Capstone Project 1: Insurance Policy Management

Automation (Updated)

Introduction

Insurance companies rely heavily on web applications for customer onboarding, policy management, claim processing, and premium payments. Manual testing of these features is time-consuming and error-prone. This project automates the functional, regression, API, and data-driven testing of an Insurance Policy Management System (IPMS) using Selenium WebDriver, TestNG, and RestAssured.

Problem Statement

Insurance applications deal with multiple modules like policy issuance, renewal, premium calculation, claims management, and payments. Errors in these workflows may cause financial losses, compliance issues, or security risks. A robust automation suite is needed to ensure accuracy, improve coverage, and reduce manual efforts.

Objectives

- Automate **end-to-end test scenarios** of insurance policy management.
- Implement **data-driven testing** for multiple policy types and premium calculations.
- Integrate **API testing** for backend validation in addition to UI.
- Automate **role-based access** tests (Admin, Agent, Customer).
- Enable **parallel and cross-browser execution** with TestNG.
- Enhance reporting with **dashboards and critical failure notifications**.
- Validate **negative and edge case scenarios** for all modules.

Scope (Updated)

1. **User Authentication**
    - Role-based access automation (Admin, Agent, Customer).
    - Session timeout and secure logout.
    - Concurrent session handling.
2. **Policy Creation**
    - Automate creation across Life, Health, Vehicle insurance policies.
    - Validate field-level rules: mandatory checks, formats, patterns, duplicates.
    - Negative cases: invalid coverage amounts, duplicate policy numbers, incorrect data.
3. **Premium Calculator**
    - Validate premiums based on inputs (age, sum insured, coverage, add-ons).
    - Edge cases: invalid ranges (age < 0, coverage > maximum limit).

- Regression validation with data-driven inputs.

4. **Policy Renewal**
   - Renewal of expired policies.
   - Pro-rata calculations for partial renewals.
   - Negative case: renewal of inactive/cancelled policies.
5. **Claims Processing**
   - Submit and track claims across statuses (initiated, under review, approved, rejected).
   - Validate email/SMS notifications on claim status changes.
   - Document uploads (size/format validation, multiple attachments).
   - Negative cases: invalid claim IDs, duplicate submissions.
6. **Payment Gateway**
   - Positive flows: successful card, net banking, and UPI payments.
   - Negative flows: expired card, insufficient funds, network timeout, retries.
   - Refund and partial payment validation.
7. **API Integration**
   - Validate backend APIs (policy data, premium calculation, claim status) with RestAssured.
   - Compare backend responses with UI values.
   - Security tests for unauthorized access and invalid tokens.
8. **System Validations**
   - Session timeout after inactivity.
   - Secure logout flow.
   - Enforce role-based restrictions (e.g., customer restricted from admin actions).

Tech Stack

- **Language:** Java
- **Automation Tool:** Selenium WebDriver
- **API Testing:** RestAssured
- **Testing Framework:** TestNG
- **Reporting:** Extent Reports, TestNG HTML Reports
- **Build Tool:** Maven/Gradle
- **Data Source:** Excel/CSV (Apache POI) for data-driven testing
- **CI/CD:** Jenkins (optional)

Functionalities to Automate (Updated)

1. Login/Logout tests – valid, invalid, and role-based authentication.
2. Policy creation – multiple insurance types, field validations, and edge cases.
3. Premium calculation – valid inputs, invalid ranges, regression checks.
4. Policy renewal – expired, active, and cancelled policies.

5. Claims processing – status tracking, notifications, attachments.
6. Payment gateway – success, failure, retries, and refunds.
7. API validation – cross-verification of UI and backend data.
8. System-level validations – session timeout, secure logout, and restricted access.

Test Scenarios (Extended)

1. Verify login with valid/invalid credentials and role-specific access.
2. Validate new policy creation with correct premiums and mandatory field checks.
3. Verify renewal process for expired and partially active policies.
4. Submit claims and validate acknowledgment, status updates, and notifications.
5. Test negative cases like duplicate policy/claim submissions.
6. Validate multiple payment failures (expired card, insufficient balance).
7. Execute parallel testing of multiple policy scenarios across browsers.
8. Validate API responses for premium calculation and claim status.
9. Ensure session timeout after inactivity and secure logout.

Expected Outcomes (Updated)

- **Comprehensive regression suite** with positive, negative, and edge cases.
- **Reusable and modular test scripts** covering multiple insurance workflows.
- **Backend + UI validation** ensuring data consistency.
- **Reduced manual effort** with higher test coverage.
- **Automated reports and dashboards** for management with critical failure alerts.
- **Faster execution** through parallel and cross-browser testing.

Insurance Domain – Open Source / Free Apps

Since full-fledged open-source insurance apps are limited, simulation can be achieved via:

- **GitHub Insurance Projects:** Spring Boot + React-based systems.
- **Demo Blaze / Parabank:** Simulate insurance-like workflows.
- **Custom Insurance Portal (Mock):** Build with HTML/React and mock APIs (ReqRes, Mockoon).

Recommended: Use a GitHub Insurance Management repo or simulate insurance flows on Demo Blaze/Parabank for project execution.

.