

PSScript Comprehensive Testing Plan

- 2026



Executive Summary

This document outlines a comprehensive testing strategy for the PSScript PowerShell script management platform, following January 2026 industry best practices for full-stack React, Node.js, and PostgreSQL applications.

Testing Framework Strategy (2026): - **Vitest** for unit and component tests (faster than Jest, native ESM support) - **Playwright** for E2E tests (already configured in project) - **Supertest** for API endpoint testing - **Real PostgreSQL** with Docker for integration tests - **GitHub Actions** for CI/CD automation

Testing Levels & Scope

1. Smoke Tests (Critical Path)

Purpose: Verify basic functionality and critical user paths work **Execution Time:** ~5 minutes **Tools:** Manual browser testing + Playwright

Test Cases:

- ☐ **ST-01:** Application loads at http://localhost:3000
- ☐ **ST-02:** Login page renders correctly
- ☐ **ST-03:** Default login (green button) works
- ☐ **ST-04:** Manual login works (admin@example.com / Password123)
- ☐ **ST-05:** Dashboard renders after login
- ☐ **ST-06:** Navigation sidebar functional
- ☐ **ST-07:** Logout works
- ☐ **ST-08:** Backend API responds (health check)
- ☐ **ST-09:** Database connection active
- ☐ **ST-10:** Redis connection active

2. Frontend Unit Tests (React Components)

Purpose: Test individual components in isolation **Tools:** Vitest + React Testing Library **Target Coverage:** 80%+

Component Test Cases:

- ☐ **FU-01:** Login.tsx form validation
- ☐ **FU-02:** Login.tsx default login function
- ☐ **FU-03:** Dashboard.tsx stats cards rendering
- ☐ **FU-04:** Dashboard.tsx React Query hooks
- ☐ **FU-05:** Navbar.tsx user menu functionality
- ☐ **FU-06:** Sidebar.tsx navigation items
- ☐ **FU-07:** AuthContext.tsx state management
- ☐ **FU-08:** useAuth hook functionality
- ☐ **FU-09:** ScriptCard.tsx props handling

- ☐ **FU-10:** CategoryBadge.tsx rendering

3. Frontend Integration Tests

Purpose: Test component interactions and data flow **Tools:** Vitest + React Testing Library + MSW (Mock Service Worker)

Integration Test Cases:

- ☐ **FI-01:** Login flow with API mocking
- ☐ **FI-02:** Dashboard data fetching with React Query
- ☐ **FI-03:** Navigation state persistence
- ☐ **FI-04:** Auth token refresh flow
- ☐ **FI-05:** Protected route access control
- ☐ **FI-06:** Form submission with validation
- ☐ **FI-07:** Error boundary handling
- ☐ **FI-08:** Loading state transitions

4. Backend Unit Tests (Node.js/TypeScript)

Purpose: Test business logic and utilities **Tools:** Jest or Vitest **Target Coverage:** 85%+

Backend Test Cases:

- ☐ **BU-01:** User model password validation
- ☐ **BU-02:** User model login attempt tracking
- ☐ **BU-03:** JWT token generation
- ☐ **BU-04:** JWT token verification
- ☐ **BU-05:** bcrypt password hashing
- ☐ **BU-06:** Request validation middleware
- ☐ **BU-07:** Error handling utilities
- ☐ **BU-08:** Database query builders
- ☐ **BU-09:** Script hash calculation
- ☐ **BU-10:** File upload validation

5. Backend Integration Tests (API + Database)

Purpose: Test API endpoints with real database **Tools:** Jest/Vitest + Supertest + Docker PostgreSQL **Strategy:** Real database testing (2026 best practice)

API Endpoint Test Cases:

- [] **BI-01:** POST /api/auth/login (successful login)
- [] **BI-02:** POST /api/auth/login (invalid credentials)
- [] **BI-03:** POST /api/auth/login (user not found)
- [] **BI-04:** POST /api/auth/register (new user)
- [] **BI-05:** POST /api/auth/register (duplicate email)
- [] **BI-06:** POST /api/auth/refresh (token refresh)
- [] **BI-07:** GET /api/auth/me (user profile)
- [] **BI-08:** PUT /api/auth/user (update profile)
- [] **BI-09:** GET /api/scripts (list scripts)
- [] **BI-10:** POST /api/scripts (create script)
- [] **BI-11:** GET /api/scripts/:id (get script)
- [] **BI-12:** PUT /api/scripts/:id (update script)
- [] **BI-13:** DELETE /api/scripts/:id (delete script)
- [] **BI-14:** GET /api/categories (list categories)
- [] **BI-15:** POST /api/scripts/analyze (AI analysis)
- [] **BI-16:** GET /api/stats (dashboard stats)
- [] **BI-17:** GET /api/activity (recent activity)

6. E2E Tests (End-to-End with Playwright)

Purpose: Test complete user flows in real browser **Tools:** Playwright (already configured) **Browsers:** Chromium, Firefox, WebKit

E2E Test Cases:

- [] **E2E-01:** Complete login flow
- [] **E2E-02:** Complete logout flow
- [] **E2E-03:** Dashboard navigation
- [] **E2E-04:** Script upload flow
- [] **E2E-05:** Script analysis flow
- [] **E2E-06:** Script editing flow

- ☐ **E2E-07:** Script deletion flow
- ☐ **E2E-08:** Search functionality
- ☐ **E2E-09:** Category filtering
- ☐ **E2E-10:** User profile update
- ☐ **E2E-11:** AI assistant interaction
- ☐ **E2E-12:** Error handling (404, 500)
- ☐ **E2E-13:** Responsive design (mobile, tablet, desktop)
- ☐ **E2E-14:** Dark mode toggle
- ☐ **E2E-15:** Session persistence

7. Database Tests

Purpose: Test database operations and migrations **Tools:** Jest + pg (PostgreSQL client)

Database Test Cases:

- ☐ **DB-01:** Connection pool management
- ☐ **DB-02:** User CRUD operations
- ☐ **DB-03:** Script CRUD operations
- ☐ **DB-04:** Category CRUD operations
- ☐ **DB-05:** File hash deduplication
- ☐ **DB-06:** Vector embedding storage/retrieval
- ☐ **DB-07:** Transaction handling
- ☐ **DB-08:** Constraint violations
- ☐ **DB-09:** Migration execution
- ☐ **DB-10:** Index performance

8. UI/UX Manual Tests

Purpose: Verify user experience and visual consistency **Tools:** Manual browser testing + Chromatic (visual regression)

UI/UX Test Cases:

- ☐ **UI-01:** All buttons clickable and responsive
- ☐ **UI-02:** All forms accept valid input
- ☐ **UI-03:** All error messages displayed correctly

- ☐ **UI-04:** Loading states show spinners
- ☐ **UI-05:** Success messages display
- ☐ **UI-06:** Modal dialogs open/close
- ☐ **UI-07:** Dropdown menus function
- ☐ **UI-08:** Tooltips display on hover
- ☐ **UI-09:** Icons render correctly
- ☐ **UI-10:** Layout responsive on mobile (320px-480px)
- ☐ **UI-11:** Layout responsive on tablet (768px-1024px)
- ☐ **UI-12:** Layout responsive on desktop (1280px+)
- ☐ **UI-13:** Dark mode colors correct
- ☐ **UI-14:** Light mode colors correct
- ☐ **UI-15:** Focus states visible (accessibility)

9. Linting & Code Quality

Purpose: Ensure code quality and consistency **Tools:** ESLint, TypeScript compiler

Linting Test Cases:

- ☐ **LT-01:** Backend ESLint passes (src/backend)
- ☐ **LT-02:** Frontend ESLint passes (src/frontend)
- ☐ **LT-03:** TypeScript compilation succeeds (backend)
- ☐ **LT-04:** TypeScript compilation succeeds (frontend)
- ☐ **LT-05:** No console.error in production code
- ☐ **LT-06:** No unused imports
- ☐ **LT-07:** No unused variables
- ☐ **LT-08:** Consistent code formatting (Prettier)

10. Security Tests

Purpose: Identify security vulnerabilities **Tools:** npm audit, OWASP ZAP, manual testing

Security Test Cases:

- ☐ **SEC-01:** No dependency vulnerabilities (npm audit)
- ☐ **SEC-02:** SQL injection prevention (parameterized queries)
- ☐ **SEC-03:** XSS prevention (input sanitization)

- ☐ **SEC-04:** CSRF protection implemented
- ☐ **SEC-05:** JWT secret properly secured
- ☐ **SEC-06:** Password hashing (bcrypt 10+ rounds)
- ☐ **SEC-07:** Rate limiting on auth endpoints
- ☐ **SEC-08:** HTTPS enforcement (production)
- ☐ **SEC-09:** Secure headers (helmet middleware)
- ☐ **SEC-10:** Input validation on all endpoints
- ☐ **SEC-11:** File upload restrictions (size, type)
- ☐ **SEC-12:** Authentication required for protected routes
- ☐ **SEC-13:** Authorization checks (role-based access)
- ☐ **SEC-14:** Session timeout working
- ☐ **SEC-15:** Sensitive data not in logs

11. Performance Tests

Purpose: Ensure acceptable performance under load **Tools:** Artillery, Lighthouse, React DevTools Profiler

Performance Test Cases:

- ☐ **PERF-01:** Page load time < 3s (Lighthouse)
- ☐ **PERF-02:** Time to Interactive < 5s
- ☐ **PERF-03:** First Contentful Paint < 1.5s
- ☐ **PERF-04:** API response time < 200ms (avg)
- ☐ **PERF-05:** Database query time < 100ms (avg)
- ☐ **PERF-06:** 100 concurrent users supported
- ☐ **PERF-07:** No memory leaks in React components
- ☐ **PERF-08:** Bundle size < 500KB (main chunk)
- ☐ **PERF-09:** Images optimized (WebP/AVIF)
- ☐ **PERF-10:** Code splitting implemented

Testing Tools & Setup

Frontend Testing Stack

```
# Install Vitest for frontend unit/integration tests
npm install -D vitest @vitest/ui @testing-library/react @testing-library/jest-dom

# Install MSW for API mocking
npm install -D msw

# Configure Vitest
# Create vitest.config.ts in src/frontend
```

Backend Testing Stack

```
# Install testing dependencies
npm install -D jest supertest @types/jest @types/supertest

# Or use Vitest for consistency
npm install -D vitest
```

E2E Testing (Already Configured)

```
# Playwright is already installed
npx playwright test

# Run specific test
npx playwright test login.spec.ts

# Debug mode
npx playwright test --debug
```

Linting Commands

```
# Run all linting checks  
npm run lint
```

```
# Fix linting issues  
npm run lint:fix
```

```
# Backend only  
npm run lint:backend
```

```
# Frontend only  
npm run lint:frontend
```



Test Execution Plan

Phase 1: Smoke Tests (Day 1)

Duration: 1-2 hours **Priority:** CRITICAL - Execute all smoke tests manually - Document any failures - Fix critical issues immediately

Phase 2: Linting & Code Quality (Day 1)

Duration: 1 hour **Priority:** HIGH - Run ESLint on frontend and backend - Fix critical errors, document warnings - Run TypeScript compiler checks

Phase 3: Manual UI/UX Tests (Day 1-2)

Duration: 2-3 hours **Priority:** HIGH - Test all buttons and interactive elements - Verify responsive design - Check accessibility features

Phase 4: API Integration Tests (Day 2)

Duration: 3-4 hours **Priority:** HIGH - Test all authentication endpoints - Test all script management endpoints - Test with real database

Phase 5: E2E Tests (Day 2-3)

Duration: 4-6 hours **Priority:** MEDIUM - Run existing Playwright tests - Add missing test coverage - Fix failing tests

Phase 6: Unit Tests (Day 3-4)

Duration: 8-12 hours **Priority:** MEDIUM - Write unit tests for critical components - Write unit tests for business logic - Aim for 80%+ coverage

Phase 7: Security & Performance (Day 4-5)

Duration: 4-6 hours **Priority:** MEDIUM - Run npm audit - Run Lighthouse audits - Test load handling



Success Criteria

Minimum Acceptance Criteria

- ☒ All smoke tests pass (100%)
- ☒ No critical ESLint errors
- ☒ All authentication endpoints functional
- ☒ Dashboard renders and displays data
- ☒ No security vulnerabilities (high/critical)
- ☒ Lighthouse score > 80

Target Criteria

- ☒ 80%+ unit test coverage
- ☒ 90%+ integration test coverage
- ☒ All E2E tests pass
- ☒ Lighthouse score > 90
- ☒ All UI elements accessible (WCAG 2.1 AA)



CI/CD Integration (GitHub Actions)

Recommended Workflow

```
name: CI/CD Pipeline

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]

jobs:
  lint:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with:
          node-version: '18'
      - run: npm ci
      - run: npm run lint

  test-backend:
    runs-on: ubuntu-latest
    services:
      postgres:
        image: pgvector/pgvector:pg15
        env:
          POSTGRES_PASSWORD: postgres
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
    steps:
      - uses: actions/checkout@v3
```

- uses: actions/setup-node@v3
- run: npm ci
- run: npm run test:backend

test-frontend:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3
- uses: actions/setup-node@v3
- run: npm ci
- run: npm run test:frontend

e2e:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3
- uses: actions/setup-node@v3
- run: npx playwright install --with-deps
- run: npm run test:e2e
- uses: actions/upload-artifact@v3
 - if: failure()
 - with:
 - name: playwright-report
 - path: playwright-report/



Framework Improvements Recommended

1. Testing Infrastructure

- **Add Vitest** for faster unit tests (replace Jest if present)
- **Add MSW** for API mocking in frontend tests
- **Add testing-library/user-event** for better user interaction simulation
- **Add @axe-core/playwright** for automated accessibility testing

2. Code Quality Tools

- **Add Prettier** for consistent code formatting
- **Add husky** for pre-commit hooks
- **Add lint-staged** to lint only changed files
- **Add commitlint** for conventional commits

3. Performance Monitoring

- **Add Lighthouse CI** for automated performance tracking
- **Add Bundle Analyzer** to monitor bundle size
- **Add React DevTools Profiler** integration

4. Security Enhancements

- **Add Snyk** for continuous vulnerability scanning
- **Add OWASP Dependency Check**
- **Add SonarQube** for code quality and security analysis

5. Documentation

- **Add JSDoc** comments for public APIs
- **Add Storybook** for component documentation
- **Add API documentation** (Swagger/OpenAPI)
- **Add architecture decision records** (ADRs)

6. Developer Experience

- **Add Docker Compose** test environment
- **Add VS Code** testing extensions configuration
- **Add test coverage** reports with Istanbul/c8
- **Add test utilities** helpers for common patterns

Testing Best Practices (2026)

Based on industry research from January 2026:

1. **Test Features, Not Functions** - Focus on user-facing behavior rather than implementation details
2. **Use Real Databases** - Docker makes it easy to test against actual PostgreSQL
3. **Prefer Integration Tests** - They catch more bugs than pure unit tests
4. **Keep E2E Tests Focused** - Test critical paths only, they're slow and expensive
5. **Mock External Services** - Don't make real API calls in tests
6. **Use Migrations in Tests** - Same database setup as production
7. **Run Tests in Parallel** - Faster feedback with isolated test environments
8. **Fail Fast** - Run linting and unit tests before slower integration/E2E tests
9. **Visual Regression Testing** - Catch UI bugs with screenshot comparisons
10. **Accessibility Testing** - Automated checks with axe-core

References (January 2026)

Based on current industry standards:

- [Vitest vs Jest 30: Why 2026 is the Year of Browser-Native Testing](#)
 - [Top Testing Libraries for React in 2026](#)
 - [Node.js Testing Best Practices](#)
 - [Comprehensive Guide to Testing React Apps](#)
 - [Modern Frontend Testing with Vitest, Storybook, and Playwright](#)
 - [CI/CD Best Practices with GitHub Actions](#)
 - [Postgres and Integration Testing in Node.js Apps](#)
 - [Scalable React and Secure Node.js in 2025](#)
-

*Testing Plan Created: January 8, 2026 Framework: React + Node.js + PostgreSQL
Testing Strategy: Vitest + Playwright + Real Database Testing Status: 🚧 READY
FOR EXECUTION*

Generated 2026-01-16 21:23 UTC