# Playwright Documentation & Best Practices

## Introduction to Playwright

Playwright is an open-source automation framework for end-to-end testing of web applications. Developed by Microsoft, it supports multiple browsers (Chromium, Firefox, and WebKit) and platforms. Playwright provides powerful tools for testing modern web applications efficiently.

## Key Features

- Supports multiple browsers: Chromium, WebKit, Firefox, and Edge.  
- Runs tests across different platforms (Windows, Linux, macOS).  
- Supports mobile emulation and headless execution.  
- Built-in auto-waiting mechanism for reliable tests.  
- Native support for parallel and isolated test execution.  
- API testing capabilities.

## Installation

To install Playwright, use the following command:  
  
```  
npm init playwright@latest  
```  
This will set up Playwright with default configurations and test examples.  
  
Alternatively, if integrating into an existing project:  
  
```  
npm install --save-dev @playwright/test  
```  
To install browsers:  
  
```  
npx playwright install  
```

## Basic Test Example

Create a `tests/example.spec.js` file:  
  
```javascript  
const { test, expect } = require('@playwright/test');  
  
test('Basic Test', async ({ page }) => {  
 await page.goto('https://example.com');  
 await expect(page).toHaveTitle(/Example Domain/);  
});  
```  
  
Run the test using:  
  
```  
npx playwright test  
```

## Best Practices

### 1. Test Structure and Organization  
- Store tests in a dedicated `tests/` folder.  
- Use descriptive file names (e.g., `login.spec.js`, `checkout.spec.js`).  
- Group related tests using `describe()`.  
  
### 2. Selectors and Locators  
- Prefer `getByRole()`, `getByText()`, and `getByTestId()` over generic selectors.  
- Avoid brittle selectors like `XPath` and dynamic class names.  
- Example:  
```javascript  
await page.getByRole('button', { name: 'Submit' }).click();  
```

## Coding Standards

- Follow \*\*camelCase\*\* for variable names and functions.  
- Use \*\*ESLint\*\* for code formatting.  
- Keep tests \*\*independent\*\* and \*\*idempotent\*\*.  
- Maintain \*\*consistent indentation\*\* (2 or 4 spaces).  
- Add meaningful \*\*comments\*\* and \*\*documentation\*\*.  
- Avoid hard-coded values; use \*\*environment variables\*\* where possible.

## Conclusion

This guide provides an overview of Playwright, best practices, and coding standards to ensure robust, efficient, and maintainable test automation. Regularly update tests and follow industry best practices to improve test reliability and efficiency.