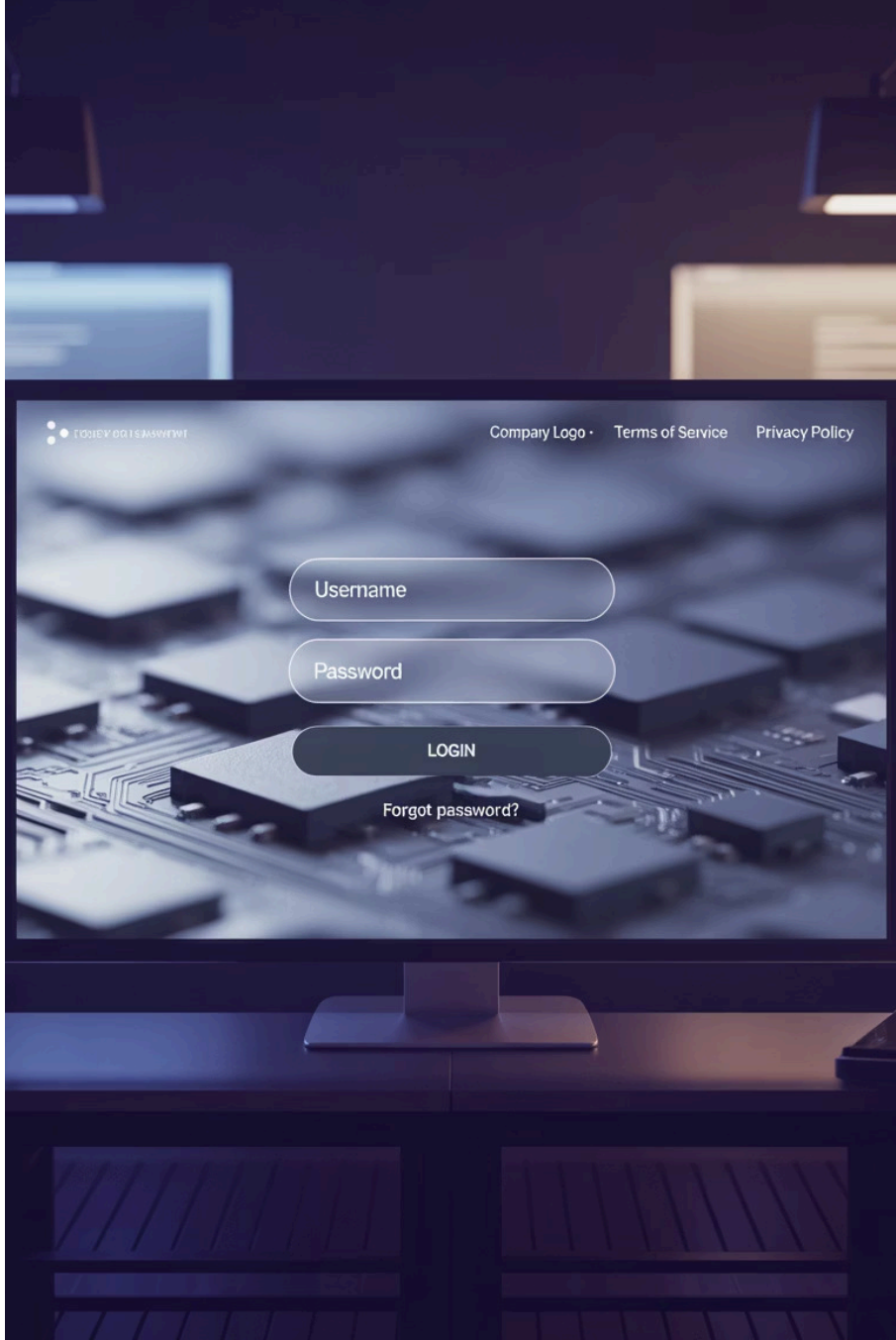
End-to-End Login Testing with Playwright

A comprehensive guide to implementing robust, maintainable login automation tests using Playwright. This workshop will equip QA engineers and developers with the knowledge to build reliable end-to-end tests that integrate seamlessly into CI/CD pipelines.



Problem Statement & Objectives

Our mission is to automate and validate the web application's login flow comprehensively. We need to ensure that correct credentials successfully authenticate users whilst incorrect or malformed inputs are handled gracefully. The resulting tests must be stable, readable, and production-ready for CI integration.

Primary Goals

- Verify core login functionality through happy path scenarios
- Validate comprehensive error handling for edge cases
- Ensure session persistence and logout functionality
- Generate actionable test artifacts and reports
- Seamlessly integrate with CI/CD workflows

Test Coverage Scope

01

Navigation

Verify users can successfully navigate to the login page from various entry points within the application.

02

Successful Authentication

Test valid credentials result in proper redirection to dashboard with expected UI elements visible and accessible.

03

Invalid Credentials

Validate error messaging for incorrect passwords, non-existent email addresses, and malformed input data.

04

Field Validation

Ensure empty required fields trigger appropriate validation messages and prevent form submission.

05

Session Management

Test "Remember me" functionality, token persistence, and proper session cleanup during logout.

06

Advanced Scenarios

Optional testing for 2FA flows, rate limiting, account lockouts, and social authentication pathways.

Acceptance Criteria & Prerequisites

Success Criteria

Test Implementation

All core login scenarios automated with comprehensive coverage of happy paths and edge cases.

Local Execution

Tests execute reliably with `npx playwright test` and generate detailed HTML reports for analysis.

CI Integration

Automated test execution on pull requests with proper artifact collection and failure reporting.

Security & Maintainability

Page Object Model implementation with secure credential handling via environment variables.

Local Environment

- **Node.js:** Version 18+ (LTS recommended)
- **Package Manager:** npm or yarn
- **Browser Drivers:** Installed via Playwright
- **TypeScript:** Recommended for enhanced maintainability

Project Initialisation

Follow these commands to establish your Playwright testing environment quickly and efficiently:

1. Create Project Directory

```
mkdir pw-login-automation && cd pw-login-automation
npm init -y
```

2. Install Dependencies

```
# Install Playwright test runner + dotenv
npm i -D @playwright/test dotenv
```

3. Install Browser Drivers

```
# Download browser binaries
npx playwright install
```

Pro Tip: The `npx playwright install` command downloads Chromium, Firefox, and WebKit browsers optimised for testing. This ensures consistent behaviour across different environments.

Recommended Project Structure

Organised Architecture

A well-structured project enhances maintainability and team collaboration. This architecture separates concerns effectively whilst keeping related files logically grouped.

```
pw-login-automation/
├── tests/
│   ├── login.spec.ts
├── pages/
│   ├── loginPage.ts
├── fixtures/
│   ├── credentials.ts
├── playwright.config.ts
├── .env # gitignored
├── package.json
└── README.md
```

- **tests/**

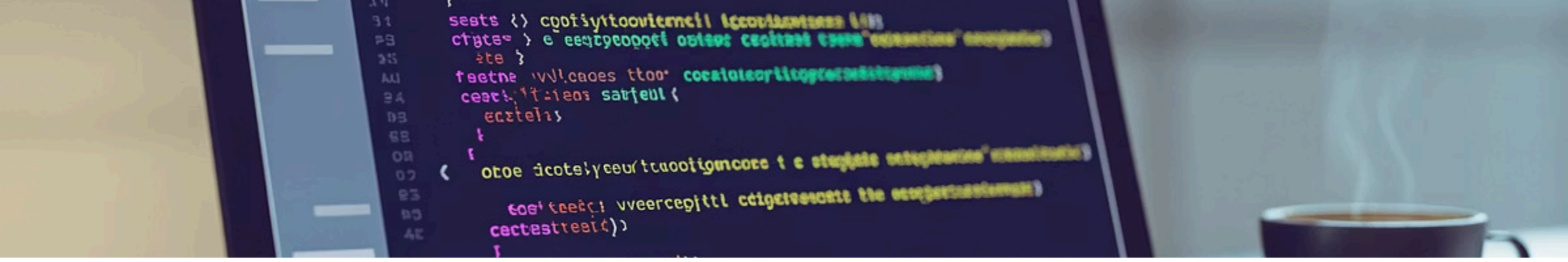
  Contains all test specifications

- **pages/**

  Page Object Model classes

- **fixtures/**

  Test data and utilities



Playwright Configuration

The `playwright.config.ts` file serves as the central configuration hub for your testing environment. This setup ensures consistent behaviour across local development and CI environments.

Key Configuration Features

Multi-Browser Testing

Configured to run tests across Chromium, Firefox, and WebKit for comprehensive compatibility coverage.

Failure Artifacts

Automatic screenshot and video capture on test failures, with traces enabled for retry attempts.

Parallel Execution

Optimised worker allocation for faster test execution whilst maintaining stability and resource management.

Remember to load environment variables using dotenv to access your BASE_URL and credentials securely during test execution.

Secure Credential Management

Environment Variables

Create a `.env` file (never commit to version control) containing your test environment configuration:

```
BASE_URL=http://localhost:3000
TEST_USER_EMAIL=test.user@example.com
TEST_USER_PASSWORD=Password123
```

Critical: Always add `.env` to your `.gitignore` file to prevent accidental credential exposure.

Best Practices

- Use CI secrets for production environments
- Implement credential rotation policies
- Separate test and production credentials
- Document required environment variables
- Validate environment setup in tests

Implementation Architecture

The Page Object Model (POM) provides a maintainable, reusable structure for your test automation. This approach encapsulates page-specific logic and promotes clean separation of concerns.

LoginPage Class

Encapsulates all login page interactions, selectors, and validation methods within a cohesive, reusable class structure.

Test Specifications

Clean, readable test cases that focus on business logic rather than implementation details or DOM manipulation.

Test Fixtures

Centralised test data management with credential helpers and utility functions for consistent test setup.

> "The Page Object Model transforms brittle, hard-to-maintain tests into robust, readable specifications that evolve with your application."

Essential Playwright Features

Trace on Retry

Captures comprehensive step-by-step execution traces during test retries, providing invaluable debugging information for flaky tests and intermittent failures.

Smart Artifacts

Automatically captures screenshots on failures and records videos for failed test runs, reducing debugging time significantly.

Network Mocking

Intercept and mock API responses using `page.route` for deterministic testing and isolation from backend dependencies.

Test Steps

Group related actions with `test.step` to create logical sections in reports and improve test readability and maintenance.

Network Interception Example

```
// Mock authentication API for deterministic testing
await page.route('**/api/auth/login', route => {
 route.fulfill({
 status: 200,
 body: JSON.stringify({ success: true, token: 'mock-jwt' })
 });
});
```

Test Execution & CI Integration

Local Development

Run All Tests

```
npx playwright test
```

Single Test File

```
npx playwright test tests/login.spec.ts
```

View HTML Report

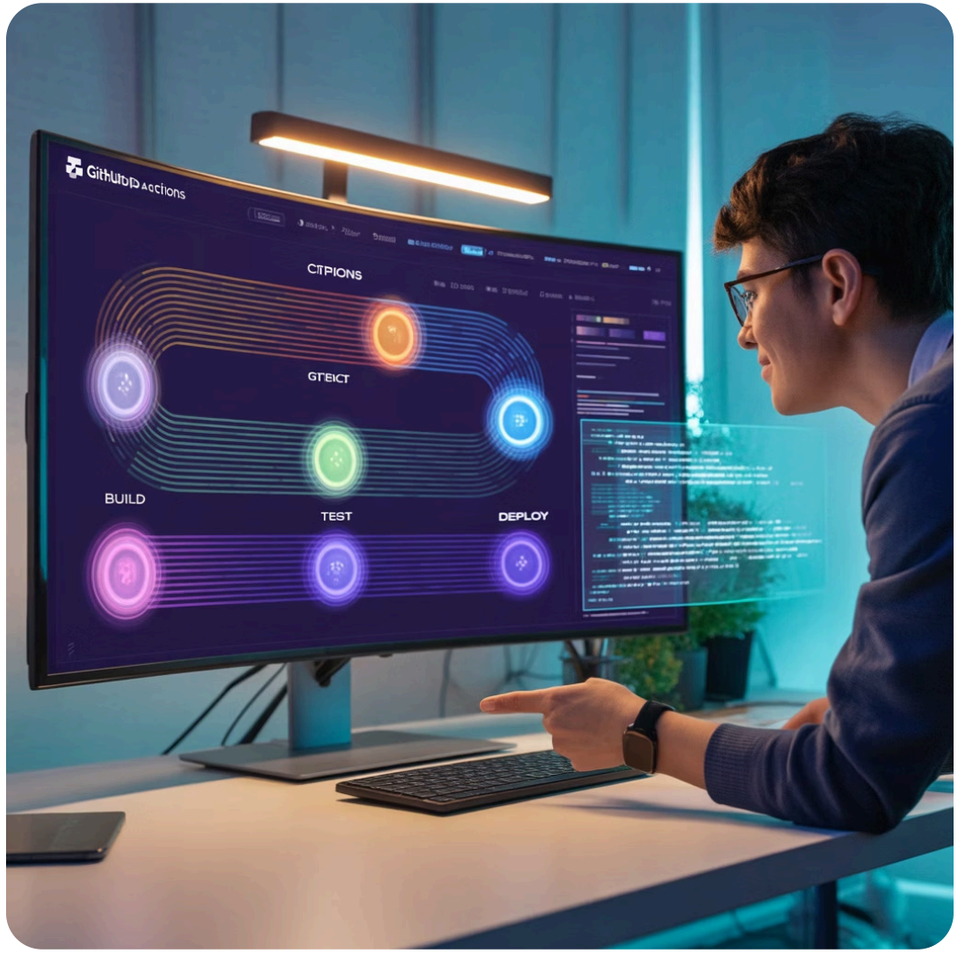
```
npx playwright show-report
```

Debug Mode

```
npx playwright test --headed
```

GitHub Actions CI

Implement automated testing in your CI pipeline with proper artifact collection and failure reporting. The workflow should trigger on pull requests and collect test results.



Configure the workflow to upload test reports, screenshots, and videos as GitHub artifacts for easy debugging of failed test runs.

Stability & Debugging Best Practices

Robust Selectors

Prefer `locator` and `expect(locator).toBeVisible()` over `page.waitForTimeout`. Use `data-testid` attributes for reliable element selection that survives UI changes.

Race Condition Prevention

Use `await Promise.all([page.waitForNavigation(), action])` to handle navigation and form submission races effectively.

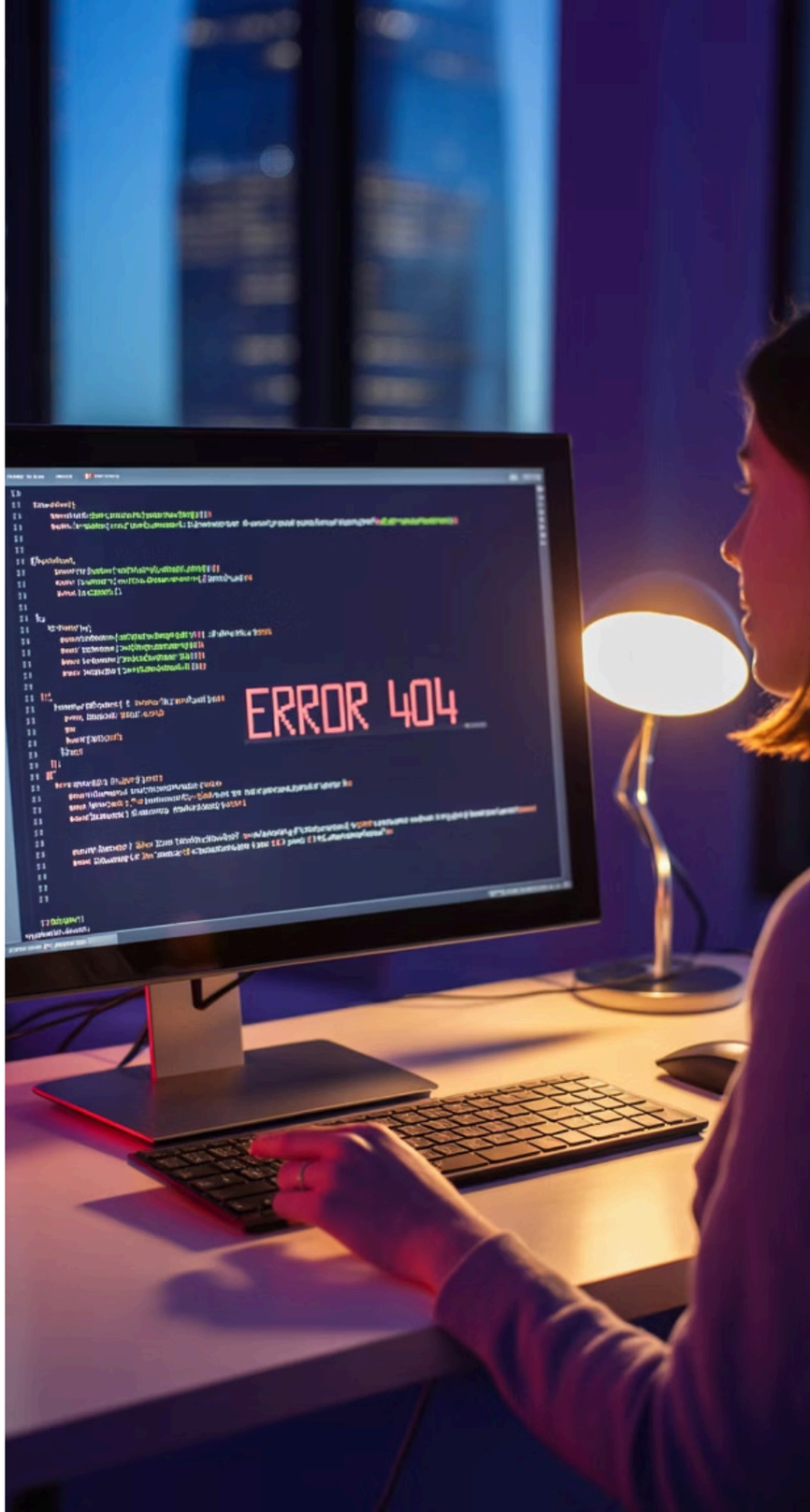
Comprehensive Tracing

Configure `trace: 'on-first-retry'` to capture detailed execution traces for debugging flaky tests and intermittent failures.

Network Isolation

Mock API responses for deterministic results and faster test execution. Keep network-dependent tests isolated from unit-style tests.

⚠ **Avoid Common Pitfalls:** Don't select elements by text content that might change. Configure retries conservatively to prevent masking real issues.



Project Deliverables Checklist

Ensure your login automation project meets professional standards with this comprehensive deliverable checklist. Each component contributes to a maintainable, secure, and production-ready testing solution.

1

Core Configuration

- `playwright.config.ts` with environment-specific settings
- TypeScript configuration for enhanced maintainability

2

Implementation Files

- `pages/loginPage.ts` implementing Page Object Model
- `tests/login.spec.ts` with comprehensive test coverage
- `fixtures/credentials.ts` for secure data management

3

CI/CD Integration

- `.github/workflows/playwright.yml` for automated testing
- Artifact collection and failure reporting configuration

4

Documentation & Security

- `.env.example` template (without secrets)
- `README.md` with setup and execution instructions
- `.gitignore` protecting sensitive files and build artifacts

Success Metrics

- Tests execute reliably in CI/CD pipeline
- Comprehensive error handling and reporting
- Zero hardcoded credentials in repository
- Clear documentation for team onboarding