AULA 08

TESTES AUTOMATIZADOS NO DJANGO

O QUE VEREMOS HOJE

01 TESTES AUTOMATIZADOS

02 ESTRUTURAS DE TESTES NO DJANGO

03 TESTES DAS VIEWS

04 TESTES DOS MODELS

TESTES AUTOMATIZADOS

Testes são ferramentas cruciais para garantir a qualidade do código, encontrando erros e falhas antes que eles cheguem aos usuários finais.

Eles verificam se o código funciona como esperado, proporcionando confiança e estabilidade.

O ciclo de vida do teste envolve a identificação de requisitos, planejamento de casos de teste, execução de testes, análise de resultados e correção de defeitos encontrados.

TIPOS DE TESTE

- **Testes de unidade**: focam em testar partes isoladas do código, como funções ou métodos, para garantir que cada unidade funcione corretamente.

- **Testes de integração**: verificam como diferentes partes do código interagem entre si, garantindo que o sistema funcione como um todo.

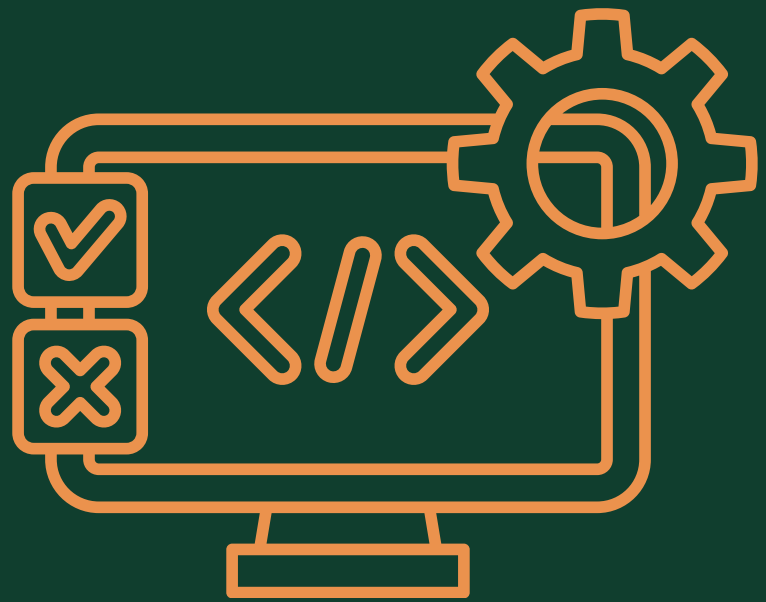
- **Testes funcionais**: avaliam o sistema do ponto de vista do usuário final, garantindo que as funcionalidades entregues atendam aos requisitos e funcionem conforme o esperado.

TESTES NO DJANGO

O Django já possui uma estrutura pronta para testes:

- Cada aplicação tem um arquivo **tests.py** por padrão.
- Os testes usam a biblioteca **unittest** do Python.

Além disso, quando são executados os testes, o Django cria um banco de dados temporário que é destruído após a execução dos testes. Isso é para evitar impactar o banco de dados real.



ESTRUTURA DOS TESTES

No Django, um teste é composto por:

- **Setup**: Configurações iniciais no método setUp(). Nele é possível criar objetos no banco de dados temporário que serão usados em seus testes.

- **Testes**: Métodos que verificam funcionalidades específicas. Todos devem começar com o prefixo test_, para que o Django consiga identificá-los automaticamente.

- **Asserts**: Verificações que comparam o resultado esperado com o resultado obtido (ex.: assertEqual).

```python
from django.test import TestCase

class ExampleTest(TestCase):

    # Método que é executado antes de cada teste
    def setUp(self):
        # Configurações antes dos testes, como criar objetos no banco de dados para serem usados nos testes
        pass

    # Todos testes devem começar com a palavra test
    def test_algo(self):
        self.assertEqual(1 + 1, 2)
```

TESTES DE VIEWS

No Django, as views são responsáveis por toda lógica de conexão entre o banco de dados e os templates.

Os casos de uso mais comuns em testes das views são:

- Verificar o status de resposta (200, 404, 302, etc.)
- Verificar se as views estão retornando os templates corretos.
- Verificar se usuários não autenticados são redirecionados.

TESTES DE VIEWS

No teste das views é necessário fazer a requisição para URL e verificar a resposta usando o assertEqual para verificar o status code e o assertTemplateUsed para verificar o template.

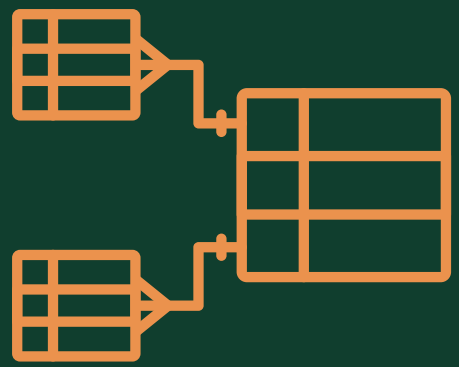
```python
from django.test import TestCase, Client
from .models import Event


class EventViewTest(TestCase):
    def setUp(self):
        self.client = Client()
        self.event = Event.objects.create(
            title='Django Event',
            description='Event to learn Django',
            date='2018-09-01T12:00:00Z',
            location='Infinity School'
        )

    def test_list_events(self):
        response = self.client.get('/events/')
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'list.html')
```

TESTES DO MODEL

Os testes de models garantem que as tabelas no banco de dados estejam funcionando corretamente. Eles verificam se os objetos são criados com os atributos esperados, valores padrões estão corretos e se métodos, se houver, retornam os resultados apropriados.

Os casos de uso mais comuns em testes de model são:

- Verificar se o objeto foi salvo corretamente no banco.
- Validar se os valores salvos correspondem ao esperado.
- Garantir que os valores padrão definidos no model estão funcionando.
- Verificar se os campos obrigatórios estão sendo validados.



TESTES DE MODEL

Nos testes de model, utilizamos o método setUp para criar dados que serão utilizados nos testes, garantindo um ambiente controlado.

Nas funções de teste, acessamos esses dados através de **self** para validar se os objetos foram criados corretamente, se seus atributos estão com valores esperados

Para isso, usamos métodos como **assertEqual** para comparar valores e **assertIsNotNone** para verificar a criação de IDs ou atributos obrigatórios.

```python
from django.test import TestCase
from .models import Event

class EventTest(TestCase):
    def setUp(self):
        # Cria um novo evento para ser usado nos testes
        self.event = Event.objects.create(
            title='Django Event',
            description='Event to learn Django',
            date='2018-09-01T12:00:00Z',
            location='Infinity School'
        )

    def test_create(self):
        # Verifica se o evento foi criado corretamente
        self.assertEqual(self.event.title, 'Django Event')
        self.assertEqual(self.event.description, 'Event to learn Django')
        self.assertIsNotNone(self.event.id)
```

TESTES NO MODELS - READ, UPDATE E DELETE

- **Leitura (Read)**: Verificar se os objetos podem ser recuperados corretamente do banco.
- **Atualização (Update)**: Testar se atributos podem ser modificados e salvos corretamente.
- **Exclusão (Delete)**: Garantir que os objetos podem ser removidos do banco.

```python

def test_read_event(self):
    """Teste de leitura de um objeto do banco de dados"""
    event = Event.objects.get(title='Django Event')
    self.assertEqual(event.title, 'Django Event')
    self.assertEqual(event.description, 'Event to learn Django')

def test_update_event(self):
    """Teste de atualização de um objeto"""
    event = Event.objects.get(title='Django Event')
    event.title = 'Updated Event'
    event.save()

    updated_event = Event.objects.get(id=event.id)
    self.assertEqual(updated_event.title, 'Updated Event')

def test_delete_event(self):
    """Teste de exclusão de um objeto"""
    event = Event.objects.get(title='Django Event')
    event.delete()

    with self.assertRaises(Event.DoesNotExist):
        Event.objects.get(title='Django Event')

```

ATIVIDADE PRÁTICA

No App de Tarefas implemente os testes no model de Tarefas e implemente testes de response e de template em pelo menos duas views.

SUGESTÕES DE CONTEÚDOS

- **Criação de APIs no Django**
  - Views que retornam JSON, ao invés de HTML
  - Django REST Framework
- **Envio de e-mails com Django**
  - Aprender a configurar e-mails no Django
- **Conexão do Django com Banco de Dados MySQL**
  - mysqlclient
- **Datas e Horários (Timezones e Agendamento)**
  - Exibição e formação de datas no template
  - Tarefas agendados para envio de notificações
- **Upload de Arquivos**
  - Campos FileField e ImageField
  - Configuração pra o Django salvar arquivos
- **Paginação**
  - Dividir grandes listas de dados em várias páginas
  - Django Paginator