QA Analysis of Road Sign Recognition and Testing Framework Integration

Report for Quality Assurance

Executive Summary

This pull request introduces a comprehensive testing suite, including unit, instrumented, and UI tests, while significantly refactoring the core road sign recognition application. Key functional changes include improved user feedback mechanisms (Text-to-Speech, vibration for prohibition signs), speed limit warnings, and a silent mode. From a Quality Assurance perspective, this requires a multi-faceted testing approach to validate the new features, verify the extensive refactoring, ensure the stability of the recognition pipeline, and confirm the reliability of the new automated tests.

Testing Strategy Overview

The testing strategy must be comprehensive due to the large-scale changes. It will combine automated and manual testing. The primary goals are to: 1) Leverage and validate the newly introduced Espresso UI tests and instrumented tests. 2) Perform extensive manual testing on the combinations of user-configurable options from the main screen. 3) Execute real-world scenario testing, either simulated or through actual driving, to assess performance, accuracy, and usability. 4) Conduct rigorous regression testing on the entire sign recognition and display pipeline, which has been heavily refactored.

- Leverage new automated tests for baseline functionality.
- Focus manual testing on feature interactions and edge cases.
- Prioritize real-world performance and user experience (e.g., battery life, detection accuracy).
- Implement a thorough regression suite to catch issues from refactoring.

Functional Testing Requirements

Functional testing will focus on validating that all features work as described in the README and as implemented in the code.

- Verify all combinations of the main screen switches: Text/Image Info, Speech Info, Vibration, Speed Measurement, and Silent Mode.
- Test the sign recognition model with a variety of signs under different conditions (lighting, angle, partial occlusion, distance).
- Validate the Text-to-Speech (TTS) feature, including the handling of missing language packs and the accuracy of spoken sign names.
- Confirm the speed measurement feature's accuracy against a reliable GPS source and ensure the UI updates correctly.
- Test the speed limit warning functionality, specifically for the 'teren zabudowany' sign when speed exceeds 50 km/h.
- Verify the Silent Mode correctly mutes and unmutes notification, alarm, and ring streams, and handles pre-existing silent states.
- Check all application dialogs (e.g., errors for model loading, missing GPS, missing TTS pack) for correctness and clarity.
- Ensure the app gracefully handles permission denials for Camera and Location.

Non-Functional Testing Considerations

Beyond core functionality, the application's performance and resource consumption are critical for a good user experience, especially for a real-time processing app.

- Performance Testing: Monitor CPU usage, memory consumption, and battery drain during prolonged use (e.g., a 30-minute session). Assess the impact of recognition on camera FPS.
- Usability Testing: Evaluate the clarity of audio and visual alerts. Ensure they are helpful and not overly distracting to a driver. Check UI responsiveness on various devices.
- Security Testing: Verify that all requested permissions (Camera, Location, Notification Policy) are necessary and that the app provides clear justifications. Ensure the app is not vulnerable to issues related to mock location usage in production.
- Compatibility Testing: Test on a range of Android devices with varying screen sizes, OS versions (API 28+), and hardware capabilities (e.g., with/without a powerful GPU).

Integration Testing Requirements

Testing the application's interaction with external services and the underlying operating system is crucial.

- Firebase Integration: Verify the app can successfully download the TensorFlow Lite model and labels file from Firebase Storage. Test the app's behavior with slow or no internet connection. Confirm it correctly fetches sign image URLs and types from the Realtime Database.
- Operating System Integration: Test how Silent Mode interacts with incoming phone calls and notifications from other apps. Validate app behavior during lifecycle events (backgrounding, foregrounding, rotation if applicable).
- Hardware Integration: Test the vibration feature on devices with different haptic feedback motors. Verify camera and GPS hardware are initialized and released correctly.

Test Environment Setup Needs

A specific setup is required to effectively test all aspects of this application.

- Physical Android devices (minimum 2) with cameras and GPS, running different Android OS versions (e.g., Android 9, Android 11).
- Access to a test Firebase project to manage the ML model, labels file, and sign database without affecting production.
- Tools for mock location services to reliably simulate different speeds and test the speed limit features without actual driving.
- A controlled environment with printed road signs to test recognition accuracy under repeatable conditions.
- A device with the Polish TTS language pack uninstalled to test the corresponding error-handling flow.

Risk-Based Testing Prioritization

Given the scope of changes, testing efforts should be prioritized based on risk and user impact.

- High Priority: Core sign recognition pipeline (refactored `objectDetector`), Silent Mode functionality (OS interaction risk), and speed limit violation alerts (safety feature).
- Medium Priority: Combinations of all user-selectable options, TTS functionality and error handling, Firebase model downloading, and app performance/battery usage.

- Low Priority: Static UI elements, layout changes on the main screen, and README content accuracy.

■ Recommended Test Scenarios

- Scenario: All Features Enabled. Enable all switches. Verify that when a 'Zakaz wjazdu' (No Entry) sign is shown, the app displays its name/image, speaks the name, triggers a vibration, and continues to show vehicle speed.
- Scenario: Graceful Degradation (No Internet). Launch the app for the first time with network connectivity disabled. Verify the app displays an informative error regarding the inability to download the model/labels and does not crash.
- Scenario: Permission Denial (Location). Enable the 'Speed Measurement' switch but deny the location permission when prompted. Verify the app proceeds to the detector screen but shows an error that speed cannot be measured.
- Scenario: OS Integration (Silent Mode). With the app's Silent Mode enabled, receive an incoming phone call. Verify the device does not ring or vibrate for the call.
- Scenario: Edge Case (Speed Limit). Using a mock location provider, simulate driving while a 'Teren zabudowany' (Built-up Area) sign is displayed. Increase speed from 49 km/h to 51 km/h. Confirm the over-speed limit warning is triggered.
- Scenario: TTS Language Pack Missing. On a device without the Polish TTS pack, enable the 'Speech Info' switch. Verify the app presents a dialog prompting the user to install the language pack with a shortcut to settings.
- Scenario: Recognition in Adverse Conditions. Test sign recognition in a low-light environment to assess model performance and ensure the camera feed remains usable.
- Scenario: App Lifecycle. While the detector is active, send the app to the background and then bring it back to the foreground. Confirm that the camera view resumes and detection continues without any crashes.
- Scenario: Input Validation. On the main screen, uncheck the 'Text/Image', 'Speech', and 'Vibration' options. Attempt to start the detector. Verify a toast message appears preventing launch and the detector activity does not start.
- Scenario: Performance Under Load. Run the detector continuously for 20 minutes. Monitor for excessive battery drain (e.g., >15% drop), device overheating, and any degradation in UI responsiveness.

- Scenario: Vibration Logic. With the vibration switch enabled, show the camera a non-prohibition sign (e.g., a warning sign like 'Uwaga Dzieci'). Verify that no vibration occurs.
- Scenario: UI State Persistence. Toggle the camera background view in the DetectorActivity, then navigate back to the main screen and re-enter the detector. Verify the camera view preference is not persisted and resets to the default.
- Scenario: Two-Sign Display. Show the camera sign A, then quickly show sign B. Verify that sign B appears as the primary detection and sign A moves to the secondary (previous detection) display slot.

★ Recommendations

- Execute all newly created automated tests (`ActivityUITest`, `DetectorTest`, `UnitTest`) as part of the CI/CD pipeline.
- Perform a full manual regression test cycle focusing on the high-priority risk areas before merging.
- Conduct at least one real-world driving test to evaluate the end-to-end user experience.
- Add logging around the `adjustAudio` function to better debug silent mode issues on different devices.
