PROGRAMMAZIONE WEB

SERVER-SIDE SCRIPTING - DJANGO

Prof. Ada Bagozi
ada.bagozi@unibs.it

Cos'è il Django?

- ✓ *Django è un framework web open source scritto in Python*
- ✓ *Favorisce uno sviluppo rapido e sicuro*
- ✓ *Filosofia Batteries included – molte funzionalità integrate*

Caratteristiche principali

- ✓ *ORM integrato*

  semplifica l'interazione con i database, permettendo agli sviluppatori di interagire con le tabelle del database usando oggetti Python invece di SQL

- ✓ *Sistema di template potente (sintassi {% … %})*
- ✓ *Interfaccia di amministrazione automatica*
- ✓ *Sicurezza integrata (CSRF, XSS, SQL injection)*
- ✓ *Supporto per API REST (Django REST Framework)*

Architettura MVC (Laravel)

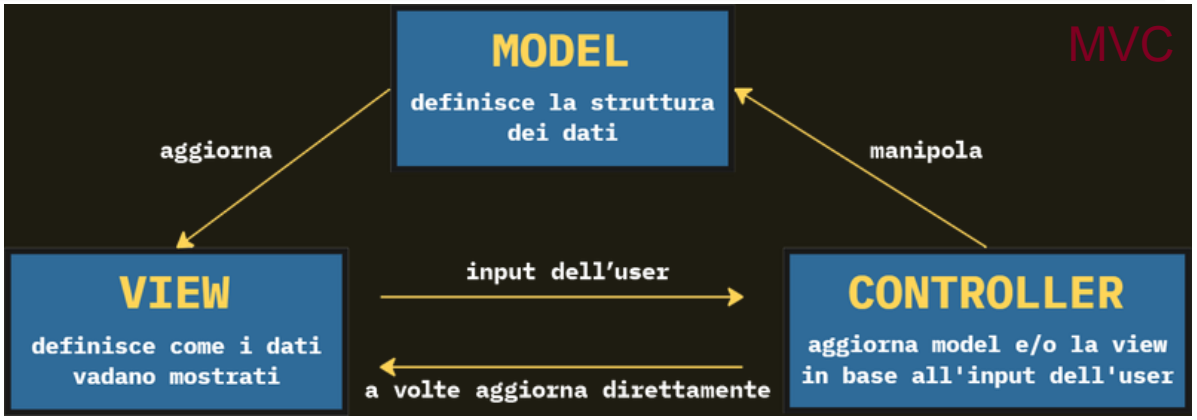
- ✓ Model: definisce la struttura dei dati, ovvero quali dati l'app debba contenere
- ✓ View: descrive come i dati dell'app vadano mostrati
- ✓ Controller: contiene la logica che aggiorna i Model e/o le View in risposta agli input forniti dall'utente

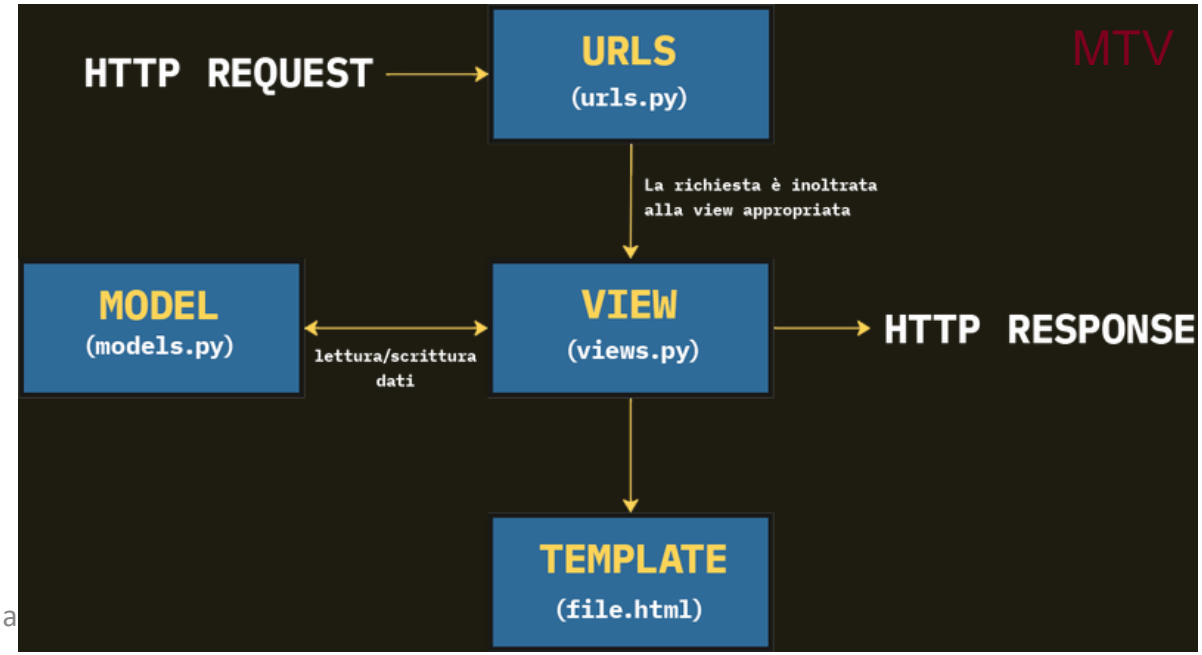
https://www.programmareinpython.it/blog/che-cosa-rende-django-speciale-miglior-web-framewo/

Architettura MTV (Django)

- ✓ Model: Definisce la struttura delle entità del nostro sito, tradotte poi in tabelle nel databas (data access layer)
- ✓ Template: Descrive come i dati vadano mostrati nelle pagine web (presentation layer)
- ✓ View: Gestisce le richieste e le risposte HTTP, dispone della logica per sapere a quali dati accedere tramite i model e delega la formattazione della risposa ai template (business logic layer)

MCV vs MTV


MVC
MODEL
definisce la struttura dei dati
aggiorna
manipola
VIEW
definisce come i dati vadano mostrati
input dell'user
a volte aggiorna direttamente
CONTROLLER
aggiorna model e/o la view in base all'input dell'user


MTV
HTTP REQUEST
URLS
(urls.py)
La richiesta è inoltrata alla view appropriata
MODEL
(models.py)
lettura/scrittura dati
VIEW
(views.py)
HTTP RESPONSE
TEMPLATE
(file.html)

Django vs. Laravel (I)

Criterio	Django (Python)	Laravel (PHP)
Linguaggio	Python – leggibile, usato anche in data science	PHP – molto diffuso nel web, facile da trovare hosting
Architettura	MTV (Model–Template–View) semplice e diretto	MVC (Model–View–Controller) ben noto e separazione netta
ORM	ORM integrato, potente e semplice da usare	Eloquent ORM, intuitivo e fluente
Sicurezza	Protezioni predefinite (CSRF, XSS, SQLi, clickjacking)	Protezioni buone, ma più configurabili manualmente
Admin	Interfaccia admin automatica out-of-the-box	Pacchetti come Laravel Nova (più personalizzabili)
CLI	manage.py – semplice, comandi ben strutturati	artisan – molto potente e ricco di comandi utili
Routing	Esplicito e leggibile	Elegante e supporta route closures
Template Engine	Django Template Language – chiaro e sicuro	Blade – flessibile, supporta estensioni e direttive personalizzate
APIs REST	Django REST Framework – completo e robusto	Laravel Sanctum, Passport – moderne soluzioni per API

Django vs. Laravel (II)

Criterio	Django (Python)	Laravel (PHP)
Performance	Efficiente, adatto a progetti complessi	PHP 8 ha migliorato molto le prestazioni
Ecosistema	Ottima integrazione con strumenti Python (es. AI, ML, scientific computing)	Ricco ecosistema di pacchetti Composer
Community	Attiva nel mondo Python, scientifico e accademico	Vibrante, tutorial/video abbondanti, grande supporto PHP
Hosting	Può richiedere configurazione su server (es. WSGI, ASGI)	Ampio supporto su hosting tradizionali (shared hosting, cPanel, ecc.)
Documentazione	Ufficiale molto dettagliata, chiara e autorevole	Chiara, con molti esempi, screencast e risorse community
Curva di apprendimento	Richiede familiarità con Python e il pattern MTV	Rapida per chi già conosce PHP

Installazione di Django

Install django and create project

```
pip install django
django-admin startproject my_project
cd project
python manage.py runserver
```

Create app

```
django-admin startapp my_app
```

```
my_project/
├── manage.py
├── my_project/
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── my_app/
    ├── migrations/
    │   └── __init__.py
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── models.py
    ├── tests.py
    ├── urls.py
    └── views.py
```

Directory principale del progetto

- ✓ È il nucleo dell'intero progetto Django
- ✓ Contiene i file di configurazione globali
- ✓ Organizza e coordina le funzionalità dell'intero sistema web

```
my_project/
├── manage.py
├── my_project/
│   ├── __init__.py
│   ├── asgi.py
│   ├─ settings.py
│   ├─ urls.py
│   ├── wsgi.py
├── my_app/
    ├── migrations/
    │   └── __init__.py
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── models.py
    ├── tests.py
    ├── urls.py
    └── views.py
```

File principali nella directory del progetto


```
django-admin startapp my_app
```

- ✓ manage.py: Interfaccia per i comandi Django (runserver, migrate, etc.)
- ✓ settings.py: Configurazione generale (database, app, middleware)
- ✓ urls.py: Dispatcher delle URL: collega URL a viste
- ✓ wsgi.py: Gateway per server di produzione (WSGI)
- ✓ asgi.py: Gateway asincrono per ASGI server
- ✓ init.py: Rende la directory un package Python

```
my_project/
├── manage.py
├── my_project/
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── my_app/
│   ├── migrations/
│   │   └── __init__.py
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── models.py
│   ├── tests.py
│   ├── urls.py
│   └── views.py
```

Directory di un'app Django

- ✓ Ogni app è un modulo indipendente all'interno del progetto
- ✓ Struttura generata automaticamente da startapp

```
my_project/
├── manage.py
├── my_project/
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── my_app/
    ├── migrations/
    │   └── __init__.py
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── models.py
    ├── tests.py
    ├── urls.py
    └── views.py
```

Directory di un'app Django

- ✓ models.py: Definizione dei dati tramite ORM
- ✓ views.py: Logica delle richieste/risposte
- ✓ admin.py: Configurazione dell'interfaccia admin
- ✓ tests.py: Test automatici
- ✓ migrations/: Tracciamento delle modifiche ai modelli
- ✓ apps.py: Configurazione dell'app (Opzionale)
- ✓ forms.py, urls.py, utils.py

```
my_project/
├── manage.py
├── my_project/
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── my_app/
    ├── migrations/
    │   └── __init__.py
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── models.py
    ├── tests.py
    ├── urls.py
    └── views.py
```

Il concetto di App Riutilizzabili

- ✓ Ogni app può essere:
  - ✓ Indipendente
  - ✓ Trasferibile tra progetti
  - ✓ Manutenibile separatamente
- ✓ Vantaggi:
  - ✓ Modularità e chiarezza
  - ✓ Codice riutilizzabile
  - ✓ Standardizzazione tra progetti
  - ✓ Facilità di aggiornamento
  - ✓ Contributi dalla community

```
my_project/
├── manage.py
├── my_project/
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── my_app/
    ├── migrations/
    │   └── __init__.py
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── models.py
    ├── tests.py
    ├── urls.py
    └── views.py
```

Esempio di gerarchia di progetto Django

- ✓ static/: file statici (CSS, JS, immagini)
- ✓ media/: contenuti caricati dagli utenti
- ✓ templates/: template HTML condivisi

```
myproject/
├── manage.py
├── myproject/
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── blog/
├── utenti/
├── static/
├── media/
└── templates/
```

Convenzioni di nomi

- ✓ App: minuscolo con underscore (es. my_app)
- ✓ File Python: minuscolo con underscore (es. views.py)
- ✓ Classi: CamelCase (es. MyModel)
- ✓ Variabili e funzioni: snake_case (es. get_data)

Buone pratiche di progettazione

- ✓ Separare responsabilità per app
- ✓ Usare Class-Based Views dove utile
- ✓ Inheritance nei modelli
- ✓ Inheritance nei template
- ✓ Modulo utils.py per funzioni comuni

Gestione delle impostazioni (settings.py)

- ✓ Utilizzare variabili di ambiente per credenziali e chiavi
- ✓ Separare settings per ambiente (sviluppo, produzione)
- ✓ Tenere DEBUG = False in produzione

Middleware in Django

- ✓ Il middleware è una serie di hook che processano la richiesta e la risposta
- ✓ Può essere usato per:
  - ✓ Autenticazione
  - ✓ Compressione delle risposte
  - ✓ Gestione della sessione
  - ✓ Protezione CSRF

```
MIDDLEWARE = [
  "django.middleware.security.SecurityMiddleware",
  "django.contrib.sessions.middleware.SessionMiddleware",
  "django.middleware.common.CommonMiddleware",
  "django.middleware.csrf.CsrfViewMiddleware",
  "django.contrib.auth.middleware.AuthenticationMiddleware",
  "django.contrib.messages.middleware.MessageMiddleware",
  "django.middleware.clickjacking.XFrameOptionsMiddleware",
]
```

Esempio di Middleware

Un middleware personalizzato per loggare ogni richiesta:

```
class LogMiddleware:
        def __init__(self, get_response):
                self.get_response = get_response

        def __call__(self, request):
                print(f"Request path: {request.path}")
                response = self.get_response(request)
                return response
```

Aggiungerlo in settings.py:

```
MIDDLEWARE = [
        'mia_app.middleware.LogMiddleware',
        ...
]
```

Controllo degli Accessi alle Viste

Usare il decoratore @login_required

```
from django.contrib.auth.decorators import login_required

@login_required
def area_riservata(request):
   return render(request, 'riservata.html')
```

Per le class-based views

```
from django.contrib.auth.mixins import LoginRequiredMixin
from django.views.generic import TemplateView

class AreaRiservataView(LoginRequiredMixin, TemplateView):
  template_name = 'riservata.html'
```

Se l'utente non è autenticato, verrà reindirizzato alla login.

Modelli in Django (models.py)

- ✓ Ogni modello è una classe Python che rappresenta una tabella nel database
- ✓ Utilizza il sistema ORM integrato per creare/modificare dati

```python
from django.db import models


class Author(models.Model):
  first_name = models.CharField(max_length=30)
  last_name = models.CharField(max_length=30)
```

```
CREATE TABLE myapp_author (
  "id" bigint NOT NULL PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
  "first_name" varchar(30) NOT NULL,
  "last_name" varchar(30) NOT NULL
);
```

Le modifiche ai modelli si riflettono nel database tramite makemigrations e migrate

https://docs.djangoproject.com/en/5.2/topics/db/models/

Configurazione del Database

- ✓ In settings.py, sezione DATABASES

```
DATABASES = {
  "default": {
                    "ENGINE": "django.db.backends.sqlite3",
                    "NAME": BASE_DIR / "db.sqlite3",
    }
}
```

- ✓ Altri motori disponibili:
  - ✓ PostgreSQL: 'django.db.backends.postgresql'
  - ✓ MySQL: 'django.db.backends.mysql'
  - ✓ Oracle: 'django.db.backends.oracle'

Modello Book


```
class Book(models.Model):
        title = models.CharField(max_length=100)
        author = models.ForeignKey(Author, on_delete=models.CASCADE)
        publication_date = models.DateField()
        isbn = models.CharField(max_length=13, unique=True)
        pages = models.IntegerField()
        cover_image = models.ImageField(upload_to='covers/')
        language = models.CharField(max_length=30)
        summary = models.TextField()
        genre = models.CharField(max_length=30)
```

Esempi di accesso ai dati


```
#Creare un oggetto:
autore = Author.objects.create(first_name="Umberto", last_name="Eco")

#Recuperare tutti i libri:
libri = Book.objects.all()

#Recuperare uno specifico libro
libro = Book.objects.get(id=1)

#Filtrare
libro = Book.objects.filter(title="Il nome della rosa").first()

#Ordinare
libri = Book.objects.order_by('titolo')
#Eliminare
libro = Book.objects.get(id=1)
libro.delete()
```

Versionamento del codice e sicurezza

- ✓ Usare Git per versionare
- ✓ Includere un .gitignore
- ✓ Evitare di tracciare:
  - ✓ file .pyc
  - ✓ cartelle __pycache__
  - ✓ cartelle env/, .venv/
  - ✓ db.sqlite3, se locale
  - ✓ …

PROGRAMMAZIONE WEB

SERVER-SIDE SCRIPTING - DJANGO

Prof. Ada Bagozi
ada.bagozi@unibs.it