DEVX360

Sipho Sehlapelo
Kesley Hamann
Sibusiso Mngomezulu
Lwando Msindo
David Musa-Aisien

Contents

DevX360 Testing Policy

Introduction

The purpose of this Testing Policy is to define the approach, tools, and processes used to verify and validate the DevX360 system. Testing ensures that the system meets both functional and non-functional requirements, achieves the expected level of quality, and is reliable for end-user deployment.

This policy applies to:

- Unit testing of all individual components,
- Integration testing of system interactions,
- System and end-to-end testing of workflows,
- Non-functional testing of performance, scalability, and reliability.

Testing is automated wherever possible and is enforced through our CI/CD pipeline.

Scope

The testing policy applies to all components that are directly developed and maintained by the DevX360 team. This ensures that every part of the system under our control is thoroughly verified for correctness, reliability, and performance before deployment.

In Scope:
Testing covers the following aspects of the system:

- Frontend React components – All user interface elements, dashboards, and interactive components will be tested to confirm that they render correctly, respond to user input, and maintain consistent behavior across devices.
- Backend API services – The Node.js/Express backend will be tested to ensure endpoints respond correctly, enforce business logic, and integrate seamlessly with external services.
- AI analytics module – The AI-driven insights and recommendations will be tested for correctness of outputs and robustness when handling different input conditions.
- MongoDB database integration – Data storage, retrieval, and updates will be verified through integration and system tests to ensure consistency and accuracy.

- External integrations (GitHub API via Octokit) – While the GitHub API itself is external, the way our system interacts with it (requests, data handling, error management) will be fully tested.
- Deployed web application – End-to-end testing will be conducted on the production-like deployed environment to validate that the application works for real end users.

Out of Scope:
Certain factors fall outside the scope of our testing, as they are not under our team's control:

- Availability and reliability of third-party services – For example, outages or downtime of the GitHub API cannot be tested or guaranteed by us, although we will test our system's resilience in handling such failures gracefully.
- External infrastructure beyond our control – This includes the hosting provider's uptime guarantees, network availability, and hardware-level reliability, which are assumed to be managed by the respective service providers.

Testing Strategy

We adopt a multi-level testing strategy: The main advantage of a multi-level testing strategy is that it allows for the early and cost-effective detection of defects by testing software components in isolation before combining and validating them as an integrated system.

Unit Testing

- Purpose: Verify correctness of isolated functions and modules.
- Frameworks: Jest (backend/frontend), React Testing Library (UI).
- Coverage Target: ≥80%.

Integration Testing

- Purpose: Validate interactions between backend, database, and external APIs.
- Tools: Jest + Supertest.
- Scope: DORA metric calculations, team management workflows, AI analysis requests.

System / End-to-End Testing

- Purpose: Test the complete system in a deployed environment.
- Tool: Cypress.
- Scope: User stories (login, team creation, dashboard metrics and AI insights).

The non-functional testing scope includes:

- **Performance**
    - Dashboard load (QR 1.1): Dashboard must load in ≤2 seconds for 95% of requests.
    - DORA metrics calculation (QR 1.2): New commits should be processed and reflected in metrics within ≤5 seconds per ingestion.
    - API endpoints (QR 1.3): 95% of API requests should complete within 500–2000 ms.
- **Load / Scalability**
    - User capacity (QR 2.1): The system must support 50 concurrent users performing common actions without performance degradation or error rates exceeding 1%.
- **Reliability**
    - Real-time updates (QR 3.2): Updates to commits and issues must appear on the dashboard within ≤300 seconds of becoming available.
    - Uptime (QR 3.1): The deployed system must maintain ≥99.5% annual uptime.
- **Automated Tests (Maintainability & Code Quality)**
    - Code quality (QR 7.2): The project must maintain ≥80% unit test coverage across components.

Test Automation Policy

Automated testing is tightly integrated into the DevX360 development workflow through continuous integration and continuous deployment (CI/CD) using GitHub Actions.

Pipeline Execution

The following automated workflows are enforced:

- Pull Requests & Merges
    - All tests run automatically on every pull request and commit to the feature/frontend branch as well as the feature/Ai-analysis branch.
    - All tests also run on every merge into main to ensure production-readiness.
- Branch-Specific Workflows
    - Frontend Workflow (feature/frontend):
      Specializes in testing React components, UI rendering, and client-side integrations. This workflow runs automatically on every commit to the feature/frontend branch.
    - Backend AI Workflow (feature/AI-analysis):
      Specializes in testing the AI analytics module, backend API services, and data pipelines. This workflow runs automatically on every commit to the feature/AI-analysis branch.

Failure Conditions

A pipeline build is marked as failed if any of the following conditions occur:

- Unit or integration tests do not pass.
- Overall code coverage drops below 80%.
- Any workflow step in the specialized frontend or backend pipelines fails.

Outcome

This policy ensures that all commits are validated against automated quality gates before being merged into shared branches. By enforcing test automation early in the development lifecycle, DevX360 maintains high reliability, prevents regression, and upholds consistent quality across both the frontend and backend components.

Testing Tools

Category	Tool	Justification
Unit & Integration	Jest, React Testing Library, Supertest	Standard for Node.js/React, integrates with CI/CD
End-to-End	Cypress	Simulates user workflows in browser
Performance/Load	JMeter, Locust	Stress testing & concurrent user simulation
CI/CD	GitHub Actions	Native GitHub integration, automated enforcement
Monitoring	UptimeRobot, New Relic	Verifies uptime and deployment reliability

Test Case Management

All test cases for DevX360 are stored in the repository under the _tests_ directory of either the feature/AI-analysis or feature/frontend branch. There is also a feature/testing branch that stores all tests for archiving purposes. Each test case is traceable back to a user story or system requirement, ensuring coverage across both functional and non-functional goals.

Each test case definition follows a consistent structure:

- Preconditions – required setup or state.
- Steps – sequence of actions executed.
- Expected Result – intended outcome as defined by requirements.
- Actual Result – observed outcome from execution.
- Pass/Fail Status – determined automatically by the testing framework.

Automated Test Execution

- Tests are run automatically as part of the CI/CD pipeline on GitHub Actions.
- Execution logs clearly indicate which test suites and individual test cases passed.
- The pipeline considers a run successful only if all test suites pass.

Coverage & Quality Gates

- Coverage reports are automatically generated using Jest with Istanbul (nyc).
- The following quality gates determine success:
  - Test suites: All must pass (e.g., "11 passed, 11 total").
  - Tests: 100% of executed tests must pass (e.g., "53 passed, 53 total").
  - Coverage: Overall project coverage must remain ≥80%.
- Coverage breakdowns are reported per file, including statements, branches, functions, and lines.

Reporting & Visibility

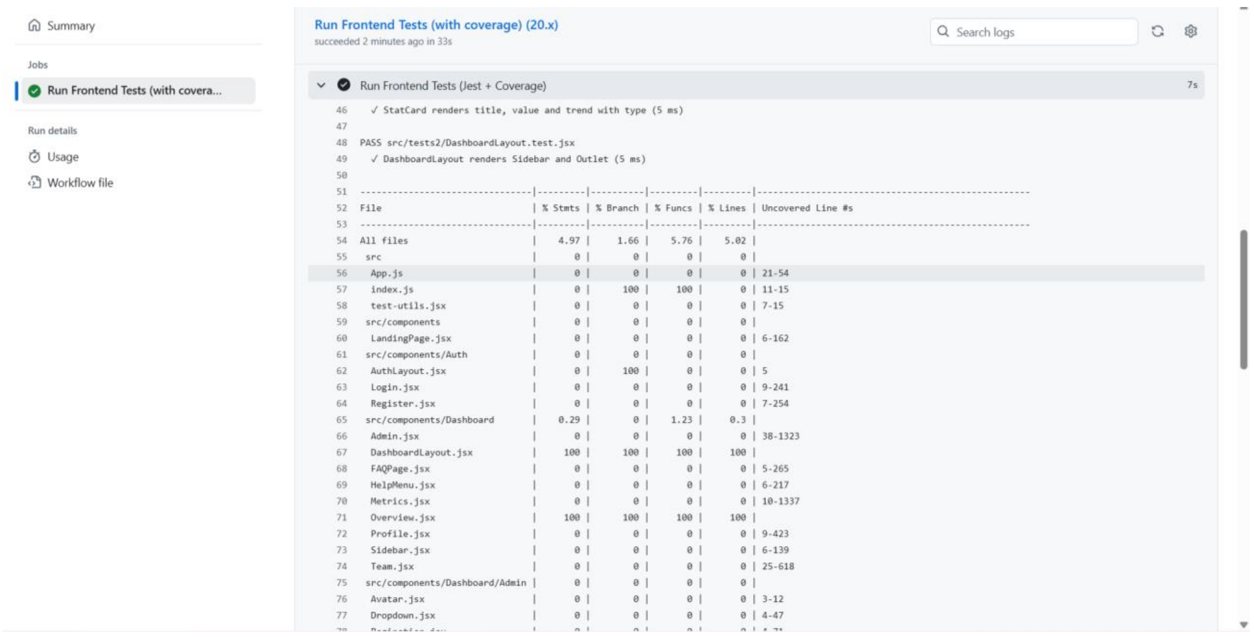
- Reports are surfaced directly in the CI logs on GitHub Actions.
- Coverage details are generated at the end of every run and stored as build artifacts.
- Test results serve as traceable evidence of requirement validation, to be included in the Test Evidence and Results section with screenshots of successful CI runs.

CI/CD Configuration

The DevX360 frontend is fully integrated into an automated CI/CD pipeline. All test suites in the src/tests2 folder are executed using Jest on every push or pull request to the feature branch. The workflow ensures that core components are validated automatically, and coverage thresholds can be configured to meet project standards or expanded as the codebase grows.

Understanding Coverage Results:

Column	**Meaning**
% Stmts	Percentage of executable statements executed during testing.
% Branch	Percentage of conditional branches (e.g., if/else) that were tested.
% Funcs	Percentage of functions invoked by the tests.
% Lines	Percentage of lines executed by the tests.
Uncovered Line #s	Specific lines that were not reached by the tests.



Run Frontend Tests (with coverage) (20.x)

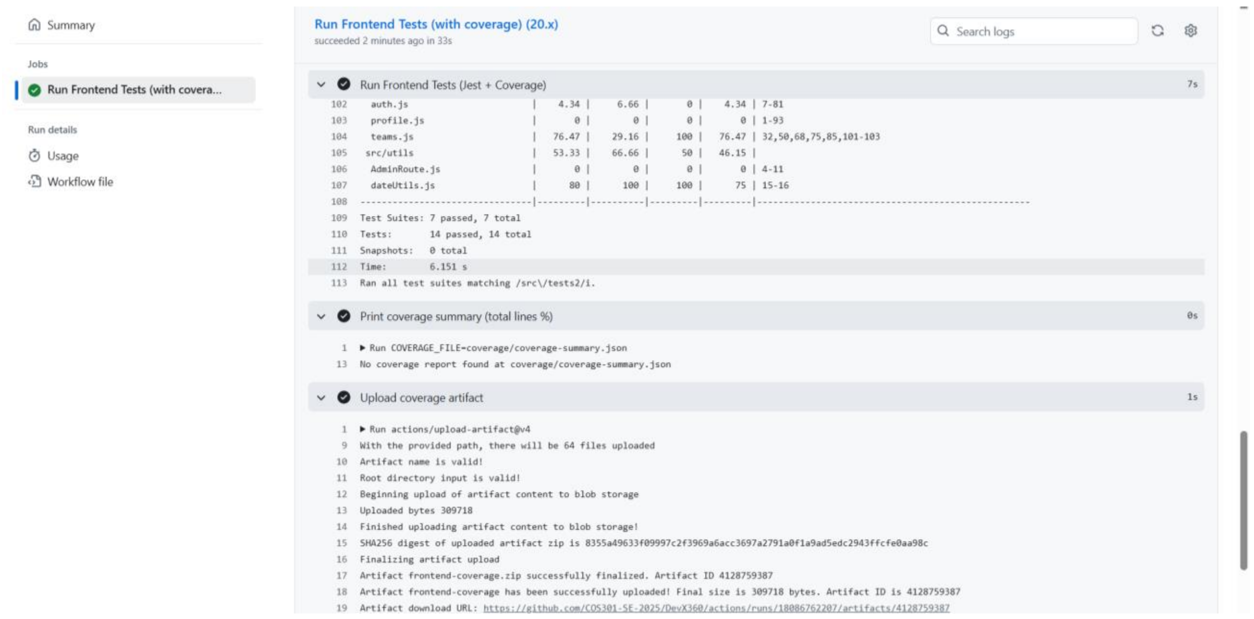
succeeded 2 minutes ago in 33s

Run Frontend Tests (Jest + Coverage) 7s

```
  ✓ StatCard renders title, value and trend with type (5 ms)

PASS src/tests2/DashboardLayout.test.jsx
  ✓ DashboardLayout renders Sidebar and Outlet (5 ms)

--------------------------------|---------|----------|---------|---------|-------------------------------------------------
File                            | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
--------------------------------|---------|----------|---------|---------|-------------------------------------------------
All files                       |    4.97 |     1.66 |    5.76 |    5.02 |
 src                            |       0 |        0 |       0 |       0 |
  App.js                        |       0 |        0 |       0 |       0 | 21-54
  index.js                      |       0 |      100 |     100 |       0 | 11-15
  test-utils.jsx                |       0 |        0 |       0 |       0 | 7-15
 src/components                 |       0 |        0 |       0 |       0 |
  LandingPage.jsx               |       0 |        0 |       0 |       0 | 6-162
 src/components/Auth            |       0 |        0 |       0 |       0 |
  AuthLayout.jsx                |       0 |      100 |       0 |       0 | 5
  Login.jsx                     |       0 |        0 |       0 |       0 | 9-241
  Register.jsx                  |       0 |        0 |       0 |       0 | 7-254
 src/components/Dashboard       |    0.29 |        0 |    1.23 |     0.3 |
  Admin.jsx                     |       0 |        0 |       0 |       0 | 38-1323
  DashboardLayout.jsx           |     100 |      100 |     100 |     100 |
  FAQPage.jsx                   |       0 |        0 |       0 |       0 | 5-265
  HelpMenu.jsx                  |       0 |        0 |       0 |       0 | 6-217
  Metrics.jsx                   |       0 |        0 |       0 |       0 | 10-1337
  Overview.jsx                  |     100 |      100 |     100 |     100 |
  Profile.jsx                   |       0 |        0 |       0 |       0 | 9-423
  Sidebar.jsx                   |       0 |        0 |       0 |       0 | 6-139
  Team.jsx                      |       0 |        0 |       0 |       0 | 25-618
 src/components/Dashboard/Admin |       0 |        0 |       0 |       0 |
  Avatar.jsx                    |       0 |        0 |       0 |       0 | 3-12
  Dropdown.jsx                  |       0 |        0 |       0 |       0 | 4-47
```



Run Frontend Tests (with coverage) (20.x)

succeeded 2 minutes ago in 33s

Run Frontend Tests (Jest + Coverage) 7s

```
  auth.js                       |    4.34 |     6.66 |       0 |    4.34 | 7-81
  profile.js                    |       0 |        0 |       0 |       0 | 1-93
  teams.js                      |   76.47 |    29.16 |     100 |   76.47 | 32,50,68,75,85,101-103
 src/utils                      |   53.33 |    66.66 |      50 |   46.15 |
  AdminRoute.js                 |       0 |        0 |       0 |       0 | 4-11
  dateUtils.js                  |      80 |      100 |     100 |      75 | 15-16
--------------------------------|---------|----------|---------|---------|-------------------------------------------------
Test Suites: 7 passed, 7 total
Tests:       14 passed, 14 total
Snapshots:   0 total
Time:        6.151 s
Ran all test suites matching /src\/tests2/i.
```

Print coverage summary (total lines %) 0s

```
▶ Run COVERAGE_FILE=coverage/coverage-summary.json
No coverage report found at coverage/coverage-summary.json
```

Upload coverage artifact 1s

```
▶ Run actions/upload-artifact@v4
With the provided path, there will be 64 files uploaded
Artifact name is valid!
Root directory input is valid!
Beginning upload of artifact content to blob storage
Uploaded bytes 309718
Finished uploading artifact content to blob storage!
SHA256 digest of uploaded artifact zip is 8355a49633f09997c2f3969a6acc3697a2791a0f1a9ad5edc2943ffcfe0aa98c
Finalizing artifact upload
Artifact frontend-coverage.zip successfully finalized. Artifact ID 4128759387
Artifact frontend-coverage has been successfully uploaded! Final size is 309718 bytes. Artifact ID is 4128759387
Artifact download URL: https://github.com/COS301-SE-2025/DevX360/actions/runs/18086762207/artifacts/4128759387
```