

STEP3-1

Practical

宿題パッケージ補足資料

Ver2.1

# 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+SQLite（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、STEP3後半の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

└技術要素の解説「SQLite」

※Next.js、FastAPIはstarterと同様のため割愛

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

Output

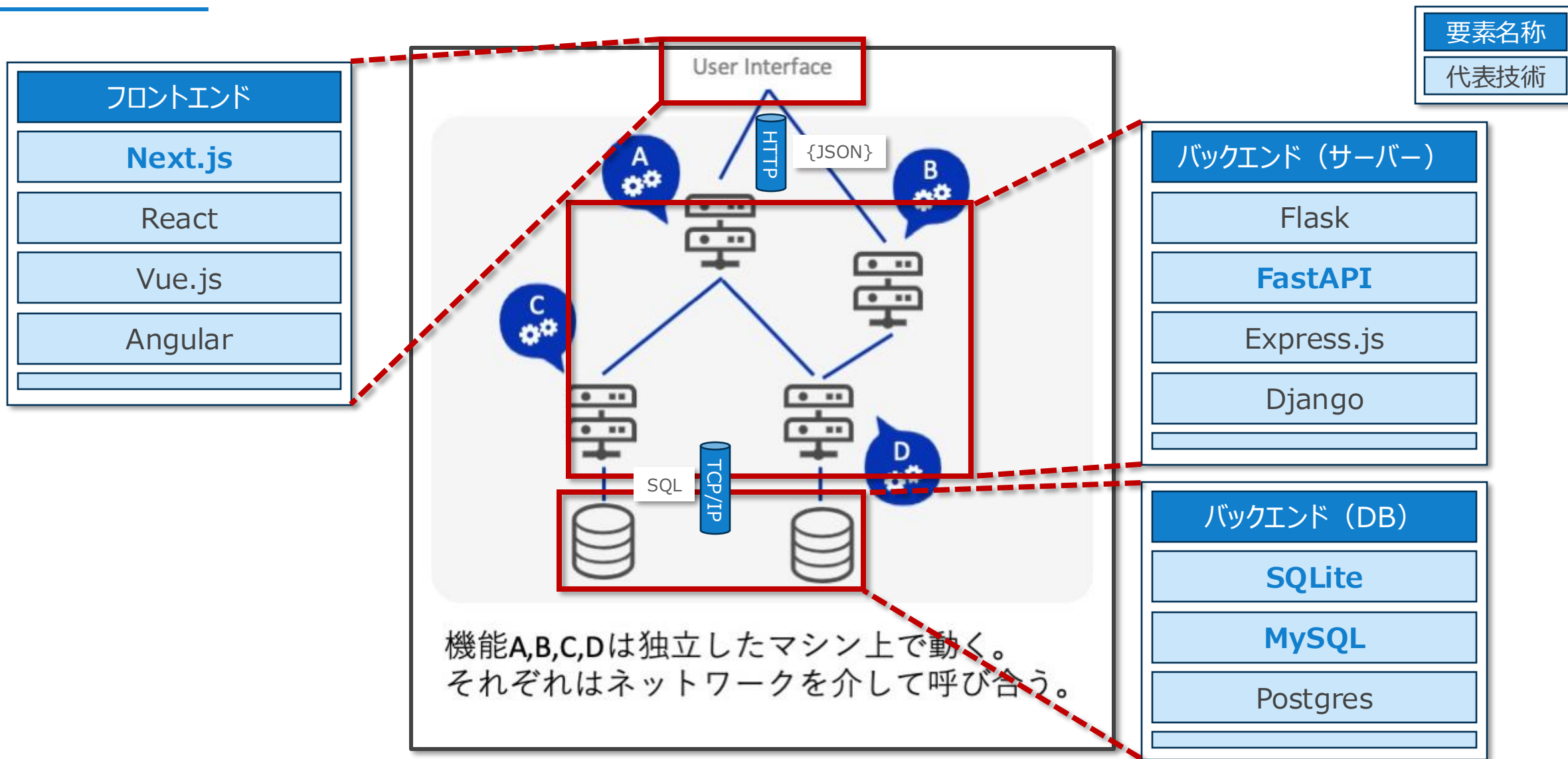
└E0001

└E0002

└E0003

# 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）



# 技術要素：SQLite

軽量で、サーバーレスのデータベース。外部のデータベースサーバー不要で、ローカルストレージに直接埋め込み可能。小～中規模のアプリケーションに適している。設定や管理がほぼ不要でpythonの標準ライブラリに組み込まれており、使いやすい。

## コードサンプル

```
import sqlite3

# データベースに接続する。存在しない場合は新規作成される。
conn = sqlite3.connect('sample.db')

# カーソルオブジェクトを作成する
c = conn.cursor()

# サンプルテーブルの作成。すでに存在する場合は上書きされない。
c.execute('''
CREATE TABLE IF NOT EXISTS sample_table (
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    age INTEGER NOT NULL
)
''')

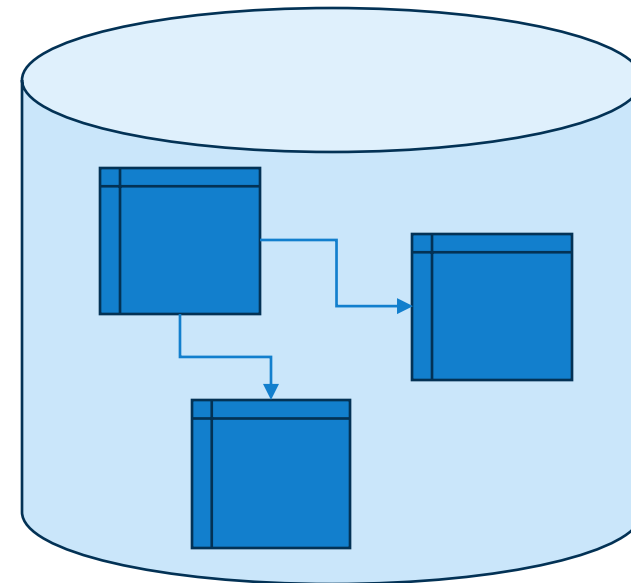
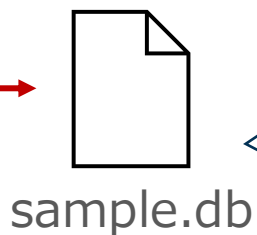
# サンプルデータの挿入
# 既存のデータをクリアするために、テーブルの内容を一度削除します
c.execute('DELETE FROM sample_table')

# サンプルデータを挿入
sample_data = [(1, 'Alice', 30), (2, 'Bob', 25), (3, 'Charlie', 35)]
c.executemany('INSERT INTO sample_table VALUES (?, ?, ?)', sample_data)

# 変更をコミット
conn.commit()

# データベース接続を閉じる
conn.close()
```

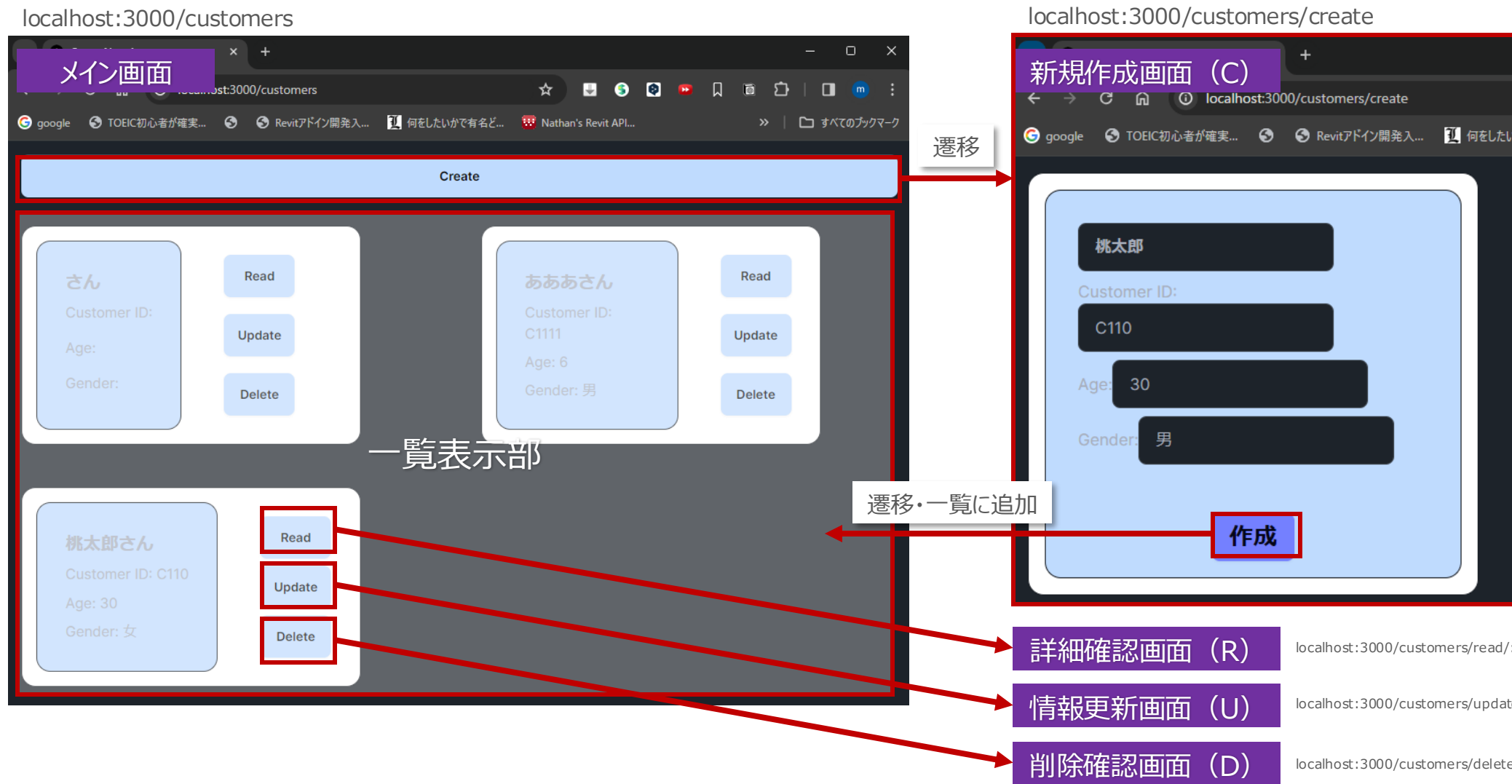
## データベースは.db拡張子のファイルに格納



テーブル構造（メタデータ）および実データすべてを包含するバイナリデータ

# サンプルアプリの解説 - 挙動

顧客マスタのCRUD操作（Create/Read/Update/Delete）を行うことができるシンプルなWebアプリケーション



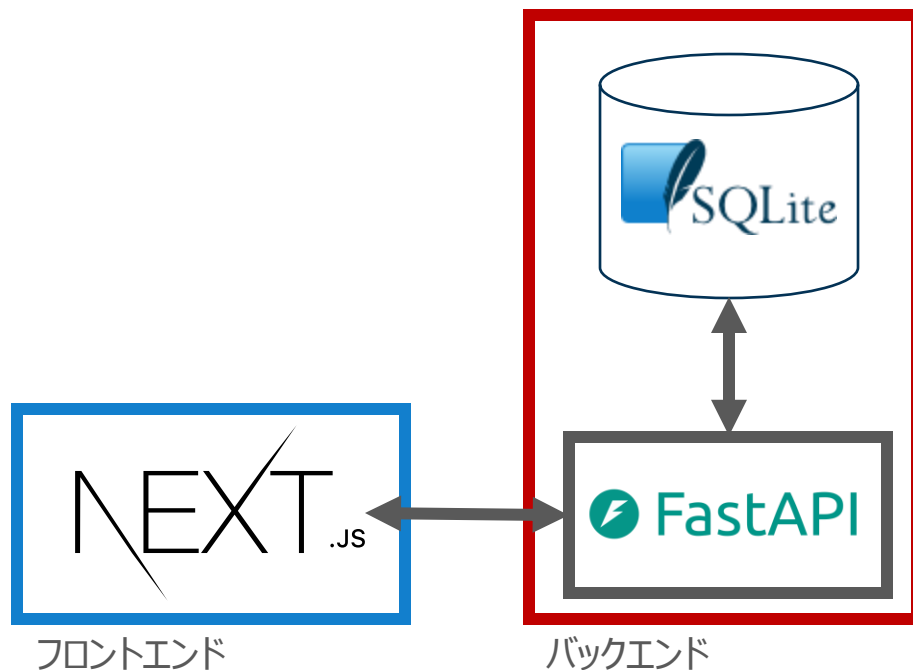
# サンプルアプリの解説 - アーキテクチャ

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+SQLite）のプロジェクトを含む構成になっている。

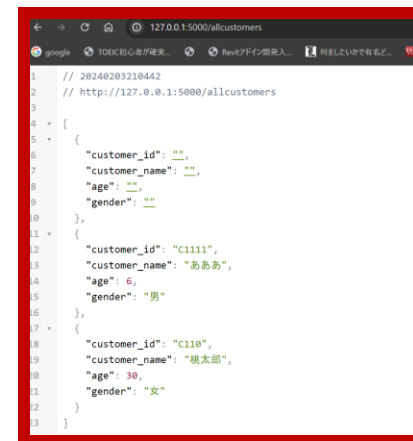
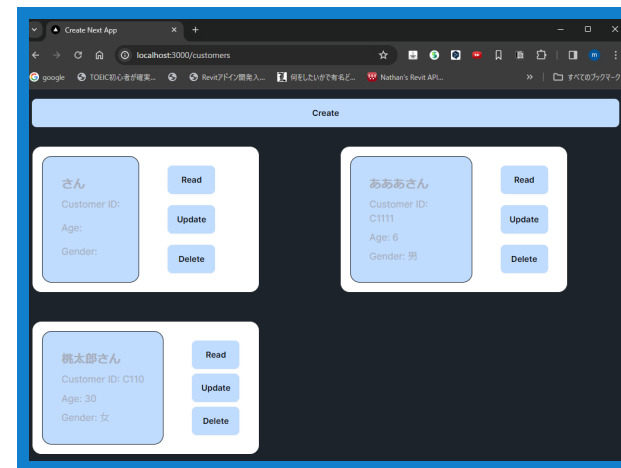
## リポジトリ構成

<ul style="list-style-type: none"><li>▼ backend<ul style="list-style-type: none"><li>&gt; __pycache__</li><li>&gt; db_control</li><li>≡ .cursorignore</li><li>◆ .gitignore</li><li>📄 app.py</li><li>≡ requirements.txt</li></ul></li><li>▼ frontend<ul style="list-style-type: none"><li>&gt; .next</li><li>&gt; node_modules</li><li>▼ src\app<ul style="list-style-type: none"><li>&gt; components</li><li>&gt; customers</li><li>★ favicon.ico</li><li># globals.css</li><li>📄 layout.jsx</li><li>📄 page.jsx</li></ul></li><li>≡ .cursorignore</li><li>🔗 .eslintrc.json</li><li>◆ .gitignore</li><li>{ } jsconfig.json</li><li>JS next.config.js</li><li>{ } package-lock.json</li><li>{ } package.json</li><li>JS postcss.config.js</li><li>📄 README.md</li><li>JS tailwind.config.js</li></ul></li><li>📄 README.md</li></ul>	<p>バックエンド (FastAPI+SQLite)</p> <p>フロントエンド (Next.js)</p>
---	---

## アーキテクチャ

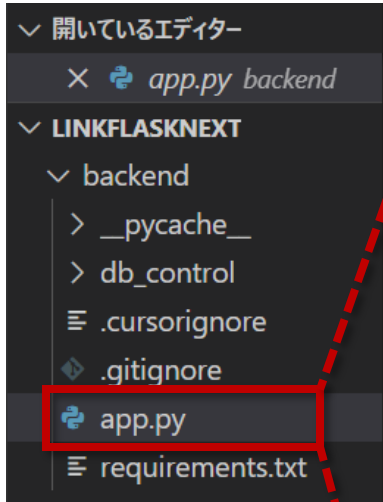


## イメージ



# 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、/customersリソースに対するメソッドにて、POST=create、GET=read、PUT=update、DELETE=deleteの機能が書かれている



```
@app.get("/customers")
def read_one_customer(customer_id: str = Query(...)):
    result = crud.myselect(mymodels.Customers, customer_id)
    if not result:
        raise HTTPException(status_code=404, detail="Customer not found")
    result_obj = json.loads(result)
    return result_obj[0] if result_obj else None
```

←特定idのデータを返す機能 (read)

```
@app.post("/customers")
def create_customer(customer: Customer):
    values = customer.dict()
    tmp = crud.myinsert(mymodels.Customers, values)
    result = crud.myselect(mymodels.Customers, values.get("customer_id"))

    if result:
        result_obj = json.loads(result)
        return result_obj if result_obj else None
    return None
```

←JSONデータの内容をinsertする機能 (create)

```
@app.put("/customers")
def update_customer(customer: Customer):
    values = customer.dict()
    values_original = values.copy()
    tmp = crud.myupdate(mymodels.Customers, values)
    result = crud.myselect(mymodels.Customers, values_original.get("customer_id"))
    if not result:
        raise HTTPException(status_code=404, detail="Customer not found")
    result_obj = json.loads(result)
    return result_obj[0] if result_obj else None
```

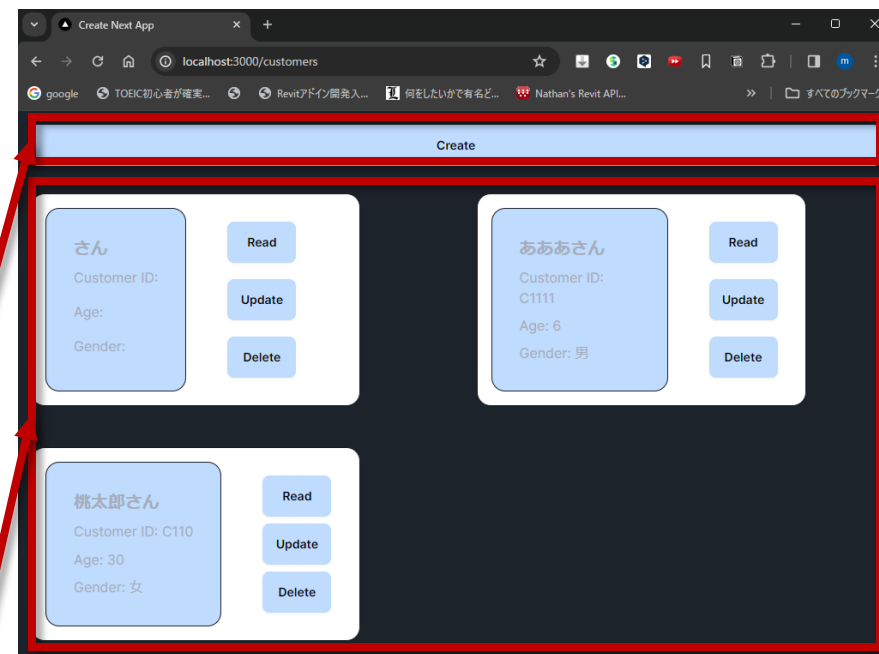
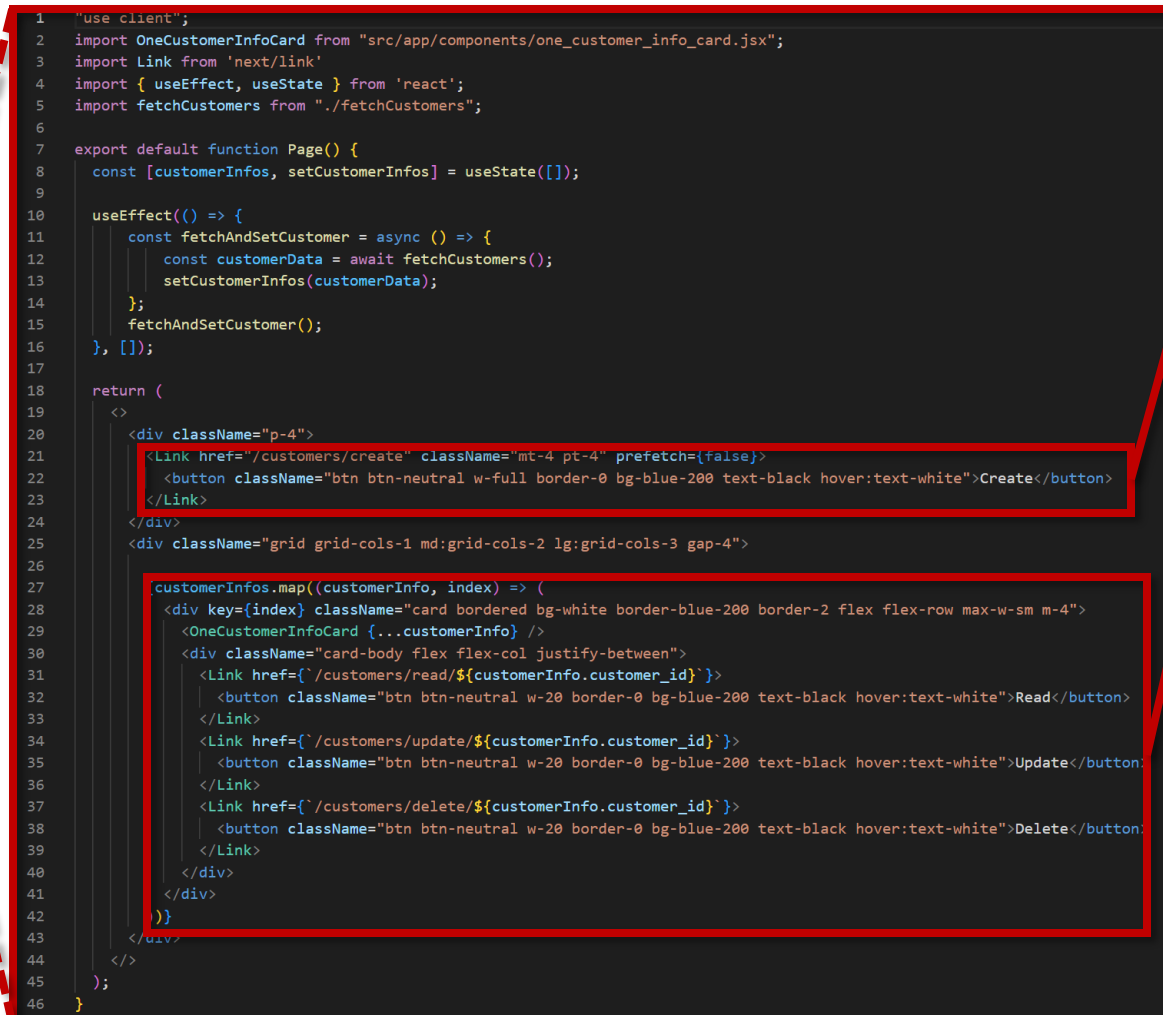
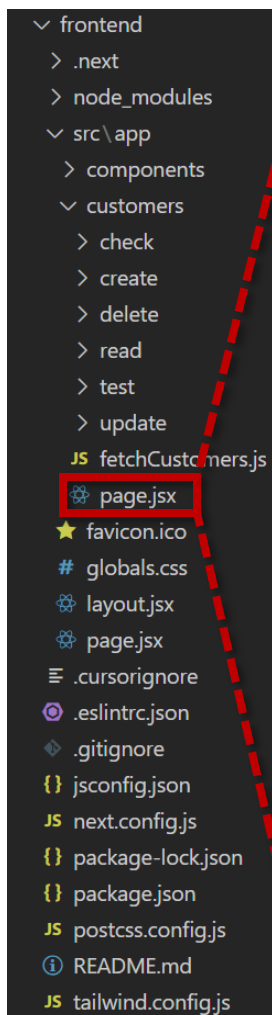
←特定idのデータを上書きする機能 (update)

```
@app.delete("/customers")
def delete_customer(customer_id: str = Query(...)):
    result = crud.mydelete(mymodels.Customers, customer_id)
    if not result:
        raise HTTPException(status_code=404, detail="Customer not found")
    return {"customer_id": customer_id, "status": "deleted"}
```

←特定idのデータを削除する機能 (delete)

# サンプルアプリの解説 - Next.js

ルート直下はダミーページであり、/customers以下に機能実装されている。例としてcustomers/page.jsxにて/customersページの主要コンポーネントが記述されている。

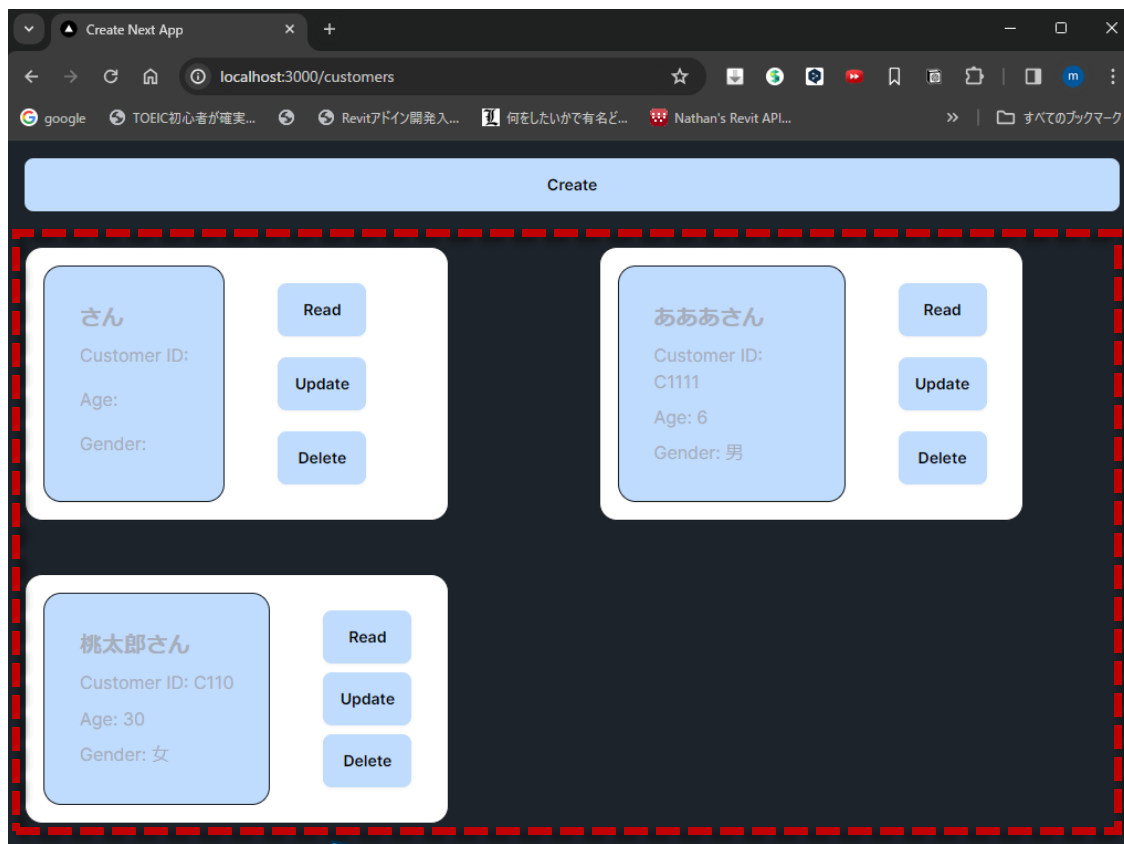




## 演習：エラーID-E0001

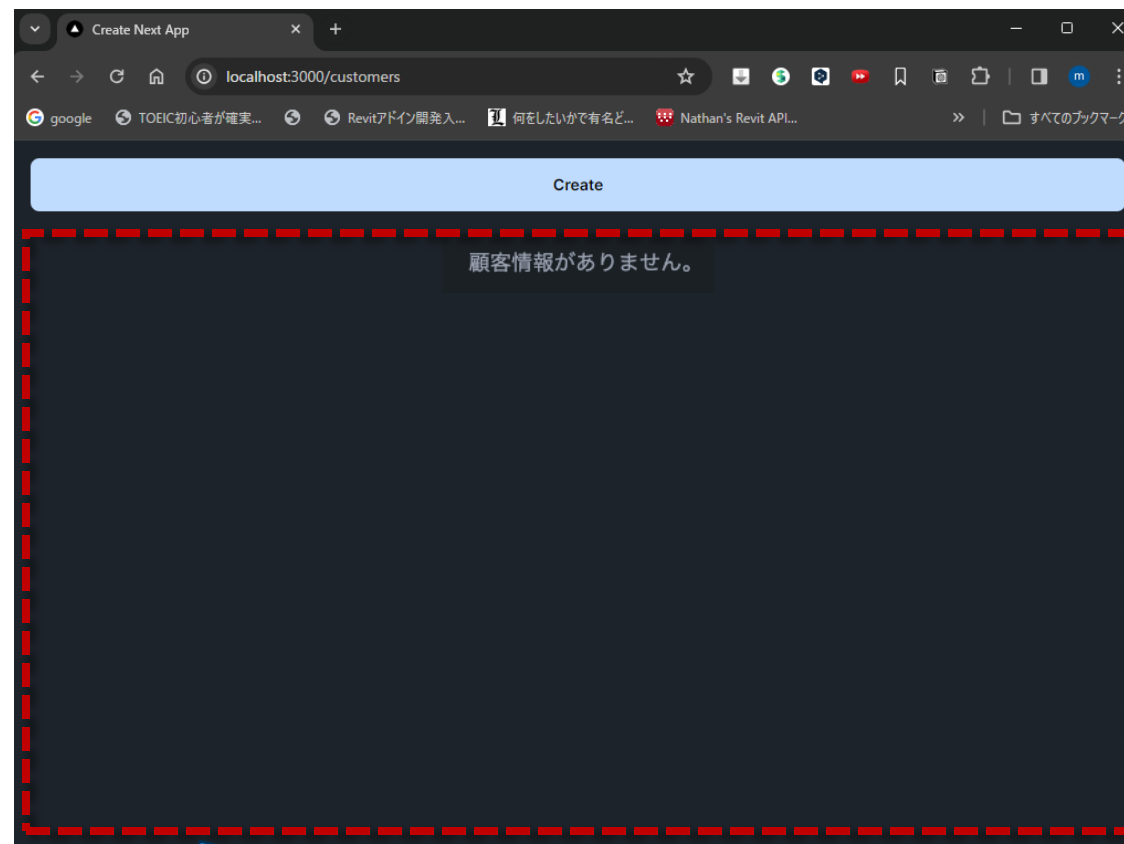
本来CREATEボタンの下に表示されるはずのデータ（顧客）一覧が表示されない。表示されるようにしてください。

### 期待する表示



1レコードが1カードに対応して一覧表示されることが期待される挙動

### 現状



しかし、一覧にカードが表示されない

## 演習：エラーID-E0002

任意のデータの「UPDATE」ボタンをクリックし、遷移先の画面でデータを修正したのちに「更新」をクリックすればデータが更新されるはずだがされない。直してください。

### 生じている事象

The screenshot shows a web application interface. On the left, there is a list of customers. Each customer entry has a 'Read', 'Update', and 'Delete' button. The 'Update' button for the customer 'あああさん' (Aaaa-san) is highlighted with a red box. A red dashed line connects this button to a larger, detailed view of the customer on the right. In this detailed view, the customer's information is displayed, and there is a red box around the '更新' (Update) button. Below the detailed view, there is a red error message box that says '3 errors' with a close button. A blue speech bubble points to the error message box with the text: '任意の修正後「更新」ボタンをクリックすればデータが修正されるはずだが、errorが発生' (After any correction, clicking the 'Update' button should update the data, but an error occurs).

任意の修正後「更新」ボタンをクリックすればデータが修正されるはずだが、errorが発生

## 演習：エラーID-E0003

idが空白（null）のデータを作成できてしまい、当該データはフロントエンドの仕様上CRUD操作ができなくなってしまう。このようなデータが発生してしまわないような恒久対策を講じてください。

生じている事象

The screenshot shows a web browser at localhost:3000/customers. The page has a blue header with the text '生じている事象'. Below the header is a 'Create' button. The main content area displays a list of customer cards. The first card, highlighted with a red dashed box, has the name 'さん', a blank 'Customer ID' field, and buttons for 'Read', 'Update', and 'Delete'. The second card has the name 'あああさん', 'Customer ID: C111', and the same buttons. The third card has the name '桃太郎さん', 'Customer ID: C110', 'Age: 30', 'Gender: 女', and the same buttons. A blue callout box points to the first card with the following text:

ID空欄のままデータが登録できてしまう。しかし、UpdateやDeleteの個別ページはidがないとアクセスできないため、修正も削除もできないゴミデータが作成されてしまう状態にある。  
→なんらかの恒久対応が必要