STEP3-1

Starter
宿題パッケージ

Ver3.0

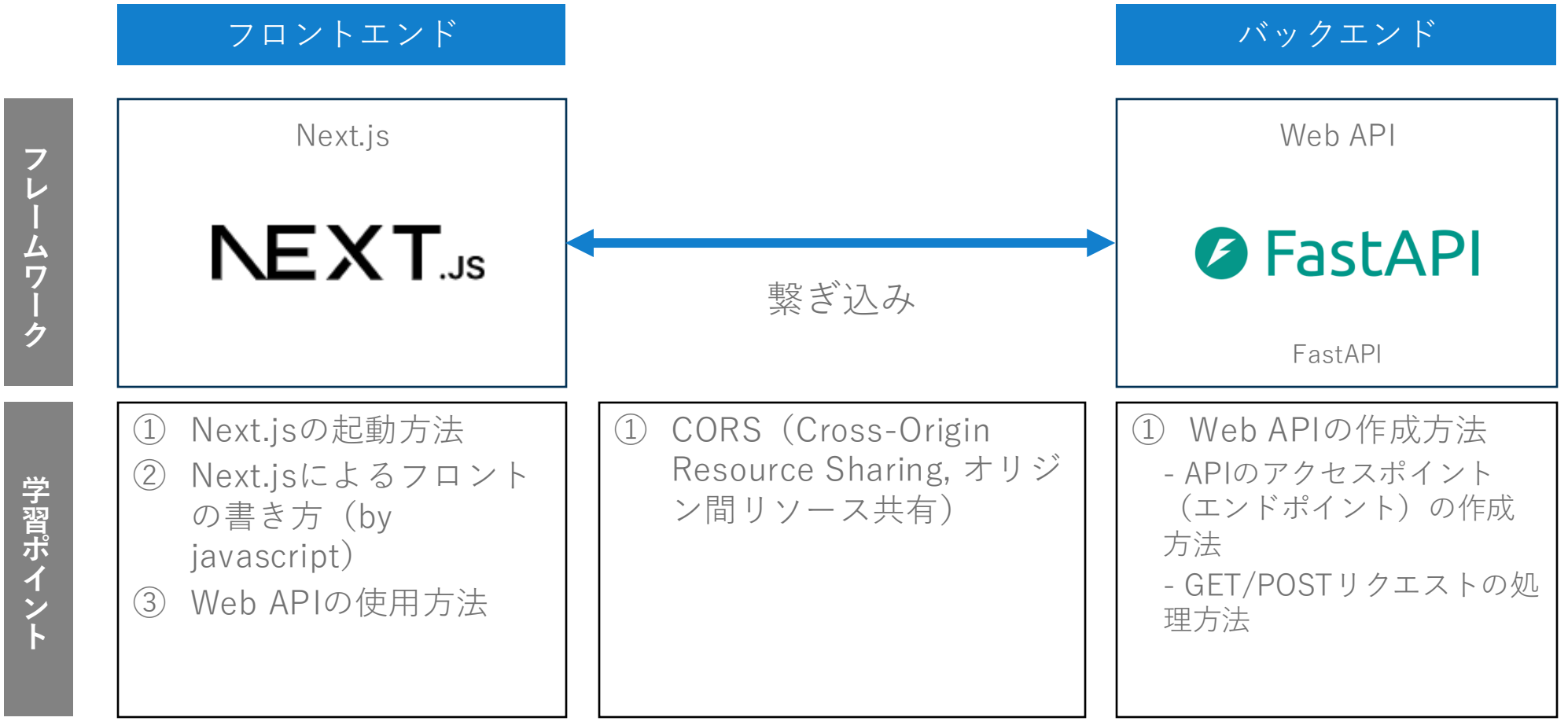
宿題パッケージ：for Starterの目指す姿

- Next.js,FastAPIってなに…？
- フロントエンドとバックエンドってどうやってつなぐの？

- Next.js,FastAPIのコードがなんとなく読める！
- フロントエンド↔バックエンド間のリクエスト・レスポンスの関係がわかるようになった！

宿題パッケージ：for Starter

Next.jsとFastAPIのそれぞれの利用方法とつなぎ込みをシンプルに実装したサンプルです。
Starterではシンプルな繋ぎ込みを実践します。



本資料の目的

マイクロサービスアーキテクチャの概念を理解し、Next.js（フロントエンド）、FastAPI（バックエンド）というモダンなフレームワークの構築を行います。for Practicalにむけて、シンプルで簡単な実装から始めます。

アジェンダ

Input

マイクロサービスアーキテクチャ
└モノリシックアーキテクチャとの対比
└技術要素の解説
└Next.js
└FastAPi

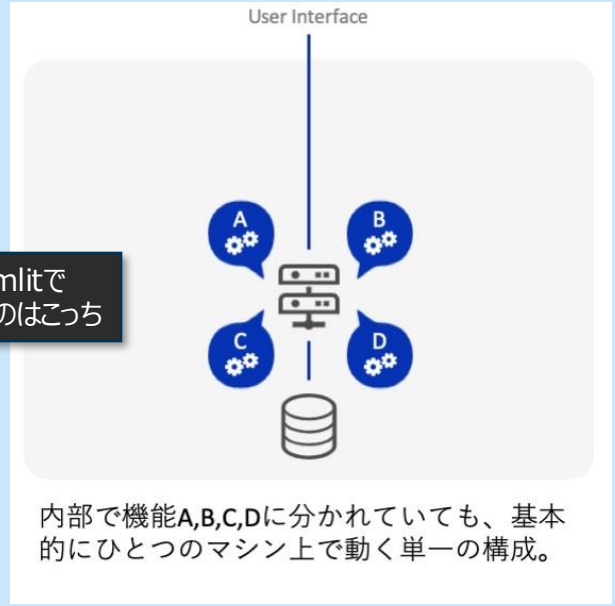
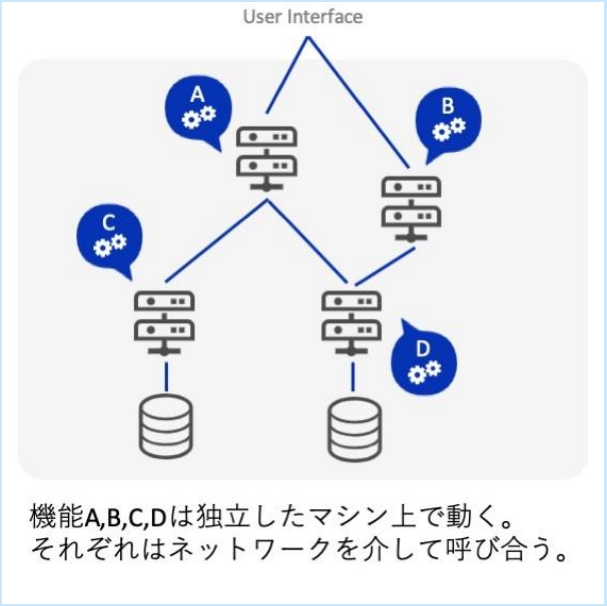
サンプルアプリ
└サンプルアプリの挙動
└アーキテクチャ
└フロントエンド/バックエンドの構成

Output

└演習
└ローカルビルド（チュートリアル）
└ Starterパッケージのゴール（宿題）

モノリシックアーキテクチャとマイクロサービスアーキテクチャの対比

フロントエンドもバックエンドも一体化した**モノリシックアーキテクチャ**はデプロイが単純になるがスケールが難しくなる傾向があります。各サービスを独立させる**マイクロサービスアーキテクチャ**は全体の複雑性が上がるものの柔軟な開発が可能となります。

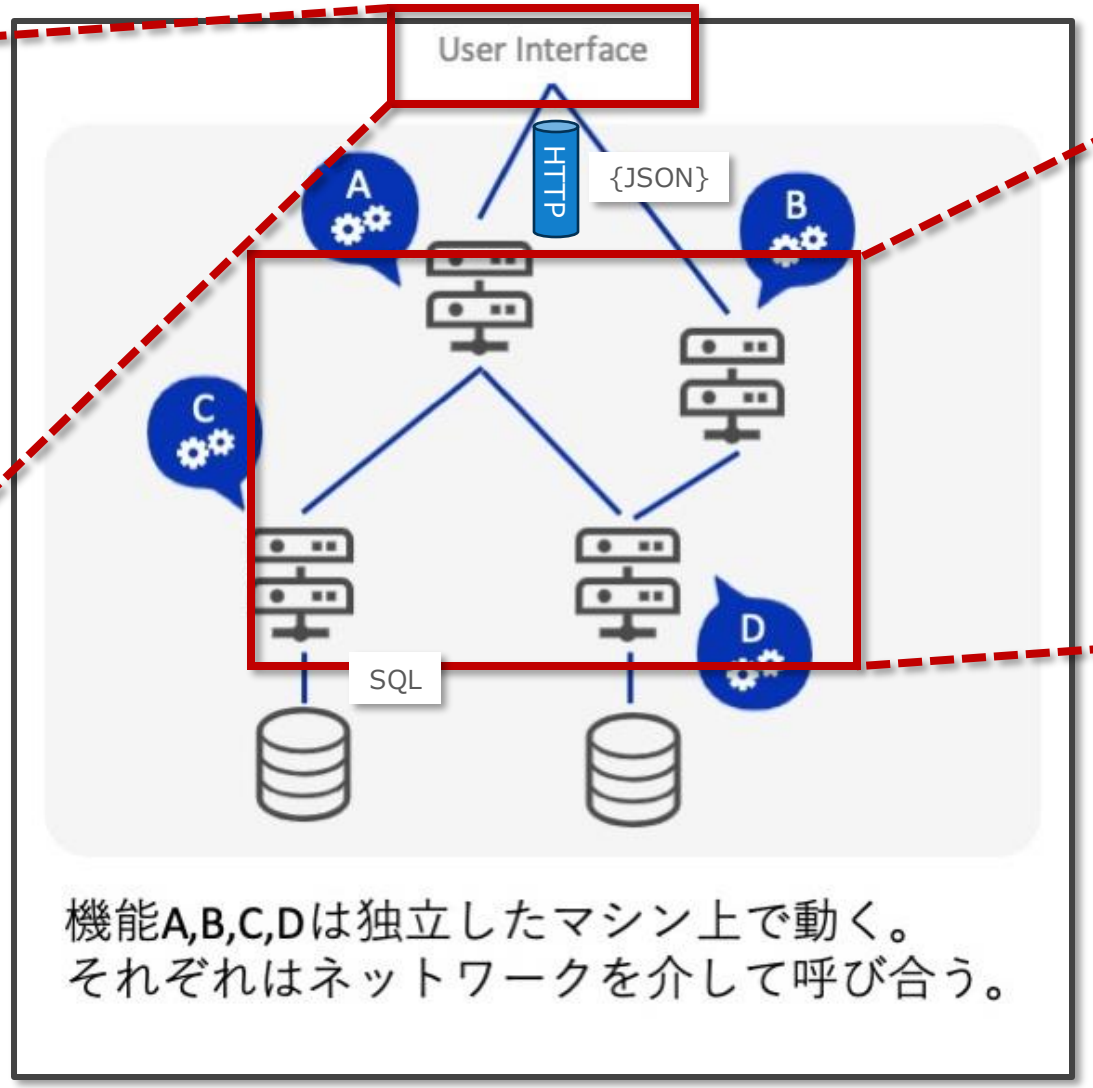
		モノリシックアーキテクチャ	マイクロサービスアーキテクチャ
構成図		 Streamlitで やっていたのはこっち 内部で機能A,B,C,Dに分かれていても、基本的にひとつのマシン上で動く単一の構成。	 機能A,B,C,Dは独立したマシン上で動く。 それぞれはネットワークを介して呼び合う。
特徴	長所	• デプロイが単純なため、管理が容易 • 依存関係が少ないため、開発とデバッグがしやすい	• 各サービスが独立しているため、必要な部分だけを効率的にスケール可能 • 異なる技術を柔軟に採用できるため、各サービスに最適な技術スタックを選択できる
	短所	• アプリケーションが肥大化すると、新機能の追加や更新が難しくなる	• マイクロサービス間の多数の通信により、ネットワーク遅延やデータ整合性の確保が課題になる可能性がある

技術要素

ユーザー接点部分をフロントエンド、それ以外の裏側をバックエンド（サーバー・DB）と分類することができ、それぞれにおいて様々な技術スタックの選択肢が存在する。（**太字**は今回扱うもの）

要素名称
代表技術

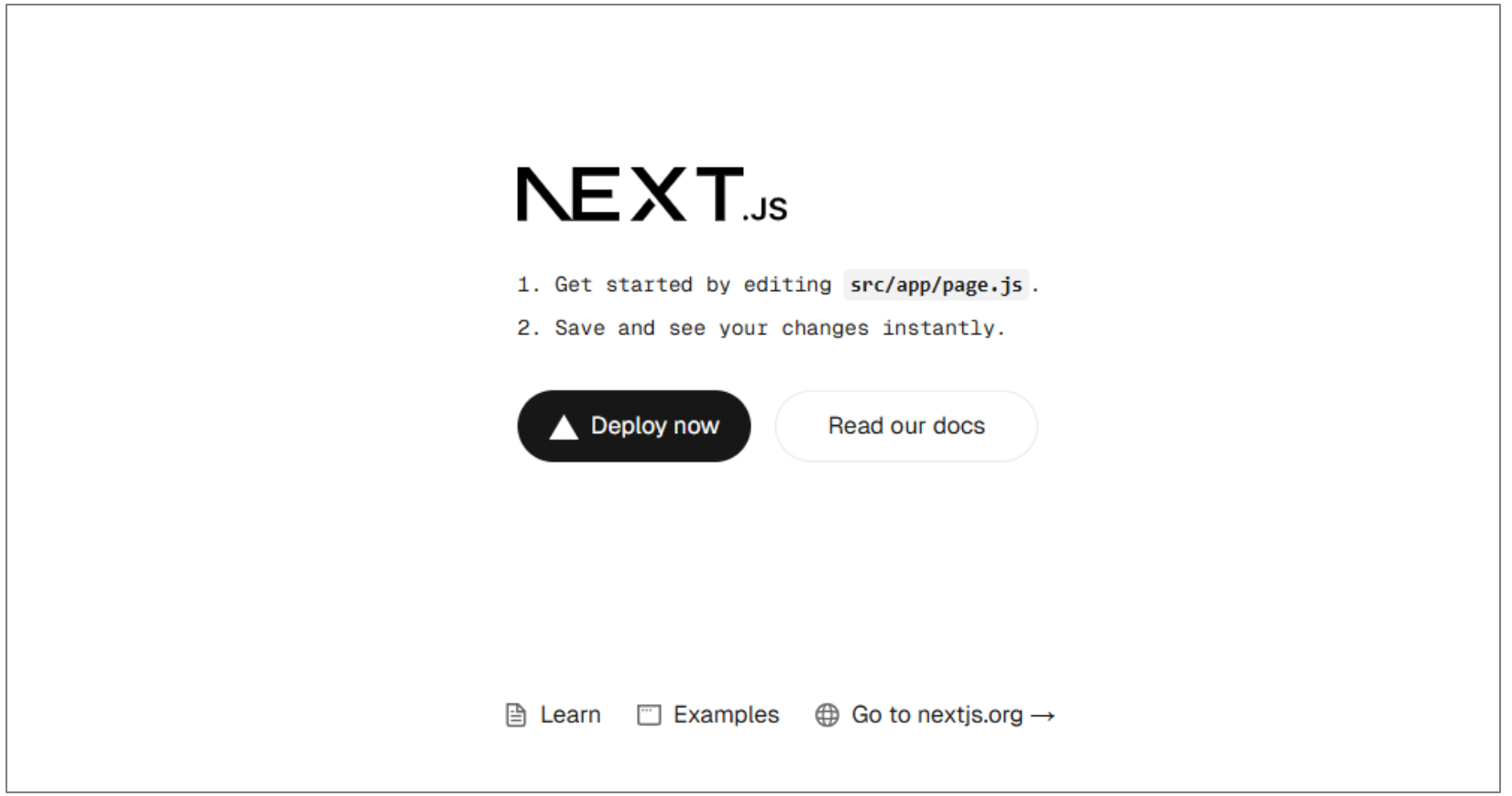
フロントエンド
- **Next.js**
- React
- Vue.js
- Angular



バックエンド（サーバー）
- **FastAPI**
- Flask
- Express.js
- Django

技術要素：Next.js

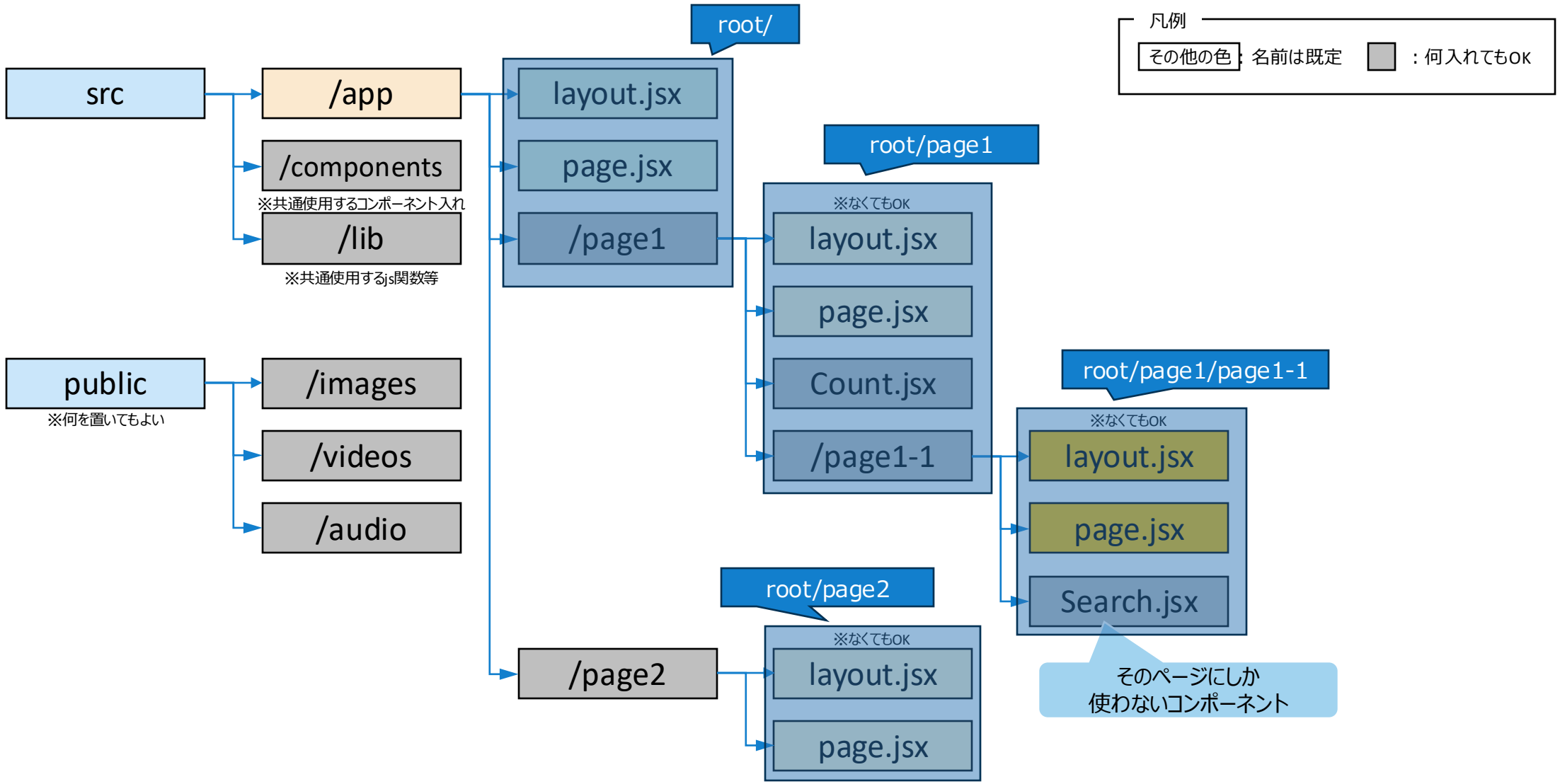
ReactベースのJavaScriptフレームワークで、サーバーサイドレンダリング、静的サイト生成、APIルートなどの機能を提供し、高速なWebアプリケーション開発を可能にする。開発者体験とパフォーマンスの最適化に焦点を当てている。



※今回はフロントエンドフレームワークとしてNext.jsを利用します。機能としてはバックエンドも包含していますが今回は利用しません。

Next.jsのフォルダ構成とAppRouter

Next.js 13以降はAppRouterというファイルベースのルーティングシステムを採用しています。プロジェクトの”app”フォルダ配下のファイル構造がそのままURLのRouting構造に反映され、layoutやpage等の特別な名前を持つファイルによってページを記述をします。

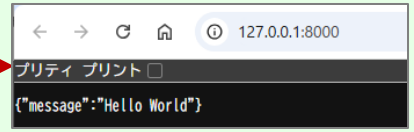


技術要素：FastAPI

Pythonで高性能なAPIを開発するためのWebフレームワークです。高性能かつ非同期処理に対応し、簡潔な構文や自動ドキュメント生成、強力な型システムを備えています。拡張性にも優れ、効率的で安全なAPI開発が可能です。

FastAPIのバックエンド側

```python
@app.get("/")
def hello():
    return {"message": "Hello World"}
```



@app.get()デコレーターを使って「GETリクエスト」を処理するエンドポイント
/はルートパスなので　http://localhost:3000/　アクセスで実行される
hello関数の戻り値としてハローメッセージを返す処理を行います

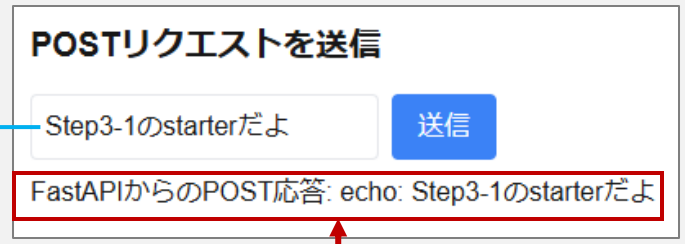
```python
@app.get("/api/hello")
def hello_world():
    return {"message": "Hello World by FastAPI"}
```

```python
@app.post("/api/echo")
def echo(message: EchoMessage):
    print("echo")
    if not message:
        raise HTTPException(status_code=400, detail="Invalid JSON")

    echo_message = message.message if message.message else "No message provided"
    return {"message": f"echo: {echo_message}"}
```

@app.post()デコレーターを使って「POSTリクエスト」を処理するエンドポイント
/api/echoにリクエストを送るとecho関数が実行されて受け取ったメッセージをそのまま返します
ここではエラーハンドリングも行っています

Next.jsで実装したフロントエンド側



本資料の目的　Output編（前半）

マイクロサービスアーキテクチャの概念を理解し、Next.js（フロントエンド）、FastAPI（バックエンド）というモダンなフレームワークの構築を行います。for Practicalにむけて、シンプルで簡単な実装から始めます。

アジェンダ

Input

マイクロサービスアーキテクチャ
└モノリシックアーキテクチャとの対比
└技術要素の解説
　└Next.js
　└FastAPi

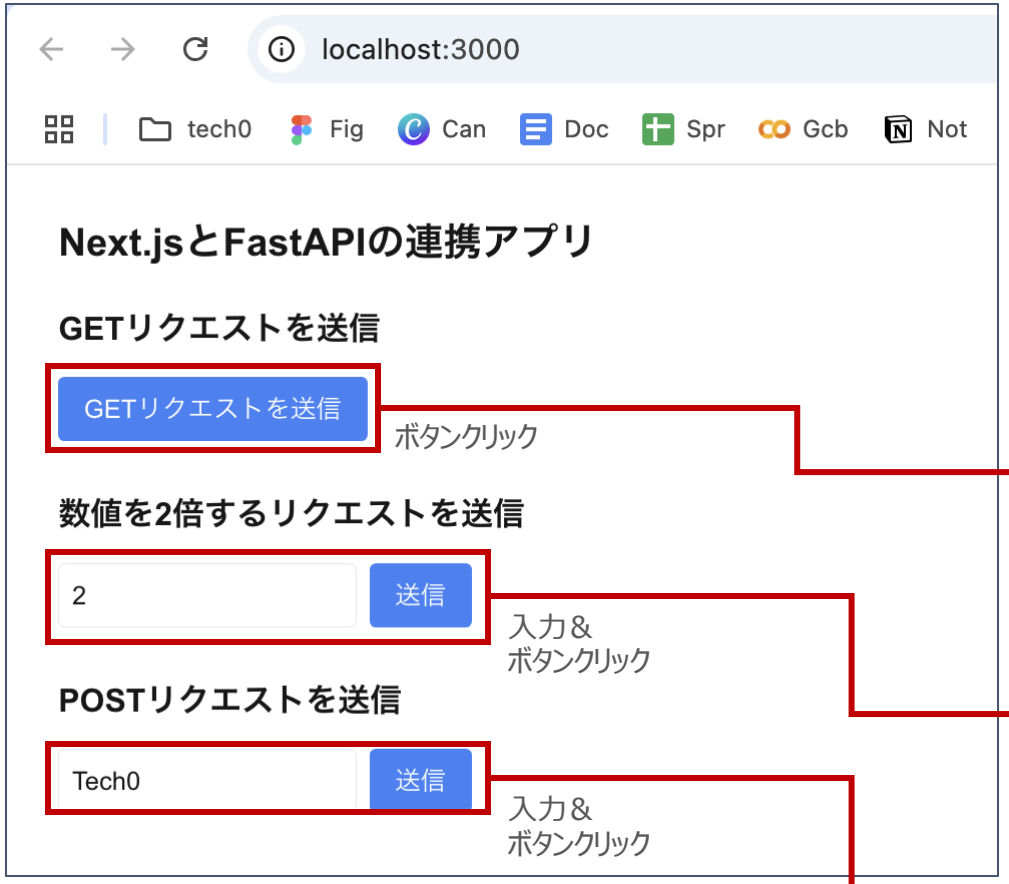
サンプルアプリ
└サンプルアプリの挙動
└アーキテクチャ
└フロントエンド/バックエンドの構成

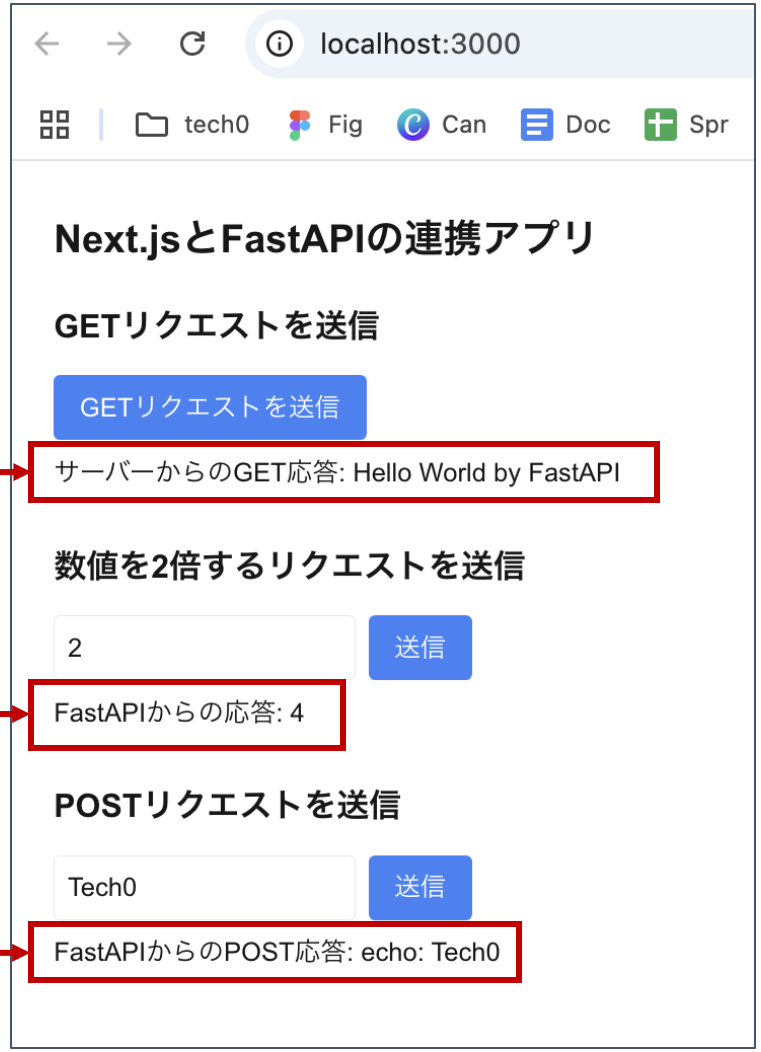
Output

└演習
　└ローカルビルド（チュートリアル）
　└Starterパッケージのゴール（宿題）

サンプルアプリの解説-挙動

Next.jsからFastAPIで設定したAPIに対して、GET/POSTリクエストを送信し、レスポンスを表示させる





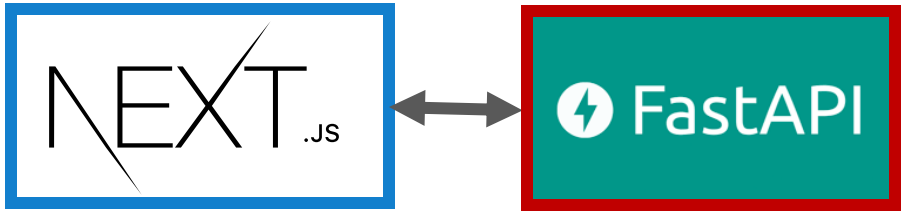
各種ボタン操作により、表示が変更される。

サンプルアプリの解説-アーキテクチャ

サンプルアプリは単一のリポジトリにフロントエンド（Next.js）、バックエンド（FastAPI）のプロジェクトを含む構成になっている。

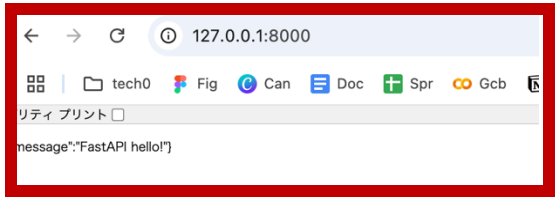
リポジトリ構成

アーキテクチャ



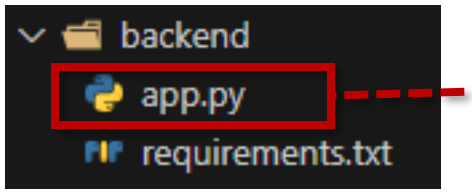
フロントエンド　バックエンド

イメージ



サンプルアプリの解説-FastAPI

APIサーバーとしての実装はapp.pyに記述されている。主に、root/{エンドポイント}にリクエストがあった際の挙動が記載されている。




```python
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel

app = FastAPI()

# CORSの設定 フロントエンドからの接続を許可する部分
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://localhost:3000"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"]
)

# データのスキーマを定義するためのクラス
class EchoMessage(BaseModel):
    message: str | None = None

@app.get("/")
def hello():
    return {"message": "FastAPI hello!"}

@app.get("/api/hello")
def hello_world():
    return {"message": "Hello World by FastAPI"}

@app.get("/api/multiply/{id}")
def multiply(id: int):
    print("multiply")
    doubled_value = id * 2
    return {"doubled_value": doubled_value}

@app.post("/api/echo")
def echo(message: EchoMessage):
    print("echo")
    if not message:
        raise HTTPException(status_code=400, detail="Invalid JSON")

    echo_message = message.message if message.message else "No message provided"
    return {"message": f"echo: {echo_message}"}
```

CORSの設定で、特定の通信先にアクセス許可を与えている。

”/root”にGETリクエストがあると、”{‘message’: ‘FastAPI hello!}”を返す

”/api/hello”にGETリクエストがあると、”{"message": "Hello World by FastAPI"}”を返す

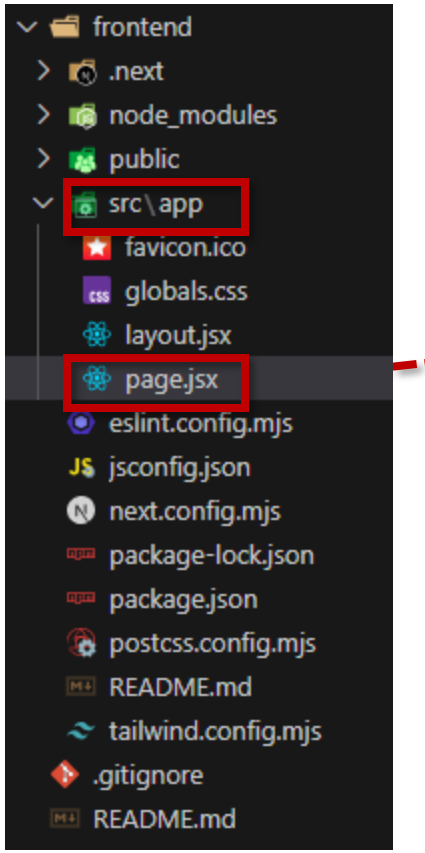
”/api/multiply/id”にGETリクエストがあると、渡されたidの２倍の値を返す

”/api/echo”にPOSTリクエストがあると、postされたテキストに”echo”を付けて返す

データの受け渡しは**json形式**で実施

サンプルアプリの解説-Next.js

src/app/page.jsxにて/rootページのコンポーネントが記述されている。



技術要素：page.jsxの見方

Next.jsのpage.jsxは定数や関数の処理部分と表示に関する部分で構成されている。

```
'use client';
import { useState } from 'react';

export default function Home() {
  // useStateを使った値（状態）管理
  const [getMessage, setGetMessage] = useState('');
  const [multiplyNumber, setMultiplyNumber] = useState('');
  const [multiplyResult, setMultiplyResult] = useState('');
  const [postMessage, setPostMessage] = useState('');
  const [postResult, setPostResult] = useState('');

  // FastAPIのエンドポイント設定
  const handleGetRequest = async () => {
    try {
      const response = await fetch('http://localhost:8000/api/hello');
      const data = await response.json();
      setGetMessage(data.message);
    } catch (error) {
      console.error('Error:', error);
    }
  };

  const handleMultiplyRequest = async () => {
    try {
      const response = await fetch(`http://localhost:8000/api/multiply/${multiplyNumber}`);
      const data = await response.json();
      setMultiplyResult(data.doubled_value.toString());
    } catch (error) {
      console.error('Error:', error);
    }
  };

  const handlePostRequest = async () => {
    try {
      const response = await fetch('http://localhost:8000/api/echo', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({ message: postMessage }),
      });
      const data = await response.json();
      setPostResult(data.message);
    } catch (error) {
      console.error('Error:', error);
    }
  };

  // ユーザーインターフェースの構築
  return (
    <div className="p-8">
      <h1 className="text-2xl font-bold mb-6">Next.jsとFastAPIの連携アプリ</h1>
      <div className="space-y-8">
        {/* GETリクエスト */}
        <section>
          <h2 className="text-xl font-bold mb-4">GETリクエストを送信</h2>
          <button
            onClick={handleGetRequest}
            className="bg-blue-500 hover:bg-blue-600 text-white px-4 py-2 rounded"
          >
            GETリクエストを送信
          </button>
          {getMessage && (
            <p className="mt-2">サーバーからのGET応答: {getMessage}</p>
          )}
        </section>

        {/* ID指定のGET */}
        <section>…
        </section>

        {/* POSTリクエスト */}
        <section>…
        </section>
      </div>
    </div>
  );
}
```

○定数と関数の処理

■定数：usestateを利用した状態管理
const [getMessage, setGetMessage] = useState(‘’);

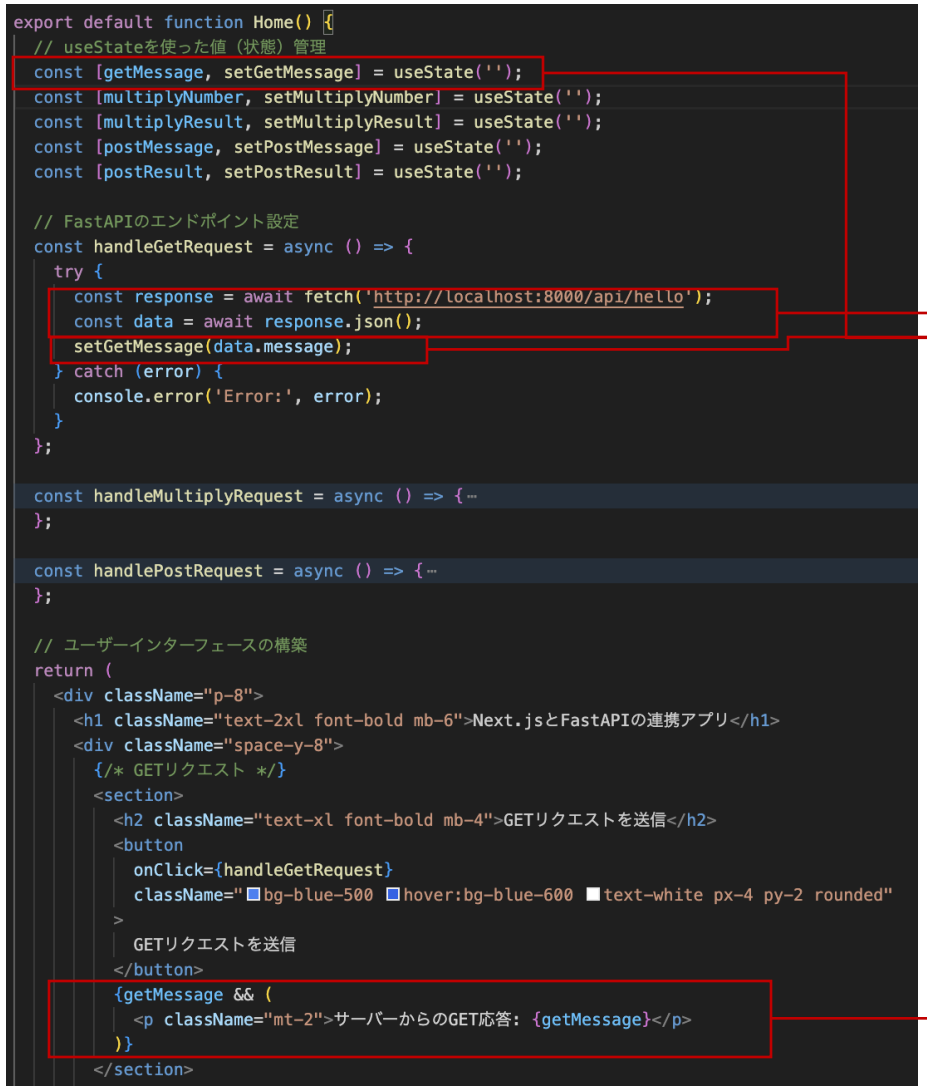
■関数：ボタン操作時の挙動を定義
（以下でボタン操作時の関数として定義されている）
const handleGetRequest = async () => {
　const res = await ・・・・

○画面表示
・Htmlと同様の書き方（jsxという記述方法）
・以下の**{handleGetRequest}**と**{getResponse}**は上記の定数と関数に対応している

<button onClick=**{handleGetRequest}**>GETリクエストを送信
</button>
　{**getResponse** && サーバーからのGET応答:
{getResponse} }

サンプルアプリの解説：fetchとuseState

最新のNext.jsはAppRouterというファイルベースのルーティングシステムを採用しており、プロジェクトの”app”フォルダ配下のファイル構造がそのままURLのRouting構造に反映され、layoutやpage等の特別な名前を持つファイルによってページ記述をする。



```tsx
export default function Home() {
  // useStateを使った値（状態）管理
  const [getMessage, setGetMessage] = useState('');
  const [multiplyNumber, setMultiplyNumber] = useState('');
  const [multiplyResult, setMultiplyResult] = useState('');
  const [postMessage, setPostMessage] = useState('');
  const [postResult, setPostResult] = useState('');

  // FastAPIのエンドポイント設定
  const handleGetRequest = async () => {
    try {
      const response = await fetch('http://localhost:8000/api/hello');
      const data = await response.json();
      setGetMessage(data.message);
    } catch (error) {
      console.error('Error:', error);
    }
  };

  const handleMultiplyRequest = async () => {…
  };

  const handlePostRequest = async () => {…
  };

  // ユーザーインターフェースの構築
  return (
    <div className="p-8">
      <h1 className="text-2xl font-bold mb-6">Next.jsとFastAPIの連携アプリ</h1>
      <div className="space-y-8">
        {/* GETリクエスト */}
        <section>
          <h2 className="text-xl font-bold mb-4">GETリクエストを送信</h2>
          <button
            onClick={handleGetRequest}
            className="bg-blue-500 hover:bg-blue-600 text-white px-4 py-2 rounded"
          >
            GETリクエストを送信
          </button>
          {getMessage && (
            <p className="mt-2">サーバーからのGET応答: {getMessage}</p>
          )}
        </section>
```

useState

getMessage:定数名
setGetMessage：関数。引数に渡した数値を「getMessage」に格納している

getMessageに値が入ったら<p>タグが表示される

fetch
特定のURLにリクエストを送信する。他にもaxios等でも実装可能。

〇fetch(URL)でリクエストを実施。今回の場合は以下となる。
・URL:http://localhost:8000/api/hello

〇結果は、”await response.json()”で取得している

本資料の目的　Output編（前半）

マイクロサービスアーキテクチャの概念を理解し、Next.js（フロントエンド）、FastAPI（バックエンド）というモダンなフレームワークの構築を行います。for Practicalにむけて、シンプルで簡単な実装から始めます。

アジェンダ

Input

マイクロサービスアーキテクチャ
- └モノリシックアーキテクチャとの対比
- └技術要素の解説
  - └Next.js
  - └FastAPi

サンプルアプリ
- └サンプルアプリの挙動
- └アーキテクチャ
- └フロントエンド/バックエンドの構成

Output

- └演習
  - └ローカルビルド（チュートリアル）
  - └Starterパッケージのゴール（宿題）

本資料の目的

マイクロサービスアーキテクチャの概念を理解し、Next.js（フロントエンド）、FastAPI（バックエンド）というモダンなフレームワークを演習を通して体得する。それによって、MVP実装に必要な技術力を養う。

Input

Output

アジェンダ

マイクロサービスアーキテクチャ
└モノリシックアーキテクチャとの対比
└技術要素の解説
└Next.js
└FastAPI

サンプルアプリ
└サンプルアプリの挙動
└アーキテクチャ
└フロントエンド/バックエンドの構成

└演習
└ローカルビルド（チュートリアル）
└Starterパッケージのゴール（宿題）

手順

- リポジトリClone
- 環境を整えて
  Next.jsとFastAPIを起動する

チュートリアル：まずはサンプルアプリをローカル環境で動かそう-リポジトリクローン

リポジトリのクローンをSourceTreeまたはGitコマンド経由で行う。あるいはDLでもOK
https://github.com/takuya-tech0/LinkFastAPINextStarter

CloneまたはDLの方法

Git

CLI

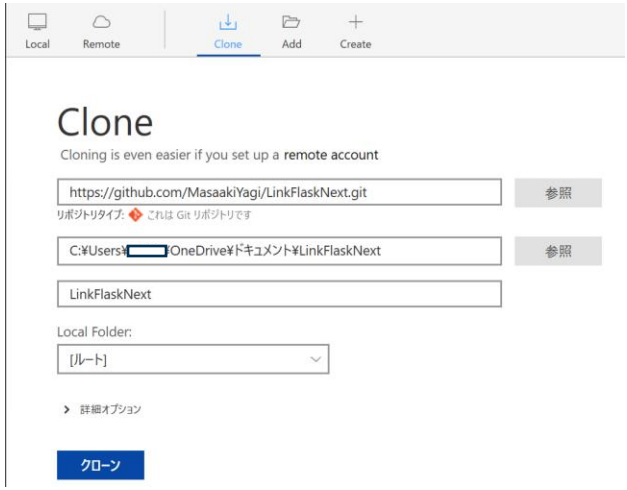
- クローンしたいローカルのパスにcdコマンドで移動
- git clone https://github.com/wkzhiro/nextjs-fastapi-sample.git

SourceTree

- Cloneタブにてリポジトリの URLを指定して「クローン」を実行

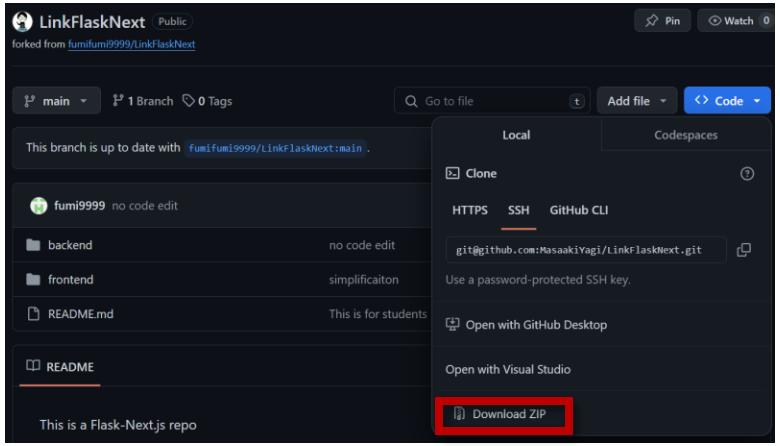
再掲

参考：GitHub flowハンズオン



Zipダウンロード

- リポジトリページでCode>Download ZIP



本資料の目的

マイクロサービスアーキテクチャの概念を理解し、Next.js（フロントエンド）、FastAPI（バックエンド）というモダンなフレームワークを演習を通して体得する。それによって、MVP実装に必要な技術力を養う。

Input

Output

アジェンダ

マイクロサービスアーキテクチャ
└モノリシックアーキテクチャとの対比
└技術要素の解説
└Next.js
└FastAPI

サンプルアプリ
└サンプルアプリの挙動
└アーキテクチャ
└フロントエンド/バックエンドの構成

└演習
└ローカルビルド（チュートリアル）
└Starterパッケージのゴール（宿題）

手順

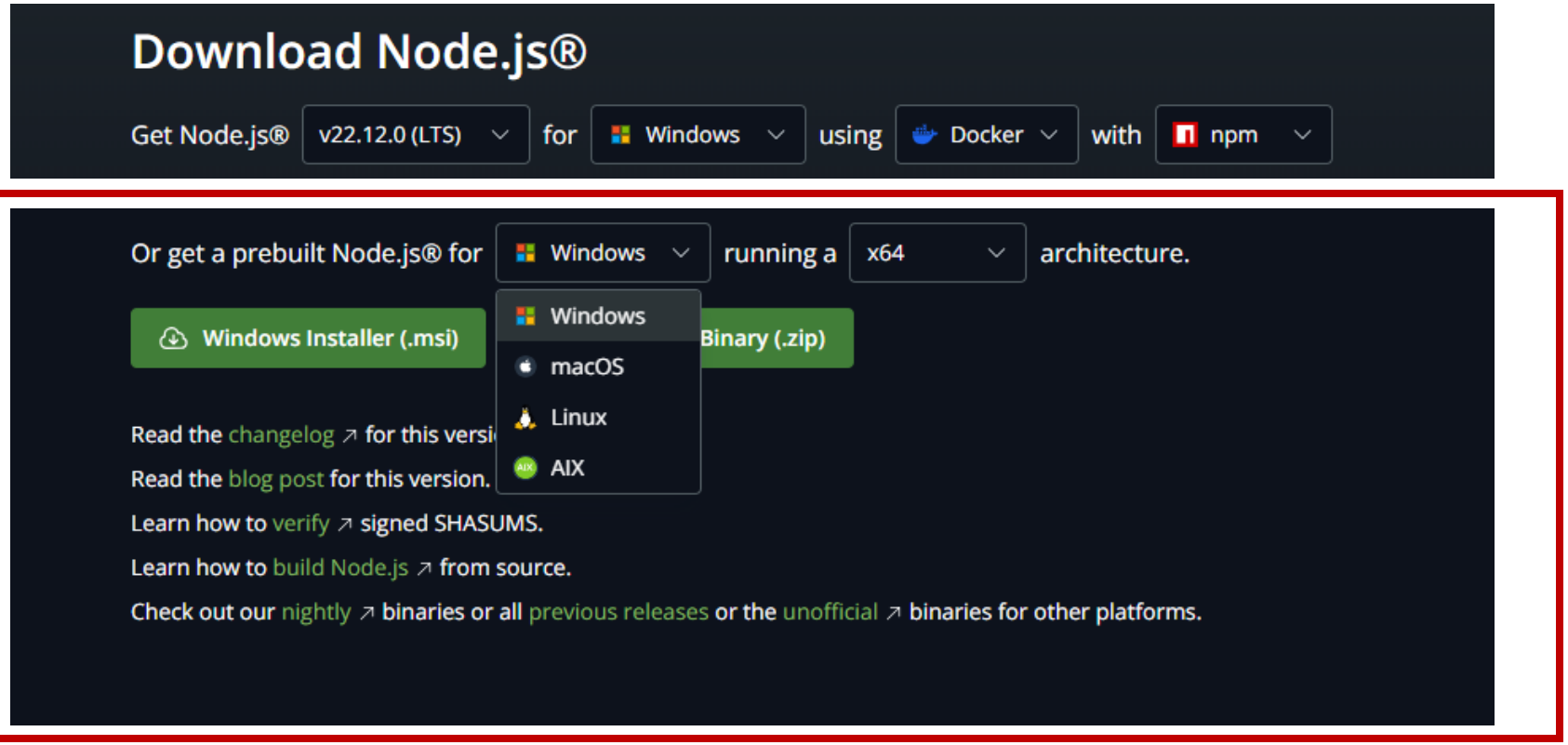
- リポジトリClone
- 環境を整えて
  Next.jsとFastAPIを起動する

環境を整えてNext.jsとFastAPIを起動する（１）　Node.js

ローカル環境にNode.jsをインストールしていない場合、公式サイトからインストーラをダウンロードしインストールする。Node.jsとはサーバーサイドのJavaScript実行環境です。Next.js 15はNode.jsのバージョン18.18が要件となっています。

Node.js公式
https://nodejs.org/en
https://nodejs.org/en/download

参考サイト：https://qiita.com/sefoo0104/items/0653c935ea4a4db9dc2b
※Voltaというnode.jsのパッケージ管理システムを使うこともお勧めします。
今回は使用していませんので、興味ある方はお試しください。
Voltaのインストール：https://www.geeklibrary.jp/counter-attack/volta/
next.js×Volta：https://sinpe-pgm.com/nextjs-volta-nodejs/

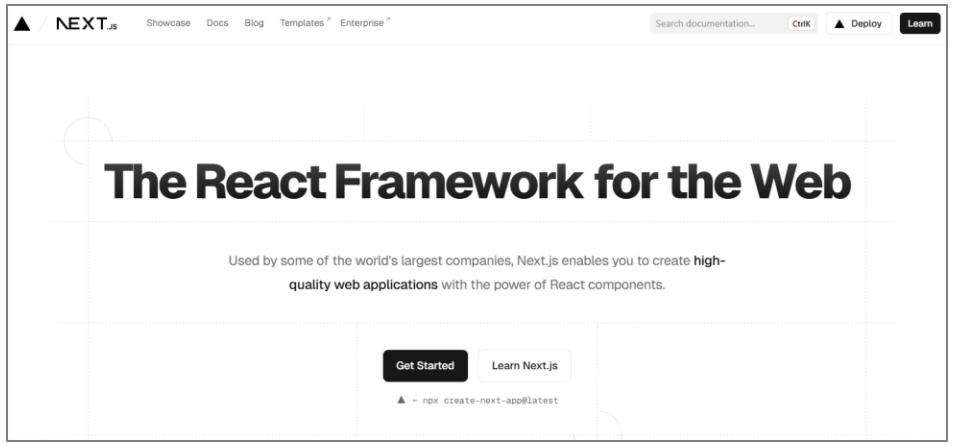


環境を整えてNext.jsとFastAPIを起動する（２）　Next.js-1

Next.jsをインストールのため、ターミナルを開きプロジェクトを作成したいディレクトリに移動して以下のコマンドを実行します。

“ npx create-next-app@latest ”

Next.js公式※e-learningあり
https://nextjs.org/
https://nextjs.org/docs/app/getting-started



環境を整えてNext.jsとFastAPIを起動する（２）　Next.js-2

Next.jsの一式が保存されているディレクトリに移動して　“npm run dev “を実行して起動します。

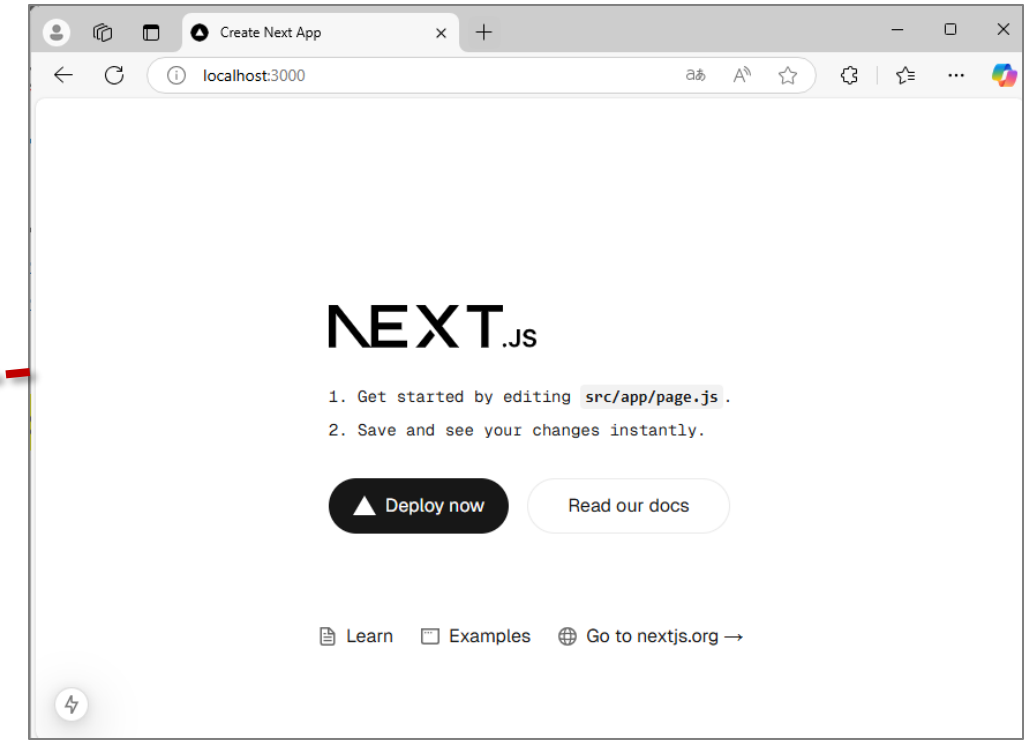
```
(pra2) PS C:\Users\                    \pra_test\starter> npm run dev

> starter@0.1.0 dev
> next dev

   ▲ Next.js 15.1.3
   - Local:        http://localhost:3000
   - Network:      http://192.168.0.249:3000

 ✓ Starting...
 ✓ Ready in 2.4s
 ○ Compiling / ...
 ✓ Compiled / in 4.7s (662 modules)
 GET / 200 in 5230ms
 ✓ Compiled in 530ms (306 modules)
 GET / 200 in 64ms
```

CTRL+クリックで
ブラウザが立ち上がります。



http://localhost:3000/に
Next.Jsのデフォルトページが表示されていればＯＫ！

環境を整えてNext.jsとFastAPIを起動する（３）　FastAPI-1

もし余裕があれば、バックエンド用にPythonの仮想環境を作成してみてください。
https://zenn.dev/mom/books/ca5cefe5d0855e/viewer/16d956

・Python の仮想環境を作成します
環境名は仮でvenv
すでにvenvで作成している場合は
別名で作成してください

```
python -m venv venv
```

・仮想環境を有効化します

macOS/Linuxの場合:

```
source venv/bin/activate
```

Windowsの場合:

```
venv¥Scripts¥activate
```

環境を整えてNext.jsとFastAPIを起動する（３）　FastAPI-2

指定の仮想環境をアクティブにする、もしくはデフォルトの環境でPythonのコードを書いて実行するのみです。公式のチュートリアルも充実しています（ https://fastapi.tiangolo.com/ja/tutorial/ ）

FastAPIのインストール

```
pip install fastapi
pip install uvicorn
```

app.pyファイルを任意のディレクトリ（例backend）に作成する

```
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
async def root():
    return {"message": "Hello World"}
```

環境を整えてNext.jsとFastAPIを起動する（３）　FastAPI-3

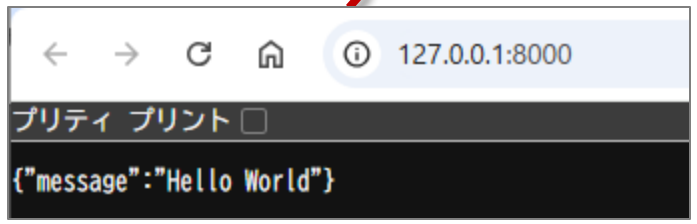
指定の仮想環境をアクティブにする、もしくはデフォルトの環境でPythonのコードを書いて実行するのみです。公式のチュートリアルも充実しています（ https://fastapi.tiangolo.com/ja/tutorial/ ）

app.pyが保存されているディレクトリに移動して、uvicorn app:app –reload　を実行します
※資料作成者はminicondaを利用しています

実行コード

```
(pra1) PS C:\Users\                                    \pra1\nextjs-fastapi-starter\backend> uvicorn app:app --reload
INFO:     Will watch for changes in these directories: ['C:\\Users\\shida\\OneDrive\\00_tech
s-fastapi-starter\\backend']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [15272] using StatReload
INFO:     Started server process [36056]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

CTRL+クリックで
ブラウザが立ち上がります。



http://127.0.0.1:8000/に
app.pyで指定した文字列が表示されていればＯＫ！

本資料の目的

マイクロサービスアーキテクチャの概念を理解し、Next.js（フロントエンド）、FastAPI（バックエンド）というモダンなフレームワークを演習を通して体得する。それによって、MVP実装に必要な技術力を養う。

アジェンダ

Input

マイクロサービスアーキテクチャ
- └モノリシックアーキテクチャとの対比
- └技術要素の解説
  - └Next.js
  - └FastAPI

サンプルアプリ
- └サンプルアプリの挙動
- └アーキテクチャ
- └フロントエンド/バックエンドの構成

Output

- └演習
  - └ローカルビルド（チュートリアル）
  - └Starterパッケージのゴール（宿題）

宿題ゴールについて　その①（システム理解フェーズ）

以下から1つ選んで実施してください。Mustは１つ。発展は１つ以上。

オプション1：処理フロー図の作成
「既存の3つの機能について、ユーザーがボタンを押してから結果が表示されるまでの流れを図解せよ」
シーケンス図、フロー図、手書きメモなど形式は自由
フロントエンド→バックエンド→フロントエンドのデータの流れを可視化
学習効果：API通信の仕組みを理解

オプション2：要件定義書の作成
「追加する2つの新機能（数値を2で割る、文字数カウント）について、以下を含む要件定義書を作成せよ」
機能概要
入力/出力仕様
想定されるエラーケース
学習効果：実装前の設計思考を養う

オプション3：コードリーディングレポート
「既存コードを読み、以下を説明せよ」
Next.jsのどのファイルがFastAPIのどのエンドポイントを呼んでいるか
各ファイルの役割（routing、API呼び出し、UI表示など）
気づいた点や改善案　など
学習効果：コードを読む力の向上

オプション4：API仕様書の作成
「現在の3つのエンドポイントと、追加する2つのエンドポイントについてAPI仕様書を作成せよ」
エンドポイント、HTTPメソッド、リクエスト/レスポンス形式
表形式やMarkdownなど形式は自由
学習効果：API設計の理解

オプション5：アーキテクチャ説明動画/スライド
「このアプリの構成を後輩に説明するための資料を作成せよ（3-5分程度）」
フロントエンドとバックエンドの役割分担
なぜこの構成にしているのか　など
学習効果：全体像の把握と説明力

宿題ゴールについて　その②（システム実装フェーズ）

Next.js、FastAPIのプロジェクトを正常に立ち上げた後、以下の通りに修正してください。

Must

① 以下3つのコンポーネントがローカル環境で
正常に動くようNext.js、FastAPIを立ち上げてください
- ✓ GETリクエストを送信
→”サーバーからのGET応答～“確認
- ✓ 数値を2倍するリクエストを送信
→2倍された数値が表示されることを確認
- ✓ POSTリクエストを送信
→送信したメッセージがechoされることを確認

② GETリクエストを送信のメッセージを変更
Hello World by FastAPI→Hello!氏名になるよう
コードを修正してください

③「数値を2で割るリクエストを送信」
コンポーネントを追加してください

Want

④「文字数をカウントするリクエストを送信」
コンポーネントを追加してください