FARM Stackとは

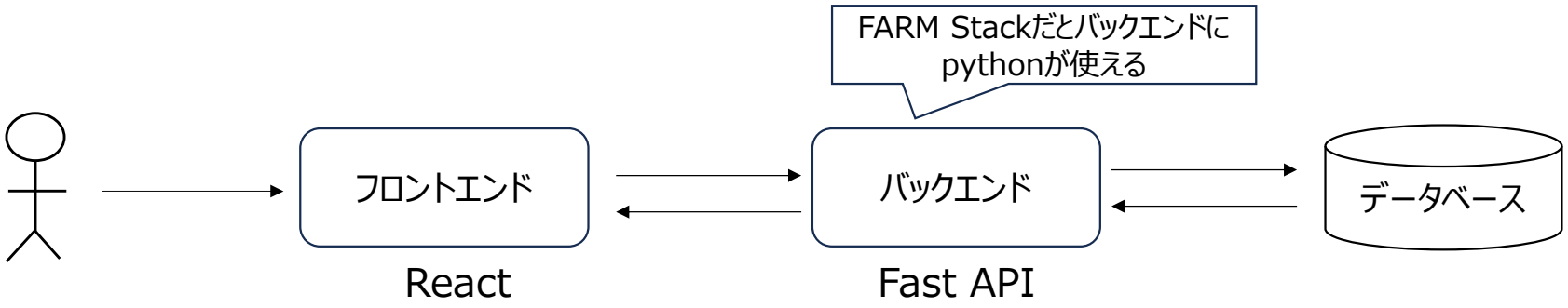
FARM Stack

バックエンド	**Fast API**
フロントエンド	**React**
データベース	**Mongo DB**

※本講座ではデータベースはsqliteを使用

MERN Stack

バックエンド	**Node.js Express**
フロントエンド	**React**
データベース	**Mongo DB**



【React】 アプリ① アンケートフォーム　で学ぶ主な項目

- 環境を整える（npx create-react-appの使い方）
- JSX記法
- コンポーネントの使い方（export,importなどの方法）
- イベント処理（ボタンを押下した場合など）
- useState（コンポーネントのstateを保持、更新するためのフック）
- useNavigate（ページ遷移のためのフック）
- useLocation（ページ遷移時にstateを渡すためのフック）
- CSSフレームワークmuiの使い方

まずはmui無しで作って、その後、muiを使ってデザインしてみる

create-react-app ～ npm start （セクション２）

①node.jsをインストール

②React開発環境を一括でインストール
→npx create-react-app@5.0.1 section1-react
※section1-reactはフォルダ名に該当し、任意の名称。

section1-reactという名前を付けてしまいましたが、
セクション2とセクション3で利用するフォルダです。

③②で作ったフォルダへ移動
cd section1-react

④プロジェクトの起動
npm start

create-react-app ～ npm start （セクション４）

①udemyフォルダ（section1-reactのフォルダの上位フォルダ）にいる状態

②React開発環境を一括でインストール
→npx create-react-app@5.0.1 section4-react
※section4-reactはフォルダ名に該当し、任意の名称。

③②で作ったフォルダへ移動
cd section4-react

④プロジェクトの起動
npm start

muiの基本的な使い方（最初に学ぶとき）

①Containerで全体を囲む

→maxWidthなどを定める

②Boxでひとかたまりの要素群を囲む

→sxを使って、marginやdisplay、flexdirection、alignItemsなどを定める

③各要素(TypographyやTextField、Select、RadioGroup、Button)を作る

→それぞれのデザインや変数を付与する。

【React】 アプリ②のログイン画面 で学ぶ主な項目

- header,footer（各ページ共通のデザイン）
- useContext（各コンポーネント間でデータ共有する）
- axios（APIを利用できるようにする）
- カスタムフック（プログラムの見通しをよくする。→まずはカスタムフック無しで）

＜主な復習項目＞

- useState
- ルーティング関連（useNavigate、Link）
- muiによるデザイン

異なるコンポーネント間でデータ共有する方法

① useNavigateの第２引数にstateとしてデータを含めて、遷移させる。(section3の復習)

② 親コンポーネントから子コンポーネントへpropsとして渡す。

③ useContextを使ってグローバル変数として扱えるようにする。

※本講座では扱いませんが、子コンポーネントから親コンポーネントへ渡す方法もあります。

useContextを使ったグローバル変数　手順

< LoginUserProvider.jsにて >

①createContextを使い、 LoginUserContextを作る。名前付きexportとする。

②LoginUserProviderを作り、変数(loginUser)と更新関数(setLoginUser)をuseStateを使って定義する。

returnの中で、<LoginUserContext.Provider>で{children}を囲む。その際value={}の中にloginUserとsetLoginUserを入れる。名前付きexportとする。

<App.jsにて>

④ JSX記法の範囲全体を<LoginUserProvider>で囲む

<Login.jsにて>

⑤LoginUserContextをインポートし、useContext(LoginUserContext)で更新関数(setLoginUser)を使えるようにする。インポートした更新関数を通常通り使って、変数を更新する。

<Header.jsにて>

⑥ LoginUserContextをインポートし、useContext(LoginUserContext)で変数(loginUser)を使えるようにする。

フロントエンド、バックエンド間でのAPIを構築する手順

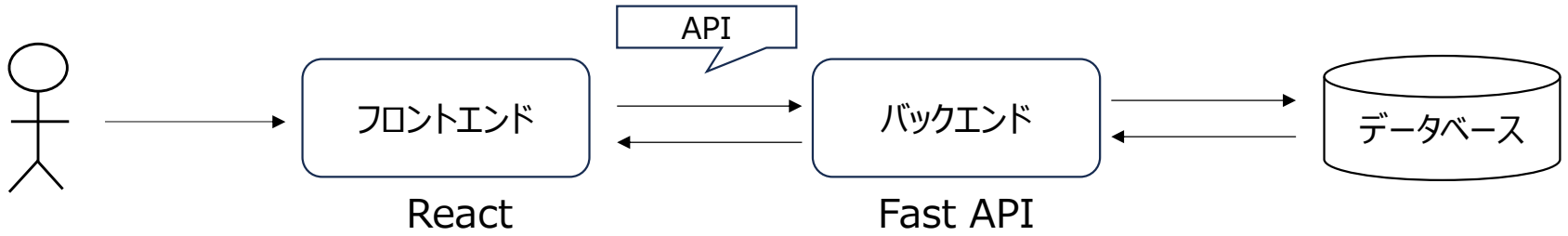
①データベースのテーブルを考える

→主にdatabase.pyに反映する

②必要なAPIのメソッドや引数を考える

→schemas.py,crud.py,main.pyに反映する

②必要なAPIのメソッドや引数を考える



①ユーザを読み込む(get)
→何種類かあると便利そう(全件取得、nameで絞る等)

②ユーザを登録する(post)
→nameとpasswordが登録に必要な情報

③売上データを読み込む(get)
→何種類かあると便利そう(全件取得、yearで絞る等)

④売上データを登録する(post)
→year,department,salesが登録に必要な情報

usersテーブル
- id
- name
- password
- is_active

salesテーブル
- year
- department
- sales

Fast API 公式ドキュメントのフォルダ構成

ファイル名	役割
database.py	• データベースへの接続、セッション管理などを行う。 • sqlalchemyのcreate_engineでSessionLocalを作る。 • sqlalchemyのdeclarative_baseを使って基底クラスであるBaseを作る。
models.py	• データベース内に出来上がるテーブルをclassとして作る。 • その際は、database.pyからBaseを継承する。
schemas.py	• Pydanticスキーマを定義する。=PydanticのBaseModelを継承したclass。 • ここでのPydanticスキーマは、リクエストのバリデーションを行うためのもの。 ※バリデーション：データが所定の形式に合っているかを確認すること。
crud.py	• CRUD（Create, Read, Update, Delete）操作を実際に行う関数を記述する。 • 各関数の引数に必ず、Session(データベースへの接続）が含まれる。 • 各関数の引数に対するバリデーションとしてschemas.pyで定義したPydanticスキーマを使う。 • 各関数の中で、models.pyで定義したclassを用いてCRUD操作を行う。
main.py	• 起点となるファイル。ルーティングの設定、APIサーバーの起動などを記述する。 • Session(データベースへの接続）をdatabase.pyから取得する。 • アクセスされたパスに応じて、crud.pyで定義した関数そのものや関数の戻り値を返却する。 • 各関数の戻り値のバリデーションとしてschemas.pyで定義したPydanticスキーマを使う。

一発で作れる(作るべき)

行ったり来たりしながら作る

Fast API 公式ドキュメントのフォルダ構成

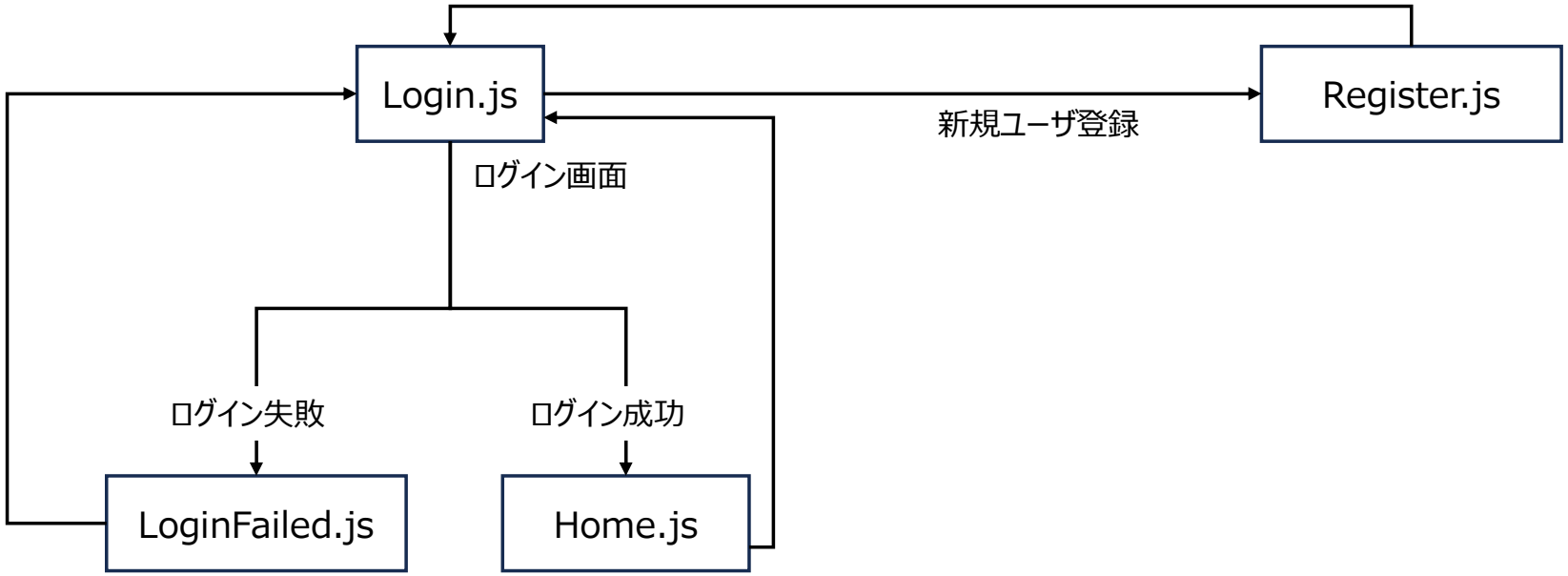
ファイル名	役割
database.py	• データベースへの接続、セッション管理などを行う。 • sqlalchemyのcreate_engineでSessionLocalを作る。 • sqlalchemyのdeclarative_baseを使って基底クラスであるBaseを作る。
models.py	• データベース内に出来上がるテーブルをclassとして作る。 • その際は、database.pyからBaseを継承する。
schemas.py	• Pydanticスキーマを定義する。=PydanticのBaseModelを継承したclass。 • ここでのPydanticスキーマは、リクエストのバリデーションを行うためのもの。 ※バリデーション：データが所定の形式に合っているかを確認すること。
crud.py	• CRUD（Create, Read, Update, Delete）操作を実際に行う関数を記述する。 • 各関数の引数に必ず、Session(データベースへの接続）が含まれる。 • 各関数の引数に対するバリデーションとしてschemas.pyで定義したPydanticスキーマを使う。 • 各関数の中で、models.pyで定義したclassを用いてCRUD操作を行う。
main.py	• 起点となるファイル。ルーティングの設定、APIサーバーの起動などを記述する。 • Session(データベースへの接続）をdatabase.pyから取得する。 • アクセスされたパスに応じて、crud.pyで定義した関数そのものや関数の戻り値を返却する。 • 各関数の戻り値のバリデーションとしてschemas.pyで定義したPydanticスキーマを使う。

各種インストール

npm install react-router-dom

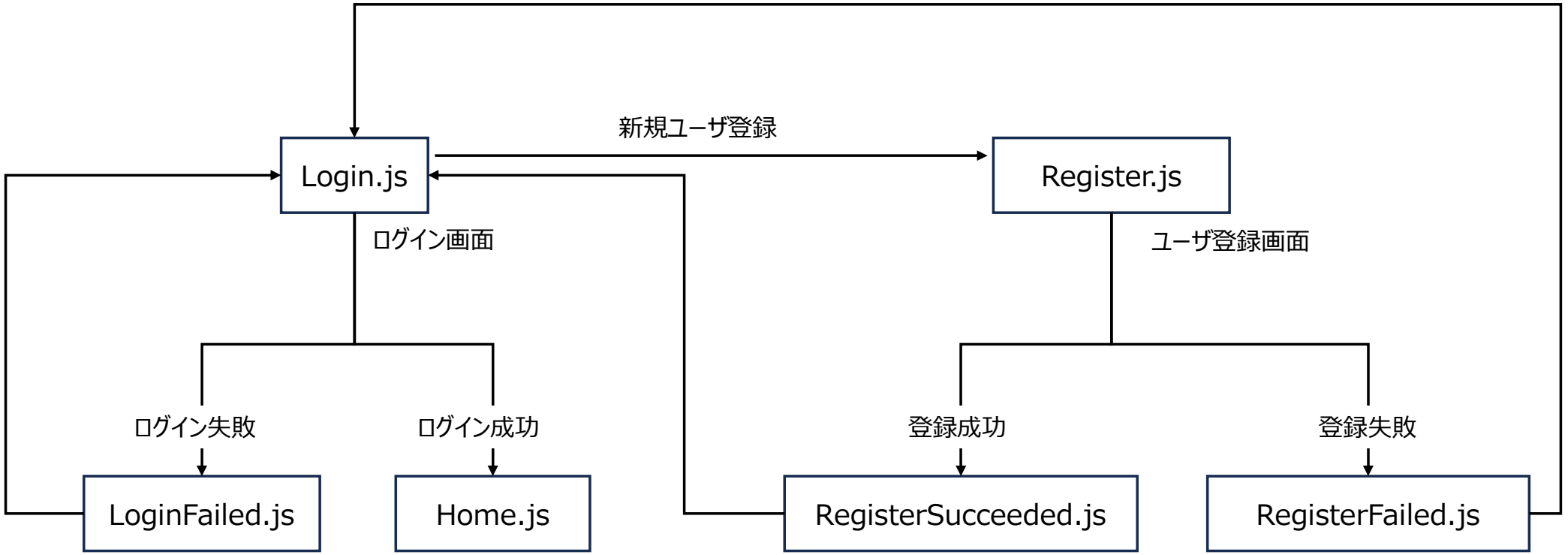
npm install @mui/material @emotion/react @emotion/styled

pagesの構成（セクション4の段階）



NotFound.js

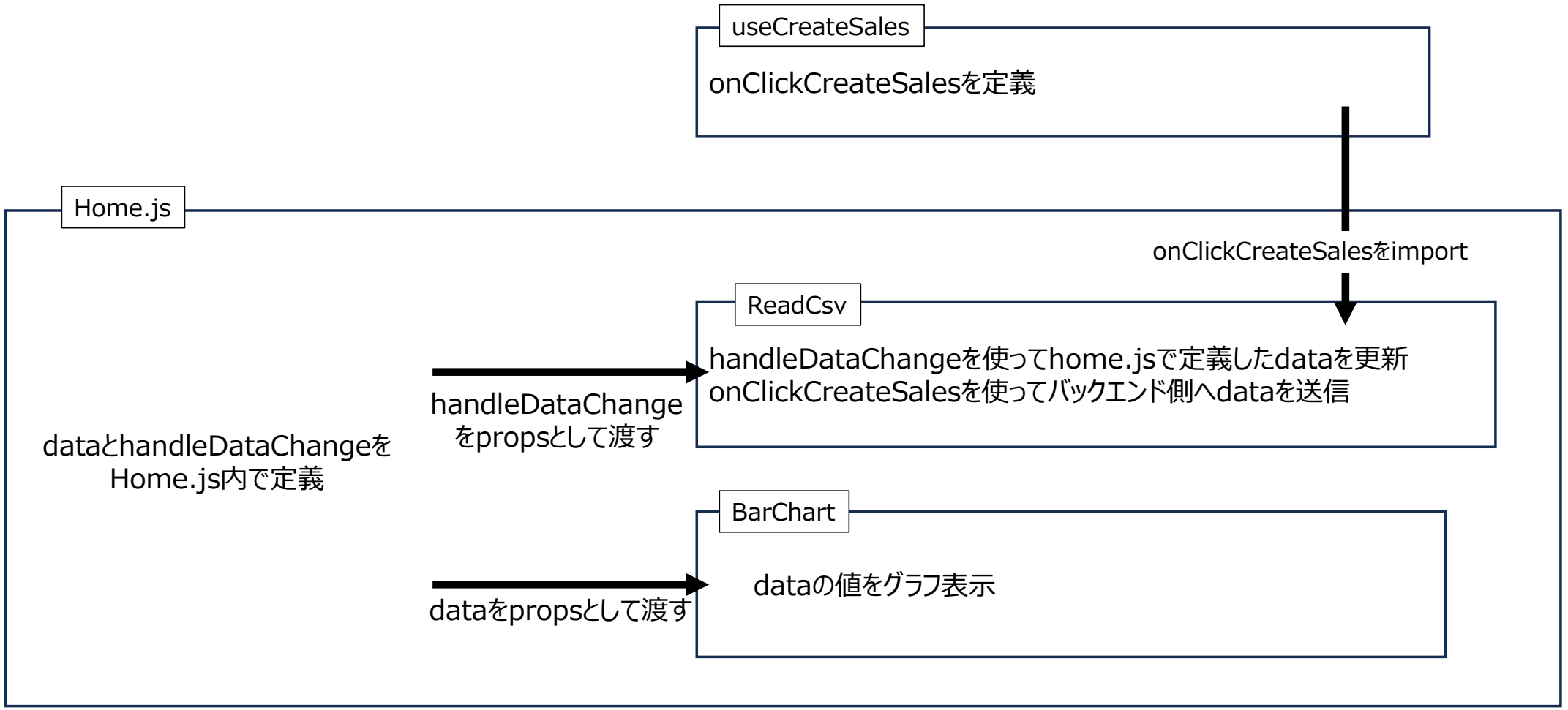
pagesの構成（セクション8の段階）



NotFound.js

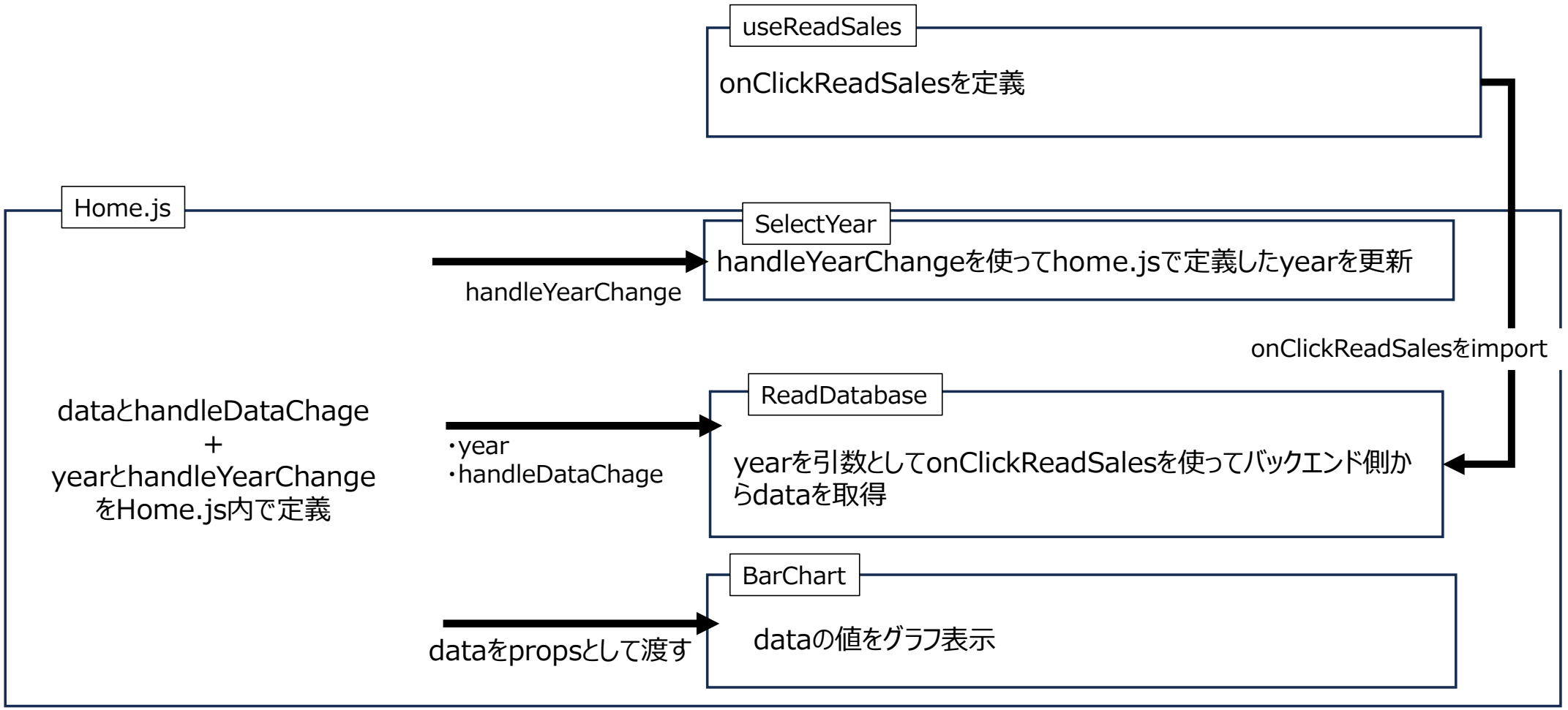
Home.jsの構成（csv読み込み→データベース格納＆グラフ表示）

１段階目



Home.jsの構成（年選択→データベースから読み込み＆グラフ表示）

2段階目



グラフ表示しやすいように、２次元配列をオブジェクトへ変換する

２次元配列

```
[
['department', '第１営業部', '第２営業部', '第３営業部']
['year', '2022', '2022', '2022']
['sales', '100', '150', '200’]
]
```

オブジェクト型

```
{
department: ['第１営業部', ’第2営業部', ‘第３営業部’],
sales: [100,150,200]
year: [2022,2022,2022]
}
```

ReadDatabase.jsの役割
(APIで取得したデータを２次元配列へ変換する)

APIで取得したデータ
どんな形？

２次元配列
[
['department', '第１営業部', '第２営業部', '第３営業部']
['year', '2022', '2022', '2022']
['sales', '100', '150', '200’]
]

useEffectとは？

基本形：
useEffect(ある処理,[依存する値])

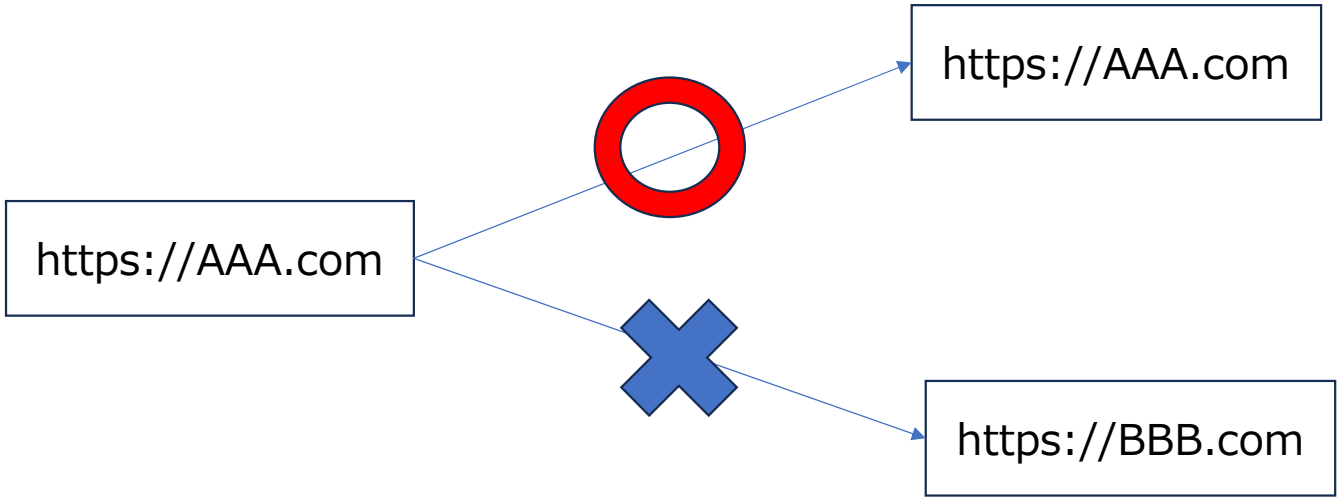
役割：
依存する値に変更があったときにのみ、ある処理を実行する

オリジンとブラウザでの同一オリジンポリシー(Same-Origin policy)

https://AAA.com:443/users

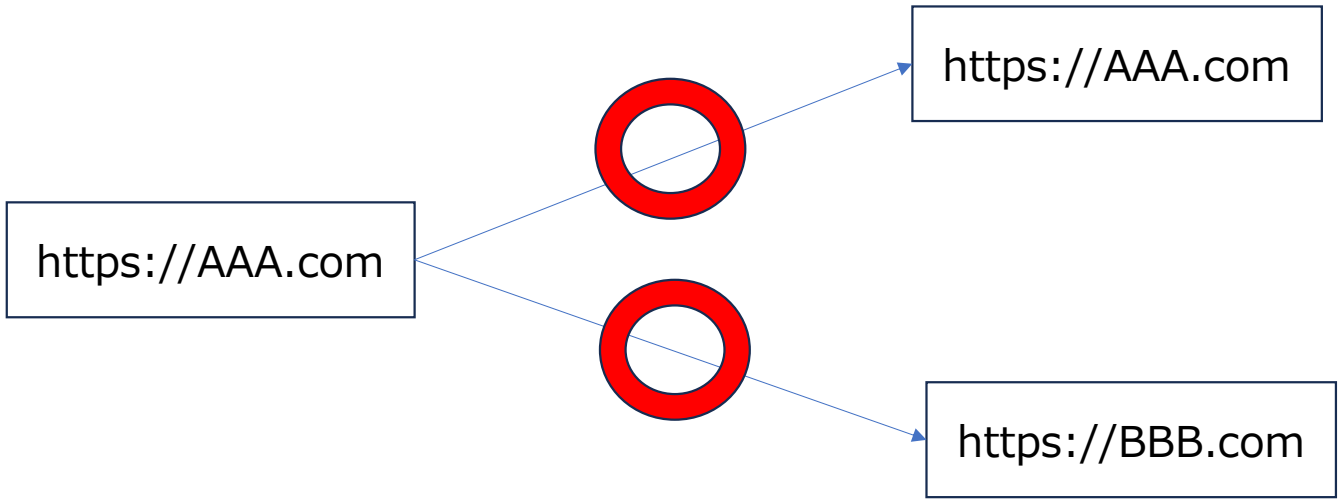
スキーム名　ホスト名　ポート番号　パス名

オリジン



オリジンが異なることを
「クロスオリジン」という

CORS(オリジン間リソース共有,Cross-Origin Resource Sharing)



オリジンをまたいでアクセスを実現する仕組み
（接続先が信用に足るかは別途要確認）

JavaScriptの配列の取り扱い

		使い方例	左記の結果
filter	配列から特定条件だけの要素を残した配列を作る	const words = [‘a’, ‘ab', ‘abc', ‘abcd’]; const result = words.filter((word) => { return word.length > 2; });	[“abc", “abcd"]

map	配列の各要素に対してある計算をした配列を作る	const array1 = [1, 4, 9, 16];
		const map1 = array1.map((x) => {
		return x * 2
		});
			[2, 8, 18, 32]

forEach	配列の要素を取り出す	const array1 = ['a', 'b', 'c’]; array1.forEach((element) => console.log(element));	”a” “b” “c”
	配列の要素を取り出して、オブジェクトに格納する	let array1 = [‘first’, ‘second’, ‘third’]; let obj = {}; array1.forEach((element, index) => { obj['key' + index] = element; });	{ key0: 'first', key1: 'second', key2: 'third' }