**サンプルプロジェクト:**

Vue 3、Vite、TypeScript、テストツール（Vitest、Jest、Cypress、Vue Test Utils）を使用し、ログイン機能を作成してテストを実施

**実験: 各ツールでのログイン機能テスト比較**

上記の評価を踏まえ、実際にVue 3＋TypeScriptで簡単なログイン機能を実装したサンプルプロジェクトを用いて各ツールの挙動を比較しました。ログインフォームコンポーネントには以下のような機能を持たせています。

1. ユーザー名（メールアドレス）とパスワードの入力フォーム:

* メールアドレスは必須項目であり、正しい形式（test@example.com）でない場合、エラーメッセージを表示する。
* パスワードは必須項目であり、未入力の場合、エラーメッセージを表示する。
* 両フィールドが埋まっていない場合、ログインボタンを無効化する。

1. パスワード強度チェック:

* パスワードが一定の強度を満たさない場合、警告メッセージを表示します。具体的には以下のルールを適用しています。
* パスワードが8文字未満の場合、「8文字以上の長さが必要」という警告を表示。
* パスワードに大文字が含まれていない場合、「1つの大文字が含まれている必要がある」という警告を表示。
* パスワードに数字が含まれていない場合、「1つの番号が含まれている必要がある」という警告を表示。

1. 認証処理:ダミーの認証APIを想定し、ログインボタン押下時に以下の処理を行います。

* 正しい認証情報（test@example.com / Password123）が入力された場合、ログイン成功メッセージ（「ログインが成功」）を表示。
* 誤った認証情報が入力された場合、エラーメッセージ（「ログインメールとパスワードは一致しません」）を表示。
* 認証処理中はログインボタンのテキストを「ログイン中...」に変更し、ユーザーに処理中の状態をフィードバック。

1. 状態管理:

* ログイン成功時、失敗時、および処理中に適切なメッセージやUI状態を表示する。
* 各入力フィールドのバリデーションは、フォーカスが外れたタイミング（@blurイベント）で即時反映される。

このコンポーネントに対し、以下のテストを作成しました。

* **Vitest＋Vue Test Utils**: ユニットテストとしてコンポーネントをマウントし、バリデーションロジックや表示メッセージを検証するテストを計6件作成。
* **Jest＋Vue Test Utils**: 上記Vitestとほぼ同様のテスト内容をJest環境で実装。
* **Cypress**: E2Eテストとして、実際にブラウザ上でフォームに値を入力し、ボタンの有効/無効切り替えやメッセージ表示、成功/失敗時の挙動を確認するシナリオテストを5件作成（正常系1件、異常系4件）。

各テストsuiteを実行し、「テスト実行時間」と「バグ検出率（仕込んだ不具合を検出した割合）」を測定しました。不具合シナリオとしては以下の3点を意図的に実装に忍ばせました。

1. **バグA**: パスワードの流れのチェックのロジックに誤りがあり、パスワード筋力警告が表示されない。
2. **バグB**: 認証失敗時に本来表示されるべきエラーメッセージが表示されない。
3. **バグC**: フィールド未入力時にログインボタンを無効化すべきところ、バグで無効化されずクリックできてしまう。

各ツールで上記のバグを検出できるか確認するため、対応するテストケースを用意しました（ただしユニットテストではバグBについてのシナリオを網羅しない想定）。

**テスト実行時間の比較結果**（ローカル環境: MacBook Pro Intel, Node 18使用）:

**ソースコードGithub Url:**

Vue3 + TypeScript + Jest: <https://github.com/bishaloec/Vue-Test-Tools>

Vue3 + TypeScript + Vitest/Cypress: <https://github.com/bishaloec/Vue3-Vitest-Cypress>



Vitest＋VTU: 約2.01秒（6件のユニットテスト実行)。非常に高速で、ほぼ瞬時に完了しました。

**A screen shot of a computer

AI-generated content may be incorrect.**

Jest＋VTU: 約3.8秒（同じく6件)。Vitestよりわずかに時間がかかりましたが、それでも2秒以内で完了し十分に速いです。



Cypress: 約7.0秒（同じく6件シナリオ実行、ブラウザ起動含む）。最初のテストでブラウザとアプリを立ち上げる処理があるため時間を要しました。

以上の結果から、**ユニットテスト系（Vitest/Jest）では主にロジック面の不具合検出に強く、E2Eテスト（Cypress）はUIや統合面の不具合検出に強い**ことが確認できました。ユニットテストは「内部状態や各関数の網羅的なチェック」に適しており、E2Eテストは「ユーザー体験に影響する一連の流れの検証」に適しているため、それぞれ効率よくバグを見つけられる箇所が異なるのです。

**考察: 結果の分析と改善点・拡張の可能性**

実験の結果、VitestおよびJestを用いたユニットテストは極めて高速に実行でき、ロジックの細部に潜むバグ検出に有効であることがわかりました。一方、CypressによるE2Eテストは実行に時間がかかるものの、実際の画面操作を通じて**統合上の不具合**を洗い出す上で強力であると確認できました。Vue Test Utilsはユニットテスト内でVueコンポーネントを扱うための便利な手段を提供し、特にVitestとの組み合わせではVue 3コンポーネントのテストが快適に行えることが示されました。

**改善点**として、現実的には**すべての種類のテストを単独で完璧にするのは非効率**です。ユニットテストとE2Eテストにはそれぞれ得意分野があるため、組み合わせて用いることで相補的に品質を高めるのが理想的です。例えば今回のようにまずVitest等でカバーできるロジック部分を高速にテストし、UI全体の流れや外部連携部分はCypressで定期的にテストするといった戦略が考えられます。このように役割分担させることで、CI上でもユニットテストはプルリクエストごとに実行して素早いフィードバックを行い、E2Eテストは夜間ビルドやリリース前にまとめて実行して総合的な検証を行うといった運用が可能です。

**拡張の可能性**としては、他の種類のテストやツールの導入があります。今回取り上げなかったものの、例えば**統合テスト**（ユニットとE2Eの中間的なテスト）にはPlaywrightやPuppeteer、またVueコンポーネントに特化した**スナップショットテスト**や**Visual Regression Testing**（外観の変化検知）にはStorybook＋Chromatic等を組み合わせる方法もあります。さらに、Vue Test Utilsと相性の良い**Vue Testing Library**を使えば「ユーザー視点のクリーンなテストコード」を書くことができ、テスト可読性と保守性を向上させる余地もあります。プロジェクトの要求に応じてこれらを拡張的に取り入れることで、テストの網羅性と信頼性を一層高めることが可能です。