

Tecnologias e Programação Web

José Mendes 107188

2023/2024

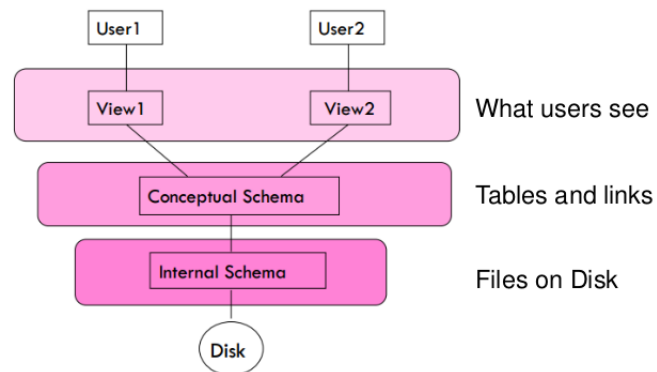


universidade
de aveiro

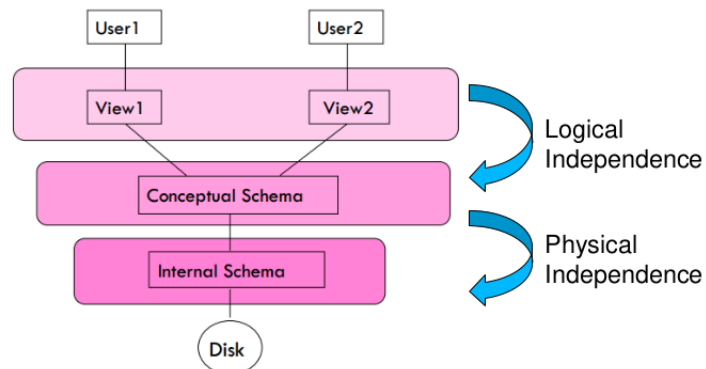
1 Arquiteturas de Aplicações Web

1.1 Independência dos Dados nas Bases de Dados

1.1.1 Arquitetura da Base de Dados e Views



1.1.2 Independência Lógica e Física



Cada nível é independente dos níveis a baixo.

1.1.3 Independência dos Dados

Independência Física: Capacidade de alterar o schema interno sem alterar o schema conceptual.

- Espaço de armazenamento pode mudar;
- O tipo de alguns dados pode mudar por razões de eficiência/otimização;

Independência Lógica: Capacidade de alterar o schema conceptual sem alterar as Views ou os programas de aplicação.

- Pode adicionar novos campos, novas tabelas sem alterar as Views;
- Pode alterar a estrutura das tabelas sem alterar as Views;

Nota: Manter a **View** (aquilo que o user vê) independente do **Modelo** (domain knowledge).

1.2 Arquiteturas N-Tier

1.2.1 Significado de "Tiers"

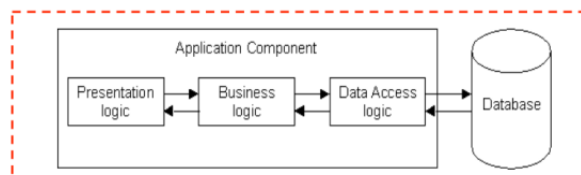
Arquiteturas N-Tier têm as mesmas camadas:

- **Presentation Tier:** Camada de apresentação;
- **Business/Logic Tier:** Camada de lógica/de negócio;
- **Data Tier:** Camada de dados;

Arquiteturas N-Tier tentam separar as camadas em diferentes tiers.

- **Camada:** Separação lógica;
- **Tier:** Separação física;

1.2.2 Arquitetura 1-Tier

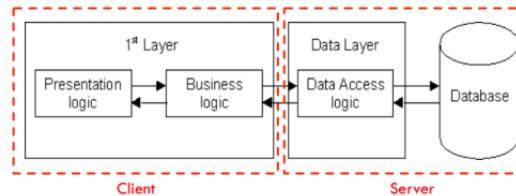


As 3 camadas estão no mesmo computador (máquina). Isto é, todo o código e processamento está numa única máquina.

As camadas de apresentação, lógica e de dados estão firmemente conectados.

- **Escalabilidade:** Um único processador implica que é difícil de aumentar o volume de processamento;
- **Portabilidade:** Mudar para outro computador pode implicar reescrever tudo;
- **Manutenção:** Mudar uma camada requer mudar outras camadas;

1.2.3 Arquitetura 2-Tier

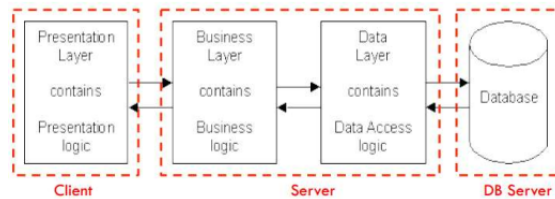


A base de dados corre no servidor, separado do cliente, sendo fácil de mudar para uma base de dados diferente.

As camadas de apresentação e lógica ainda estão firmemente conectados.

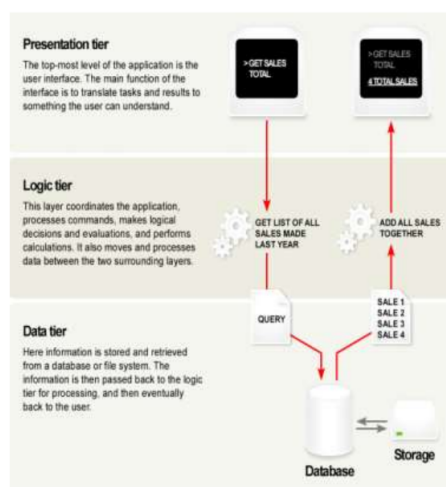
- Carga pesada no servidor;
- Potencial congestionamento da rede;
- Apresentação ainda está firmemente conectada à lógica de negócio;

1.2.4 Arquitetura 3-Tier



Cada camada pode, potencialmente, correr numa máquina diferente. As camadas de apresentação, lógica e de dados estão desconectadas.

Típica Arquitetura 3-Tier



Princípios de Arquitetura:

- Arquitetura **Cliente-Servidor**;
- Cada camada (Apresentação, Lógica, Dados) deve ser independente e não deve expor dependências relacionadas com a implementação;
- Camadas desconectadas não devem comunicar;
- Alterações numa plataforma afetam apenas a camada que corre que essa plataforma;

Camada de Apresentação:

- Fornece a interface do utilizador;
- Lida com a interação com o utilizador;
- Por vezes chamada de GUI, client view ou **front-end**;
- Não deve conter lógica de negócio ou código de acesso aos dados;

Camada de Lógica:

- O conjunto de regras para processar a informação;
- Pode acomodar vários utilizadores;
- Por vezes chamada de **middleware** ou back-end;
- Não deve conter apresentação ou código de acesso aos dados;

Camada de Dados:

- A camada de armazenamento físico, para persistir os dados;
- Gere o acesso à base de dados ou sistema de ficheiros;
- Por vezes chamada de **back-end**;
- Não deve conter apresentação ou código da lógica de negócio;

1.2.5 Arquitetura 3-Tier para Aplicações Web

Camada de Apresentação: Conteúdo estático ou dinâmico, renderizado pelo browser (**front-end**);

Camada de Lógica: Processamento de conteúdo dinâmico, geração de servidores de aplicação (e.g. Java EE, ASP.NET, Python Django Framework) (**middleware**);

Camada de Dados: Base de dados, composta por ambos, conjunto dos dados e o sistema de gestão de base de dados ou software RDBMS que gere e fornece acesso aos dados (**back-end**);

1.2.6 Arquitetura 3-Tier - Vantagens

Independência das camadas

- Mais fácil de manter;
- Componentes são reutilizáveis;
- Desenvolvimento mais rápido (divisão do trabalho, Web Designer faz a apresentação, Engenheiros de Software fazem a lógica e administradores de base de dados fazem o modelo de dados);

1.3 Padrões de Desenho

1.3.1 Padrões de Desenho e Decisões

- Construção e teste;
 - como construímos uma aplicação web?
 - que tecnologias devemos escolher?
- Reutilizar;
 - podemos usar componentes normais?
- Escalabilidade;
 - como vai a nossa aplicação web lidar com um elevado número de pedidos?
- Segurança;
 - como proteger contra um attack, vírus, acesso malicioso dos dados, denial of service?
- Diferentes views de dados;
 - tipos de users, contas individuais, proteção de dados

Nota: Precisamos de uma solução geral e reutilizável: **Padrões de Desenho**

1.3.2 O que é um Padrão de Desenho?

Uma solução geral e reutilizável para um problema recorrente no desenho de software. É um template para como resolver um problema que tenha sido usado em várias situações diferentes. **NÃO** é um design completo, o padrão precisa de ser adaptado à aplicação, não podemos simplesmente traduzir o padrão para código.

1.3.3 Origem do Padrão de Desenho

- Arquitetura conceptual por Christopher Alexander (1977/1979).
- Adaptado para programação OO por Kent Beck e Ward Cunningham (1987).
- Ganhou popularidade em CS com o livro "Design Patterns: Elements of Re-usable Object-Oriented Software" (1994), por Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides (Gang of Four).
- Agora bastante utilizado em Engenharia de Software.

1.4 O Padrão de Desenho MVC

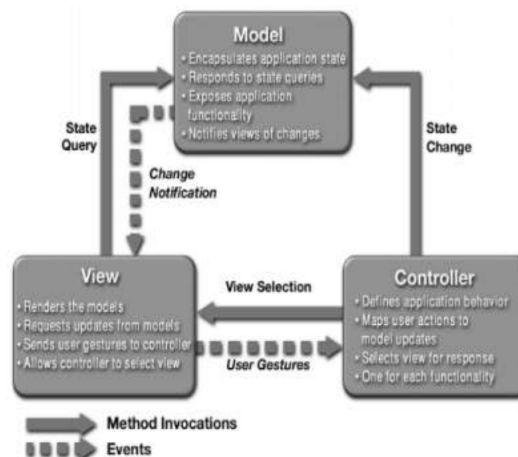
1.4.1 O Problema do Desenho

- Preciso mudar o look-and-feel sem mudar o core/logic;
- Preciso de apresentar os mesmos dados de diferentes formas (e.g. computadores bons, web, dispositivos móveis);
- Preciso de interagir/acessar os dados de diferentes formas (e.g. ecrã tátil nos dispositivos móveis, teclado no computador);
- Preciso de manter várias views para a mesma informação (e.g lista, thumbnails, detalhes, ...);

1.4.2 A Solução do Desenho

- Separar a funcionalidade chave da apresentação e lógica que usa esta funcionalidade;
- Permitir múltiplas views para partilhar o mesmo modelo de dados;
- Tornar o suporte de vários clientes mais fácil de implementar, testar e manter;

1.4.3 O Padrão Model-View-Controller



Padrão de Desenho para sistemas gráficos que **promove a separação entre o modelo e a view**. Com este padrão, a lógica necessária para manutenção dos dados (base de dados, ficheiro de texto), **é separada** de como os dados são apresentados (gráfico, numerico) e como os dados são manipulados (GUI, linha de comandos, touch).

Model

- Gere o comportamento e os dados do domínio da aplicação;
- Responde a pedidos por informação sobre o estado (normalmente da view), segue instruções para alterar o estado (normalmente do controller);

View

- Renderiza o modelo para uma forma apropriada para a interação, tipicamente uma interface do utilizador (várias views podem existir para o mesmo modelo com diferentes intenções);

Controller

- Recebe os inputs e inicia uma resposta realizando chamadas em objetos do modelo;
- Aceita input do utilizador e intrui o modelo e a view para realizar ações baseadas no input;

1.4.4 O Padrão MVC na prática

Model

- Contém conhecimento específico do domínio;
- Regista o estado da aplicação (e.g quais items estão no carrinho de compras);
- Normalmente conectado com a base de dados;
- Independente da view (um model pode ter várias views);

View

- Apresenta dados ao utilizador;
- Permite interação com o utilizador;
- Não faz processamento;

Controller

- Define como a interface do utilizador reage a inputs (eventos);
- Recebe mensagens da view (de onde os eventos vêm);
- Envia mensagens ao modelo (diz quais dados mostrar)

1.4.5 O MVC para Aplicações Web

Model

- Tabelas da base de dados (dados persistentes);
- Informações sobre a sessão (dados atuais do sistema de dados)
- Regras sobre transações;

View

- (X)HTML;
- CSS style sheets;
- Templates server-side;

Controller

- Scripts client-side;
- Processamento de pedidos HTTP;
- Lógica de negócio/preprocessamento;

1.4.6 MVC - Vantagens

Clareza de Design

- métodos do modelo dão uma API para os dados e o estado;
- o design da view e do controller são fáceis;

Modularidade eficiente

- qualquer componente pode ser facilmente substituído;

Várias views

- Várias views podem ser criadas conforme apropriado;
- Cada uma usa a mesma API para o modelo;

Mais fácil de contruir e manter

- Views simples (baseado em texto) durante o desenvolvimento (construção);
- Mais views e controllers podem ser adicionados mais tarde;
- Fácil de contruir interfaces estáveis;

Distribuível

- Ajuste natural para ambientes distribuídos;

1.4.7 3-Tier Architecture vs Padrão MVC

- Comunicação
 - **3-Tier:** A camada de apresentação nunca comunica diretamente com a camada de dados, apenas através da camada de lógica (topologia linear);
 - **MVC:** Todas as camadas comunicam diretamente (topologia triangular);
- Usabilidade
 - **3-Tier:** É usada em aplicações web onde os tiers cliente, middleware e os dados corram em plataformas fisicamente separadas;
 - **MVC:** Historicamente usada em aplicações que correm numa única estação de trabalho gráfica;
 - * No contexto de aplicações web, contudo, os componentes lógicos podem ser desacoplados para cumprir com uma verdadeira arquitetura 3-Tier;

2 Introdução à Plataforma Django

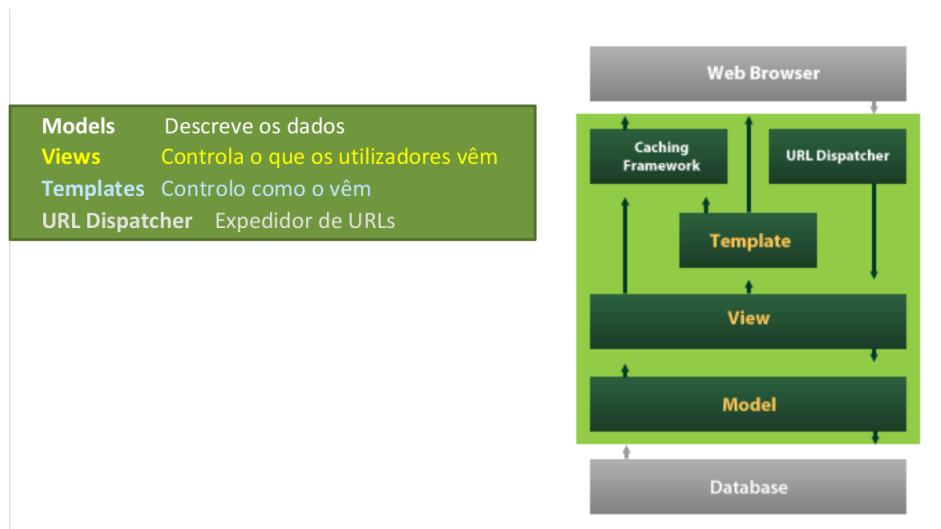
2.1 Introdução

Django é uma plataforma gratuita e open-sourc, escrita em Python, para o desenvolvimento de aplicações web. Tem o nome do famoso guitarrista de jazz Django Reinhardt. É mantida pela Django Software Foundation (DSF), uma organização independente. Fomenta o desenvolvimento rápido, limpo e pragmático. Criada em 2003, lançada open-source em 2005.

2.2 Características

- Segue, parcialmente, o padrão MVC;
- Possui um ORM (Object-Relational Mapper) para processar dados;
- Focada na automatização, aderindo ao princípio DRY (Don't Repeat Yourself);
- Usa um sistema de templates;
- Sistema de personalização Admin, para facilitar o CRUD;
- Desenho elegante de routing de URLs;
- Possui um light web server embotido (para testes);
- Possibilita a utilização de moddleware personalizado;
- Possui facilidades para: autenticação, internacionalização, caching;

2.3 Arquitetura



2.4 Estrutura de um Projeto Django

```
webproj/ ----- Pasta para o projeto. Pode ter qualquer nome.
manage.py -- Utilitário em commando de linha para interagir com o projeto.
webproj/ --- Pacote do projeto. Nome usado para imports.
__init__.py --- Ficheiro que define esta pasta como um pacote, em Python.
settings.py --- Configurações do projeto Django.
urls.py ----- Mapping/routing das URLs para este projeto.
wsgi.py ----- Um ponto de entrada para webserver compatíveis com WSGI.
app/ ----- Aplicação web individual, podendo coexistir várias.
templates/ ---- Ficheiros HTML, invocados pelas views.
static/ ----- CSS, JS, imagens, etc. - configurável em "settings.py"
__init__.py -- Ficheiro que define esta pasta como um pacote, em Python.
views.py ----- Recebe os pedidos dos clientes e devolve as respostas.
models.py ----- Modelos dos dados.
admin.py ----- Criação automática de interface para o modelo de dados.
forms.py ----- Permite a receção de dados enviados pelos clients.
```

2.5 Settings

Possui um ficheiro de configuração, **settings.py**, do projeto Django, sobrepõe as configurações padrão (ficheiro `python/lib/sitepackages/django/conf/global_settings.py`).

Possui alguns atributos como: **DEBUG** (True/False), **DATABASES ENGINES** (sqlite3, mysql, postgresql, oracle), **ROOT_URLCONF** (nome do ficheiro de routing de URLs), **MEDIA_ROOT** (para ficheiros enviados pelo utilizador), **MEDIA_URL** (para ficheiros multimédia), **STATIC_ROOT** (pasta para ficheiros estáticos, CSS, JS), **STATIC_URL** (pasta de ficheiros estáticos), **TEMPLATE_DIRS** (pasta de templates)

2.6 Criação de um Projeto Django

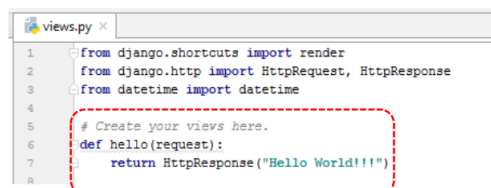
Nesta cadeia vamos usar o PyCharm, pelo que para criar um projeto Django basta seleccionar a opção **Django** e chamar a pasta dos templates de **app/templates** e a pasta da aplicação de **app**.

Para correr basta clicar em Run e aceder ao link **http://127.0.0.1:8000**.

Ou podemos usar o comando **python manage.py runserver**.

2.6.1 Views

No ficheiro "app/views.py" podemos inserir views:



```
1 from django.shortcuts import render
2 from django.http import HttpRequest, HttpResponse
3 from datetime import datetime
4
5 # Create your views here.
6 def hello(request):
7     return HttpResponse("Hello World!!!")
8
```

Para criar uma nova, basta criar uma nova view function.

2.6.2 Configuração da URL

No ficheiro "nome_do_projeto/urls.py" podemos configurar as URLs:



```
14 2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
15
16 from django.contrib import admin
17 from django.urls import path
18
19 from app import views
20
21 urlpatterns = [
22     path('hello/', views.hello, name='hello'),
23 ]
```

Para criar uma nova, basta inserir mais uma route para a view.

2.6.3 Templates

Podemos usar variáveis usando `{{ var_name }}` e template tags (como if) usando `{% if ... %}`.

2.6.4 Ficheiros estáticos

São ficheiros que se pretende simplesmente referenciar e servir ao cliente, sem qualquer processamento prévio.

O seu acesso é público, pois o cliente apenas necessita do URL para os mesmos.

Exemplos:

- Imagens (jpg, png, gif, ...);
- Style Sheets (CSS);
- Scripts (JavaScript);
- ...

Localização

Os ficheiros denominados por static files residem em pastas pré-determinadas, dentro ou fora da “app” (normalmente uma pasta denominada de “static”).

Configuração

No ficheiro “settings.py” podemos configurar os ficheiros estáticos:

1. o módulo ‘django.contrib.staticfiles’ deve aparecer nas aplicações instaladas “INSTALLED_APPS”
2. o atributo STATIC_URL deve ser definido (ex: ‘/static/’)
3. a pasta dos static files deve estar definida no atributo STATIC_ROOT (ex: `os.path.join(BASE_DIR, 'app/static')`)

Uso

Para usar basta usar a tag `{% load static %}` e depois referenciar o ficheiro.