MVC

Model/view/controller

O que é?

É um **padrão de arquitetura de software** usado para organizar projetos, separando responsabilidades em três partes principais:

Model (Modelo)

- Representa os **dados** e as **regras de negócio**.
- É onde ficam as informações que o sistema manipula (por exemplo: usuários, produtos, pedidos).
- Ele **não sabe nada sobre a interface**, só sobre os dados.

Exemplo:

Uma classe `Usuário` com atributos (nome, email, senha) e métodos (validar Senha, cadastrar).

View (Visão)

- É a parte **visual**, o que o usuário vê e interage.
- Mostra os dados que vêm do *Model*.
- Não deve conter lógica de negócio, só de **apresentação**.

Exemplo:

Um formulário HTML, uma tela em um app, ou uma página que exibe a lista de produtos.

Controller (Controlador)

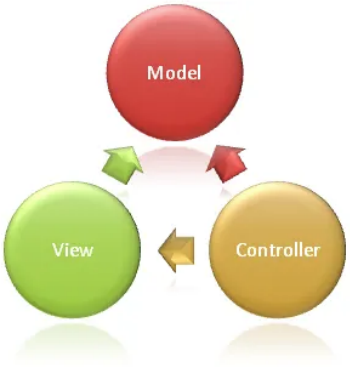
- Faz a **ponte entre o usuário, a View e o Model**.
- Recebe as ações do usuário (cliques, formulários enviados), processa e decide o que fazer.
- Ele pede para o *Model* atualizar/buscar dados e depois manda para a *View* exibir.

Exemplo:

Um `Usuário Controller` que recebe o pedido de cadastro, chama o `Model` para salvar e redireciona para a `View` de sucesso.

Como funciona o ciclo:

1. O **usuário interage** com a View (ex: clica em "Salvar").
2. A **View envia a ação** para o Controller.
3. O **Controller processa** a ação e chama o Model.
4. O **Model atualiza ou busca dados**.
5. O **Controller passa os dados para a View**.
6. A **View exibe o resultado** para o usuário.



O Django segue o padrão **MVC**, mas com uma adaptação que eles chamam de **MTV (Model – Template – View)**.

```
# Django MVT - Updating data
person = Person.objects.get(id=1)
person.age = 30
person.save()

# Flask MVC - Updating data
person = Person.query.get(1)
person.age = 30
db.session.commit()
```

Na prática, é quase a mesma coisa, só muda o nome de algumas partes.

Estrutura no Django

Model (Modelo)

- Fica em `models.py`.
- Define as tabelas do banco de dados (usando classes do Django).

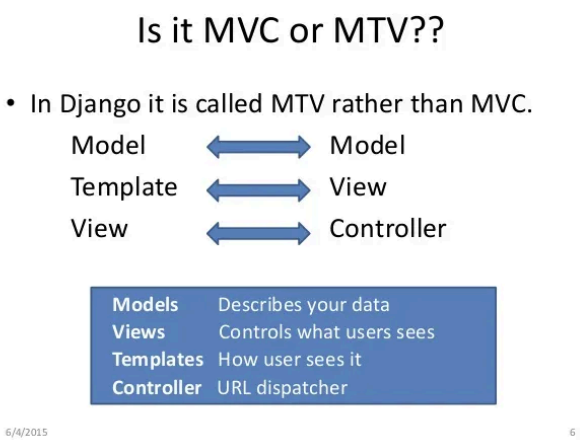
- Contém regras de negócio e manipulação dos dados.

Template (equivale à View no MVC)

- Fica na pasta `templates/`.
- É o HTML (ou outra forma de exibir dados) que o usuário vê.
- Pode receber dados enviados pela **View** do Django.

View (no Django, é o Controller do MVC)

- Fica em `views.py`.
- Recebe a requisição do usuário (URL), acessa o **Model**, e passa os dados para o **Template**.



Referências:
https://medium.com/shecodeafrica/understanding-the-mvc-pattern-in-django-edda05b9f43f
https://pt.stackoverflow.com/questions/246881/qual-a-diferen%C3%A7a-entre-a-arquitetura-mvc-e-a-mtv-do-django

---

O que é um framework?

- É uma estrutura (conjunto de classes, objetos, padrões) que oferece ferramentas, guias e componentes para facilitar o desenvolvimento de software.
- Ajuda a evitar começar do zero, pois já provê funcionalidades comuns.
- Diferença importante: *framework* vs *biblioteca*. A biblioteca é usada sob demanda, enquanto o framework dita mais como a aplicação será estruturada.

Para que servem

- Resolver problemas recorrentes de forma padronizada.
- Ajudar a acelerar o desenvolvimento, garantir mais segurança, reutilização de código

Vantagens de usar frameworks

- Ganho de tempo: há funcionalidades prontas ou semiprontas.
- Comunidade ativa + documentação, facilita aprendizado e solução de problemas.
- Segurança: correções de vulnerabilidades são feitas pelos mantenedores.
- Padronização de código, menos bugs, custos operacionais menores

Oque é um django? Características principais:

Django é um **framework web** escrito em **Python** que ajuda a criar sites e aplicações de forma rápida, organizada e segura. Ele segue o padrão **MTV (Model–Template–View)**, que é uma variação do MVC

- **Rápido para desenvolver:** Vem com várias funcionalidades prontas (autenticação, administração, ORM, formulários).
- **Seguro:** Protege contra ataques comuns (SQL Injection, XSS, CSRF, etc.).
- **Escalável:** Suporta desde pequenos sites até grandes aplicações (ex: Instagram começou com Django).
- **ORM (Object Relational Mapper):** Permite interagir com o banco de dados usando Python em vez de SQL direto.
- **Admin automático:** Gera um painel administrativo completo baseado nos Models.
-