

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

Output

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

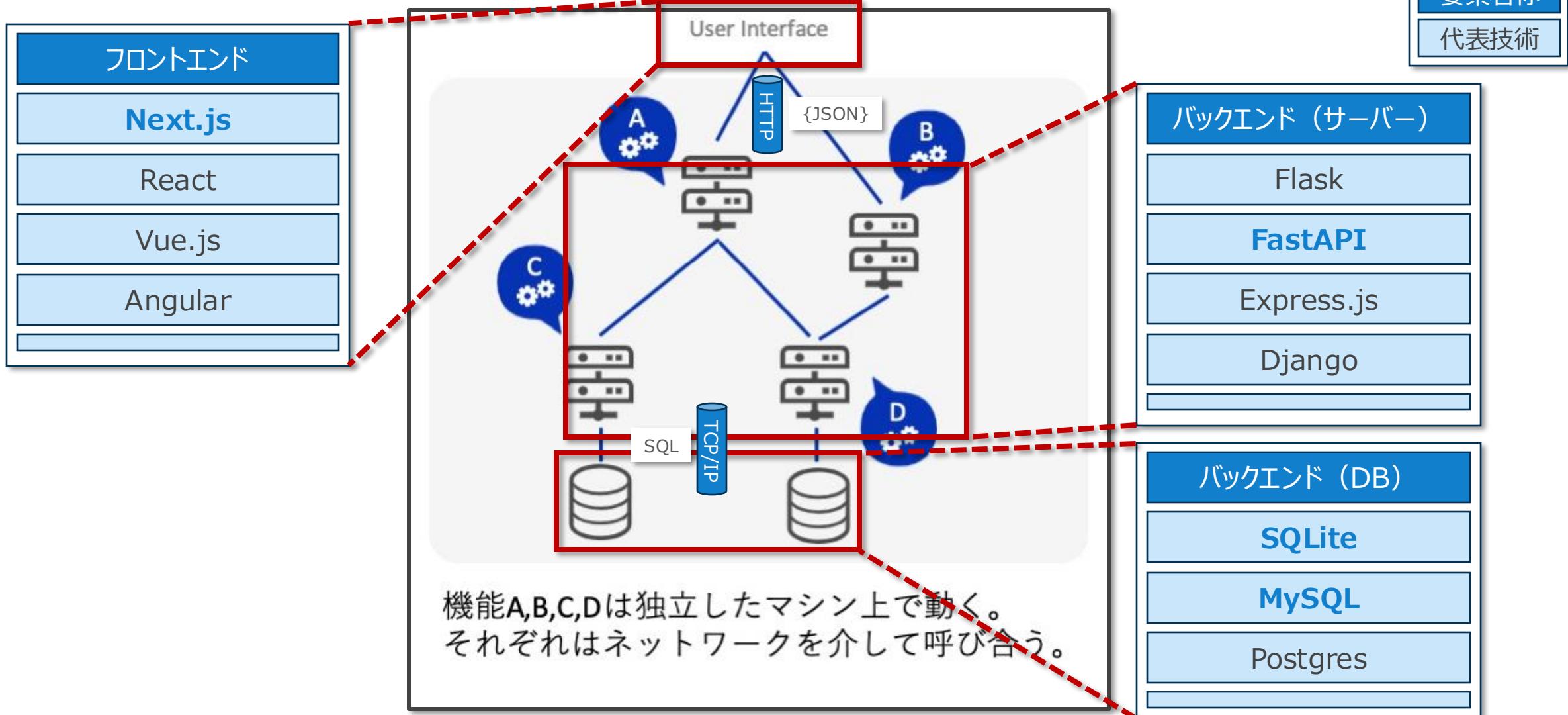
 └ E0001

 └ 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
)
''')

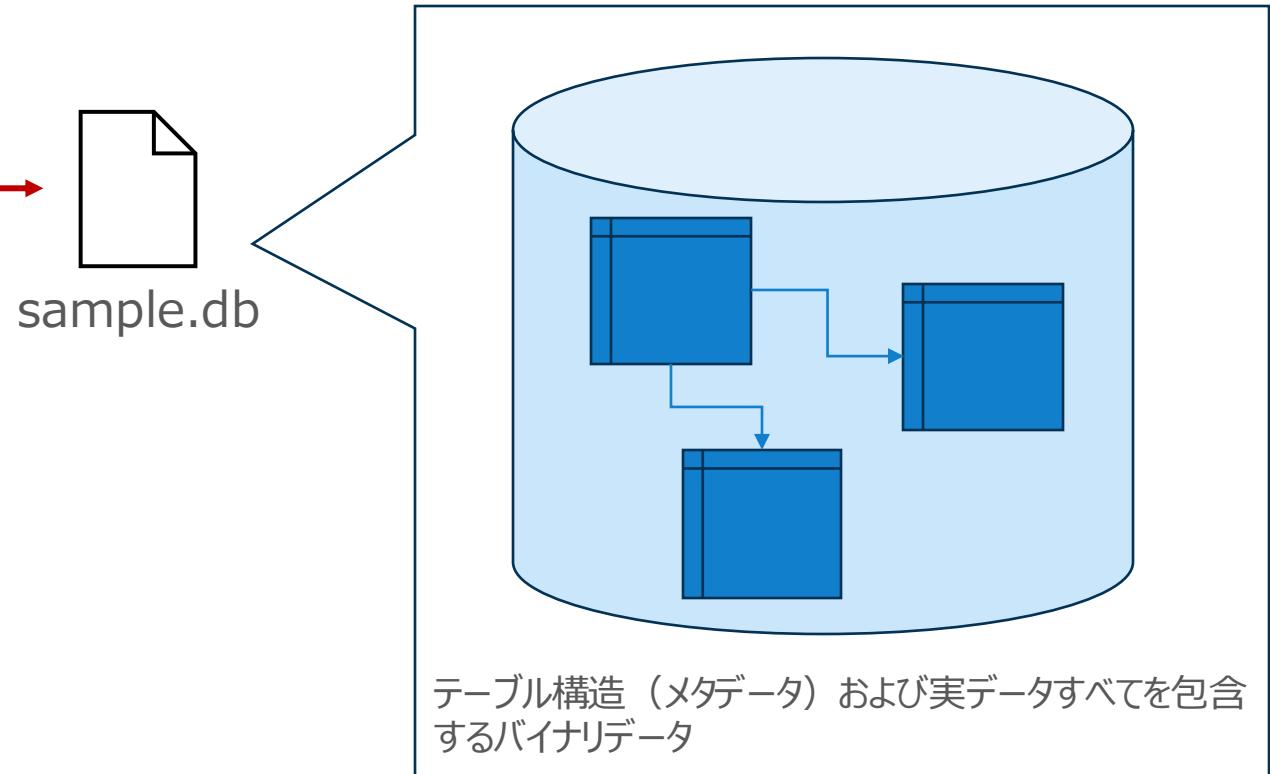
# サンプルデータの挿入
# 既存のデータをクリアするために、テーブルの内容を一度削除します
c.execute('DELETE FROM sample_table')

# 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()
```

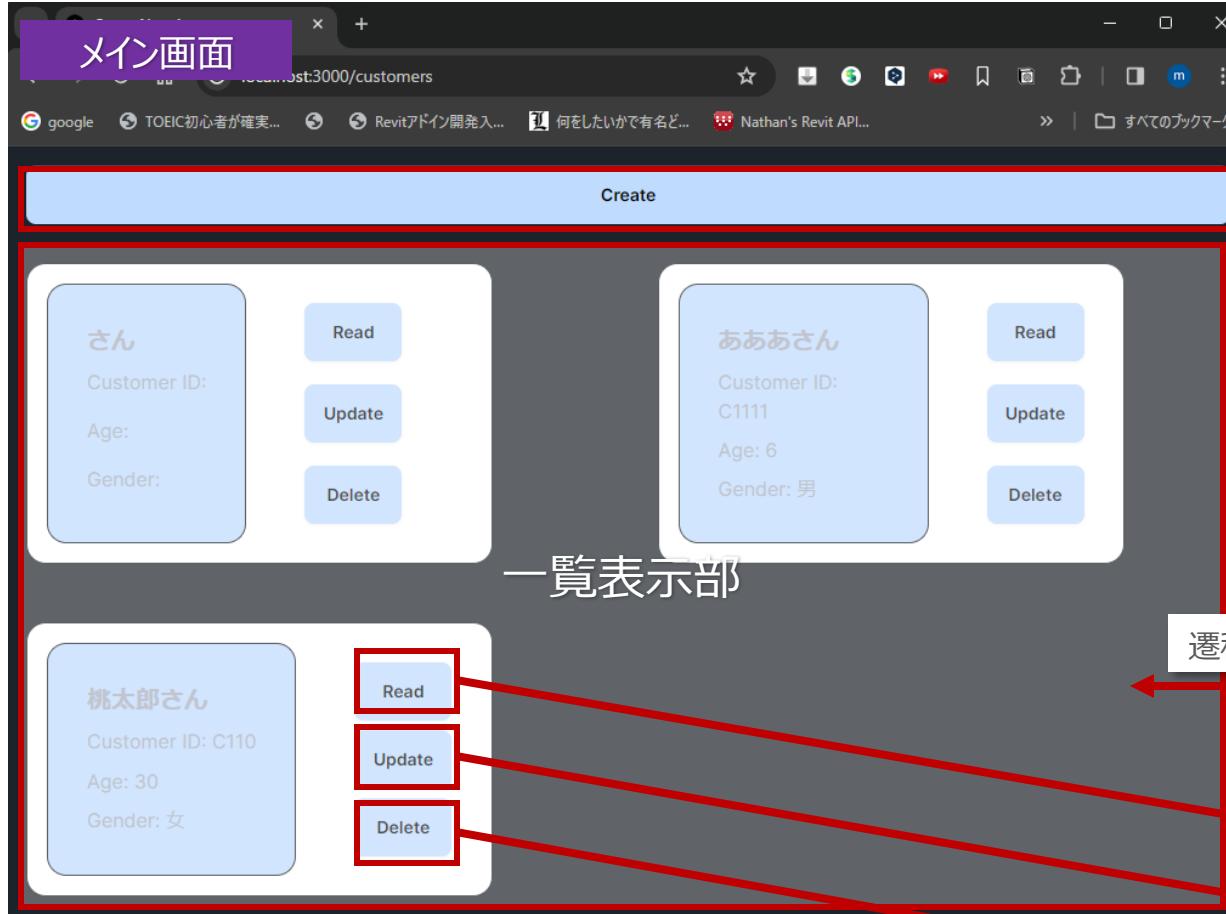
データベースは.db拡張子のファイルに格納



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

localhost:3000/customers



localhost:3000/customers/create



詳細確認画面 (R)

localhost:3000/customers/read/:id

情報更新画面 (U)

localhost:3000/customers/update/:id

削除確認画面 (D)

localhost:3000/customers/delete/:id

(省略。
上とほぼ同じ)

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。

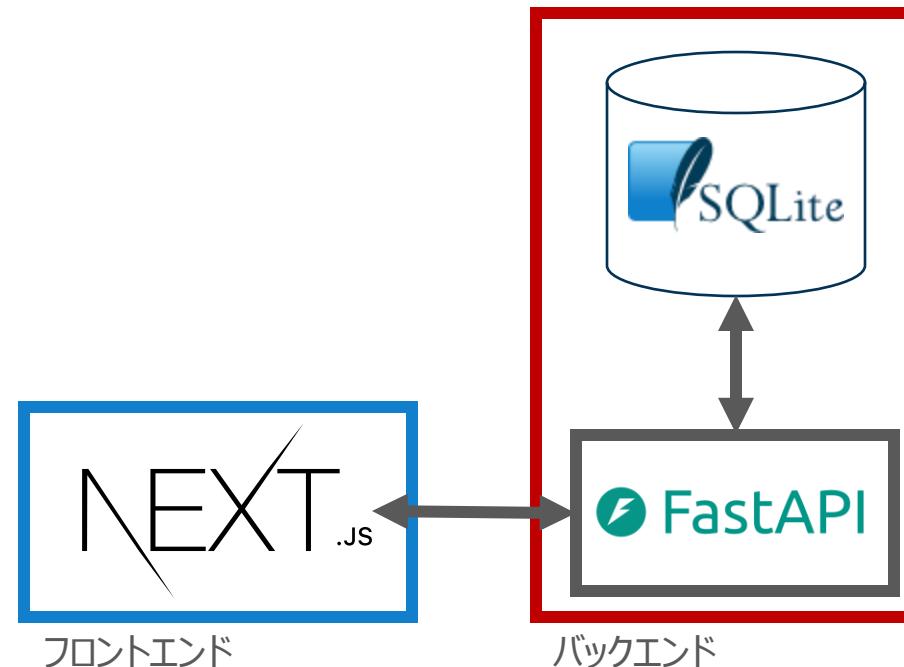
リポジトリ構成

```
backend  
  > __pycache__  
  > db_control  
  ≡ .cursorignore  
  ♦ .gitignore  
  📄 app.py  
  ≡ requirements.txt  
  
frontend  
  > .next  
  > node_modules  
  ✓ src/app  
    > components  
    > customers  
    ★ favicon.ico  
    # globals.css  
    ✎ layout.jsx  
    ✎ page.jsx  
    ≡ .cursorignore  
    ⚙ .eslintrc.json  
    ♦ .gitignore  
    { jsonconfig.json  
    JS next.config.js  
    { package-lock.json  
    { package.json  
    JS postcss.config.js  
    ⓘ README.md  
    JS tailwind.config.js  
  
ⓘ README.md
```

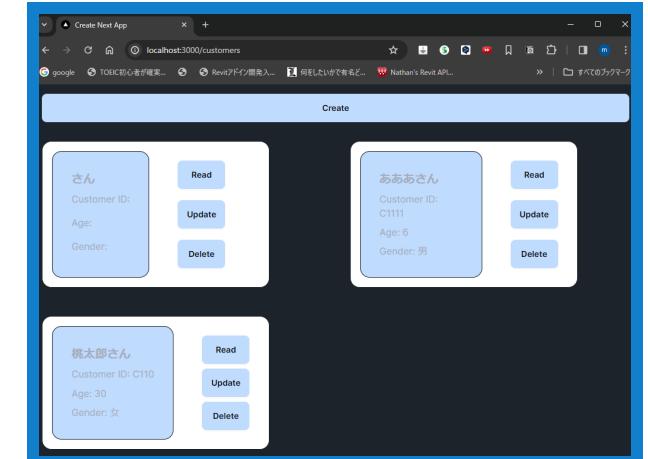
バックエンド
(FastAPI+SQLite)

フロントエンド
(Next.js)

アーキテクチャ



イメージ

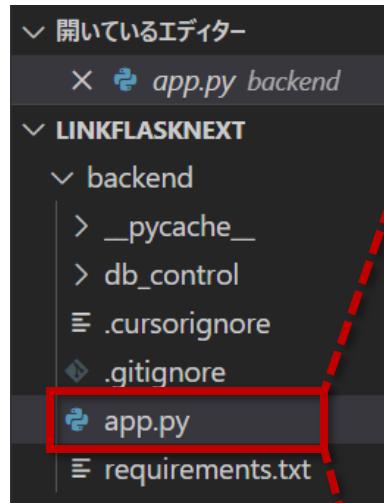


A screenshot of a terminal window titled "allcustomers". The URL is "http://127.0.0.1:5000/allcustomers". The output shows a JSON array of customer data:

```
[{"customer_id": "", "customer_name": "", "age": "", "gender": ""}, {"customer_id": "C1111", "customer_name": "あああさん", "age": 6, "gender": "男"}, {"customer_id": "C110", "customer_name": "桃太郎さん", "age": 30, "gender": "女"}]
```

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

@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

@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

@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のデータを返す機能 (read)

←JSONデータの内容をinsertする機能 (create)

←特定idのデータを上書きする機能 (update)

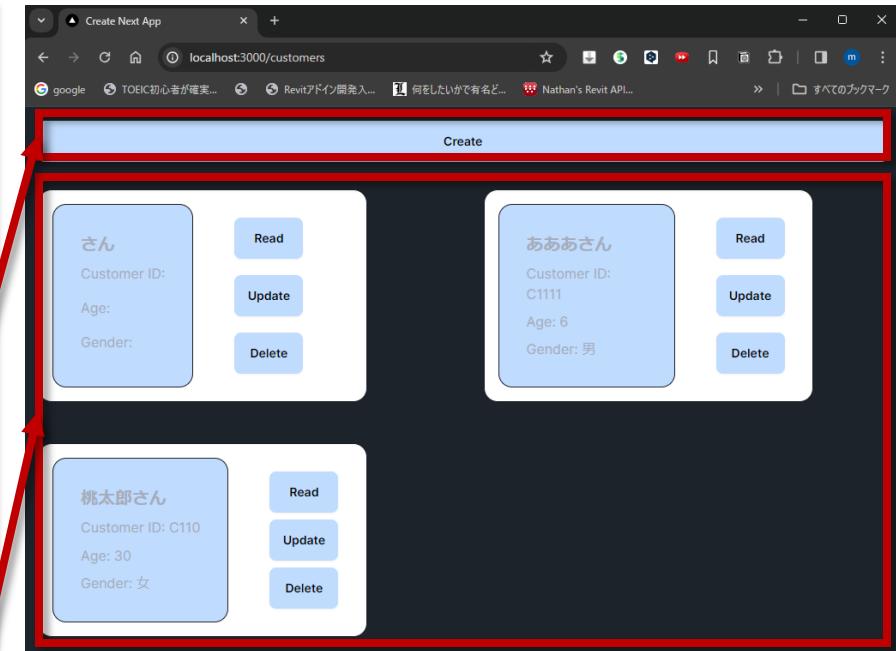
←特定idのデータを削除する機能 (delete)

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。

- ✓ frontend
 - > .next
 - > node_modules
 - ✓ src\app
 - > components
 - ✓ customers
 - > check
 - > create
 - > delete
 - > read
 - > test
 - > update
 - JS fetchCustomers.js
 - JS page.jsx
 - ★ favicon.ico
 - # globals.css
 - ⌘ layout.jsx
 - ⌘ page.jsx
 - ≡ .cursorignore
 - ⌚ .eslintrc.json
 - ❖ .gitignore
 - { } jsonconfig.json
 - JS next.config.js
 - { } package-lock.json
 - { } package.json
 - JS postcss.config.js
 - ⓘ README.md
 - JS tailwind.config.js

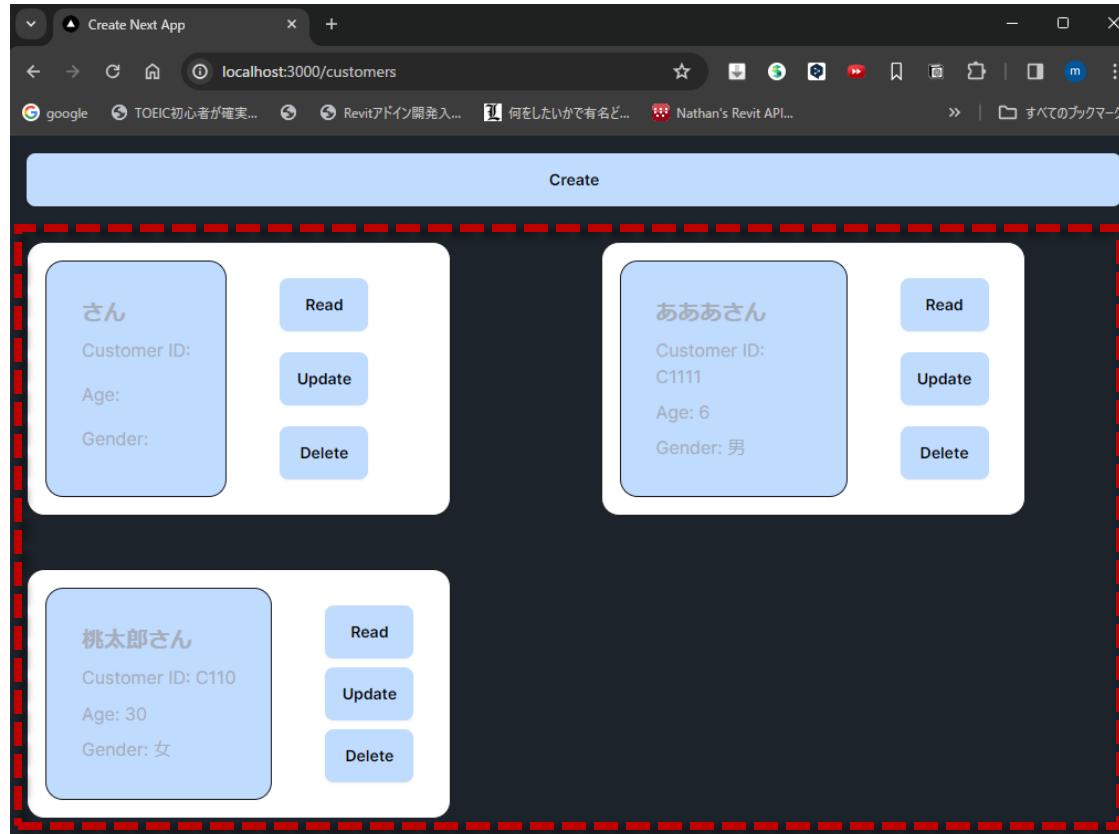
```
1 "use client";
2 import OneCustomerInfoCard from "src/app/components/one_customer_info_card.jsx";
3 import Link from 'next/link'
4 import { useEffect, useState } from 'react';
5 import fetchCustomers from "./fetchCustomers";
6
7 export default function Page() {
8   const [customerInfos, setCustomerInfos] = useState([]);
9
10  useEffect(() => {
11    const fetchAndSetCustomer = async () => {
12      const customerData = await fetchCustomers();
13      setCustomerInfos(customerData);
14    };
15    fetchAndSetCustomer();
16  }, []);
17
18  return (
19    <>
20      <div className="p-4">
21        <Link href="/customers/create" className="mt-4 pt-4" prefetch={false}>
22          <button className="btn btn-neutral w-full border-0 bg-blue-200 text-black hover:text-white">Create</button>
23        </Link>
24      </div>
25      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
26
27        {customerInfos.map((customerInfo, index) => (
28          <div key={index} className="card bordered bg-white border-blue-200 border-2 flex flex-row max-w-sm m-4">
29            <OneCustomerInfoCard {...customerInfo} />
30            <div className="card-body flex flex-col justify-between">
31              <Link href={`/customers/read/${customerInfo.customer_id}`}>
32                <button className="btn btn-neutral w-20 border-0 bg-blue-200 text-black hover:text-white">Read</button>
33              </Link>
34              <Link href={`/customers/update/${customerInfo.customer_id}`}>
35                <button className="btn btn-neutral w-20 border-0 bg-blue-200 text-black hover:text-white">Update</button>
36              </Link>
37              <Link href={`/customers/delete/${customerInfo.customer_id}`}>
38                <button className="btn btn-neutral w-20 border-0 bg-blue-200 text-black hover:text-white">Delete</button>
39              </Link>
40            </div>
41          </div>
42        ))}
43      </div>
44    </>
45  );
46}
```



演習：エラーID-E0001

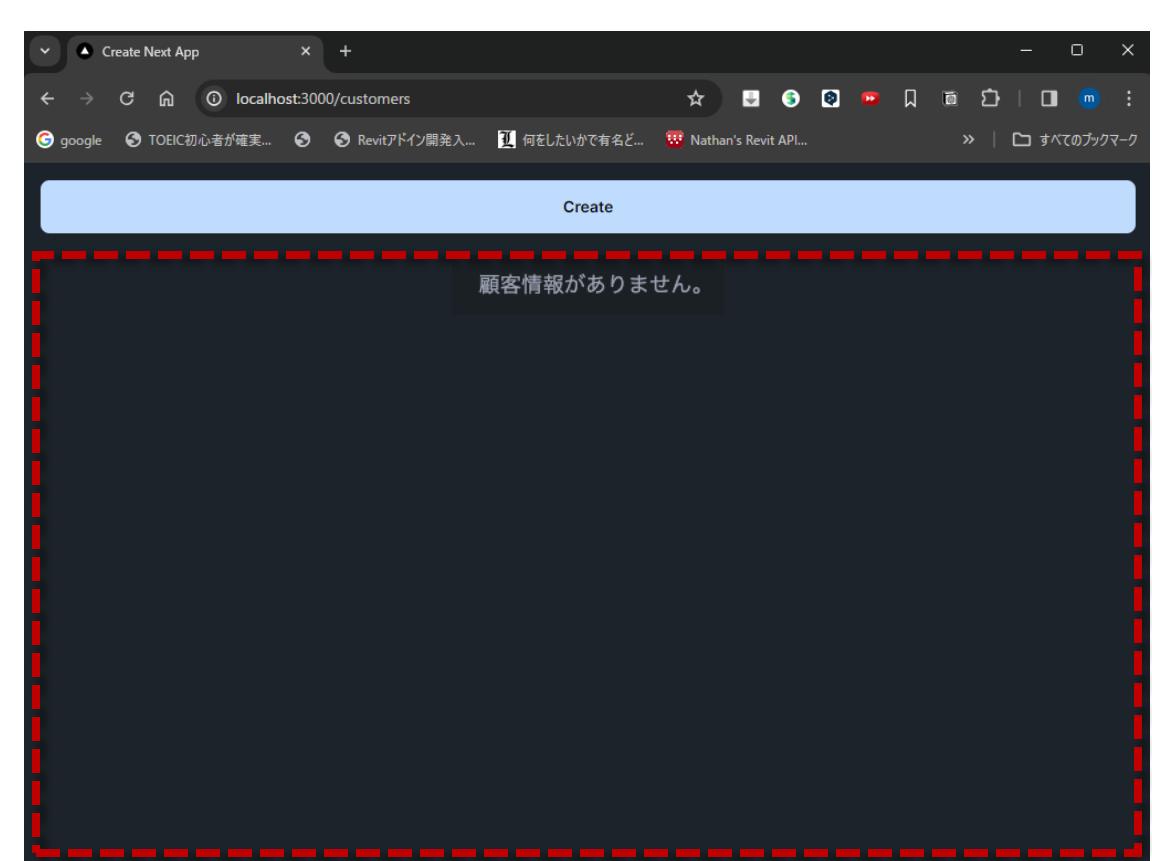
本来CREATEボタンの下に表示されるはずのデータ（顧客）一覧が表示されない。表示されるようにしてください。

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

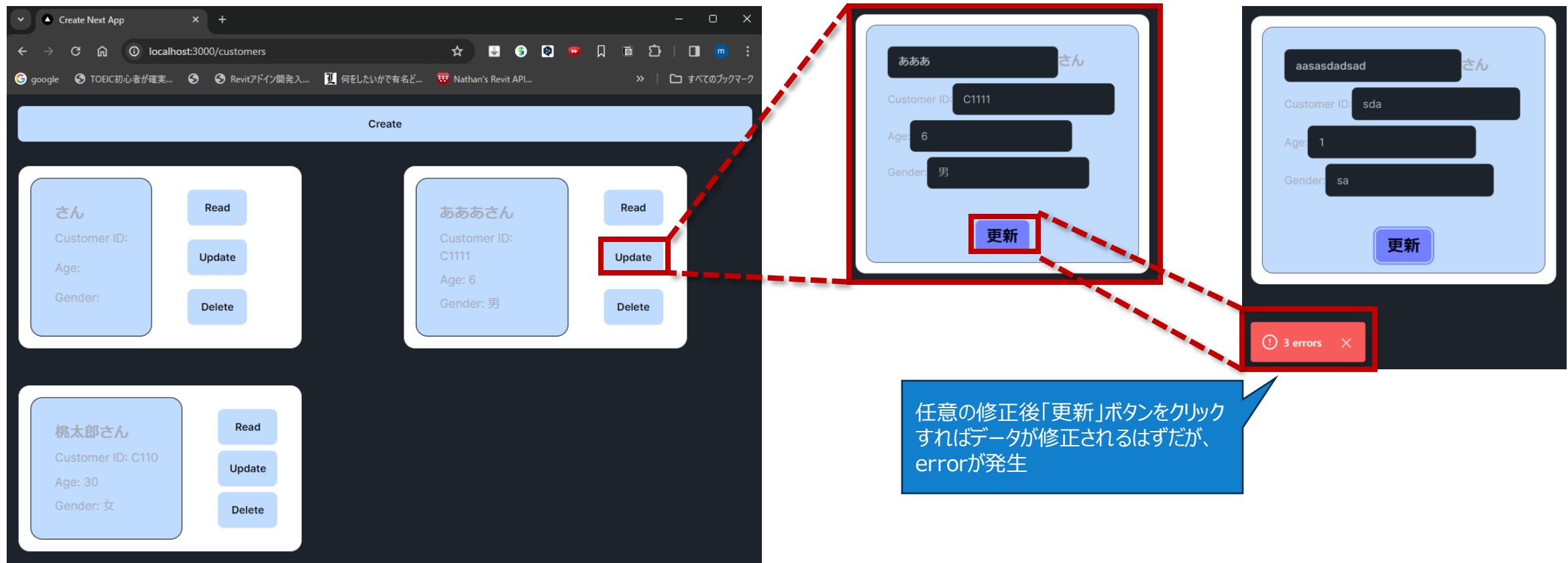


しかし、一覧にカードが表示されない

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



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。

