Next.js × Django × MySQL を用いたフォーム入力とデータ保存フローのまとめ

1. 概要

この資料では、フロントエンドにNext.jsを使用してフォームを作成し、バックエンドのDjangoを経由してMySQLにデータを保存する方法をまとめる。
フローとしては以下の通りである:

Next.js（フォーム入力）→ **API**リクエスト**(post)** →

Django（受信・処理）→ **MySQL**（保存）

Next.js（表示リクエスト）→ **API**リクエスト**(get)** →

Django（受信・処理）← **MySQL**（データ）

Django（送信・処理）→ **Next.js**（表示）

2. バックエンド（Django）

2.1 Djangoモデル作成

MySQLに保存するデータ用のモデルを作成する。

```
# api/name/models.py
from django.db import models

class Name(models.Model):
    name = models.CharField(max_length=100)
    created_at = models.DateTimeField(auto_now_add=True)

   class Meta:
      db_name = 'names'
```

*point
  class Metaでテーブルの設定ができる(名前、ソート順, 制約など)
  ここではテーブルの名前を'names'にしている

設定がなければテーブルの名前は自動的にapi_name_nameとなる

3.2 シリアライザー

REST API用にシリアライザーを作成する。

→なぜ必要か

①DjangoモデルとJsonを変換するためのもの

②バリデーションの設定ができるから

```
# api/name/serializers.py
from rest_framework import serializers
from .models import Name

class NameSerializer(serializers.ModelSerializer):

    # フィールド単位のバリデーション
    def validate_name(self, value):
        if not value:
            raise serializers.ValidationError("名前は必須です。")
        if value.isdigit():
            raise serializers.ValidationError("名前は数字のみNG")
        if len(value) > 10:
            raise serializers.ValidationError("名前は10文字以内)
        return value

    class Meta:
        model = Name
        fields = ['id', 'name', 'created_at']
```

*point
①モデルで格納されていたとJsonデータの変換
(1)フロントから受け取るとき(JSON → Pythonオブジェクト（モデル）)
{ "name": "Taro" }→Nameモデルのインスタンスに変換

*name以外は自動生成されるのでそれ以外のデータはフロントから必要ない

(2)DBに保存されたデータを返すとき(Pythonオブジェクト（モデル） → JSON)
Nameモデルのインスタンス

```
→
{
  "id": 1,
  "name": "Taro",
  "created_at": "2025-11-25T12:00:00Z"
}
```

②バリデーションの設定(frontからデータが送られたときのチェック)
validate_フィールド名(self, value)
でフィールドに対するバリデーションを設定できる

今回は{“name”}が送られてきているのでnameフィールドに対しての
バリデーションを設定する

→問題があればフロントに400 Bad Requestとエラー内容のJSONなどを返す

2.3 ビュー（Django REST Framework）

POSTリクエストを受け取り、MySQLにデータを保存する。

GETリクエストでMySQLにデータをフロントに渡す。

```python
# api/name/views.py
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
from .serializers import NameSerializer
from .models import Name

class NameCreateAPIView(APIView):

    # GET：データ一覧を取得する
    def get(self, request):
        names = Name.objects.all().order_by('-created_at')
        serializer = NameSerializer(names, many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)

    # POST：データを新規作成する
    def post(self, request):
        serializer = NameSerializer(data=request.data)
        if serializer.is_valid():
```

```
            serializer.save()
            return Response({"message": "データ保存成功"},
status=status.HTTP_201_CREATED)
        return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)
```

*point
(1)getメソッド
Nameモデルに保存されているデータをすべて取得
シリアライザーでモデル→JSONに変換
シリアライザーのデータ(JSON)とステータスコード200をフロントに送信

(例)

```
    [
      {
        "id": 5,
        "name": "Taro",
        "created_at": "2025-11-25T05:12:33Z"
      },
      {
        "id": 4,
        "name": "Hanako",
        "created_at": "2025-11-25T04:00:11Z"
      }
    ]
```

(2)postメソッド
シリアライザーでフロントからのデータとNameモデルとの紐づけの準備をする
serializer = NameSerializer(data=request.data)

シリアライザーのバリデーションに成功
→保存＆データ保存成功メッセージ＆ステータスコード201

シリアライザーのバリデーションに失敗
→serializer.pyで定義したエラーメッセージ＆ステータスコード400
をfrontendに返す。

2.4 URL設定

APIエンドポイントの設定

```
# api/urls.py
from django.urls import path
from .views import NameCreateAPIView

urlpatterns = [
    path('names/', NameCreateAPIView.as_view(), name='name-create'),
]


# project/urls.py
from django.urls import path, include

urlpatterns = [
    path('api/', include('api.urls')),
]
```

3. フロントエンド（Next.js）

基本的なフォルダ構成

```
/frontend
|
├─ /app                          # Next.js 13+ の App Router
|    ├─ page.tsx                 # トップページ（フォーム + 一覧を表示）
|    └─ layout.tsx               # 共通レイアウト
|
├─ /components                   # 共通コンポーネントを格納
|    ├─ NameInputForm.tsx        # 名前入力フォームコンポーネント
|    ├─ NameList.tsx             # 名前一覧表示コンポーネント
|    └─ ...他コンポーネント
|
├─ /lib                          # API 呼び出し関数など
|    └─ api.ts                   # Django API と通信する関数（GET/POST）
```

3.1 API呼び出し関数

Next.jsからDjangoへPOST、GETリクエストを送信する関数を作成する。

```
#lib/api.ts

const API_URL = "http://localhost:8000";

// POST：新しい名前を作成
export async function createName(data) {
  const res = await fetch(`${API_URL}/api/name/`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(data),
  });

  if (!res.ok) throw new Error("API Error");
  return res.json();
}

// GET：名前の一覧を取得
export async function getNames() {
  const res = await fetch(`${API_URL}/api/name/`, {
    method: "GET",
    headers: { "Content-Type": "application/json" },
  });

  if (!res.ok) throw new Error("API Error");
  return res.json();
}
```

3.2 フォームコンポーネントの作成

```
# /components/NameInputForm.tsx
'use client';
import { useState, FormEvent } from "react";
```

*point
親コンポーネントからonSubmit関数を受け取り、名前が送信されたときに
その名前を引数にして親に渡す仕組み

```
type Props = {
  onSubmit: (name: string) => void;
};



export default function NameInputForm({ onSubmit }: Props) {
```

name→入力欄に表示される文字列
setName→入力が変わるたびに値を更新して、画面を更新。
更新した値をnameに代入するする関数
setName(“値”)で画面の再レンダリングとnameへの”値の代入”
が行われる。(初期値は””)

```
  const [name, setName] = useState("");

  const handleSubmit = (e: FormEvent) => {
```

ページリロードを防ぐ

```
    e.preventDefault();
```

フロントエンドのバリデーションの設定を行う

```
    if (!name.trim()) {
       alert("名前を入力してください");
       return;
    }

    if (name.length > 20) {
        alert("名前は20文字以内で入力してください");
        return;
    }
```

さっき作ったonSubmmit関数でnameを親コンポネントに渡す

```
    onSubmit(name);
```

入力欄をリセット

```
    setName("");
  };
```

```
  <UIの設定>
  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        value={name}
        onChange={(e) => setName(e.target.value)}
        placeholder="名前を入力"
      />
      <button type="submit">送信</button>
    </form>
  );
}
```

3.3 一覧コンポーネントの作成

```
# /components/NameList.tsx
```

要求している JSON データの形をTypeScriptで定義している

①一つの名前のデータ

```
type Name = {
  id: number;
  name: string;
  created_at: string;
};
```

②Nameが複数定義されている場合

```
type Props = {
  names: Name[];
};
```

→例

```
      {
        "names": [
          {
            "id": 1,
            "name": "太郎",
            "created_at": "2025-01-20T10:00:00Z"
          },
```

```
        {
          "id": 2,
          "name": "花子",
          "created_at": "2025-01-21T09:00:00Z"
        }
      ]
    }
```

のデータを受け取る

<UIの設定　外部にNameListとして扱えるようにする>

```
export default function NameList({ names }: Props) {
  return (
    <div>
      <h3>登録済みの名前一覧</h3>
      <ul>
        {names.map((n) => (
          <li key={n.id}>
            {n.name} ({new Date(n.created_at).toLocaleString()})
          </li>
        ))}
      </ul>
    </div>
  );
}
```

3.4 コンポーネントの利用と画面の表示

```
# /app/page.tsx
'use client';
import { useState, useEffect } from "react";
import NameInputForm from "@/components/NameInputForm";
import NameList from "@/components/NameList";
import { createName, getNames } from "@/lib/api";

export default function HomePage() {
```

namesList：現在表示している名前一覧（配列）
message：登録成功やエラーメッセージを表示するための文字列

```
  const [namesList, setNamesList] = useState([]);
```

```
const [message, setMessage] = useState("");

ページが読み込まれた最初の1回だけ名前一覧のデータを持ってくる
useEffect(() => {
  fetchNames();
}, []);


#名前一覧取得の関数
const fetchNames = async () => {
  try {
   lib/apis.tsのgetNames()を使いdjanoからデータを取得
    const data = await getNames();

   現在のnamelistの値を最新のものへと更新
    setNamesList(data);

  } catch (error) {
    console.error("データ取得に失敗しました", error);
  }
};


#名前送信処理の関数
 →子コンポーネントのhandleSubmitからnameが渡されている
   そのnameを今度はlub/apis.tsxのcreateName()に渡して
   nameをDjnagoに送っている
const handleSubmit = async (name: string) => {
  try {
    const response = await createName({ name });
    setMessage(response.message);

   nameを送った後にNameListを改めてとってくる
    fetchNames();

  } catch (error) {
    setMessage("送信に失敗しました");
  }
};

return (
```

```
    <div>
      <NameInputForm onSubmit={handleSubmit} />
      <p>{message}</p>
       最新のnameListを子コンポーネントのNameListに送っている
      <NameList names={namesList} />
    </div>
  );
}
```