**Plano de Testes para o Projeto OMAUM**

**Índice**

1. [Visão Geral](vscode-webview://06hbuukvsb50cf0ntpl95qmqkaue7gsbb5ef56lkmqpsj4kouj33/index.html?id=14b73ff2-6d31-45d6-a013-b69d1ebfe8cc&parentId=1&origin=374b4b85-d931-417f-ba03-1c7d73f1dc42&swVersion=4&extensionId=sourcegraph.cody-ai&platform=electron&vscode-resource-base-authority=vscode-resource.vscode-cdn.net&parentOrigin=vscode-file%3A%2F%2Fvscode-app&purpose=webviewView)
2. [Estratégia de Testes](vscode-webview://06hbuukvsb50cf0ntpl95qmqkaue7gsbb5ef56lkmqpsj4kouj33/index.html?id=14b73ff2-6d31-45d6-a013-b69d1ebfe8cc&parentId=1&origin=374b4b85-d931-417f-ba03-1c7d73f1dc42&swVersion=4&extensionId=sourcegraph.cody-ai&platform=electron&vscode-resource-base-authority=vscode-resource.vscode-cdn.net&parentOrigin=vscode-file%3A%2F%2Fvscode-app&purpose=webviewView)
3. [Testes Unitários](vscode-webview://06hbuukvsb50cf0ntpl95qmqkaue7gsbb5ef56lkmqpsj4kouj33/index.html?id=14b73ff2-6d31-45d6-a013-b69d1ebfe8cc&parentId=1&origin=374b4b85-d931-417f-ba03-1c7d73f1dc42&swVersion=4&extensionId=sourcegraph.cody-ai&platform=electron&vscode-resource-base-authority=vscode-resource.vscode-cdn.net&parentOrigin=vscode-file%3A%2F%2Fvscode-app&purpose=webviewView)
4. [Testes de Integração](vscode-webview://06hbuukvsb50cf0ntpl95qmqkaue7gsbb5ef56lkmqpsj4kouj33/index.html?id=14b73ff2-6d31-45d6-a013-b69d1ebfe8cc&parentId=1&origin=374b4b85-d931-417f-ba03-1c7d73f1dc42&swVersion=4&extensionId=sourcegraph.cody-ai&platform=electron&vscode-resource-base-authority=vscode-resource.vscode-cdn.net&parentOrigin=vscode-file%3A%2F%2Fvscode-app&purpose=webviewView)
5. [Testes de Ponta a Ponta](vscode-webview://06hbuukvsb50cf0ntpl95qmqkaue7gsbb5ef56lkmqpsj4kouj33/index.html?id=14b73ff2-6d31-45d6-a013-b69d1ebfe8cc&parentId=1&origin=374b4b85-d931-417f-ba03-1c7d73f1dc42&swVersion=4&extensionId=sourcegraph.cody-ai&platform=electron&vscode-resource-base-authority=vscode-resource.vscode-cdn.net&parentOrigin=vscode-file%3A%2F%2Fvscode-app&purpose=webviewView)
6. [Proposta de Dados de Teste](vscode-webview://06hbuukvsb50cf0ntpl95qmqkaue7gsbb5ef56lkmqpsj4kouj33/index.html?id=14b73ff2-6d31-45d6-a013-b69d1ebfe8cc&parentId=1&origin=374b4b85-d931-417f-ba03-1c7d73f1dc42&swVersion=4&extensionId=sourcegraph.cody-ai&platform=electron&vscode-resource-base-authority=vscode-resource.vscode-cdn.net&parentOrigin=vscode-file%3A%2F%2Fvscode-app&purpose=webviewView)
7. [Cronograma e Priorização](vscode-webview://06hbuukvsb50cf0ntpl95qmqkaue7gsbb5ef56lkmqpsj4kouj33/index.html?id=14b73ff2-6d31-45d6-a013-b69d1ebfe8cc&parentId=1&origin=374b4b85-d931-417f-ba03-1c7d73f1dc42&swVersion=4&extensionId=sourcegraph.cody-ai&platform=electron&vscode-resource-base-authority=vscode-resource.vscode-cdn.net&parentOrigin=vscode-file%3A%2F%2Fvscode-app&purpose=webviewView)

**Visão Geral**

O projeto OMAUM é um sistema de gestão educacional/organizacional com diversos módulos interconectados. Este plano de testes visa garantir a qualidade e confiabilidade do sistema através de uma abordagem abrangente de testes em diferentes níveis.

**Módulos Principais Identificados**

* Alunos
* Atividades (Acadêmicas e Ritualísticas)
* Cargos
* Cursos
* Frequências
* Iniciações
* Matrículas
* Notas
* Pagamentos
* Presenças
* Punições
* Turmas

**Estratégia de Testes**

**Ferramentas a Serem Utilizadas**

* **Testes Unitários**: pytest
* **Testes de Integração**: django.test.TestCase, SimpleClient
* **Testes de Ponta a Ponta**: Selenium
* **Cobertura de Código**: pytest-cov
* **Dados de Teste**: Factory Boy, Faker

**Ambientes de Teste**

1. **Ambiente de Desenvolvimento**: Para testes unitários e de integração durante o desenvolvimento
2. **Ambiente de Teste**: Para testes de integração e ponta a ponta
3. **Ambiente de Homologação**: Para testes de aceitação

**Configuração Inicial do Ambiente de Testes**

import pytest

from django.contrib.auth.models import User

from django.test import Client

@pytest.fixture

def usuario\_autenticado():

"""Cria um usuário e retorna um cliente autenticado."""

user = User.objects.create\_user(

username='testuser',

password='testpassword',

email='test@example.com'

)

client = Client()

client.login(username='testuser', password='testpassword')

return client, user

@pytest.fixture

def usuario\_admin():

"""Cria um usuário administrador e retorna um cliente autenticado."""

admin = User.objects.create\_superuser(

username='admin',

password='adminpassword',

email='admin@example.com'

)

client = Client()

client.login(username='admin', password='adminpassword')

return client, admin

Copy

conftest.py

**Testes Unitários**

**Passo 1: Configuração dos Testes Unitários**

import pytest

from django.conf import settings

def test\_debug\_setting():

"""Verifica se o DEBUG está desativado em ambiente de teste."""

assert settings.DEBUG is False, "DEBUG deve estar desativado em ambiente de teste"

def test\_installed\_apps():

"""Verifica se todas as aplicações necessárias estão instaladas."""

required\_apps = [

'alunos',

'atividades',

'cargos',

'core',

'cursos',

'frequencias',

'iniciacoes',

'matriculas',

'notas',

'pagamentos',

'presencas',

'punicoes',

'turmas',

]

for app in required\_apps:

assert app in settings.INSTALLED\_APPS, f"A aplicação {app} não está instalada"

Copy

test\_config.py

**Passo 2: Testes para o Módulo de Alunos**

import pytest

from alunos.models import Aluno

from django.core.exceptions import ValidationError

from django.db import IntegrityError

@pytest.mark.django\_db

class TestAlunoModel:

"""Testes para o modelo Aluno."""

def test\_criar\_aluno\_valido(self):

"""Testa a criação de um aluno com dados válidos."""

aluno = Aluno.objects.create(

cpf="12345678900",

nome="João da Silva",

email="joao@exemplo.com",

data\_nascimento="1990-01-01",

sexo="M",

situacao="ativo"

)

assert aluno.pk is not None

assert aluno.nome == "João da Silva"

assert aluno.cpf == "12345678900"

def test\_cpf\_unico(self):

"""Testa se o CPF é único."""

Aluno.objects.create(

cpf="12345678900",

nome="João da Silva",

email="joao@exemplo.com",

data\_nascimento="1990-01-01"

)

with pytest.raises(IntegrityError):

Aluno.objects.create(

cpf="12345678900", # CPF duplicado

nome="Maria Souza",

email="maria@exemplo.com",

data\_nascimento="1992-05-15"

)

def test\_aluno\_str(self):

"""Testa a representação string do aluno."""

aluno = Aluno.objects.create(

cpf="12345678900",

nome="João da Silva",

email="joao@exemplo.com",

data\_nascimento="1990-01-01"

)

assert str(aluno) == "João da Silva"

Copy

test\_models.py

import pytest

from alunos.forms import AlunoForm

from datetime import date

class TestAlunoForm:

"""Testes para o formulário de Aluno."""

def test\_form\_valido(self):

"""Testa se o formulário é válido com dados corretos."""

form\_data = {

'cpf': '12345678900',

'nome': 'João da Silva',

'email': 'joao@exemplo.com',

'data\_nascimento': '1990-01-01',

'sexo': 'M',

'situacao': 'ativo',

}

form = AlunoForm(data=form\_data)

assert form.is\_valid(), f"Formulário inválido: {form.errors}"

def test\_form\_invalido\_campos\_obrigatorios(self):

"""Testa se o formulário é inválido quando campos obrigatórios estão ausentes."""

form\_data = {

'email': 'joao@exemplo.com',

}

form = AlunoForm(data=form\_data)

assert not form.is\_valid()

assert 'cpf' in form.errors

assert 'nome' in form.errors

assert 'data\_nascimento' in form.errors

def test\_form\_invalido\_email(self):

"""Testa se o formulário é inválido com email incorreto."""

form\_data = {

'cpf': '12345678900',

'nome': 'João da Silva',

'email': 'email\_invalido',

'data\_nascimento': '1990-01-01',

'sexo': 'M',

}

form = AlunoForm(data=form\_data)

assert not form.is\_valid()

assert 'email' in form.errors

def test\_form\_data\_nascimento\_futura(self):

"""Testa se o formulário rejeita datas de nascimento futuras."""

future\_date = date.today().replace(year=date.today().year + 1).strftime('%Y-%m-%d')

form\_data = {

'cpf': '12345678900',

'nome': 'João da Silva',

'email': 'joao@exemplo.com',

'data\_nascimento': future\_date,

'sexo': 'M',

}

form = AlunoForm(data=form\_data)

assert not form.is\_valid()

assert 'data\_nascimento' in form.errors

Copy

test\_forms.py

**Passo 3: Testes para o Módulo de Atividades**

import pytest

from django.utils import timezone

from atividades.models import AtividadeAcademica, AtividadeRitualistica

from turmas.models import Turma

from django.core.exceptions import ValidationError

@pytest.mark.django\_db

class TestAtividadeAcademicaModel:

"""Testes para o modelo AtividadeAcademica."""

def test\_criar\_atividade\_academica(self):

"""Testa a criação de uma atividade acadêmica."""

atividade = AtividadeAcademica.objects.create(

nome="Aula de Filosofia",

descricao="Introdução à Filosofia",

data\_inicio=timezone.now(),

responsavel="Prof. Silva",

tipo\_atividade="aula",

status="agendada"

)

assert atividade.pk is not None

assert atividade.nome == "Aula de Filosofia"

assert atividade.status == "agendada"

def test\_atividade\_academica\_str(self):

"""Testa a representação string da atividade acadêmica."""

atividade = AtividadeAcademica.objects.create(

nome="Aula de Filosofia",

data\_inicio=timezone.now(),

tipo\_atividade="aula",

status="agendada"

)

assert str(atividade) == "Aula de Filosofia"

def test\_atividade\_academica\_com\_turmas(self):

"""Testa a associação de turmas a uma atividade acadêmica."""

turma1 = Turma.objects.create(nome="Turma A", codigo="TA-001")

turma2 = Turma.objects.create(nome="Turma B", codigo="TB-001")

atividade = AtividadeAcademica.objects.create(

nome="Aula de Filosofia",

data\_inicio=timezone.now(),

tipo\_atividade="aula",

status="agendada"

)

atividade.turmas.add(turma1, turma2)

assert atividade.turmas.count() == 2

assert turma1 in atividade.turmas.all()

assert turma2 in atividade.turmas.all()

@pytest.mark.django\_db

class TestAtividadeRitualisticaModel:

"""Testes para o modelo AtividadeRitualistica."""

def test\_criar\_atividade\_ritualistica(self):

"""Testa a criação de uma atividade ritualística."""

turma = Turma.objects.create(nome="Turma A", codigo="TA-001")

atividade = AtividadeRitualistica.objects.create(

nome="Ritual de Iniciação",

descricao="Ritual para novos membros",

data=timezone.now().date(),

hora\_inicio="19:00",

hora\_fim="21:00",

local="Templo Principal",

turma=turma

)

assert atividade.pk is not None

assert atividade.nome == "Ritual de Iniciação"

assert atividade.turma == turma

def test\_atividade\_ritualistica\_str(self):

"""Testa a representação string da atividade ritualística."""

turma = Turma.objects.create(nome="Turma A", codigo="TA-001")

data = timezone.now().date()

atividade = AtividadeRitualistica.objects.create(

nome="Ritual de Iniciação",

data=data,

hora\_inicio="19:00",

hora\_fim="21:00",

local="Templo Principal",

turma=turma

)

assert str(atividade) == f"Ritual de Iniciação - {data}"

def test\_validacao\_horarios(self):

"""Testa se a hora de início é anterior à hora de fim."""

turma = Turma.objects.create(nome="Turma A", codigo="TA-001")

atividade = AtividadeRitualistica(

nome="Ritual de Iniciação",

data=timezone.now().date(),

hora\_inicio="21:00", # Hora de início posterior à hora de fim

hora\_fim="19:00",

local="Templo Principal",

turma=turma

)

with pytest.raises(ValidationError):

atividade.full\_clean()

Copy

test\_models.py

import pytest

from atividades.forms import AtividadeAcademicaForm, AtividadeRitualisticaForm

from turmas.models import Turma

from django.utils import timezone

@pytest.mark.django\_db

class TestAtividadeAcademicaForm:

"""Testes para o formulário de AtividadeAcademica."""

def test\_form\_valido(self):

"""Testa se o formulário é válido com dados corretos."""

turma = Turma.objects.create(nome="Turma A", codigo="TA-001")

form\_data = {

'nome': 'Aula de Filosofia',

'descricao': 'Introdução à Filosofia',

'data\_inicio': timezone.now().strftime('%Y-%m-%d'),

'data\_fim': (timezone.now() + timezone.timedelta(hours=2)).strftime('%Y-%m-%d'),

'responsavel': 'Prof. Silva',

'local': 'Sala 101',

'tipo\_atividade': 'aula',

'status': 'agendada',

'turmas': [turma.id],

}

form = AtividadeAcademicaForm(data=form\_data)

assert form.is\_valid(), f"Formulário inválido: {form.errors}"

def test\_form\_invalido\_campos\_obrigatorios(self):

"""Testa se o formulário é inválido quando campos obrigatórios estão ausentes."""

form\_data = {

'descricao': 'Introdução à Filosofia',

}

form = AtividadeAcademicaForm(data=form\_data)

assert not form.is\_valid()

assert 'nome' in form.errors

assert 'data\_inicio' in form.errors

def test\_form\_data\_fim\_anterior\_data\_inicio(self):

"""Testa se o formulário rejeita data de fim anterior à data de início."""

form\_data = {

'nome': 'Aula de Filosofia',

'data\_inicio': timezone.now().strftime('%Y-%m-%d'),

'data\_fim': (timezone.

Copy

test\_forms.py

prossiga até o final

PromptsClaude 3.7 Sonnet

**Context— none fetched**

**Plano de Testes para o Projeto OMAUM (Continuação)**

import pytest

from atividades.forms import AtividadeAcademicaForm, AtividadeRitualisticaForm

from turmas.models import Turma

from django.utils import timezone

@pytest.mark.django\_db

class TestAtividadeAcademicaForm:

"""Testes para o formulário de AtividadeAcademica."""

def test\_form\_valido(self):

"""Testa se o formulário é válido com dados corretos."""

turma = Turma.objects.create(nome="Turma A", codigo="TA-001")

form\_data = {

'nome': 'Aula de Filosofia',

'descricao': 'Introdução à Filosofia',

'data\_inicio': timezone.now().strftime('%Y-%m-%d'),

'data\_fim': (timezone.now() + timezone.timedelta(hours=2)).strftime('%Y-%m-%d'),

'responsavel': 'Prof. Silva',

'local': 'Sala 101',

'tipo\_atividade': 'aula',

'status': 'agendada',

'turmas': [turma.id],

}

form = AtividadeAcademicaForm(data=form\_data)

assert form.is\_valid(), f"Formulário inválido: {form.errors}"

def test\_form\_invalido\_campos\_obrigatorios(self):

"""Testa se o formulário é inválido quando campos obrigatórios estão ausentes."""

form\_data = {

'descricao': 'Introdução à Filosofia',

}

form = AtividadeAcademicaForm(data=form\_data)

assert not form.is\_valid()

assert 'nome' in form.errors

assert 'data\_inicio' in form.errors

def test\_form\_data\_fim\_anterior\_data\_inicio(self):

"""Testa se o formulário rejeita data de fim anterior à data de início."""

form\_data = {

'nome': 'Aula de Filosofia',

'data\_inicio': timezone.now().strftime('%Y-%m-%d'),

'data\_fim': (timezone.now() - timezone.timedelta(days=1)).strftime('%Y-%m-%d'),

'tipo\_atividade': 'aula',

'status': 'agendada',

}

form = AtividadeAcademicaForm(data=form\_data)

assert not form.is\_valid()

assert 'data\_fim' in form.errors

@pytest.mark.django\_db

class TestAtividadeRitualisticaForm:

"""Testes para o formulário de AtividadeRitualistica."""

def test\_form\_valido(self):

"""Testa se o formulário é válido com dados corretos."""

turma = Turma.objects.create(nome="Turma A", codigo="TA-001")

form\_data = {

'nome': 'Ritual de Iniciação',

'descricao': 'Ritual para novos membros',

'data': timezone.now().date().strftime('%Y-%m-%d'),

'hora\_inicio': '19:00',

'hora\_fim': '21:00',

'local': 'Templo Principal',

'turma': turma.id,

}

form = AtividadeRitualisticaForm(data=form\_data)

assert form.is\_valid(), f"Formulário inválido: {form.errors}"

def test\_form\_invalido\_campos\_obrigatorios(self):

"""Testa se o formulário é inválido quando campos obrigatórios estão ausentes."""

form\_data = {

'descricao': 'Ritual para novos membros',

}

form = AtividadeRitualisticaForm(data=form\_data)

assert not form.is\_valid()

assert 'nome' in form.errors

assert 'data' in form.errors

assert 'hora\_inicio' in form.errors

assert 'hora\_fim' in form.errors

assert 'turma' in form.errors

def test\_form\_hora\_fim\_anterior\_hora\_inicio(self):

"""Testa se o formulário rejeita hora de fim anterior à hora de início."""

turma = Turma.objects.create(nome="Turma A", codigo="TA-001")

form\_data = {

'nome': 'Ritual de Iniciação',

'data': timezone.now().date().strftime('%Y-%m-%d'),

'hora\_inicio': '21:00',

'hora\_fim': '19:00', # Hora de fim anterior à hora de início

'local': 'Templo Principal',

'turma': turma.id,

}

form = AtividadeRitualisticaForm(data=form\_data)

assert not form.is\_valid()

assert 'hora\_fim' in form.errors or '\_\_all\_\_' in form.errors

Copy

test\_forms.py

**Passo 4: Testes para o Módulo de Turmas**

import pytest

from turmas.models import Turma

from django.utils import timezone

from django.core.exceptions import ValidationError

@pytest.mark.django\_db

class TestTurmaModel:

"""Testes para o modelo Turma."""

def test\_criar\_turma(self):

"""Testa a criação de uma turma."""

turma = Turma.objects.create(

nome="Turma de Filosofia 2023",

codigo="FIL-2023",

data\_inicio=timezone.now().date(),

status="A"

)

assert turma.pk is not None

assert turma.nome == "Turma de Filosofia 2023"

assert turma.codigo == "FIL-2023"

assert turma.status == "A"

def test\_turma\_str(self):

"""Testa a representação string da turma."""

turma = Turma.objects.create(

nome="Turma de Filosofia 2023",

codigo="FIL-2023"

)

assert str(turma) == "Turma de Filosofia 2023"

def test\_codigo\_turma\_unico(self):

"""Testa se o código da turma é único."""

Turma.objects.create(

nome="Turma A",

codigo="TURMA-001"

)

with pytest.raises(Exception): # Pode ser IntegrityError ou ValidationError

Turma.objects.create(

nome="Turma B",

codigo="TURMA-001" # Código duplicado

)

def test\_data\_fim\_posterior\_data\_inicio(self):

"""Testa se a data de fim é posterior à data de início."""

data\_inicio = timezone.now().date()

data\_fim = data\_inicio - timezone.timedelta(days=1) # Data de fim anterior à data de início

turma = Turma(

nome="Turma de Filosofia 2023",

codigo="FIL-2023",

data\_inicio=data\_inicio,

data\_fim=data\_fim,

status="A"

)

with pytest.raises(ValidationError):

turma.full\_clean()

Copy

test\_models.py

import pytest

from turmas.forms import TurmaForm

from django.utils import timezone

@pytest.mark.django\_db

class TestTurmaForm:

"""Testes para o formulário de Turma."""

def test\_form\_valido(self):

"""Testa se o formulário é válido com dados corretos."""

form\_data = {

'nome': 'Turma de Filosofia 2023',

'codigo': 'FIL-2023',

'data\_inicio': timezone.now().date().strftime('%Y-%m-%d'),

'status': 'A',

}

form = TurmaForm(data=form\_data)

assert form.is\_valid(), f"Formulário inválido: {form.errors}"

def test\_form\_invalido\_campos\_obrigatorios(self):

"""Testa se o formulário é inválido quando campos obrigatórios estão ausentes."""

form\_data = {

'status': 'A',

}

form = TurmaForm(data=form\_data)

assert not form.is\_valid()

assert 'nome' in form.errors

assert 'codigo' in form.errors

def test\_form\_data\_fim\_anterior\_data\_inicio(self):

"""Testa se o formulário rejeita data de fim anterior à data de início."""

form\_data = {

'nome': 'Turma de Filosofia 2023',

'codigo': 'FIL-2023',

'data\_inicio': timezone.now().date().strftime('%Y-%m-%d'),

'data\_fim': (timezone.now().date() - timezone.timedelta(days=1)).strftime('%Y-%m-%d'),

'status': 'A',

}

form = TurmaForm(data=form\_data)

assert not form.is\_valid()

assert 'data\_fim' in form.errors or '\_\_all\_\_' in form.errors

Copy

test\_forms.py

**Testes de Integração**

**Passo 5: Testes de Integração para o Módulo de Alunos**

from django.test import TestCase, Client

from django.urls import reverse

from alunos.models import Aluno

from django.contrib.auth.models import User

class AlunosViewsTestCase(TestCase):

"""Testes de integração para as views do módulo de alunos."""

def setUp(self):

"""Configuração inicial para os testes."""

# Criar um usuário para autenticação

self.user = User.objects.create\_user(

username='testuser',

password='testpassword'

)

# Criar alguns alunos para testes

self.aluno1 = Aluno.objects.create(

cpf="12345678900",

nome="João da Silva",

email="joao@exemplo.com",

data\_nascimento="1990-01-01"

)

self.aluno2 = Aluno.objects.create(

cpf="98765432100",

nome="Maria Souza",

email="maria@exemplo.com",

data\_nascimento="1992-05-15"

)

# Cliente para fazer requisições

self.client = Client()

def test\_listar\_alunos\_view\_nao\_autenticado(self):

"""Testa se a view de listagem de alunos redireciona para login quando não autenticado."""

response = self.client.get(reverse('alunos:listar\_alunos'))

self.assertEqual(response.status\_code, 302) # Redirecionamento

self.assertIn('/login/', response.url) # Redirecionamento para a página de login

def test\_listar\_alunos\_view\_autenticado(self):

"""Testa se a view de listagem de alunos funciona quando autenticado."""

# Fazer login

self.client.login(username='testuser', password='testpassword')

# Acessar a página de listagem de alunos

response = self.client.get(reverse('alunos:listar\_alunos'))

# Verificar se a resposta foi bem-sucedida

self.assertEqual(response.status\_code, 200)

# Verificar se os alunos estão no contexto

self.assertIn('alunos', response.context)

self.assertEqual(len(response.context['alunos']), 2)

# Verificar se os nomes dos alunos estão na página

self.assertContains(response, "João da Silva")

self.assertContains(response, "Maria Souza")

def test\_detalhar\_aluno\_view(self):

"""Testa se a view de detalhes do aluno funciona corretamente."""

# Fazer login

self.client.login(username='testuser', password='testpassword')

# Acessar a página de detalhes do aluno

response = self.client.get(

reverse('alunos:detalhar\_aluno', args=[self.aluno1.cpf])

)

# Verificar se a resposta foi bem-sucedida

self.assertEqual(response.status\_code, 200)

# Verificar se o aluno está no contexto

self.assertIn('aluno', response.context)

self.assertEqual(response.context['aluno'].cpf, self.aluno1.cpf)

# Verificar se os dados do aluno estão na página

self.assertContains(response, self.aluno1.nome)

self.assertContains(response, self.aluno1.email)

def test\_criar\_aluno\_view\_get(self):

"""Testa se a view de criação de aluno exibe o formulário corretamente."""

# Fazer login

self.client.login(username='testuser', password='testpassword')

# Acessar a página de criação de aluno

response = self.client.get(reverse('alunos:criar\_aluno'))

# Verificar se a resposta foi bem-sucedida

self.assertEqual(response.status\_code, 200)

# Verificar se o formulário está no contexto

self.assertIn('form', response.context)

def test\_criar\_aluno\_view\_post(self):

"""Testa se a view de criação de aluno cria um novo aluno corretamente."""

# Fazer login

self.client.login(username='testuser', password='testpassword')

# Dados para o novo aluno

novo\_aluno\_data = {

'cpf': '11122233344',

'nome': 'Pedro Oliveira',

'email': 'pedro@exemplo.com',

'data\_nascimento': '1995-10-20',

'sexo': 'M',

}

# Enviar o formulário para criar um novo aluno

response = self.client.post(

reverse('alunos:criar\_aluno'),

data=novo\_aluno\_data,

follow=True

)

# Verificar se o redirecionamento foi bem-sucedido

self.assertEqual(response.status\_code, 200)

# Verificar se o aluno foi criado no banco de dados

self.assertTrue(Aluno.objects.filter(cpf='11122233344').exists())

# Verificar se a mensagem de sucesso está na página

self.assertContains(response, "Aluno cadastrado com sucesso")

def test\_editar\_aluno\_view(self):

Copy

test\_views.py

prossiga

PromptsClaude 3.7 Sonnet

**Context— none fetched**

**Plano de Testes para o Projeto OMAUM (Continuação)**

def test\_editar\_aluno\_view(self):

"""Testa se a view de edição de aluno funciona corretamente."""

# Fazer login

self.client.login(username='testuser', password='testpassword')

# Acessar a página de edição do aluno

response = self.client.get(

reverse('alunos:editar\_aluno', args=[self.aluno1.cpf])

)

# Verificar se a resposta foi bem-sucedida

self.assertEqual(response.status\_code, 200)

# Verificar se o formulário está no contexto

self.assertIn('form', response.context)

# Verificar se os dados do aluno estão preenchidos no formulário

self.assertEqual(response.context['form'].instance.cpf, self.aluno1.cpf)

# Dados atualizados para o aluno

dados\_atualizados = {

'cpf': self.aluno1.cpf, # CPF não deve mudar

'nome': 'João Silva Atualizado',

'email': 'joao.atualizado@exemplo.com',

'data\_nascimento': '1990-01-01',

'sexo': 'M',

}

# Enviar o formulário para atualizar o aluno

response = self.client.post(

reverse('alunos:editar\_aluno', args=[self.aluno1.cpf]),

data=dados\_atualizados,

follow=True

)

# Verificar se o redirecionamento foi bem-sucedido

self.assertEqual(response.status\_code, 200)

# Verificar se o aluno foi atualizado no banco de dados

aluno\_atualizado = Aluno.objects.get(cpf=self.aluno1.cpf)

self.assertEqual(aluno\_atualizado.nome, 'João Silva Atualizado')

self.assertEqual(aluno\_atualizado.email, 'joao.atualizado@exemplo.com')

# Verificar se a mensagem de sucesso está na página

self.assertContains(response, "Aluno atualizado com sucesso")

def test\_excluir\_aluno\_view(self):

"""Testa se a view de exclusão de aluno funciona corretamente."""

# Fazer login

self.client.login(username='testuser', password='testpassword')

# Acessar a página de confirmação de exclusão

response = self.client.get(

reverse('alunos:excluir\_aluno', args=[self.aluno1.cpf])

)

# Verificar se a resposta foi bem-sucedida

self.assertEqual(response.status\_code, 200)

# Verificar se o aluno está no contexto

self.assertIn('aluno', response.context)

# Enviar a confirmação de exclusão

response = self.client.post(

reverse('alunos:excluir\_aluno', args=[self.aluno1.cpf]),

follow=True

)

# Verificar se o redirecionamento foi bem-sucedido

self.assertEqual(response.status\_code, 200)

# Verificar se o aluno foi excluído do banco de dados

self.assertFalse(Aluno.objects.filter(cpf=self.aluno1.cpf).exists())

# Verificar se a mensagem de sucesso está na página

self.assertContains(response, "Aluno excluído com sucesso")

Copy

test\_views.py

**Passo 6: Testes de Integração para o Módulo de Atividades**

from django.test import TestCase, Client

from django.urls import reverse

from django.contrib.auth.models import User

from atividades.models import AtividadeAcademica, AtividadeRitualistica

from turmas.models import Turma

from django.utils import timezone

class AtividadesAcademicasViewsTestCase(TestCase):

"""Testes de integração para as views de atividades acadêmicas."""

def setUp(self):

"""Configuração inicial para os testes."""

# Criar um usuário para autenticação

self.user = User.objects.create\_user(

username='testuser',

password='testpassword'

)

# Criar uma turma para associar às atividades

self.turma = Turma.objects.create(

nome="Turma A",

codigo="TA-001",

status="A"

)

# Criar algumas atividades para testes

self.atividade = AtividadeAcademica.objects.create(

nome="Aula de Filosofia",

descricao="Introdução à Filosofia",

data\_inicio=timezone.now(),

responsavel="Prof. Silva",

tipo\_atividade="aula",

status="agendada"

)

self.atividade.turmas.add(self.turma)

# Cliente para fazer requisições

self.client = Client()

def test\_listar\_atividades\_academicas\_view(self):

"""Testa se a view de listagem de atividades acadêmicas funciona corretamente."""

# Fazer login

self.client.login(username='testuser', password='testpassword')

# Acessar a página de listagem de atividades acadêmicas

response = self.client.get(reverse('atividades:listar\_atividades\_academicas'))

# Verificar se a resposta foi bem-sucedida

self.assertEqual(response.status\_code, 200)

# Verificar se as atividades estão no contexto

self.assertIn('atividades', response.context)

self.assertEqual(len(response.context['atividades']), 1)

# Verificar se o nome da atividade está na página

self.assertContains(response, "Aula de Filosofia")

def test\_detalhar\_atividade\_academica\_view(self):

"""Testa se a view de detalhes da atividade acadêmica funciona corretamente."""

# Fazer login

self.client.login(username='testuser', password='testpassword')

# Acessar a página de detalhes da atividade acadêmica

response = self.client.get(

reverse('atividades:detalhar\_atividade\_academica', args=[self.atividade.pk])

)

# Verificar se a resposta foi bem-sucedida

self.assertEqual(response.status\_code, 200)

# Verificar se a atividade está no contexto

self.assertIn('atividade', response.context)

self.assertEqual(response.context['atividade'].pk, self.atividade.pk)

# Verificar se os dados da atividade estão na página

self.assertContains(response, self.atividade.nome)

self.assertContains(response, self.atividade.descricao)

def test\_criar\_atividade\_academica\_view(self):

"""Testa se a view de criação de atividade acadêmica funciona corretamente."""

# Fazer login

self.client.login(username='testuser', password='testpassword')

# Acessar a página de criação de atividade acadêmica

response = self.client.get(reverse('atividades:criar\_atividade\_academica'))

# Verificar se a resposta foi bem-sucedida

self.assertEqual(response.status\_code, 200)

# Verificar se o formulário está no contexto

self.assertIn('form', response.context)

# Dados para a nova atividade

nova\_atividade\_data = {

'nome': 'Nova Aula de História',

'descricao': 'História do Brasil',

'data\_inicio': timezone.now().strftime('%Y-%m-%d'),

'responsavel': 'Prof. Santos',

'local': 'Sala 102',

'tipo\_atividade': 'aula',

'status': 'agendada',

'turmas': [self.turma.id],

}

# Enviar o formulário para criar uma nova atividade

response = self.client.post(

reverse('atividades:criar\_atividade\_academica'),

data=nova\_atividade\_data,

follow=True

)

# Verificar se o redirecionamento foi bem-sucedido

self.assertEqual(response.status\_code, 200)

# Verificar se a atividade foi criada no banco de dados

self.assertTrue(AtividadeAcademica.objects.filter(nome='Nova Aula de História').exists())

# Verificar se a mensagem de sucesso está na página

self.assertContains(response, "Atividade acadêmica criada com sucesso")

class AtividadesRitualisticasViewsTestCase(TestCase):

"""Testes de integração para as views de atividades ritualísticas."""

def setUp(self):

"""Configuração inicial para os testes."""

# Criar um usuário para autenticação

self.user = User.objects.create\_user(

username='testuser',

password='testpassword'

)

# Criar uma turma para associar às atividades

self.turma = Turma.objects.create(

nome="Turma A",

codigo="TA-001",

status="A"

)

# Criar algumas atividades para testes

self.atividade = AtividadeRitualistica.objects.create(

nome="Ritual de Iniciação",

descricao="Ritual para novos membros",

data=timezone.now().date(),

hora\_inicio="19:00",

hora\_fim="21:00",

local="Templo Principal",

turma=self.turma

)

# Cliente para fazer requisições

self.client = Client()

def test\_listar\_atividades\_ritualisticas\_view(self):

"""Testa se a view de listagem de atividades ritualísticas funciona corretamente."""

# Fazer login

self.client.login(username='testuser', password='testpassword')

# Acessar a página de listagem de atividades ritualísticas

response = self.client.get(reverse('atividades:listar\_atividades\_ritualisticas'))

# Verificar se a resposta foi bem-sucedida

self.assertEqual(response.status\_code, 200)

# Verificar se as atividades estão no contexto

self.assertIn('atividades', response.context)

self.assertEqual(len(response.context['atividades']), 1)

# Verificar se o nome da atividade está na página

self.assertContains(response, "Ritual de Iniciação")

Copy

test\_views.py

**Passo 7: Testes de Integração para o Módulo de Turmas**

from django.test import TestCase, Client

from django.urls import reverse

from django.contrib.auth.models import User

from turmas.models import Turma

from alunos.models import Aluno

from matriculas.models import Matricula

from django.utils import timezone

class TurmasViewsTestCase(TestCase):

"""Testes de integração para as views do módulo de turmas."""

def setUp(self):

"""Configuração inicial para os testes."""

# Criar um usuário para autenticação

self.user = User.objects.create\_user(

username='testuser',

password='testpassword'

)

# Criar algumas turmas para testes

self.turma1 = Turma.objects.create(

nome="Turma de Filosofia 2023",

codigo="FIL-2023",

data\_inicio=timezone.now().date(),

status="A"

)

self.turma2 = Turma.objects.create(

nome="Turma de História 2023",

codigo="HIS-2023",

data\_inicio=timezone.now().date(),

status="A"

)

# Criar alguns alunos para testes

self.aluno1 = Aluno.objects.create(

cpf="12345678900",

nome="João da Silva",

email="joao@exemplo.com",

data\_nascimento="1990-01-01"

)

self.aluno2 = Aluno.objects.create(

cpf="98765432100",

nome="Maria Souza",

email="maria@exemplo.com",

data\_nascimento="1992-05-15"

)

# Cliente para fazer requisições

self.client = Client()

def test\_listar\_turmas\_view(self):

"""Testa se a view de listagem de turmas funciona corretamente."""

# Fazer login

self.client.login(username='testuser', password='testpassword')

# Acessar a página de listagem de turmas

response = self.client.get(reverse('turmas:listar\_turmas'))

# Verificar se a resposta foi bem-sucedida

self.assertEqual(response.status\_code, 200)

# Verificar se as turmas estão no contexto

self.assertIn('turmas', response.context)

self.assertEqual(len(response.context['turmas']), 2)

# Verificar se os nomes das turmas estão na página

self.assertContains(response, "Turma de Filosofia 2023")

self.assertContains(response, "Turma de História 2023")

def test\_detalhar\_turma\_view(self):

"""Testa se a view de detalhes da turma funciona corretamente."""

# Fazer login

self.client.login(username='testuser', password='testpassword')

# Acessar a página de detalhes da turma

response = self.client.get(

reverse('turmas:detalhar\_turma', args=[self.turma1.pk])

)

# Verificar se a resposta foi bem-sucedida

self.assertEqual(response.status\_code, 200)

# Verificar se a turma está no contexto

self.assertIn('turma', response.context)

self.assertEqual(response.context['turma'].pk, self.turma1.pk)

# Verificar se os dados da turma estão na página

self.assertContains(response, self.turma1.nome)

self.assertContains(response, self.turma1.codigo)

def test\_criar\_turma\_view(self):

"""Testa se a view de criação de

Copy

test\_views.py

continue

PromptsClaude 3.7 Sonnet

**Context— none fetched**

**Plano de Testes para o Projeto OMAUM (Continuação)**

def test\_criar\_turma\_view(self):

"""Testa se a view de criação de turma funciona corretamente."""

# Fazer login

self.client.login(username='testuser', password='testpassword')

# Acessar a página de criação de turma

response = self.client.get(reverse('turmas:criar\_turma'))

# Verificar se a resposta foi bem-sucedida

self.assertEqual(response.status\_code, 200)

# Verificar se o formulário está no contexto

self.assertIn('form', response.context)

# Dados para a nova turma

nova\_turma\_data = {

'nome': 'Turma de Geografia 2023',

'codigo': 'GEO-2023',

'data\_inicio': timezone.now().date().strftime('%Y-%m-%d'),

'status': 'A',

}

# Enviar o formulário para criar uma nova turma

response = self.client.post(

reverse('turmas:criar\_turma'),

data=nova\_turma\_data,

follow=True

)

# Verificar se o redirecionamento foi bem-sucedido

self.assertEqual(response.status\_code, 200)

# Verificar se a turma foi criada no banco de dados

self.assertTrue(Turma.objects.filter(codigo='GEO-2023').exists())

# Verificar se a mensagem de sucesso está na página

self.assertContains(response, "Turma criada com sucesso")

def test\_matricular\_aluno\_view(self):

"""Testa se a view de matrícula de aluno funciona corretamente."""

# Fazer login

self.client.login(username='testuser', password='testpassword')

# Acessar a página de matrícula de aluno

response = self.client.get(

reverse('turmas:matricular\_aluno', args=[self.turma1.pk])

)

# Verificar se a resposta foi bem-sucedida

self.assertEqual(response.status\_code, 200)

# Verificar se o formulário está no contexto

self.assertIn('form', response.context)

# Dados para a matrícula

matricula\_data = {

'aluno': self.aluno1.cpf,

'data\_matricula': timezone.now().date().strftime('%Y-%m-%d'),

'status': 'A',

}

# Enviar o formulário para matricular o aluno

response = self.client.post(

reverse('turmas:matricular\_aluno', args=[self.turma1.pk]),

data=matricula\_data,

follow=True

)

# Verificar se o redirecionamento foi bem-sucedido

self.assertEqual(response.status\_code, 200)

# Verificar se a matrícula foi criada no banco de dados

self.assertTrue(Matricula.objects.filter(aluno=self.aluno1, turma=self.turma1).exists())

# Verificar se a mensagem de sucesso está na página

self.assertContains(response, "Aluno matriculado com sucesso")

Copy

test\_views.py

**Testes de Ponta a Ponta**

**Passo 8: Configuração dos Testes de Ponta a Ponta**

import pytest

from selenium import webdriver

from selenium.webdriver.chrome.options import Options

from django.contrib.staticfiles.testing import StaticLiveServerTestCase

from django.contrib.auth.models import User

from alunos.models import Aluno

from turmas.models import Turma

from atividades.models import AtividadeAcademica, AtividadeRitualistica

from django.utils import timezone

@pytest.fixture(scope='session')

def browser():

"""Configuração do navegador para testes E2E."""

chrome\_options = Options()

chrome\_options.add\_argument('--headless') # Executar sem interface gráfica

chrome\_options.add\_argument('--no-sandbox')

chrome\_options.add\_argument('--disable-dev-shm-usage')

browser = webdriver.Chrome(options=chrome\_options)

browser.implicitly\_wait(10)

yield browser

browser.quit()

@pytest.fixture

def live\_server\_with\_data(live\_server):

"""Configura o servidor de teste com dados iniciais."""

# Criar um usuário para autenticação

user = User.objects.create\_user(

username='testuser',

password='testpassword',

email='test@example.com'

)

# Criar alguns alunos

aluno1 = Aluno.objects.create(

cpf="12345678900",

nome="João da Silva",

email="joao@exemplo.com",

data\_nascimento="1990-01-01",

sexo="M",

situacao="ativo"

)

aluno2 = Aluno.objects.create(

cpf="98765432100",

nome="Maria Souza",

email="maria@exemplo.com",

data\_nascimento="1992-05-15",

sexo="F",

situacao="ativo"

)

# Criar algumas turmas

turma1 = Turma.objects.create(

nome="Turma de Filosofia 2023",

codigo="FIL-2023",

data\_inicio=timezone.now().date(),

status="A"

)

turma2 = Turma.objects.create(

nome="Turma de História 2023",

codigo="HIS-2023",

data\_inicio=timezone.now().date(),

status="A"

)

# Criar algumas atividades

atividade1 = AtividadeAcademica.objects.create(

nome="Aula de Filosofia",

descricao="Introdução à Filosofia",

data\_inicio=timezone.now(),

responsavel="Prof. Silva",

tipo\_atividade="aula",

status="agendada"

)

atividade1.turmas.add(turma1)

atividade2 = AtividadeRitualistica.objects.create(

nome="Ritual de Iniciação",

descricao="Ritual para novos membros",

data=timezone.now().date(),

hora\_inicio="19:00",

hora\_fim="21:00",

local="Templo Principal",

turma=turma1

)

return live\_server

Copy

conftest.py

**Passo 9: Testes de Ponta a Ponta para o Fluxo de Alunos**

import pytest

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from django.urls import reverse

@pytest.mark.django\_db

class TestAlunosE2E:

"""Testes de ponta a ponta para o fluxo de alunos."""

def test\_login\_e\_listar\_alunos(self, browser, live\_server\_with\_data):

"""Testa o login e a listagem de alunos."""

# Acessar a página de login

browser.get(f"{live\_server\_with\_data.url}/login/")

# Preencher o formulário de login

browser.find\_element(By.NAME, "username").send\_keys("testuser")

browser.find\_element(By.NAME, "password").send\_keys("testpassword")

browser.find\_element(By.CSS\_SELECTOR, "button[type='submit']").click()

# Verificar se o login foi bem-sucedido

WebDriverWait(browser, 10).until(

EC.presence\_of\_element\_located((By.CSS\_SELECTOR, ".navbar-brand"))

)

# Acessar a página de listagem de alunos

browser.get(f"{live\_server\_with\_data.url}{reverse('alunos:listar\_alunos')}")

# Verificar se a página carregou corretamente

WebDriverWait(browser, 10).until(

EC.presence\_of\_element\_located((By.TAG\_NAME, "h1"))

)

# Verificar se os alunos estão na página

assert "João da Silva" in browser.page\_source

assert "Maria Souza" in browser.page\_source

def test\_criar\_aluno(self, browser, live\_server\_with\_data):

"""Testa a criação de um novo aluno."""

# Fazer login

browser.get(f"{live\_server\_with\_data.url}/login/")

browser.find\_element(By.NAME, "username").send\_keys("testuser")

browser.find\_element(By.NAME, "password").send\_keys("testpassword")

browser.find\_element(By.CSS\_SELECTOR, "button[type='submit']").click()

# Acessar a página de criação de aluno

browser.get(f"{live\_server\_with\_data.url}{reverse('alunos:criar\_aluno')}")

# Verificar se a página carregou corretamente

WebDriverWait(browser, 10).until(

EC.presence\_of\_element\_located((By.ID, "id\_cpf"))

)

# Preencher o formulário

browser.find\_element(By.ID, "id\_cpf").send\_keys("11122233344")

browser.find\_element(By.ID, "id\_nome").send\_keys("Carlos Pereira")

browser.find\_element(By.ID, "id\_email").send\_keys("carlos@exemplo.com")

browser.find\_element(By.ID, "id\_data\_nascimento").send\_keys("1988-03-15")

# Selecionar sexo

browser.find\_element(By.CSS\_SELECTOR, "select#id\_sexo option[value='M']").click()

# Enviar o formulário

browser.find\_element(By.CSS\_SELECTOR, "button[type='submit']").click()

# Verificar se o aluno foi criado com sucesso

WebDriverWait(browser, 10).until(

EC.presence\_of\_element\_located((By.CSS\_SELECTOR, ".alert-success"))

)

# Verificar se o aluno aparece na listagem

browser.get(f"{live\_server\_with\_data.url}{reverse('alunos:listar\_alunos')}")

assert "Carlos Pereira" in browser.page\_source

Copy

test\_alunos\_e2e.py

**Passo 10: Testes de Ponta a Ponta para o Fluxo de Atividades**

import pytest

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait, Select

from selenium.webdriver.support import expected\_conditions as EC

from django.urls import reverse

@pytest.mark.django\_db

class TestAtividadesE2E:

"""Testes de ponta a ponta para o fluxo de atividades."""

def test\_criar\_atividade\_academica(self, browser, live\_server\_with\_data):

"""Testa a criação de uma atividade acadêmica."""

# Fazer login

browser.get(f"{live\_server\_with\_data.url}/login/")

browser.find\_element(By.NAME, "username").send\_keys("testuser")

browser.find\_element(By.NAME, "password").send\_keys("testpassword")

browser.find\_element(By.CSS\_SELECTOR, "button[type='submit']").click()

# Acessar a página de criação de atividade acadêmica

browser.get(f"{live\_server\_with\_data.url}{reverse('atividades:criar\_atividade\_academica')}")

# Verificar se a página carregou corretamente

WebDriverWait(browser, 10).until(

EC.presence\_of\_element\_located((By.ID, "id\_nome"))

)

# Preencher o formulário

browser.find\_element(By.ID, "id\_nome").send\_keys("Aula de Matemática")

browser.find\_element(By.ID, "id\_descricao").send\_keys("Introdução à Álgebra")

browser.find\_element(By.ID, "id\_data\_inicio").send\_keys("2023-12-01")

browser.find\_element(By.ID, "id\_responsavel").send\_keys("Prof. Oliveira")

browser.find\_element(By.ID, "id\_local").send\_keys("Sala 103")

# Selecionar tipo de atividade

Select(browser.find\_element(By.ID, "id\_tipo\_atividade")).select\_by\_value("aula")

# Selecionar status

Select(browser.find\_element(By.ID, "id\_status")).select\_by\_value("agendada")

# Selecionar turma (primeira opção)

turmas\_select = Select(browser.find\_element(By.ID, "id\_turmas"))

turmas\_select.select\_by\_index(0)

# Enviar o formulário

browser.find\_element(By.CSS\_SELECTOR, "button[type='submit']").click()

# Verificar se a atividade foi criada com sucesso

WebDriverWait(browser, 10).until(

EC.presence\_of\_element\_located((By.CSS\_SELECTOR, ".alert-success"))

)

# Verificar se a atividade aparece na listagem

browser.get(f"{live\_server\_with\_data.url}{reverse('atividades:listar\_atividades\_academicas')}")

assert "Aula de Matemática" in browser.page\_source

def test\_calendario\_atividades(self, browser, live\_server\_with\_data):

"""Testa a visualização do calendário de atividades."""

# Fazer login

browser.get(f"{live\_server\_with\_data.url}/login/")

browser.find\_element(By.NAME, "username").send\_keys("testuser")

browser.find\_element(By.NAME, "password").send\_keys("testpassword")

browser.find\_element(By.CSS\_SELECTOR, "button[type='submit']").click()

# Acessar a página do calendário de atividades

browser.get(f"{live\_server\_with\_data.url}{reverse('atividades:calendario\_atividades')}")

# Verificar se a página carregou corretamente

WebDriverWait(browser, 10).until(

EC.presence\_of\_element\_located((By.ID, "calendar"))

)

# Verificar se o calendário foi renderizado

Copy

test\_atividades\_e2e.py

Prossiga até o final

PromptsClaude 3.7 Sonnet

**Context— none fetched**

**Plano de Testes para o Projeto OMAUM (Continuação)**

def test\_calendario\_atividades(self, browser, live\_server\_with\_data):

"""Testa a visualização do calendário de atividades."""

# Fazer login

browser.get(f"{live\_server\_with\_data.url}/login/")

browser.find\_element(By.NAME, "username").send\_keys("testuser")

browser.find\_element(By.NAME, "password").send\_keys("testpassword")

browser.find\_element(By.CSS\_SELECTOR, "button[type='submit']").click()

# Acessar a página do calendário de atividades

browser.get(f"{live\_server\_with\_data.url}{reverse('atividades:calendario\_atividades')}")

# Verificar se a página carregou corretamente

WebDriverWait(browser, 10).until(

EC.presence\_of\_element\_located((By.ID, "calendar"))

)

# Verificar se o calendário foi renderizado

assert "fc-view-harness" in browser.page\_source

# Verificar se há eventos no calendário (pode variar dependendo do mês atual)

# Aguardar o carregamento dos eventos via AJAX

WebDriverWait(browser, 10).until(

EC.presence\_of\_element\_located((By.CSS\_SELECTOR, ".fc-daygrid-day-events"))

)

Copy

test\_atividades\_e2e.py

**Passo 11: Testes de Ponta a Ponta para o Fluxo de Turmas**

import pytest

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait, Select

from selenium.webdriver.support import expected\_conditions as EC

from django.urls import reverse

@pytest.mark.django\_db

class TestTurmasE2E:

"""Testes de ponta a ponta para o fluxo de turmas."""

def test\_listar\_e\_detalhar\_turma(self, browser, live\_server\_with\_data):

"""Testa a listagem e visualização de detalhes de uma turma."""

# Fazer login

browser.get(f"{live\_server\_with\_data.url}/login/")

browser.find\_element(By.NAME, "username").send\_keys("testuser")

browser.find\_element(By.NAME, "password").send\_keys("testpassword")

browser.find\_element(By.CSS\_SELECTOR, "button[type='submit']").click()

# Acessar a página de listagem de turmas

browser.get(f"{live\_server\_with\_data.url}{reverse('turmas:listar\_turmas')}")

# Verificar se a página carregou corretamente

WebDriverWait(browser, 10).until(

EC.presence\_of\_element\_located((By.TAG\_NAME, "h1"))

)

# Verificar se as turmas estão na página

assert "Turma de Filosofia 2023" in browser.page\_source

assert "Turma de História 2023" in browser.page\_source

# Clicar no botão de detalhes da primeira turma

browser.find\_element(By.LINK\_TEXT, "Detalhes").click()

# Verificar se a página de detalhes carregou corretamente

WebDriverWait(browser, 10).until(

EC.presence\_of\_element\_located((By.CSS\_SELECTOR, ".card-header"))

)

# Verificar se os detalhes da turma estão na página

assert "Informações da Turma" in browser.page\_source

def test\_criar\_turma(self, browser, live\_server\_with\_data):

"""Testa a criação de uma nova turma."""

# Fazer login

browser.get(f"{live\_server\_with\_data.url}/login/")

browser.find\_element(By.NAME, "username").send\_keys("testuser")

browser.find\_element(By.NAME, "password").send\_keys("testpassword")

browser.find\_element(By.CSS\_SELECTOR, "button[type='submit']").click()

# Acessar a página de criação de turma

browser.get(f"{live\_server\_with\_data.url}{reverse('turmas:criar\_turma')}")

# Verificar se a página carregou corretamente

WebDriverWait(browser, 10).until(

EC.presence\_of\_element\_located((By.ID, "id\_nome"))

)

# Preencher o formulário

browser.find\_element(By.ID, "id\_nome").send\_keys("Turma de Geografia 2023")

browser.find\_element(By.ID, "id\_codigo").send\_keys("GEO-2023")

browser.find\_element(By.ID, "id\_data\_inicio").send\_keys("2023-01-15")

# Selecionar status

Select(browser.find\_element(By.ID, "id\_status")).select\_by\_value("A")

# Enviar o formulário

browser.find\_element(By.CSS\_SELECTOR, "button[type='submit']").click()

# Verificar se a turma foi criada com sucesso

WebDriverWait(browser, 10).until(

EC.presence\_of\_element\_located((By.CSS\_SELECTOR, ".alert-success"))

)

# Verificar se a turma aparece na listagem

browser.get(f"{live\_server\_with\_data.url}{reverse('turmas:listar\_turmas')}")

assert "Turma de Geografia 2023" in browser.page\_source

Copy

test\_turmas\_e2e.py

**Proposta de Dados de Teste**

**Passo 12: Configuração de Factories para Dados de Teste**

import factory

from factory.django import DjangoModelFactory

from django.utils import timezone

from alunos.models import Aluno

from turmas.models import Turma

from atividades.models import AtividadeAcademica, AtividadeRitualistica

from django.contrib.auth.models import User

import random

from datetime import timedelta

class UserFactory(DjangoModelFactory):

"""Factory para criar usuários."""

class Meta:

model = User

username = factory.Sequence(lambda n: f'user{n}')

email = factory.LazyAttribute(lambda obj: f'{obj.username}@example.com')

password = factory.PostGenerationMethodCall('set\_password', 'password')

is\_active = True

class AlunoFactory(DjangoModelFactory):

"""Factory para criar alunos."""

class Meta:

model = Aluno

cpf = factory.Sequence(lambda n: f'{n:011d}')

nome = factory.Faker('name')

email = factory.Faker('email')

data\_nascimento = factory.Faker('date\_of\_birth', minimum\_age=18, maximum\_age=70)

sexo = factory.Iterator(['M', 'F'])

situacao = 'ativo'

@factory.post\_generation

def add\_details(self, create, extracted, \*\*kwargs):

if not create:

return

# Adicionar detalhes aleatórios

self.nacionalidade = random.choice(['Brasileira', 'Portuguesa', 'Italiana', 'Espanhola'])

self.naturalidade = factory.Faker('city').generate({})

self.rua = factory.Faker('street\_name').generate({})

self.numero\_imovel = str(random.randint(1, 999))

self.bairro = factory.Faker('city\_suffix').generate({})

self.cidade = factory.Faker('city').generate({})

self.estado = factory.Faker('state\_abbr').generate({})

self.cep = f'{random.randint(10000, 99999)}-{random.randint(100, 999)}'

# Salvar as alterações

self.save()

class TurmaFactory(DjangoModelFactory):

"""Factory para criar turmas."""

class Meta:

model = Turma

nome = factory.Sequence(lambda n: f'Turma {n}')

codigo = factory.Sequence(lambda n: f'T{n:03d}')

data\_inicio = factory.LazyFunction(timezone.now)

status = 'A'

@factory.post\_generation

def add\_details(self, create, extracted, \*\*kwargs):

if not create:

return

# Adicionar data de fim aleatória (entre 6 meses e 1 ano após o início)

dias\_aleatorios = random.randint(180, 365)

self.data\_fim = self.data\_inicio + timedelta(days=dias\_aleatorios)

# Salvar as alterações

self.save()

class AtividadeAcademicaFactory(DjangoModelFactory):

"""Factory para criar atividades acadêmicas."""

class Meta:

model = AtividadeAcademica

nome = factory.Sequence(lambda n: f'Atividade Acadêmica {n}')

descricao = factory.Faker('paragraph')

data\_inicio = factory.LazyFunction(lambda: timezone.now() + timedelta(days=random.randint(1, 30)))

responsavel = factory.Faker('name')

local = factory.Sequence(lambda n: f'Sala {n}')

tipo\_atividade = factory.Iterator(['aula', 'palestra', 'workshop', 'seminario', 'outro'])

status = factory.Iterator(['agendada', 'em\_andamento', 'concluida', 'cancelada'])

@factory.post\_generation

def turmas(self, create, extracted, \*\*kwargs):

if not create:

return

if extracted:

# Adicionar turmas específicas

for turma in extracted:

self.turmas.add(turma)

else:

# Adicionar turmas aleatórias

turmas\_count = random.randint(1, 3)

turmas = TurmaFactory.create\_batch(turmas\_count)

for turma in turmas:

self.turmas.add(turma)

@factory.post\_generation

def add\_details(self, create, extracted, \*\*kwargs):

if not create:

return

# Adicionar data de fim aleatória (entre 1 e 5 dias após o início)

dias\_aleatorios = random.randint(1, 5)

self.data\_fim = self.data\_inicio + timedelta(days=dias\_aleatorios)

# Salvar as alterações

self.save()

class AtividadeRitualisticaFactory(DjangoModelFactory):

"""Factory para criar atividades ritualísticas."""

class Meta:

model = AtividadeRitualistica

nome = factory.Sequence(lambda n: f'Ritual {n}')

descricao = factory.Faker('paragraph')

data = factory.LazyFunction(lambda: timezone.now().date() + timedelta(days=random.randint(1, 30)))

hora\_inicio = factory.LazyFunction(lambda: f'{random.randint(18, 20)}:00')

hora\_fim = factory.LazyFunction(lambda: f'{random.randint(21, 23)}:00')

local = factory.Iterator(['Templo Principal', 'Sala de Rituais', 'Salão Nobre'])

@factory.post\_generation

def turma(self, create, extracted, \*\*kwargs):

if not create:

return

if extracted:

# Usar a turma especificada

self.turma = extracted

else:

# Criar uma nova turma

self.turma = TurmaFactory.create()

# Salvar as alterações

self.save()

@factory.post\_generation

def participantes(self, create, extracted, \*\*kwargs):

if not create:

return

if extracted:

# Adicionar participantes específicos

for aluno in extracted:

self.participantes.add(aluno)

else:

# Adicionar participantes aleatórios

alunos\_count = random.randint(5, 15)

alunos = AlunoFactory.create\_batch(alunos\_count)

for aluno in alunos:

self.participantes.add(aluno)

Copy

factories.py

**Passo 13: Script para Geração de Dados de Teste**

import os

import django

import random

from datetime import timedelta

# Configurar o ambiente Django

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'omaum.settings')

django.setup()

from tests.factories import (

UserFactory, AlunoFactory, TurmaFactory,

AtividadeAcademicaFactory, AtividadeRitualisticaFactory

)

from django.utils import timezone

from matriculas.models import Matricula

def generate\_test\_data():

"""Gera dados de teste para o sistema."""

print("Gerando dados de teste...")

# Criar usuários

print("Criando usuários...")

admin = UserFactory(username='admin', is\_staff=True, is\_superuser=True)

users = UserFactory.create\_batch(5)

# Criar alunos

print("Criando alunos...")

alunos = AlunoFactory.create\_batch(50)

# Criar turmas

print("Criando turmas...")

turmas = TurmaFactory.create\_batch(10)

# Criar matrículas

print("Criando matrículas...")

for turma in turmas:

# Selecionar alunos aleatórios para esta turma

alunos\_selecionados = random.sample(alunos, random.randint(5, 20))

for aluno in alunos\_selecionados:

# Criar matrícula

data\_matricula = turma.data\_inicio - timedelta(days=random.randint(1, 30))

Matricula.objects.create(

aluno=aluno,

turma=turma,

data\_matricula=data\_matricula,

status='A'

)

# Criar atividades acadêmicas

print("Criando atividades acadêmicas...")

for \_ in range(30):

# Selecionar turmas aleatórias

turmas\_selecionadas = random.sample(turmas, random.randint(1, 3))

AtividadeAcademicaFactory(turmas=turmas\_selecionadas)

# Criar atividades ritualísticas

print("Criando atividades ritualísticas...")

for turma in turmas:

# Selecionar alunos matriculados nesta turma

alunos\_matriculados = [m.aluno for m in Matricula.objects.filter(turma

Copy

generate\_test\_data.py

Prossiga até o final

PromptsClaude 3.7 Sonnet

**Context— none fetched**

**Plano de Testes para o Projeto OMAUM (Continuação)**

def generate\_test\_data():

"""Gera dados de teste para o sistema."""

print("Gerando dados de teste...")

# Criar usuários

print("Criando usuários...")

admin = UserFactory(username='admin', is\_staff=True, is\_superuser=True)

users = UserFactory.create\_batch(5)

# Criar alunos

print("Criando alunos...")

alunos = AlunoFactory.create\_batch(50)

# Criar turmas

print("Criando turmas...")

turmas = TurmaFactory.create\_batch(10)

# Criar matrículas

print("Criando matrículas...")

for turma in turmas:

# Selecionar alunos aleatórios para esta turma

alunos\_selecionados = random.sample(alunos, random.randint(5, 20))

for aluno in alunos\_selecionados:

# Criar matrícula

data\_matricula = turma.data\_inicio - timedelta(days=random.randint(1, 30))

Matricula.objects.create(

aluno=aluno,

turma=turma,

data\_matricula=data\_matricula,

status='A'

)

# Criar atividades acadêmicas

print("Criando atividades acadêmicas...")

for \_ in range(30):

# Selecionar turmas aleatórias

turmas\_selecionadas = random.sample(turmas, random.randint(1, 3))

AtividadeAcademicaFactory(turmas=turmas\_selecionadas)

# Criar atividades ritualísticas

print("Criando atividades ritualísticas...")

for turma in turmas:

# Selecionar alunos matriculados nesta turma

alunos\_matriculados = [m.aluno for m in Matricula.objects.filter(turma=turma)]

if alunos\_matriculados:

# Criar entre 1 e 3 atividades ritualísticas para esta turma

for \_ in range(random.randint(1, 3)):

# Selecionar participantes aleatórios entre os alunos matriculados

participantes = random.sample(

alunos\_matriculados,

min(len(alunos\_matriculados), random.randint(3, len(alunos\_matriculados)))

)

AtividadeRitualisticaFactory(turma=turma, participantes=participantes)

print("Dados de teste gerados com sucesso!")

print(f"- {len(users) + 1} usuários (incluindo admin)")

print(f"- {len(alunos)} alunos")

print(f"- {len(turmas)} turmas")

print(f"- {Matricula.objects.count()} matrículas")

print(f"- {AtividadeAcademicaFactory.\_meta.model.objects.count()} atividades acadêmicas")

print(f"- {AtividadeRitualisticaFactory.\_meta.model.objects.count()} atividades ritualísticas")

if \_\_name\_\_ == "\_\_main\_\_":

generate\_test\_data()

Copy

generate\_test\_data.py

**Documentação do Plano de Testes**

**Estrutura do Plano de Testes**

# Plano de Testes - Projeto OMAUM

## Introdução

Este documento descreve o plano de testes para o sistema OMAUM, um sistema de gestão para organizações místicas. O plano abrange testes unitários, de integração e de ponta a ponta (E2E) para garantir a qualidade e o funcionamento correto do sistema.

## Objetivos

- Verificar o funcionamento correto de todos os componentes do sistema

- Garantir que os requisitos funcionais sejam atendidos

- Identificar e corrigir bugs antes da implantação

- Assegurar a qualidade do código e a manutenibilidade do sistema

## Escopo

O plano de testes abrange os seguintes módulos do sistema:

1. **\*\*Alunos\*\***: Cadastro, edição, exclusão e listagem de alunos

2. **\*\*Turmas\*\***: Gerenciamento de turmas e matrículas

3. **\*\*Atividades\*\***: Gerenciamento de atividades acadêmicas e ritualísticas

## Tipos de Testes

### Testes Unitários

Os testes unitários verificam o funcionamento correto de componentes individuais do sistema, como modelos, formulários e funções utilitárias.

### Testes de Integração

Os testes de integração verificam a interação entre diferentes componentes do sistema, como views, modelos e templates.

### Testes de Ponta a Ponta (E2E)

Os testes E2E simulam a interação do usuário com o sistema através de um navegador web, verificando fluxos completos de uso.

## Ferramentas

- **\*\*pytest\*\***: Framework de testes para Python

- **\*\*pytest-django\*\***: Plugin para integração do pytest com Django

- **\*\*Selenium\*\***: Ferramenta para automação de navegadores web (testes E2E)

- **\*\*Factory Boy\*\***: Biblioteca para criação de objetos de teste

## Estrutura dos Testes

```

tests/

├── alunos/

│ ├── test\_models.py

│ ├── test\_forms.py

│ ├── test\_views.py

│ └── test\_utils.py

├── turmas/

│ ├── test\_models.py

│ ├── test\_forms.py

│ └── test\_views.py

├── atividades/

│ ├── test\_models.py

│ ├── test\_forms.py

│ └── test\_views.py

├── e2e/

│ ├── conftest.py

│ ├── test\_alunos\_e2e.py

│ ├── test\_turmas\_e2e.py

│ └── test\_atividades\_e2e.py

├── factories.py

├── conftest.py

└── generate\_test\_data.py

```

## Execução dos Testes

### Pré-requisitos

- Python 3.8+

- Django 3.2+

- pytest

- pytest-django

- Selenium

- Factory Boy

- WebDriver para Chrome ou Firefox

### Comandos

```bash

# Instalar dependências

pip install -r requirements-dev.txt

# Executar todos os testes

pytest

# Executar testes unitários

pytest tests/alunos/test\_models.py tests/alunos/test\_forms.py

# Executar testes de integração

pytest tests/alunos/test\_views.py

# Executar testes E2E

pytest tests/e2e/

# Gerar dados de teste

python tests/generate\_test\_data.py

```

## Dados de Teste

O script `generate\_test\_data.py` cria dados de teste para o sistema, incluindo:

- Usuários

- Alunos

- Turmas

- Matrículas

- Atividades acadêmicas

- Atividades ritualísticas

Estes dados são úteis para testes manuais e para popular o ambiente de desenvolvimento.

## Cobertura de Testes

O objetivo é atingir pelo menos 80% de cobertura de código para os módulos principais do sistema. A cobertura pode ser verificada com o comando:

```bash

pytest --cov=alunos --cov=turmas --cov=atividades

```

## Manutenção dos Testes

Os testes devem ser atualizados sempre que:

- Novos recursos são adicionados ao sistema

- Bugs são corrigidos

- Requisitos são alterados

- Comportamentos existentes são modificados

## Responsabilidades

- **\*\*Desenvolvedores\*\***: Escrever testes unitários e de integração para seu código

- **\*\*QA\*\***: Revisar os testes e escrever testes E2E

- **\*\*Tech Lead\*\***: Garantir a cobertura adequada de testes e revisar a qualidade dos testes

## Conclusão

Este plano de testes fornece uma estrutura abrangente para garantir a qualidade do sistema OMAUM. Seguindo este plano, a equipe pode identificar e corrigir problemas cedo no ciclo de desenvolvimento, resultando em um produto final mais robusto e confiável.

Copy

plano\_de\_testes.md

**Conclusão**

O plano de testes apresentado fornece uma abordagem abrangente para garantir a qualidade do sistema OMAUM. Ele inclui:

1. **Testes Unitários** para verificar o funcionamento correto de componentes individuais
2. **Testes de Integração** para verificar a interação entre diferentes componentes
3. **Testes de Ponta a Ponta** para simular a interação do usuário com o sistema
4. **Factories** para geração de dados de teste consistentes
5. **Script de Geração de Dados** para popular o ambiente de desenvolvimento e testes

Ao implementar este plano de testes, a equipe de desenvolvimento pode:

* Identificar e corrigir bugs cedo no ciclo de desenvolvimento
* Garantir que novos recursos não quebrem funcionalidades existentes
* Facilitar refatorações e melhorias no código
* Documentar o comportamento esperado do sistema
* Aumentar a confiança na qualidade do software

Recomenda-se que os testes sejam executados regularmente durante o desenvolvimento e como parte do processo de integração contínua, garantindo que problemas sejam identificados e corrigidos rapidamente.