Test Planning Document

Drone Delivery System - Authentication & Safety Requirements

This Test Planning Document supports Learning Outcome 2 of the Software Testing portfolio. It describes how testing activities are planned for a selected subset of system requirements, including prioritisation, verification and validation strategy, required scaffolding and instrumentation, and integration of testing into a defined software lifecycle.

R-A: Restricted Airspace Safety

Requirement: R.3 – The system shall prevent drones from entering restricted or no-fly zones.

Category: Safety, functional, system-level

Rationale: This is a safety requirement in the sense defined by Y&P: it exists solely to prevent a hazard (illegal or dangerous flight), rather than to provide functionality. Failure could result in regulatory violations or physical harm. Such requirements demand high assurance.

Priority and Pre-requisites

Priority: HIGH - This is a safety-critical requirement with regulatory implications. Failure to prevent no-fly zone violations could result in:

- Legal liability and regulatory penalties
- Physical danger to people or property
- Loss of operational license
- Reputational damage

Pre-requisites for Testing:

- Geographic coordinate system implementation (R.5)
- Flight movement rules engine (R.4)
- Region definition parser and validator (R.3.1-R.3.3)
- Pathfinding algorithm implementation

Testing Approach - Multiple Verification Methods:

Following the principles of Chapter 3, this high-priority safety requirement demands at least two different T&A approaches:

1. **Property-Based Testing (Primary)**
    a. **Rationale:** Safety properties like "never enter restricted zones" are universal invariants that must hold across infinite input spaces. Property-based testing generates hundreds of random test cases to verify this invariant.
    b. **Implementation:** jqwik framework generating random positions, paths, and no-fly zones
    c. **Coverage:** Tests boundary conditions, overlapping zones, edge cases automatically
2. **Contract-Based Testing (Secondary)**
    a. **Rationale:** Contracts define pre-conditions and post-conditions for pathfinding methods, ensuring that any valid input produces a safe output
    b. **Implementation:** Pre-condition contracts verify inputs are valid; post-condition contracts verify no path segment intersects restricted regions
    c. **Advantage:** Catches violations at the point of occurrence, not after full path execution
3. **Integration Testing (Tertiary)**
    a. **Rationale:** Verifies that the geometry service correctly interacts with the pathfinding algorithm
    b. **Tests:** Known routes around Edinburgh with real no-fly zone data
    c. **Purpose:** Validates the system behaves correctly with realistic geographic data

Decomposition Strategy (Partition Principle):

Following Chapter 3's partition principle, we decompose R.3 into:

- **Geometric calculation component:** Determines if a point is inside a region (R.3.1-R.3.3)
- **Path validation component:** Checks each segment of a proposed path

- **Route planning component:** Generates paths that avoid restricted zones

Each component can be tested independently before integration testing.

R-B: Secure Delivery Authentication

Requirement: R.7 / R.8 – The system shall verify delivery requesters using OTP verification and verify delivery recipients using facial recognition.

Category: Security, system-level

Rationale: This requirement enforces a secure chain of trust in a medical context. Incorrect behaviour does not merely degrade functionality but enables fraud or misdelivery of medication. Security requirements are explicitly high-risk and require layered testing approaches.

Priority and Pre-requisites

Priority: HIGH - Security requirement protecting medical deliveries. Failures could result in:

- Medication theft or diversion
- Privacy violations (GDPR/HIPAA concerns)
- Patient harm from incorrect medication delivery
- Loss of trust in the delivery system

Pre-requisites for Testing:

- OTP generation service with cryptographic random number generator (R.7.1)
- Email delivery mechanism for OTP distribution (R.7.4)
- OTP storage with expiry tracking (R.7.2, R.7.3)
- Facial recognition service integration
- Reference image storage system
- Similarity threshold configuration

Testing Approach - Layered Security Verification:

1. **Fault Injection Testing (Primary for OTP)**

    a. **Rationale:** Security systems must gracefully handle failure scenarios. Fault injection simulates real-world failures to verify resilience.
    b. **Scenarios tested:**
        i. Expired OTPs (R.7.2)
        ii. Email delivery failures (R.7.4)
        iii. Brute force attempts (Q.2)
        iv. OTP reuse attempts (R.7.3)
        v. Concurrent verification attempts
    c. **Implementation:** Mockito-based fault injection with time manipulation and exception throwing
2. **Unit Testing (Coverage for Individual Components)**
    a. **OTP generation:** Verify 6-digit format, cryptographic randomness, uniqueness
    b. **OTP verification:** Correct validation logic, case sensitivity, format checking
    c. **Expiry mechanism:** Time-based invalidation works correctly
    d. **Email service:** Proper message formatting and delivery
3. **Integration Testing (Service Composition)**
    a. **End-to-end flow:** Register → Generate OTP → Send Email → Verify OTP → Face Match
    b. **State management:** Verify OTP state transitions (generated → sent → verified → invalidated)
    c. **Cross-service communication:** OTP service ↔ Email service ↔ Face recognition service
4. **API Contract Testing**
    a. **Rationale:** Security endpoints must reject malformed requests and enforce proper authentication flows
    b. **Coverage:** HTTP status codes (200, 400, 404), input validation, content-type enforcement
    c. **Implementation:** MockMvc-based tests verifying API contracts

Security-Specific Testing Considerations:

- **Q.2 Compliance:** Probability of guessing valid OTP < 1 in 1,000,000
    - 6-digit OTP = 1,000,000 combinations
    - Single-use enforcement prevents multiple attempts
    - 5-minute expiry limits attack window
    - Testing verifies: random distribution, no predictable patterns, proper invalidation

- **Q.1 Replay Attack Prevention:**
    - Tests verify OTP cannot be reused after successful verification
    - Tests verify expired OTPs are rejected
    - Tests verify OTPs are removed from storage after use

R-C: End-to-End Delivery Workflow

Requirement: S.1 – The system shall support end-to-end medical delivery workflows from request to completion.

Category: System-level integration, functional

Rationale: This is an overarching system requirement that validates the integration of all sub-requirements (authentication, routing, verification). It represents the actual user journey and must work reliably in production.

Priority and Pre-requisites

Priority: HIGH - This is the ultimate validation that the system delivers value to end users. All other requirements exist to support this workflow.

Pre-requisites for Testing:

- Delivery registration endpoint (R.7, R.8)
- OTP generation and verification (R.7)
- Facial recognition verification (R.8)
- Drone routing and availability (R.1, R.2)
- All supporting services operational

Testing Approach - System-Level Validation:

1. **End-to-End System Testing**
    a. **Rationale:** Only a complete system test can verify that all components integrate correctly to deliver the core workflow
    b. **Implementation:** REST-assured based integration tests with real HTTP calls
    c. **Coverage:** Complete user journey from delivery request through authentication to collection
2. **Test Scenario:**

1. User registers delivery with photo and email
2. System generates unique delivery ID and sends OTP via email
3. User verifies OTP
4. User presents face for verification at collection
5. System validates face match and releases medication

3. **Validation Points:**
    a. Registration returns valid delivery ID (R.7)
    b. OTP is sent and can be verified (R.7.1-R.7.4)
    c. Facial recognition correctly matches reference image (R.8)
    d. HTTP responses follow API contracts
    e. State transitions are correct (registered → OTP verified → face verified → delivered)

Why System-Level Testing is Essential Here:

- **Integration Validation:** Tests that services communicate correctly (not just that they work in isolation)
- **Workflow Validation:** Verifies the sequence of operations matches the intended user experience
- **Data Flow Validation:** Confirms delivery ID propagates correctly through all stages
- **Error Propagation:** Ensures failures at any stage are handled gracefully

Scaffolding and Instrumentation

For R-A (No-Fly Zones)

Required Scaffolding:

- **Mock pathfinding service:** Returns controlled test paths for property verification
- **Test data generators:** jqwik arbitraries for positions, regions, and no-fly zones
- **Geometry test utilities:** Helper methods for point-in-region calculations

Instrumentation:

- **Path validation logging:** Records each path segment checked against no-fly zones
- **Violation detection:** Flags and reports any detected zone entry
- **Coverage tracking:** Ensures all code paths in geometry service are exercised

Scheduled Tasks:

1. Implement geometry service (Week 3)
2. Build property-based test generators (Week 4)
3. Develop contract tests for pathfinder (Week 4)
4. Integration testing with real Edinburgh data (Week 5)

For R-B (Authentication)

Required Scaffolding:

- **Mock email sender:** JavaMailSender mock to prevent actual email sending in tests
- **Time manipulation:** ReflectionTestUtils to control OTP expiry for testing
- **Mock facial recognition:** Controlled similarity scores for deterministic testing
- **Test image fixtures:** Reference images for facial recognition tests

Instrumentation:

- **OTP generation logging:** Track OTP creation, attempts, and invalidation
- **Email capture:** ArgumentCaptor to verify email content and recipients
- **Timing attack detection:** Measure verification timing to detect potential vulnerabilities
- **State inspection:** Reflection-based access to internal OTP storage for verification

Scheduled Tasks:

1. Implement OTP service with cryptographic RNG (Week 3)
2. Build fault injection test suite (Week 4)
3. Implement facial recognition integration (Week 5)
4. API contract tests (Week 5)
5. Security audit and timing analysis (Week 6)

For R-C (End-to-End Workflow)

Required Scaffolding:

- **REST-assured test framework:** HTTP client for API testing
- **Test resource management:** Real image files for upload testing
- **Test database/storage:** Isolated test environment for delivery state
- **Test data cleanup:** Ensures tests don't interfere with each other

Instrumentation:

- **Request/response logging:** Full HTTP transaction capture for debugging
- **State verification:** Check delivery state transitions at each step
- **Performance metrics:** Measure end-to-end workflow completion time
- **Error scenario testing:** Verify failure handling at each workflow stage

Scheduled Tasks:

1. Implement delivery registration endpoint (Week 4)
2. Integrate OTP and face recognition services (Week 5)
3. Build E2E test suite (Week 6)
4. Performance and reliability testing (Week 7)

Process and Risk Integration

Chosen Lifecycle: Iterative development with Continuous Integration (CI/CD)

Testing Integration into Lifecycle

Early Stage (Weeks 1-3):

- Requirements analysis and test planning (this document)
- Unit test development alongside implementation (TDD where appropriate)
- Mock-based testing for components not yet implemented

Middle Stage (Weeks 4-6):

- Integration testing as services are completed
- Property-based testing for safety requirements
- Fault injection testing for security requirements
- API contract validation

Late Stage (Weeks 7-9):

- End-to-end system testing
- Performance testing with realistic loads
- Security audit and penetration testing

- Coverage analysis and gap identification

Continuous Activities:

- Automated unit tests on every commit (git hooks)
- Full test suite on every push to main (GitHub Actions)
- JaCoCo coverage reporting
- Test result documentation

Risk Assessment and Mitigation

R-A Risks:

- **Risk:** Property tests may not cover all geometric edge cases
    - **Likelihood:** Medium
    - **Impact:** High (safety violation)
    - **Mitigation:** Supplement with deterministic boundary tests for known problematic geometries
- **Risk:** Real-world geographic data may differ from test data
    - **Likelihood:** Medium
    - **Impact:** Medium
    - **Mitigation:** Integration tests with actual Edinburgh no-fly zone data; validation testing in staging environment

R-B Risks:

- **Risk:** Email delivery failures in production
    - **Likelihood:** Low-Medium
    - **Impact:** High (user cannot verify delivery)
    - **Mitigation:** Fault injection tests validate graceful degradation; consider fallback mechanisms (SMS, app notification)
- **Risk:** Timing attacks could leak OTP information
    - **Likelihood:** Low
    - **Impact:** Medium
    - **Mitigation:** Constant-time comparison implementation; timing analysis in test suite
- **Risk:** Facial recognition false positives/negatives
    - **Likelihood:** Medium
    - **Impact:** High (security breach or legitimate user lockout)

    - **Mitigation:** Threshold tuning; extensive testing with diverse faces; human review fallback

R-C Risks:

- **Risk:** Integration failures between services
    - **Likelihood:** Medium
    - **Impact:** High (complete workflow failure)
    - **Mitigation:** Comprehensive E2E tests; contract testing; staged rollout
- **Risk:** Test environment differs from production
    - **Likelihood:** Medium
    - **Impact:** Medium
    - **Mitigation:** Docker-based test environment matching production; staging environment for pre-release testing