React研修2025 第1回

2025年2月23日

Keita Ito

目的

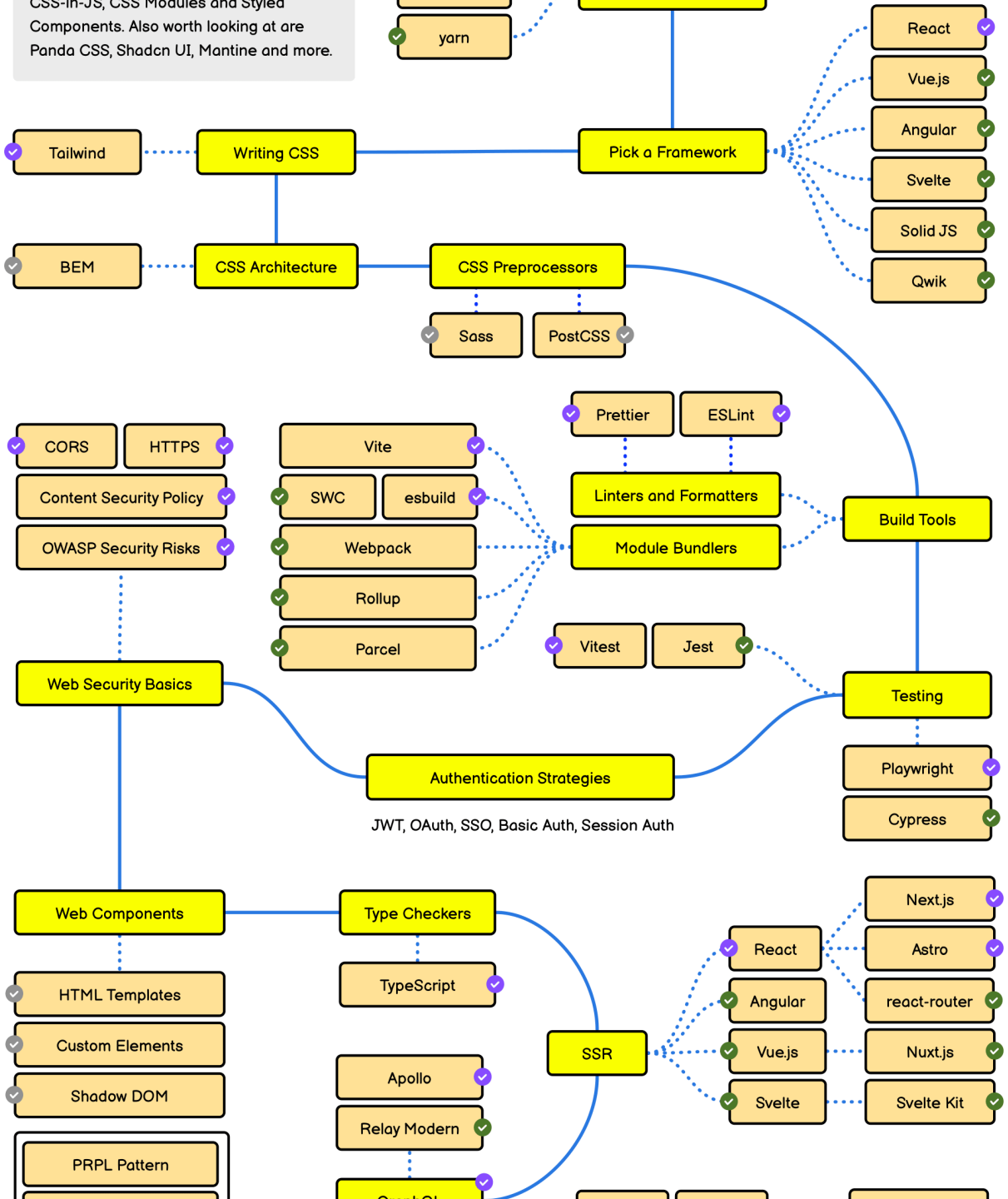
- Reactを用いてウェブアプリを作成できるようになる
- 保守性が高いコードを書けるようになる
  保守性が高い=他の人が読みやすい（チーム開発では必須）
  自分の過去のコードにも影響
  ベストプラクティス多数存在→たくさん書いて、経験を積むしかない

今回の流れ

i. ウェブサイトが表示される仕組み
ii. ウェブ開発の歴史
iii. Reactの必要性
iv. モダンなウェブ開発の流れ

Frontend Developer - roadmap.sh（参考）

https://roadmap.sh/frontend



ウェブサイトが表示される仕組み

リクエスト

- HTTPリクエスト（GET、POST、PUT、DELETE）
- ブラウザ、JavaScript (ajax)、curl

レスポンス

- HTML、CSS、JavaScript、画像、動画、フォント、JSON


center

SPAとMPA

SPA (Single Page Application)

- 1つのHTMLファイルで全てのページを表示
- ページ遷移時にJavaScriptでコンテンツを変更
- 例）React、Vue、Angular

MPA (Multi Page Application)

- 複数のHTMLファイルでページを表示
- ページ遷移時に新しいHTMLファイルを読み込む
- 例）PHP、Ruby on Rails

Next.js...?

DOM

- DOM = Document Object Model
- ツリー構造で表現される（XMLの形そのまま）
- JavaScriptからアクセス可能で、好き放題できる
  `jQuery React Vue`.jsなどのライブラリ・フレームワークがDOMを動的に操作

```
<!DOCTYPE html>
<html>
  <head>
    <title>DOM</title>
  </head>
  <body>
    <h1>Hello, DOM!</h1>
  </body>
</html>
```

DOM操作の例

JavaScriptでDOMを操作する例

```
<body>
  <h1 id="title">Hello, DOM!</h1>
  <script>
    const title = document.getElementById("title");
    title.style.color = "red";
  </script>
</body>
```

ウェブ開発の歴史

> 参考 https://speakerdeck.com/recruitengineers/react-yan-xiu-2024

1990年代

- CGI (Common Gateway Interface)
- Perl、PHP、Java Servlet
- HTML 4.0 (1997)、CSS (1996)

サーバーサイドで動的なコンテンツを生成するための仕組み = MPA

SSR (Server Side Rendering) = サーバーサイドでHTMLを生成

 center

MVCフレームワーク

MVC = Model View Controller
例）Ruby on Rails、Spring Framework、Django、Laravel

- Viewのテンプレート

```
<body>
  <h1>Hello, {{ name }}</h1>
</body>
```

2000年代

- Ajax (Asynchronous JavaScript and XML)

```
fetch("https://api.example.com/data")
  .then((response) => response.json())
  .then((data) => console.log(data));
```

- jQuery (2006)

```
$.ajax({
  url: "https://api.example.com/data",
  success: function (data) {
    console.log(data);
  },
});
```

- jQueryを使わない例

```
<body>
  <h1 id="title">Hello, jQuery!</h1>
  <script>
    const title = document.getElementById("title");
    title.style.color = "red";
  </script>
</body>
```

- jQueryの例

```
<body>
  <h1 id="title">Hello, jQuery!</h1>
  <script>
    $("#title").css("color", "red");
  </script>
</body>
```

jQueryはDOM操作を簡単にするライブラリ

2010年代

SPA (Single Page Application)
CSR (Client Side Rendering) = クライアントサイドでHTMLを生成

- AngularJS (2010) by Google
- React (2013) by Facebook
- Vue.js (2014)

通信の流れ

- 単一のHTMLファイルを取得（真っ白な画面）
  `<div id="root"></div>` だけが存在
- JavaScriptでコンテンツを取得
  ajaxでデータを取得し、JavaScriptでコンテンツを生成

  元のHTML `<div id="root"></div>` + JSが取得したデータ `{ "name": "Alice" }`
  ↓
  生成されたHTML

```
<div id="root">
  <h1>Hello, Alice</h1>
</div>
```

SPA (CSR) の課題

- 初期表示が遅い
  JavaScriptを実行するまで、何も表示されない
- SEO (Search Engine Optimization) が難しい
  GoogleなどのJavaScriptを実行しない
  → ページの内容が取得できない
- ユーザーの体験が悪い
  ページ遷移時に白い画面が表示される

2010年代後半

SSR (Server Side Rendering) + SPA (CSR)

- Next.js (2016) by Vercel
- Gatsby (2017) by GatsbyJS
- (Nuxt.js (2017) by NuxtJS)

サーバーサイドでHTMLを生成してから、クライアントサイドでJavaScriptを実行

- 初期表示が速い・SEOがしやすい・ユーザーの体験が向上

i. サーバーサイドでHTMLを生成
Node.jsでReactをレンダリング、HTMLを生成
この時点でJavaScriptを実行しなくても、コンテンツが表示される（≠真っ白）

ii. その後はSPAと同じ

レンダリング方法まとめ

レンダリングの種類	生成方法	フレームワーク例
CSR (Client Side Rendering)	クライアントサイドでHTMLを生成	~~Recat (CRA)~~、Vite
SSR (Server Side Rendering)	サーバーサイドでHTMLを生成 リクエストが来るたびにHTMLを生成 = 動的	Next.js、Remix
SSG (Static Site Generation)	静的サイトジェネレーション ビルド時に全てのHTMLを生成 = 静的	Next.js、Gatsby
ISR (Incremental Static Regeneration)	SSR+SSG 例）10分キャッシュ	Next.js、Gatsby

今後の研修予定について

- 第二回
  "良い"Reactの書き方を学ぶ
- 第三回
  Next.jsを利用してより良いウェブサイトの作成を行う
  バックエンドも絡んできます
- 第四回
  最終発表会

最終発表会の概要 (1)

- React研修で学んだことを活かし、ウェブアプリを作成
- 学んだことをアウトプットする機会
- 3月16日 17:00-18:00
- 対面でもオンラインでも可
- 4分間のプレゼン（デモ含む） + 1分間の質疑応答

最終発表会の概要 (2)

以下の内容を含めてください

i. 作成したウェブアプリの概要
ii. デモ
iii. 技術的に工夫した点
iv. 今後の課題・展望
v. 質疑応答

開発環境の整備できていますか？

Node.jsのインストール

ターミナルで`node -v`と`npm -v`を打って正常にバージョンが出るか
Node.jsは^22、npmは^11のバージョンを前提に説明を行います。

Reactが動くか検証

参考 TypeScript

https://typescriptbook.jp/