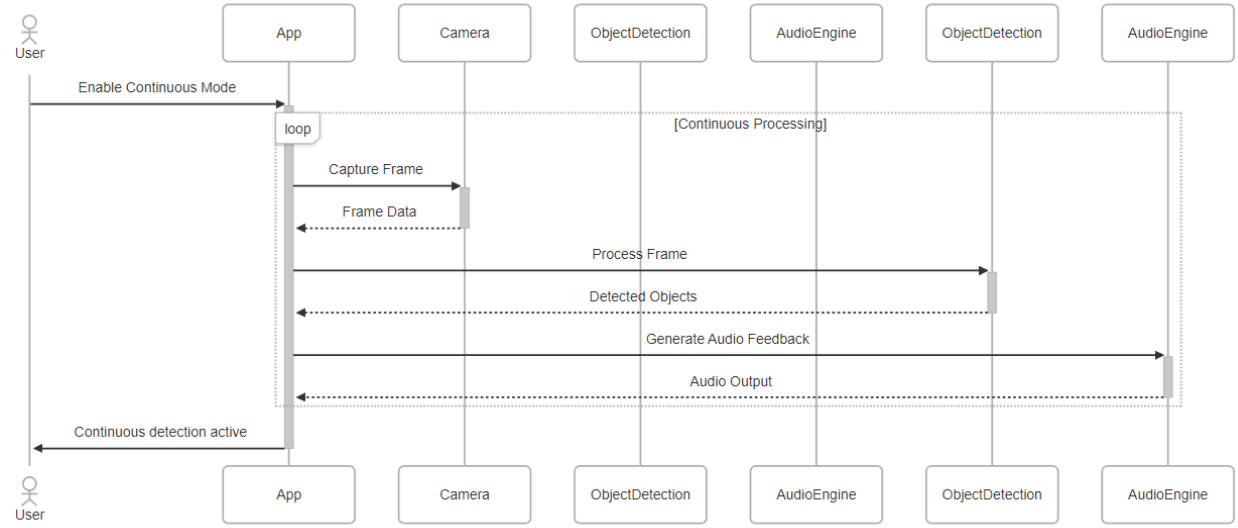
The sequence diagram illustrates the step-by-step interaction between the User and the Object Detection App. It starts with the User launching the app, which triggers the initialization of object detection, followed by the detection of objects by the Object Detection Module. Once objects are detected, the Audio Feedback provides notifications to the User, and the Display shows the results.

This sequence highlights important processes such as message passing between the Object Detection App and various internal modules, the return of feedback, and any optional flows like the use of voice commands via Siri.

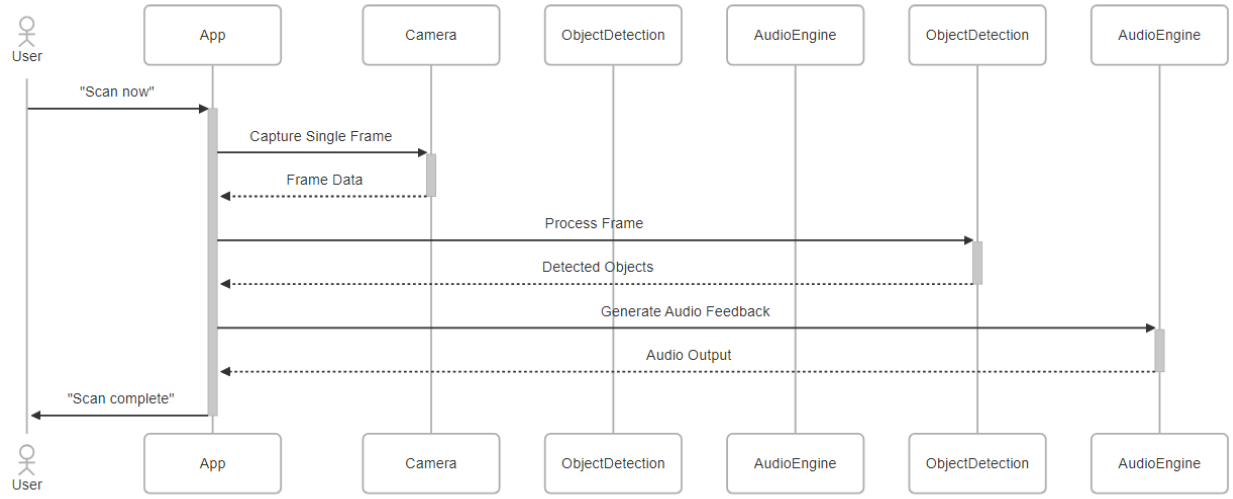
4.3.1 Launch Application



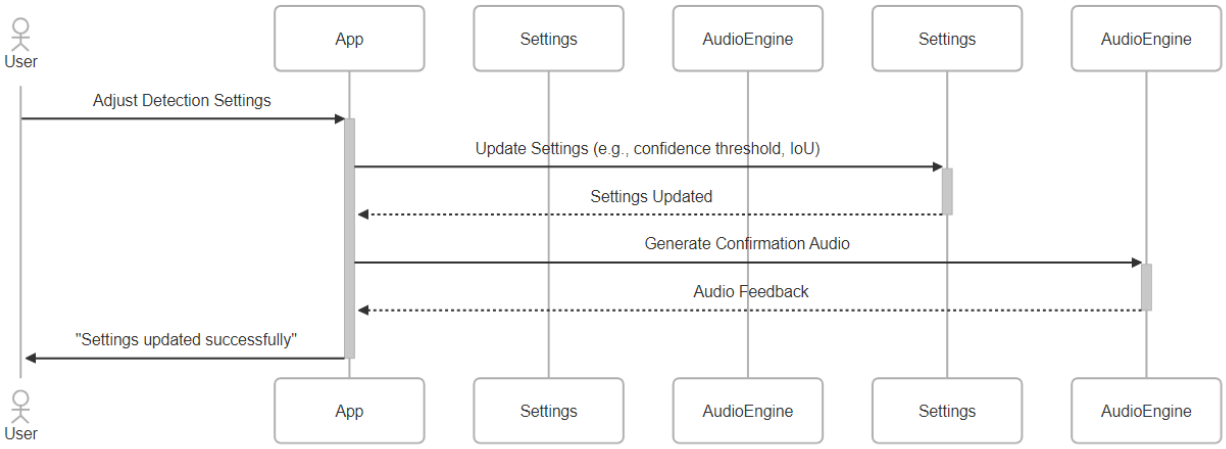
4.3.2 Object Detection Mode



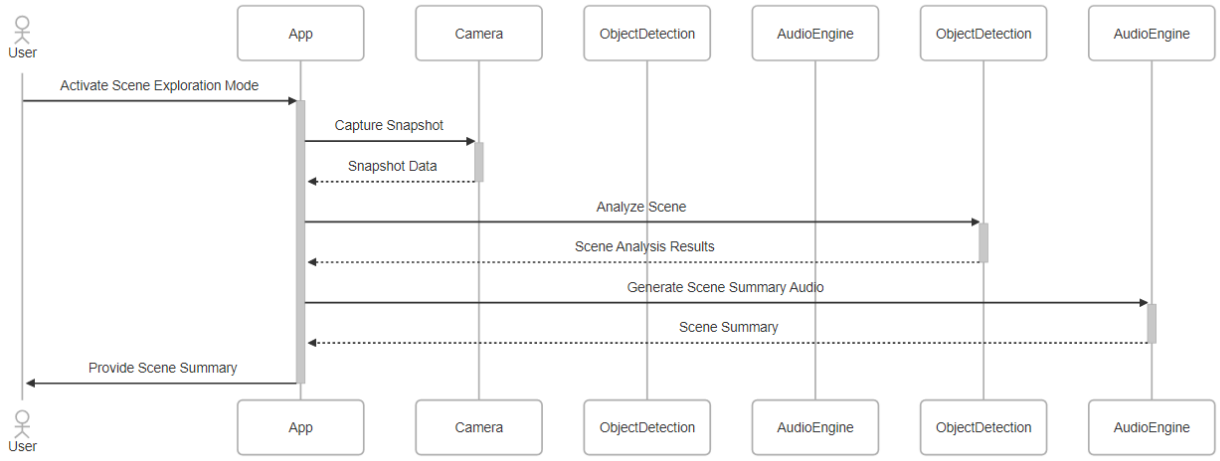
4.3.3 Object Detection in on-Demand Mode



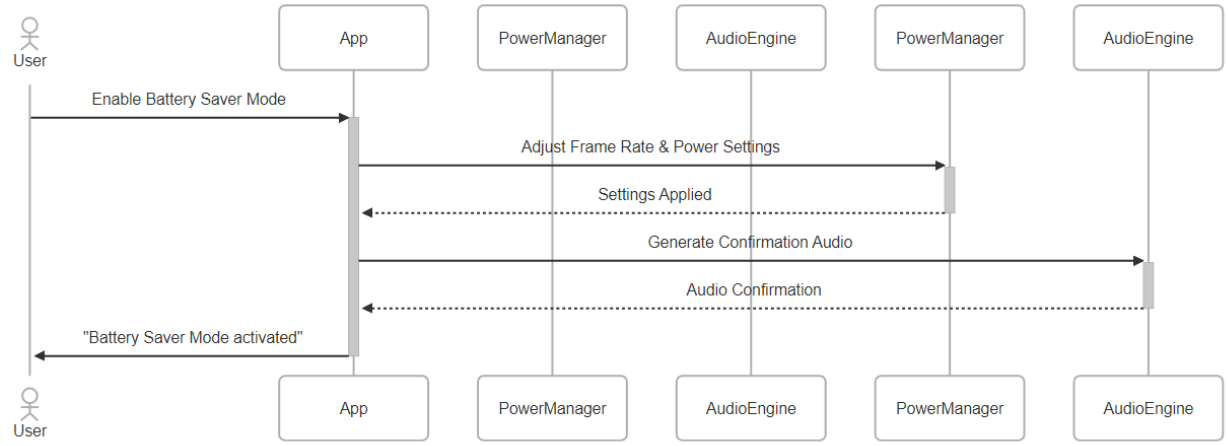
4.3.4 Adjust Object Detection Settings



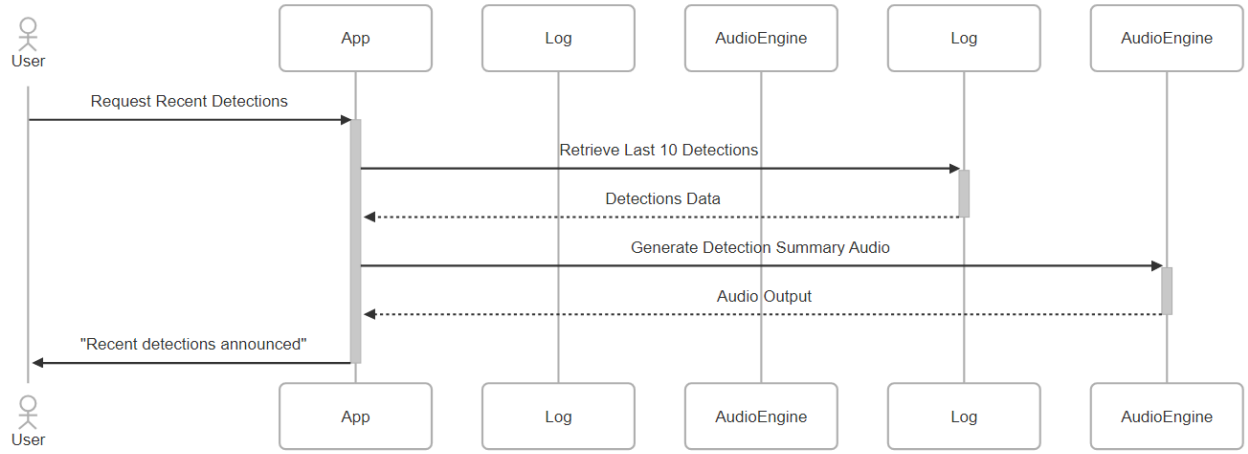
4.3.5 Scene Exploration Mode



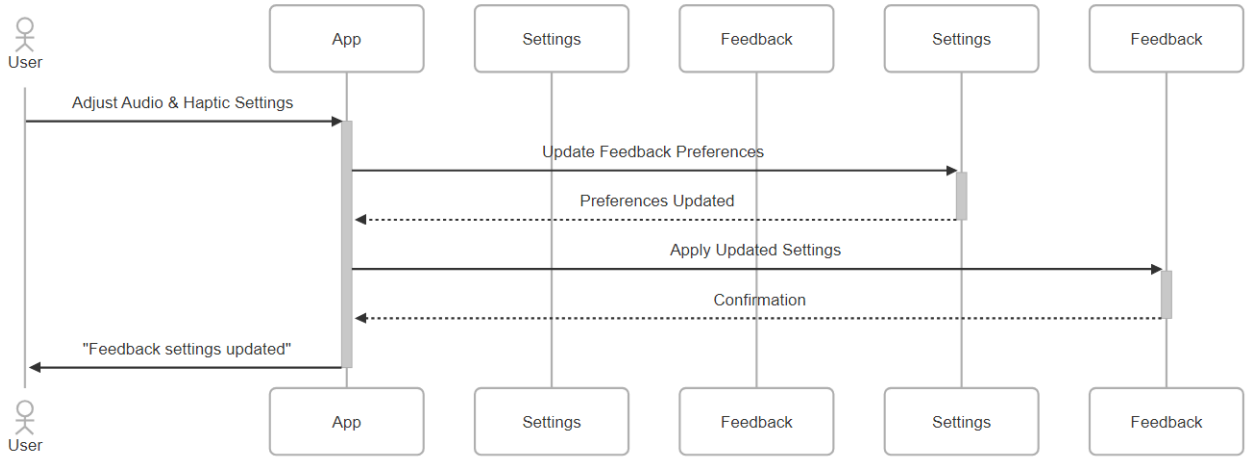
4.3.6 Enable Battery Saver Mode



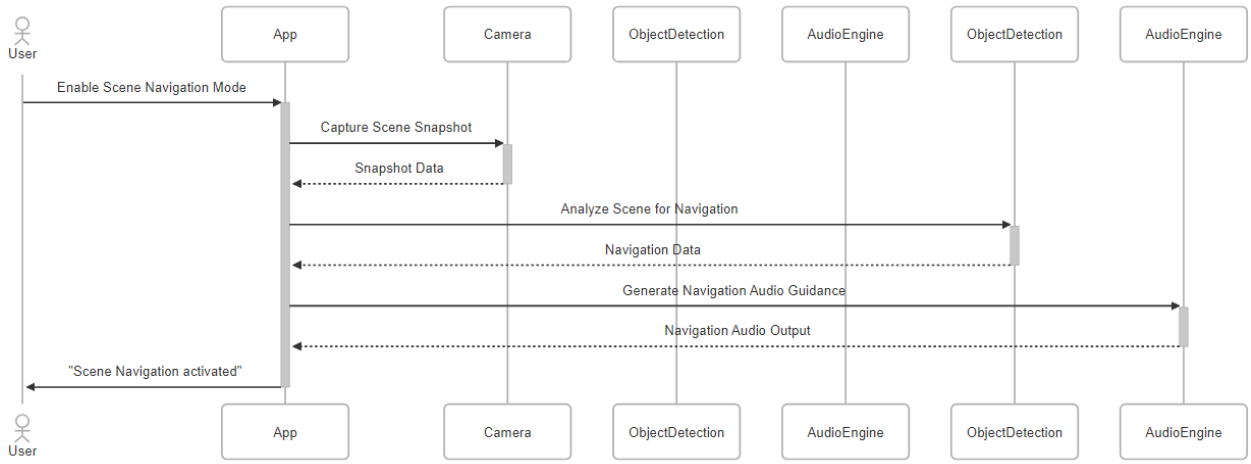
4.3.7 Review Recent Object Detections



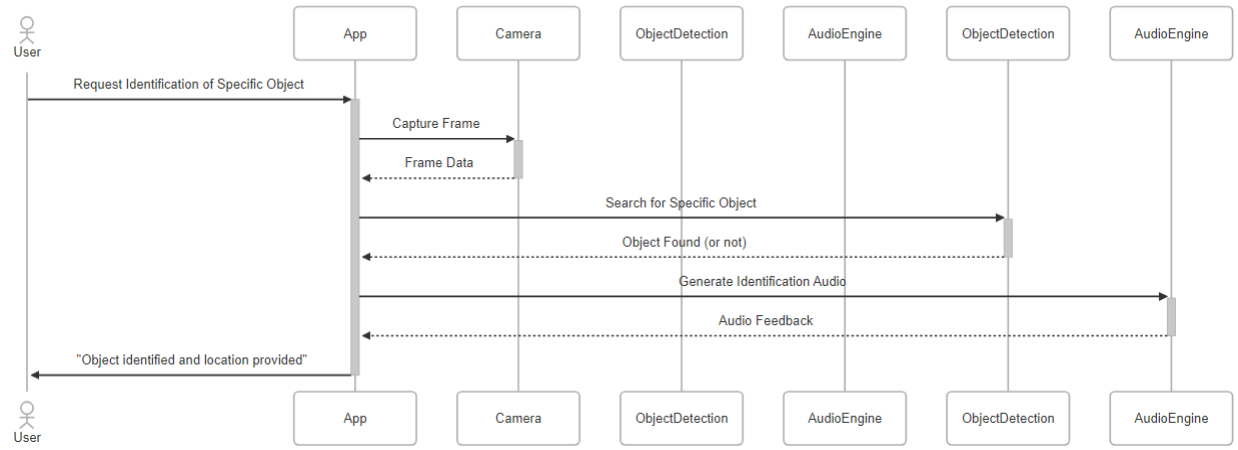
4.3.8 Configure Audio & Haptic Feedback



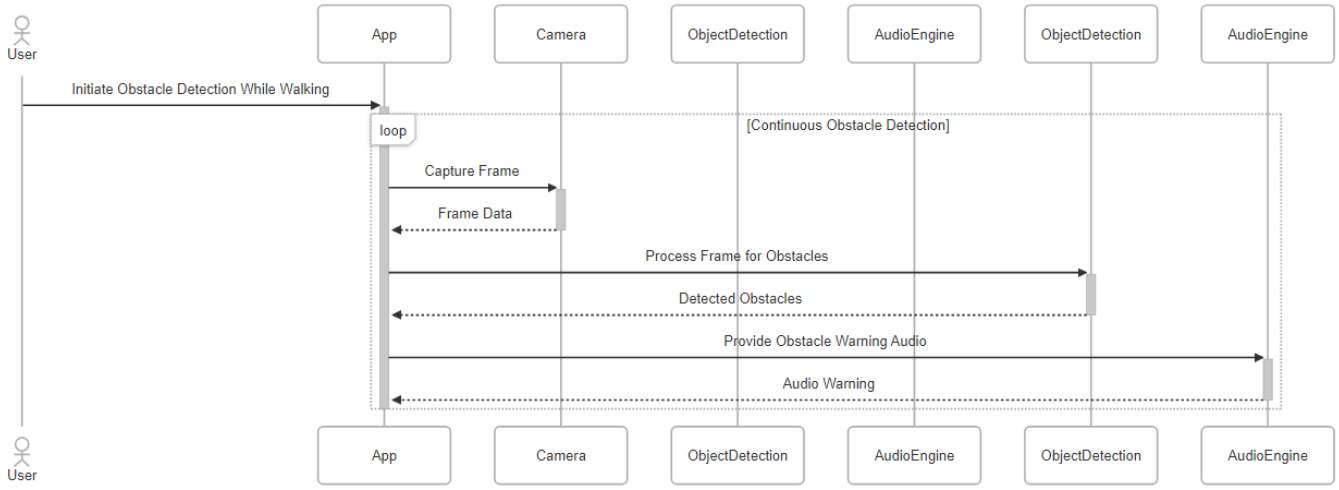
4.3.9 Enable Scene Navigation Mode



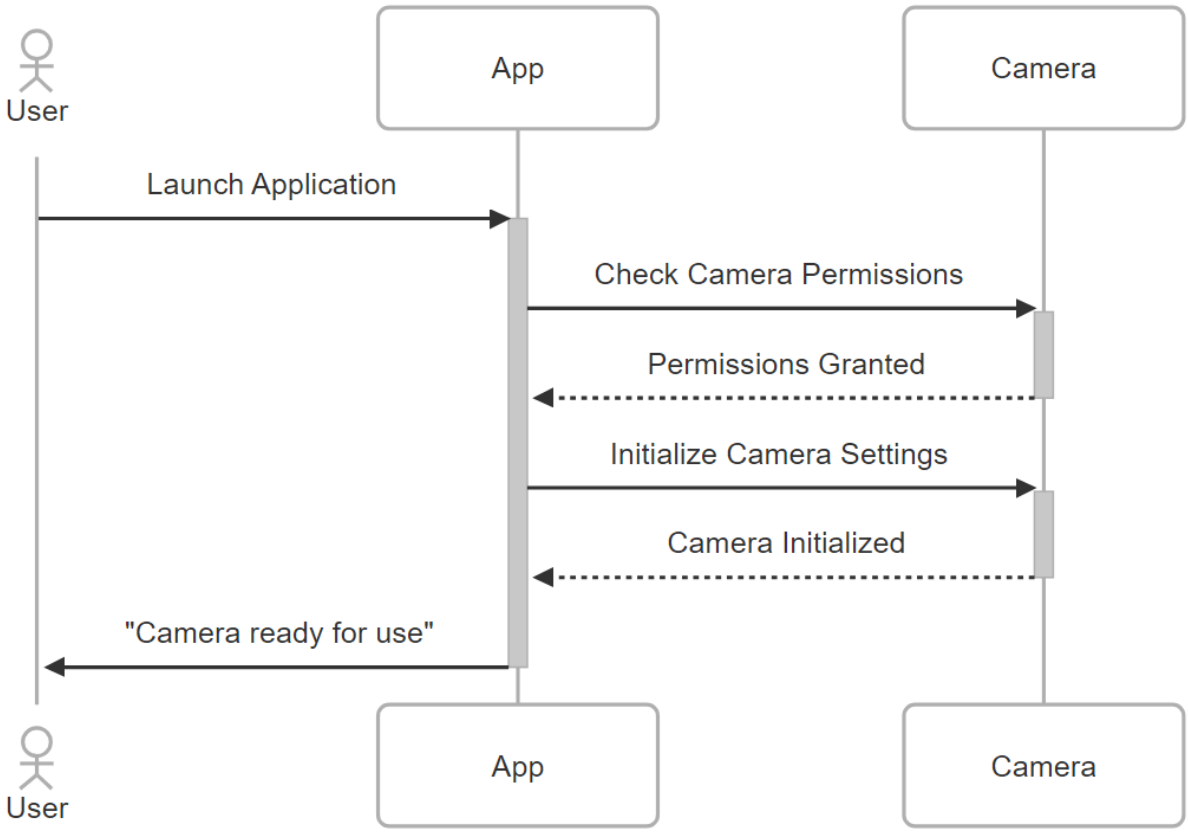
4.3.10 Identify Specific Object in View



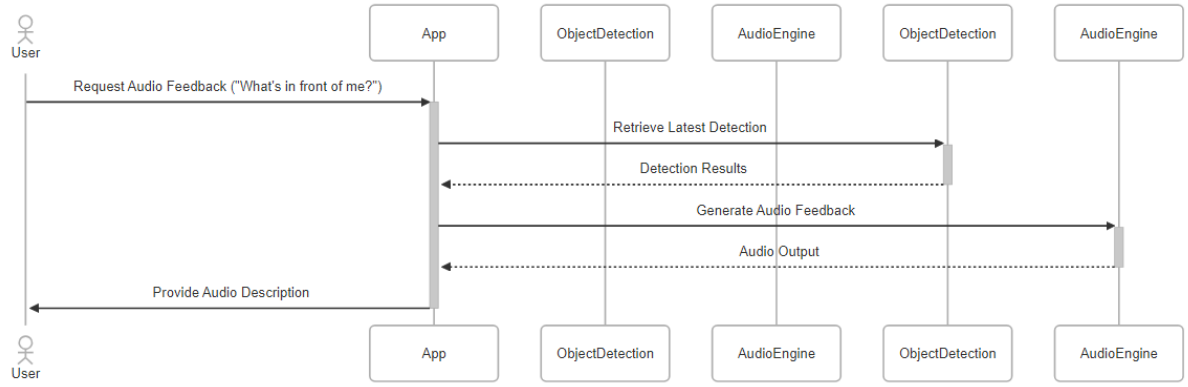
4.3.11 Detect Obstacles While Walking



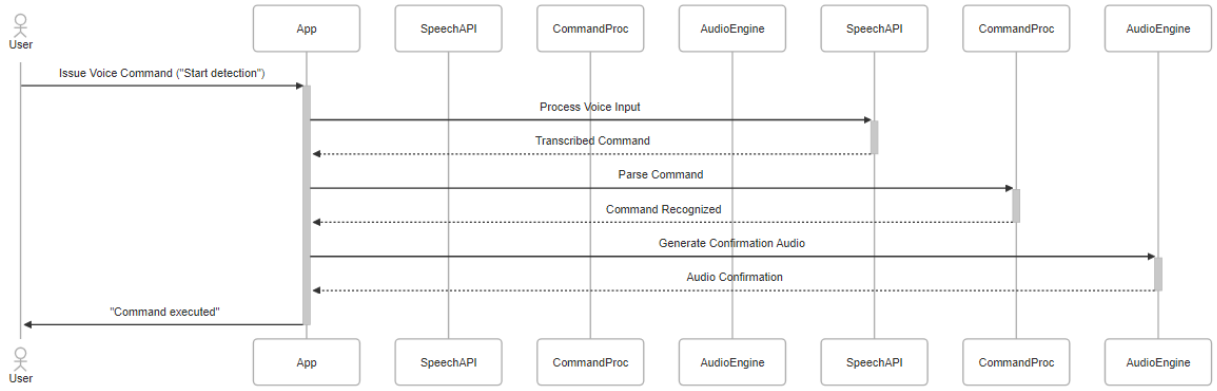
4.3.12 Camera Initialization



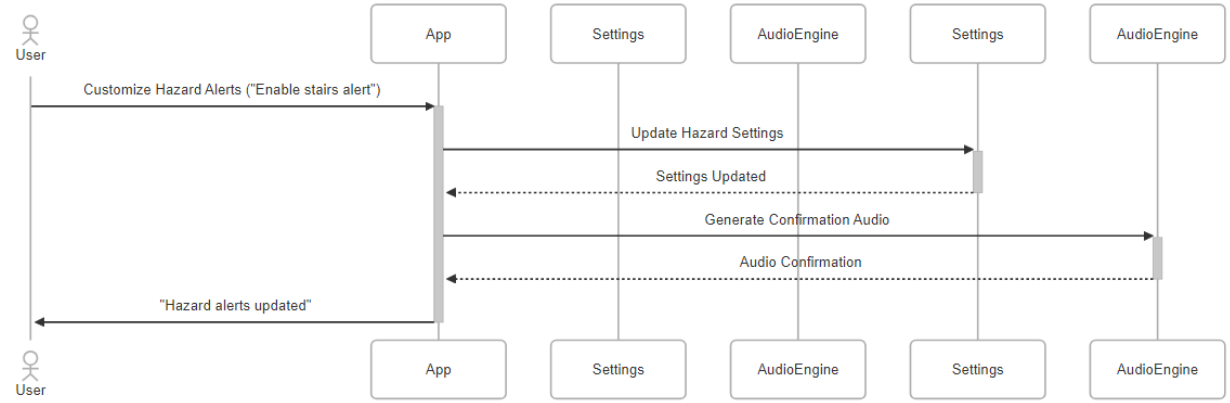
4.3.13 Provide Audio Feedback



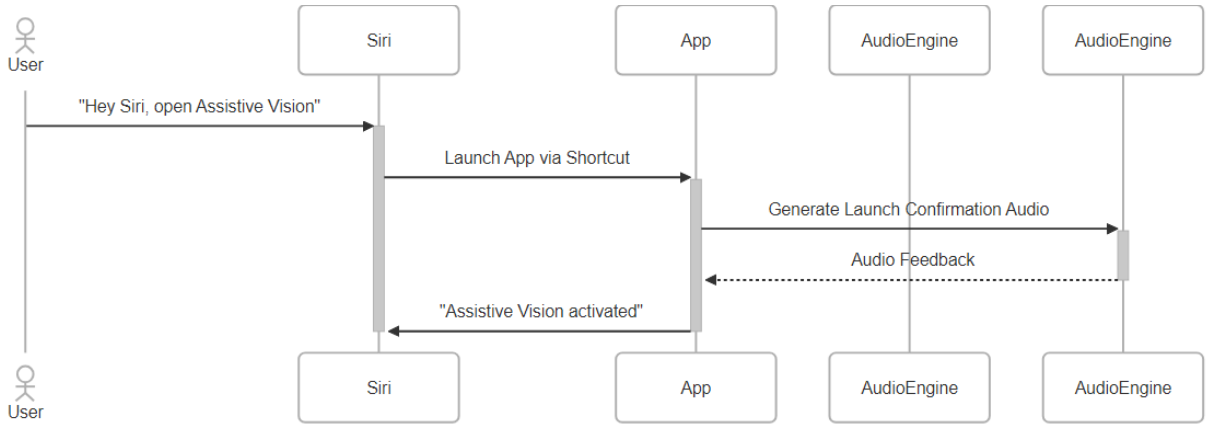
4.3.14 Voice Command Interface



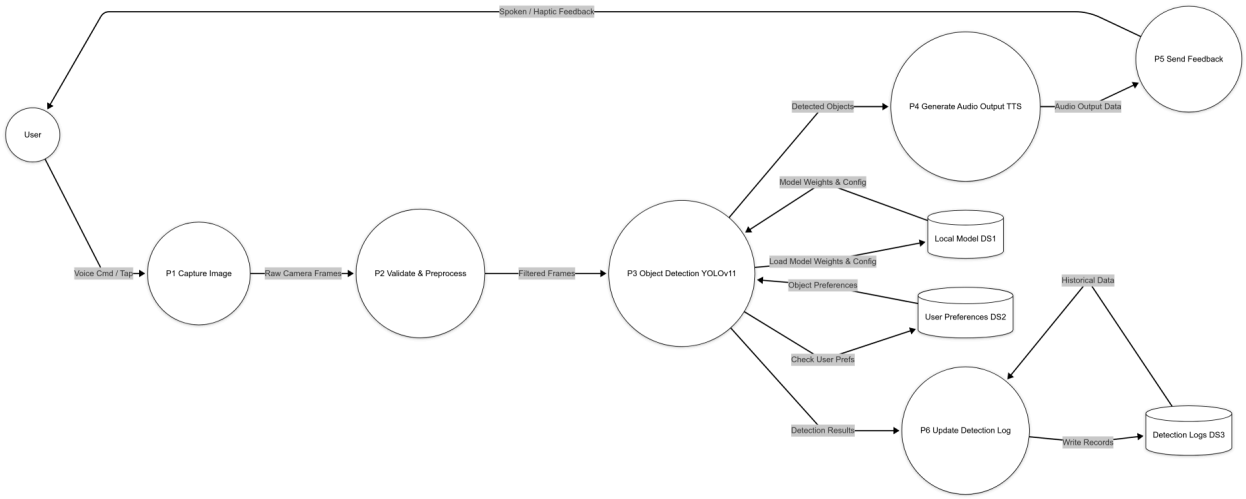
4.3.15 Hazard Alert Customization



4.3.16 Siri Shortcuts



4.4 Data Flow Diagram



A Data Flow Diagram (DFD) is a visual representation of how data moves through a system. It illustrates the flow of information between external entities, processes, and data stores. DFDs are useful for understanding system operations, analyzing workflows, and identifying areas for improvement.

Overview of the Process

1. User Interaction:
    - The user initiates the process via a tap or voice command.
    - This triggers the Capture Image (P1) process, which collects raw camera frames.
2. Preprocessing and Object Detection:
    - The Validate & Preprocess (P2) stage filters and prepares the captured frames.
    - The frames are then sent to Object Detection (P3), which uses the YOLOv11m model for identifying objects.
3. Integration with System Components:
    - The object detection model loads from the Local Model (DS1), which contains model weights and configurations.
    - User Preferences (DS2) influence object detection settings, ensuring personalized results.
    - Detection logs are updated in Detection Logs (DS3) to store historical data.
4. Generating Audio Output:
    - Detected objects are sent to Generate Audio Output (TTS) (P4) to create spoken descriptions.
    - The system then sends feedback (P5) via phone speakers or headphones.
5. Logging and Learning:
    - The system records detection events in Update Detection Log (P6).
    - The historical data helps refine detection accuracy and user preferences over time.

4.5 Database Design

This document provides a step-by-step guide to designing a relational database schema for an object detection application using a mobile phone camera. The database is structured to store and manage user interactions, detection history, settings, and analytics, ensuring efficient data retrieval and organization. By leveraging Draw.io (diagrams.net), this guide enables a systematic approach to database design, allowing for clarity in relationships, constraints, and functionalities.

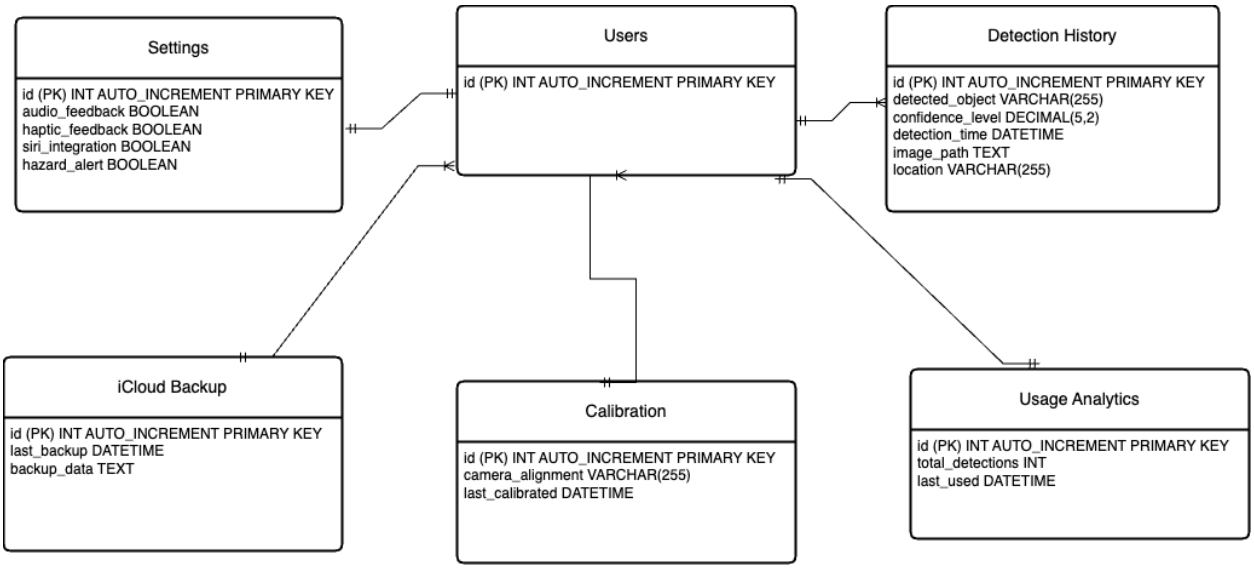
Overview of the Database Diagram

The database schema consists of six key tables:

1. Users – Stores user details, ensuring unique identification for each registered user.
2. Detection History – Maintains records of detected objects, confidence levels, and timestamps for historical tracking.
3. Settings – Saves user preferences, including detection modes, audio feedback, and Siri integration.
4. Calibration – Handles camera alignment settings to optimize detection accuracy.
5. Usage Analytics – Collects data on user activity and detection patterns for performance monitoring.
6. iCloud Backup – Ensures backup and restoration of user settings and detection records.

Each table is connected through primary keys (PKs) and foreign keys (FKs) to maintain referential integrity. The Users table serves as the central entity, linking to other tables in a one-to-many (1:N) relationship, allowing each user to have multiple detection history records, settings configurations, calibrations, and backups.

The database structure is designed to support real-time object detection, audio feedback, Siri integration, and offline operation while ensuring data security and efficient retrieval. The diagram, created in Draw.io, visually represents the schema, helping developers and database administrators understand and implement the system effectively.



4.6 Class Diagram

Overall Structure:

- The class diagram provides a detailed structural representation of the object-oriented design for the Assistive Vision iOS application.
- It defines the key classes, their attributes, and the relationships among them to satisfy both functional and non-functional requirements.

Core Controller – ViewController:

- Serves as the primary controller for processing camera input, performing object detection, and delivering audio feedback.
- Integrates tightly with the VideoCapture component for handling camera operations.
- Aggregates multiple BoundingBoxView instances to display detection overlays.
- Utilizes ThreshProvider for setting detection thresholds and may optionally integrate a GenerativeModel for AI queries.

Camera Operations – VideoCapture:

- Manages the capture of video frames through the device's camera.
- Establishes and configures the camera session, ensuring proper video orientation and preview layer management.
- Communicates captured frames back to the ViewController via the VideoCaptureDelegate protocol.

Detection Overlays – BoundingBoxView:

- Provides a visual overlay for detected objects by managing shape and text layers.
- Is aggregated by the ViewController to render real-time detection results on the video preview.

Configuration – ThreshProvider:

- Acts as a feature provider that supplies configurable thresholds (e.g., confidence and IoU) for the CoreML detection models.
- Enables dynamic updates to detection parameters based on user preferences.

Global Settings – SharedData:

- Implements a singleton pattern to store global app settings and user preferences such as detection thresholds, scanning modes, language, and haptic feedback.
- Is accessed by multiple view controllers (TutorialViewController, SettingsViewController, FAQViewController) to maintain consistency across the application.

Supplementary Components:

- **AppDelegate:** Manages the application lifecycle, initializing essential services (e.g., keeping the device awake, battery monitoring) and launching the ViewController.
- **TutorialViewController, SettingsViewController, FAQViewController:**

Assistive Vision

- Offer specialized interfaces for user interactions, such as AI-guided tutorials, configuration adjustments via voice commands, and frequently asked questions.

- Each accesses SharedData to ensure global settings are reflected consistently across the app.

- **GenerativeModel:** Supports AI-based content generation for dynamic responses, integrated optionally within the ViewController and TutorialViewController.

- **VideoCaptureDelegate:** A protocol defining the callback for delivering video frame data from VideoCapture to the ViewController.

4.7 Application Program Interfaces

In our project, we utilize several Apple-provided APIs to enhance functionality, optimize performance, and streamline development. These APIs are accessible through the Apple Developer Portal, allowing us to leverage powerful system frameworks efficiently. Below is an overview of the key APIs we use and their roles within our application:

4.7.1 UIKit

UIKit is the primary framework for building and managing our application's user interface. It provides essential components such as views, buttons, labels, navigation controllers, and animations, ensuring a seamless and intuitive user experience.

- Why we use it: It allows us to create a dynamic and interactive interface while maintaining consistency with iOS design guidelines.
- Key functionalities we utilize: UI layout, event handling, view controllers, and animation management.

4.7.2 Foundation

Foundation provides fundamental data structures and utilities, including strings, collections, date handling, and networking.

- Why we use it: It helps us manage essential data types, localization, and efficient memory management.
- Key functionalities we utilize: String manipulation, data persistence, JSON parsing, and file handling.

4.7.3 AVFoundation

AVFoundation is a robust framework for working with audiovisual media, including playback, recording, and processing.

- Why we use it: We rely on AVFoundation for handling audio and video processing, enabling features such as media recording and playback.
- Key functionalities we utilize: Audio and video capture, media playback, and real-time audio processing.

4.7.4 CoreVideo

CoreVideo is designed for high-performance video processing and efficient pixel buffer management.

- Why we use it: It allows us to work with raw video frames efficiently, which is essential for media processing and integration with CoreML and Vision.
- Key functionalities we utilize: Pixel buffer manipulation, frame rendering, and real-time video analysis.

4.7.5 CoreML

CoreML is Apple's machine learning framework that enables on-device AI processing.

- Why we use it: We leverage CoreML for real-time machine learning tasks, such as image recognition and speech processing.
- Key functionalities we utilize: Running pre-trained ML models, integrating with Vision, and performing real-time inference.

4.7.6 CoreMedia

CoreMedia provides low-level media processing tools, working closely with AVFoundation for time-based audiovisual data management.

- Why we use it: It ensures synchronization and efficient handling of time-based media in our application.
- Key functionalities we utilize: Timecode management, sample buffering, and synchronization of audio/video streams.

4.7.7 Vision

Vision is a powerful computer vision framework that simplifies image analysis tasks such as object detection, face recognition, and text recognition.

- Why we use it: We utilize Vision for image and video-based analysis, enhancing user interactions through AI-powered visual recognition.
- Key functionalities we utilize: Face detection, text recognition (OCR), object tracking, and image classification.

4.7.8 Speech

The Speech framework enables real-time speech recognition and transcription.

- Why we use it: It allows us to implement voice-driven interactions and transcribe spoken content within our application.
- Key functionalities we utilize: Real-time speech recognition, dictation processing, and voice command interpretation.

4.8 User Interface Design

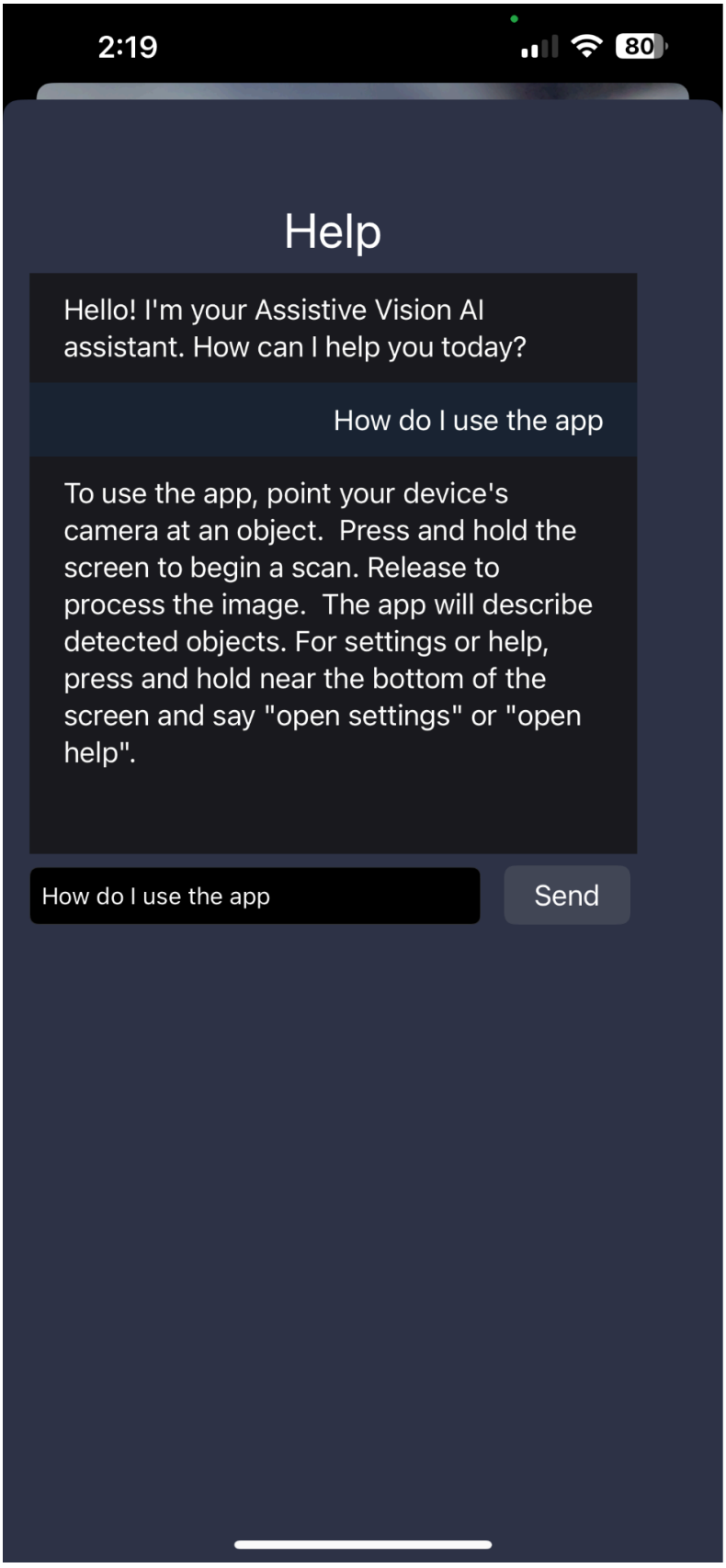
The Assistive Vision iOS application follows a structured and accessible user interface (UI) design, optimized for visually impaired users. The design prioritizes clarity, ease of navigation, and high contrast to enhance usability. It adheres to Apple's Human Interface Guidelines, incorporating voice commands, haptic feedback, and large, readable text to ensure accessibility. The primary color scheme consists of a dark navy blue background, providing strong contrast against white and light gray text for readability. Purple accents highlight interactive elements such as buttons and navigation indicators, while yellow bounding boxes emphasize detected objects in the camera view. Red indicators are used sparingly for warnings like low battery notifications.

The application's UI includes several key components to enhance user experience. The Launch Screen displays the Assistive Vision logo against a dark background for brand recognition. The Object Detection Interface allows users to choose between Continuous Mode and On-Demand Mode, with detected objects highlighted in yellow and described through audio feedback. The Detection Log provides a history of detected objects with timestamps and frequency counts for reference. A dedicated FAQ & Tutorials section offers voice-guided assistance, ensuring easy onboarding and troubleshooting. The Settings Panel enables users to customize preferences such as confidence thresholds, haptic feedback, audio settings, and scanning rate, making the experience highly adaptable. The UI is designed with a minimalist yet functional approach, ensuring that visually impaired users can interact seamlessly through high-contrast visuals, large buttons, intuitive controls, and voice commands for hands-free navigation.

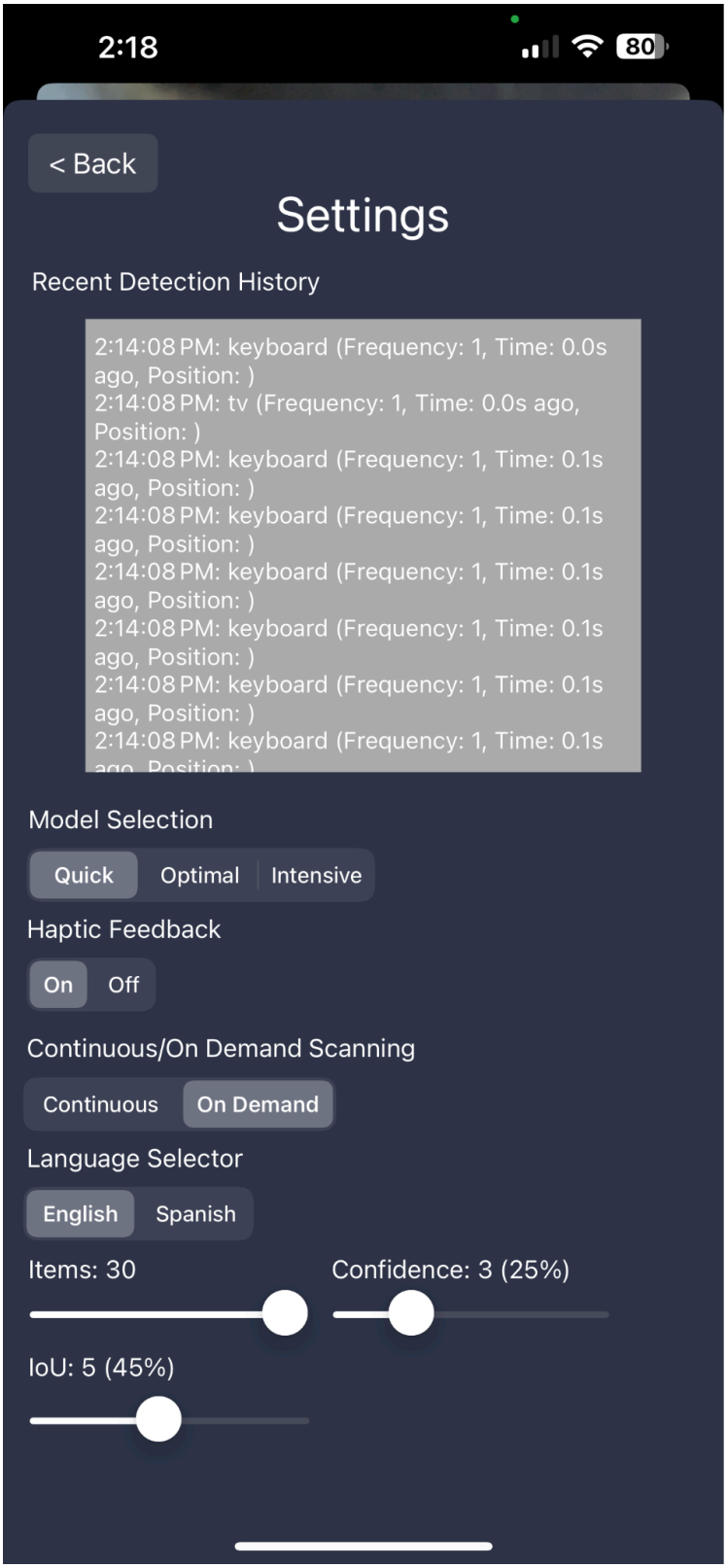
4.8.1 Launch Screen

4.8.2 Main Interface/Detection Mode

4.8.3 Help


2:19
80
Help
Hello! I'm your Assistive Vision AI assistant. How can I help you today?
How do I use the app
To use the app, point your device's camera at an object. Press and hold the screen to begin a scan. Release to process the image. The app will describe detected objects. For settings or help, press and hold near the bottom of the screen and say "open settings" or "open help".
How do I use the app
Send

4.8.5 Settings/Detection History



Appendix A: Key Terms

The following table provides definitions for terms relevant to this document.

Term	Definition
Assistive Vision	The iOS application designed to aid visually impaired users by providing real-time object detection, audio feedback, and accessible interaction methods.
Object Detection	The process by which the application analyzes camera input using a machine learning model (YOLOv11m) to identify and locate objects in the environment.
Continuous Mode	A scanning mode in which the app continuously captures camera frames and processes them for real-time object detection and audio feedback.
On-Demand Mode	A scanning mode that allows users to manually trigger object detection via voice command, resulting in a single analysis of the camera feed.
Scene Exploration Mode	A mode where the app captures a static snapshot of the environment, then provides a detailed audio summary of the scene for enhanced spatial awareness.
Voice Command Interface	The system that allows users to control the application hands-free by issuing spoken commands, integrated with iOS's speech recognition and Siri Shortcuts.
Haptic Feedback	Tactile responses provided by the device (such as vibrations) to complement audio feedback and enhance user awareness of events or alerts.
Battery Saver Mode	A mode designed to reduce power consumption by lowering the frame processing rate and optimizing resource usage when battery levels are low.

Siri Integration	The feature that allows the app to be controlled via Siri, enabling users to execute commands using voice shortcuts for a hands-free experience.
iCloud Backup	A mechanism for securely storing and synchronizing user settings and detection history using Apple's iCloud service, ensuring data persistence across devices.
CoreML	Apple’s machine learning framework used for on-device processing, enabling efficient and private object detection without reliance on cloud-based services.
Apple Neural Engine	Dedicated hardware within iOS devices that accelerates machine learning tasks, ensuring fast and efficient object detection performance.
Model-View-Controller (MVC)	A design pattern that separates the application into three interconnected components—Model (data processing), View (user interface), and Controller (business logic)—to improve organization and maintainability.
WCAG	The Web Content Accessibility Guidelines that inform the app’s design decisions, ensuring the interface is accessible to users with visual impairments.
AVFoundation	An iOS framework used for handling multimedia tasks, including camera access and text-to-speech functionalities for generating audio feedback.
Detection History	A feature that logs details of previously detected objects (including timestamps and confidence levels) for review by the user.
Calibration Manager	The component responsible for adjusting and aligning the camera settings to optimize

	detection accuracy.
Usage Analytics	Data collected on user interactions and detection events, used to monitor system performance and improve user experience over time.

References

[1] Apple Developer Documentation, “Swift,” https://developer.apple.com/documentation/swift/

[2] Apple Developer Documentation, “AVFoundation Framework,” https://developer.apple.com/documentation/avfoundation

[3] Mermaid Documentation, “Mermaid.js Documentation,” https://mermaid.js.org/intro/

[4] Ultralytics Documentation, “Ultralytics Documentation,” https://docs.ultralytics.com/