### Test Automation Strategy for Smart Doorbell Camera

Contents

[**1. Objectives:** 1](#_Toc190355738)

[**2. Scope of Automation:** 1](#_Toc190355739)

[**3. Tools, Frameworks, and Technologies:** 1](#_Toc190355740)

[**4. Test Case Design:** 2](#_Toc190355741)

[**5. Implementation:** 2](#_Toc190355742)

**1. Objectives:**

* Quickly catch regressions in core functionalities and key user journeys.
* Expand test coverage for **facial recognition, motion detection**, and web app functionalities.
* Support Continuous Integration and Continuous Delivery (CI/CD).
* Reduce manual effort for repetitive test cases.
* Enhance product quality and speed up release cycles.

**2. Scope of Automation:**

**Inclusions:**

* **Smoke Tests:** Ensure fundamental features work (doorbell button, video recording, and motion detection).
* **Core Functional Tests:** Covering crucial user interactions (user login, video playback, settings updates).
* **Facial Recognition Tests:**
  + Validate accuracy in different conditions (lighting, angles, known/unknown faces, partial visibility, and multiple individuals in a frame).
  + Performance benchmarking to measure processing efficiency.
* **Web Application Tests:** Verify user workflows (adding/editing known individuals, viewing history, managing notifications).
* **Video Processing Tests:**
  + Confirm video recording quality.
  + Check video compression and storage efficiency.
  + Validate automated video tagging and retrieval.
* **API Tests:** Directly test APIs for user management, device settings, and data retrieval.
* **Integration Tests:** Ensure smooth interaction between the doorbell, local hub, and web application.

**3. Tools, Frameworks, and Technologies:**

**Test Automation Frameworks:**

* **Cypress:** Automates UI and API testing.
* **Playwright:** Ensures compatibility across browsers.

**BDD Framework:**

* **Cucumber with Gherkin syntax** for writing clear, human-readable test cases.

**Programming Language:**

* **JavaScript/TypeScript** for web automation.
* **Python** for video and image processing.

**Assertion & Reporting:**

* **Chai** (integrated with Cypress for validations).
* **Mochawesome** for generating detailed HTML reports.

**Facial Recognition & Video Processing Libraries:**

* **OpenCV** (Python/JavaScript) for video analysis and image processing.
* **dlib** for facial recognition and feature extraction.
* **Face-recognition (Python)** for real-time face detection and identity matching.
* **FFmpeg** for automated video analysis, compression, and format conversion.
* **TensorFlow/Keras** (for deep learning-based facial recognition models if needed).

**4. Test Case Design:**

**BDD Approach:**

**Scenario Examples:**

* **Facial Recognition:**
  + Verify known individual detection under different lighting conditions.
  + Ensure unknown individuals are not falsely identified.
  + Test partial face visibility scenarios.
  + Handle multiple faces in a single frame.
* **Video Processing:**
  + Confirm video quality (length, resolution, format).
  + Verify automatic tagging and retrieval accuracy.
  + Evaluate compression techniques to optimize storage.

**5. Implementation:**

**Phase 1:** Set up test environments, install necessary tools, and develop smoke tests for core functionalities.

**Phase 2:** Automate core functionality tests and establish API testing framework.

**Phase 3:** Implement facial recognition tests with various conditions and edge cases.

**Phase 4:** Develop web application test automation to cover key user workflows.

**Phase 5:** Integrate automation into the CI/CD pipeline, expand API testing, and enhance test coverage.

**Phase 6:** Ongoing maintenance, refinement, and optimization of the test automation suite.