Djangoの説明

Djangoとは

Python製のフルスタックWebフレームワーク

Webアプリ開発に必要な要素（認証、管理画面、DB操作、テンプレート、セキュリティなど）が最初から揃っており、最小のコードで堅牢な**Web**サービスを構築できる。

出来ること

- 認証・ユーザー管理、パスワードリセット、権限ロール設定
- 管理画面（`/admin`）の自動生成
- ORM（Object Relational Mapper）によるDB操作（SQL不要でマイグレーション管理）
- Djangoテンプレート（Jinjaライク）によるHTML生成
- `Form` / `ModelForm` での入力検証・エラーメッセージ処理
- ファイルアップロード、画像処理（`ImageField`）
- 多言語化（i18n）、タイムゾーン対応
- キャッシュ、ミドルウェア、CSRF・クリックジャッキング対策などのセキュリティ機構
- 非同期ビュー（ASGI）対応、WebSocketは **Django Channels** で利用可能
- REST APIは **Django REST Framework**（**DRF**）で高速実装
- テスト環境標準装備（UnitTest / pytestで回しやすい）
- 大規模開発に向く構造化（アプリ分割・signals・appsによるモジュール管理）

主な用途

- 業務システム（顧客管理・案件管理・在庫／受発注・ワークフロー）
- ダッシュボード／内部ツール（集計・可視化・社内データ管理）
- コンテンツサイト／**CMS**（ニュース・メディア・社内ポータル）
- **EC**／予約サイト（カート・在庫・決済・予約スロット）
- 会員制サービス（ログイン・課金・通知・プロフィール管理）
- **REST**／**GraphQL API**バックエンド（SPAやモバイルアプリの裏側）
- 教育・学習管理（学習記録・テスト配信・進捗可視化）
- 認証付きミニツール（社内フォーム、情報収集アプリなど）

Djangoの強み

- 初速が速い：ログイン、管理画面、DB連携が初日から動く
- 保守性が高い：アプリ分割構成で大規模化しても破綻しにくい
- セキュリティが堅い：CSRF、XSS、クリックジャッキング対策が標準
- エコシステムが充実：DRF、Channels、Celery、Wagtail、django-allauth など強力プラグイン群
- ドキュメント・情報量が豊富

Djangoより他が向く場面

- 軽量**API**やワンエンドポイントのみ → Flask / FastAPI が適
- 高頻度のリアルタイム通信や**WebSocket**特化 → FastAPI + 専用スタック
- 関数一発のサーバレス構成 → Cloud Functions / AWS Lambda + FastAPI

Django × DRF（API構築）

- シリアライザで入力・出力を型定義 → 自動バリデーション
- `ViewSet` + `Router` で CRUD API を一括生成
- 認証（Token / JWT）、権限制御、ページネーション、フィルタ、検索などが標準で揃う

DjangoとFlask/FastAPIの違い

- **Django**：フルスタック。管理画面・認証・ORM・フォームが標準搭載。安全かつ高速に「全部」構築。
- **Flask**：マイクロ。必要な部品を自分で選んで足す。小〜中規模、自由度重視。
- **FastAPI**：型定義ベース・高速・非同期処理に強い。API専用構成。OpenAPIドキュメントを自動生成。

🔧 Djangoプロジェクトの始め方

前提

- OS：Windows + WSL2（Ubuntu）または macOS / Linux
- Python：3.10以上
- VS Code 推奨（Dev Containers or venv対応）

仮想環境の作成

Pythonパッケージを隔離して扱うため、必ず仮想環境を作る。

これをやっておくと後で他のプロジェクトに影響しない。

```
# プロジェクトフォルダ作成
mkdir ~/projects/Django_app
cd ~/projects/Django_app

# 仮想環境作成（.venvという名前が標準的）
python3 -m venv .venv

# 有効化
source .venv/bin/activate   # Linux / macOS
# Windows の場合
# .venv\Scripts\activate
```

1.Djangoをインストール

```
pip install --upgrade pip
pip install django
```

バージョン確認：

```
python -m django --version
```

2.プロジェクト作成

Djangoでは「プロジェクト」＝全体、「アプリ」＝機能単位

まず全体のプロジェクトを作る：

```
django-admin startproject config .
```

「.」を付けている理由：無駄な二重フォルダ（config/config/）にならない為

生成後の構成：

```
Django_app/
├── manage.py
├── config/
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── .venv/
```

3.サーバー起動テスト

```
python manage.py runserver
```

アクセス:

```
http://127.0.0.1:8000/
```

Djangoのウェルカムページ（ロケット画面）が出たらOK

4.アプリ作成

次に、個別機能（例：電卓）を「アプリ」として追加する

```
python manage.py startapp calculator
```

構成：

```
Django_app/
├── manage.py
├── config/
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── calculator/
│   ├── admin.py
│   ├── apps.py
│   ├── models.py
│   ├── tests.py
│   ├── urls.py      ← 自分で新しく作成(8.calculator/urls.pyで説明)
│   ├── views.py
│   └── templates/
│       └── calculator/
│           └── index.html ← 自分で新しく作成
└── .venv/
```

5.settings.py にアプリを登録

Djangoに「このアプリを使用する」と教える設定

登録しないとモデル（データベース）やテンプレートが読み込まれない

※新しくアプリが作られるたびに追加

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # --- custom apps ---
    'calculator',
]
```

6.config/urls.py にルートを追加

WebページのURLをアプリに繋ぐ設定

```
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    # Django標準の管理画面URL
    path('admin/', admin.site.urls),

    # --- シンプル構成（アプリが1〜2個のとき）---
    #「/calculator/」にアクセスされたら、calculator/urls.py の設定が使われる
    path('calculator/', include('calculator.urls')),

    # 例: 他のアプリを追加する場合
    # path('studylog/', include('studylog.urls')),
]

# --- もしアプリが3つ以上に増えたら、namespace付きで書くと安全 ---
# （name='index' のようなURL名が重複しても衝突しなくなる）

# urlpatterns = [
#     path('admin/', admin.site.urls),
#     path('calculator/', include(('calculator.urls', 'calculator'),
namespace='calculator')),
#     path('studylog/', include(('studylog.urls', 'studylog'),
namespace='studylog')),
#     path('inputValue/', include(('inputValue.urls', 'inputValue'),
namespace='inputValue')),
# ]

# namespace付きなら、テンプレートでこう呼べる:
# {% url 'calculator:index' %}  → 電卓トップページ
# {% url 'studylog:index' %}    → 学習ログトップページ
# {% url 'inputValue:index' %}  → 入力画面トップページ
```

7.calculator/urls.py 作成

電卓アプリの中でページを振り分ける。ここでは「`/calculator/` にアクセスされたときに、8. の `def index(request)` を呼び出す」設定をしている。

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

urlpatterns = [
    path('', views.index, name='index'),
]
```

8.calculator/views.pyに処理追加

ページを開いたときに実行される処理を定義

```
from django.shortcuts import render

def index(request):
    return render(request, 'calculator/index.html')
```

9.テンプレート作成（電卓）

実際に表示されるHTMLファイル

templates/calculator/index.html

```
<!DOCTYPE html>
<html lang="ja">
<head>
    <meta charset="UTF-8">
    <title>電卓</title>
</head>
<body>
    <h1>Django 電卓アプリ</h1>
    <p>最初のページが表示されました。</p>
</body>
</html>
```

10.起動確認

開発用サーバーを立ち上げて確認（URL: http://127.0.0.1:8000/calculator/）

```
python manage.py runserver
```

アクセス：

```
http://127.0.0.1:8000/calculator/
```

ブラウザでアクセスして、「Django 電卓アプリ」と出ればOK。

11.Git管理

変更履歴を残せるようにする（後でGitHubにも上げられる）

```
# Gitリポジトリを初期化（このフォルダをGit管理下にする）
git init

# 仮想環境フォルダ（.venv）をGitの管理対象から除外する設定ファイルを作成
echo ".venv/" > .gitignore

# Pythonのキャッシュフォルダ（__pycache__）も除外リストに追記
echo "__pycache__/" >> .gitignore

# すべてのファイルをステージに追加（＝次のコミット対象にする）
git add .

# 最初のコミット（履歴に保存）を作成。メッセージで「Djangoプロジェクト初期設定」と説明
git commit -m "Initial Django project setup"
```

12.requirements.txt を作る（環境再現用）

使っているライブラリを一覧にしておく。他の環境でも同じ状態を再現できる。

```
# 現在の仮想環境に入っているライブラリを一覧にして requirements.txt に書き出す
pip freeze > requirements.txt
```

次に別の環境で再現する場合：

```
# requirements.txt に書かれたライブラリを一括インストール
#（別のPCや本番サーバーなどで、同じ環境を再現するときに使う）
pip install -r requirements.txt
```

なぜ必要なのか

`requirements.txt` は「このプロジェクトが動くために必要なライブラリのリスト」を保存するためのファイル
開発中は Django や他のツールを `pip install` で追加していくけど、時間がたつと「何を入れたのか」「どのバージョンだったのか」がわからなくなる。

このファイルを作っておけば：

- 他の人が同じ環境を再現できる
- 本番サーバーや別のPCでも同じ設定で動かせる
- 将来の自分も環境を復元できる

例：生成される中身

```
asgiref==3.8.1
Django==5.1.2
sqlparse==0.5.1
tzdata==2024.2
```

こうしておくと、どんな環境でもこのDjangoプロジェクトをまったく同じ状態で再現できる

13.VS Code用設定（任意）

VS Code はフォルダごとに設定を保存できる。

プロジェクト直下（＝`Django_app/`）に `.vscode` フォルダを作って、その中に設定ファイルを置く

📁 フォルダ構成

```
Django_app/
├── .vscode/   ← 自分で新しく作成
│   └── settings.json   ← 自分で新しく作成
├── manage.py
├── config/
├── calculator/
└── .venv/
```

⚙️ `.vscode/settings.json` の中身

```
{
    // Pythonの実行環境をこのプロジェクト専用の仮想環境に固定
    "python.defaultInterpreterPath": ".venv/bin/python",

    // VS Codeがコード解析する時の参照パス（config やアプリを認識させる）
    "python.analysis.extraPaths": ["./config", "./calculator"],

    // ファイル保存時に自動でコード整形（黒いフォーマッタを使う）
    "editor.formatOnSave": true,

    // インデントを統一（Pythonでは4スペースが標準）
    "editor.tabSize": 4
}
```

設定をするメリット

- `.venv` が自動で認識される
- 補完機能が正確に動く（importエラーが減る）
- 保存するたびにコードが整う
- チーム開発でも見た目が統一される

今後、新しくアプリが増えていくなら

```
"python.analysis.extraPaths": ["./config", "./calculator", "./studylog"]
```