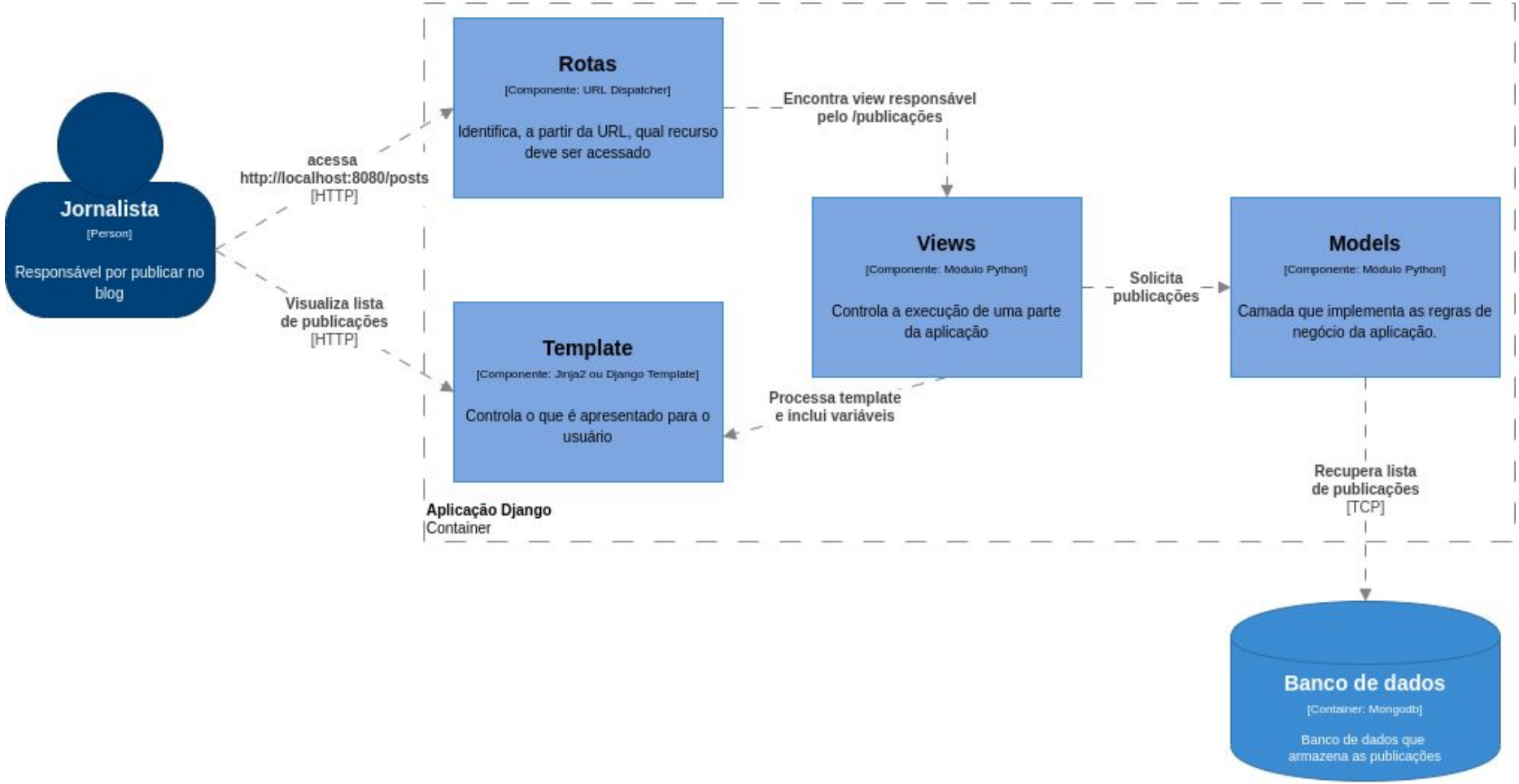
Desenvolvendo aplicativos Django mais robustos com sinais, middlewares customizados e decoradores.

Agenda

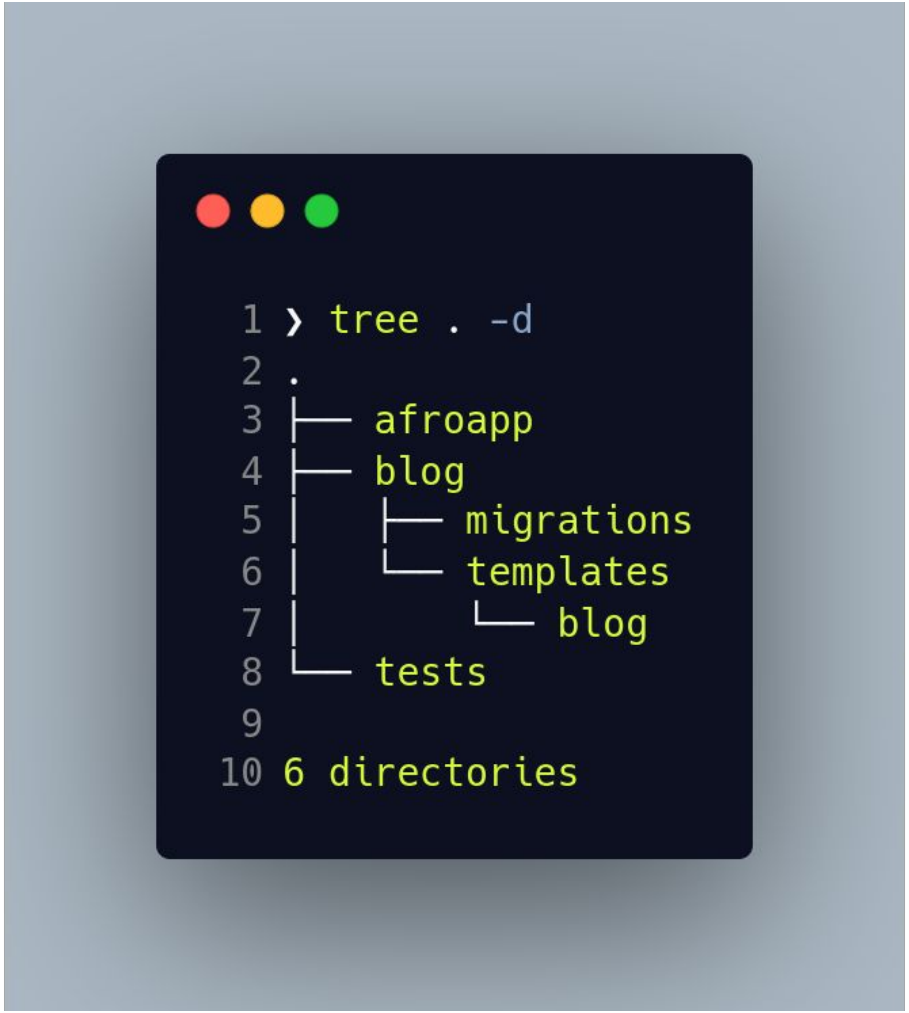
1. Sobre mim.
2. Arquitetura MVT.
3. AfroApp.
4. Sinais.
5. Middlewares.
6. Decoradores.

Sobre mim

Arquitertura MVT


Jornalista
[Person]
Responsável por publicar no blog
acessa http://localhost:8080/posts [HTTP]
Visualiza lista de publicações [HTTP]
Rotas
[Componente: URL Dispatcher]
Identifica, a partir da URL, qual recurso deve ser acessado
Encontra view responsável pelo /publicações
Views
[Componente: Módulo Python]
Controla a execução de uma parte da aplicação
Solicita publicações
Models
[Componente: Módulo Python]
Camada que implementa as regras de negócio da aplicação.
Template
[Componente: Jinja2 ou Django Template]
Controla o que é apresentado para o usuário
Processa template e inclui variáveis
Recupera lista de publicações [TCP]
Aplicação Django
Container
Banco de dados
[Container: Mongodb]
Banco de dados que armazena as publicações

AfroApp 



- **afroapp**: é o nosso projeto Django. Ele controla como a aplicação será executada, além de gerenciar as configurações do ambiente.
- **blog**: é o nosso app Django. Dentro dele teremos a estrutura MVT.
- **db.sqlite3**: banco de dados da aplicação.
- **pyproject.yml**: versionamento das dependências. Utilizamos poetry como gestor de dependências.

Sinais

Requisito

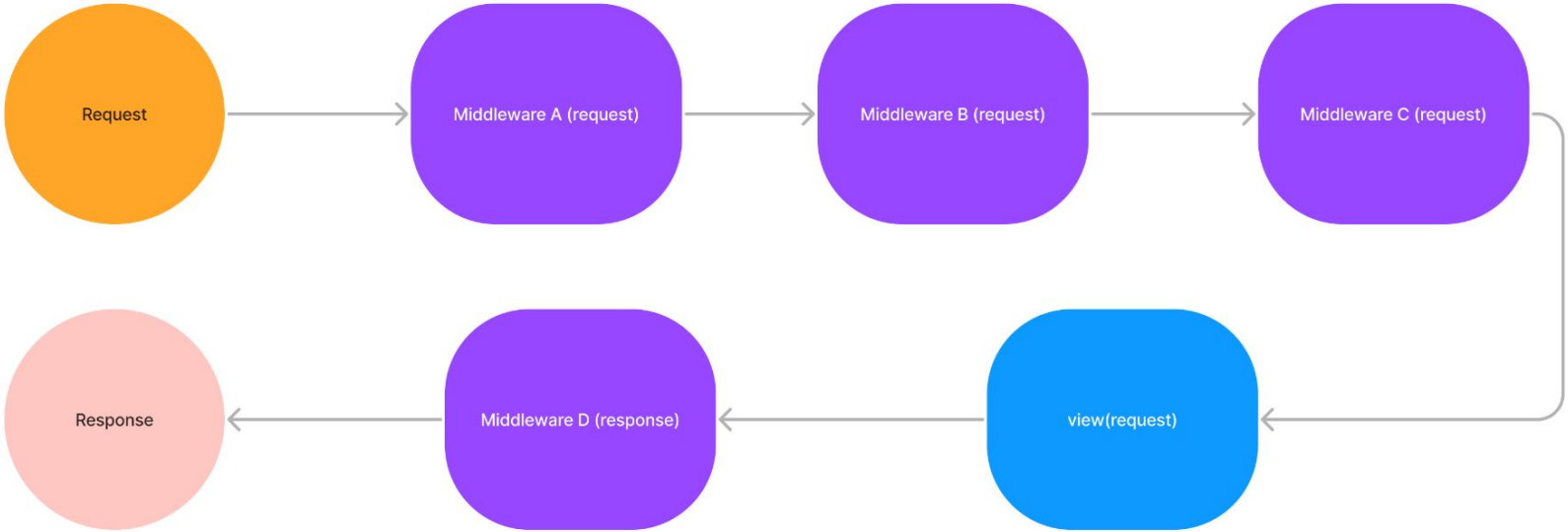
Toda vez que um jornalista criar uma nova publicação na nossa aplicação um email é enviado para uma lista de revisão. Há três formas de implementar essa nova funcionalidade...

```python
def create_without_signal_v1(request):
    post = Post(title="primeiro post", body="esse é o primeiro post do nosso blog")
    post.save()
    send_mail(
        'Um novo post foi criado',
        f'Título do post: {post.title}',
        'davidcarlos@pencillabs.com.br',
        ['g1@globo.com'],
        fail_silently=True,
    )
    return render(request, 'blog/create.html', {'post': post})
```

```python
# views.py
def create_without_signal_v2(request):
    from .helper import MailerHelper

    post = Post(title="primeiro post", body="esse é o primeiro post do nosso blog")
    post.save()
    MailerHelper(post).send()
    return render(request, "blog/create.html", {"post": post})


# helper.py
@dataclass
class MailerHelper:
    post_title: str

    def send(self):
        send_mail(
            "Um novo post foi criado",
            f"Título do post: {self.post_title}",
            "davidcarlos@pencillabs.com.br",
            ["g1@globo.com"],
            fail_silently=True,
        )

```

```python
# views.py
def create_with_signal(request):
    post = Post(title="primeiro post", body="esse é o primeiro post do nosso blog")
    post.save()
    return render(request, "blog/create.html", {"post": post})



# signals.py
from django.db.models.signals import post_save
from django.dispatch import receiver
from django.core.mail import send_mail

from blog.models import Post


@receiver(post_save, sender=Post)
def send_email_callback(sender, **kwargs):
    post = kwargs.get("instance")
    print(post.title)
    send_mail(
        "Um novo post foi criado",
        f"Título do post: {post.title}",
        "davidcarlos@pencillabs.com.br",
        ["g1@globo.com"],
        fail_silently=True,
    )
```

Middleware


Request
Middleware A (request)
Middleware B (request)
Middleware C (request)
view(request)
Middleware D (response)
Response

Requisito

Toda vez que o usuário acessar um post ou a lista de publicações, a aplicação precisa apresentar a cotação do dólar naquele momento específico. Podemos implementar isso de duas formas:

```python
# views.py
def detail_without_middleware(request, post_id):
    from .helper import DollarHelper

    post = Post.objects.get(id=post_id)
    return render(
        request,
        "blog/detail.html",
        {"post": post, "dollar_today": DollarHelper.get_dollar_today()},
    )

#helper.py
class DollarHelper:
    @staticmethod
    def get_dollar_today():
        # acessa alguma API remota para recuperar a cotação do dolar
        return 9

```

```python
# middleware.py
from blog.helper import DollarHelper


class DollarMiddleware(object):
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        request.dollar_today = DollarHelper.get_dollar_today()
        response = self.get_response(request)
        return response

#views.py
def detail_middleware(request, post_id):
    post = Post.objects.get(id=post_id)
    return render(request, "blog/detail_middleware.html", {"post": post})

#settings.py
MIDDLEWARE = [
    "django.middleware.security.SecurityMiddleware",
    "django.contrib.sessions.middleware.SessionMiddleware",
    "django.middleware.common.CommonMiddleware",
    "django.middleware.csrf.CsrfViewMiddleware",
    "django.contrib.auth.middleware.AuthenticationMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware",
    "django.middleware.clickjacking.XFrameOptionsMiddleware",
    "blog.middleware.DollarMiddleware",
]
```

Decorators

Requisito

Apenas jornalistas que tenham um identificador válido podem publicar no blog. Esse identificador será informado via url.

```python
#views.py
def create_without_decorator(request):
    unique_id = request.GET.get('id')
    if IdValidatorHelper.validate(unique_id):
        post = Post(title="primeiro post", body="esse é o primeiro post do nosso blog")
        post.save()
        return render(request, "blog/create.html", {"post": post})
    return render(request, "blog/invalid_id_error.html")

# helper.py
class IdValidatorHelper:
    @staticmethod
    def validate(id):
        # alguma regra maluca de validação
        return id and int(id) > 100
```

```python
#views.py
from blog.decorators import user_has_valid_id

@user_has_valid_id
def create_with_decorator(request):
    post = Post(title="primeiro post", body="esse é o primeiro post do nosso blog")
    post.save()
    return render(request, "blog/create.html", {"post": post})

# decorators.py
from blog.helper import IdValidatorHelper
from django.shortcuts import render
from functools import wraps

def user_has_valid_id(view):
    # copia os métodos mágicos (__name__) para o decorador,
    # lembrando que tudo em python herda de object
    @wraps(view)
    def inner(request, *args, **kwargs):
        unique_id = request.GET.get('id')
        if IdValidatorHelper.validate(unique_id):
            return view(request, *args, **kwargs)
        return render(request, "blog/invalid_id_error.html")
    return inner
```

Referências

- https://www.programiz.com/python-programming/decorator
- https://github.com/django/django/blob/main/django/views/decorators/http.py
- https://docs.djangoproject.com/en/4.0/topics/http/decorators/
- https://docs.djangoproject.com/en/4.0/topics/signals/
- https://docs.djangoproject.com/en/4.0/topics/http/middleware/
- https://python-poetry.org/docs/basic-usage/
- https://carbon.now.sh/

Obrigado!

@mr_davidcarlos

@mr_davidcarlos

https://davidcarlos.me

Roda de conversa
de Homens Negros

Pencillabs
Desenhando o futuro, juntos.