**Manual de Implantação e Arquitetura da Plataforma IAprovas**

**Versão 1.2 - 19 de Junho de 2025**

Este documento descreve o passo a passo completo para configurar, desenvolver e implantar a plataforma IAprovas a partir do zero, incluindo uma descrição da função de cada arquivo no projeto.

**Parte 1: Configuração do Ambiente de Desenvolvimento Local**

Estas são as ações que você executa no seu computador para poder trabalhar no projeto.

* **1.1. Procedimento de Início de Dia:** Sempre que for começar a trabalhar, abra o VS Code e o terminal integrado e siga estes passos:
  1. Navegue até a pasta do projeto:

Bash

cd caminho/para/seu/projeto/plataforma\_concursos-e9b4feb71d9463b63ef688c227673a5f287b9c50

* 1. Ative o ambiente virtual do Python:

Bash

source venv/bin/activate

* 1. Inicie o servidor de backend:

Bash

python3 app.py

* 1. Inicie o servidor do frontend: No VS Code, clique com o botão direito no arquivo index.html e selecione "Open with Live Server".
* **1.2. Procedimento de Fim de Dia (Salvar no GitHub):** Ao final de cada sessão de trabalho, para salvar seu progresso de forma segura:
  1. No terminal, pare o servidor app.py com Ctrl + C.
  2. Verifique as alterações: git status
  3. Adicione todas as alterações: git add .
  4. Crie um "pacote" com uma mensagem descritiva (Commit):

Bash

git commit -m "fix: Descreve a correção ou feat: para nova funcionalidade"

* 1. Envie para o GitHub: git push
  2. Desative o ambiente virtual: deactivate
  3. Feche o VS Code.

**Parte 2: Integração de Serviços Externos**

Configuramos quatro serviços principais: Firebase, Stripe, OpenAI e Resend.

* **2.1. Firebase (Banco de Dados e Autenticação):**
  + **Configuração no Código:** As credenciais de frontend do Firebase (apiKey, authDomain, etc.) estão no arquivo firebase-config.js. A segurança é garantida pelas Regras de Segurança configuradas no painel do Firestore.
  + **Configuração no Painel do Firebase:**
    - **Domínios Autorizados:** Para que o login e cadastro funcionem em iaprovas.com.br, fomos em Authentication > Settings > Domínios autorizados e adicionamos iaprovas.com.br e www.iaprovas.com.br.
    - **Chave de API:** Para evitar o erro 403 (Forbidden), fomos no Google Cloud Console > APIs e serviços > Credenciais, selecionamos a chave de API do nosso app web e, em "Restrições de aplicativos", adicionamos iaprovas.com.br e \*.iaprovas.com.br aos "Referenciadores HTTP".
    - **Conta de Serviço (Backend):** Para que nosso app.py pudesse acessar o Firestore com permissões de administrador, geramos uma chave de serviço em Configurações do projeto > Contas de serviço. O arquivo JSON baixado foi renomeado para serviceAccountKey.json e colocado no ambiente de produção através dos "Secret Files" da Render.
* **2.2. Stripe (Pagamentos e Assinaturas):**
  + **Estrutura de Produtos e Preços:** Utilizamos um único Produto na Stripe chamado "IAprovas Assinatura". Associados a este produto, foram criados múltiplos Preços para cada plano oferecido (Básico, Intermediário, Premium Mensal, Premium Anual).
  + **Portal do Cliente (Customer Portal):** A funcionalidade "Gerenciar Assinatura" é implementada através de uma rota /create-portal-session no app.py. Essa rota utiliza o stripeCustomerId (salvo no Firestore durante o checkout) para gerar um link de acesso seguro ao portal do cliente da Stripe.
  + **Webhooks:** O webhook checkout.session.completed está configurado no painel da Stripe para apontar para https://iaprovas-backend.onrender.com/stripe-webhook. Ele é responsável por atualizar o plano do usuário e salvar o stripeCustomerId no Firestore.
* **2.3. OpenAI (Inteligência Artificial):**
  + **Controle de Custos:** Um limite de uso mensal (budget) foi configurado em Settings > Limits no painel da OpenAI para controlar os gastos.
* **2.4. Resend (E-mails Transacionais):**
  + **Configuração:** Optou-se pelo Resend devido à sua API moderna e plano gratuito. A chave de API foi configurada como variável de ambiente no backend.
  + **Domínio Verificado:** O domínio iaprovas.com.br foi verificado no Resend para garantir a máxima entregabilidade dos e-mails.

**Parte 3: Deploy na Render (Produção)**

Colocamos a aplicação no ar usando a plataforma Render.

* **3.1. Arquivos de Configuração para Deploy:**
  + requirements.txt: Lista todas as bibliotecas Python que o backend precisa.
  + build.sh: Script executado pelo Render para instalar as dependências (pip install -r requirements.txt).
  + Procfile: Define o comando para iniciar o servidor: web: gunicorn app:app.
* **3.2. Deploy do Backend (Web Service):**
  + Criamos um "Web Service" na Render conectado ao nosso GitHub.
  + **Configurações:** Runtime: Python 3, Build Command: bash build.sh, Start Command: gunicorn app:app.
  + **Variáveis de Ambiente:** Configuramos um "Environment Group" na Render com as chaves: OPENAI\_API\_KEY, STRIPE\_SECRET\_KEY (produção), STRIPE\_WEBHOOK\_SECRET (produção) e o conteúdo do serviceAccountKey.json como um "Secret File".
* **3.3. Deploy do Frontend (Static Site):**
  + Criamos um "Static Site" na Render conectado ao GitHub.
  + **Configurações:** Publish Directory: . (ponto), para publicar todos os arquivos HTML/CSS/JS.

**Parte 4: Arquitetura do Projeto (O que cada arquivo faz)**

* **4.1. Arquivos HTML (A Estrutura):**
  + index.html: Página inicial, com a apresentação das funcionalidades e planos.
  + login.html, cadastro.html: Páginas de autenticação.
  + home.html, cronograma.html, exercicios.html, discursivas.html, dicas-estrategicas.html: Páginas de funcionalidades do dashboard do usuário.
  + meu-perfil.html: Página de gerenciamento da conta do usuário, incluindo dados pessoais, alteração de senha, gestão de assinatura e exclusão de conta.
  + termos.html, privacidade.html, sucesso.html, cancelado.html: Páginas de fluxo e conteúdo legal.
  + contratacao.html: Página para iniciar o fluxo de pagamento, com um modal para guiar usuários não logados.
* **4.2. Arquivos de Estilo CSS (A Aparência):**
  + styles.css: Estilo principal para páginas públicas (como index.html, login.html).
  + dashboard-lovable.css: Estilo para o painel do usuário e páginas de fluxo de pagamento e perfil. Contém todos os estilos para os componentes do dashboard, modais, formulários e layouts das páginas internas.
* **4.3. Arquivos JavaScript (A Interatividade):**
  + **main-app.js (Atualizado):** Orquestrador principal do dashboard. Gerencia o estado de autenticação do usuário, carrega seus dados do Firestore, controla o acesso às funcionalidades com base no plano e dispara o evento userDataReady para sincronizar a inicialização de outras páginas.
  + **auth.js (Atualizado):** Controla a lógica de login e cadastro, utilizando o parâmetro de URL ?returnTo= para redirecionar o usuário de volta ao fluxo de compra após a autenticação.
  + **api.js (Atualizado):** Centraliza todas as chamadas fetch para o backend. A API\_BASE\_URL é definida dinamicamente para funcionar em ambiente local e de produção. Contém funções para todas as interações com o servidor, como gerarPlanoDeEstudos e deletarContaUsuario.
  + **perfil-page.js (Atualizado):** Contém toda a lógica da página "Meu Perfil". Ouve o evento userDataReady e configura os listeners para salvar perfil, alterar senha, gerenciar assinatura e excluir conta.
  + **cronograma-page.js (Atualizado):** Gerencia a página de cronogramas.
    - O formulário agora chama uma rota que inicia um **trabalho assíncrono** no backend.
    - Utiliza um listener em tempo real do Firestore (onSnapshot) para "ouvir" as atualizações do cronograma.
    - A interface exibe o status "Gerando..." e atualiza automaticamente para "Abrir" quando o plano está pronto, sem precisar recarregar a página.
    - A exibição do plano e a exportação para Excel são feitas após buscar o documento completo no Firestore.
  + **Outros scripts de página (exercicios-page.js, discursivas-page.js, etc.):** Lógica específica para cada funcionalidade.
* **4.4. Backend e Arquivos de Servidor (app.py):**
  + **app.py (Atualizado):** O motor da plataforma.
    - **Geração de Cronograma Assíncrona:** A rota /gerar-plano-estudos agora é assíncrona. Ela recebe a solicitação, cria um documento "placeholder" no Firestore com status: 'processing', inicia a chamada demorada para a OpenAI em um threading.Thread (tarefa de segundo plano) e retorna uma resposta imediata 202 Accepted para o frontend. A tarefa de segundo plano, ao ser concluída, atualiza o documento no Firestore com o cronograma completo e o status completed ou failed.
    - **Demais Rotas:** Mantém as rotas para gerar exercícios, dicas, corrigir discursivas, criar sessões de checkout do Stripe, gerenciar assinaturas (/create-portal-session) e excluir contas de usuário (/delete-user-account).
* **4.5. Configuração do Projeto e Deploy:**
  + .gitignore: Lista os arquivos e pastas que não devem ser enviados para o repositório do GitHub (ex: venv/, .env, serviceAccountKey.json).
  + .env, requirements.txt, build.sh, Procfile: Arquivos de configuração do projeto para o ambiente local e para o deploy no Render.