Web標準

2024年12月5日　23:17

ツール選定

<table>
<tr><th></th><th></th><th>選定理由</th><th>他候補</th><th>適用状況</th><th>React bullet-proofの選定</th></tr>
<tr><td>ビルドツール</td><td>Vite</td><td>開発時のビルドにおいては、モジュールを動的に交換できる。現時点で最速のビルドツール。
JavaScriptビルドツールの整理 各ツールの機能と依存関係</td><td>ビルドツールとして、トランスパイル・ファイル統合・ミニファイ・開発サーバ実行ができるのは下記三つ。
・WebPack
・Parcel
・Next.js
そのなかで、開発時のビルドから実行までが最速なのがVite。</td><td>各自の開発環境、CI/CDで常時使用</td><td>Vite</td></tr>
<tr><td>FEテスト用モックサーバ</td><td>MockServiceWorker</td><td>FE側でリクエストをインターセプトして返す形式なので、本番コードに影響を与えずにテストができる。フロントエンドで完結するので単体テストにも組み入れやすい。
Next.jsにおけるモックサーバ</td><td>API Router
JSON Server
<table><tr><th>項目</th><th>API Routes</th><th>JSON server</th><th>MSW</th></tr><tr><td>設定の簡単さ</td><td>○</td><td>△</td><td>×</td></tr><tr><td>環境によらない</td><td>－（Next の機能）</td><td>○</td><td>△</td></tr><tr><td>本番コードに与える影響</td><td>△</td><td>△</td><td>◎</td></tr></table></td><td>通信が関わる処理のユニットテストで使用。</td><td>MockServiceWorker</td></tr>
<tr><td>静的解析
サイクロマティック複雑度解析</td><td>ESLint</td><td>Web開発においてデファクト標準的に使用されているツール</td><td>なし</td><td>各自の開発環境、CI/CDで常時使用。
12/20時点でWarning 0 件を維持。
サイクロマティック複雑度はmax10以下。</td><td>ESLint</td></tr>
<tr><td>CSS</td><td>Styled Component</td><td>CSSモジュールやTailWindと比較して、再利用や管理が容易で大規模なFEアプリに向いていること。
パフォーマンスは他より劣るが、CSSのパフォーマンスがネックになるようなUIではない。</td><td><table><tr><th></th><th>メリット</th><th>デメリット</th><th></th></tr><tr><td>CSS モジュール</td><td>• コンポーネントベースのスタイリングが可能で、スタイルのカプセル化が容易
• クラス名が自動的に一意になるため、スタイルの競合や衝突のリスクが低い。
• CSSファイルをコンポーネントごとに分割し、必要なスタイルのみを読み込むことができます。</td><td>• スタイルの共有が困難であり、スタイルの再利用性が低くなる場合があります。
• クラス名の命名に工夫が必要であり、大規模なプロジェクトでは管理が難しくなることがあります。</td><td></td></tr><tr><td>TailwindCSS</td><td>• 事前に定義されたクラスを使用してスタイリングを行うことで、迅速かつ一貫性のあるデザインを実現できる
• カスタマイズ性が高く、自由度のあるデザインが可能。
• クラスベースのスタイリングにより、スタイルの再利用性が向上する</td><td>• スタイルの管理が容易でなく、大規模なプロジェクトでは混乱する可能性がある
• プロジェクトのスタイルがテーマに依存するため、他のプロジェクトへの移植が難しい。</td><td></td></tr><tr><td>Styled Component</td><td>• コンポーネントベースのスタイリングが可能で、スタイルとコンポーネントが密接に結びついている
• JavaScriptの力を借りて、動的なスタイリングが可能
• スタイルの再利用性が高く、コンポーネント単位でのカプセル化が容易</td><td>• インラインスタイルが増える可能性があり、パフォーマンスに影響を及ぼす場合があります。</td><td></td></tr></table></td><td>StyleComponentでUIを実装中。</td><td>TaiWindCSS</td></tr>
<tr><td>コードレビュー</td><td>GitHubCopilot</td><td>他のAIを使ったコードレビューツールは、GitHubアカウントが必要そう。セキュリティ面においても使用するのが難しい。</td><td>ChatGPT　CodeReview（GitHubアカウントが必要）
CodeRabbit（GitHubアカウント必要）</td><td>レビュワーに確認してもらう前の事前チェックに使用中。</td><td></td></tr>
<tr><td>単体テストツール
カバレッジツール</td><td>Vitest + React testing Library</td><td>現時点で最速の単体テストツール。カバレッジの測定も可能。</td><td>Jest
機能的にはVitestとJestはほぼ同等だが、Vitestのほうがテスト実行速度は高速。</td><td>12/20時点で単体テストのC0カバレッジ47%。</td><td>Jest＋React testing Library</td></tr>
<tr><td>パッケージ静寂性チェック</td><td>npm audit</td><td>無料ツールではデファクト。</td><td>Snykなどの有料ツール。</td><td>12/20で脆弱性High 0 件。</td><td></td></tr>
<tr><td>コードフォーマッター</td><td>Prettier</td><td>WebFEにおいてはデファクト。</td><td>なし</td><td>使用中。</td><td>Prettier</td></tr>
<tr><td>UIデザイン確認ツール</td><td>Storybook</td><td>Web開発ではデファクト</td><td>なし</td><td>使用中</td><td>Storybook</td></tr>
<tr><td>国際化ツール</td><td>i18n</td><td>Webにおいてはデファクト</td><td>なし</td><td>使用中</td><td></td></tr>
<tr><td>CI/CD</td><td>Teamcity</td><td>V192やKSWで使用実績が豊富</td><td>Jenkins
Azure DevOps</td><td>リンクを張る</td><td></td></tr>
<tr><td>パフォーマンス測定</td><td></td><td></td><td>JMeter
LightHouse</td><td></td><td></td></tr>
<tr><td>E2Eテストツール</td><td></td><td></td><td>PlayWright, Cypress, Cypress</td><td></td><td>PlayWright</td></tr>
<tr><td>セキュリティテスト</td><td></td><td></td><td>OWASP ZAP</td><td></td><td></td></tr>
</table>

技術選定

CSS	Emotion			
状態管理	Redux			

アーキテクチャ選定

状態管理	Redux			
データ更新	ポーリング			
ニーモニック				
ドメイン駆動				

ぎゅっと絞ると、何を解決するために「大事なこと」を伝えるのか？
物件化？　物件でないと、組織。

メンバの年代や、仕事内容が多様化してきても、大事なことが共通基盤となり、

1. **拠り所となる価値観や行動指針がないと、組織を成長・拡大させていくことができない。**
   （人数は増えても、チーム別にばらばらに開発しているだけで、組織力は向上しない。）

今後、新しいリーダを増やしながら組織を拡大するためには、よりどころとなる大事なことを共通理解にする必要がある。

組織の成長を持続するため。
組織内に一体感や帰属意識が生まれ、従業員同士の信頼関係が強化されます。これにより、チームワークが向上し、業務の生産性も高まります
[4]

貼り付け元 <https://www.perplexity.ai/search/zu-zhi-tositeda-shi-nakoto-ren-L5kHoJpgQhqfINkL.CkX3g>

チーム（横）、年代を超えたよりどころ