**MANUAL DE IMPLANTAÇÃO E ARQUITETURA DA PLATAFORMA IAPROVAS**

**1. VISÃO GERAL DO PROJETO**

**1.1 Descrição**

A IAprovas é uma plataforma SaaS (Software as a Service) de apoio a estudantes de concursos públicos que utiliza Inteligência Artificial para personalizar e otimizar o processo de estudo. A plataforma oferece funcionalidades como cronogramas personalizados, exercícios inteligentes, correção de discursivas, flashcards e dicas estratégicas.

**1.2 Tecnologias Principais**

A arquitetura foi revisada para adotar tecnologias que garantem maior performance, escalabilidade e uma melhor experiência de desenvolvimento.

* **Frontend:** React (ou Vue.js) para uma interface reativa e componentizada, utilizando HTML5, CSS3, e JavaScript (ES6+).
* **Backend:** Python com **FastAPI** para alta performance e suporte nativo a operações assíncronas.
* **Banco de Dados:** Firebase Firestore, para dados NoSQL flexíveis e escaláveis.
* **Autenticação:** Firebase Auth, para um sistema de autenticação robusto e seguro.
* **IA e Processamento de Linguagem:** OpenAI **GPT-4**.
* **Banco Vetorial:** **Pinecone** ou um provedor similar para otimizar as buscas semânticas do sistema RAG.
* **Filas de Tarefas Assíncronas:** **Redis** e **Celery** para gerenciar jobs em background (geração de planos, correções), garantindo que a API principal permaneça responsiva.
* **Pagamentos:** Stripe.
* **Email:** Resend.
* **Deploy:** Render, para uma gestão simplificada de infraestrutura como código.
* **Analytics:** Google Analytics.

**2. ARQUITETURA DO SISTEMA**

**2.1 Arquitetura Geral Revisada**

A arquitetura foi atualizada para refletir o desacoplamento de serviços assíncronos, melhorando a resiliência e a performance do sistema. A API principal (FastAPI) lida com as requisições síncronas e delega tarefas pesadas para os workers (Celery), que se comunicam através de uma fila (Redis).

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐

│ Frontend │ │ Backend │ │ Serviços │

│ (React) │◄──►│ (FastAPI) │◄──►│ Externos │

└─────────────────┘ └───────┬─────────┘ └─────────────────┘

▲ │ │ - OpenAI

│ │ │ - Stripe

│ │ │ - Resend

│ ▼ └─────────────────┘

┌───────────────┴─┐ ┌─────────────────┐

│ Firebase │ │ Worker (Celery) │

│ - Firestore │ └───────┬─────────┘

│ - Auth │ │

└─────────────────┘ ▼

┌─────────────────┐

│ Redis │

└─────────────────┘

**2.2 Estrutura de Diretórios Otimizada**

A estrutura de diretórios foi reorganizada para refletir uma abordagem modular, separando as responsabilidades e incluindo um diretório dedicado para testes.

plataforma\_concursos/

├── app/

│ ├── \_\_init\_\_.py

│ ├── main.py # Ponto de entrada da API FastAPI

│ ├── core/ # Configurações centrais, middlewares

│ │ └── config.py

│ ├── features/ # Módulos por funcionalidade (Routers)

│ │ ├── \_\_init\_\_.py

│ │ ├── schedules/

│ │ ├── exercises/

│ │ └── payments/

│ ├── services/ # Lógica de integração com serviços externos

│ │ ├── openai\_service.py

│ │ ├── pinecone\_service.py

│ │ └── stripe\_service.py

│ └── workers/ # Definição das tarefas assíncronas (Celery)

│ └── tasks.py

├── tests/ # Diretório de testes

│ ├── \_\_init\_\_.py

│ ├── test\_schedules.py

│ └── test\_exercises.py

├── .env # Variáveis de ambiente

├── .gitignore

├── requirements.txt # Dependências Python

├── build.sh # Script de build

├── docker-compose.yml # Configuração do ambiente local

└── README.md

**3. CONFIGURAÇÃO DO AMBIENTE**

**3.1 Pré-requisitos**

* Python 3.9+
* Node.js 16+
* Docker e Docker Compose
* Contas nos serviços: Firebase, OpenAI, Stripe, Resend, Pinecone e Render.

**3.2 Configuração Local com Docker**

O uso de Docker é o método recomendado para garantir consistência entre ambientes.

1. **Crie o arquivo docker-compose.yml:**

YAML

version: '3.8'

services:

backend:

build: .

ports:

- "8000:8000"

volumes:

- ./app:/app

env\_file:

- .env

depends\_on:

- redis

redis:

image: "redis:alpine"

worker:

build: .

command: celery -A app.workers.tasks worker --loglevel=info

volumes:

- ./app:/app

env\_file:

- .env

depends\_on:

- redis

1. **Crie o arquivo .env** na raiz com as variáveis de ambiente necessárias (OpenAI, Stripe, etc.).
2. **Suba o ambiente:**

Bash

docker-compose up --build

**4. FUNCIONALIDADES PRINCIPAIS E FLUXOS TÉCNICOS**

**4.1 Elaboração de Cronogramas (Fluxo Assíncrono)**

* **Objetivo:** Gerar um plano de estudos detalhado sem bloquear a interface do usuário.
* **Fluxo Técnico:**
  1. O Frontend (React) envia uma requisição POST para /gerar-plano-estudos com os parâmetros do usuário.
  2. A API (FastAPI) valida os dados com Pydantic, cria uma entrada de "tarefa pendente" no Firestore e enfileira um job no Celery, passando os parâmetros necessários.
  3. A API retorna imediatamente uma resposta 202 Accepted com um ID de tarefa para o frontend. O usuário pode continuar navegando.
  4. O Worker (Celery) pega a tarefa da fila (Redis), constrói o prompt para o GPT-4, faz a chamada à API da OpenAI, recebe o cronograma em JSON, e o salva no documento correspondente no Firestore, atualizando o status para "concluído".
  5. O Frontend pode usar o ID da tarefa para fazer polling de status ou ser notificado em tempo real (via WebSockets ou Firebase Realtime Updates) quando o cronograma estiver pronto.

**4.2 Exercícios Inteligentes com RAG (Retrieval-Augmented Generation)**

* **Objetivo:** Fornecer questões inéditas e factualmente precisas, baseadas em uma base de conhecimento confiável.
* **Arquitetura RAG Implementada:**
  1. **Ingestão de Dados (Processo Offline):** Um script de ingestão lê os materiais de estudo (PDFs, etc.), os divide em "chunks" (pedaços de texto), gera vetores de *embeddings* para cada chunk via API da OpenAI, e armazena esses vetores no banco de dados vetorial **Pinecone**, junto com o texto original.
  2. **Geração em Tempo Real (Processo Online):** a. O frontend requisita questões sobre um tópico. b. A API (FastAPI) recebe a requisição, cria um embedding da consulta do usuário. c. **Retrieval:** O backend consulta o **Pinecone** para encontrar os N chunks de texto cujos vetores são mais semanticamente similares ao vetor da consulta. Essa busca é ultrarrápida. d. **Augmentation & Generation:** Estes chunks recuperados são inseridos em um prompt para o GPT-4, com a instrução de gerar uma questão baseada *exclusivamente* naquele contexto.
  3. A questão formatada em JSON é retornada ao frontend.

**5. ESTRUTURA DO BANCO DE DADOS (FIRESTORE)**

A estrutura do Firestore é mantida, pois é sólida e bem definida. As coleções principais incluem users, schedules, flashcardDecks, e userMetrics.

**6. API ENDPOINTS E DOCUMENTAÇÃO**

A API segue o padrão OpenAPI, com documentação interativa gerada automaticamente pelo FastAPI, o que torna os testes e a integração muito mais simples.

* **Documentação Interativa:** Disponível em /docs (Swagger UI) e /redoc (ReDoc) da URL base da API.
* **Endpoints Principais:**
  + POST /gerar-plano-estudos-async: Inicia a geração de um cronograma.
  + POST /gerar-exercicios: Gera exercícios usando o sistema RAG.
  + POST /corrigir-discursiva-async: Inicia a correção de uma questão discursiva.
* **Validação de Dados:** Todos os dados de entrada são rigorosamente validados via Pydantic, garantindo maior robustez e segurança.

**7. DEPLOY, INFRAESTRUTURA E CI/CD**

**7.1 Pipeline de CI/CD com GitHub Actions**

O deploy é automatizado com um pipeline de CI/CD. Cada push na branch main dispara o seguinte fluxo:

YAML

# .github/workflows/deploy.yml

name: CI/CD Pipeline

on:

push:

branches: [ main ]

jobs:

build-and-deploy:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- name: Set up Python

uses: actions/setup-python@v4

with:

python-version: '3.9'

- name: Install Dependencies

run: pip install -r requirements.txt

- name: Run Tests

run: pytest

- name: Deploy to Render

if: success() # Só faz o deploy se os testes passarem

run: curl -X POST ${{ secrets.RENDER\_DEPLOY\_HOOK\_URL }}

**7.2 Render (Produção)**

* **Build Command:** pip install -r requirements.txt
* **Start Command:** gunicorn -w 4 -k uvicorn.workers.UvicornWorker app.main:app (Otimizado para FastAPI)

**8. ESTRATÉGIA DE TESTES**

* **Testes Unitários (Pytest):** Foco em testar funções e classes de forma isolada (services, lógica de negócio).
* **Testes de Integração (Pytest):** Garantir que a integração entre a API e o Firebase funciona, usando um banco de dados de teste.
* **Testes End-to-End (Cypress/Playwright):** Simular a jornada completa do usuário no frontend, validando fluxos críticos.

**9. SEGURANÇA AVANÇADA**

* **Segurança de Dependências:** Análise de vulnerabilidades com **Dependabot** (GitHub) no pipeline.
* **Regras de Segurança do Firestore:** Regras rigorosas para garantir que um usuário só possa acessar seus próprios dados.
* match /users/{userId}/{document=\*\*} {
* allow read, write: if request.auth.uid == userId;
* }
* **Validação de Entrada:** O FastAPI com Pydantic valida todos os dados de entrada da API, prevenindo ataques de injeção.

**10. ESCALABILIDADE**

O plano de escalabilidade é suportado pela nova arquitetura:

* **Workers Assíncronos (Celery):** Permitem escalar o processamento de tarefas pesadas de forma independente da API principal.
* **Banco Vetorial Dedicado (Pinecone):** Garante que a busca semântica, um gargalo em potencial, seja escalável e performática.

**11. CUSTOS E OTIMIZAÇÕES**

* **Cache de Embeddings:** Para textos que não mudam (ex: artigos de lei), os embeddings podem ser cacheados para evitar chamadas repetidas e caras à API da OpenAI.
* **Workers Otimizados:** Ajustar o número de workers Celery conforme a demanda para equilibrar performance e custo.

**12. ROADMAP E MELHORIAS**

**12.1 Curto Prazo (1-3 meses)**

* [ ] Otimização de performance do frontend (code splitting, lazy loading).
* [ ] Aumentar a cobertura de testes para 80%.
* [ ] Melhorias na UI/UX baseadas no feedback inicial.
* [ ] Sistema de notificações via Firebase Cloud Messaging.

**12.2 Médio Prazo (3-6 meses)**

* [ ] Expansão para ENEM e OAB.
* [ ] App mobile (React Native ou Flutter).
* [ ] Integração com Google Calendar/Apple Calendar.
* [ ] Sistema de gamificação para engajamento.