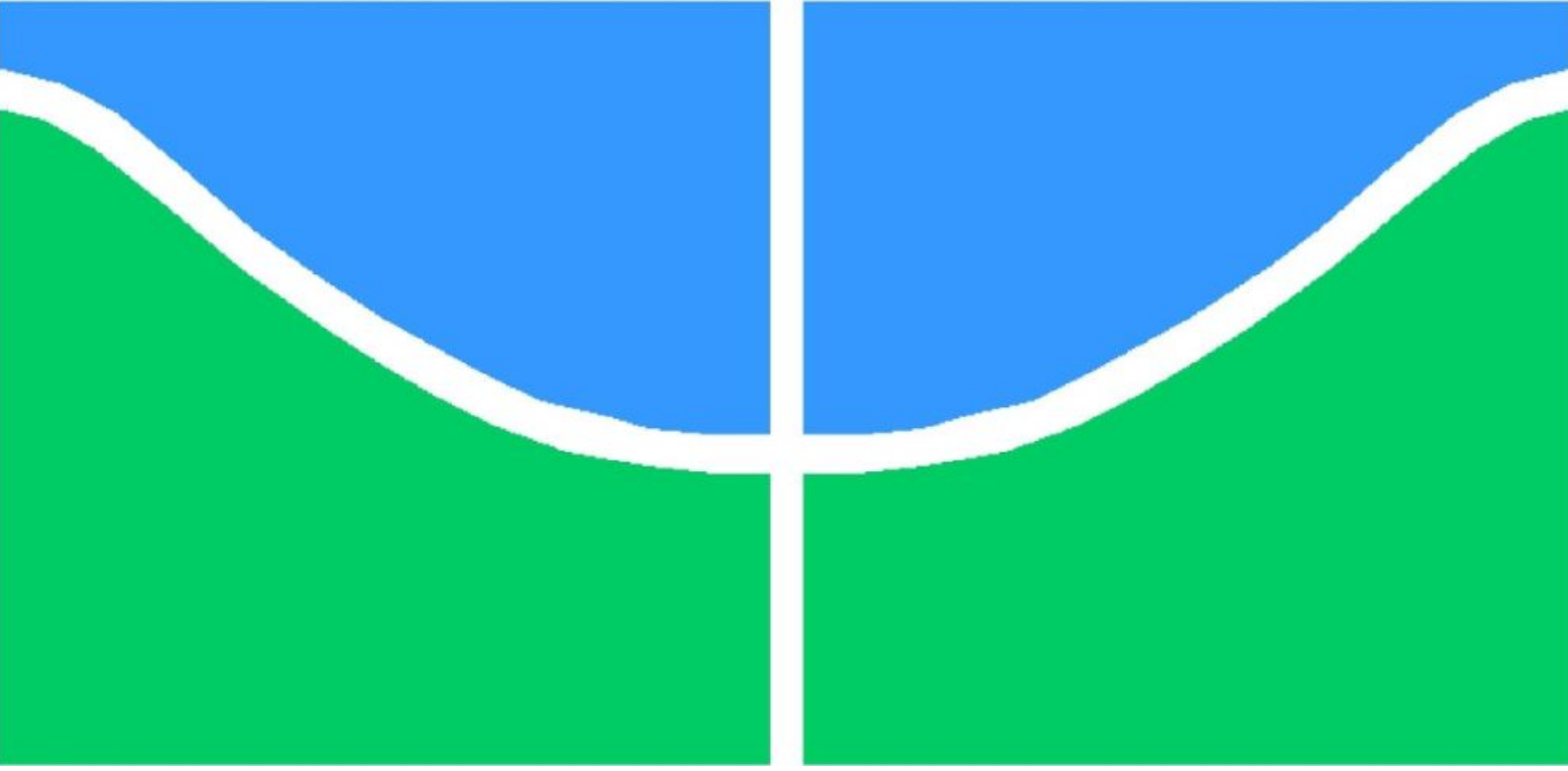


Frameworks / Desenvolvimento Baseado em Componentes (DBC) Django

Membros:

Allan Nobre 15/0029624

João Paulo N. Soares 15/0038267

Thiago Nogueira 15/0047142

Ronyell Henrique 15/0046073

Marcelo Augusto 14/0046411

Brasília, 23 de Junho de 2018

Sumário

1. Introdução

Dada a contextualização no enunciado motivador deste trabalho, a análise requisitada de um determinado framework será baseada em conceitos apresentados pelo artigo de [4] Spagnoli et. al, 2003.

Para o foco de estudo neste trabalho foi escolhido o Django Framework, devido, principalmente, a afinidade da equipe com este e a quantidade de vezes em que foi empregado em projetos onde os participantes deste time trabalharam.

Assim, este trabalho busca responder as questões explicitadas no enunciado motivador deste, tendo como objeto de estudo o Django Framework e como fonte de informações fundamentais o artigo de [4] Spagnoli et. al, 2003.

2. Apresentando o Django

2.1. O que é? Pra que serve?

O Django é um framework Web Python de alto nível, grátis e de código aberto que incentiva o rápido desenvolvimento, design limpo (MTV - Model Template View) e pragmático. Como característica comum em um framework web, ele cuida de grande parte do que há por trás do desenvolvimento da Web, para que seja possível se concentrar em escrever a aplicação sem precisar passar por todas as partes comuns a maioria do sistemas webs.

Vale ressaltar que este framework, além de empacotar configurações bases e/ou repetitivas, porém, necessárias a criação de sites, ele oferece soluções pré prontas de componentes comuns utilizados nestas aplicações, como: autenticação de usuárias (inscrever-se, realizar login, realizar logout), painel de gerenciamento do site, formulários, upload de arquivos, níveis de permissão etc.

2.2. Estrutura

A estrutura do Django funciona baseada no modelo (MTV - Model Template View), que tem como base o padrão de projeto MVC (Model View Controller). Descrição das camadas de dados do MVT:

Model - Esta camada é responsável por gerir, modelar e persistir os dados, tendo como principais funções controlar, cuidar das regras de negócio da aplicação e controlar as transações com o banco de dados da aplicação.

View - É a camada encarregada de interpretar entradas vindas de outros sistemas ou da interface do próprio sistema, ou seja, é uma ponte entre a model e o template. Nesta camada é implementado a lógica de comunicação da aplicação com a Model, além de direcionar as informações que serão apresentadas para o template correto.

Template - Camada incumbida de ser a interface do sistema, sendo responsável por toda a comunicação entre usuário e aplicação. Na maioria dos casos todas as tecnologias usadas nesta camada do sistema são voltadas a interação com o usuário, procurando gerar uma interface agradável.

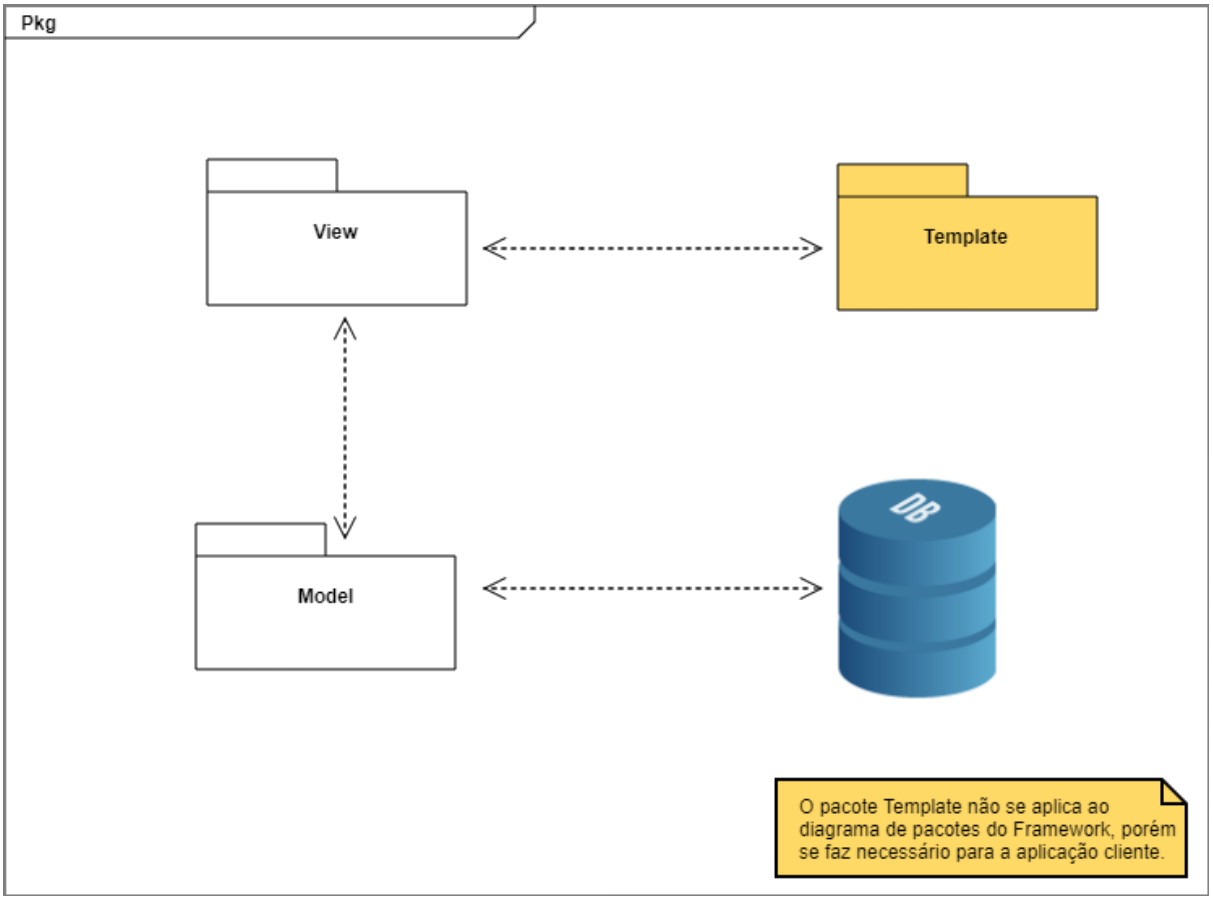


Imagem demonstrativa da interação MVT

Quando chega uma requisição para o servidor web, ela é passada para o Django, que identifica do que ela se trata. Assim, ele abstrai o endereço web toma de determinada ação baseada no endereço.

Essa abstração é feita pelo *urlresolver* do Django. (Vale ressaltar que o endereço de um site se chama URL - Uniform Resource Locator, em português Localizador de Recursos Uniforme, dessa forma o nome *urlresolver*, ou resolvedor de urls, faz sentido). Ou seja, ele pega uma lista de padrões e tenta corresponder

com a URL. O Django verifica os padrões e, se algo é correspondido, passa a solicitação para a função associada (que é chamada/criada View).

Então a camada View gera uma resposta (após executar o que foi programada para realizar para aquela instrução requisitada) e o Django pode enviá-la para o navegador web do cliente formatada de acordo com a necessidade do desenvolvedor definida na camada Template.

2.3. Pontos de Extensão

Levando em consideração o conceito de *hot-spot* e *frozen-spots*, temos que:

- ***Hot-spots*** são pontos extensíveis de um framework, é aí que as adaptações são feitas, desse modo os hot-spots são projetadas de forma genérica cabendo à aplicação que o utiliza moldá-lo para o seu contexto.
- ***Frozen-spots*** são pontos fixos de um framework, define como seus componentes básicos são e seus relacionamentos. Os frozen-spots também fazem parte do *core* do *framework.*

Nem tudo em um framework é hot-spot, portanto nem tudo é extensível. Dessa forma foram identificados os seguintes componentes como sendo hot-spots:

- ***Models***:
  - *Model*: Na camada de modelos todas as classes devem estender da classe *Model*, podendo existir uma árvore de herança onde uma classe base herda da classe *Model* e esta é herdada pelas classes nas folhas da hierarquia. Portanto, a classe *Model* define bem pontos de refinamento para que elas possam se adaptar ao contexto da aplicação. É importante ressaltar que todas as models de um sistema criado com auxílio do *framework* Django devem obrigatoriamente herdar, mesmo que indiretamente, da classe *Model*.

- **Persistência:**
  - *Manager:* Nas classes modelo todas as operações relacionadas a persistência são feitas no banco de dados. Existe, por padrão, uma já implementada. Mas é possível a criação de uma específica para um objetivo específico, porém é muito incomum que isso ocorra.

- ***Views*:**

- *GenericViews*: Uma *view* é basicamente uma requisição que é processada e após esse processamento retorna uma resposta. No django é possível que se faça uma *view* apenas com uma função ou que seja implementada a partir de classes pré-definidas. Estas são chamadas de *GenericViews* que já têm implementações genéricas para algumas tarefas simples como detalhamento de objetos, listagens, etc.

- ***Templates*:**
  - *Forms*: O django fornece formas de auxiliar na criação de formulários. O formulário precisa especificar duas coisas: qual a url deve ser utilizada para enviar do dado de entrada e qual método será utilizado para o retorno. Os tipos de retorno possíveis são apenas *POST* e *GET*. Uma das implementações mais rebuscadas do *Forms*, feita pelo próprio django, é o *ModelForms* que utiliza dos atributos definidos na model para construir o formulário. Os Forms são “traduzidos” para <form> html.

3. Componentes no Django

3.1. Modelo de Componente

3.1.1. Tipos de Componentes

Os componentes são definidos de acordo com as interfaces que estes implementam. Desse modo os seguintes componentes, de maneira geral, são definidos:

Url: A url define o caminho de acesso para um objeto especificado, geralmente uma página html. As requisições são feitas através de http/https. No django as urls definem um acesso a uma view específica, seja essa view uma classe ou um método. As urls são especificadas em um arquivo url, através do módulo URLconf. Portanto, quando há uma requisição o URLconf transforma essa requisição em um objeto python do tipo HTTPRequest. Essas requisições são *frozen-spots,* pois não são extensíveis como os componentes especificados no item anterior.

A forma de utilizar uma url em django, ou seja, sua interface, é feita da seguinte forma:

```
urlpatterns = [
```

```
    path('', viewA , name='namea'), # url estática
    path('<int:id>/', viewB, name='nameb'), # url dinâmica
]
```

Onde *urlpatterns* define quais são as urls acessíveis na aplicação web, e cada item da lista é uma url específica. Como cada url é um objeto python, a url é um objeto dinâmico, ou seja, pode variar de acordo com a necessidade.

Regras básicas de uso:

- Não devem existir urls idênticas
- Uma mesma url pode ser usada tanto para um método POST quanto para um método GET.

Model: A modelo é a única fonte de dados do framework. Nela são definidas as estruturas de dados das entidades do sistema.

Regras básicas de uso:

- Todas as models do sistema devem herdar da classe *django.db.models.Model*, dessa forma todas as classes que representam estruturas de dados de uma entidade devem ser subclasses de Model.
- Cada atributo, sem exceção, representa um campo na tabela de banco de dados
- A abstração do banco de dados (mapeamento objeto-relacional) é feita automaticamente pela api que o django fornece.
- Todas as models devem estar, em princípio, no mesmo arquivo com nome models.py ou em arquivos individuais em um diretório com nome models com um arquivo __init__.py indicando quais classes/métodos devem ser importadas.

Os principais atributos herdados da *model* são o ***id*** do tipo inteiro,que é a chave primária de cada tupla no banco de dados, e o ***objects*** do tipo Manager, que são as interfaces responsáveis pelas operações de consultas no banco de dados e usadas para retornar instâncias de *models* do banco de dados.

Em relação aos métodos os mais importantes são o ***save()***, que utiliza o objeto manager para persistir no banco de dados, e o ***delete()***, responsável por excluir uma tupla, que está relacionada ao objeto, do banco de dados.

View: As *views* são responsáveis por encapsular a lógica de processamento de uma requisição de usuário e retornar uma resposta, a este

mesmo usuário. Portanto uma view basicamente recebe uma requisição a partir de um objeto HttpRequest, processa essa requisição e retorna um objeto do tipo HttpResponse, geralmente um objeto do tipo HTML..

Regras básicas de uso:

- Deve existir uma requisição do tipo HttpRequest
- Deve existir uma resposta do tipo HttpResponse
- A lógica da view fica entre a requisição e a resposta

Como mencionado acima, não é necessário extensão de classes para isso, basta que seja criado um método que tem como parâmetro uma requisição HttpRequest e tem como retorno um objeto HttpResponse. Da seguinte forma:

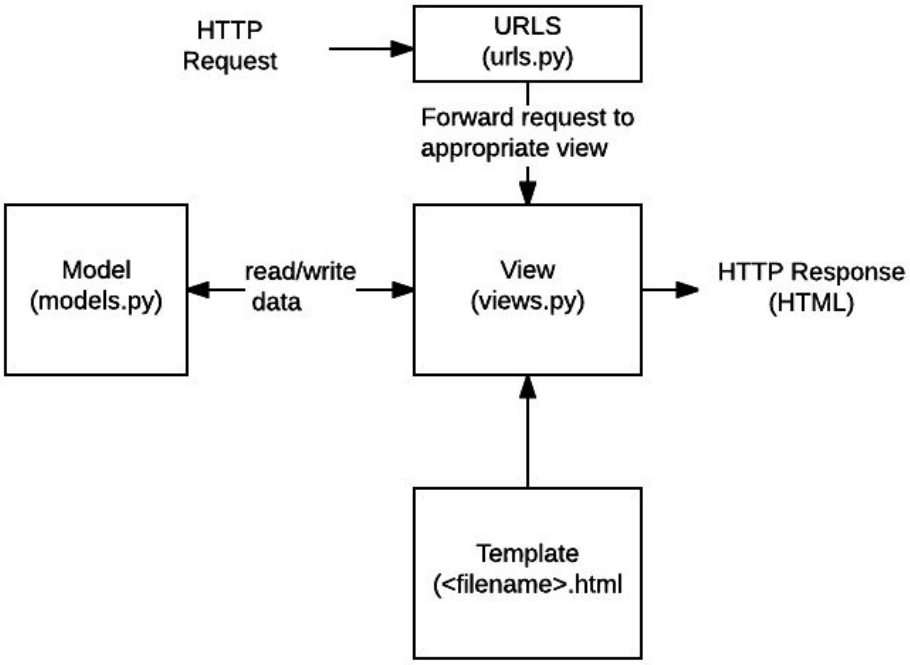
```
def current_datetime(request):
    now = datetime.datetime.now()
    html = "<html><body>It is now %s.</body></html>" % now
    return HttpResponse(html)
```

Outra maneira é a extensão de uma classe GenericView para que isso seja feito, nele existirá métodos que têm as mesmas características, uma requisição do tipo HttpRequest, lógica e uma resposta do tipo HttpResponse.

Template: O django é um framework web e por isso é necessário a criação de Html dinamicamente. Portanto, os componentes do template são responsáveis pela construção disso. O django utiliza linguagem de marcação de texto html como base e a parte dinâmica é criada com o auxílio da linguagem de template do Django. Exemplo:

```
<html>
        <h1>
        {% if user.is_authenticated %}Hello, {{ user.username }}.{%
endif %}
        </h1>
</htlm>
```

3.1.2. Formas de Interação



- **Url - View**
  - A comunicação é feita via HttpRequest da Url para a view, nessa requisição pode não ter parâmetros ou com parâmetros. Caso tenha parâmetros pode ser feita via GET ou via POST. Via GET os dados são anexados a url, caso seja um formulário. Via POST os dados são encapsulados antes de serem enviados.

  - Após todo o processamento a view retorna uma resposta à quem requisitou a Url. Esse retorno é um arquivo html.

- **View - Model**
  - Em seguida a requisição a view deve processar a requisição, e quase sempre, envolve as estruturas de dados do sistema, ou seja, models. Portanto, os componentes views são responsáveis por instanciar objetos model de acordo com os parâmetros passados nas requisições que são recebidas. A instanciação quase sempre envolve operações relacionadas a persistência.

- **Model - Database**
  - A model realiza operações relacionadas a persistência através do atributo *objects* que é do tipo *Manager.* Esse objeto é

responsável por fazer consultas, criação, deleção… etc. Após alguma dessas operações é retornado os dados para a model. Esses dados são recebidos e retornados para a view que o invocou.

- **View - Template**
  - A view envia os dados, dinamicamente, para um template html que o exibe os dados passados com auxílio da linguagem de template django. Após esse processamento a view encaminha esse html como resposta a requisição feita.

3.2. Detalhes dos Componentes no Framework

Um componente que é bem difundido na comunidade que utiliza o framework Django é o componente de formulário *Forms*. Trabalhar com formulários normalmente é uma tarefa complexa, por isso é um componente que foi criado pensando em facilitar de forma segura e eficiente a criação e gerenciamento de formulários nas aplicações web a utilização.

A estrutura do componente se dá através da classe principal Form, que possui características de uma modelo para descrever as estruturas lógicas do objeto formulário, seu comportamento e como essas partes serão apresentadas para quem estiver utilizando deste componente. O Form se encaixa na camada de template do framework Django, pois é ele que define quais campos serão apresentados no formulário da página.

A classe que implementa este componente mapeia os campos da sua modelo no formulário que será apresentado na aplicação web. Os inputs dessa classe são realizados através dos campos do formulário HTML e a lógica por de trás da Forms se encarrega por persistir esses dados.

No framework Django a maioria dos componentes são empacotados e baixados através do gerenciador de pacotes PIP. Em geral os componentes deste framework apresentam classes abstratas tendo os métodos tendo seu comportamento padrão definido, mas caso a utilização demande algo não implementado ainda é possível sobrescrever os métodos dessas classes abstratas. O pacotes baixados através do PIP devem ser referenciados nos imports que indicam o que é necessário para a execução correta da classe.

- Como deve ser definida a comunicação do Django:

Tendo em vista que um framework é uma aplicação quase completa, mas com pedaços faltando, ao receber um framework é preciso prover os pedaços que são específicos para cada aplicação, algumas técnicas básicas são utilizando alguns padrões de desenho de software como Template Method e Composição.

Os projetos Django seguem um padrão arquitetural MVT, próprio do Django, aderente ao MVC. O MVT separa o projeto estruturalmente em três partes: Model, View e Template.

O framework Django facilita na interface com o banco de dados e cuida de toda a comunicação entre as camadas MVT. Cada classe da modelo se compara a uma tabela do banco de dados, e as instâncias destas classes, representam os registros destas tabelas. Para adicionar valores ao banco, basta defini-los nas respectivas variáveis. Esta camada contém qualquer coisa e tudo sobre os dados: como acessá-lo, validá-lo, e quais comportamentos que têm e as relações entre os dados. Para o mapeamento dos dados, não será necessário utilizar códigos em SQL para garantir a persistência dos dados no banco. Por fim, pode-se ver como são feito outros tipos de extensões do framework no tópico 2.3, em que se fala dos hot spots do framework.

4. Abordagens Comuns em Frameworks

4.1. Composição - Componentes/Frameworks

Segundo Bachmann existem dois tipos de entidades gerais que podem ser compostas, sendo estas: componentes e frameworks de componentes. De forma geral os modos utilizados para a composição de componentes / frameworks são classificados por Bachmann em:

- **Componente - Componente**
  - Composição que permite a interação entre componentes e que define as funcionalidades de uma aplicação.O compromisso que especifica essa comunicação pode ser classificado como contrato em nível de aplicação;

- **Framework - Componente**
  - Composição que permite interações entre os frameworks de componentes e seus componentes.O compromisso que especifica essa comunicação pode ser classificado como contrato em nível de sistema;

- **Framework - Framework**
  - Composição que permite interações entre frameworks e permite a composição de componentes definidos em diferentes frameworks.O compromisso que especifica essa comunicação pode ser classificado como contrato de interoperação;

Esses contratos apresentados são descritos por Bachmann nas seguintes formas de composição:

- **Utilização do Componente**
  - Os componentes precisam ser empregados através de um framework antes de serem compostos e executados.Sua utilização envolve a definição da interface que o componente deve implementar de modo que o framework possa gerenciar seus recursos, implementando dessa forma o contrato em nível de sistema.

- **Utilização do framework**
  - Frameworks podem ser utilizados em outros frameworks, onde os mesmos podem apresentar diferentes camadas. O contrato dessa utilização é análoga ao contrato de utilização do componente, ou seja, é um contrato em nível de sistema entre os frameworks.

- **Composição simples**
  - Componentes utilizados em um mesmo framework podem ser combinados. A composição feita expressa funcionalidades específicas dos componentes e da aplicação, implementando assim um contrato em nível de aplicação.Os mecanismos necessários para esta interação são providos pelo framework.

- **Composição heterogênea**
  - Composição de componentes através de diferente frameworks, suportada pelos frameworks em camadas.É necessária a implementação de contratos em nível de sistema e de contratos ponte de modo a possibilitar a interação de modelos de componentes bastante genéricos.

- **Extensão de framework**

- Os frameworks também podem ser considerados como componentes, e desta forma, podem ser compostos como outros componentes.Permite a parametrização do comportamento do framework através de seus plug-ins.São cada vez mais comuns em frameworks comerciais.

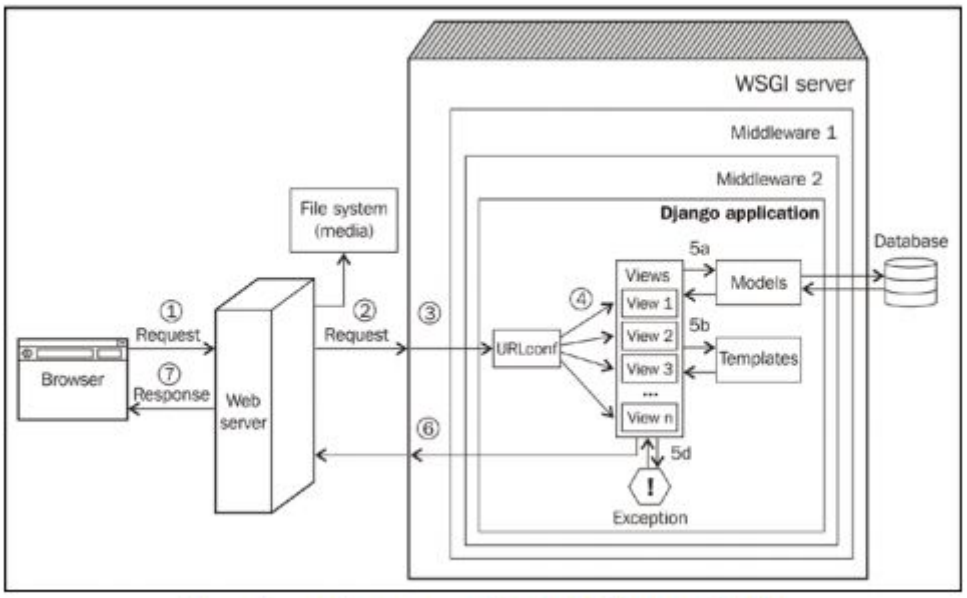
- **União de componentes.**
  - Nesse tipo de composição, os componentes são compostos e outras uniões parciais de componentes, as quais podem conter um ou mais componentes. Estas composições entre componentes implementam contratos em nível de aplicação.

4.2. Comunicação entre Componentes/Frameworks

A comunicação entre componentes / frameworks se dá por meio das interações que são especificadas pelos compromissos por meio dos contratos em diferentes níveis de formatos. Esses contratos definem como cada um deve se comportar para que a comunicação ocorra sem problemas, e todos os parâmetros necessários sejam corretamente enviados às partes interessadas.

O framework Django se utiliza das diversas formas de comunicação entre componentes/frameworks em sua implementação, provendo assim que a junção destes seja realizada da forma mais simples possível ao usuário que está utilizando os serviços disponibilizados pelo mesmo.

Internamente no que podemos chamar de core do Django, ocorre uma comunicação entre os diversos componentes que constituem o framework. Essa comunicação pode ser vista na imagem a seguir, que representa como uma requisição web é processada dentro de uma aplicação Django:



How web requests are processed in a typical Django application

[5] Django Design Patterns, página 5

Além da comunicação interna dos componentes no “core” do framework, o mesmo também dispõe de mecanismos para que a comunicação com outros frameworks seja simples. Documentando os padrões que precisam ser seguidos e assim permitindo ao usuário realizar a comunicação que desejar. Exemplos dessa comunicações entre frameworks está na comunicação do Django com o Django REST que tem sua configuração simples e a comunicação do Django com frameworks de gerenciamento de base de dados.

Na documentação do framework não existem diagramas que representem essas comunicações de forma clara e direta. As informações em relação a comunicação dos componentes estão presentes em forma de texto durante a explicação de cada funcionalidade do framework. Cabendo assim ao usuário do framework compreender as especificidades e utilizá-las da forma que considerar mais coerente com suas necessidades.

Para que um componente seja utilizado por outro é necessário que se realize uma declaração de importação no arquivo, indicando qual componente do Django você irá utilizar e quais as funcionalidades do mesmo deseja obter.

5. Referências

[1] https://docs.djangoproject.com/pt-br/2.0/topics/forms/
[2] https://django-portuguese.readthedocs.io/en/1.0/topics/forms/modelforms.html
[3] http://www.pucrs.br/facin-prov/wp-content/uploads/sites/19/2016/03/tr026.pdf
[4] SPAGNOLI, L. A.; BECKER, K. Um Estudo Sobre o Desenvolvimento Baseado em Componentes. Porto Alegre. 2003.
[5] RAVINDRAN, Arun. Django Design Patterns and Best Practices.Packt Publishing, 2015.
[6] BACHMANN, Felix et al. Volume II: Technical Concepts of Component-Based Software Engineering-2nd Edition.Disponível em: <https://dimap.ufrn.br/~jair/ES/artigos/cbse2.pdf>
[7] https://tutorial.djangogirls.org/pt/django/