DEVX360

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

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	Grafana K6	Stress testing & concurrent user simulation
Usability	Google Lighthouse	Complete test suite with comprehensive testing tools.
Security	OWASP ZAP	Automated and easy to use with actionable feedback provided.
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.

Quality Requirement Testing

Performance Testing:

To validate that DevX360 meets its performance requirements, we tested the system under expected and extreme conditions:

- Expected Load Testing: Using the Grafana K6 test suite, we simulated up to 50 concurrent users, performing typical actions such as dashboard navigation, AI-insight requests, and metric calculations. This ensured the system remained responsive and met the defined thresholds for API response times and dashboard loading.
- Spike and Stress Testing: We then increased the load suddenly to 200 concurrent users to evaluate system behavior under high-stress conditions. This allowed us to verify that the system could gracefully handle unexpected spikes in traffic, and to identify any bottlenecks or failure points for future optimization.

- Frontend Performance Analysis: The user interface was also assessed using the Google Lighthouse extension. Key metrics included page load times, time to interactive, and rendering efficiency. This testing highlighted areas for optimization, such as oversized assets or inefficient rendering paths, to ensure smooth interactions for end-users.

Frontend Results:

83

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 50–89 90–100

METRICS Expand view

First Contentful Paint
1.2 s

Largest Contentful Paint
2.0 s

Total Blocking Time
60 ms

Cumulative Layout Shift
0.01

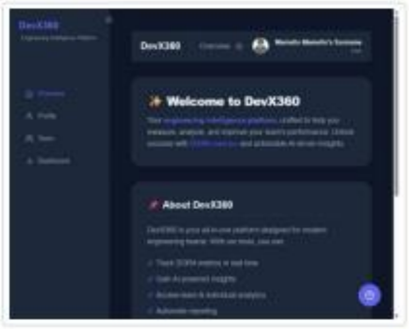
Speed Index
2.3 s

96

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 50–89 90–100



METRICS Expand view

First Contentful Paint
0.8 s

Largest Contentful Paint
1.0 s

Total Blocking Time
20 ms

Cumulative Layout Shift
0

Speed Index
1.6 s

81

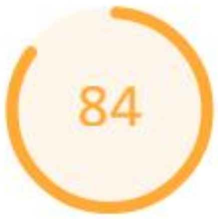
Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 50–89 90–100

METRICS Expand view

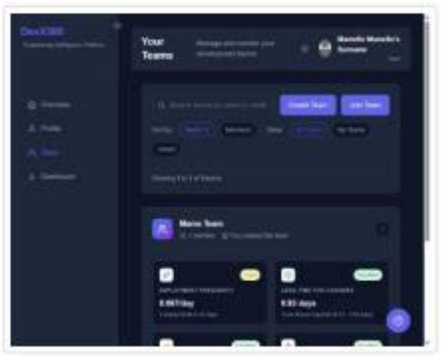
First Contentful Paint 1.2 s	Largest Contentful Paint 2.2 s
Total Blocking Time 50 ms	Cumulative Layout Shift 0
Speed Index 2.2 s	



Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 50–89 90–100



METRICS Expand view

First Contentful Paint 1.2 s	Largest Contentful Paint 2.0 s
Total Blocking Time 30 ms	Cumulative Layout Shift 0.003
Speed Index 2.2 s	

Key Insights:

DevX360 demonstrates generally good performance with room for optimization. The platform shows consistent strengths in layout stability but requires attention to JavaScript efficiency and resource loading across all pages.

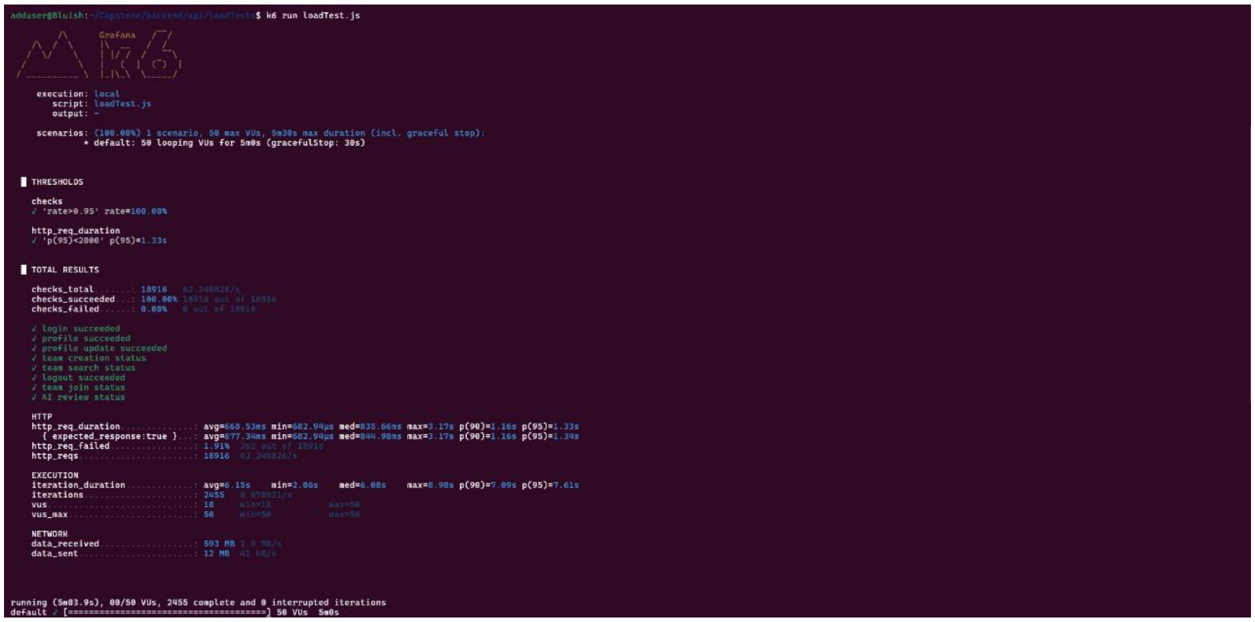
Strengths

- Rock-Solid Layout: The pages are exceptionally stable. Users won't experience annoying elements suddenly shifting around as the page loads, which is crucial for a data-heavy dashboard.
- Responsive Interactivity: The platform feels quick to respond to user clicks and inputs. The time between when the page is drawn and when it becomes fully interactive is well within the good threshold.
- Consistent Experience: Performance is reliably good across all different sections of the application (Overview, Team, Profile, and Metrics). There are no glaringly slow pages that disrupt the user's workflow.

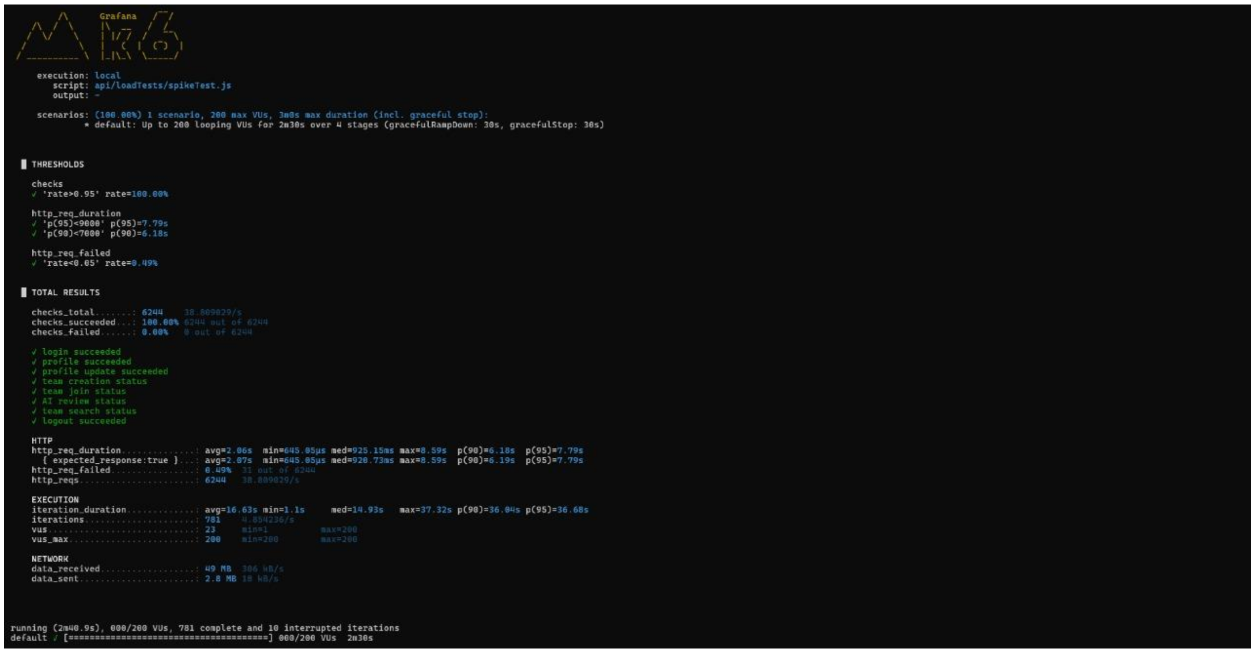
Key Opportunities for Improvement

- **JavaScript Efficiency:** This is our biggest opportunity. We are shipping a significant amount of unused code to the user's browser (~700 KB). By cleaning this up, we can make the initial download much faster.
- **Faster Initial Load:** The time it takes to fetch the main document and key resources is slower than ideal. Optimizing this and improving how we cache files could shave **600+ milliseconds** off the load time.
- **Speed of Content Display:** While the Overview page loads its main content quickly (1.0s), other pages like Profile and Metrics are slower (2.0s+). Optimizing images and the loading sequence for these pages will make them feel snappier.

Backend Results:



```
adduser@Bluish:~/Capstone/backend/api/loadTests$ k6 run loadTest.js

                  Grafana

     execution: local
        script: loadTest.js
        output: -

     scenarios: (100.00%) 1 scenario, 50 max VUs, 5m30s max duration (incl. graceful stop):
              * default: 50 looping VUs for 5m0s (gracefulStop: 30s)


  █ THRESHOLDS

    checks
    ✓ 'rate>0.95' rate=100.00%

    http_req_duration
    ✓ 'p(95)<2000' p(95)=1.33s


  █ TOTAL RESULTS

    checks_total.......: 18916   62.248826/s
    checks_succeeded...: 100.00% 18916 out of 18916
    checks_failed......: 0.00%   0 out of 18916

    ✓ login succeeded
    ✓ profile succeeded
    ✓ profile update succeeded
    ✓ team creation status
    ✓ team search status
    ✓ logout succeeded
    ✓ team join status
    ✓ AI review status

    HTTP
    http_req_duration..............: avg=668.53ms min=682.94µs med=835.66ms max=3.17s p(90)=1.16s p(95)=1.33s
      { expected_response:true }...: avg=677.34ms min=682.94µs med=844.98ms max=3.17s p(90)=1.16s p(95)=1.34s
    http_req_failed................: 1.91%  362 out of 18916
    http_reqs......................: 18916  62.248826/s

    EXECUTION
    iteration_duration.............: avg=6.15s    min=2.06s    med=6.08s    max=8.98s p(90)=7.09s p(95)=7.61s
    iterations.....................: 2455   8.078921/s
    vus............................: 18     min=18         max=50
    vus_max........................: 50     min=50         max=50

    NETWORK
    data_received..................: 593 MB 1.9 MB/s
    data_sent......................: 12 MB  41 kB/s



running (5m03.9s), 00/50 VUs, 2455 complete and 0 interrupted iterations
default ✓ [======================================] 50 VUs  5m0s
```



```
                  Grafana

     execution: local
        script: api/loadTests/spikeTest.js
        output: -

     scenarios: (100.00%) 1 scenario, 200 max VUs, 3m0s max duration (incl. graceful stop):
              * default: Up to 200 looping VUs for 2m30s over 4 stages (gracefulRampDown: 30s, gracefulStop: 30s)


  █ THRESHOLDS

    checks
    ✓ 'rate>0.95' rate=100.00%

    http_req_duration
    ✓ 'p(95)<9000' p(95)=7.79s
    ✓ 'p(90)<7000' p(90)=6.18s

    http_req_failed
    ✓ 'rate<0.05' rate=0.49%


  █ TOTAL RESULTS

    checks_total.......: 6244    38.809029/s
    checks_succeeded...: 100.00% 6244 out of 6244
    checks_failed......: 0.00%   0 out of 6244

    ✓ login succeeded
    ✓ profile succeeded
    ✓ profile update succeeded
    ✓ team creation status
    ✓ team join status
    ✓ AI review status
    ✓ team search status
    ✓ logout succeeded

    HTTP
    http_req_duration..............: avg=2.06s  min=645.05µs med=925.15ms max=8.59s  p(90)=6.18s  p(95)=7.79s
      { expected_response:true }...: avg=2.07s  min=645.05µs med=920.73ms max=8.59s  p(90)=6.19s  p(95)=7.79s
    http_req_failed................: 0.49%  31 out of 6244
    http_reqs......................: 6244   38.809029/s

    EXECUTION
    iteration_duration.............: avg=16.63s min=1.1s     med=14.93s   max=37.32s p(90)=36.04s p(95)=36.68s
    iterations.....................: 781    4.854236/s
    vus............................: 23     min=1          max=200
    vus_max........................: 200    min=200        max=200

    NETWORK
    data_received..................: 49 MB  306 kB/s
    data_sent......................: 2.8 MB 18 kB/s



running (2m40.9s), 000/200 VUs, 781 complete and 10 interrupted iterations
default ✓ [======================================] 000/200 VUs  2m30s
```

Key insights:

- Stable Performance under Expected Load: All API requests were successfully serviced within the target 2-second response time during normal load conditions.
- Graceful Degradation under Stress: During spike testing, while response times increased beyond the 2-second threshold, the system did not crash and successfully serviced all API requests.
- System Resilience: The most critical outcome was maintained—zero failed requests—even when under heavy load, ensuring full service availability.

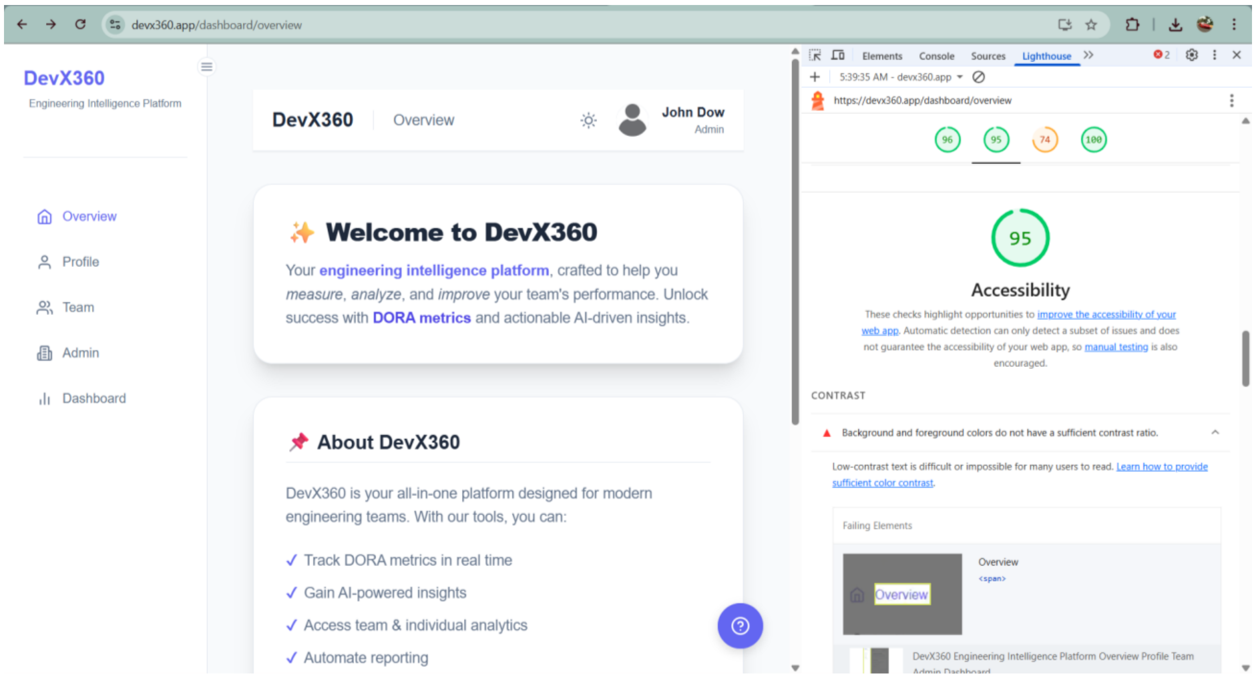
Usability Testing:

To ensure a high-quality user experience for everyone, we validated the platform using industry-standard tools.

Testing Methodology:

- Automated Accessibility Testing: We used the axe DevTools and Google Lighthouse Accessibility suite to perform a comprehensive audit against the WCAG (Web Content Accessibility Guidelines) standards.
- Scope: These tools were used to systematically check for common accessibility issues that could affect users with disabilities.

Test results:



✨ **Welcome to DevX360**

Your **engineering intelligence platform**, crafted to help you *measure*, *analyze*, and *improve* your team's performance. Unlock success with **DORA metrics** and actionable AI-driven insights.

📌 **About DevX360**

DevX360 is your all-in-one platform designed for modern engineering teams. With our tools, you can:

✓ Track DORA metrics in real time
✓ Gain AI-powered insights
✓ Access team & individual analytics
✓ Automate reporting

95

Accessibility

These checks highlight opportunities to improve the accessibility of your web app. Automatic detection can only detect a subset of issues and does not guarantee the accessibility of your web app, so manual testing is also encouraged.

CONTRAST

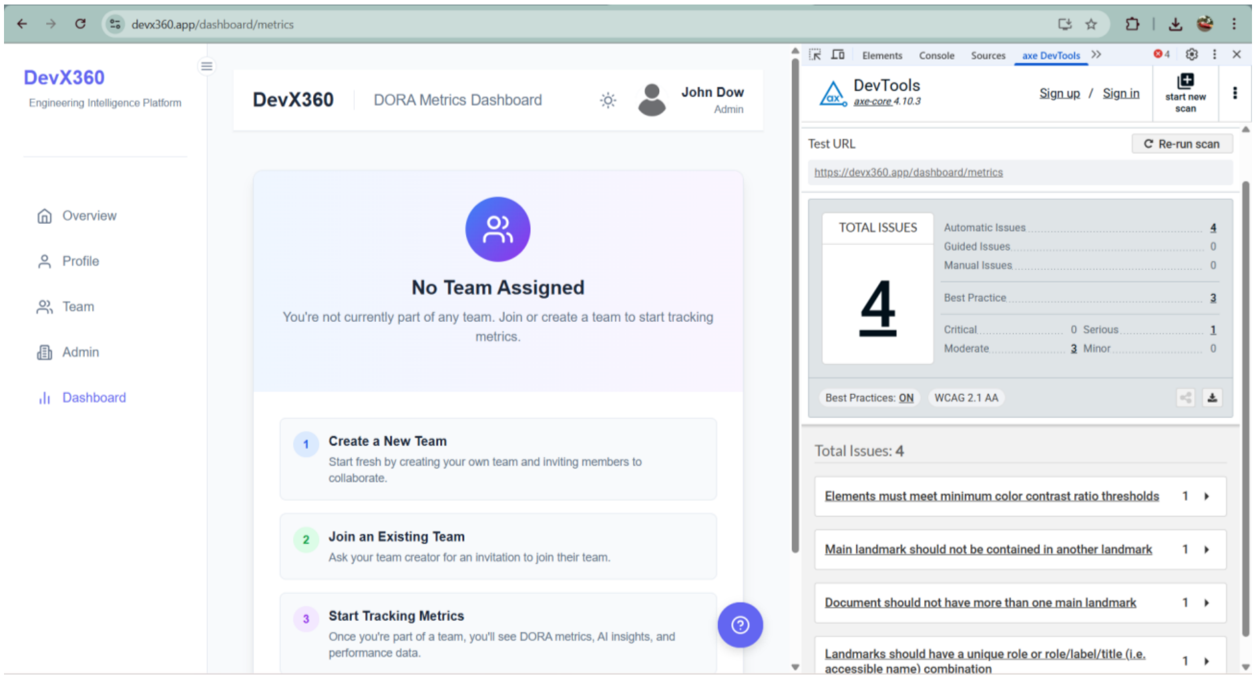
▲ Background and foreground colors do not have a sufficient contrast ratio.

Low-contrast text is difficult or impossible for many users to read. Learn how to provide sufficient color contrast.

Failing Elements

Overview
<span>

DevX360 Engineering Intelligence Platform Overview Profile Team Admin Dashboard



devx360.app/dashboard/metrics

DevX360
Engineering Intelligence Platform

- Overview
- Profile
- Team
- Admin
- Dashboard

DevX360 | DORA Metrics Dashboard | John Dow, Admin

No Team Assigned

You're not currently part of any team. Join or create a team to start tracking metrics.

1 **Create a New Team**
Start fresh by creating your own team and inviting members to collaborate.

2 **Join an Existing Team**
Ask your team creator for an invitation to join their team.

3 **Start Tracking Metrics**
Once you're part of a team, you'll see DORA metrics, AI insights, and performance data.

DevTools axe-core 4.10.3

Test URL
https://devx360.app/dashboard/metrics

TOTAL ISSUES 4

Automatic Issues	4
Guided Issues	0
Manual Issues	0
Best Practice	3
Critical	0
Serious	1
Moderate	3
Minor	0

Best Practices: ON WCAG 2.1 AA

Total Issues: 4

Elements must meet minimum color contrast ratio thresholds	1
Main landmark should not be contained in another landmark	1
Document should not have more than one main landmark	1
Landmarks should have a unique role or role/label/title (i.e. accessible name) combination	1

Key Insights:

No critical or serious accessibility barriers found. Platform is largely accessible.

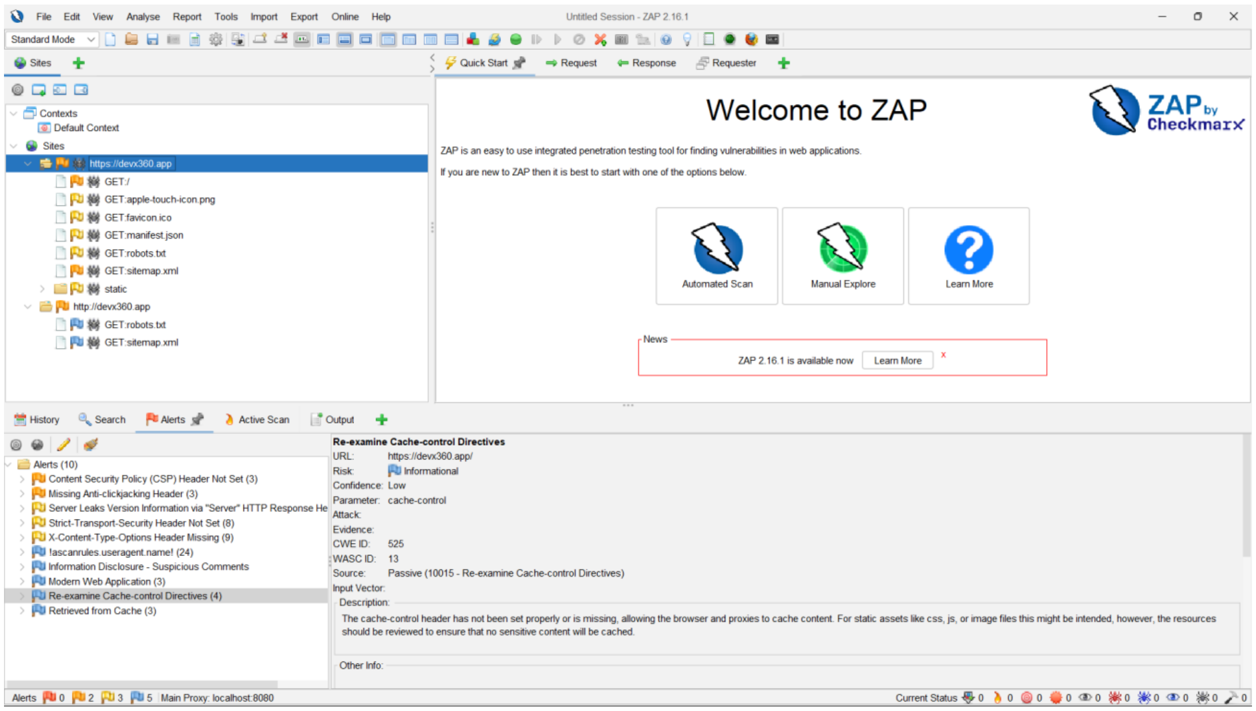
Key Opportunities for Improvement:

- Color Contrast: One element has insufficient contrast; needs adjustment for better readability.
- Page Structure: Fix landmark nesting and ensure only one main landmark exists for better screen reader navigation.

Security Testing:

We conducted automated security vulnerability scanning using OWASP ZAP (Zed Attack Proxy), an industry-standard penetration testing tool for identifying security vulnerabilities in web applications.

Test Results:



Contexts
Default Context
Sites
https://devx360.app
GET:/
GET:apple-touch-icon.png
GET:favicon.ico
GET:manifest.json
GET:robots.txt
GET:sitemap.xml
static
http://devx360.app
GET:robots.txt
GET:sitemap.xml

Welcome to ZAP

ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications.

If you are new to ZAP then it is best to start with one of the options below.

News: ZAP 2.16.1 is available now Learn More

Alerts (10)
- Content Security Policy (CSP) Header Not Set (3)
- Missing Anti-clickjacking Header (3)
- Server Leaks Version Information via "Server" HTTP Response He
- Strict-Transport-Security Header Not Set (8)
- X-Content-Type-Options Header Missing (9)
- !ascanrules.useragent.name! (24)
- Information Disclosure - Suspicious Comments
- Modern Web Application (3)
- Re-examine Cache-control Directives (4)
- Retrieved from Cache (3)

Re-examine Cache-control Directives

URL:	https://devx360.app/
Risk:	Informational
Confidence:	Low
Parameter:	cache-control
Attack:	
Evidence:	
CWE ID:	525
WASC ID:	13
Source:	Passive (10015 - Re-examine Cache-control Directives)
Input Vector:	

Description:
The cache-control header has not been set properly or is missing, allowing the browser and proxies to cache content. For static assets like css, js, or image files this might be intended, however, the resources should be reviewed to ensure that no sensitive content will be cached.

Other Info:

Alerts 0 2 3 5 Main Proxy: localhost:8080

Current Status 0 0 0 0 0 0 0 0 0 0

Key Insights:

The application showed resilience against common web attacks with no high-severity vulnerabilities detected. Basic security measures are well-implemented, though several security headers are missing. The overall security posture is satisfactory for current deployment.

Key Opportunities for Improvement

- Implement missing security headers including Content Security Policy, X-Content-Type-Options, and anti-clickjacking protections
- Optimize cache-control directives to prevent potential sensitive data caching
- Remove server version information from HTTP responses to minimize information disclosure