Documentação Técnica de Projeto

SyncHub - Plataforma de Integração para E-commerce

Versão Final (Corrigida)

Gerado por: Gemini (IA do Google)

29 de Agosto de 2025

Histórico de Versões

Versão	Data	Principais Alterações
1.0	Ago, 2025	Criação inicial do documento, definição do problema e da solução com arquitetura baseada em API + SPA (React).
2.0	Ago, 2025	Revisão da arquitetura para **Monolítico com SSR** (Django-fullstack). Adição de HTMX/Alpine.js ao stack. Escolha da **AWS** como provedor de nuvem principal.
3.0	Ago, 2025	Adição de **Apêndices** com detalhes técnicos aprofundados sobre a infraestrutura na nuvem (AWS VPC/Fargate), contêineres (Docker), exemplos de payload de API e schema do banco de dados.
Final	**29/08/2025**	**Consolidação final e correção de sintaxe LaTeX**. Adição do histórico de versões para formalizar o artefato.

Sumário

1 Introdução

1.1 Visão Geral do Projeto

O SyncHub é uma plataforma SaaS (Software as a Service) projetada para centralizar e automatizar a gestão de vendas para pequenos e médios comerciantes que atuam em múltiplos marketplaces online, como Mercado Livre, Shopee, Amazon, entre outros.

1.2 O Problema

Vendedores que utilizam mais de um canal de venda enfrentam desafios operacionais significativos, sendo o principal a gestão de estoque. A falta de sincronização em tempo real leva a dois cenários prejudiciais: `Overselling` (Venda sem Estoque) e `Underselling` (Perda de Vendas).

2 Arquitetura do Sistema

2.1 Visão Geral da Arquitetura

O sistema será desenvolvido como uma aplicação web **monolítica coesa**, utilizando o framework Django para todas as camadas. A arquitetura adotada será a de **Renderização no Lado do Servidor (Server-Side Rendering - SSR)**. Isso maximiza a produtividade, simplifica o stack tecnológico e oferece excelentes benefícios de SEO por padrão.

3 Stack Tecnológico

3.1 Backend

- **Framework:** Django (Python)

3.2 Frontend

- **Motor de Renderização:** Django Template Engine (DTL)
- **Melhorias de Interatividade:** `HTMX` e `Alpine.js`

3.3 Banco de Dados

- **SGBD:** PostgreSQL

3.4 Infraestrutura e Hospedagem (Nuvem)

- **Provedor Principal:** AWS (Amazon Web Services)
- **Serviços Essenciais:** Amazon ECS/Fargate, RDS for PostgreSQL, S3, ElastiCache for Redis.

3.5 Containerização e DevOps

- **Containerização:** Docker

A Arquitetura Detalhada de Infraestrutura

A.1 Estrutura na Nuvem (AWS)

A aplicação será implantada dentro de uma **VPC (Virtual Private Cloud)** para garantir o isolamento e a segurança da rede. A estrutura será dividida em:

- **Sub-redes Públicas:** Conterão o **Application Load Balancer (ALB)**, que é o ponto de entrada para todo o tráfego HTTP/S, e um **NAT Gateway**.
- **Sub-redes Privadas:** Conterão os componentes principais da aplicação, sem acesso direto da internet:
  - **AWS Fargate (para ECS):** Orquestrador de contêineres serverless que executará os serviços da aplicação (Django Web App, Celery Workers).
  - **Amazon RDS for PostgreSQL:** O banco de dados gerenciado.
  - **Amazon ElastiCache for Redis:** O broker para o Celery e cache.
- **CI/CD Pipeline:** O deploy será automatizado via **GitHub Actions**, que irá: construir a imagem Docker, enviá-la para o **Amazon ECR (Elastic Container Registry)** e atualizar o serviço no Fargate.

A.2 Estrutura de Contêineres (Docker)

A.2.1 Dockerfile (Aplicação Django)

```
# 1. Imagem base
FROM python:3.11-slim

# 2. Vari veis de ambiente
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# 3. Diret rio de trabalho
WORKDIR /app

# 4. Instalar depend ncias
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# 5. Copiar o c digo da aplica  o
COPY . .

# 6. Comando para executar a aplica  o (usando Gunicorn)
CMD ["gunicorn", "synchub.wsgi:application", "--bind", "0.0.0.0:8000"]
```

Listing 1: Dockerfile

A.2.2 Docker Compose (Desenvolvimento Local)

```yaml
version: '3.8'

services:
  web:
    build: .
    command: python manage.py runserver 0.0.0.0:8000
    volumes:
      - .:/app
    ports:
      - "8000:8000"
    depends_on:
      - db
      - redis

  db:
    image: postgres:15
    environment:
      - POSTGRES_DB=synchub
      - POSTGRES_USER=user
      - POSTGRES_PASSWORD=pass
    volumes:
      - postgres_data:/var/lib/postgresql/data/

  redis:
    image: redis:7

volumes:
  postgres_data:
```

Listing 2: docker-compose.yml

B Estrutura de Dados das APIs Externas

Aviso: As estruturas abaixo são exemplos ilustrativos. A documentação oficial de cada API deve ser consultada.

B.1 Exemplo de Webhook de Pedido (Mercado Livre)

```json
{
  "_id": "abc123def456",
  "resource": "/orders/456789123",
  "user_id": 123456789,
  "topic": "orders_v2",
  "application_id": 987654321
}
```

Listing 3: Notificação de Pedido do Mercado Livre

Detalhes do pedido:

```
{
  "id": 456789123,
  "status": "paid",
  "order_items": [
    {
      "item": { "seller_sku": "TQ-AZUL-01" },
      "quantity": 2
    }
  ]
}
```

Listing 4: Detalhes do Pedido do Mercado Livre

B.2 Exemplo de Webhook de Pedido (Shopee)

```
{
  "order_list": [
    {
      "ordersn": "250823ABCD123",
      "order_status": "PAID",
      "item_list": [
        {
          "item_sku": "TQ-AZUL-01",
          "model_quantity_purchased": 2
        }
      ]
    }
  ]
}
```

Listing 5: Detalhes do Pedido da Shopee

C Estrutura Detalhada do Banco de Dados (Schema SQL)

Abaixo, uma representação do schema para as tabelas principais.

C.1 Tabela `synchub_produtocentral`

```
CREATE TABLE synchub_produtocentral (
    id SERIAL PRIMARY KEY,
    sku VARCHAR(100) NOT NULL UNIQUE,
    titulo VARCHAR(200) NOT NULL,
    estoque_total INTEGER NOT NULL,
    usuario_id INTEGER NOT NULL REFERENCES auth_user(id)
);
```

C.2 Tabela synchub_loja

```
CREATE TABLE synchub_loja (
    id SERIAL PRIMARY KEY,
    plataforma VARCHAR(3) NOT NULL, -- 'ML' ou 'SHP'
    access_token VARCHAR(255) NOT NULL,
    refresh_token VARCHAR(255) NOT NULL,
    expires_in TIMESTAMP WITH TIME ZONE,
    usuario_id INTEGER NOT NULL REFERENCES auth_user(id)
);
```

C.3 Tabela synchub_anuncio

```
CREATE TABLE synchub_anuncio (
    id SERIAL PRIMARY KEY,
    id_anuncio_plataforma VARCHAR(100) NOT NULL,
    produto_central_id INTEGER NOT NULL REFERENCES synchub_produtocentral(id),
    loja_id INTEGER NOT NULL REFERENCES synchub_loja(id),
    UNIQUE (loja_id, id_anuncio_plataforma)
);
```