MySQL × Django × Next.js を Docker コンテナでつなぐ勉強記録

1. 目的

- Django をバックエンド、Next.js をフロントエンド、MySQL をデータベースとしてコンテナ上で連携させる
- Docker と Docker Compose を使ったマルチコンテナ開発環境を理解する
- Next.js から Django API を呼び出す手順を確認する

2. 使用技術・バージョン

- Docker / Docker Compose
- Django 5.x
- Next.js 14.x
- MySQL 8.x
- Python 3.12
- Node.js 20.x

3. 環境構築の流れ

3.1 Docker Compose の構成（説明に必要のない設定は省いている）

```
version: "3.9"


services:

  db:

    image: mysql:8.0
```

```
    container_name: tabelog-db
    ports:
      - "3307:3306"

  backend:
    container_name: tabelog-backend
    ports:
      - "8000:8000"
    depends_on:
      db

  frontend:
    container_name: tabelog-frontend
    ports:
      - "3000:3000"
```

*ポイント

docker-composeでdb, frontend, backendを入れてbackendにはdbを依存させる設定を行う

→これよりdocker-composeを起動すれば、このdockerのネットワークが構築されbackend側'db'の設定が認識できるようになり、frontendから'backend'の設定を呼び出すことができるようになる。

例)

Next.jsからDjangoを呼び出すとき"http://backend:8000"でAPIにアクセスできる

DjangoからMySQLを呼び出すときDATABASESの設定でdbの情報を扱える

3.2 Django 設定

- データベース設定 (`settings.py`):

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': ‘tabelog',
        'USER': 'user',
        'PASSWORD': 'pass',
        'HOST': 'db',
        'PORT': '3306',
    }
}
```

*ポイント

docker-composeで設定したdbの情報(user, password, host, port(dbコンテナの))を入力すればいい

- CORS 設定 (Next.js からアクセスする場合):

  フロントからDjandoへのアクセス許可設定を行う

```
INSTALLED_APPS = [
    'corsheaders',
    ...
]
```

```
MIDDLEWARE = [
    'corsheaders.middleware.CorsMiddleware',
    ...
]

CORS_ALLOWED_ORIGINS = [
    "http://localhost:3000",(ホスト側からアクセスしるとき)
     ”http://frontend:3000”,（コンテナ側からアクセスするとき）
]
```

4. Django + MySQL + Next.js 連携手順

4 - 1. Django アプリの作成

```
docker-compose exec backend python manage.py startapp hellow
```

1, api/hello/apps.py の設定

```
from django.apps import AppConfig

class HellowConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'api.hello'
```

ポイント
Django に `api.hello` という名前のアプリケーション情報が登録される

2, config/settings.py の設定

```
INSTALLED_APPS = [

    'api.hellow',

]
```

ポイント
Django 側にapi.hellow アプリを使うと宣言する。

4 - 2. モデル定義

api/hello/models.py

```
from django.db import models


class Hello(models.Model):

    message = models.CharField(max_length=100)


    class Meta:

        db_table = 'hellow'
```

ポイント
class Meta ではモデルに関する設定が可能。

- データベースに登録するテーブル名
- デフォルトの並び順 など

4 - 3. マイグレーションで MySQL にテーブル作成

```
docker-compose exec backend python manage.py makemigrations
```

```
docker-compose exec backend python manage.py migrate
```

ポイント

Docker コンテナ内のネットワークで実行する必要がある。

`db` と `backend` が同じネットワークにいる必要がある
→ これで MySQL 内の`tabelog`データベースに`hello` テーブルが作成される。

4 - 4. MySQL Workbench で仮データを追加

1. MySQL Workbench を起動
2. 新しい接続を作成（接続情報は `docker-compose` 設定に合わせる）

```
Hostname: 127.0.0.1
Port: 3307   （ホスト側で公開しているポート）
Username: user
Password: password
```

ポイント

接続情報は Workbench → localhost:3307 → db コンテナの 3306 の順

なのでWorkBenchの接続情報はコンテナで定義したMySQLの情報ではなくホストPC からの情報を設定する必要がある

3. 左側の **SCHEMAS** で `tabelog` を選択
4. `hello` テーブルを右クリック → **Select Rows - Limit 1000**
5. 表が開いたら下の行に自由にデータを入力して保存

これで Django 側から取得できるデータが MySQL に登録できる

4 - 5. DjangoでMySQLから からデータを取得する

api/hello/views.py

```
from rest_framework.views import APIView
from rest_framework.response import Response
from .models import Hellow

class Backend(APIView):
    def get(self, request, format=None):
        entry = Hellow.objects.get(id=1)
        return Response({"message": entry.message})
```

4 - 6. バックエンドのルーティング設定

1,api/hello/urls.py

```
from django.urls import path
from .views import Backend

urlpatterns = [
    path('backend/', Backend.as_view()),
]
```

2,config/urls.py

```
from django.urls import path, include


urlpatterns = [

    path('api/hello', include('api.hello.urls')),

]
```

これで http://backend:8000/api/hello/backend/ でアクセス可能になる。

4 - 7. Next.js から Django API を呼び出す

1, API 呼び出し用関数

lib/api.ts

```
const API_URL = process.env.NEXT_PUBLIC_API_URL ??
"http://backend:8000";


export async function getHello() {

  const res = await fetch(`${API_URL}/api/hellow/backend/`);

  if (!res.ok) {

    throw new Error("API Error");

  }

  return res.json();

}
```

2, コンポーネント側での利用

app/page.tsx

```
import { getHello } from "@/lib/api";

export default async function Home() {
  const data = await getHello();

  return (
    <main>
      <h1>メッセージ: {data.message}</h1>
    </main>
  );
}
```