Django framework e Desenvolvimento Baseado em Componentes

Universidade de Brasília - UNB
Faculdade Gama – FGA

Brasília, DF
Junho, 2018

Hugo Alves dos Santos Barbosa - 15/0036884

Iago Vasconcelos de Carvalho - 15/0011849

Lucas Oliveira Silva - 15/0016018

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. André Luiz Peron Martins Lanna

Disciplina: Desenvolvimento Avançado de Software

Índice

Lista de ilustrações **5**

Introdução **6**
- Propósito do Django Framework 6
- Estrutura do Django Framework 6
- Pontos de extensão e funcionalidades 7
  - Componentes do Django 7
  - Componentes Extensíveis 7
  - Seu Código 8
  - Componentes Externos 8

Modelo de componentes do Django framework **8**
- Definição e estrutura de um Componente Django 8
- Detalhes de implementação de componente 9
- Interfaces de componente 10
- Adaptações de componente 10

Composição de componentes **11**
- Comunicações de componentes e frameworks 12
- Diagrama comportamental de comunicações 14

Referências **15**

Histórico de versões

Versão	Descrição	Autor
1.0.0	Criação do template do documento	Iago Carvalho, Hugo Alves, Lucas Oliveira
1.0.1	Composição de componentes	Hugo Alves
1.0.2	Definição e estrutura de um componente do Django Framework	Iago Carvalho
1.0.3	Adaptações de componente	Iago Carvalho
1.0.4	Interfaces de Componente	Iago Carvalho
1.0.5	Detalhes de implementação de Componente	Iago Carvalho
1.1.0	Comunicação de componentes e framework	Hugo Alves
1.1.1	Diagrama comportamental de comunicações	Hugo Alves
1.1.2	Propósito do Django Framework	Lucas Oliveira
1.1.3	Pontos de extensão e funcionalidades	Lucas Oliveira
1.1.4	Modelo de componentes do Django framework	Lucas Oliveira
1.1.5	Formatação	Lucas Oliveira
1.2.0	Lista de Figuras	Lucas Oliveira

Lista de ilustrações

Figura 1 - Fluxograma dos componentes do Django

Figura 2 - Componentes do Django (Adaptado de Arun Ravindran)

Introdução

Propósito do Django Framework

O Django é um framework python desenhado para tornar tarefas comuns de desenvolvimento web mais rápidas e fáceis. Ele trata desde a comunicação com o usuário até o processamento da requisição e resposta.

Esse processo segue as seguintes etapas:

1. O framework determina qual o módulo URLconf root deve ser usado. Geralmente armazenado na variável de configuração ROOT_URLCONF.
2. O Django carrega o módulo urls.py e procura pela variável urlpatterns que armazena uma lista de django.urls.path() e/ou django.urls.re_path().
3. A lista em urlpatterns é percorrida em ordem e para na primeira url que corresponde ao padrão.
4. Uma vez que a url é encontrada, o Django importa e executa a view (uma função em Python). Esta view recebe os seguintes argumentos:
    a. Um objeto do tipo HttpRequest.
    b. Se a url correspondente não tiver retornado nenhum grupo nomeado, as correspondências da expressão regular serão fornecidas como argumentos posicionais.
    c. Os argumentos são compostos por qualquer das partes nomeadas que correspondem à expressão, sobrescrito por qualquer argumento especificado no argumento opcional kwargs.
5. Se nenhuma url for encontrada ou se uma exceção for lançada durante o processo o Django executa uma view para manipulação de erros.

Estrutura do Django Framework

O Django é estruturado utilizando a arquitetura em camadas MVT: model, view e template. A camada Model é a camada mais baixa e é onde fica a descrição dos dados armazenados pela aplicação. Cada model possui os campos e comportamentos desses dados e, geralmente, correspondem a uma tabela no banco de dados.

A segunda camada é a View, responsável por receber as requisições dos usuários e retornar uma resposta de volta para o usuário. Essa resposta pode ser uma mensagem, um arquivo, um JSON ou o que for necessário para atender as necessidades do usuário. Também é nessa camada onde é feito o processamento de dados necessários para gerar a resposta.

Por fim, a camada mais alta é a Template que cuida da interação com o usuário. Por ser um framework web, essa camada trabalha com arquivos contendo algumas partes HTML estáticos, mas provendo recursos para partes dinâmicas.

Pontos de extensão e funcionalidades

No tópico de introdução que falava sobre o propósito do Django foi abordado como o Django processa uma requisição do usuário. Nesta seção esse processo será aprofundado com o foco nos pontos de extensão do framework, ou seja, onde seu código é inserido.

Primeiramente serão apresentado os componentes do Django, seguido pelos extensíveis, depois um breve comentário sobre onde o desenvolvedor escreve seu código e por último os componentes externos.

Componentes do Django

Os componentes do Django são o modwsgi e URLConf. O modwsgi é um módulo Apache capaz de hospedar qualquer aplicação Web Server Gateway Interface (WSGI) escrita em Python - o que inclui o Django. O WSGI é uma especificação que descreve como um servidor web se comunica com as aplicações e como as aplicações podem ser encadeadas para processar uma requisição. Esse módulo pode operar nos modos embutido ou daemon.

O modo embutido incorpora o Python dentro do Apache e carrega o código na memória quando o servidor inicia, isso aumenta significativamente a performance em comparação a outros arranjos de servidores. No modo daemon o modwsgi dispara um processo independente para manipular as requisições, assim ele pode ser reiniciado sempre que necessário sem precisar reiniciar o servidor. Além disso, ele pode ser executado como um usuário diferente, o que aumenta a segurança.

O segundo componente do Django é o URLConf (do inglês, URL configuration). Esse componente é um módulo escrito em Python que mapeia urls para métodos (views). O mapeamento pode ter o tamanho que o desenvolvedor achar necessário, pode referenciar outros mapeamentos e pode ser construído dinamicamente.

Componentes Extensíveis

Os componentes extensíveis são Request, Response e Views middlewares, Filtros, Tags, Contexto e Gerentes. Middleware é um framework de *hooks* dentro das requisições e respostas do Django. Também pode ser descrito como um sistema para alterar as entradas e saídas do Django. Cada middleware possui uma função única e específica.

Os filtros e tags são usados para apresentações dinâmicas da aplicação que não possuem suporte no conjunto de principal de funções do template. Os gerentes são responsáveis por pela interface entre as queries do banco de dados e cada model do Django.

É importante frisar que cada um desses componentes citados pode ser customizado e/ou criado de acordo com as necessidades do programador.

Seu Código

Existem quatro componentes que são totalmente escritos pelo desenvolvedor: as models, views, templates e urls. O foco neste tópico serão as urls, pois os componentes já foram descritos na Estrutura do Framework. Como já dito anteriormente, as urls no Django guiam uma requisição até a view que irá processá-la e. Para criar urls para uma aplicação é necessário modificar o arquivo urls.py e acrescentar o padrão desejado e a view que responsável por ele.

Componentes Externos

Por último existem dois componentes externos: o banco de dados e o browser. O banco de dados é onde os dados são realmente armazenados e nele existe uma tabela para cada model utilizada. O browser é onde os templates são renderizados e apresentados para o usuário. Por serem componentes externos, o Django não limita quais podem ou não ser usados.

Modelo de componentes do Django framework

Definição e estrutura de um Componente Django

O framework Django possui uma série de componentes já implementados, sendo que grande parte deles foram construídos com base nos fundamentos dos padrões de projeto *GoF.* O Django tem como princípio acelerar o processo de desenvolvimento de aplicações web provendo componentes e ferramentas com funcionalidades gerais, cabendo ao desenvolvedor aproveitar estes recursos e codificar funcionalidades específicas ao seu contexto. Existe uma série de funcionalidades que são comuns ao desenvolvimento de aplicações web, permitindo o reuso de código, pois, ainda que as novas funcionalidades desenvolvidas com o framework possam estar relacionadas a um contexto, as aplicações Django podem ser divididas e estruturadas em *apps*, que são componentes reutilizáveis que podem ser aproveitados por outros projetos desenvolvidos em Django, assim como utilizados na composição para criação de novos componentes e *apps,* consistindo de um conjunto de funcionalidades que pode ser considerado um nível hierárquico acima das classes. Um exemplo de um *app* Django é um componente de “carrinho de compras”, que pode ser utilizado na composição de diversos projetos Django de *e-commerce* diferentes com pequenas adaptações. *Apps* em projetos Django são idealizados e desenvolvidos para reutilização, sendo que a refatoração para desenvolvimento de *apps* atômicos é uma prática muito incentivada. O Django oferece um grande suporte e infraestrutura para o empacotamento e distribuição de

componentes *apps*, além do suporte para a composição de um projeto Django a partir de vários *apps*, sendo estes os pontos fortes de reutilização de software para desenvolvimento de aplicações web com esse framework. Os outros componentes deste framework são estruturados de maneira similar, auxiliando no desenvolvimento e teste de aplicações.

O Django utiliza a arquitetura MVT, onde projetos e *apps* são estruturados de modo a separar as responsabilidades de criação de classes que representam a base dados através de ORMs, classes de processamento de requisição e apresentação através de templates e linguagens de marcação. Essa estrutura atende ao padrão MVC com um funcionamento diferente, porém nem todos os componentes de um projeto apresentam necessariamente todas essas camadas definidas. Dentro da estrutura de cada componente as models são responsáveis por prover uma aproximação orientada a objetos de lidar com as *databases*, permitindo manipular e estruturar os dados da sua aplicação web através da herança de componentes de model que são fornecidos pelo próprio Django. Uma model é um componente base para composição de vários outros componentes, por isso é recomendado evitar dependências desnecessárias. O Django ORM é o componente que provém o mapeamento das classes model e a base de dados, servindo como mediador de *queries*. O componente view é responsável por encapsular a lógica para processamento de uma requisição de usuário e retorno de resposta. Os componentes de Template apresentam funcionalidades para linguagem de marcação e apresentação informação para o usuário. Além desses componentes principais, *apps* e projetos Django são estruturados ainda através de componentes como middlewares, componentes de autenticação, Log e Forms. Os componentes incluídos pelo próprio Django possuem nomes reservados, por isso é recomendado que os desenvolvedores evitem nomear projetos com essas palavras para evitar possíveis problemas e conflitos.

Detalhes de implementação de componente

Os componentes Django são considerados "*Pythonic*" por serem implementados, acessados e alterados seguindo as diretrizes e convenções da linguagem python. Os próximos tópicos abordam como o Django incorpora novos componentes e como suas interfaces são definidas, porém a ocultação de informação não é abordada com a mesma severidade de outros frameworks. O python utiliza convenções sobre políticas de acesso a membros e atributos como o uso de *underscores*, ao invés de recursos de proteção como protected e private, onde os recursos de um componente não são completamente ocultos do framework e nem de outros componentes, permitindo de certa forma que o desenvolvedor altere os dados internos de um componente. Apesar do Django se tratar de um grande "pacote" de componentes e apresentar estes recursos de acesso, seu nível de encapsulamento provém componentes que podem ser facilmente substituídos

por outros componentes com a mesma interface, separando os detalhes de sua implementação.

Interfaces de componente

Como descrito em tópicos anteriores, o Django framework separa os componentes de maneira limpa e organizada, através de camadas com responsabilidades bem definidas, como as camadas de aplicação e base de dados. Não existem exigências de como a interface de comunicação de um componente reusável deve ser propriamente definida, os componentes devem prover funcionalidades, sendo que tais funcionalidades alinhadas a documentação apropriada ditam como o componente deve ser utilizado. Na maioria das vezes a interface entre componentes e sua composição são transparentes por utilizarem convenções e sub-módulos em comum, sendo estabelecidas através dos recursos de importação, configuração e mapeamento de urls do próprio Django ou subclasses. Para garantir a comunicação de componentes reaproveitáveis é necessário desacoplar um componente das URLconf a nível de projeto. Além dos recursos que indicam como utilizar a interface de um componente os apps django normalmente incluem um rst com comandos de configuração caso necessitem de detalhes extra para reuso. Devido a essas vantagens e a arquitetura baseada em componentes *shared-nothing,* onde cada parte da arquitetura é independente e pode ser substituída, o framework Django consegue se comunicar com qualquer outro framework *Client-side,* permitindo a transmissão de conteúdo em diversos formatos como JSON, HTML, etc. Por se tratar de um framework opinionated o Django possui interfaces bem definidas para seus componentes, com documentação clara para a composição de componentes, porém isso restringe um pouco a escolha de componentes fora de seu domínio.

Adaptações de componente

A documentação do Django framework o define como sendo um ecossistema de componentes plugáveis. Os componentes Django em sua maioria são adaptados para o framework através de herança, com seus métodos e atributos disponibilizados às subclasses para utilização de acordo com seu nível de proteção. Para esse tipo de adaptação de componentes Django acontecer é necessário disponibilizar os aspectos internos ao componente adaptado com documentação apropriada, além de um conhecimento das funcionalidades da superclasse. A herança para adaptação de componentes pode ocorrer de forma concreta, abstrata ou proxy, devido ao fato das models e alguns componentes como Forms, Views, etc, do Django se tratarem de classes. Na herança com proxy é possível adicionar comportamento a classe pai.

O Django possui uma série de componentes disponíveis devido aos aspectos colaborativos e vantagens de plug de componentes, sendo que grande parte desses componentes são códigos *open source*, permitindo a adaptação através da adição dos diretórios do componente e alteração direta de código, caso seja necessária devido a problemas e incompatibilidades, como diferenças de componentes desenvolvidos para python 2.x e 3.x.

Para adaptar um novo componente para o Django framework normalmente são necessárias alterações no settings.py e configuração de urls no arquivo urls.py de um projeto.

Composição de componentes

Como foi visto, há dois conceitos principais que regem a forma como o Django foi idealizado e como o reuso de software é feito seguindo algumas diretrizes da linguagem Python: os *apps* e *packages.* Os *packages* são basicamente grupos de códigos de Python que são facilmente reutilizáveis, sendo que *packages* que contém vários arquivos Python são chamados de *modules.* Normalmente esses módulos e pacotes são utilizados através de *imports* da classe inteira ou somente de alguns dos métodos presentes nela. O Django em si, por exemplo, é simplesmente um pacote Python!

Como já foi mostrado um *app* é um pacote Python desenhado para ser usado e reusado dentro de um projeto Django. O próprio Django mantém um agregados de apps, frameworks e pacotes feitos pela comunidade e que podem ser úteis para agilizar o desenvolvimento ao reutilizá-los. Para isso basta instalá-los dentro do projeto e importar o que for necessário, ou seja, com isso o Django leva a reutilização de código a outro patamar.

Comunicações de componentes e frameworks

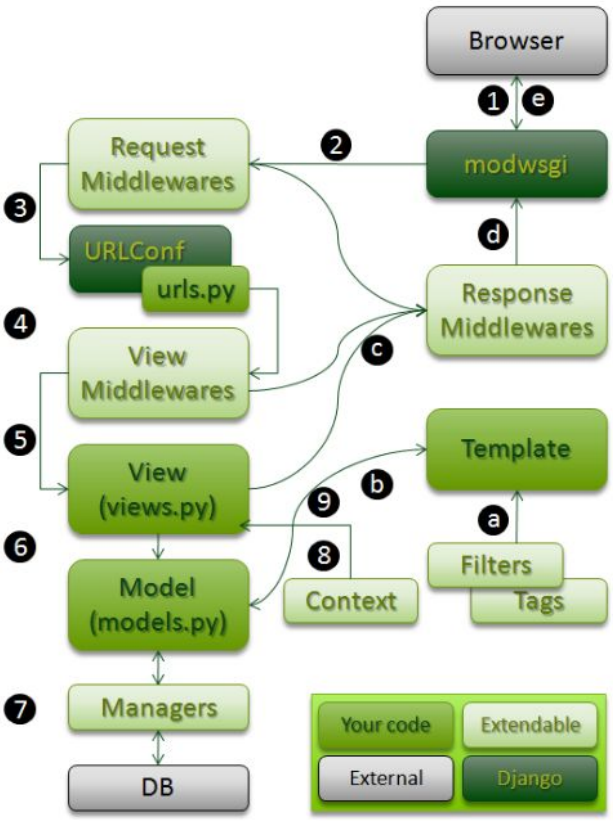

Figura 1 - Fluxograma dos componentes do Django

A comunicação entre os componentes do Django, seguindo o fluxo do MVT, é mediada muitas vezes por *middlewares,* que são componentes responsáveis pelo processamento dos requests e responses dentro do framework, sendo muitas vezes vistos como um sistema de plugins leves e de baixo nível que alteram globalmente as entradas e saídas do Django. Existem alguns deles presentes no framework, como o *AuthenticationMiddleware,* que adiciona a informação de que o usuário está logado em todo objeto HttpRequest, porém o programador tem a liberdade de criar seu próprio, caso nenhum se adeque ao seu contexto específico.

Quando essa comunicação não é feita através dos middlewares, ela é feita normalmente de forma mais direta, respeitando a hierarquia do MVT. Por exemplo, em um site de e-commerce é necessário buscar as informações de todos os produtos dentro do carrinho de um usuário. Mesmo o app do usuário sendo diferente do carrinho, é possível instanciar o objeto da model que descreve o carrinho e usar algum método dessa classe para buscar essas informações dentro do banco de dados, sendo que muitos desses métodos são oriundos da herança da model criada com a do Django.

Como já foi dito, o Django agrega frameworks baseados em Django que podem ser utilizados por programadores que desejam agilizar ainda mais o desenvolvimento de um site, além de poder contribuir ainda mais com o framework. Normalmente eles são disponibilizados em sites como Github e Gitlab e apresentam uma documentação mostrando como baixá-lo e instalá-lo dentro do projeto, além dos seus pontos de extensão.

Existe um arquivo muito importante no Django chamado *settings.py,* que descreve as configurações que o Django irá utilizar no projeto. Nesse arquivo podem ser descritos as classes middlewares a serem usadas, o banco de dados, fuso horário, tipos de logs, dentre outras diversas configurações. Sabendo disso, é importante bastante cautela ao modificar esse arquivo, pois ele pode causar problemas graves no projeto.

Também é possível gerar APIs usando o framework Django REST, usado para serializar dados para fazer a comunicação entre diferentes aplicações. Para isso ele transforma dados complexos, como querysets, para tipos nativos de dados do Python, como JSON e XML, funcionando de forma similar à classes de *Form* e *ModelFom*. O framework provê uma classe *Serializer,* que é ao mesmo tempo poderosa e genérica de acordo com o modo que as saídas são controladas, além da classe *ModelSerializer,* que fornece uma maneira mais simples de lidar com modelos de instância e querysets. Essa abordagem vem ganhando muita força nos últimos anos, pois é desenvolvido uma aplicação para lidar com o processamento dos dados e várias outras aplicações, até mesmo de plataformas diferentes, para somente consumir esses dados.

Por também ser um pacote Python, para utilizá-lo basta ter o Django instalado, baixá-lo e adicioná-lo no arquivo de configuração. Além disso há vários pacotes úteis e que podem agilizar o desenvolvimento da aplicação, basta achar os que se adequam melhor ao que se deseja fazer.

Diagrama comportamental de comunicações

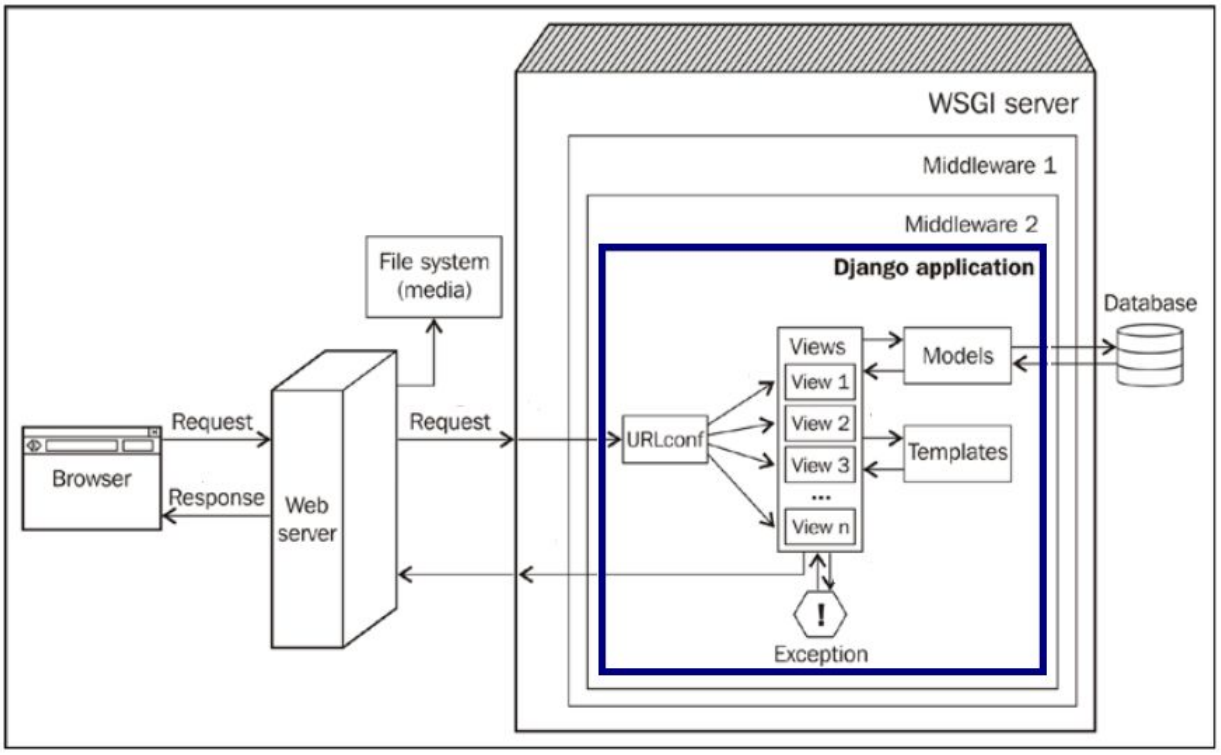

Figura 2 - Componentes do Django (Adaptado de Arun Ravindran)

A imagem acima mostra o caminho que uma requisição percorre até chegar a uma resposta, passando pela aplicação em Django. Porém, o foco principal está nos componentes presentes no quadrado em azul, que são justamente os componentes principais que o Django disponibiliza. É importante ressaltar que quando o projeto é organizado por *apps,* essa organização de componentes será semelhante, ou seja, cada um terá seus arquivos que correspondem às suas models, views, templates ou outros componentes que podem fazer parte do app, como formulários, testes e middlewares.

É possível ver claramente a forma que os componentes, especialmente no que diz respeito ao MVT, se comunicam entre si. Quando uma requisição chega no componente *URLconf*, uma view que lidará com ela de acordo com o que foi requisitado na URL é selecionada, transformando-a também em um objeto Python conhecido como *HttpRequest.* Essa view poderá provocar uma exceção, renderizar um template ou tentar buscar um dado no banco de dados. Caso ela tente essa opção a model será usada e, em conjunto com o Django ORM (que não é mostrado na imagem), manipulam os dados do banco de dados, sendo esse um componente externo que é plugado no projeto pelo desenvolvedor. Por fim o objeto *HttpResponse* é renderizado, normalmente por uma view, como uma string e deixa a aplicação Django, sendo renderizada por fim como uma página em um browser.

Referências

GANTAN, Xiaonuo. *Package Your Django Application into a Reusable Component*. pythoncentral. 2013

DSF. *Django Packages*. 2010. Disponível em: <https://djangopackages.org/> Acesso em 24 de Junho de 2018.

DSF. *Django Documentation. Django Documentation - Release 2.2.dev20180625133916*. 2018

RAVINDRAN, Arun. *Django Design Patterns and Best Practices*. Packt Publishing. 2015

CHRISTIE, Tom. Django Rest Framework Serializers. Disponível em: <http://www.django-rest-framework.org/api-guide/serializers/> Acesso em 25 de Junho de 2018.

MOZILLA. *Django Introduction*. 2018. Disponível em: <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction> Acesso em 25 de Junho de 2015.

DUBE, Manjula. *Python - Encapsulation Does It Exist?*. Medium. 2017