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

**1. VISÃO GERAL DO PROJETO**

**1.1 Descrição**

A **IAprovas** é uma plataforma SaaS 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, uma evolução natural do Flask.
* **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, desacoplando do provedor de embeddings.
* **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.

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

│ 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 com "Blueprints" ou "Routers", 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

│ │ ├── \_\_init\_\_.py

│ │ ├── auth/

│ │ ├── plans/

│ │ └── payments/

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

│ │ ├── openai\_service.py

│ │ └── stripe\_service.py

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

│ └── tasks.py

├── tests/ # Diretório de testes

│ ├── \_\_init\_\_.py

│ ├── test\_auth.py

│ └── test\_plans.py

├── .env # Variáveis de ambiente

├── .gitignore

├── requirements.txt # Dependências Python

├── build.sh # Script de build

└── README.md

**3. CONFIGURAÇÃO DO AMBIENTE**

**3.1 Pré-requisitos**

* Python 3.9+
* Node.js 16+
* Docker e Docker Compose (para ambiente local padronizado)
* Contas nos serviços: Firebase, OpenAI, Stripe, Resend, e Render.

**3.2 Configuração Local com Docker**

Para simplificar o setup e garantir consistência entre ambientes, recomendamos o uso de Docker.

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

YAML

version: '3.8'

services:

backend:

build: .

ports:

- "8000:8000"

volumes:

- .:/app

env\_file:

- .env

redis:

image: "redis:alpine"

1. **Crie o arquivo .env** na raiz do projeto com as variáveis de ambiente.
2. **Suba o ambiente:**

Bash

docker-compose up --build

**3.3 Configuração de Serviços**

As configurações de serviços externos (firebase-config.js, stripe-config.js) devem ser gerenciadas como variáveis de ambiente no frontend, injetadas durante o processo de build para maior segurança.

**4. FUNCIONALIDADES PRINCIPAIS**

As funcionalidades foram mantidas, mas o sistema de geração de exercícios foi aprimorado.

**4.1 Exercícios Inteligentes com RAG**

* **Sistema RAG**: A geração de questões utiliza o padrão Retrieval-Augmented Generation (RAG).
* **Banco Vetorial**: Os embeddings das questões são armazenados em um banco de dados vetorial como o Pinecone, permitindo buscas semânticas ultrarrápidas e escaláveis, independentemente da origem dos embeddings.

**5. ESTRUTURA DO BANCO DE DADOS**

A estrutura do Firestore é mantida, pois é sólida e bem definida para as necessidades da aplicação.

**6. API ENDPOINTS E DOCUMENTAÇÃO**

A API agora segue o padrão **OpenAPI**, com documentação interativa gerada automaticamente pelo FastAPI.

* **Documentação Interativa**: Disponível em /docs (Swagger UI) e /redoc (ReDoc).
* **Endpoints**: Os endpoints foram mantidos, mas agora com validação de dados via Pydantic, garantindo maior robustez.
  + POST /gerar-plano-estudos
  + POST /gerar-exercicios-async
  + POST /corrigir-discursiva-async

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

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

O deploy foi automatizado com um pipeline de CI/CD, eliminando a necessidade de scripts manuais e garantindo a qualidade do código.

**Arquivo .github/workflows/deploy.yml:**

YAML

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

# Trigger Render deploy hook via curl

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

**7.2 Render (Produção)**

A configuração no Render permanece, mas agora é disparada pelo pipeline de CI/CD.

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

**8. ESTRATÉGIA DE TESTES**

Uma estratégia de testes robusta é fundamental para a qualidade da plataforma.

**8.1 Testes Unitários**

* **Ferramenta**: Pytest.
* **Objetivo**: Testar funções e classes de forma isolada, com foco na lógica de negócio e nos serviços.

**8.2 Testes de Integração**

* **Objetivo**: Garantir que a integração entre a API e o banco de dados (Firebase) funciona como esperado, utilizando um banco de dados de teste.

**8.3 Testes End-to-End (E2E)**

* **Ferramenta**: Cypress ou Playwright.
* **Objetivo**: Simular a jornada completa do usuário no frontend, validando os fluxos críticos como cadastro, geração de cronograma e pagamento.

**9. SEGURANÇA AVANÇADA**

**9.1 Segurança de Dependências**

* **Análise de Vulnerabilidades**: Implementação do **Dependabot** (GitHub) ou pip-audit no pipeline de CI/CD para identificar e alertar sobre dependências vulneráveis.

**9.2 Regras de Segurança do Firestore**

* **Regras Detalhadas**: As regras devem ser rigorosas, garantindo que um usuário só possa ler e escrever seus próprios dados. Exemplo:
* match /users/{userId}/{document=\*\*} {
* allow read, write: if request.auth.uid == userId;
* }

**9.3 Validação e Sanitização**

* **Validação de Entrada**: O FastAPI, com Pydantic, valida todos os dados de entrada da API, prevenindo ataques de injeção e garantindo a integridade dos dados.

**10. ESCALABILIDADE**

O plano de escalabilidade foi mantido, pois já contempla as práticas recomendadas como cache, CDN e a eventual migração para microserviços.

**11. CUSTOS E OTIMIZAÇÕES**

As otimizações foram mantidas, com ênfase no cache de embeddings da OpenAI para reduzir custos recorrentes.

**12. ROADMAP E MELHORIAS**

O roadmap foi atualizado para incluir débitos técnicos e melhorias na experiência do desenvolvedor.

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

* [ ] Otimização de performance do frontend.
* [ ] **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.

**Versão**: 3.0 **Data**: Julho 2025 **Autor**: Equipe IAprovas (Revisado por Desenvolvedor Sênior) **Última Atualização**: Baseada na refatoração da arquitetura para maior escalabilidade e adoção de melhores práticas de desenvolvimento.