

Django

**Introdução, Projetos e aplicações,
rotas, views e templates**

Ely – elydasilvamiranda@gmail.com

Django

- Framework de alto nível feito em Python
- Framework Full Stack:
 - desenvolve-se todos os aspectos de uma aplicação somente com ele.
- Criado em 2003 e disponibilizado como Opensource;
- Ágil e plugável: foco na automação e nos princípios DRY (Don't Repeat Yourself);

Principais elementos

- ORM: Mapeamento Objeto-Relacional;
- Interface administrativa;
- Internacionalização;
- Sistema de templates;
- Infraestrutura de Cache;
- Validação de formulários;
- Gerenciador de perfis, autorização e autenticação.

Verificando a versão

- Verificando a versão instalada:

```
>>> import django
```

```
>>> django.get_version()
```

ou:

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

Instalando o Django

- Para instalar o Django usaremos o pip;
- O pip é a ferramenta de instalação de pacotes do Python;
- Baixa-se diretamente da web pacotes com as mais diversas funcionalidades;
- Sintaxe:

```
pip install nome_do_pacote==versão
```
- Instalando o Django:

```
>>> pip install django==2.0
```
- Caso exista outra versão, desinstale-a primeiro:
- ```
>>> sudo pip uninstall django
```

# Criando um projeto

- Em Django devemos criar projetos;
- Projetos podem ter mais de uma aplicação;
- Para nosso estudo, criaremos:
  - Um projeto chamado **connectedin**;
  - Uma aplicação inicial chamada **perfis**;
- Deve-se utilizar o comando `django-admin.py` para criar projetos.

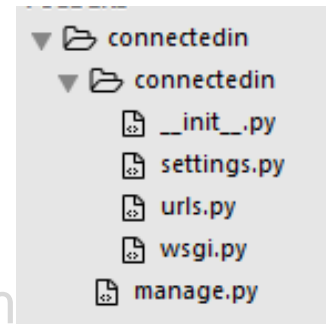
*Nota: esse arquivo deve estar no path do S.O.*

# Criando um projeto

- Sintaxe:

`django-admin.py startproject <nome_projeto>`

- Os nomes de projetos e aplicações não devem usar palavras reservadas da linguagem ou nomes específicos do django;
- Criando um projeto chamado `connectedin`:  
`>>> django-admin startproject connectedin`
- Ao criar um projeto, cria-se uma pasta com o nome do projeto:



# Criando um projeto

- Sobre as pastas e arquivos criados:
- `connectedin`: pasta onde o projeto está guardado;
  - `connectedin`: projeto em si que não deve ser renomeada;
    - `__init__.py`: arquivo vazio (indica um package);
    - `settings.py`: arquivo de configuração do projeto;
    - `urls.py`: definições de URLs do projeto;
    - `wsgi.py`: protocolo parecido com fastCGI serve HTTP;
  - `manage.py`: utilitário semelhante ao `django-admin.py`.



# Criando o banco

- O `manage.py` é um script para gerenciar a aplicação com Django;
- Deve ser executado através do interpretador python dentro da pasta do projeto;
- O próximo passo é configurar o BD do projeto;
- Por padrão, o Django vem com o SQLite;
- Na pasta do projeto digite:  

```
>>> python manage.py migrate
```
- O arquivo `connectedin/db.sqlite3` então representa o arquivo do banco.

# Configs. de banco

- O tipo de banco são configuradas no arquivo settings.py:

```
trecho do arquivo conf.py
```

```
DATABASES = {
 'default': {
 'ENGINE': 'django.db.backends.sqlite3',
 'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
 }
}
```

- Backends suportados por padrão:
  - 'django.db.backends.sqlite3', 'django.db.backends.postgresql', 'django.db.backends.mysql' e 'django.db.backends.oracle'

# Configs. de banco

- Backends disponibilizados por terceiros:
  - <https://docs.djangoproject.com/en/2.0/ref/databases/#third-party-notes>
- Configurando usuários e senhas:
  - <https://docs.djangoproject.com/en/2.0/ref/settings/#std:setting-DATABASES>

# Ajuda do manage.py

- Para obter ajuda de algum comando do manage.py, utilize:

`manage.py help <comando>`

Ex:

`manage.py help migrate`

# Testando o projeto

- O Django possui um servidor web interno para ser usado no ambiente de desenvolvimento;

*Nota: esse é um servidor de testes, não devendo ser usado em "produção"*

- O servidor local possui:
  - recarga automática de módulos;
  - serve os arquivos estáticos (javascripts, css, imagens, etc.) sem configurações adicionais;
- O comando para executa-lo é:  
`>>> python manage.py runserver porta.`

*Onde a porta é opcional, por padrão é a 8000*

# Testando o projeto

- Usa-se o comando runserver para "subir" o nosso projeto:

```
>>> python manage.py runserver
```

```
D:\temp\connectedin>python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
March 21, 2017 - 21:01:02
Django version 1.7.4, using settings 'connectedin.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[21/Mar/2017 21:01:14] "GET / HTTP/1.1" 200 1759
[21/Mar/2017 21:01:14] "GET /favicon.ico HTTP/1.1" 404 1932
```

- Após isso, deve-se testar a aplicação pelo navegador: <http://localhost:8000>

[django](#)

[View release notes for Django 2.0](#)



The install worked successfully! Congratulations!

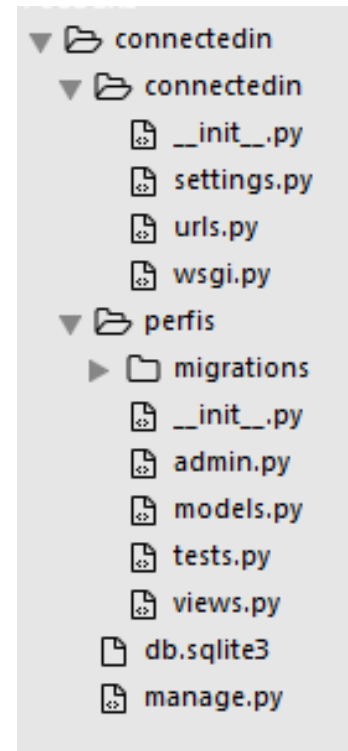
You are seeing this page because `DEBUG=True` is in your settings file and you have not configured any URLs.

# Criando uma aplicação

- Um projeto pode ter várias aplicações;
- Uma aplicação no Django:
  - é uma forma de dividir a responsabilidade dentro do projeto;
  - é um módulo que confina dentro dele determinada responsabilidade;
- As aplicações ficam armazenadas dentro da pasta do projeto.

# Criando uma aplicação

- Para criar uma nova aplicação, deve-se entrar no diretório do projeto pelo prompt de comando;
- Aplicações são criadas através do comando:  
`python manage.py startapp <nome_aplicação>`
- Criando uma aplicação chamada perfis:  
`>>> python manage.py startapp perfis`





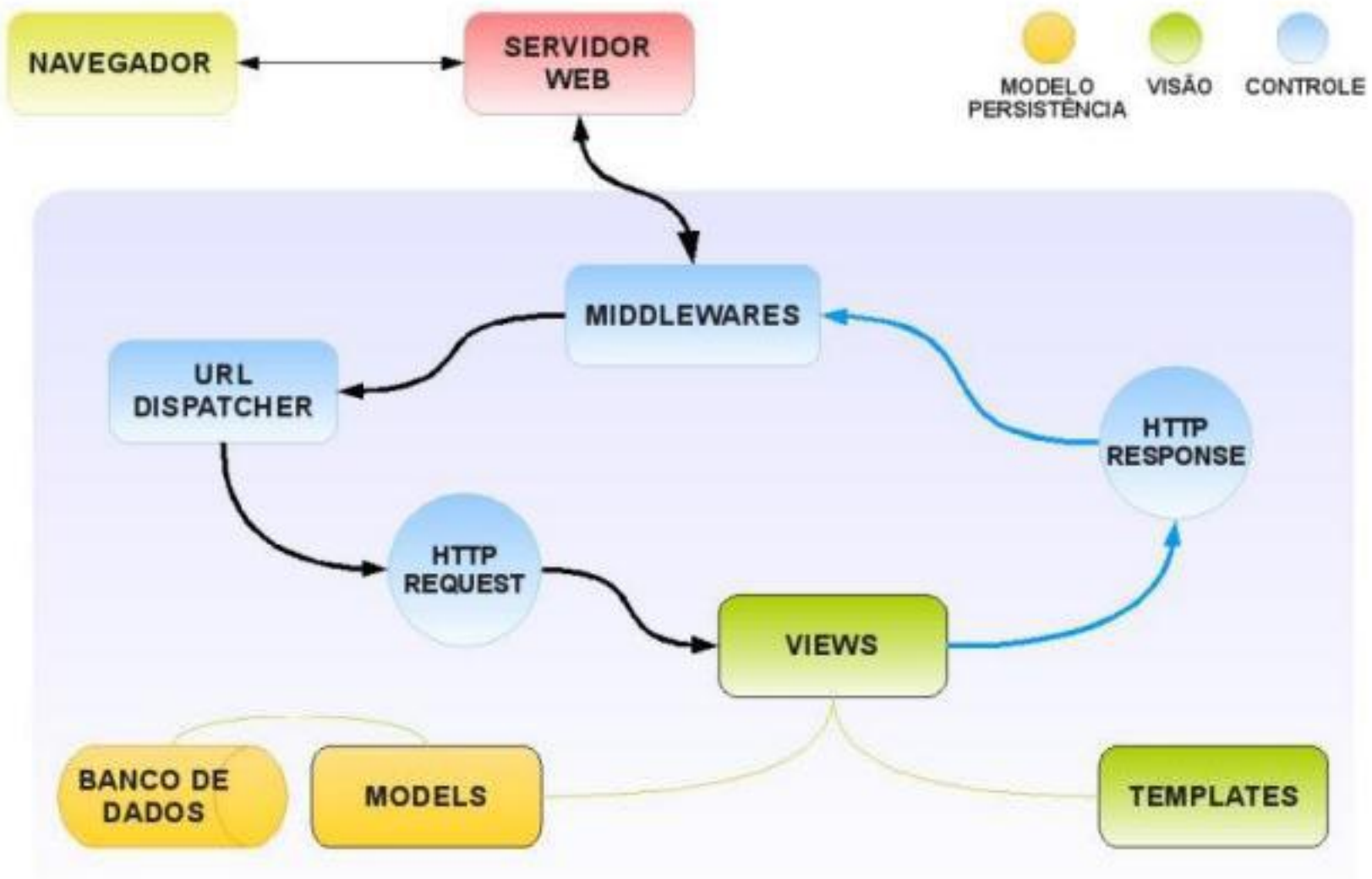
# Registrando uma aplicação

- Deve-se registrar a aplicação no arquivo settings.py do projeto;
- Nesse arquivo, há uma declaração chamada `INSTALLED_APPS`;
- Cada nova aplicação deve ser adicionada como o último elemento usando aspas simples:

```
código anterior omitido
INSTALLED_APPS = (
 'django.contrib.admin',
 'django.contrib.auth',
 'django.contrib.contenttypes',
 'django.contrib.sessions',
 'django.contrib.messages',
 'django.contrib.staticfiles',
 'perfis',
)
```

```
código posterior omitido
```

# Diagrama de requisições



# Views

- Views é um módulo python que agrupa um conjunto de actions;
- Toda view deve:
  - receber um objeto “HTTPRequest” como primeiro parâmetro;
  - retornar um objeto “HTTPResponse” como resposta.
- O objeto “HTTPRequest” é fornecido automaticamente pelo Django;
- O objeto “HTTPResponse” é de responsabilidade do desenvolvedor.

# Criando uma view

- Dentro de cada aplicação do projeto existe um arquivo chamado views.py;
- Nesse arquivo definimos a "resposta" para o usuário após uma requisição via browser;
- Para definir uma view, deve-se editar esse arquivo:

```
connectedin/perfis/views.py
```

```
from django.shortcuts import render
from django.http import HttpResponse
```

```
def index(request):
 return HttpResponse('Bem-vindo ao Connectedin')
```

# Criando uma view

- Sobre o código anterior:
  - A função index (poderia ser qualquer nome) representa uma página a ser acessada;
  - Recebe um parâmetro chamado request, que representa a requisição do usuário;  
(<https://docs.djangoproject.com/en/2.0/ref/request-response/#django.http.HttpRequest>)
  - O objeto responsável pela resposta deve ser importado e instanciado;
  - A resposta contém um texto de boas vindas;
- Para acessar essa função, deve-se especificar uma rota através do sistema/módulo de rotas do Python.

# Urls

- Urls é um módulo python responsável por realizar o roteamento de URLs do projeto;
- Todas as urls podem ficar em um único arquivo `urls.py`;
- É recomendável que cada aplicação contenha seu próprio arquivo `urls.py`;
- Posteriormente o arquivo `urls.py` do projeto deve importar os módulos `urls.py` de cada aplicação.

# Definindo uma URL

- As rotas do projeto são definidas no arquivo `connectedin/connectedin/urls.py`;
- Deve-se adicionar uma rota a mais ao final do arquivo:

```
connectedin/connectedin/urls.py
```

```
from django.urls import path
from django.contrib import admin
from perfis import views
```

```
urlpatterns = [
 path('admin/', admin.site.urls),
 path('', views.index),
]
```

# Definindo uma URL

- Argumentos da função URL:
  - O primeiro parâmetro é o caminho acessado via navegador;
  - O segundo é uma view que deve ser executada;
  - Existe um terceiro parâmetro chamado name, que será visto posteriormente;
- Mais sobre URLs:
  - <https://docs.djangoproject.com/en/2.0/topics/http/urls/>



# Testando a view

- Rode a aplicação:  
`>>> python manage.py runserver`
- Acesse o endereço:  
`http://localhost:8000/`

# Templates

- Devolvemos anteriormente apenas um texto "solto";
- O ideal é devolver uma página como resposta ao usuário;
- O Django padroniza que as páginas devem ficar em uma pasta chamada "templates";
- Dessa forma, devemos criar a pasta e nela criar uma página chamada index.html.

# Templates

```
<!-- connectedin/perfis/templates/index.html -->
```

```
<!DOCTYPE html>
```

```
<html lang="pt-br">
```

```
 <head>
```

```
 <meta charset="utf-8">
```

```
 <title>ConnectedIn</title>
```

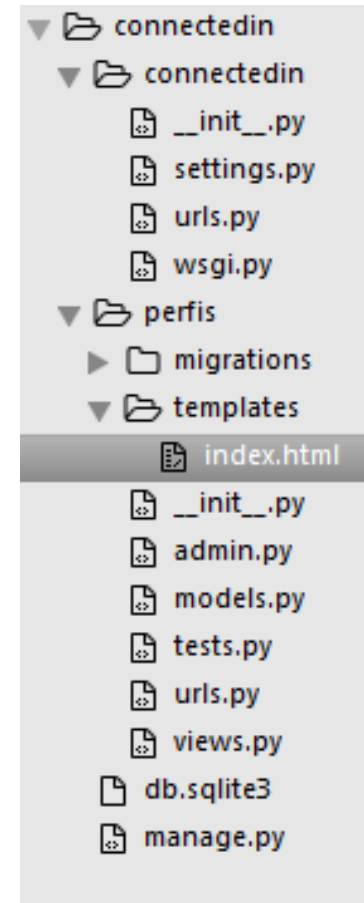
```
 </head>
```

```
 <body>
```

```
 <h1>Bem-vindo ao Connectedin</h1>
```

```
 </body>
```

```
</html>
```



# Templates

- Ao retornar um "template", deve-se alterar o métodos da view que o referencia;
- Deve-se "renderizar" uma página, passando como parâmetro o nome do arquivo:

```
connectedin/perfis/views.py
```

```
from django.shortcuts import render
```

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

# Páginas dinâmicas

- A página index.html é estática, ou seja, não muda entre requisições;
- Páginas como perfis de redes sociais são templates:
  - mudam de acordo com o usuário;
  - são chamadas dinâmicas;
- Definiremos mais adiante uma página de perfil:
  - Os dados de um usuário serão carregados de forma dinamicamente;
  - A partir de dados da requisição, a página será montada e devolvida ao usuário.

# A página de Perfil

- Crie pasta templates uma página chamada perfil.html;

```
<!-- connectedin/perfis/templates/perfil.html -->
<!DOCTYPE html>
<html lang="pt-br">
 <head>
 <meta charset="utf-8">
 <title>ConnectedIn</title>
 </head>
 <body>
 <h1>Detalhe do Perfil</h1>
 </body>
</html>
```

# Definindo a nova view

- Deve-se adicionar uma nova função ao arquivo views.py;
- Ao acessarmos "exibir" na urls, retornaremos a página perfil.html;

```
connectedin/perfis/views.py
```

```
from django.shortcuts import render
```

```
def index(request):
```

```
 return render(request, 'index.html')
```

```
def exibir(request):
```

```
 return render(request, 'perfil.html')
```

# Registrando a nova rota

```
connectedin/urls.py
from django.conf.urls import path
from perfis import views

urlpatterns = [
 path('admin/', admin.site.urls),
 path('', views.index),
 path('perfil/', views.exibir),
]
```



# Registrando a nova rota

- Para definir a rota para a view criada, deve-se editar o arquivo `urls.py` da aplicação;
- A ideia é futuramente acessar um perfil pelo seu "id". Exemplo:
  - `http://localhost:8000/perfis/1`
  - O Django exibiria o perfil cujo id é 1;
  - Esse padrão de URL é fortemente recomendado (REST);
  - Define-se via expressões regulares;
  - o Django receberá esse "parâmetro" de um ou mais dígitos e permitirá buscar perfis pelo id.

# Testando a nova rota

- Acesso o navegador pelos seguintes endereços:
  - `http://localhost:8000/perfis/10`
  - `http://localhost:8000/perfis/`
  - `http://localhost:8000/perfis/ab`
- Apenas a primeira URL funciona, pois pela regex definida deve-se usar o padrão `'/perfil/dígitos'`.

# Repassando o id ao Django

- Define-se o nome do parâmetro a ser recuperado na função da view;
- Usa-se o padrão <tipo:nome>;
- 

```
connectedin/urls.py
```

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

```
urlpatterns = [
 path('admin/', admin.site.urls),
 path('', views.index),
 path('perfil/<int:id>', views.exibir),
]
```

# Recebendo o id no Django

- O nome do parâmetro id repassado deve ser acrescentado na função da view:

```
connectedin/perfis/views.py
```

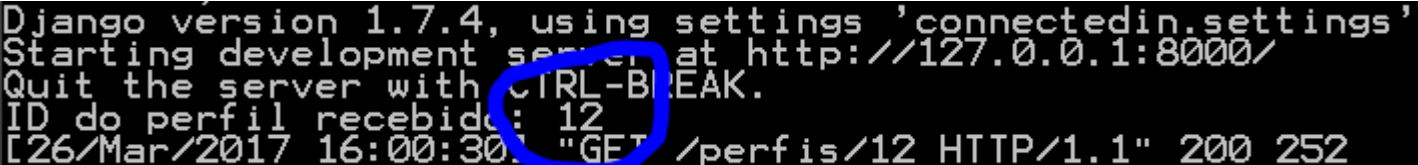
```
from django.shortcuts import render
```

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

```
def exibir(request, perfil_id):
 print ('ID do perfil recebido: %s' % (perfil_id))
 return render(request, 'perfil.html')
```



localhost:8000/perfis/12



```
Django version 1.7.4, using settings 'connectedin.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
ID do perfil recebido: 12
[26/Mar/2017 16:00:30] "GET /perfis/12 HTTP/1.1" 200 252
```

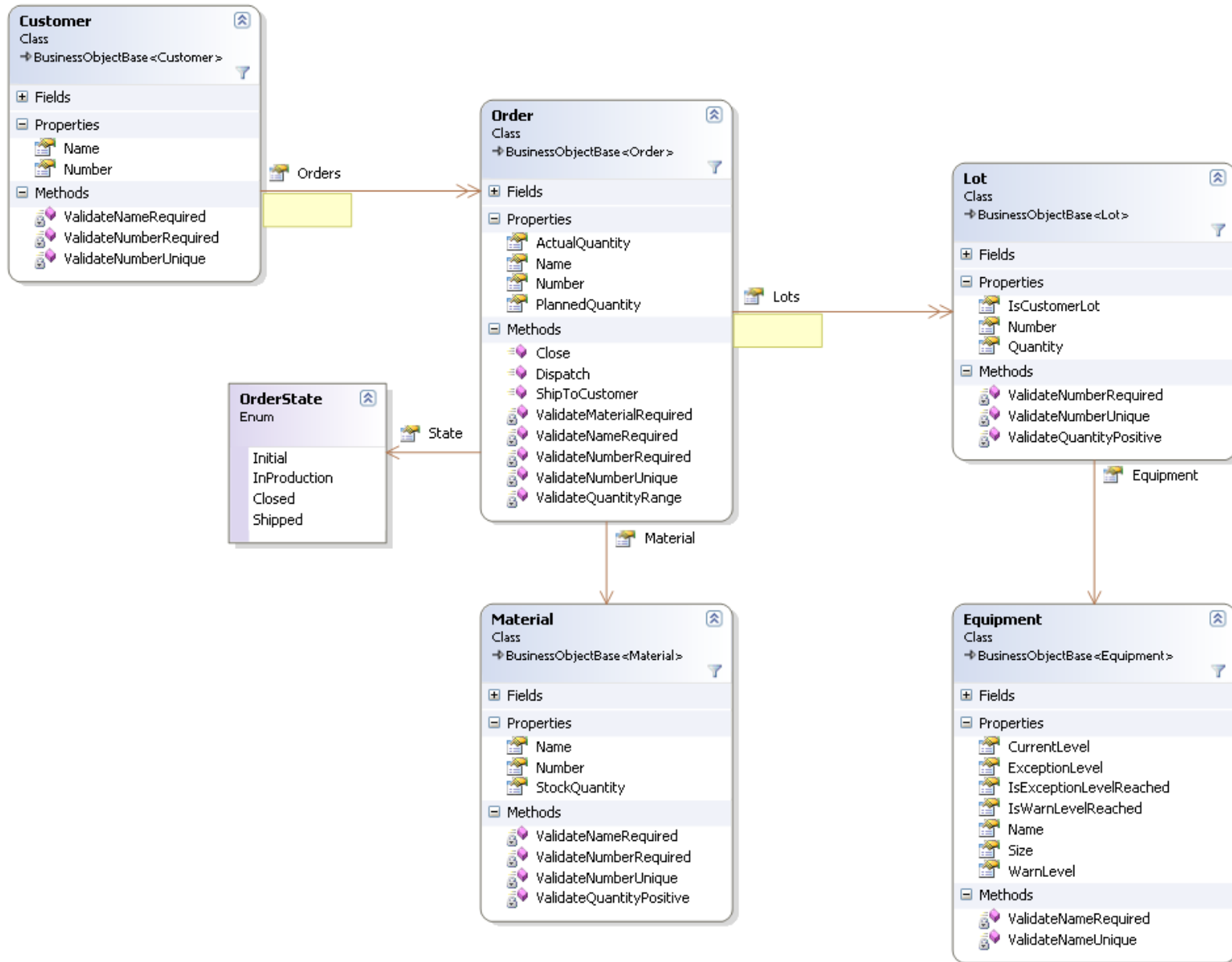
# Montando uma página dinâmica

- Os dados de páginas dinâmicas tipicamente vêm de bancos de dados;
- Um alternativa por enquanto é criar dados em memória;
- Para isso, é preciso:
  - Obter o id passado por parâmetro (feito);
  - Criar um objeto Perfil e devolvê-lo à página;
  - Preencher a página em pontos definidos com os dados do perfil.

# Models

- Modelo em Django é uma classe básica, uma entidade do sistema;
- Tipicamente são classes que possuem atributos a serem persistidos;
- Fazem parte do modelo de objetos do sistema.
- São classes que ficam armazenadas em meios de persistência como banco de dados;
- Exemplos: Conta, Pessoa, Perfil, Livro, Processo, Produto;
- No Django, colocamos essas classes no arquivo **models.py**.

# Exemplo de classes de modelos



# Classe Perfil

```
connectedin/perfis/models.py
```

```
from django.db import models
```

```
class Perfil(object):
```

```
 def __init__(self, nome='', email='',
 telefone= '',
 nome_empresa=' '):
```

```
 self.nome = nome
```

```
 self.email = email
```

```
 self.telefone = telefone
```

```
 self.nome_empresa = nome_empresa
```



# Retornando Perfis

- Deve-se instanciar alguns perfis;
- ... E retornar um dicionário para a página:

```
connectedin/perfis/views.py
```

```
from django.shortcuts import render
```

```
from perfis.models import Perfil
```

```
código omitido
```

```
def exibir(request, perfil_id):
```

```
 perfil = Perfil()
```

```
 if perfil_id == '1':
```

```
 perfil = Perfil('Elvis', 'elvis@gmail.com',
 '99999-9999', 'IFPI')
```

```
 if perfil_id == '2':
```

```
 perfil = Perfil('Lucas', 'lucas@gmail.com',
 '99987-4567', 'TCE')
```

```
 return render(request, 'perfil.html', { "perfil" : perfil})
```

# Exibindo um perfil

- Para fazer acesso ao dicionário com o objeto perfil, são utilizadas chaves;
- Dentro das chaves, deve-se acessar o dicionário e os atributos:

```
<!-- connectedin/perfis/templates/perfil.html -->
<!DOCTYPE html>
 <!-- código omitido -->
 <body>
 <h1>Detalhe Perfil</h1>
 nome: {{perfil.nome}}, email: {{perfil.email}},
 telefone: {{perfil.telefone}},
 empresa: {{perfil.nome_empresa}}
 </body>
</html>
```

# Prática

- Execute os seguintes passos:
  1. Crie um projeto chamado mysite;
  2. Crie o banco;
  3. Suba o servidor;
  4. Acesso o endereço da aplicação;
  5. Crie uma aplicação chamada pools;
  6. Registre a aplicação;
  7. Defina uma view;
  8. Defina uma url;
  9. Suba o servidor;
  10. Teste novamente o endereço da aplicação

# Prática

1. Crie uma página de boas vindas chamada index.html e exiba-a;
2. Crie uma página question.html, uma classe Question com os campos "question\_text" e pub\_date, representando a data da publicação da pergunta;
3. Altere os arquivos de views, rotas etc. para exibir essa página;
4. Faça uma alteração no na aplicação para que seja possível exibir 3 questões consultando pelo id a partir da barra de endereços do navegador.

# Django

**Introdução, Projetos e aplicações,  
rotas, views e templates**

**Ely – [elydasilvamiranda@gmail.com](mailto:elydasilvamiranda@gmail.com)**