Playwright ベストプラクティスまとめ

概要

近年、Webアプリケーション開発においてEnd-to-End (E2E) テストの重要性が高まっています。E2Eテストは、ユーザーの視点からアプリケーション全体をテストすることで、実際のユーザーエクスペリエンスに近い形で品質を検証することができます。[1] Playwrightは、Microsoftが開発したオープンソースのE2Eテストフレームワークであり、クロスブラウザ対応、自動待ち機能、強力なデバッグツールなど、多くの利点を備えています。

本レポートでは、Playwrightのベストプラクティスについて調査し、E2Eテストを効果的に実施するための方法をまとめました。公式ドキュメント、ブログ記事、GitHubリポジトリ、Qiitaなどの情報を収集し、重要度と実現容易度を考慮して分類しました。

ベストプラクティス調査

公式ドキュメント

Playwrightの公式ドキュメント[2]では、以下のベストプラクティスが紹介されています。

- ユーザーに見える振る舞いをテストする：実装の詳細に依存せず、ユーザーが実際に目にする動作を検証する。
  - 例：ボタンをクリックしたときに、モーダルウィンドウが表示されることを確認するテストケースでは、モーダルウィンドウをDOMに追加するJavaScriptの関数を直接呼び出すのではなく、実際にボタンをクリックする操作をシミュレートする。
- テストの独立性を高める：各テストは独立して実行され、互いに影響を与えないようにする。
  - 例：テストケースごとに独立したブラウザコンテキストを使用し、テストケース間でCookieやローカルストレージが共有されないようにする。
- サードパーティの依存関係をテストしない：外部サイトやサービスへのリンクなど、制御できない要素はテスト対象外とする。
  - 例：外部の決済サービスへのリンクの動作確認は、E2Eテストの対象外とする。
- データベースを使用する場合は、データを管理し、ステージング環境に対してテストを実行する：本番環境のデータに影響を与えないように、テスト用のデータセットを用意する。
  - 例：テスト用のデータベースを構築し、テストケース実行前にテストデータを投入する。

ブログ記事・GitHubリポジトリ

公式ブログやGitHubリポジトリ、技術ブログ[2]から、以下のベストプラクティスに関する情報が得られました。

テストの安定性を向上させる

- 安定したセレクターを使用する[3]：要素を特定するためのセレクターは、変更されにくい属性や要

素を使用する。
    - 例：id 属性や data-testid 属性など、開発者が明示的に設定した属性をセレクターとして使用する。CSSクラスや要素の階層構造をセレクターとして使用するのは避ける。
- 固定遅延を避ける [8]：固定遅延はテストの不安定さにつながる可能性があるため、Playwrightの待機メカニズムを使用する。
    - 例：waitForSelector や waitForResponse を使用して、要素の表示やAPIレスポンスを待つ。
- テストを集中させ、分離する [3]：各テストは特定の機能に焦点を当て、他のテストから独立させる。
    - 例：ログイン処理のテストは、他の機能のテストとは別のファイルに記述する。

テストの可読性と保守性を向上させる

- エンドユーザーの視点からアサーションを書く [3]：ユーザーが期待する結果に基づいてアサーションを記述する。
    - 例：「ユーザーはログイン後に自分の名前が表示されることを期待する」というテストケースでは、expect(page.locator('text=ユーザー名')).toBeVisible() のように記述する。
- わかりやすいテスト名とステップタイトルを使用する [3]：テストの意図を明確にするために、記述的な名前とタイトルを使用する。
    - 例：test('ユーザーはログインできる', async ({ page }) => { ... }); のように、テストの内容がわかるように記述する。
- **Page Object Model (POM)** を使用する [4]：ページの要素や操作をクラスとして抽象化することで、テストコードの可読性と保守性を向上させる。
    - 例：ログインページの要素と操作を LoginPage クラスとして定義し、テストケースでは LoginPage クラスのメソッドを呼び出すことで、テストコードを簡潔にする。
- 状態をプログラムで設定する [4]：UIを操作して状態を設定するのではなく、APIやコードを使用して状態を直接設定する。
    - 例：ユーザーのログイン状態をテストする際に、UIからログイン操作を行うのではなく、APIを呼び出してログイン状態を設定する。

テストの実行効率を向上させる

- すべての関連ブラウザでテストする [3]：クロスブラウザ互換性を確保するために、主要なブラウザでテストを行う。
    - 例：Chrome、Firefox、Safariなどの主要ブラウザでテストを実行する。
- テストを自動化し、監視する [3]：CI/CDパイプラインにE2Eテストを組み込み、定期的に実行して結果を監視する。
    - 例：GitHub ActionsやCircleCIなどのCI/CDツールを使用して、プッシュやマージのたびにE2Eテストを実行する。
- **Playwright**のツールを活用する [3]：Playwrightが提供するデバッグツールやレポート機能を活用する。
    - 例：Playwright Inspectorを使用して、テスト実行中のブラウザの状態を確認する。Trace Viewerを使用して、テスト実行のトレース情報を分析する。

その他

- **Web**ファーストアサーションを使用する [4]：toBeVisible() など、Playwrightが提供するアサーションを使用する。
    - 例：要素が表示されていることを確認するために、

expect(page.locator('#myElement')).toBeVisible() のように記述する。
- **CI**の失敗には、ビデオやスクリーンショットの代わりに**Playwright**トレースビューアーを使用する[4]:トレースビューアーを使用することで、エラーの原因をより効率的に特定できる。
  - 例:CIでテストが失敗した場合、トレースビューアーでテスト実行時の詳細なログやスクリーンショットを確認し、エラーの原因を特定する。
- テストステップを使用して**Playwright**のドキュメントを改善する[13]:テストステップをドキュメントとして使用することで、テストケースの理解と保守を容易にする。
  - 例:テストケースに test.step を使用して、各ステップの目的を明確に記述する。
- テストをパラメータ化または動的に生成する[7]:反復的なテストを効率化するために、テストをパラメータ化または動的に生成する。
  - 例:複数のユーザーでログインテストを行う場合、ユーザー名をパラメータとして渡すことで、同じテストコードを繰り返し記述する必要性をなくす。
- テストを定期的にレビューおよび更新する[8]:アプリケーションの変更に合わせてテストスイートを最新の状態に保つ。
  - 例:アプリケーションのUIに変更があった場合、それに合わせてテストコードのセレクターやアサーションを更新する。
- ロケーターのベストプラクティスに従う[6]:要素を特定するためのロケーターは、ユーザーがページを認識する方法を反映した、安定したロケーターを使用する。
  - 例:getByRole を使用して、アクセシビリティの観点からも適切なロケーターを選択する。
  - 参考:
  - https://blog.autify.jp/article/best-practices-for-element-locating-in-e2e-testing
  - https://testing-library.com/docs/queries/about/#priority

今準拠すべきベストプラクティス

上記の調査結果に基づき、E2Eテストを効果的に実施するために、今準拠すべきベストプラクティスを以下の表にまとめました。

ベストプラクティス	重要度	実現容易度	説明	例
ユーザーに見える振る舞いをテストする	高	高	ユーザーの視点からアプリケーションの動作を検証することで、真の品質を担保する。	ボタンクリックをシミュレートしてモーダルウィンドウが表示されることを確認する。
テストの独立性を高める	高	中	各テストが独立して実行で	各テストケースで独立した

ベストプラクティス	重要度	実現容易度	説明	例
			きるようにすることで、保守性と信頼性を向上させる。具体的には、beforeEach や afterEach などのフックを活用して、各テストケースの前後で必要な初期化やクリーンアップ処理を行う。[2] 例えば、各テストケースの前にログイン処理を行い、テストケースの後にログアウト処理を行うことで、テストケース間の依存関係を排除することができます。	ブラウザコンテキストを使用する。
サードパーティの依存関係をテストしない	高	高	制御できない要素をテスト対象外とすることで、テストの安定性を確保する。	外部の決済サービスへのリンクはテスト対象外とする。
安定したセレクターを使用する	中	中	変更されにくい属性や要素をセレクターとして使用することで、テストの保守性を向上させる。	getByRole, data-testid 属性をセレクターとして使用する。

ベストプラクティス	重要度	実現容易度	説明	例
			data-testid 属性や、getByRole などのアクセシビリティに配慮したロケーターを使用することが推奨される。[6]	参考： https://blog.autify.jp/article/best-practices-for-element-locating-in-e2e-testing https://testing-library.com/docs/queries/about/#priority
Page Object Model (POM) を使用する	中	中	ページの要素や操作をクラスとして抽象化することで、テストコードの可読性と保守性を向上させる。POMを使用することで、コードの重複を減らし、要素ロケーターやアクションを一元管理することができます。	ログインページの要素と操作を LoginPage クラスとして定義する。
Webファーストアサーションを使用する	中	高	Playwrightが提供するアサーションを使用することで、テストコードの記述を簡潔にする。Playwrightのアサーションは、Webページの要素に対	expect(page.locator('#myElement')).toBeVisible() を使用して要素が表示されていることを確認する。

ベストプラクティス	重要度	実現容易度	説明	例
			する検証に特化しており、より正確で効率的なアサーションを記述することができます。	
テストを集中させ、分離する	中	中	各テストは特定の機能に焦点を当て、他のテストから独立させることで、テストの目的を明確にする。	ログイン処理のテストは、他の機能のテストとは別のファイルに記述する。
わかりやすいテスト名とステップタイトルを使用する	中	高	テストの意図を明確にするために、記述的な名前とタイトルを使用する。テスト名とステップタイトルを明確にすることで、テストの目的を理解しやすくなり、保守性が向上します。	test('ユーザーはログインできる', async ({ page }) => { ... }); のように、テストの内容がわかるように記述する。
固定遅延を避ける	中	高	固定遅延はテストの不安定さにつながる可能性があるため、Playwrightの待機メカニズム (waitFor など) を使用す	waitForSelector を使用して、要素が表示されるまで待つ。

ベストプラクティス	重要度	実現容易度	説明	例
			る。[8] waitForやwaitForSelector などのPlaywrightの待機メカニズムを使用することで、要素の読み込み完了を動的に待ち、テストの安定性を向上させることができます。	

結論

Playwrightは、強力な機能と柔軟性を備えたE2Eテストフレームワークであり、Webアプリケーションの品質向上に大きく貢献します。本レポートで紹介したベストプラクティスを参考に、テストコードの設計、実装、保守を行うことで、より効果的なE2Eテストを実施することができます。

具体的には、上記のベストプラクティスを適用することで、以下の効果が期待できます。

- テストの信頼性向上：安定したセレクターの使用、固定遅延の回避、テストの独立性確保などにより、テスト結果の信頼性を高めることができます。
- 保守性の向上：Page Object Modelの導入、わかりやすいテスト名とステップタイトルの使用などにより、テストコードの可読性と保守性を向上させることができます。
- 開発効率の向上：Playwrightのツールを活用することで、テストの作成、実行、デバッグを効率化し、開発プロセス全体の効率を向上させることができます。

これらの効果により、高品質なWebアプリケーションをより効率的に開発し、ユーザーに優れたエクスペリエンスを提供することができます。

引用文献

1. Playwright を使いこなすためのベストプラクティス #テスト自動化 ..., 1月 8, 2025にアクセス、https://qiita.com/ore88ore/items/bd5a1b65166027806096
2. Best Practices - Playwright, 1月 8, 2025にアクセス、https://playwright.dev/docs/best-practices
3. 9 Playwright Best Practices and Pitfalls to Avoid | Better Stack Community, 1月 8, 2025にアクセス、https://betterstack.com/community/guides/testing/playwright-best-practices/
4. Notes of Best Practices for writing Playwright tests - Rule of Tech, 1月 8, 2025にアクセス、https://ruleoftech.com/2024/notes-of-best-practices-for-writing-playwright-tests
5. 23. Best practices - Automation using Playwright - BigBinary Academy, 1月 8, 2025にアクセ

ス、https://courses.bigbinaryacademy.com/learn-qa-automation-using-playwright/best-practices/
6. Playwright Locators: Best Practices for Robust Test Automation - Bondar Academy, 1月 8, 2025にアクセス、https://www.bondaracademy.com/blog/playwright-locators-best-practices
7. Playwright framework best practices/structure? : r/QualityAssurance - Reddit, 1月 8, 2025にアクセス、
https://www.reddit.com/r/QualityAssurance/comments/1248csz/playwright_framework_best_practicesstructure/
8. 7 Key Playwright Techniques to Eliminate Test Flakiness and Boost ..., 1月 8, 2025にアクセス、
https://blog.magicpod.com/7-key-playwright-techniques-to-eliminate-test-flakiness-and-boost-reliability
9. Mastering Playwright: Best Practices for Web Automation with the Page Object Model | by Luc Gagan | Medium, 1月 8, 2025にアクセス、
https://medium.com/@lucgagan/mastering-playwright-best-practices-for-web-automation-with-the-page-object-model-3541412b03d1
10. clerk/playwright-e2e-template: Playwright End to End testing suite template - GitHub, 1月 8, 2025にアクセス、https://github.com/clerk/playwright-e2e-template
11. Setting up CI - Playwright, 1月 8, 2025にアクセス、https://playwright.dev/docs/ci-intro
12. Playwright GitHub Repositories: Testing & Automation Mastery - testomat.io, 1月 8, 2025にアクセス、https://testomat.io/blog/best-playwright-github-repositories/
13. Improve Your Playwright Documentation with Test Steps - Checkly, 1月 8, 2025にアクセス、
https://www.checklyhq.com/blog/improve-your-playwright-documentation-with-steps/