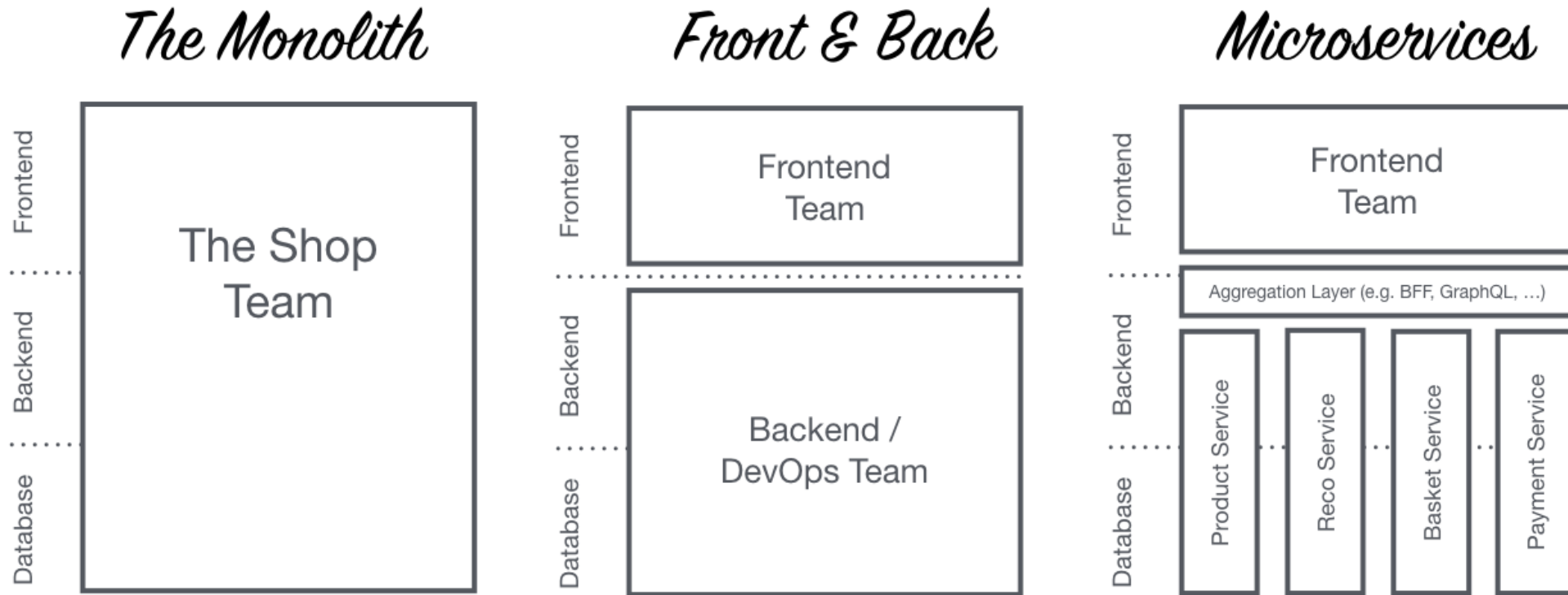


MicroFrontends調査報告

2020/10/26

MicroFrontendsとは

マイクロサービスの考え方をフロントエンドに拡張したものです。



モノリシックなフロントエンド

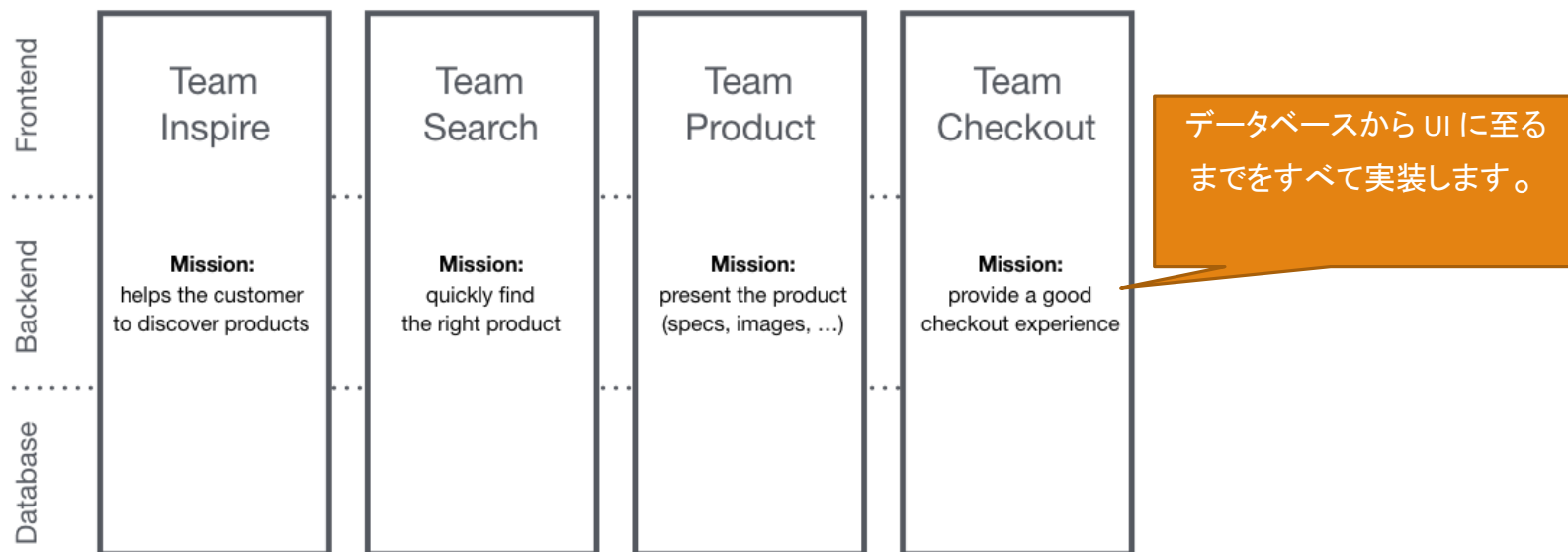
開発をすすめていくと、特に複数のチームで管理している場合
フロントエンド層が肥大化して管理が難しくなりがちです。
これを「モノリシックなフロントエンド」と呼びます。

MicroFrontendsの考え方

それぞれのチームは特定の領域やミッションに特化しています。

チームは組織横断的です。

End-to-End Teams with Micro Frontends



Web開発のトレンド

現在のWebのトレンドは多機能なSPAです。

SPAとはSPA(Single Page Application)の略です。

単一のWebページでアプリケーションを構成する設計構造の名称です。

特徴:

- ・単一のWebページでコンテンツ切り替えを行う
- ・ページ遷移の必要がなくなり
- ・ブラウザの挙動に縛られないWeb表現を可能

Web開発のトレンド

幅広いUIを実装できることで、ネイティブアプリの代用として使うことができます。

何かしらのWebアプリを開発し、そのiOSアプリ、Androidアプリをリリースする流れでは非常に手間がかかってしまいます。

仮にWebアプリのサーバサイドをNode.jsで書いたとしても、HTML/CSS、JavaScript、Swift、Javaの記述が必要です。

しかし、SPAを導入することで、オフラインでのページ閲覧、プッシュ通知、ホーム画面からの起動など、ネイティブアプリで主に使われるような機能を実装することができます

なぜ MicroFrontends

- ・より小さく
- ・よりまとまりがあり
- ・保守しやすいコードベースです
- ・分離された自律的なチームを持つ、よりスケーラブルな組織
- ・フロントエンドの一部を以前よりも段階的にアップグレード、更新、または書き換える機能

スケーラブル: 利用者や仕事の増大に適応できる能力・度合いのこと

MicroFrontendsのコアアイデア

- **Be Technology Agnostic**

各チームは技術において他チームの影響を受けません。

各チームは、他のチームと調整しなくても、スタックを選択してアップグレードできる必要があります。カスタム要素は、他のユーザーに中立的なインターフェイスを提供しながら、実装の詳細を隠すための優れた方法です。

- **Isolate Team Code**

技術同様、実際のコードも共有しません。また、状態やグローバル変数、コーディングルールなども互いに依存しないよう独立して機能させます。

- すべてのチームが同じフレームワークを使用している場合でも、ランタイムを共有しないでください。自己完結型の独立したアプリを構築します。共有状態またはグローバル変数に依存しないでください。

MicroFrontendsのコアアイデア

- **Establish Team Prefixes** チームの Prefix を定める

互いのコンフリクトを避けるため、チームの Prefix を定めて管理することが推奨されています。

所有権を明確にするための名前空間CSS、イベント、ローカルストレージおよびCookie。

本当にクロスチームAPIを構築する必要がある場合は、できるだけシンプルに保つようにしてください。

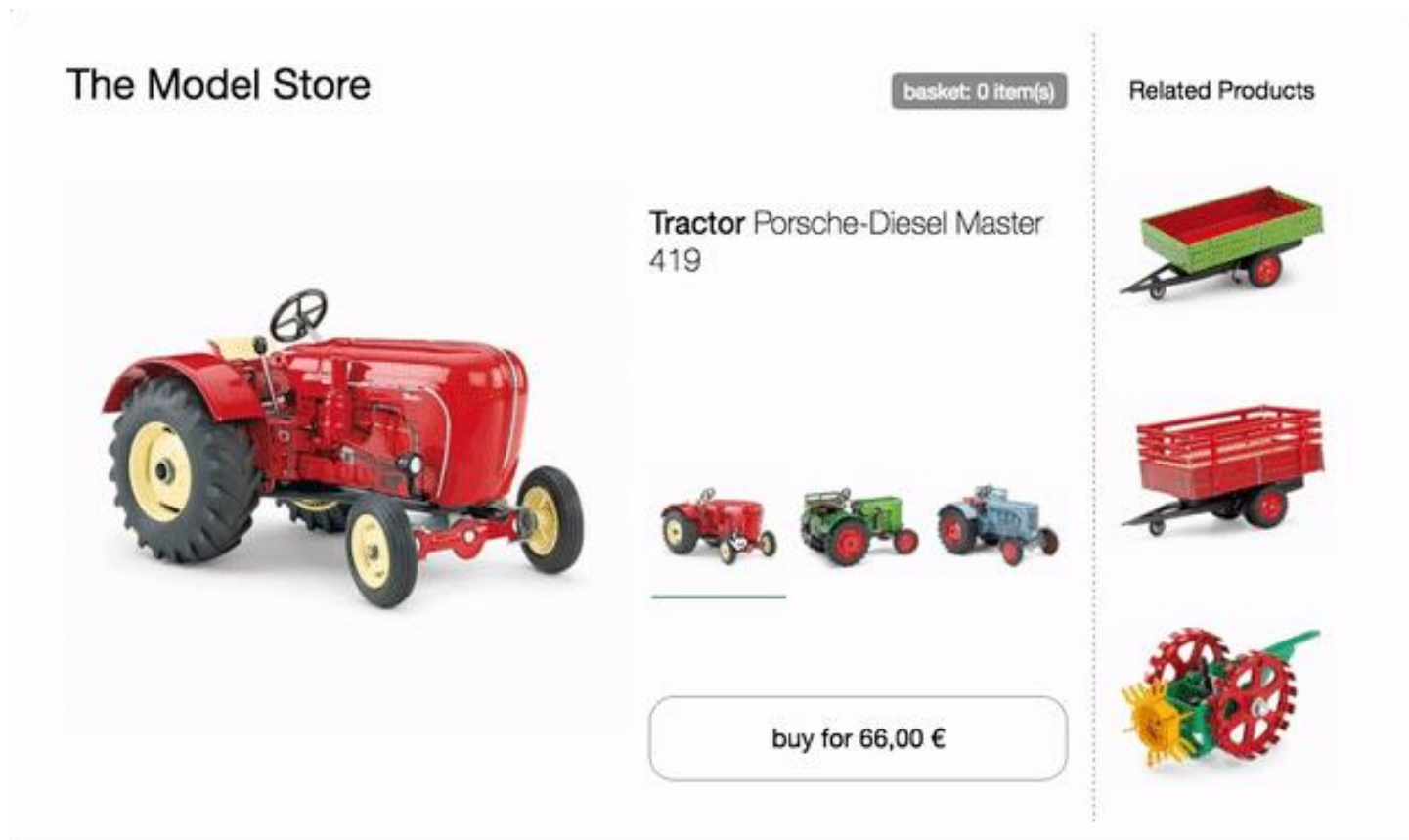
- **Build a Resilient Site** 回復力があるサイトを構築する。

JavaScriptが失敗したり、まだ実行されていない場合でも、この機能は役立つはずです。

※詳しい情報

<https://resilientwebdesign.com/>

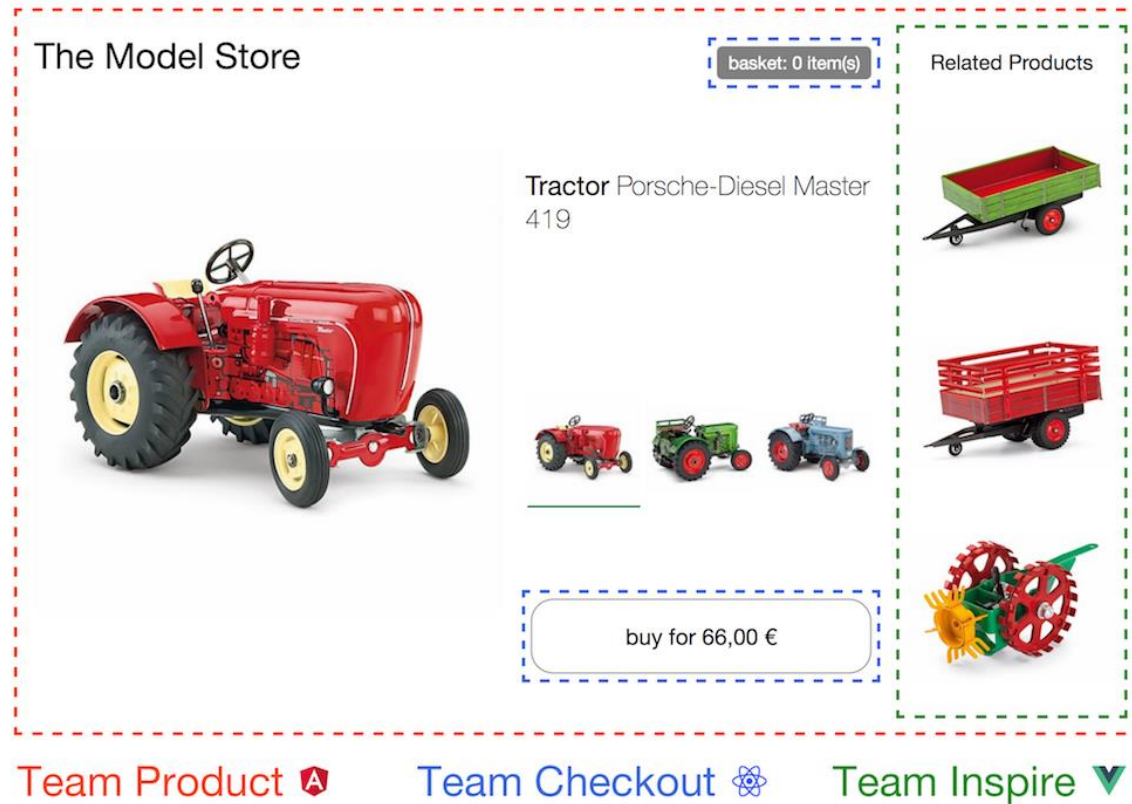
サンプル



トラクターストアの製品ページに下記の機能があります。

- ①3つ異なるトラクターモデルを切り替える。
 - ②製品イメージを変更すると、名前、価格、推奨事項が更新されます。
 - ③選択したモデルをバスケットに追加する。
- 購入ボタンを押すと、それに応じて更新される上部のミニバスケットもあります。

サンプル — Missionによりチームを分ける



ページは3つチームが所有する個別のコンポーネントに分割されています。

どの機能を含めるか、およびレイアウトのどこに配置するかを決定します。
製品名、画像、利用可能なバリエーションなど、チーム製品自体が提供できる情報が含まれています。
ただし、他のチームのフラグメント(カスタム要素)も含まれています。

購入プロセスに関するすべての責任を負います。

このページの製品の推奨事項を管理します。

Micro Frontendsの実装

現時点で Micro Frontends を実現する方法は2つ選択肢があります。

- iframe

iframe を使いたい人が今の所存在しない

- Web Components (Custom Elements)

React や Angular と Custom Elements でどこまでできるか、というのが話の焦点になってきます。

実装

- **Clientside Integration**
- **Serverside Rendering**
- **Custom Elements + Server Side Includes**

Clientside Integration

「Custom Element V1 Spec」とは

カスタム要素 v1: 再利用可能なウェブ コンポーネント

「Custom Element」を使用して、新しい HTML タグを作成したり、既存の HTML タグを拡張したり、他のデベロッパーが作成したコンポーネントを拡張したりすることができます。

Chrome、Safari、Operaでサポートされています。

参照URL:

<https://developers.google.com/web/fundamentals/web-components/customelements?hl=ja>

「Custom Element」例

```
<!DOCTYPE html>
<html lang="en">
  <head>
  </head>
  <body>
  </body>
</html>
```

[blue-buy] →
connectedCallback()を呼出す

```
class BlueBuy extends HTMLElement {
  connectedCallback() {
    this.innerHTML = `<button type="button">buy for 66,00 €</button>`;
  }

  disconnectedCallback() { ... }
}

window.customElements.define('blue-buy', BlueBuy);
```

Framework Compatibility

カスタム要素はWeb標準であるため、Angular、React、Preact、Vue、Hyperappなどのすべての主要なJavaScriptフレームワークをサポートしています。

詳細に入ると、いくつかのフレームワークにはまだいくつかの実装上の問題があるというコメントがあります。

Serverside Rendering

Serverside Rendering

「it's good to think about what happens to the site if the JavaScript fails or is blocked。」

JavaScriptが失敗したり、まだ実行されていない場合でも画面はWhiteにならない

「[Resilient Web Design](#)」

ServerSide Include

Server Side Includes (SSI) はWebサーバの機能の1つである。

HTMLの中にWebサーバ側で実行するコマンドを埋め込んでおき、
その実行結果をクライアントサーバに返す仕組みである。

ただしWebサーバ自体がSSIに対応またはサービス利用可になっている必要がある。

https://en.wikipedia.org/wiki/Server_Side_Includes

ServerSide

各チームには独自のExpress Serverがあり、カスタム要素のrender()メソッドにもURLからアクセスできます。

```
$ curl http://127.0.0.1:3000/blue-buy?sku=t_porsche  
<button type="button">buy for 66,00 €</button>
```

#includeコメントは、Webサーバーがページ全体をブラウザに送信する前に、/blue-buy?sku=t_porscheの応答に置き換えられます。

```
<blue-buy sku="t_porsche">  
  <!--#include virtual="/blue-buy?sku=t_porsche" -->  
</blue-buy>
```

Custom Elements + Server Side Includes

各チームには独自のexpress serverがあり、カスタム要素のrender()メソッドにもURLからアクセスできます。

```
server {  
    listen 3000;  
    ssi on;  
  
    location /blue {  
        proxy_pass http://team_blue;  
    }  
    location /green {  
        proxy_pass http://team_green;  
    }  
    location /red {  
        proxy_pass http://team_red;  
    }  
    location / {  
        proxy_pass http://team_red;  
    }  
}
```

```
upstream team_blue {  
    server team_blue:3001;  
}  
upstream team_green {  
    server team_green:3002;  
}  
upstream team_red {  
    server team_red:3003;  
}
```

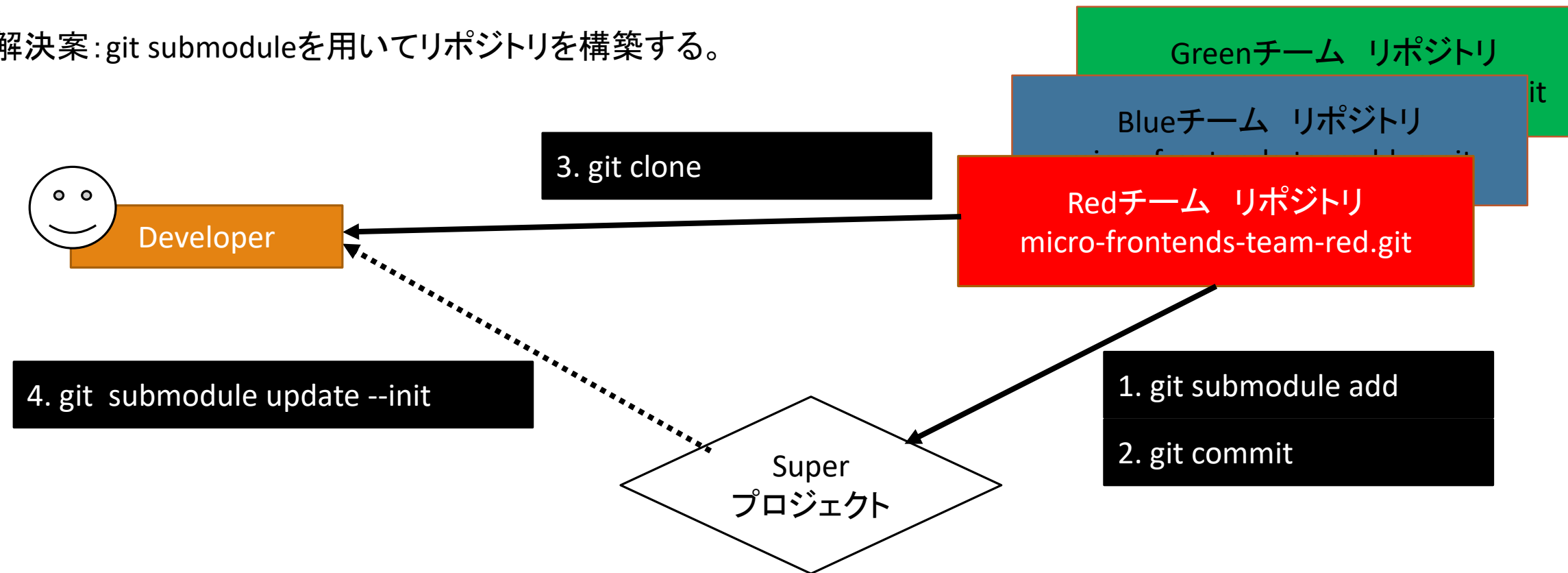
サーバーがnginx であれば、
configは左のようです。

/ blueで始まるすべてのURL
が team_blue:3001 のアプ
リケーションにルーティング
される

ソース管理

各チームには独自のExpress Serverがあり、独自のリポジトリも考えられます。

解決案: git submoduleを用いてリポジトリを構築する。



ソース管理

各チームのリポジトリを作成する。

例:

■Superプロジェクト

`https://{giturl}/micro-frontends.git`

■Subモジュール

`https://{giturl}/micro-frontends-team-red.git`

`https://{giturl}/micro-frontends-team-blue.git`

`https://{giturl}/micro-frontends-team-green.git`

関連コマンド

>git clone https://{giturl}/micro-frontends.git

>cd micro-frontends

>git submodule add https://{giturl}/micro-frontends-team-blue.git team-blue

>git submodule add https://{giturl}/micro-frontends-team-red.git team-red

>git submodule add https://{giturl}/micro-frontends-team-green.git team-green

>git add -A

>git commit -m "Add all modules"

>git push

Superプロジェクトの様子

サブモジュール

		🕒 2 commits
📁	team-blue @ 356a308	
📁	team-green @ 08e1dbf	
📁	team-red @ b561b84	
📄	.gitmodules	
📄	README.md	

開発者の作業手順

>git clone https://{giturl}/ micro-frontends-team-blue.git team-blue

>git submodule update -init

※ まず git submodule init でローカルの設定ファイルを初期化し、次に git submodule update でプロジェクトからのデータを取得し、親プロジェクトで指定されている適切なコミットをチェックアウトします。

※詳しいことは下記のURLを参照してください。

<https://git-scm.com/book/ja/v2/Git-%E3%81%AE%E3%81%95%E3%81%BE%E3%81%96%E3%81%BE%E3%81%AA%E3%83%84%E3%83%BC%E3%83%AB-%E3%82%B5%E3%83%96%E3%83%A2%E3%82%B8%E3%83%A5%E3%83%BC%E3%83%AB>

React とは

React.jsはM-V-CのViewの一部の機能を提供するライブラリです。

- ・UIのパーツ(構成部品)を作るためのライブラリです。

なぜReact.jsは注目されるのか

- Facebookの製品
- フロントエンドの考え方を根本的に覆す
- Virtual DOM

リアルなDOMは遅い、仮想DOM(React.js)は速い

<http://steps.dodgson.org/b/2014/12/11/why-is-real-dom-slow/>

- コンポーネント指向
- リアクティブプログラミング
- 関数型言語(プログラミング)

React どの企業を利用しているのか

- Facebook
- Instagram
- YahooやAirbnb
- GitHubのATOM (<https://atom.io/>)

HelloWorld の実装

■実装環境

OS: Microsoft Windows 10 Enterprise

バージョン: 10.0.17763

■準備

nodeJsをインストールする。

(インストール手順を略)

```
>node --version
```

v10.16.3

```
>npm --version
```

6.9.0

■必要なライブラリをインストールする。

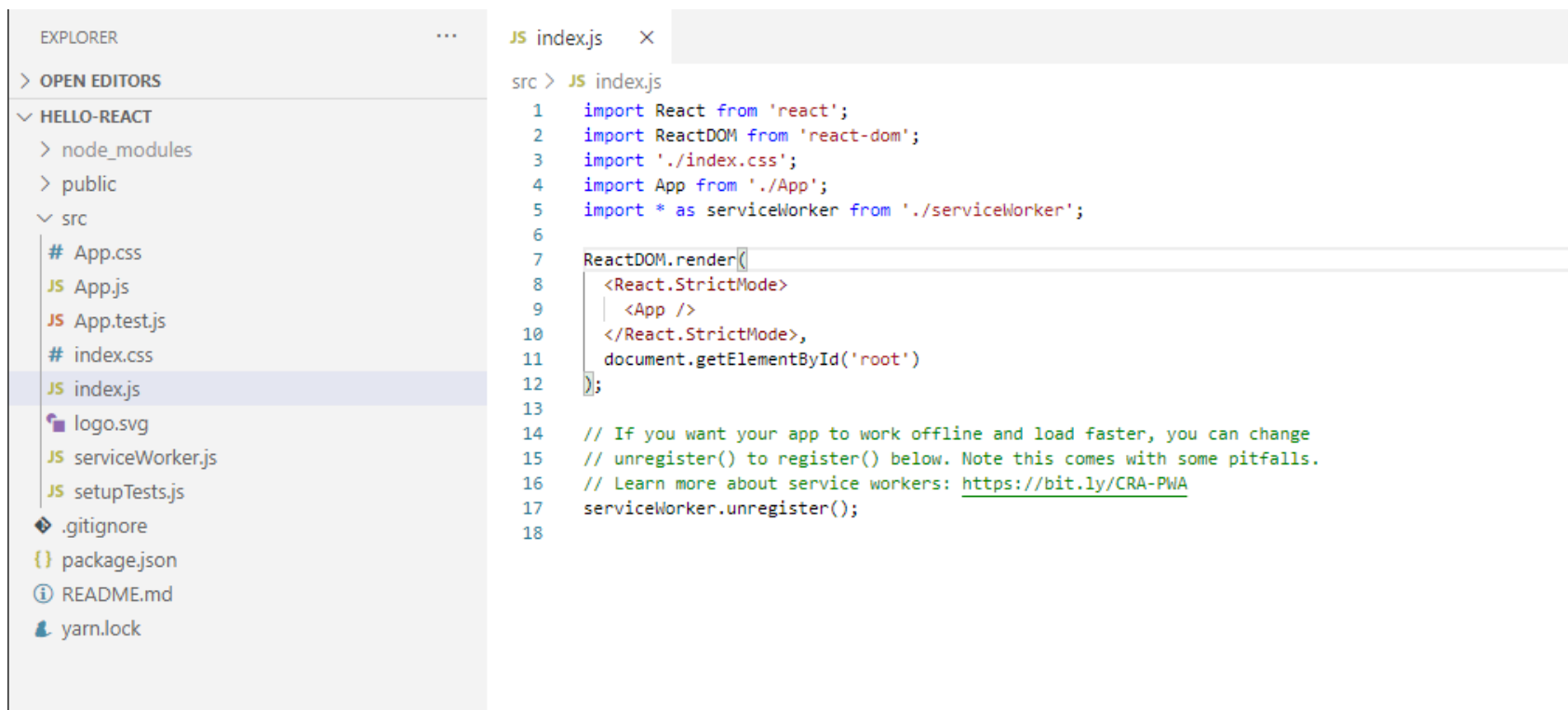
```
>npm install -g create-react-app
```

■プロジェクトを作成する。

```
>create-react-app hello-react
```

HelloWorld の実装

生成したソースの様子



The image shows a screenshot of the Visual Studio Code editor. On the left, the 'EXPLORER' sidebar displays the project structure for 'HELLO-REACT'. The 'src' directory is expanded, showing files like 'App.css', 'App.js', 'App.test.js', 'index.css', 'index.js' (which is selected), 'logo.svg', 'serviceWorker.js', and 'setupTests.js'. On the right, the 'index.js' file is open in the editor. The code is as follows:

```
src > JS index.js
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import './index.css';
4  import App from './App';
5  import * as serviceWorker from './serviceWorker';
6
7  ReactDOM.render(
8    <React.StrictMode>
9      <App />
10    </React.StrictMode>,
11    document.getElementById('root')
12  );
13
14  // If you want your app to work offline and load faster, you can change
15  // unregister() to register() below. Note this comes with some pitfalls.
16  // Learn more about service workers: https://bit.ly/CRA-PWA
17  serviceWorker.unregister();
18
```

HelloWorld の実装

サーバーを起動する。

> **npm start**

> hello-react@0.1.0 start c:¥project¥study¥hello-react

> react-scripts start

i 「wds」: Project is running at http://192.168.21.129/

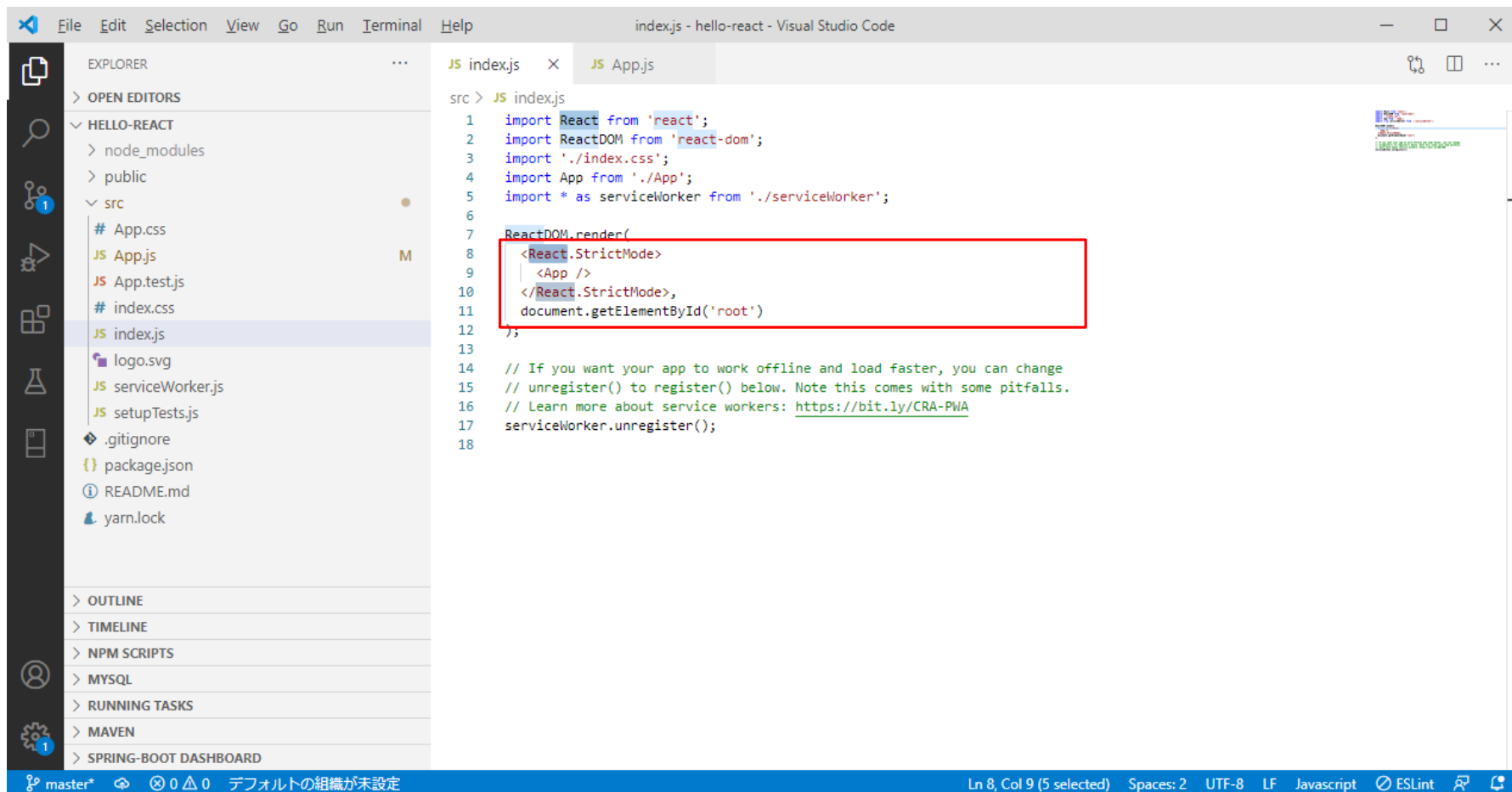
i 「wds」: webpack output is served from

i 「wds」: Content not from webpack is served from c:¥project¥study¥hello-react¥public

i 「wds」: 404s will fallback to /

Starting the development server...

HelloWorld の実装



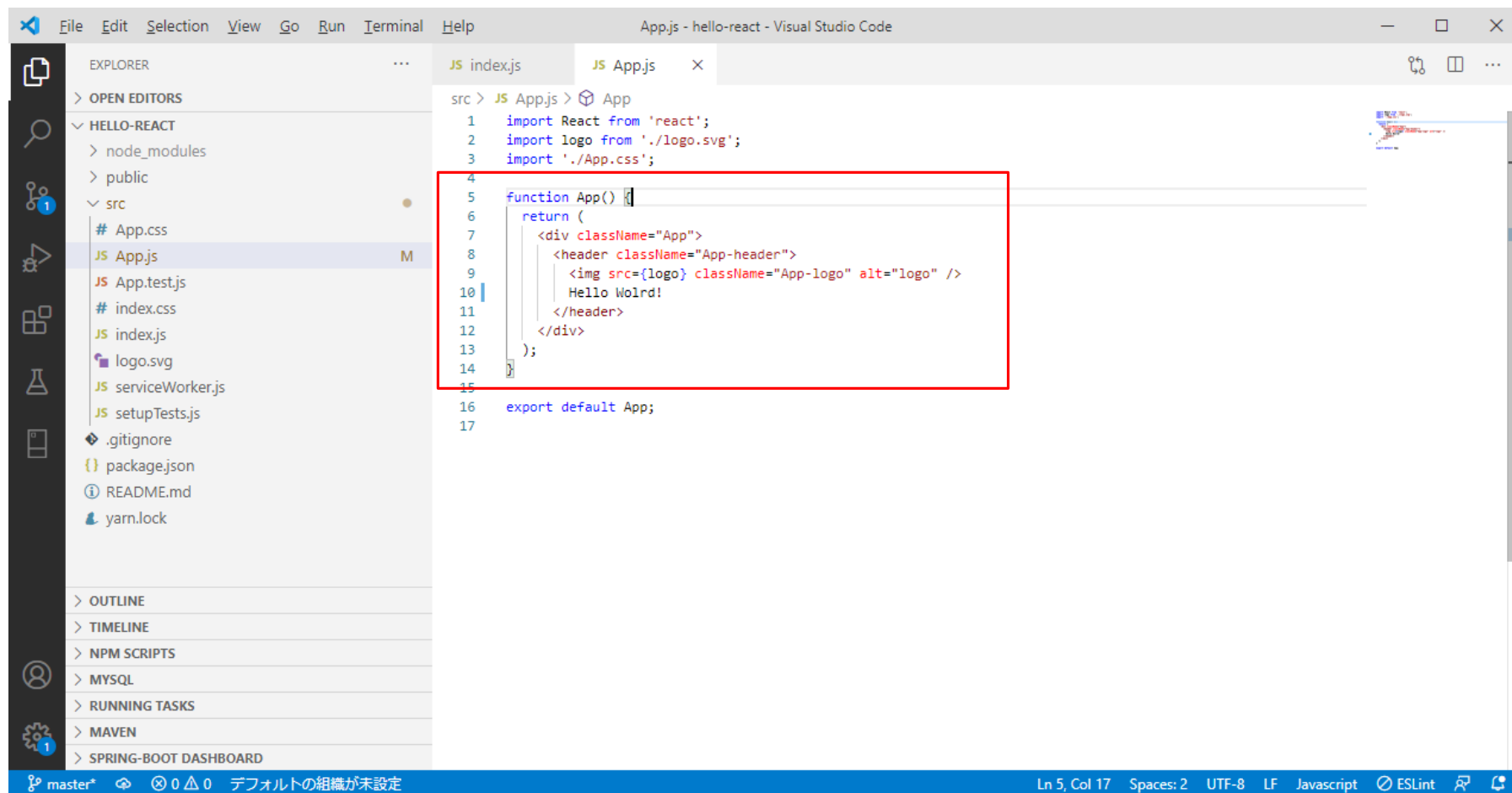
The screenshot shows the Visual Studio Code interface with a project named 'HELLO-REACT'. The Explorer sidebar on the left shows the file structure, including 'src' with files like 'App.css', 'App.js', 'App.test.js', 'index.css', 'index.js', 'logo.svg', 'serviceWorker.js', and 'setupTests.js'. The main editor displays the 'index.js' file, which contains the following code:

```
src > JS index.js
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import './index.css';
4  import App from './App';
5  import * as serviceWorker from './serviceWorker';
6
7  ReactDOM.render(
8    <React.StrictMode>
9      <App />
10    </React.StrictMode>,
11    document.getElementById('root')
12  );
13
14  // If you want your app to work offline and load faster, you can change
15  // unregister() to register() below. Note this comes with some pitfalls.
16  // Learn more about service workers: https://bit.ly/CRA-PWA
17  serviceWorker.unregister();
18
```

A red rectangle highlights the `ReactDOM.render` call and its arguments, specifically the `<React.StrictMode>` and `<App />` components, and the `document.getElementById('root')` target.

The status bar at the bottom indicates the current file is 'index.js' at line 8, column 9 (5 selected), with 2 spaces, UTF-8 encoding, LF line endings, and the language set to Javascript. It also shows ESLint and other tool icons.

HelloWorld の実装



```
File Edit Selection View Go Run Terminal Help App.js - hello-react - Visual Studio Code

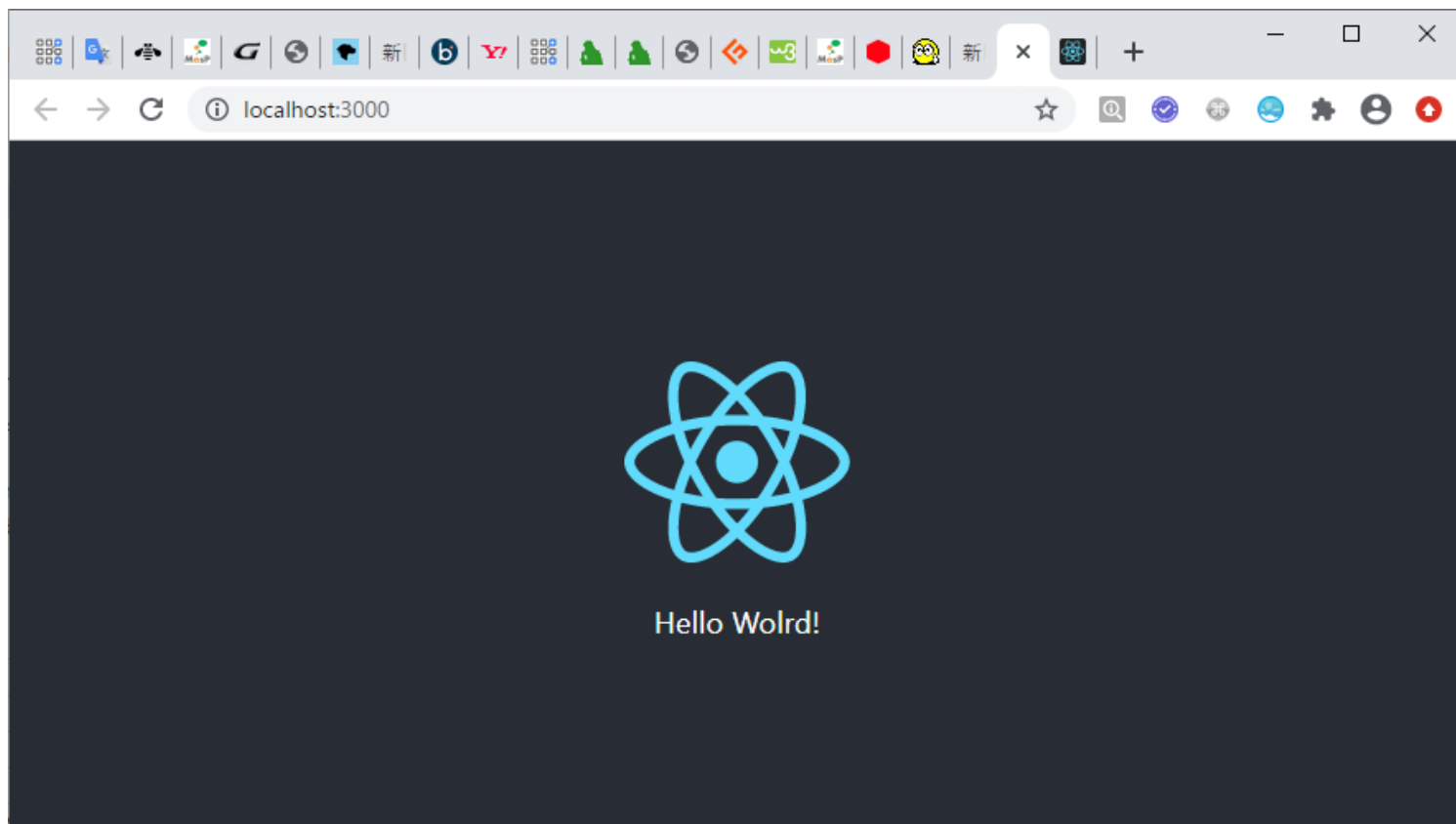
EXPLORER
> OPEN EDITORS
HELLO-REACT
  > node_modules
  > public
  > src
    # App.css
    JS App.js
    JS App.test.js
    # index.css
    JS index.js
    logo.svg
    JS serviceWorker.js
    JS setupTests.js
    .gitignore
    {} package.json
    README.md
    yarn.lock

OUTLINE
TIMELINE
NPM SCRIPTS
MYSQL
RUNNING TASKS
MAVEN
SPRING-BOOT DASHBOARD

src > JS App.js > App
1 import React from 'react';
2 import logo from './logo.svg';
3 import './App.css';
4
5 function App() {
6   return (
7     <div className="App">
8       <header className="App-header">
9         <img src={logo} className="App-logo" alt="logo" />
10        Hello World!
11      </header>
12    </div>
13  );
14
15
16 export default App;
17
```

Ln 5, Col 17 Spaces: 2 UTF-8 LF Javascript ESLint

HelloWorld の実装



初期化サンプルを改造

EXPLORER

OPEN EDITORS 3 UNSAVED

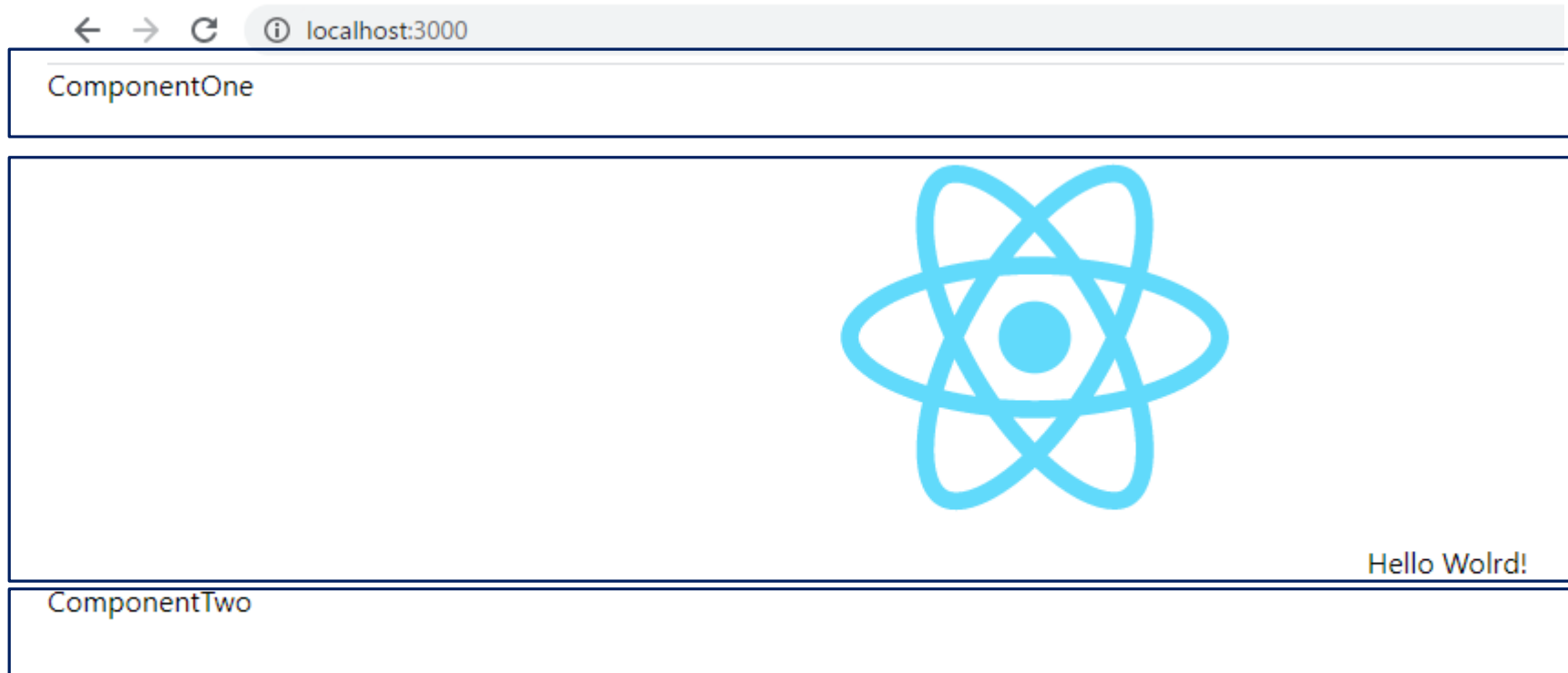
HELLO-REACT

- manifest.json
- robots.txt
- src
 - App.css
 - App.js M
 - App.test.js
 - ComponentOne.js U
 - ComponentTwo.js U
 - index.css
 - index.js M
 - logo.svg
 - serviceWorker.js
 - setupTests.js
- .gitignore
- package.json
- README.md

src > JS index.js

```
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import './index.css';
4 import App from './App';
5 import ComponentOne from './ComponentOne';
6 import ComponentTwo from './ComponentTwo';
7 import * as serviceWorker from './serviceWorker';
8
9 ReactDOM.render(
10   <React.StrictMode>
11     <ComponentOne />
12     <App />
13     <ComponentTwo />
14   </React.StrictMode>,
15   document.getElementById('root')
16 );
17
18 // If you want your app to work offline and load faster, you can change
19 // unregister() to register() below. Note this comes with some pitfalls.
20 // Learn more about service workers: https://bit.ly/CRA-PWA
21 serviceWorker.unregister();
22
```

初期化サンプルを改造



参照資料

React.jsはFacebookが作っていますので、
Facebookの画面（UI）が参考になると思います。

GitHubのATOM <https://atom.io/>

ElementalUI <http://elemental-ui.com/>

Micro-frontends <https://micro-frontends.org/>

MicroFrontends調査報告

ご清聴ありがとうございました。