QA Analysis for Road Sign Detection and Testing Framework PR

Report for Quality Assurance

Executive Summary

This pull request introduces a significant enhancement to the application by establishing a formal testing framework with unit, instrumented, and UI tests using Espresso and Mockito. Concurrently, it involves a major refactoring of the core `DetectorActivity` and `objectDetector` components, including a switch to a GPU delegate for model inference and changes to image normalization. From a QA perspective, this necessitates a comprehensive testing strategy focusing on the newly implemented features, the integrity of refactored logic, and potential performance and accuracy regressions.

Testing Strategy Overview

The testing strategy should be a hybrid approach, combining the newly added automated tests with rigorous manual and exploratory testing. The primary focus will be on the high-risk areas identified in the refactored code. The strategy involves:

- Leveraging and expanding the new Espresso UI tests to cover all user flows and option combinations.
- Executing targeted functional tests on the core features: sign detection, user feedback (visual, audio, haptic), speed monitoring, and silent mode.
- Conducting non-functional testing to validate performance, stability, and resource consumption, especially concerning the new GPU delegate.
- Performing extensive regression testing on the entire detection pipeline to ensure the refactoring and normalization changes have not negatively impacted accuracy.
- Validating all integration points, particularly with Firebase services (Model Downloader, Database) and Android system services (Location, Audio, TTS).

Functional Testing Requirements

Functional testing must validate that all features described in the README and implemented in the code work as expected across various conditions.

- **Options Configuration:** Verify that all toggles on the `MainActivity` (Text/Image, Speech, Vibration, Speed, Silent Mode) correctly enable or disable their respective features in `DetectorActivity`.
- **Input Validation:** Test the new logic that requires at least one of the primary feedback mechanisms (Text/Image, Speech, or Vibration) to be selected before starting the detector.
- **Sign Detection:** Confirm that the app correctly identifies a variety of road signs in real-time. Test both the current detection display and the 'previous detection' history feature.
- **Feedback Mechanisms:** Individually and in combination, test the visual (text/image), audio (Text-to-Speech), and haptic (vibration for prohibition signs) feedback systems.
- **Speed Monitoring:** Validate the accuracy of the speed measurement feature against a reliable GPS source. Test the speed limit warning functionality (e.g., >50 km/h in a 'teren zabudowany' zone).
- **Silent Mode:** Test the silent mode's ability to mute notifications and calls while the detector is active.
- **Error Handling:** Verify that the application gracefully handles errors such as missing TTS language packs, failed model downloads, denied permissions, and poor GPS signal, presenting appropriate alerts to the user.

Non-functional Testing Considerations

Beyond core functionality, the app's performance and usability are critical for a positive user experience, especially given its real-time processing nature.

- **Performance Testing:** Profile the application during active detection to measure CPU/GPU usage, memory consumption, and battery drain. Establish a baseline to ensure the GPU delegate provides a net performance benefit without causing excessive heat or battery loss. Measure detection latency/FPS.
- **Security Testing:** Verify that all requested permissions (Camera, Location, Notification Policy, Vibrate) are necessary and used appropriately. Confirm that the app handles permission denial gracefully at any point in the user flow. Assess the integrity of the model download process from Firebase.

- **Usability Testing (UAT):** Evaluate the app's performance under diverse environmental conditions (e.g., low light, night, rain, sun glare). Assess the clarity of the Text-to-Speech announcements and the noticeability of the vibration feedback. Test with partially obscured, faded, or non-standard signs to understand the model's limitations.

Test Environment Setup Needs

A diverse test environment is required to ensure broad compatibility and robustness.

- **Hardware:** A range of real Android devices with varying specifications (CPU, GPU support, RAM, camera quality) and running different OS versions (Android 9-11, API 28-30).
- **Software:** Android Studio Profiler for performance monitoring. A mock location provider app for simulating movement and testing speed-related features.
- **Connectivity:** Ability to test with and without Wi-Fi and cellular data to validate offline behavior and error handling for Firebase connections.
- **Physical Setup:** A controlled environment with high-quality images, videos, or physical mock-ups of various Polish road signs to test detection accuracy systematically.

Risk-based Testing Prioritization

Testing efforts should be prioritized based on the risk and impact of the changes.

- **High Priority:** `objectDetector.java` - The change in image normalization logic (dividing by 255.0f) is a critical risk to model accuracy and requires immediate, thorough regression testing with a curated dataset.
- **High Priority:** `DetectorActivity.java` - As the central hub for all features, its extensive refactoring makes it susceptible to integration bugs. End-to-end testing of all feature combinations is crucial.
- **Medium Priority:** Performance on devices with and without strong GPU support. The GPU delegate could introduce instability or performance degradation on some chipsets.
- **Medium Priority:** Error handling and permissions flows. These are critical for a stable user experience and must be tested for all edge cases.
- **Low Priority:** Minor UI changes in layout XML files and resource modifications.

■ Recommended Test Scenarios

- Verify that attempting to start the detector with 'Text/Image', 'Speech', and 'Vibration' switches all turned off displays the 'Wybierz przynajmniej jedną z pierwszych trzech opcji' toast and prevents navigation.
- Launch the app with the 'Speed' option enabled but location permission denied. Verify the app displays the 'Nie udzielono dostępu do lokalizacji' alert and does not crash.
- Enable 'Silent Mode', start the detector, and simulate an incoming call. Verify the device's ringer is silenced as per the `adjustAudio` logic.
- On a device without the Polish TTS language pack, enable the 'Speech' option and start the detector. Verify the app correctly displays the alert prompting the user to install the language pack.
- Enable the 'Vibration' option and present a 'Zakaz wjazdu' (No Entry) sign. Verify the sign is correctly identified and the device vibrates.
- Using a mock location provider, simulate a speed of 65 km/h and present a 'Teren zabudowany' (Built-up Area) sign. Verify the UI visually indicates a speed limit violation.
- Clear app data and launch for the first time with no internet connection. Verify a user-friendly error message is displayed regarding the failure to download the model or labels.
- With the detector running, use Android Profiler to monitor CPU and GPU usage for 5 minutes. Record the average FPS and check for memory leaks or excessive battery drain.
- While simulating movement, present sign A, then quickly switch to sign B. Verify sign A moves to the 'previous sign' display when sign B becomes the current detection.
- Test sign detection in a low-light environment to determine the operational limits of the camera and model.
- In `DetectorActivity`, toggle the camera mode button. Verify the background correctly switches between the gradient and the transparent camera feed without performance degradation.
- Regression Test: Run detection on a pre-defined set of 20 road sign images and verify that the detection accuracy is within an acceptable threshold, confirming the new normalization logic works as intended.

★ Recommendations

- Establish a 'golden dataset' of standard road sign images to create an automated regression test for detection accuracy. This will be vital for validating future model updates or changes to the processing pipeline.
- Expand unit tests in `UnitTest.java` to cover more logic within `DetectorActivity`, such as the conditions for speed limit warnings and parsing sign data.
- Implement performance benchmarks to run with each build, automatically flagging significant regressions in FPS, memory usage, or battery consumption.
