Plano de Implementação Detalhado: Plataforma de Gestão de Portfólio

Aluna: Évila Maria de Souza Carneiro
RA: 22553868

1. Introdução

Este documento apresenta o plano de implementação detalhado para o desenvolvimento de uma plataforma de gestão de portfólio, denominada Portfólio HUB para a disciplina Bootcamp I. O objetivo principal é criar um sistema seguro e de fácil manutenção, utilizando uma arquitetura baseada em Python/Django e MySQL, com integração direta à API do GitHub para sincronização automática de projetos.

O plano está estruturado em seis fases sequenciais, abrangendo desde a fundação do ambiente de desenvolvimento até a implantação em produção e a formalização das políticas de segurança e colaboração.

2. Stack Tecnológica Recomendada

A escolha da stack é fundamentada na necessidade de robustez, segurança administrativa e integração com serviços externos (OAuth e API do GitHub).

Componente	**Tecnologia**	**Função Principal**
Back-end	Django (Python)	Framework principal para lógica de negócios e gestão de conteúdo.
Banco de Dados	MySQL	Armazenamento persistente de dados de projetos, experiências e habilidades.
Autenticação	Django-Allauth	Gestão de acesso seguro à área administrativa via OAuth (GitHub).
API Interna	Django REST Framework (DRF)	Criação de *endpoints* JSON para consumo pelo *front-end*.
Servidor de Produção	Gunicorn/Whitenoise	Servidor WSGI e gestão de arquivos estáticos em ambiente de produção.

3. Fases de Implementação

Fase 1: Fundação e Ambiente (Git e Stack)

Esta fase estabelece a base técnica do projeto, garantindo um ambiente de desenvolvimento isolado, replicável e com controle de versão rigoroso. Define a estrutura fundamental, assegura o versionamento do trabalho desde o início e previne conflitos de dependência.

Passos de Configuração:

1. **Configuração do Repositório:**
    - Criação de um repositório privado no GitHub (`portfolio-hub`).
    - Inclusão de um arquivo `.gitignore` (utilizando o template "Python").
2. **Ambiente Local:**
    - Clonagem do repositório e criação de um ambiente virtual Python (`venv`).
    - Ativação do ambiente virtual.
3. **Estratégia de Branching:**
    - Adoção de um fluxo de trabalho simplificado (GitFlow).
    - Definição das branches: `main` (produção), `develop` (desenvolvimento principal) e `feature/...` (funcionalidades).
    - Configuração de `develop` como a branch padrão.
4. **Instalação da Stack:**
    - Instalação das dependências base (`django`, `mysqlclient`, `python-dotenv`).
    - Criação do projeto Django (`django-admin startproject portfolio_hub .`) e do *app* principal (`python manage.py startapp core`).
5. **Configuração Inicial do Django:**
    - Inclusão do *app* `core` em `INSTALLED_APPS`.
    - Configuração do `settings.py` para conexão com o banco de dados MySQL local.
6. **Commit Inicial:**
    - Registro da estrutura inicial do projeto no controle de versão.

Fase 2: Gestão de Acesso e Segurança (Back-end)

O foco desta fase é proteger a área administrativa da plataforma, implementando um sistema de autenticação de usuário único via serviço externo (GitHub OAuth). Garante que apenas o administrador autenticado possa acessar e modificar o conteúdo, atendendo aos requisitos de segurança e gestão de usuário.

Passos de Configuração:

1. **Modelagem de Dados:**

- Definição dos modelos de dados (`Project`, `Experience`, `Skill`) no `core/models.py`.
- O modelo `Project` deve incluir campos para armazenar dados sincronizados do GitHub (ex: `repo_name`, `stars_count`).
- Execução das migrações (`makemigrations` e `migrate`).

2. **Registro do OAuth App no GitHub:**
   - Registro da aplicação na seção *Developer settings* do GitHub.
   - Configuração das URLs de *Homepage* e *Authorization callback* (para ambiente local e, posteriormente, produção).
3. **Armazenamento Seguro de Chaves:**
   - Criação de um arquivo `.env` para armazenar chaves sensíveis (`GITHUB_CLIENT_ID`, `GITHUB_CLIENT_SECRET`, `DJANGO_SECRET_KEY`).
   - Inclusão imediata do `.env` no `.gitignore`.
   - Uso de `python-dotenv` para carregar as variáveis de ambiente no `settings.py`.
4. **Configuração do `django-allauth`:**
   - Instalação e configuração do `django-allauth` para usar o provedor GitHub.
   - Integração das URLs de autenticação no `urls.py` principal.
5. **Criação do Painel Administrativo:**
   - Registro dos modelos no `core/admin.py` para permitir a gestão de conteúdo via Django Admin, que agora está protegido pelo login do GitHub.

Fase 3: Integração com API do GitHub (Sincronização de Conteúdo)

Esta fase implementa a funcionalidade de sincronização de projetos, permitindo que o sistema popule o banco de dados automaticamente com informações dos repositórios do GitHub. Automatiza a gestão de conteúdo, mantendo o portfólio atualizado com base na atividade do repositório externo.

Passos de Configuração:

1. **Serviço de Sincronização:**
   - Criação de um módulo de serviço (`core/github_service.py`).
   - Implementação de uma função que utiliza a biblioteca `requests` (ou similar) para fazer chamadas autenticadas à API do GitHub (`api.github.com/user/repos`).
   - A função deve iterar sobre os repositórios e utilizar `Project.objects.update_or_create()` para persistir/atualizar os dados no MySQL.
2. **Gatilho de Sincronização:**
   - Implementação de um *Management Command* do Django (ex: `python manage.py sync_projects`) para executar a sincronização via terminal.
   - Este comando pode ser agendado para execução automática em produção (via Cron Job).

Fase 4: API Pública e Front-end (Exibição de Conteúdo)

Esta fase foca na exposição pública dos dados de forma profissional, separando a camada de gestão (Back-end) da camada de apresentação (Front-end). Permite que o conteúdo gerenciado internamente seja exibido de forma estruturada e acessível a visitantes.

Passos de Configuração:

1. **Criação da API REST:**
    - Instalação do Django REST Framework (`djangorestframework`).
    - Criação de *Serializers* (`core/serializers.py`) para converter modelos MySQL em formato JSON.
    - Definição de *Views* de API (ex: `ListAPIView`) para criar *endpoints* públicos (ex: `/api/v1/projects/`).
2. **Desenvolvimento do Front-end:**
    - O *front-end* será *read-only* (somente leitura).
    - **Opção Recomendada (Django Templates):** Criação de *views* normais que buscam os dados diretamente do banco (`Project.objects.all()`) e os renderizam em templates HTML (`index.html`, `projects.html`) usando a *template engine* do Django.
    - **Opção Avançada:** Desenvolvimento de uma aplicação *front-end* separada (ex: React, Vue) que consumirá os *endpoints* JSON criados pelo DRF.

Fase 5: Implantação e Políticas de Segurança (Produção)

Esta fase abrange a migração do ambiente local para um servidor público, garantindo estabilidade, desempenho e segurança em produção. Finaliza o ciclo de desenvolvimento, tornando a plataforma acessível publicamente e implementando as políticas de segurança essenciais para um ambiente de produção.

Passos de Configuração:

1. **Preparação para Produção:**
    - Instalação de dependências de produção (`gunicorn`, `whitenoise`).
    - Geração do arquivo `requirements.txt`.
    - Configuração do `whitenoise` no `settings.py` para servir arquivos estáticos.
2. **Configurações de Segurança:**
    - Definição de `DEBUG = False` no `settings.py`.
    - Configuração de `ALLOWED_HOSTS` para o domínio de produção.
    - Habilitação de proteções de cookies (`CSRF_COOKIE_SECURE = True`, `SESSION_COOKIE_SECURE = True`).
3. **Deploy na Plataforma:**
    - Escolha de uma plataforma de hospedagem que suporte Python/Django e MySQL (ex: Render, PythonAnywhere).
    - Criação de um banco de dados MySQL de produção.

- Configuração das Variáveis de Ambiente (Environment Variables) na plataforma, replicando as chaves do `.env`.
- Definição dos comandos de *Build* (`pip install -r requirements.txt && python manage.py migrate`) e *Start* (`gunicorn portfolio_hub.wsgi`).

4. **Atualização do GitHub OAuth App:**
    - Atualização das URLs de *callback* do OAuth App no GitHub para o domínio de produção.
5. **Testes Finais:**
    - Verificação do carregamento via HTTPS, funcionalidade do login de admin via GitHub e exibição correta do conteúdo.

Fase 6: Documentação e Colaboração (Workflow)

A fase final formaliza o processo de desenvolvimento e as práticas de colaboração, garantindo a qualidade e a compreensibilidade do projeto. Documenta o processo de configuração e o fluxo de trabalho profissional, facilitando a manutenção e a colaboração futura.

Passos de Configuração:

1. **Documentação `README.md`:**
    - Atualização do `README.md` com a descrição do projeto, a stack de tecnologias e instruções detalhadas de **instalação local** (clonagem, `venv`, dependências, `.env`, migrações e execução do servidor).
2. **Documentação do Workflow:**
    - Criação de um arquivo `CONTRIBUTING.md` (ou seção no `README.md`).
    - Descrição do fluxo de Git adotado:
        - Criação de *feature branches* a partir de `develop`.
        - Envio da *feature branch* para o GitHub.
        - Abertura de um **Pull Request (PR)** para `develop`.
        - Revisão e *Merge* do PR.
        - Eliminação da *feature branch* após o *merge*.
    - Esta prática assegura um histórico de código limpo e disciplinado.