フロントエンド学習用リポジトリ

Reactの基礎からNext.jsまで、段階的に学習するための教材です。
各フォルダは番号順（01〜06）に進めることで、スムーズに学習できるように設計されています。

📚 学習ロードマップ

順番	フォルダ	テーマ	学習内容
01	`01react-only`	**Reactの基礎（Client Only）**	状態管理、 コンポーネント設計
02	`02express-api`	**REST APIの基礎**	HTTP通信、CRUD操作
03	`03react-todo`	**ReactとAPIの連携**	fetchの使い方、useEffect
04	`04nextjs-client-only`	**Next.js Client Components**	`'use client'` ディレクティブ
05	`05nextjs-server-only`	**Next.js Server Components**	サーバーサイド処理
06	`06nextjs-todo`	**Next.js 実践（完全版）**	API Routes + Components

01 React Only (`01react-only/`)

🎯 このフォルダで学ぶこと

- Reactコンポーネントの基本
- `useState` フックで状態管理
- イベントハンドリング
- Todoアプリの実装

📖 詳しい説明

React Only は、**フロントエンドのみで完全に動作する** Todoアプリケーションです。
APIサーバーは不要で、すべての処理がブラウザ内で行われます。

このアプリの特徴：

- 外部サーバーに依存しない
- 状態管理は全て `useState` で行う
- ページをリロードするとデータはリセット
- シンプルなコンポーネント構造

📁 ファイル構成

```
01react-only/
├── src/
│   ├── App.jsx              ← メインコンポーネント
│   ├── App.css
│   ├── components/
│   │   ├── TodoInput.jsx    ← 入力フォーム
│   │   ├── TodoItem.jsx     ← Todo項目
│   │   └── Stats.jsx        ← 統計表示
│   └── index.js
└── package.json
```

🔑 重要なコンセプト

コンポーネント設計の例：

```
export const Stats = ({ total, done }) => (
  <div className="stats">
    全{total}件 / 完了{done}件
  </div>
);
```

- props で親から受け取ったデータを表示
- アロー関数形式で シンプルに記述
- 表示のみに専念

🚀 起動手順

```
cd 01react-only
npm install
npm start
```

ブラウザが自動で開き、 `http://localhost:3000` で起動します。

💡 学習ポイント

1. **useState とは？**
   - 関数型コンポーネントで状態を管理
   - `const [todos, setTodos] = useState([])` で初期値を設定
2. **親子間データのやり取り**
   - 親から子へ: `props` で渡す
   - 子から親へ: コールバック関数を使う
3. **CRUD操作（ローカル）**
   - Create（作成）: `setTodos([...todos, newTodo])`
   - Read（読み取り）: 配列をそのまま表示
   - Update（更新）: `todos.map()` で該当要素を変更
   - Delete（削除）: `todos.filter()` で該当要素を除外

❓ よくある質問

Q: ページをリロードするとデータが消えるのはなぜ？

- A: ローカルストレージを使用していないから。メモリ上のみに保持。

Q: どうやって複数のTodoを管理している？

- A: `useState` で配列を管理。各Todoに一意なIDを付与。

02 Express API (`02express-api/`)

🎯 このフォルダで学ぶこと

- REST APIの基本概念
- HTTPメソッド（GET、POST、PUT、DELETE）

- Express.jsの使い方
- CORSの設定
- JSONデータの送受信

📖 詳しい説明

Express APIは、**バックエンドサーバー** です。
01で作ったReact Onlyはサーバーを持たずローカルで動作していましたが、
ここからは「サーバーとクライアント」の2つの部分に分かれます。

REST API とは？

- クライアント（ブラウザ）からサーバーへ **HTTP通信** でデータをやり取り
- HTTPメソッド（GET/POST/PUT/DELETE）で「何をするのか」を表現
- JSONフォーマットでデータを送受信

🔑 HTTPメソッド

メソッド	役割	例
GET	データ取得	全Todoを表示
POST	データ作成	新しいTodoを追加
PUT	データ更新	Todoを完了にする
DELETE	データ削除	Todoを削除

📁 ファイル構成

```
02express-api/
├── server.js            ← サーバー本体
├── package.json
└── README.md
```

🔗 エンドポイント一覧

```
GET    /api/todos        → 全Todoを取得
POST   /api/todos        → 新Todoを作成
PUT    /api/todos/:id    → TodoIDで更新（完了/未完了切り替え）
DELETE /api/todos/:id    → TodoIDを削除
```

🚀 起動手順

```
cd 02express-api
npm install
node server.js
```

サーバーが `http://localhost:3001` で起動します。

💡 学習ポイント

1. **Express のルーティング**

   ```
   app.get('/api/todos', (req, res) => {
     res.json(todos);
   });
   ```

2. **CORSってなに？**
   - 異なるドメイン間のアクセス制限
   - `localhost:3000`（React）から `localhost:3001`（API）へアクセスするので必要
3. **JSONレスポンス**
   - `res.json()` でJSONを返す
   - HTTPステータスコード（200、201、404など）で結果を示す

03 React Todo (`03react-todo/`)

🎯 このフォルダで学ぶこと

- React と Express API の連携
- `useEffect` フックの使い方
- `fetch` でAPI呼び出し

- 非同期処理のハンドリング

📖 詳しい説明

React Todo は、**01react-only に API連携を加えたバージョン** です。

01と03の違い：

- 01: ブラウザのメモリにのみ保存 → ページをリロードするとデータ消失
- 03: Express API にデータを保存 → ページをリロードしてもデータが残る

📂 ファイル構成

```
03react-todo/
├── src/
│   ├── App.jsx                ← APIを呼び出す
│   ├── components/
│   │   ├── TodoInput.jsx
│   │   ├── TodoItem.jsx
│   │   └── Stats.jsx
│   └── index.js
└── package.json
```

🔑 重要な概念：useEffect

```
useEffect(() => {
  fetchTodos();  // 初回レンダリング後に1回だけ実行
}, []);  // 空の依存配列
```

- ページを最初に表示したときに、サーバーからTodoを取得
- API連携では **必ずこのパターン** を使う

🚀 起動手順

```
# ターミナル1：APIサーバーを起動
cd 02express-api
npm start

# ターミナル2：Reactアプリを起動
cd 03react-todo
npm install
npm start
```

- React: `http://localhost:3000`
- API: `http://localhost:3001`

💡 学習ポイント

1. **fetch の基本**

   ```
   const response = await fetch('http://localhost:3001/api/todos');
   const data = await response.json();
   ```

2. **非同期処理（async/await）**
   - `async` 関数内でAPI呼び出し
   - `await` で結果を待つ
3. **エラーハンドリング**

   ```
   try {
     // API呼び出し
   } catch (error) {
     console.error('エラー:', error);
   }
   ```

04 Next.js Client Only (`04nextjs-client-only/`)

🎯 このフォルダで学ぶこと

- Next.js の基本
- `'use client'` ディレクティブ

- Client Components の動き
- React との違い

📖 詳しい説明

Next.js は **React をベースにしたフレームワーク** です。

React との大きな違い：

- React: `npm create-react-app` で手動セットアップ
- Next.js: コマンド1つで自動セットアップ、プロジェクト構造も決まっている

Client Only とは？

- ブラウザ側のみで動作（React と同じ）
- `'use client'` を付けることで「これはクライアント専用」と明示

📁 ファイル構成

```
04nextjs-client-only/
├── src/
│   └── app/
│       ├── page.jsx          ← 最初に表示されるページ
│       ├── layout.jsx        ← 全ページ共通のレイアウト
│       ├── globals.css
│       └── hobby/
│           └── page.jsx      ← hobby ページ
├── components/
│   └── Navigation.jsx
└── package.json
```

🔑 Next.js ファイルベースルーティング

```
src/app/page.jsx          → http://localhost:3000/
src/app/hobby/page.jsx    → http://localhost:3000/hobby
```

ファイルの場所が **URL になる** ！

🚀 起動手順

```
cd 04nextjs-client-only
npm install
npm run dev
```

ブラウザが自動で開き、 `http://localhost:3000` で起動します。

💡 学習ポイント

1. **'use client' ディレクティブ**

   ```
   'use client';  // このコンポーネントはクライアント専用
   import { useState } from 'react';
   ```

2. **ファイルベースルーティング**
   - `page.jsx` がそれぞれのページ
   - フォルダ構造がそのまま URL パスになる
3. **layout.jsx の役割**
   - 共通のヘッダー、フッター、ナビゲーション
   - 複数ページで共有されるコンポーネント

05 Next.js Server Only (`05nextjs-server-only/`)

🎯 このフォルダで学ぶこと

- Server Components（デフォルト動作）
- サーバーサイドでの処理
- `'use client'` を使わないコンポーネント
- パフォーマンスの向上

📖 詳しい説明

Next.js は **デフォルトで Server Components** です。

Server Components とは？

- ブラウザで実行されない（サーバーだけで実行）

- JavaScriptコードがクライアントに送られない → 高速
- データベースアクセスなど、秘密の処理ができる

04（Client Only）との違い：

- 04: `'use client'` で明示的にクライアント専用
- 05: `'use client'` を書かなければサーバーで実行される

📁 ファイル構成

```
05nextjs-server-only/
├── src/
│   └── app/
│       ├── page.jsx          ← Server Component
│       ├── layout.jsx
│       ├── globals.css
│       └── hobby/
│           └── page.jsx
├── components/
│   └── Navigation.jsx        ← Server Component
└── package.json
```

🔑 Server vs Client

特徴	Server Component	Client Component
ディレクティブ	不要	`'use client'` 必須
useState使用	✗ 不可	✓ 使用可
useEffect使用	✗ 不可	✓ 使用可
イベントハンドラ	✗ 不可	✓ 使用可
実行場所	サーバー	ブラウザ
送信サイズ	小さい	大きい

🚀 起動手順

```
cd 05nextjs-server-only
npm install
npm run dev
```

ブラウザが自動で開き、 `http://localhost:3000` で起動します。

💡 学習ポイント

1. **Server Component の例**

   ```
   // 'use client' が ないので Server Component
   export default function Page() {
     return <div>Server で実行される</div>;
   }
   ```

2. **データベースアクセス**
   - Server Component なら直接データベースに接続可能
   - Client Component だと接続情報が漏れるのでNG
3. **パフォーマンスの向上**
   - サーバーで処理を済ませて完成したHTMLをクライアントに送信
   - ブラウザの負担が少ない

06 Next.js Todo - 完全版 (`06nextjs-todo/`)

🎯 このフォルダで学ぶこと

- Client Components と Server Components の使い分け
- API Routes（Next.js内のバックエンド）
- 実践的なアプリケーション設計
- デプロイまでの全体像

📖 詳しい説明

Next.js Todo は、ここまで学んだ知識を集大成したプロジェクトです。

これまでとの大きな違い：

- 01-03: React（フロントエンド） + Express（バックエンド） = 2つに分かれていた
- 06: Next.js 1つに統合 → API Routes でバックエンドも含まれる

📂 ファイル構成

```
06nextjs-todo/
├── src/
│   ├── app/
│   │   ├── page.jsx              ← Client Component
│   │   ├── layout.jsx            ← Server Component
│   │   ├── globals.css
│   │   ├── api/
│   │   │   └── todos/
│   │   │       ├── route.js      ← API (GET, POST)
│   │   │       ├── todoData.js
│   │   │       └── [id]/
│   │   │           └── route.js ← API (PUT, DELETE)
│   │   └── hobby/
│   │       └── page.jsx
│   └── components/
│       ├── TodoInput.jsx         ← Client Component
│       ├── TodoItem.jsx          ← Client Component
│       └── Stats.jsx             ← Client Component
└── package.json
```

🔑 API Routes の構造

```
api/todos/route.js
  ↓
GET  /api/todos       → route.js の export function GET
POST /api/todos       → route.js の export function POST

api/todos/[id]/route.js
  ↓
PUT    /api/todos/1   → route.js の export function PUT
DELETE /api/todos/1   → route.js の export function DELETE
```

🚀 起動手順

```
cd 06nextjs-todo
npm install
npm run dev
```

ブラウザが自動で開き、 `http://localhost:3000` で起動します。

注意: Express サーバーは不要です。Next.js が 1つですべて処理します。

💡 学習ポイント

1. **API Routes の書き方**

   ```
   // src/app/api/todos/route.js
   export async function GET(request) {
     return Response.json(todos);
   }

   export async function POST(request) {
     const data = await request.json();
     // Todoを作成
     return Response.json(newTodo, { status: 201 });
   }
   ```

2. **Client Component でのfetch**

   ```
   'use client';
   const response = await fetch('/api/todos');  // 相対パス！
   ```

3. **ファイル構造での分かりやすさ**
   - フロントエンド（components）
   - バックエンド（api）
   - が同じプロジェクト内に共存

🎓 全体的な学習フロー図

```
01: React Only
    ↓（ローカルだけで完結、データ保存なし）

02: Express API
    ↓（バックエンドサーバーの基礎を学ぶ）

03: React Todo
    ↓（フロントエンド + バックエンド の連携）

04: Next.js Client Only
    ↓（Next.js でフロントエンドを作る）

05: Next.js Server Only
    ↓（Server Components でパフォーマンスアップ）

06: Next.js Todo
    ↓（フロント＆バックエンド統合 → 完成！）
```

📊 プロジェクト比較表

項目	01 React	02 API	03 React	04 Client	05 Server	06 Todo
フロントエンド	✓	-	✓	✓	✓	✓
バックエンド	-	✓	✓	-	-	✓
データ保存	✗	✓	✓	✗	✗	✓
使用フレームワーク	React	Express	React	Next.js	Next.js	Next.js
起動ポート	3000	3001	3000,3001	3000	3000	3000
難易度	★☆☆	★★☆	★★☆	★★☆	★★★	★★★

🚀 各プロジェクトの起動方法（まとめ）

01: React Only

```
cd 01react-only && npm install && npm start
```

02: Express API

```
cd 02express-api && npm install && node server.js
```

03: React Todo（API併用）

```
# ターミナル1
cd 02express-api && npm start

# ターミナル2
cd 03react-todo && npm install && npm start
```

04: Next.js Client Only

```
cd 04nextjs-client-only && npm install && npm run dev
```

05: Next.js Server Only

```
cd 05nextjs-server-only && npm install && npm run dev
```

06: Next.js Todo

```
cd 06nextjs-todo && npm install && npm run dev
```

💡 学習のコツ

1. **順番を守る**
   - 01 → 02 → 03 と進めることで、段階的に理解できます
2. **コードを読む**
   - 提供されたコードをじっくり読んで、何をしているか理解する
   - 完全に理解してから次へ
3. **手を動かす**
   - 自分で少し変更してみる
   - 例：タスクの表示順を逆にしてみるなど
4. **エラーを楽しむ**
   - エラーが出たら何が原因か考える
   - エラーメッセージをよく読む
5. **ドキュメントを見る**
   - React・Express・Next.js の公式ドキュメントを参照
   - Google や Chat GPT で質問するのもOK

📚 参考リソース

公式ドキュメント

- React 公式
- Express.js 公式
- Next.js 公式

フレームワーク選択ガイド

- **フロントエンドのみ** → React
- **フロント＆バック分ける** → React + Express
- **フロント＆バック統合** → Next.js

✅ よくある質問

Q1: 01 と 03 の違いは？

A: 01 はメモリ保存（ページリロード時消失）、03 はサーバー保存（永続化）

Q2: 04 と 05 の違いは？

A: 04 は `'use client'` でクライアント専用、05 は `'use client'` なしでサーバー実行

Q3: 06 で Express サーバーは必要？

A: いいえ。API Routes で Next.js 内に統合されているので不要

Q4: どのプロジェクトから始めるべき？

A: 必ず **01 から順に** 進めてください

Q5: エラーが出たときは？

A: コンソールを見て、エラーメッセージをよく読む。ネット検索も活用！

📄 ライセンス

最後に、**こまめに手を動かすことが最も大切** です。頑張ってください！ 🎉