Manual Técnico
Mobile-SPS

Arquitetura Multi-Banco e Integração

Autor: Leonardo Sousa

Atualizado em: 20/10/2025

Sumário

Manual Técnico - Roteamento Multi-Tenant (Middleware e Router)

1. Visão Geral do Roteamento Multi-Tenant

O sistema adota uma arquitetura multi-tenant baseada em múltiplos bancos PostgreSQL, onde cada cliente possui seu próprio banco identificado por um slug. Esse slug é detectado em tempo de requisição pelo middleware e usado pelo router para direcionar a leitura e escrita ao banco correto.

Fluxo resumido:

1. O Middleware intercepta a requisição HTTP e extrai o slug do path ou token JWT.

2. O Router usa o slug para resolver o banco correto via função get_db_from_slug().

3. A requisição segue para os models e queries já com o banco selecionado dinamicamente.

2. Router - LicencaDBRouter

O Router controla qual banco será usado para operações de leitura e escrita. Ele consulta o slug atual no contexto da requisição e retorna o alias do banco correspondente.

Código:

```
from core.utils import get_db_from_slug
from core.middleware import get_licenca_slug

class LicencaDBRouter:
    def db_for_read(self, model, **hints):
        slug = get_licenca_slug()
        return get_db_from_slug(slug)

    def db_for_write(self, model, **hints):
        slug = get_licenca_slug()
        return get_db_from_slug(slug)

    def allow_relation(self, obj1, obj2, **hints):
        """Permite relações entre objetos se estão no mesmo banco"""
        return True

    def allow_migrate(self, db, app_label, model_name=None, **hints):
        """Permite migrações em qualquer banco"""
        return True
```

• get_licenca_slug(): busca o slug ativo do contexto local de thread.

• get_db_from_slug(): retorna o alias de banco correspondente.

• db_for_read/db_for_write: controlam o roteamento de leitura e gravação.

• allow_relation/allow_migrate: permitem relações e migrações globais.

3. Middleware - LicencaMiddleware

O Middleware é responsável por interceptar todas as requisições e identificar o slug da licença ativa. Ele também gerencia cache de módulos disponíveis e valida tokens JWT quando necessário.

Código simplificado:

```
class LicencaMiddleware:
    def __call__(self, request):
        # Ignora rotas estáticas e administrativas
        ignored_paths = ['/api/warm-cache/', '/admin/', '/static/',
'/media/']

        for path in ignored_paths:
            if request.path.startswith(path):
                return self.get_response(request)

        # Extrai o slug da URL (/api/<slug>/...)
        path_parts = request.path.strip('/').split('/')
        slug = path_parts[1] if len(path_parts) > 1 else None

        # Define o slug no contexto
        set_licenca_slug(slug)
        request.slug = slug

        # Define cache de módulos e log de performance
        cache_key = f"modulos_licenca_{slug}"
        modulos_disponiveis = cache.get(cache_key) or []
        request.modulos_disponiveis = modulos_disponiveis

        return self.get_response(request)
```

• Ignora rotas que não precisam de validação (admin, static, media).

• Extrai o slug da URL (ex: /api/casaa/... → slug='casaa').

• Define o slug no contexto de thread para uso no router.

• Usa cache de módulos por licença para otimizar consultas.

4. Fluxo Completo de Requisição

1. Requisição chega a /api/<slug>/endpoint/.
2. Middleware extrai o slug e define no contexto local.

3. Router consulta o slug e define o banco correto.
4. Django ORM executa a query no banco correspondente.
5. Response retorna com dados isolados por cliente.

5. Boas Práticas e Observações

• Mantenha os slugs sempre iguais ao alias de banco configurado em settings.DATABASES.

• Evite acesso direto ao banco sem usar get_db_from_slug().

• Em consultas complexas, use .using(get_db_from_slug(slug)) explicitamente.

• Sempre trate erros de slug inválido ou ausente no middleware.

6. App de Exemplo — Pisos

Estudo de caso prático do fluxo multi-tenant. Mostra como o slug seleciona o banco no middleware/router e como isso reflete nos models, serializers, services, views e urls.

6.1 Objetivo e Escopo

Gerenciar orçamentos e pedidos de pisos. Tabelas: orçamentos, itens de orçamento, pedidos e itens de pedido. Relacionamentos por campos de chave natural (ex: orca_empr/orca_fili/orca_nume ↔ item_empr/item_fili/item_orca).

6.2 Models (resumo)

• Orcamentopisos: cabeçalho de orçamento (primary key: orca_nume). • Itensorcapisos: itens de orçamento. • Pedidospisos: cabeçalho de pedido (primary key: pedi_nume). • Itenspedidospisos: itens de pedido.
Todos com managed=False e unique_together para garantir integridade por empresa/filial.

6.3 Serializers (camada de transformação JSON)

Serializers recebidos e aplicados no fluxo multi-banco.

```
from rest_framework import serializers
from rest_framework.exceptions import ValidationError
from decimal import Decimal, InvalidOperation, ROUND_HALF_UP
from core.serializers import BancoContextMixin
from .models import Orcamentopisos, Itensorcapisos, Itenspedidospisos,
Pedidospisos
from Licencas.models import Empresas
from Produtos.models import Produtos
from Entidades.models import Entidades
from .services.preco_service import get_preco_produto
from .services.utils_service import parse_decimal, arredondar
from .services.calculo_services import calcular_item,
calcular_ambientes, calcular_total_geral
import logging

logger = logging.getLogger(__name__)
```

```
# ... (código completo dos serializers do usuário, conforme enviado)
# Inclui: ItensorcapisosSerializer, OrcamentopisosSerializer,
ItenspedidospisosSerializer, PedidospisosSerializer
```

6.4 Services (regras de negócio isoladas)

Service desacopla regras de cálculo, validação e integração. Usa .using(banco) quando acessa o ORM.

```
# apps/pisos/services/pedidos_service.py
from decimal import Decimal, ROUND_HALF_UP
from django.db import transaction
from .utils_service import parse_decimal
from ..models import Pedidospisos, Itenspedidospisos

def q2(v):
    try:
        return Decimal(str(v)).quantize(Decimal('0.01'),
rounding=ROUND_HALF_UP)
    except Exception:
        return Decimal('0.00')

class PedidosService:
    @staticmethod
    @transaction.atomic
    def criar_pedido(banco, dados_cabecalho: dict, itens: list):
        # Sequência
        ultimo = Pedidospisos.objects.using(banco).filter(
            pedi_empr=dados_cabecalho["pedi_empr"],
            pedi_fili=dados_cabecalho["pedi_fili"],
        ).order_by("-pedi_nume").first()
        dados_cabecalho["pedi_nume"] = (ultimo.pedi_nume + 1) if ultimo
else 1

        pedido =
Pedidospisos.objects.using(banco).create(**dados_cabecalho)

        total = Decimal('0.00')
        for idx, it in enumerate(itens, start=1):
            quan = Decimal(str(it.get("item_quan") or 0))
            unit = Decimal(str(it.get("item_unit") or 0))
            suto = q2(quan * unit)
            Itenspedidospisos.objects.using(banco).create(
                item_empr=pedido.pedi_empr,
                item_fili=pedido.pedi_fili,
                item_pedi=pedido.pedi_nume,
                item_nume=idx,
```

```
                item_suto=suto,
                item_prod=it.get("item_prod"),
                item_quan=q2(quan),
                item_unit=q2(unit),
                item_m2=q2(it.get("item_m2") or 0),
                item_desc=q2(it.get("item_desc") or 0),
                item_nome_ambi=it.get("item_nome_ambi", ""),
            )
            total += suto

        pedido.pedi_tota = q2(total -
q2(dados_cabecalho.get("pedi_desc") or 0) +
q2(dados_cabecalho.get("pedi_fret") or 0))
        pedido.save(using=banco)
        return pedido
```

6.5 Views (acesso REST e ORM multi-banco)

ViewSets definem o contexto 'banco' a partir do request e o passam ao serializer. Operações de list/retrieve usam .using(banco). Criação/atualização delegam ao service quando necessário.

```
# apps/pisos/views.py
from rest_framework import viewsets, status
from rest_framework.response import Response
from rest_framework.decorators import action
from core.utils import get_licenca_db_config
from .models import Orcamentopisos, Pedidospisos
from .serializers import OrcamentopisosSerializer,
PedidospisosSerializer
from .services.pedidos_service import PedidosService

class BaseBancoContextViewSet(viewsets.ModelViewSet):
    def get_banco(self):
        # Usa o slug do middleware e resolve o alias/DSN
        return get_licenca_db_config(self.request)

    def get_serializer_context(self):
        ctx = super().get_serializer_context()
        ctx["banco"] = self.get_banco()
        ctx["request"] = self.request
        return ctx

class OrcamentopisosViewSet(BaseBancoContextViewSet):
    queryset = Orcamentopisos.objects.none()
    serializer_class = OrcamentopisosSerializer

    def get_queryset(self):
```

```
        banco = self.get_banco()
        qs = Orcamentopisos.objects.using(banco).all()
        # Filtros simples
        clie = self.request.query_params.get("orca_clie")
        if clie:
            qs = qs.filter(orca_clie=clie)
        return qs.order_by("-orca_data", "-orca_nume")

class PedidospisosViewSet(BaseBancoContextViewSet):
    queryset = Pedidospisos.objects.none()
    serializer_class = PedidospisosSerializer

    def get_queryset(self):
        banco = self.get_banco()
        qs = Pedidospisos.objects.using(banco).all()
        vend = self.request.query_params.get("pedi_vend")
        if vend:
            qs = qs.filter(pedi_vend=vend)
        return qs.order_by("-pedi_data", "-pedi_nume")

    def create(self, request, *args, **kwargs):
        banco = self.get_banco()
        serializer = self.get_serializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        dados_cabecalho = {k: v for k, v in
serializer.validated_data.items() if k not in ("itens_input",)}
        itens = request.data.get("itens_input") or
request.data.get("itens") or []
        pedido = PedidosService.criar_pedido(banco, dados_cabecalho,
itens)
        out = self.get_serializer(pedido)
        return Response(out.data, status=status.HTTP_201_CREATED)
```

6.6 URLs (exposição da API)

```
# apps/pisos/urls.py
from rest_framework.routers import DefaultRouter
from .views import OrcamentopisosViewSet, PedidospisosViewSet

router = DefaultRouter()
router.register(r"orcamentos-pisos", OrcamentopisosViewSet,
basename="orcamentopisos")
router.register(r"pedidos-pisos", PedidospisosViewSet,
basename="pedidospisos")

urlpatterns = router.urls

# core/urls.py (trecho)
```

```
# path("api/<slug:slug>/", include("apps.pisos.urls"))
```

6.7 Fluxo de Execução na Prática

1) Front envia requisição: POST /api/casaa/pedidos-pisos/ com payload do pedido.
2) Middleware lê 'casaa' do path e define no contexto; router direciona ORM para o banco de 'casaa'.
3) ViewSet resolve 'banco' via get_licenca_db_config(request) e injeta no serializer.
4) Service cria cabeçalho e itens com .using(banco). Total é calculado e salvo.
5) Resposta retorna JSON do pedido com itens e totais do banco correto.

6.8 Boas Práticas (Pisos)

• Sempre passe 'banco' no context do serializer.

• Em listas, prefira filtros por empresa/filial e ordenações por data+número.

• Padronize quantização Decimal em helpers (q2).

• Para cargas grandes, use bulk_create com cuidado para manter item_nume sequencial.

6. App de Exemplo — Pisos

Estudo de caso prático do fluxo multi-tenant. Mostra como o slug seleciona o banco no middleware/router e como isso reflete nos models, serializers, services, views e urls.

6.1 Objetivo e Escopo

Gerenciar orçamentos e pedidos de pisos. Tabelas: orçamentos, itens de orçamento, pedidos e itens de pedido. Relacionamentos por campos de chave natural.

6.2 Models (resumo)

• Orcamentopisos: cabeçalho de orçamento. • Itensorcapisos: itens de orçamento. • Pedidospisos: cabeçalho de pedido. • Itenspedidospisos: itens de pedido. Todos com managed=False.

6.3 Serializers

```
# (código completo dos serializers do usuário inserido no repositório)
# Vide seção anterior com o conteúdo fornecido.
```

6.4 Services

```
# apps/pisos/services/pedidos_service.py
from decimal import Decimal, ROUND_HALF_UP
from django.db import transaction
from ..models import Pedidospisos, Itenspedidospisos

def q2(v):
```

```
    try:
        return Decimal(str(v)).quantize(Decimal('0.01'),
rounding=ROUND_HALF_UP)
    except Exception:
        return Decimal('0.00')

class PedidosService:
    @staticmethod
    @transaction.atomic
    def criar_pedido(banco, dados_cabecalho: dict, itens: list):
        ultimo = Pedidospisos.objects.using(banco).filter(
            pedi_empr=dados_cabecalho["pedi_empr"],
            pedi_fili=dados_cabecalho["pedi_fili"],
        ).order_by("-pedi_nume").first()
        dados_cabecalho["pedi_nume"] = (ultimo.pedi_nume + 1) if ultimo
else 1
        pedido =
Pedidospisos.objects.using(banco).create(**dados_cabecalho)

        total = Decimal('0.00')
        for idx, it in enumerate(itens, start=1):
            quan = Decimal(str(it.get("item_quan") or 0))
            unit = Decimal(str(it.get("item_unit") or 0))
            suto = q2(quan * unit)
            Itenspedidospisos.objects.using(banco).create(
                item_empr=pedido.pedi_empr, item_fili=pedido.pedi_fili,
                item_pedi=pedido.pedi_nume, item_nume=idx,
                item_suto=suto, item_prod=it.get("item_prod"),
                item_quan=q2(quan), item_unit=q2(unit),
                item_m2=q2(it.get("item_m2") or 0),
                item_desc=q2(it.get("item_desc") or 0),
                item_nome_ambi=it.get("item_nome_ambi", ""),
            )
            total += suto
        pedido.pedi_tota = q2(total -
q2(dados_cabecalho.get("pedi_desc") or 0) +
q2(dados_cabecalho.get("pedi_fret") or 0))
        pedido.save(using=banco)
        return pedido
```

6.5 Views

```
# apps/pisos/views.py
from rest_framework import viewsets, status
from rest_framework.response import Response
from core.utils import get_licenca_db_config
from .models import Orcamentopisos, Pedidospisos
from .serializers import OrcamentopisosSerializer,
```

```
PedidospisosSerializer
from .services.pedidos_service import PedidosService

class BaseBancoContextViewSet(viewsets.ModelViewSet):
    def get_banco(self):
        return get_licenca_db_config(self.request)

    def get_serializer_context(self):
        ctx = super().get_serializer_context()
        ctx["banco"] = self.get_banco()
        ctx["request"] = self.request
        return ctx

class OrcamentopisosViewSet(BaseBancoContextViewSet):
    serializer_class = OrcamentopisosSerializer
    queryset = Orcamentopisos.objects.none()
    def get_queryset(self):
        banco = self.get_banco()
        qs = Orcamentopisos.objects.using(banco).all()
        clie = self.request.query_params.get("orca_clie")
        if clie:
            qs = qs.filter(orca_clie=clie)
        return qs.order_by("-orca_data", "-orca_nume")

class PedidospisosViewSet(BaseBancoContextViewSet):
    serializer_class = PedidospisosSerializer
    queryset = Pedidospisos.objects.none()
    def get_queryset(self):
        banco = self.get_banco()
        qs = Pedidospisos.objects.using(banco).all()
        vend = self.request.query_params.get("pedi_vend")
        if vend:
            qs = qs.filter(pedi_vend=vend)
        return qs.order_by("-pedi_data", "-pedi_nume")
    def create(self, request, *args, **kwargs):
        banco = self.get_banco()
        serializer = self.get_serializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        dados_cabecalho = {k: v for k, v in
serializer.validated_data.items() if k != "itens_input"}
        itens = request.data.get("itens_input") or
request.data.get("itens") or []
        pedido = PedidosService.criar_pedido(banco, dados_cabecalho,
itens)
        out = self.get_serializer(pedido)
        return Response(out.data, status=status.HTTP_201_CREATED)
```

6.6 URLs

```
# apps/pisos/urls.py
from rest_framework.routers import DefaultRouter
from .views import OrcamentopisosViewSet, PedidospisosViewSet
router = DefaultRouter()
router.register(r"orcamentos-pisos", OrcamentopisosViewSet,
basename="orcamentopisos")
router.register(r"pedidos-pisos", PedidospisosViewSet,
basename="pedidospisos")
urlpatterns = router.urls
```

6.7 Fluxo prático

POST /api/<slug>/pedidos-pisos/ → Middleware define slug → Router resolve banco → ViewSet injeta 'banco' no serializer → Service cria cabeçalho+itens com .using(banco) → resposta JSON isolada por cliente.

7. Services e Cálculos — Resumo Executivo

A camada de serviços é responsável por concentrar a lógica de negócio do módulo Pisos, mantendo as views e serializers mais limpos e desacoplados. Todos os services utilizam o padrão multi-banco com chamadas explícitas .using(banco), garantindo que os dados sejam manipulados apenas dentro do banco do slug ativo.

7.1 utils_service.py

Módulo base com funções genéricas e seguras para manipulação de números decimais e preenchimento de dados de entidades.

Principais responsabilidades:

• parse_decimal: converte valores em Decimal com fallback seguro.

• arredondar: padroniza arredondamento com ROUND_HALF_UP.

• DadosEntidadesService: preenche automaticamente endereço, cidade e estado do cliente no pedido ou orçamento.

7.2 preco_service.py

Centraliza a busca de preços de produtos. Usa ORM quando possível e recorre a SQL cru como fallback em caso de tabelas não gerenciadas.

Fluxo:

1 Tenta obter o produto e preço via ORM (Produtos, Tabelaprecos).

2 Se falhar, executa SQL direto ordenando por _log_data e _log_time.

3 Retorna preço à vista (condicao='0') ou a prazo.

Erros são tratados com logging e fallback seguro.

7.3 calculo_service.py

Responsável pelos cálculos físicos de orçamentos e pedidos: metragem, perdas, caixas e totais por ambiente.

Funções principais:

• calcular_item: calcula m^2 com perda, número de caixas e total do item.

• calcular_ambientes: agrupa e soma itens por ambiente.

• calcular_total_geral: soma todos os ambientes e retorna o total geral.

Essa camada garante consistência entre o cálculo físico e financeiro, mantendo alinhamento com o modelo de produtos e suas unidades (m^2/caixa).

7.4 OrcamentoService e PedidoService

Esses serviços consolidam o cálculo de totais, preenchimento de dados de cliente e validação final antes da persistência.

• OrcamentoService.preparar_orcamento: atualiza totais de um orçamento existente.

• PedidoService.preparar_pedido: recalcula e sincroniza totais de um pedido com seus itens.

Ambos usam os helpers de cálculo e os serviços de entidade, garantindo coerência entre os valores exibidos no front-end e armazenados no banco.

7.5 Integração com o Roteamento Multi-Banco

Todos os services dependem da função get_licenca_db_config(request) para obter o alias do banco ativo, definido pelo middleware a partir do slug da requisição. Isso assegura que operações de leitura e gravação ocorram apenas dentro do contexto correto da empresa.

Na prática, o fluxo completo se dá assim:
1) Middleware extrai o slug (ex: casaa) e define no contexto.
2) Views resolvem banco = get_licenca_db_config(request).
3) Services executam queries .using(banco).
4) Resultados são serializados e enviados ao front-end.

8. Rodando o App Em novo ambiente virtual

Primeiramente, será necessário rodar o clone do repositório na máquina, posteriormente a isso, temos alguns passos importantes:

1º Python 3.12 – Intalar na máquina a versão do Python para que seja possível dar prosseguimento ao desenevolvimento com a mesma versão

2º Depois de feito a instalação temos que dar inicio ao ambiente de desenvolvimento, rodamos o comando:

python -m venv

Com isos iremos criar um ambiente virtual para que rodemos o desenvolvimento

Ativamos o ambiente rodando:

python venv/Scripts/activate

E depois disso iremos instalar as dependencias do projeto, que constam no arquivo de requirements, para tal iremos fazer da seguinte forma:

Pip install -r requirements.txt

Com esse comando iremos ler o arquivo das dependencias e instalar no nosso ambiente

Feito a instalação das dependências precisamos ter no ambiente virtual as nossas variáveis de ambiente, onde iremos ter, as databases que iremos rodar localmente, as chaves e variáveis que temos configuradas no settings para rodar o projeto e chaves externas, como abaixo:

Criar um arquivo .env na raiz como o modelo:

SECRET_KEY=django-insecure-5k*!z0aj1xyc*l(%uv!a6@x5dviogez-aoj7lf-iwp1o!_m2

DEBUG=False

USE_LOCAL_DB=True

LOCAL_DB_NAME=indus

LOCAL_DB_USER=postgres

LOCAL_DB_PASSWORD=@spartacus201@

LOCAL_DB_HOST=localhost

LOCAL_DB_PORT=5432

REMOTE_DB_NAME=savexml1024

REMOTE_DB_USER=savexml1024

REMOTE_DB_PASSWORD=kllllnnnnnnn

REMOTE_DB_HOST=base.rtalmeida.com.br

REMOTE_DB_PORT=5432

DEMONSTRACAO_DB_USER=savexml839

DEMONSTRACAO_DB_PASSWORD=eikkmoqqssuv

CASAA_DB_USER=postgres

CASAA_DB_PASSWORD=@spartacus201@

IPA_DB_USER=savexml952

IPA_DB_PASSWORD=zzzz22222244

ALLOWED_HOSTS=localhost,127.0.0.1,192.168.1.47,192.168.10.65,172.25.0.21,192.168.2.193,172.25.0.18,192.168.10.35,192.168.10.52,192.168.10.16,192.168.20.80,192.168.0.13,192.168.20.84,192.168.10.59,192.168.10.78,192.168.10.39,192.168.20.48

CORS_ALLOWED_ORIGINS=http://localhost:8081,http://192.168.1.47,http://172.25.0.21,http://192.168.20.80:8081,http://192.168.20.46:8081,http://192.168.10.39,http://172.25.0.21:8081,http://172.25.0.2

EMAIL_BACKEND=django.core.mail.backends.smtp.EmailBackend

EMAIL_HOST=smtp.savexml.com.br

EMAIL_PORT=587

EMAIL_USE_TLS=True

EMAIL_HOST_USER=notas@savexml.com.br

EMAIL_HOST_PASSWORD=sps@nfe

DEFAULT_FROM_EMAIL=notas@savexml.com.br

GOOGLE_API_KEY=AIzaSyB3Hgh1l2cnLLHTW1j0e2hjf9mlVOrS_DM

OPENAI_API_KEY=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

SMITHERY_API_KEY=5aa32046-6caf-4f75-b2c2-a403b4c0c886

REDIS_URL=redis://localhost:6379/1

CELERY_BROKER_URL=redis://localhost:6379/0

Com esses passos já Podemos seguir com o desenvolvimento local

Para rodarmos localmente, e com o ambiente já ativado rodamos:

Python manage.py runserver 0.0.0.0:8000

8.1 Comandos em Bases novas para rodar o Mobile:

Criar a base de no core/licencas.json com os dados de:

slug: o slug é o campo base para as urls, nele contém o nome da empresa que o sistema vai rotear todo o resto das requisições

por segurança o login e a senha são feitos com jwt, e o jwt é salvo no cookie, e é verificado em cada requisição, para verificar se o usuário está logado, no front com o asynk storage

da seguinte forma:

"slug": "ipa", # nome da empresa ou o savexml

"cnpj": "37413384000184", # cnpj da empresa que vai acessar a base

"db_name": "savexml952", # nome da base de dados, esse campo é importante, e deve ir nas variáveis de ambiente também

"db_host": "base.rtalmeida.com.br", # host da base de dados, nesse caso é o servidor

"db_port": "5432" # porta da base de dados,é a padrão do postgres

depois de configurar o licencas.json, rodar o setup_mobile.py, deixar no .env local as seguintes variáveis:

DEBUG=False

USE_LOCAL_DB=True

LOCAL_DB_NAME =savexml952

LOCAL_DB_USER =savexml952

LOCAL_DB_PASSWORD =senha do banco de dados

tem que ser o banco default, e se rodar ele dessa forma, irá rodar as migrações do banco de dados, e criar as tabelas necessárias para o acesso ao rodar o setuo_mobile.py

rodar o setup_mobile.py

rodando esse script iremos ter já os dados iniciais essenciais para rodar uma base nova do mobile, pois teremos um acesso com senha a tabela de usuários, acessando usua_senh_mobi, para o usuário admin

as API's estão documentadas com swagger, e podem ser acessadas no endpoint https://mobile-sps.site/api/schema/swagger-ui/#/, onde é possível visualizar todas as rotas já criadas para cada APP

em cada APP também temos as rotas documentadas para as suas ações essenciais

8.2 Comandos e configurações necessárias para o servidor

No servidor:

configurar o gunicorn e o nginx

pra acessar o servidor linux via ssh, é necessário ter o par de chaves SSH, que pode ser gerado com o comando:

ssh-keygen -t rsa -P "" -f SPARTACUS.pem

e depois de gerado, é necessário copiar o conteúdo do arquivo SPARTACUS.pem.pub para o arquivo authorized_keys do usuário ubuntu

cat. SPARTACUS.pem.pub >> ~/.ssh/authorized_keys

e depois de feito isso, é possível acessar o servidor via ssh

ssh -i SPARTACUS.pem ubuntu@168.75.73.117

/etc/systemd/system/gunicorn.service

esse é o conteúdo do arquivo de configuração do serviço

GNU nano 6.2 /etc/systemd/system/gunicorn.service [Unit]

Description=Gunicorn Django App

After=network.target

[Service]

User=ubuntu

Group=www-data

WorkingDirectory=/home/ubuntu/mobile-sps

ExecStart=/home/ubuntu/mobile-sps/venv/bin/gunicorn \

--access-logfile - \

--workers 9 \

--threads 2 \

--timeout 120 \

--preload \

--bind unix:127.0.0.1:8000 \

core.wsgi:application

Restart=always

RestartSec=3

[Install]

WantedBy=multi-user.target

depois de ajustado o gunicorn, reiniciar o serviço

sudo systemctl daemon-reload

sudo systemctl enable gunicorn

sudo systemctl start gunicorn

sudo systemctl status gunicorn

e pra verificar o stats do gunicorn

sudo systemctl status gunicorn

sudo journalctl -u gunicorn

depois disso configurar o nginx

vamos em:

sudo nano /etc/nginx/sites-available/default

aí seguimos a configuração para que o nginx redirecione as requisições para o gunicorn

server {

listen 80;

server_name 168.75.73.117;

root /var/www/html;

```
    index index.html index.htm index.nginx-debian.html;

    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # deny access to .htaccess files, if Apache's document root
    # concurs with nginx's one
    #
    #location ~ /\.ht {
    #    deny all;
    #}

}
```

e depois restartar o nginx

sudo systemctl restart nginx

sudo systemctl status nginx

para conferir se o nginx está funcionando

sudo nginx -t

e se der tudo certo, restartar o nginx

sudo systemctl restart nginx

VER TODOS OS LOGS DE ERRO

sudo tail -n 50 /var/log/nginx/error.log

Depois do processo completo, no servidor, puxar o mobile para ver as alterações

faz o pull no servidor

cd mobile-sps

source venv/bin/activate

git pull

#instala as dependências

python manage.py collectstatic

python manage.py runserver 0.0.0.0:8000

rodamos o update do gunicorn e do daphne

sudo systemctl restart gunicorn

sudo systemctl restart daphne

#Acompanhar os logs do servidor

sudo tail -f /var/log/nginx/error.log

sudo tail -f /var/log/nginx/access.log

sudo tail -f /var/log/nginx/error.log

sudo tail -f /var/log/nginx/access.log

9. Estrutura do Projeto

```
mobile_sps/
├── core/
│   ├── middleware.py
│   ├── routers.py
│   ├── utils.py
├── apps/
│   ├── pisos/
│   │   ├── models.py
│   │   ├── serializers.py
│   │   ├── views.py
│   │   ├── services/
│   │   │   ├── calculo_service.py
│   │   │   └── preco_service.py
│   └── outros_apps/
├── manage.py
└── requirements.txt
```