CMPE 492
Senior Project 2

Coffee Machine Assistance for the Visually Impaired:
A Computer Vision-Based Mobile Application

Low-Level Design Report

Project Supervisor

Prof. Dr. Hakkı Gökhan İlk

Project Team Members

Murad Huseynov

Bedir Esen

Hazem Moustafa

Alperen Dalgıç

Table of Contents

1. Introduction 4

1.1 Purpose 4

1.2 Scope 4

1.2.1 Holistic Testing Approach 4

1.2.2 Accessibility and Computer Vision Integration 4

2. Defining Scope 4

2.1 Application Overview 4

2.2 Understanding the Application 5

2.2.1 Core Functionality and User Flow 5

2.2.2 System Modules and Components 5

2.3 Testing Needs Assessment 6

2.3.1 Number of Testers 6

2.3.2 Estimated Testing Time 6

2.3.3 Resources 6

3. Risk Analysis 7

3.1 Prioritization of Testing Efforts 7

3.2 Identification of High-Risk Areas 7

3.3 Response to High-Risk Scenarios 8

4. Scheduling 9

4.1 Development Stages 9

4.2 Test Timeline 10

5. Test Approach 11

5.1 Testing Types and Methodologies 11

5.2 Test Case Creation and Management 14

5.3 Utilization of Testing Tools 14

6. Defect Management 15

6.1 Bug Reporting 15
6.2 Bug Resolution Process 16
7. Conclusion 17
8. Appendix 17
A. References 17
B. Glossary of Terms 18

1. Introduction

1.1 Purpose

We, the development team of the "Coffee Machine Assistance for the Visually Impaired" project, present this comprehensive Test Plan Report for our innovative assistive technology mobile application. The purpose of this document is to articulate the meticulous approach, methodologies, and strategies employed in the testing phase of our computer vision-based accessibility solution.

This application represents a critical intersection of assistive technology, computer vision, and mobile development, requiring rigorous testing to ensure reliability, accuracy, and accessibility for visually impaired users operating coffee machines independently.

1.2 Scope

1.2.1 Holistic Testing Approach

This test plan encompasses an all-encompassing evaluation of the mobile application, ensuring seamless integration of computer vision detection, voice guidance, accessibility features, and user interface design. It delves into the identification of features, testing methodologies, and risk analysis associated with our cross-platform React Native application.

1.2.2 Accessibility and Computer Vision Integration

The scope extends to the validation of the EfficientDet-based object detection model's accuracy, the effectiveness of sequential voice guidance, compatibility with native screen readers (TalkBack and VoiceOver), and adherence to WCAG 2.1 (AA Level) accessibility standards. We focus on evaluating the complete detection-to-guidance loop to ensure a safe, reliable, and empowering experience for visually impaired users.

2. Defining Scope

2.1 Application Overview

The "Coffee Machine Assistance for the Visually Impaired" is a cross-platform mobile application that combines computer vision technology with sequential voice guidance to enable visually impaired individuals to independently operate a specific coffee machine model. This assistive technology solution is designed to provide real-time, step-by-step instructions through an accessible interface while maintaining complete offline functionality and user privacy.

Key Features:

- Real-time coffee machine component detection using EfficientDet
- Sequential state machine-based voice guidance
- Accessibility-first UI design (WCAG 2.1 AA compliant)
- Cross-platform support (Android 7.0+, iOS 13+)
- Native TTS integration (Android TTS and iOS VoiceOver)

- Offline functionality with local data storage
- User-customizable settings (speech rate, haptic feedback, instruction detail)

Target Audience:
- Visually impaired individuals seeking independent coffee-making capabilities
- Caregivers and assistive technology specialists supporting visually impaired users
- Accessibility advocates and researchers in assistive technology

Platforms:
- Android (7.0 and above)
- iOS (13 and above)

2.2 Understanding the Application

2.2.1 Core Functionality and User Flow

The application guides users through a structured coffee-making process via a state machine architecture. Users progress through sequential steps:

1. **Initialization**: User launches the app and navigates to the camera screen
2. **Machine Localization**: The CV system detects the coffee machine in the camera frame
3. **Component Detection**: The system identifies specific machine components (power button, water reservoir, control panel, etc.)
4. **Sequential Guidance**: Voice instructions guide the user through each operational step
5. **Validation**: Each action is verified through CV detection before advancing
6. **Error Handling**: Incorrect actions trigger contextual correction instructions
7. **Completion**: Final confirmation when coffee is ready

The entire process is designed to be hands-free and auditory-first, with haptic feedback providing additional confirmation.

2.2.2 System Modules and Components

The application is structured into four primary modules:

User Interface Module:
- React Native-based accessible interface
- Screen reader compatibility (TalkBack/VoiceOver)
- Settings management (UserPreferences class)
- Touch target optimization (minimum 48×48dp)

Input Module:
- Camera permission handling (CameraManager)
- Interval-based frame capture (1-2 second intervals)
- Frame data structure with timestamp metadata

Computer Vision Module:

- EfficientDet TFLite model integration (CoffeeMachineDetector)
- Object detection with confidence thresholding
- Detection result processing (DetectedObject, DetectionResult)

Voice Guidance & Logic Module:

- State machine controller (StateController)
- Native TTS integration (VoiceGuidance)
- Sequential brewing steps (BrewingStep)
- Error handling and recovery (ErrorHandler)

2.3 Testing Needs Assessment

2.3.1 Number of Testers

Given the specialized nature of this assistive technology application, we estimate that **10-15 testers** will be required across two distinct groups:

Technical Testers (3-5 individuals):

- Mobile developers with React Native experience
- Computer vision specialists
- Accessibility testing experts

User Testers (10-15 visually impaired individuals):

- Diverse age groups (18-60+)
- Varying degrees of visual impairment
- Different levels of technology experience
- Various prior coffee machine operation experience

This dual approach ensures both technical robustness and real-world usability validation.

2.3.2 Estimated Testing Time

The testing phase is expected to span **12-16 weeks** to ensure thorough evaluation of all features, modules, and accessibility components:

- **Weeks 1-4**: Unit and integration testing (technical team)
- **Weeks 5-8**: CV model validation and performance testing
- **Weeks 9-10**: Initial user testing (50% milestone)
- **Weeks 11-14**: Accessibility compliance and refinement
- **Weeks 15-16**: Final user acceptance testing (90% milestone)

This timeframe includes iterative testing cycles to address identified issues and incorporate user feedback.

2.3.3 Resources

Hardware Resources:

- Representative Android devices (mid-range to high-end, Android 7.0+)
- Representative iOS devices (iPhone 8 and newer, iOS 13+)

- Target coffee machine model(s) for testing
- VR headsets are NOT required (mobile-only application)
- Various lighting conditions setup (bright, dim, natural light)

Software Resources:

- React Native development environment
- TensorFlow Lite model optimization tools
- Accessibility testing tools (screen readers, contrast analyzers)
- Bug tracking system (e.g., Jira, GitHub Issues)
- Test case management tools

Human Resources:

- Partnership with vision impairment organizations for user recruitment
- Accessibility consultants for WCAG compliance validation
- Computer vision experts for model performance evaluation

3. Risk Analysis

3.1 Prioritization of Testing Efforts

The following elements significantly affect the user experience and safety of visually impaired individuals, necessitating prioritized testing:

1. Detection Accuracy (Highest Priority): Since the application's core function depends on correctly identifying coffee machine components, detection accuracy is paramount. Misidentification could lead to incorrect user actions, potential safety hazards, or user frustration. Testing must validate precision, recall, and mAP metrics across diverse environmental conditions.

2. Voice Guidance Clarity and Timing (Critical Priority): Voice instructions are the primary interface for visually impaired users. Testing must ensure instructions are clear, contextually appropriate, delivered at proper intervals, and correctly sequenced. Speech synthesis quality, instruction timing, and error recovery messages require extensive validation.

3. Accessibility Compliance (Critical Priority): Full compatibility with TalkBack (Android) and VoiceOver (iOS) is non-negotiable. Testing must verify proper focus management, semantic labeling, navigation order, and interaction patterns across all screens and states.

4. Application Performance (High Priority): Real-time performance is essential for user confidence and safety. Testing must validate detection speed, TTS latency, battery consumption, and memory usage to ensure the application remains responsive during extended use.

3.2 Identification of High-Risk Areas

Computer Vision Reliability:

- **Risk**: False positives or false negatives in component detection could guide users to

perform incorrect actions, potentially causing frustration, failed coffee attempts, or safety concerns.
- **Impact**: High - directly affects core functionality and user trust
- **Mitigation Strategy**: Extensive testing across varied lighting conditions, angles, distances, backgrounds, and occlusions

User Safety and Confidence:
- **Risk**: Unclear or mistimed voice instructions could lead to confusion, incorrect machine operation, or reduced user confidence in the application's guidance.
- **Impact**: Critical - affects user wellbeing and adoption
- **Mitigation Strategy**: User-centered testing with visually impaired participants, iterative instruction refinement based on feedback

Accessibility Barriers:
- **Risk**: Incomplete screen reader integration, poor focus management, or inadequate semantic labels could render the application unusable for target users.
- **Impact**: Critical - violates core accessibility requirements
- **Mitigation Strategy**: Compliance testing against WCAG 2.1 AA standards, extensive screen reader testing

Cross-Platform Inconsistencies:
- **Risk**: Differences in TTS quality, camera behavior, or UI rendering between Android and iOS could create inconsistent experiences.
- **Impact**: Medium to High - affects user experience consistency
- **Mitigation Strategy**: Platform-specific testing protocols, native module validation

Privacy and Data Security:
- **Risk**: Unintended data collection, storage, or transmission could violate privacy commitments and user trust.
- **Impact**: High - undermines Privacy by Design principle
- **Mitigation Strategy**: Privacy audit, verification of local-only processing, data flow analysis

Battery and Performance Degradation:
- **Risk**: Excessive battery consumption or performance degradation during extended use could prevent completion of coffee-making tasks.
- **Impact**: Medium - affects practical usability
- **Mitigation Strategy**: Extended operation testing, thermal and battery monitoring

3.3 Response to High-Risk Scenarios

Structured Communication Protocol: Test results and high-priority issues must be immediately communicated to the development team through established channels. Critical

bugs affecting safety or core functionality trigger immediate response protocols.

Regular Review Meetings: Weekly test review meetings will assess progress against high-risk areas, discuss identified issues, and adjust testing priorities as needed. Stakeholders include developers, testers, and accessibility consultants.

Iterative Refinement Process: When high-risk issues are identified:

1. **Immediate Documentation**: Detailed bug report with reproduction steps
2. **Impact Assessment**: Evaluation of severity and user impact
3. **Prioritized Resolution**: High-risk issues escalated for immediate fixing
4. **Verification Testing**: Thorough regression testing after resolution
5. **User Validation**: Confirmation with target users when appropriate

Contingency Planning: For scenarios where high-risk issues cannot be immediately resolved, fallback strategies include:

- Simplified instruction modes with additional confirmation steps
- Warning messages for known challenging conditions
- Manual override options for experienced users
- Phased release targeting controlled environments first

4. Scheduling

4.1 Development Stages

The development and testing process consists of four main stages aligned with iterative refinement:

Pre-Alpha:

Objective: Establish core module functionality and validate initial CV model performance.

Testing Activities:

- Unit testing of individual classes (CameraManager, StateController, VoiceGuidance)
- Initial CV model validation on controlled dataset
- Basic camera-to-detection pipeline verification
- Preliminary TTS integration testing

Alpha:

Objective: Integrate all modules, refine CV accuracy, and conduct initial user testing with technical team members.

Testing Activities:

- Integration testing of module interfaces
- CV model performance evaluation across diverse conditions
- Voice guidance sequence validation
- Cross-platform compatibility verification
- Internal accessibility testing
- Bug identification and prioritization

Beta:
Objective: Validate application with target users, refine based on feedback, and ensure accessibility compliance.
Testing Activities:

- User acceptance testing with 10-15 visually impaired participants
- WCAG 2.1 AA compliance verification
- Performance and battery consumption testing
- Extensive regression testing
- Error handling and recovery validation
- Edge case identification and resolution

Final:
Objective: Ensure production readiness, stability, and complete documentation.
Testing Activities:

- Final user validation testing
- Comprehensive accessibility audit
- Performance benchmarking on target devices
- Security and privacy audit
- Documentation review and completion
- Release candidate verification

4.2 Test Timeline

Phase	Start Date	End Date	Milestones	Key Deliverables
Pre-Alpha	Week 1	Week 4	- Core module development - Initial CV model training	- Unit test results - CV baseline accuracy metrics - Module interface documentation
Alpha	Week 5	Week 8	- Module integration complete - Internal testing	- Integration test report - CV performance analysis - Identified bug list with priorities - Cross-platform compatibility matrix
Beta	Week 9	Week 14	- User testing at 50% (Week 10) - Accessibility compliance (Week 12) - User testing at 90% (Week 14)	- User testing feedback reports - WCAG 2.1 compliance certification - Performance benchmark results

Phase	Start Date	End Date	Milestones	Key Deliverables
Final	Week 15	Week 16	- Final user acceptance - Production release preparation	- Refined bug resolution status - Final test report - User acceptance sign-off - Privacy and security audit results - Release documentation

5. Test Approach

5.1 Testing Types and Methodologies

Functional Testing:

Computer Vision Module:

- Validate detection accuracy (precision, recall, mAP) for each coffee machine component
- Test confidence threshold effectiveness
- Verify bounding box accuracy and component localization
- Validate detection under varied lighting conditions, angles, and distances
- Test handling of partial occlusions and background variations

Voice Guidance System:

- Verify correct instruction generation for each brewing step
- Validate state machine transitions based on detection results
- Test error handling and recovery instruction generation
- Verify instruction timing and sequencing
- Test TTS output clarity and pronunciation

User Interface:

- Validate navigation flow across all screens
- Test settings persistence and retrieval
- Verify haptic feedback patterns
- Test camera permission handling
- Validate user preference application

State Machine Logic:

- Verify correct state transitions
- Test validation logic for user actions
- Verify step advancement conditions
- Test handling of incorrect detections
- Validate completion state and reset functionality

Accessibility Testing:

Screen Reader Compatibility:

- Comprehensive TalkBack (Android) testing across all screens
- Complete VoiceOver (iOS) testing across all screens
- Verify accessibility labels, hints, and roles
- Test focus order and navigation logic
- Validate dynamic content announcements

WCAG 2.1 AA Compliance:

- Verify contrast ratios (minimum 4.5:1)
- Test touch target sizes (minimum 48×48dp)
- Validate keyboard/gesture navigation
- Test semantic HTML/markup structure
- Verify perceivable, operable, understandable, robust (POUR) principles

Performance Testing:

Detection Speed:

- Measure frames per second on target devices
- Test inference latency (time from capture to detection result)
- Validate interval-based capture effectiveness
- Monitor CPU/GPU utilization during operation

Application Responsiveness:

- Measure time from detection to voice instruction
- Test UI responsiveness during CV processing
- Validate frame capture queue management
- Monitor memory usage patterns

Battery Consumption:

- Measure battery drain per hour of active use
- Test power consumption during continuous operation
- Validate adaptive interval adjustment effectiveness
- Monitor thermal behavior

Compatibility Testing:

Android Devices:

- Test on representative devices (Samsung, Google Pixel, etc.)
- Validate Android 7.0 through latest version
- Test native TTS engine integration
- Verify NNAPI and GPU delegate functionality

iOS Devices:

- Test on iPhone 8 and newer models
- Validate iOS 13 through latest version
- Test VoiceOver integration
- Verify Core ML optimization

Usability Testing with Target Users:

Participant Testing Protocol:

- Task-based testing with coffee-making scenarios
- Think-aloud protocol during task execution
- Post-task questionnaires for subjective feedback
- Semi-structured interviews for qualitative insights
- Video recording (with consent) for interaction analysis

Evaluation Metrics:

- Task completion success rate
- Time-on-task measurements
- Error frequency and recovery success
- User satisfaction scores
- Preference and feedback analysis

Security and Privacy Testing:

Privacy Verification:

- Audit data collection practices
- Verify local-only processing (no external transmission)
- Test camera frame handling and disposal
- Validate local storage security
- Verify no PII collection

Regression Testing:

- Re-test previously validated functionality after bug fixes
- Verify no new issues introduced by changes
- Test backward compatibility with previous settings
- Validate data migration if storage schema changes

Smoke Testing:

- Verify application launches successfully
- Test camera initialization
- Validate basic voice output
- Confirm navigation to all primary screens
- Test settings persistence

5.2 Test Case Creation and Management

Test Case Creation:

1. Requirements Gathering:
    - Extract testable requirements from Low-Level Design document
    - Identify functional specifications from High-Level Design
    - Document accessibility requirements from WCAG 2.1 AA
2. Test Scenario Identification:
    - Decompose each module into testable features
    - Identify normal, boundary, and error conditions
    - Create test scenarios for each BrewingStep state
    - Define CV detection test scenarios (varied conditions)
3. Detailed Test Case Development:
    - Define preconditions and test environment setup
    - Document step-by-step test procedures
    - Specify expected results with pass/fail criteria
    - Include accessibility validation steps
    - Document required test data and resources
4. Prioritization:
    - Assign priority levels (Critical, High, Medium, Low)
    - Prioritize based on risk analysis results
    - Schedule high-priority cases for early execution

Test Suite Organization:

- **Module-Based Suites**: Separate suites for UI, Input, CV, and Voice Guidance modules
- **Feature-Based Suites**: Organized by functional areas (detection, guidance, settings)
- **Testing Type Suites**: Functional, accessibility, performance, compatibility
- **User Scenario Suites**: End-to-end workflows for user testing

Test Case Management:

- **Tool**: Utilize test management platform (e.g., TestRail, Zephyr, or GitHub Issues with labels)
- **Tracking**: Monitor execution status, results, and defects
- **Version Control**: Maintain test case versions aligned with application releases
- **Documentation**: Keep comprehensive records for regression and future reference
- **Updates**: Regularly review and update test cases as features evolve

5.3 Utilization of Testing Tools

Automated Testing Tools:

- **Jest**: Unit testing framework for JavaScript/React Native components
- **React Native Testing Library**: Component testing for UI elements
- **Detox**: End-to-end testing framework for mobile apps

Accessibility Testing Tools:

- **Accessibility Scanner (Android)**: Automated accessibility issue detection
- Xcode Accessibility Inspector (iOS): iOS accessibility validation
- **Color Contrast Analyzer**: WCAG contrast ratio verification
- **Screen Reader Testing**: Direct TalkBack and VoiceOver validation

Performance Testing Tools:

- **Android Profiler**: CPU, memory, and battery monitoring
- **Xcode Instruments**: iOS performance profiling
- **TensorFlow Lite Benchmark Tool**: Model inference performance measurement

Manual Testing Tools:

- **Camera Testing Setup**: Controlled lighting environments, tripods, target coffee machines
- **User Testing Recording**: Screen recording software, observation protocols
- **Feedback Collection**: Questionnaires, interview guides, satisfaction scales

6. Defect Management

6.1 Bug Reporting

A standardized bug reporting process ensures comprehensive documentation and efficient resolution:

Bug Identification:

Testers will identify and document any encountered bugs, defects, or deviations from expected behavior during all testing sessions, whether automated or manual.

Bug Reporting Form:

Each bug report must include:

- **Bug ID**: Unique identifier (auto-generated by tracking system)
- **Title**: Concise, descriptive summary
- **Module/Component**: Affected system module (UI, Input, CV, Voice Guidance)
- Severity Level:
    - **Critical**: Application crash, data loss, safety risk, accessibility blocker
    - **High**: Major feature malfunction, significant accessibility issue
    - **Medium**: Feature partially broken, usability problem
    - **Low**: Cosmetic issue, minor inconvenience
- **Priority**: Urgency of fix (P0-Critical, P1-High, P2-Medium, P3-Low)
- **Steps to Reproduce**: Detailed, numbered steps
- **Expected Result**: What should happen
- **Actual Result**: What actually happens

- **Environment**: Device model, OS version, application version
- **Screenshots/Videos**: Visual evidence when applicable
- **Additional Context**: Lighting conditions, user settings, any relevant details

Bug Repository:

- **Tool**: Centralized bug tracking system (e.g., Jira, GitHub Issues, Bugzilla)
- **Organization**: Categorized by module, severity, priority, and status
- **Access**: Shared access for development team, testers, and stakeholders
- **Integration**: Linked with version control for traceability

6.2 Bug Resolution Process

A streamlined bug resolution process ensures timely fixes and prevents regression:

1. Bug Triage:

- Daily or weekly triage meetings to review new bug reports
- Validation of bug reproducibility
- Assignment of accurate severity and priority levels
- Identification of duplicate or related bugs

2. Bug Assignment:

- Critical and high-priority bugs assigned immediately to appropriate developers
- Clear ownership established for each bug
- Estimated resolution timeline communicated

3. Bug Fixing:

- Developers implement fixes following coding standards
- Code review process for all bug fixes
- Unit tests added to prevent regression
- Documentation of fix approach and changes

4. Verification Testing:

- Testers verify bug fixes in development/staging environment
- Validation that fix resolves the issue without side effects
- Testing of related functionality for regression

5. Regression Testing:

- Comprehensive regression testing after significant bug fixes
- Automated test suite execution where applicable
- Manual testing of high-risk areas

6. Bug Closure:

- Bugs marked as "Resolved" after successful verification
- Status updated to "Closed" after regression confirmation
- Communication to stakeholders and team members

7. Communication and Documentation:

- Regular status updates on critical bugs
- Release notes documenting fixed issues
- Knowledge base articles for common issues and workarounds
- Post-mortem analysis for critical bugs to prevent recurrence

7. Conclusion

This Test Plan Report outlines a comprehensive, accessibility-first approach to testing the "Coffee Machine Assistance for the Visually Impaired" mobile application. The testing strategy encompasses rigorous validation of computer vision accuracy, voice guidance effectiveness, accessibility compliance, cross-platform compatibility, and real-world usability with target users.

By adhering to this structured testing plan, we aim to deliver a reliable, safe, and empowering assistive technology solution that enables visually impaired individuals to independently operate coffee machines with confidence. The testing methodology prioritizes user safety, accessibility, and privacy while ensuring technical robustness and performance.

The success of this application depends not only on technical excellence but on meaningful validation with the visually impaired community. Through iterative testing, continuous feedback integration, and unwavering commitment to accessibility standards, we will create an application that genuinely enhances independence and quality of life for our target users.

8. Appendix

A. References

1. M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and Efficient Object Detection," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020.
2. World Wide Web Consortium, "Web Content Accessibility Guidelines (WCAG) 2.1," W3C Recommendation, 2018.
3. "Mobile App Testing: A Complete Guide" by Daniel Knott
4. "Accessibility for Everyone" by Laura Kalbag
5. Android Developers, "TextToSpeech | Android Developers," [Online]. Available: https://developer.android.com/reference/android/speech/tts/TextToSpeech
6. Apple Inc., "VoiceOver | Apple Developer Documentation," [Online]. Available: https://developer.apple.com/documentation/accessibility/voiceover
7. "Software Testing Fundamentals: A Comprehensive Guide for Beginners" by A.G. Stefanescu and I. Budimac
8. React Native Documentation, "Testing Overview," [Online]. Available: https://reactnative.dev/docs/testing-overview
9. TensorFlow, "TensorFlow Lite Guide," [Online]. Available: https://www.tensorflow.org/lite/guide

B. Glossary of Terms

- **Accessibility**: The practice of making applications usable by people with disabilities, including visual, auditory, motor, and cognitive impairments.
- **Assistive Technology (AT)**: Devices, software, or equipment that help people with disabilities perform functions that might otherwise be difficult or impossible.
- **Bounding Box**: A rectangular coordinate region that identifies the location of a detected object within an image.
- **Computer Vision (CV)**: AI techniques enabling machines to interpret and understand visual input from the world.
- **Confidence Threshold**: A minimum confidence score required for a detection to be considered valid.
- **EfficientDet**: A scalable, high-efficiency object detection model family that balances accuracy and computational efficiency.
- **Intersection over Union (IoU)**: A metric used to evaluate the overlap between predicted and ground truth bounding boxes in object detection.
- **Mean Average Precision (mAP)**: A metric used to evaluate the accuracy of object detectors, calculated by taking the mean average precision across all classes and/or IoU thresholds.
- **POUR Principles**: Perceivable, Operable, Understandable, Robust - the four core principles of WCAG accessibility guidelines.
- **React Native**: A cross-platform framework for building mobile applications using JavaScript and React, enabling shared codebase for Android and iOS.
- **Regression Testing**: Verifies that changes made to the software have not introduced any new bugs or broken existing functionality.
- **Screen Reader**: Assistive technology that reads aloud screen content for visually impaired users (e.g., TalkBack, VoiceOver).
- **State Machine**: A computational model controlling application logic through well-defined states and transitions.
- **TensorFlow Lite (TFLite)**: A lightweight machine learning framework for mobile inference, enabling on-device model execution.
- **Text-to-Speech (TTS)**: Technology that converts text into synthesized speech output.
- **User-Centered Design (UCD)**: A design philosophy that prioritizes understanding and involving intended users throughout the development process.
- **WCAG (Web Content Accessibility Guidelines)**: International standards for web and mobile accessibility, defining requirements for making content accessible to people with disabilities.