

PBL 第 2 回レポート

22X3057 菅原颯太

1. 目的

第 13 回までの授業において、現代の Web 開発における基本的な技術を理解することを目的とし、Docker を用いた仮想環境の構築、PostgreSQL によるデータベース管理、FastAPI を用いたバックエンド開発、Next.js と Chakra-UI を用いたフロントエンド開発、Vercel へのデプロイという一連の開発の流れを通して知識と技術を学ぶ。

2. 方法

2-1 Docker でデータベースの作成、接続、操作

Docker Engine では上手くいかなかったため、Docker desktop を推奨設定でインストールし、ターミナルでバージョンを確認した。

```
$ docker version
```

次に、pbl ディレクトリの中に、docker-compose.yaml ファイルを作成し、以下の内容を記述した。

```
version: '3'

services:
  db:
    image: postgres:latest
    ports:
      - 5432:5432
    volumes:
      - db-store:/var/lib/postgresql/data
    environment:
      POSTGRES_USER: 'user'
      POSTGRES_PASSWORD: 'postgres'
volumes:
  db-store:
```

次に、

```
$ docker compose up
```

で Docker のコンテナを起動した。

次に、PostgreSQL の操作をするために

```
$ pip install pgcli
$ pip install psycopg[binary]
```

を実行しインストールした。

次に、

```
$ pgcli -d postgres -h localhost -p 5432 -U user
```

を実行してデータベースに接続した。

データベースの操作について、最初に

```
> create table items(id int, name text)
```

でテーブルを作成した。次に、

```
> select * from items
```

でテーブルの内容を確認した。以降テーブルに変更を加えた際は select を実行し確認した。次に、

```
> alter table items add price int default 0
```

で、テーブルに新しいカラムを追加した。次に、

```
> insert into items (id,name,price) values (1,'item1',1000)
```

でテーブルにデータを追加した。次に、

```
> insert into items (id,name,price) values  
(2,'item2',2000),(3,'item3',3000)
```

でテーブルにデータを 2 つ同時に追加した。次に、

```
> select name from items
```

で name 列のデータのみを表示した。次に、

```
> update items set price=100 where id=1
```

で id が 1 のデータの price を 100 に更新した。次に、

```
> alter table items add primary key(id)
```

で items テーブルの id データに主キー制約を設定した。次に、

```
> insert into items (id,name,price) values (1,'item4',4000)
```

を実行した。次に、

```
> create table orders (order_id int primary key, order_date date, item_id  
int, foreign key(item_id) references items(id))
```

で items テーブルの id 列を参照する外部キー制約を設定した orders テーブルを作成した。次に、

```
> insert into orders (order_id, order_date, item_id) values (1,'2024-06-  
18',1)
```

```
> insert into orders (order_id, order_date, item_id) values (2,'2024-06-  
18',4)
```

をそれぞれ実行した。最後に、

```
> select orders.order_id, orders.order_date, items.id as item_id,  
items.name, items.price from orders inner join items on orders.item_id =  
items.id
```

で items テーブルと orders テーブルを結合して表示した。

2-2 FastAPI にデータベースを組み込む

最初に、2-1 と同様の方法でデータベースにアクセスし、

```
> create table books (id integer, title text not null, price integer not null)
> insert into books (id, title, price) values (1,'book1',1200)
> insert into books (id, title, price) values (2,'book2',1000)
> insert into books (id, title, price) values (3,'book3',1500)
```

で books テーブルを作成しデータを挿入した。次に、ターミナルで

```
$ pip install SQLAlchemy
```

を実行し SQLAlchemy をインストールした。次に、pbl ディレクトリに models.py ファイルを作成し、以下の内容を記述した。

```
from sqlalchemy import Column, Integer, String
from sqlalchemy.orm import DeclarativeBase

class Base(DeclarativeBase):
    pass

class Book(Base):
    __tablename__ = "books"

    id = Column('id', Integer, primary_key=True)
    title = Column('title', String, nullable=False)
    price = Column('price', Integer, nullable=False)
```

次に、database.py ファイルを作成し、以下の内容を記述した。

```
from sqlalchemy import create_engine
from sqlalchemy.orm import Session
import models

# 接続先 DB の設定
DATABASE = 'postgresql+psycopg://user:postgres@localhost:5432/postgres'

# Engine の作成
Engine = create_engine(
    DATABASE,
    echo=True
)
```

```

# Session の作成
session = Session(
    autocommit = False,
    autoflush = True,
    bind = Engine
)

def read_books():
    return session.query(models.Book).all()

```

次に、main.py ファイルを以下のように記述した。

```

from typing import List
from typing import Union
from fastapi import FastAPI
from pydantic import BaseModel
from database import read_books

app = FastAPI()

class Item(BaseModel):
    name: str
    price: float
    is_offer: Union[bool, None] = None

class BookRead(BaseModel):
    id: int
    title: str
    price: int

@app.get("/")
def read_root():
    return {"Hello": "World"}

@app.get("/books", response_model=List[BookRead])
def get_books():

```

```

books = read_books()
return books

@app.get("/items/{item_id}")
def read_item(item_id: int, q: Union[str, None] = None):
    return {"item_id": item_id, "q": q}

@app.put("/items/{item_id}")
def update_item(item_id: int, item: Item):
    return {"item_name": item.name, "item_id": item_id}

```

次に、pbl ディレクトリで

```
$ fastapi dev main.py
```

を実行してサーバを起動し、ブラウザで <http://127.0.0.1:8000/books> にアクセスした。

2-3 Web ページの作成

最初に、ターミナルで

```
$ mise use -q node@latest
```

を実行し、node.js をインストールした。次に、

```
$ node -v
```

でバージョンを確認した。次に、

```
$ npx create-next-app@latest
```

で Next.js のプロジェクトを作成した。ここで、プロジェクト名は `emon-app` にし、初期設定は `yynynn` にした。次に、

```
$ npm install -g yarn
```

で yarn をインストールした。次に、`emon-app` ディレクトリに移動し

```
$ yarn dev
```

を実行してブラウザで <http://localhost:3000/> にアクセスした。次に、`emon-app/src/pages` に `hello.tsx` を作成し、以下のように記述した。

```

import { Inter } from "next/font/google";

const inter = Inter({ subsets: ["latin"] });

export default function hello() {
  return (
    <>
    hello

```

```

    </>
  );
}

```

その後、ブラウザで <http://localhost:3000/hello> にアクセスした。次に、emon-app ディレクトリで

```
$ npm i @chakra-ui/react @emotion/react @emotion/styled framer-motion
```

を実行し、chakra-UI をインストールした。次に、emon-app/src/pages の _app.tsx ファイルを以下のように記述した。

```

import type { AppProps } from "next/app";
import { ChakraProvider } from "@chakra-ui/react";

export default function App({ Component, pageProps }: AppProps) {
  return (
    <ChakraProvider>
      <Component {...pageProps} />
    </ChakraProvider>
  );
}

```

次に、hello.tsx を以下のように記述した。

```

import { Inter } from "next/font/google";
import { Button } from "@chakra-ui/react";

const inter = Inter({ subsets: ["latin"] });

export default function hello() {
  return (
    <Button>
      button
    </Button>
  );
}

```

その後、ブラウザで <http://localhost:3000/hello> にアクセスした。

次に、chakra-UI を使って自分のポートフォリオサイトを作成した。最初に、emon-app/public に img ディレクトリと pdf ディレクトリを作成し、img ディレクトリ内に skz.jpg (プロフィール画像) を入れ、pdf ディレクトリ内に第 1 回レポート.pdf と第 2 回レポート.pdf を入れた。次に、emon-app/src/pages の _app.tsx を以下のように記述し

た。

```
import type { AppProps } from "next/app";
import { ChakraProvider } from "@chakra-ui/react";
import Head from "next/head";

export default function App({ Component, pageProps }: AppProps) {
  return (
    <>
      <Head>
        <link rel="icon" href="img/skz.jpg" type="image/jpg" />
      </Head>
      <ChakraProvider>
        <Component {...pageProps} />
      </ChakraProvider>
    </>
  );
}
```

次に、index.tsx を以下のように記述した。

```
import { Inter } from "next/font/google";
import { Box, Flex, Heading, Text, Button, Image, VStack, Link } from
"@chakra-ui/react";

const inter = Inter({ subsets: ["latin"] });

export default function Hello() {
  return (
    <Flex direction="column" align="center" p={8}>
      { /* ヘッダー */ }
      <Box mb={8} textAlign="center">
        <Heading as="h1" size="2xl">emon</Heading>
        <Text fontSize="lg" color="gray.600">Hosei university student /
iyatomi lab / M3</Text>
      </Box>

      { /* プロフィールセクション */ }
```



```

    <Box mb={8} display="flex" flexDirection="column"
alignItems="center">
    <Image
      borderRadius="full"
      boxSize="150px"
      src="/img/skz.jpg"
      alt="Profile Image"
      mb={4}
    />
    <Text fontSize="md" color="gray.600">法政大学理工学部応用情報工学科 3
年生として勉強中</Text>
  </Box>

  { /* ワークセクション */ }
  <VStack spacing={4} align="stretch" w="full" maxW="md">
    <Box p={4} shadow="md" borderWidth="1px" borderRadius="md">
      <Heading fontSize="xl">PBL report 1</Heading>
      <Text mt={4}>第 1 回 PBL レポート<code>"Linux, GCE, mise, Git/Github,
FastAPI</code></Text>
      <Button mt={4} colorScheme="teal" onClick={() =>
window.open('/pdf/第 1 回レポート.pdf', '_blank')}>
        View
      </Button>
    </Box>
    <Box p={4} shadow="md" borderWidth="1px" borderRadius="md">
      <Heading fontSize="xl">PBL report 2</Heading>
      <Text mt={4}>第 2 回 PBL レポート<code>"Docker, PostgreSQL, Next.js,
Chakra-UI, Vercel</code></Text>
      <Button mt={4} colorScheme="teal" onClick={() =>
window.open('/pdf/第 2 回レポート.pdf', '_blank')}>
        View
      </Button>
    </Box>
  </VStack>

  { /* コンタクトセクション */ }

```

```

    <Box mt={8} textAlign="center">
      <Heading as="h3" size="lg">Contact</Heading>
      <VStack mt={4}>
        <Link href="mailto:sugakko0507@gmail.com"
color="teal.500">sugakko0507@gmail.com</Link>
      </VStack>
    </Box>

    { /* フッター */ }
    <Flex mt={8} p={4} w="full" justifyContent="center"
borderTopWidth="1px">
      <Text fontSize="sm" color="gray.600">
        © 2024 emon.
      </Text>
    </Flex>
  </Flex>
);
}

```

2-4 Vercel へのデプロイ

次に、作成したポートフォリオサイトを Vercel で公開した。最初に、emon-app ディレクトリの内容を

```

$ git add .
$ git commit -m "first commit"
$ git push origin main

```

で Github の emon-project リポジトリにプッシュした。以降更新したファイルは同様にプッシュしていった。次に、Vercel にアクセスし Github アカウントでサインアップした。次に、新しいプロジェクト emon-app-project を emon-project リポジトリを選択してインポートした。次にデプロイをクリックし、成功した後にブラウザで <https://emon-app-project.vercel.app/> にアクセスした。

3. 結果

3-1

図 1 に、Docker のバージョンを確認するコマンドの実行結果を示す。

```
sugawara@EDU2020E126:~/pbl$ docker version
Client:
 Cloud integration: v1.0.35+desktop.13
 Version:          26.1.1
 API version:      1.45
 Go version:       go1.21.9
 Git commit:       4cf5afa
 Built:            Tue Apr 30 11:46:57 2024
 OS/Arch:          linux/amd64
 Context:          default

Server: Docker Desktop
 Engine:
  Version:          26.1.1
  API version:      1.45 (minimum version 1.24)
  Go version:       go1.21.9
  Git commit:       ac2de55
  Built:            Tue Apr 30 11:48:28 2024
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.6.31
  GitCommit:       e377cd56a71523140ca6ae87e30244719194a521
 runc:
  Version:          1.1.12
  GitCommit:       v1.1.12-0-g51d5e94
 docker-init:
  Version:          0.19.0
  GitCommit:       de40ad0
```

図 1 docker version 実行結果

図 1 より、Docker Desktop のバージョンが 26.1.1 であることが確認できた。

図 2 に、docker-compose.yaml ファイル作成後に docker コンテナを起動した結果を示す。

```
○ sugawara@EDU2020E126:~/pb1$ docker compose up
WARN[0000] /home/sugawara/pb1/docker-compose.yaml: `version` is obsolete
[+] Running 1/0
✓ Container pbl-db-1 Created
Attaching to db-1
db-1 | PostgreSQL Database directory appears to contain a database; Skipping initialization
db-1 |
db-1 | 2024-08-11 05:13:30.716 UTC [1] LOG: starting PostgreSQL 16.3 (Debian 16.3-1.pgdg120+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 12.2.0-14) 12.
db-1 | 2.0, 64-bit
db-1 | 2024-08-11 05:13:30.717 UTC [1] LOG: listening on IPv4 address "0.0.0.0", port 5432
db-1 | 2024-08-11 05:13:30.717 UTC [1] LOG: listening on IPv6 address ":::", port 5432
db-1 | 2024-08-11 05:13:30.725 UTC [1] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
db-1 | 2024-08-11 05:13:30.743 UTC [29] LOG: database system was shut down at 2024-08-11 00:18:56 UTC
db-1 | 2024-08-11 05:13:30.779 UTC [1] LOG: database system is ready to accept connections
```

図 2 docker compose up 実行結果

図 2 より、データベースが起動したことが確認できた。

図 3 に、データベースに pgcli で接続した結果を示す。

```
○ sugawara@EDU2020E126:~/pb1$ pgcli -d postgres -h localhost -p 5432 -U user
Password for user:
Server: PostgreSQL 16.3 (Debian 16.3-1.pgdg120+1)
Version: 4.1.0
Home: http://pgcli.com
user@localhost:postgres>
```

図 3 データベースへの接続結果

図 3 より、docker-compose.yaml に設定したパスワード postgres を利用してデータベースに接続できた。

図 4 に、items テーブルの作成とカラムの追加結果を示す。

```
user@localhost:postgres> create table items(i
d int, name text)
CREATE TABLE
Time: 0.016s
user@localhost:postgres> select * from items
+---+-----+
| id | name |
+---+-----+
SELECT 0
Time: 0.003s
user@localhost:postgres> alter table items ad
d price int default 0;
You're about to run a destructive command.
Do you want to proceed? [y/N]: y
Your call!
ALTER TABLE
Time: 0.013s
user@localhost:postgres> select * from items
+---+-----+-----+
| id | name | price |
+---+-----+-----+
SELECT 0
Time: 0.003s
```

図 4 items テーブルの作成、カラムの追加結果

図 4 より、items テーブルのカラム id と name に price が追加されたことが分かる。

図 5 に、items テーブルにデータを挿入した結果を示す。

```
user@localhost:postgres> insert into items (i
d,name,price) values (1,'item1',1000)
INSERT 0 1
Time: 0.017s
user@localhost:postgres> select * from items
+-----+-----+
| id | name | price |
+-----+-----+
| 1 | item1 | 1000 |
+-----+-----+
SELECT 1
Time: 0.004s
user@localhost:postgres> insert into items (id,name,price) values (2,'item2',2000),(3,'item3',3000)
INSERT 0 2
Time: 0.003s
user@localhost:postgres> select * from items
+-----+-----+
| id | name | price |
+-----+-----+
| 1 | item1 | 1000 |
| 2 | item2 | 2000 |
| 3 | item3 | 3000 |
+-----+-----+
SELECT 3
Time: 0.006s
```

図 5 items テーブルへのデータ挿入結果

図 5 より、id、name、price をそれぞれ持つデータが挿入された。

図 6 に、name 列のみを表示した結果を示す。

```
user@localhost:postgres> select name from items
ms
+-----+
| name |
+-----+
| item1 |
| item2 |
| item3 |
+-----+
SELECT 3
Time: 0.004s
```

図 6 name 列表示結果

図 6 より、items テーブルの name 列のデータのみが表示された。

図 7 に、items テーブルのデータを更新した結果を示す。

```
user@localhost:postgres> update items set price=100 where id=1
You're about to run a destructive command.
Do you want to proceed? [y/N]: y
Your call!
UPDATE 1
Time: 0.022s
user@localhost:postgres> select * from items
+-----+-----+-----+
| id | name | price |
+-----+-----+-----+
| 2 | item2 | 2000 |
| 3 | item3 | 3000 |
| 1 | item1 | 100 |
+-----+-----+-----+
SELECT 3
Time: 0.003s
```

図 7 items テーブルのデータ更新結果

図 7 より、id が 1 のデータの price が 100 になった。

図 8 に、items テーブルの id に主キー制約を設定した後にデータを挿入した結果を示す。

```
user@localhost:postgres> insert into items (id,name,price) values (1,'item4',4000)
duplicate key value violates unique constraint "items_pkey"
DETAIL:  Key (id)=(1) already exists.
Time: 0.002s
```

図 8 主キー制約設定後のデータ挿入結果

図 8 より、id が 1 のデータは挿入できなかった。

図 9 に、外部キー制約を設定した orders テーブルの作成結果とデータの挿入結果を示す。

```
user@localhost:postgres> create table orders (order_id int primary key, order_date date, item_id int, foreign key (item_id) references items(id))
CREATE TABLE
Time: 0.006s
user@localhost:postgres> insert into orders (order_id, order_date, item_id) values (1,'2024-06-18',1)
INSERT 0 1
Time: 0.003s
user@localhost:postgres> insert into orders (order_id, order_date, item_id) values (2,'2024-06-19',4)
insert or update on table "orders" violates foreign key constraint "orders_item_id_fkey"
DETAIL:  Key (item_id)=(4) is not present in table "items".
Time: 0.002s
```

図 9 外部キー制約を設定した orders テーブルの作成とデータ挿入結果

図 9 より、item_id が 1 のデータは挿入可能で 4 のデータは挿入不可だった。

図 10 に、items テーブルと orders テーブルを結合して表示した結果を示す。

```
user@localhost:postgres> select orders.order_id, orders.order_date, items.id as item_id, items.name, items.price from orders inner join items
on orders.item_id = items.id
+-----+-----+-----+-----+-----+
| order_id | order_date | item_id | name | price |
+-----+-----+-----+-----+-----+
| 1        | 2024-06-18 | 1       | item1 | 100    |
+-----+-----+-----+-----+-----+
SELECT 1
Time: 0.006s
```

図 10 items と orders テーブルの結合表示結果

図 10 より、items の id が 1 のデータが、orders の item_id となって表示された。

3-2

図 11 に、books テーブルの作成結果を示す。

```
user@localhost:postgres> create table books (id integer, title text not null, price integer not null)
CREATE TABLE
Time: 0.010s
user@localhost:postgres> insert into books (id, title, price) values (1, 'book1', 1200)
INSERT 0 1
Time: 0.012s
user@localhost:postgres> insert into books (id, title, price) values (2, 'book2', 1000)
INSERT 0 1
Time: 0.018s
user@localhost:postgres> insert into books (id, title, price) values (3, 'book3', 1500)
INSERT 0 1
Time: 0.011s
user@localhost:postgres> select * from books
+-----+-----+-----+
| id | title | price |
+-----+-----+-----+
| 1 | book1 | 1200 |
| 2 | book2 | 1000 |
| 3 | book3 | 1500 |
+-----+-----+-----+
SELECT 3
Time: 0.009s
```

図 11 books テーブル作成結果

図 11 より、id、title、price をそれぞれ持つ 3 つのデータが books テーブルに挿入された。

図 12 に、FastAPI でサーバを起動した結果を示す。

```
○ sugawara@EDU2020E126:~/pbl$ fastapi dev main.py
INFO: Using path main.py
INFO: Resolved absolute path /home/sugawara/pbl/main.py
INFO: Searching for package file structure from directories with __init__.py files
INFO: Importing from /home/sugawara/pbl

Python module file
└─ main.py

INFO: Importing module main
INFO: Found importable FastAPI app

Importable FastAPI app
└─ from main import app

INFO: Using import string main:app

FastAPI CLI - Development mode

Serving at: http://127.0.0.1:8000
API docs: http://127.0.0.1:8000/docs
Running in development mode, for production use:
fastapi run

INFO: Will watch for changes in these directories: ['/home/sugawara/pbl']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [54929] using WatchFiles
INFO: Started server process [54952]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:60970 - "GET / HTTP/1.1" 200 OK
INFO: 127.0.0.1:60970 - "GET /favicon.ico HTTP/1.1" 404 Not Found
2024-08-11 15:47:41,009 INFO sqlalchemy.engine.Engine select pg_catalog.version()
2024-08-11 15:47:41,009 INFO sqlalchemy.engine.Engine [raw sql] {}
2024-08-11 15:47:41,015 INFO sqlalchemy.engine.Engine select current_schema()
2024-08-11 15:47:41,015 INFO sqlalchemy.engine.Engine [raw sql] {}
2024-08-11 15:47:41,019 INFO sqlalchemy.engine.Engine show standard_conforming_strings
2024-08-11 15:47:41,019 INFO sqlalchemy.engine.Engine [raw sql] {}
2024-08-11 15:47:41,036 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2024-08-11 15:47:41,037 INFO sqlalchemy.engine.Engine SELECT books.id AS books_id, books.title AS books_title, books.price AS books_price
FROM books
2024-08-11 15:47:41,037 INFO sqlalchemy.engine.Engine [generated in 0.00022s] {}
INFO: 127.0.0.1:44138 - "GET /books HTTP/1.1" 200 OK
```

図 12 FastAPI サーバ起動結果

図 12 より、`http://127.0.0.1:8000` でサーバが起動し、`/books` への GET リクエストへの応答が成功していることが確認できた。

図 13 に、<http://127.0.0.1:8000/books> へのアクセス結果を示す。

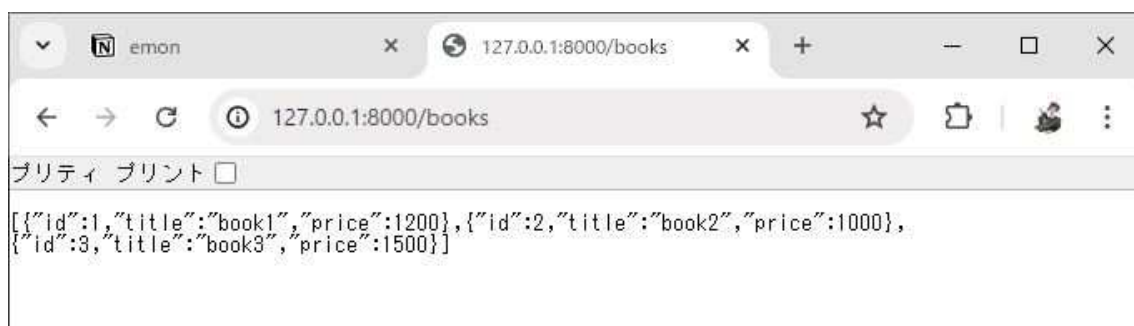


図 13 <http://127.0.0.1:8000/books> へのアクセス結果

図 13 より、books テーブルのデータが、id が 1 のものから順に表示された。

3-3

図 14 に、node.js のインストールとバージョンの確認コマンドの実行結果を示す。

```
sugawara@EDU2020E126:~$ mise use -q node@latest
mise ~/.mise.toml tools: node@20.15.1
sugawara@EDU2020E126:~$ node -v
v20.15.1
```

図 14 node.js のインストールとバージョン確認結果

図 14 より、node.js のバージョン 20.15.1 がインストールされた。

図 15 に、Next.js プロジェクトを作成して yarn をインストールし、yarn dev でサーバを起動した後にブラウザで <http://localhost:3000/> にアクセスした結果を示す。

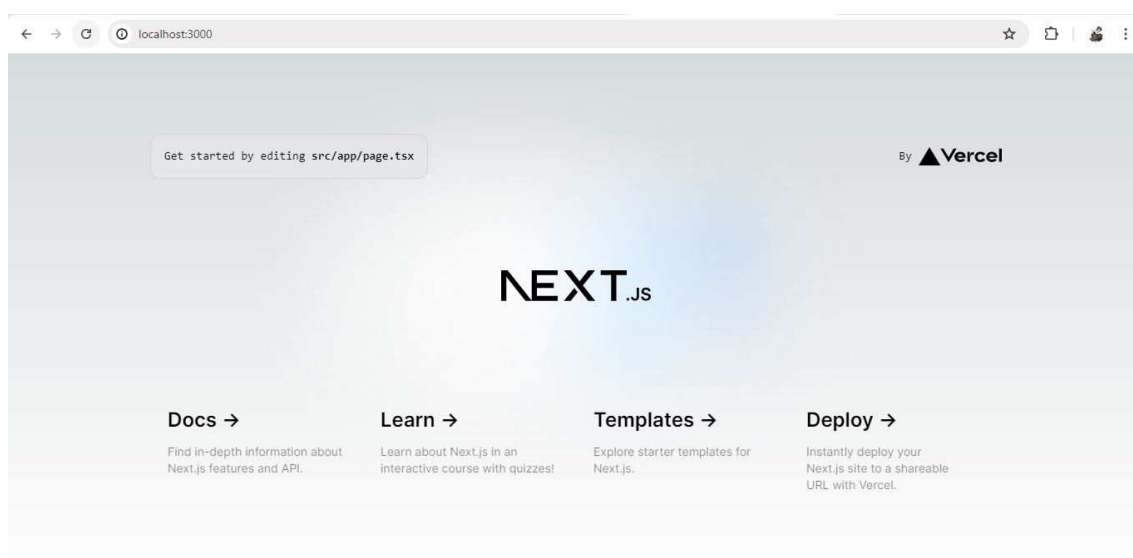


図 15 <http://localhost:3000/> へのアクセス結果

図 15 より、Next.js のサーバのデフォルトのページが表示された。

図 16 に、hello.tsx 作成後に <http://localhost:3000/hello> にアクセスした結果を示す。



図 16 <http://localhost:3000/hello> へのアクセス結果

図 16 より、hello が表示された。

図 17 に、chakra-ui をインストールし、_app.tsx と hello.tsx を書き換えた後に <http://localhost:3000/hello> にアクセスした結果を示す。

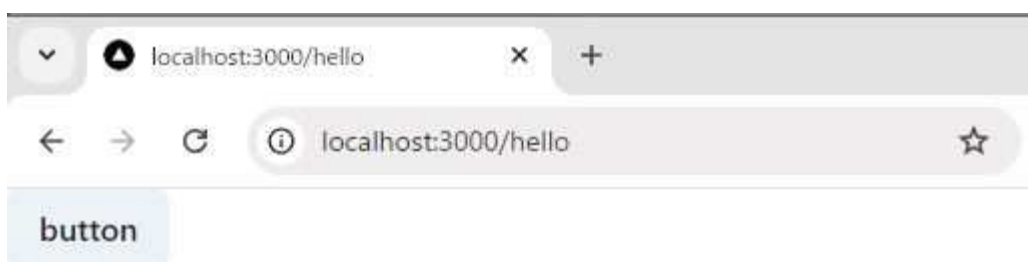


図 17 コード変更後の <http://localhost:3000/hello> へのアクセス結果

図 17 より、chakra-ui のコンポーネントの一つであるボタンを実装できた。

3-4

図 18 に、Github の emon-project リポジトリに emon-app ディレクトリの内容をプッシュした結果を示す。

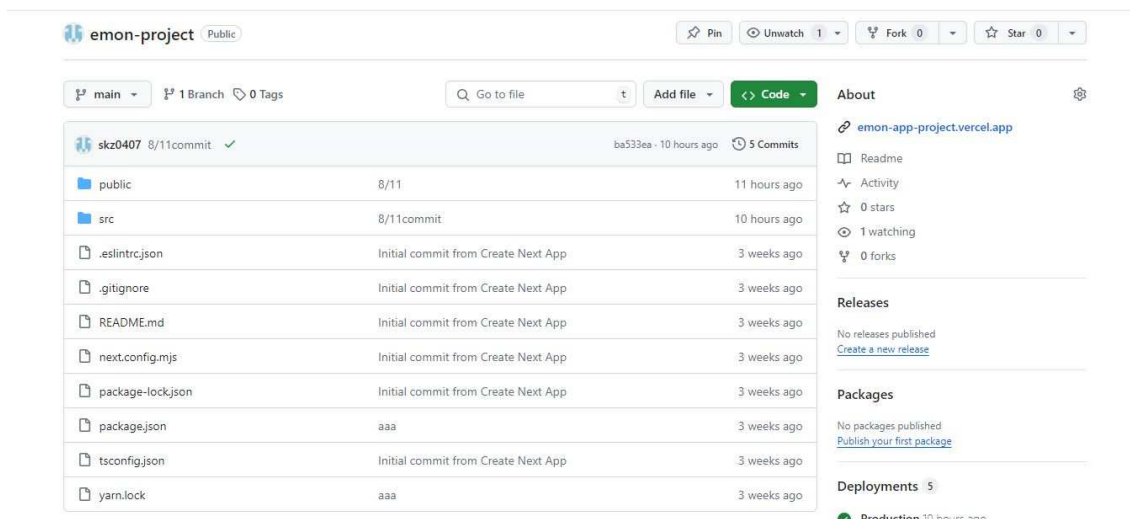


図 18 emon-app ディレクトリの内容プッシュ結果

図 18 より、変更点のあるファイルやディレクトリをプッシュして更新していった。

図 19 に、Vercel に Github の emon-project リポジトリをインポートしデプロイした後に <https://emon-app-project.vercel.app/> にアクセスした結果を示す。

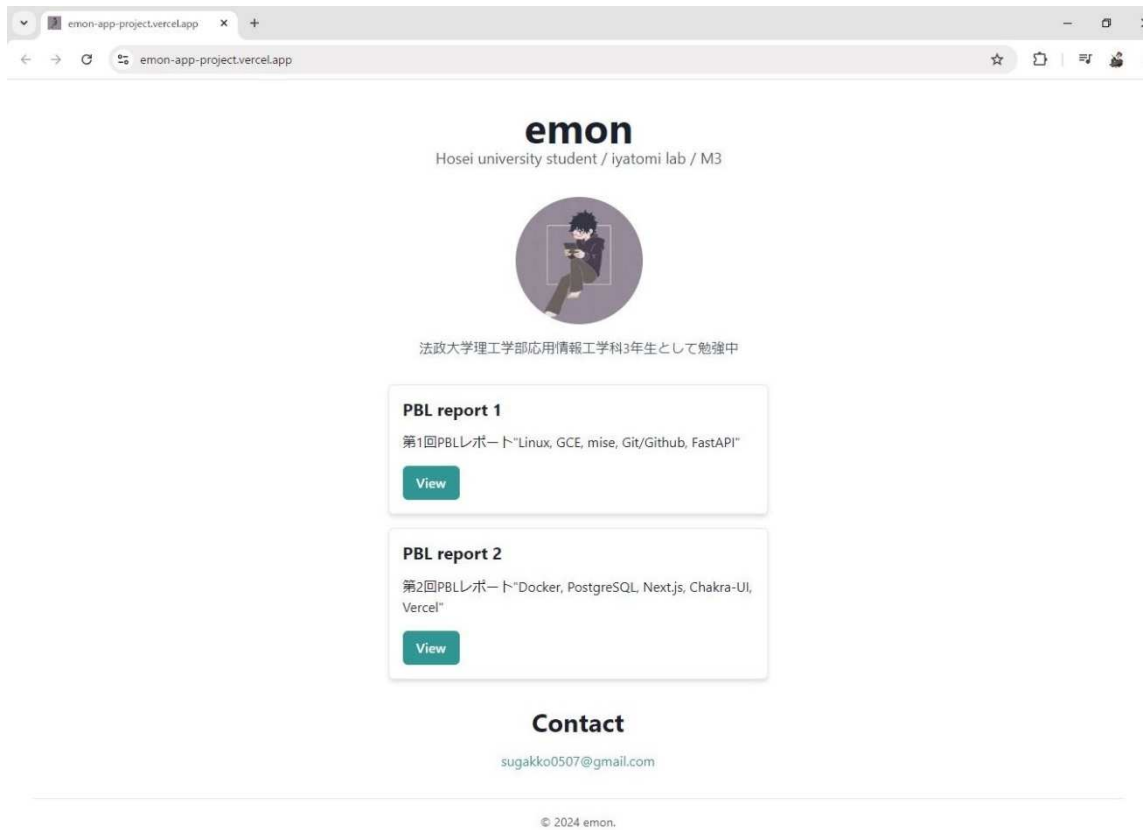


図 19 https://emon-app-project.vercel.app アクセス結果

図 19 より、作成したページが公開された。

4. 考察

4-1

`docker-compose.yaml` ファイルは、サービスを定義し、管理するための設定ファイルであり、ここではデータベースをサービスに定義し、PostgreSQL データベースを起動するようにしている。また、ポートを 5432 に設定し、データをコンテナ外の `db-store` ボリュームに保存するようにし、ユーザ名を `user`、パスワードを `postgres` に設定している。これらの設定をファイルに記述することで、コンテナの管理が容易になると考えられる。


データベースの操作について、`create table` コマンドで作成したテーブルに `alter table` コマンドによって後からカラムを追加できた。これはテーブル設計が柔軟であることを示している。また、`id` に主キー制約を設定したことで、同じ `id` を持つデータを挿入できなくなった。また、`orders` テーブルの `item_id` に対して外部キー制約を設定することで、`items` テーブルの `id` と関連付けられ、`items` テーブルに存在しない `id` を `orders` テーブルに挿入できなくなった。これらのキー制約によってデータの一意性を保つことができると考えられる。また、`inner join` を用いたテーブル結合では、`orders` テーブルと `items` テーブルを結合して関連する情報を一度に表示できた。この方法によって複数のテーブル間で関連するデータを効率よく集められるようになると考えられる。

4-2

SQLAlchemy はデータベースとのやり取りを `python` で行うためのツールであり、データベースとオブジェクト指向プログラミング間の違いを変換してくれる ORM の一つである。`models.py` ファイルでは、`books` テーブルの定義をしており、SQL では `create table` コマンドの部分と対応している。`database.py` ファイルでは、接続先のデータベースを設定しているほか、データ取得のための `read_books()` 関数を実装している。`session.query(models.Book).all()` の部分は、SQL では `select * from books` に対応している。`main.py` ファイルでは、`/books` への GET リクエストに対して `read_books()` 関数を呼び出してデータベースからデータを取得し返している。また、`BookRead` クラスを定義することで、`id`、`title`、`price` の構造と一致することを確認し、この順で応答するようにしている。これらのファイルの設定により、`http://127.0.0.1:8000/books` にアクセスすることでデータベースで作成した `books` テーブルの内容をブラウザで表示できたと考えられる。

4-3

JavaScript はブラウザで動作するプログラミング言語であり、サーバサイドで利用される汎用的なスクリプト言語である。TypeScript は JavaScript に静的型付け機能を追加したもので、JavaScript にコンパイルしてから実行される。Node.js は JavaScript をサー

バサイドで実行する環境で、高速で効率的なウェブサーバを構築できる。対抗には Node.js の製作者によって作られた Deno  がある。Deno には npm が無く node_modules が無いため import に URL を渡して外部モジュールを使用するといった違いや、デフォルトでセキュアになっているといった違いがある。Next.js は React ベースのフレームワークであり、サーバサイドレンダリングや静的サイト生成をサポートしている。対抗には、React ベースで、静的サイトの生成に特化した Gatsby がある。yarn は Node.js のパッケージマネージャーで依存関係の管理やインストール速度に優れている。対抗には Node.js に標準搭載の npm がある。Chakra-UI は React ベースの UI フレームワークでアクセシビリティに優れ、CSS を記述しなくてもパラメータ指定でスタイルを記述できる。対抗には、React ベースで Google のマテリアルデザインのガイドラインを実装した Material UI がある。コンポーネントを分割することで、同じコンポーネントを複数の場所で再利用できたり、各コンポーネントの修正が容易になったりという利点がある。

4-4

Vercel で Github の emon-project リポジトリをインポートしてデプロイしたため、リポジトリの変更が自動的に Vercel で反映されるようになっている。そのため、ローカルの変更をプッシュすることですぐに反映されたと考えられる。作成したポートフォリオサイトには、Box, Flex, Heading, Text, Button, Image, VStack, Link の chakra-UI コンポーネントを利用した。Box や Flex でレイアウトを整え、Heading で各セクションの見出しを付け、Text でテキストを表示し、Button をクリックすることで新しいタブで public/pdf に入れた PDF ファイルを開くようにし、Image で public/img に入れたプロフィール画像を表示し、VStack でワークセクションの 2 つの要素を縦に並べ、Link で Contact 用にメールアドレスのハイパーリンクを表示した。Chakra-UI のカラースキームを利用し、サイトのデザインを統一した。また、_app.tsx を編集してファビコンも変更した。

5. 結論

第 13 回までの授業において、yaml ファイルを用いた Docker の環境の設定、構築と PostgreSQL によるデータベースの操作、また FastAPI、SQLAlchemy を用いたサーバへのデータベースの組み込み、Next.js と Chakra-UI を用いたフロントエンド開発、作成したサイトの Vercel へのデプロイという各実験を通し、現代の Web 開発における基本的な知識と技術を理解した。

6. 参考文献

- 初心者でもサクッとできる Docker Compose ハンズオン
(<https://envader.plus/article/327>)
- 基本的な PostgreSQL の操作方法
(<https://qiita.com/Utsubo/items/04aa2281046cf55aee1a>)
- SQLAlchemy でテーブル設計と ORM の操作
(https://zenn.dev/shimakaze_soft/articles/6e5e47851459f5)
- Chakra UI
(<https://v2.chakra-ui.com/>)
- Deno とはなにか - 実際に試してみる
(<https://qiita.com/azukiazusa/items/8238c0c68ed525377883>)
- Next.js と Gatsby を比較
(<https://cheezblog.netlify.app/article/oeui4ddaa2jrnuumwadxp/>)
- Utility-First な CSS, UI フレームワークを比較してみた (TailwindCSS, Chakra UI, MUI)
(<https://zenn.dev/kiyokiyoabc/articles/f688f2cee95f04>)
- Next.js13(App router) x ChakraUI でポートフォリオ作ってみた
(<https://zenn.dev/sasaharumedes/articles/3e1eea0909c746>)
- Chakra UI 使ってみた話 #React
(<https://qiita.com/so1bloom/items/72ebb6119b06ff3c2507>)
- Next.js で favicon を設定する
(<https://teitei-tk.hatenablog.com/entry/2020/05/21/120000>)