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

**Versão 1.0 - 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-8a639e1bed3825304733bd4ec7746cc5cfe6fa11

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 "feat: Descrição da funcionalidade feita"

*(Use feat: para novas funcionalidades, fix: para correções, chore: para configurações).*

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 três serviços principais: Firebase, Stripe e OpenAI.

**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. Isso é seguro, pois a proteção dos dados é feita pelas **Regras de Segurança** 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, 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):**

* **Configuração no Painel da Stripe:**
  + **Produtos e Preços:** Criamos nossos produtos ("Plano Mensal", "Plano Anual") e seus respectivos preços. Copiamos os **IDs dos Preços** (ex: price\_...) para usar em nosso código.
  + **Trial via Código:** Em vez de criar um preço separado para o "trial", nós aplicamos um período de teste de 7 dias via código no app.py quando o plano é 'trial'.
* **Stripe CLI e Webhooks:**
  + Instalamos a **Stripe CLI** para criar um "túnel" para nosso ambiente de desenvolvimento.
  + Com o comando stripe listen --forward-to localhost:5000/stripe-webhook, obtivemos um **segredo de webhook de teste** (whsec\_...), que foi colocado no nosso arquivo .env local.
  + Para produção, criamos um **Endpoint de Webhook** no painel da Stripe, apontando para https://iaprovas-backend.onrender.com/stripe-webhook. O segredo de produção gerado foi adicionado como variável de ambiente na Render.

**2.3. OpenAI (Inteligência Artificial):**

* **Controle de Custos:** Configuramos um **limite de uso mensal** (budget) em **Settings > Limits** no painel da OpenAI para receber alertas e controlar os gastos. Os **Rate Limits** (limites de velocidade) padrão foram mantidos como proteção automática contra picos de uso.

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

Colocamos a aplicação no ar usando a plataforma Render.

**3.1. Arquivos de Configuração para Deploy:** Criamos três arquivos na raiz do projeto:

* **requirements.txt:** Lista todas as bibliotecas Python que o backend precisa (Flask, Stripe, OpenAI, Gunicorn, etc.).
* **build.sh:** Um script que diz à Render para executar pip install -r requirements.txt.
* **Procfile:** Um arquivo que diz à Render como iniciar o servidor: web: gunicorn app:app.

**3.2. Deploy do Backend (Web Service):**

1. Criamos um "Web Service" na Render, conectado ao nosso GitHub.
2. **Configurações:** Runtime: Python 3, Build Command: bash build.sh, Start Command: gunicorn app:app.
3. **Variáveis de Ambiente:** Em vez de usar arquivos de segredos, configuramos um **Environment Group** na Render onde adicionamos todas as nossas chaves: OPENAI\_API\_KEY, STRIPE\_SECRET\_KEY (de produção), STRIPE\_WEBHOOK\_SECRET (de produção) e o conteúdo do serviceAccountKey.json como um "Secret File".

**3.3. Deploy do Frontend (Static Site):**

1. Criamos um "Static Site" na Render, também conectado ao GitHub.
2. **Configurações:** Publish Directory: . (ponto), para publicar todos os nossos arquivos HTML/CSS/JS.

**3.4. Configuração do Domínio:**

1. **Na Render:** Em "Settings" do nosso Static Site (iaprovas-frontend), adicionamos iaprovas.com.br e www.iaprovas.com.br como "Custom Domains". A Render nos forneceu os valores de **IP (para o registro A)** e **CNAME (para o www)**.
2. **No Registro.br:** Fomos na "Configuração de Zona DNS" e adicionamos os registros A e CNAME fornecidos pela Render.

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

Aqui está um mapa completo de todos os arquivos do projeto e a responsabilidade de cada um.

**4.1. Arquivos HTML (A Estrutura e o Conteúdo das Páginas)**

Estes são os "esqueletos" de cada página que o usuário vê.

* **index.html**: A página inicial e de marketing da plataforma, onde novos visitantes conhecem o serviço e os planos.
* **login.html**: A página onde os usuários existentes entram na plataforma.
* **cadastro.html**: A página onde novos usuários criam suas contas.
* **home.html**: O painel principal (dashboard) que o usuário vê após o login, com as métricas de estudo.
* **Páginas de Funcionalidades**: cronograma.html, exercicios.html, discursivas.html, dicas-estrategicas.html, meu-perfil.html. Cada um é a base para uma ferramenta específica da plataforma.
* **Páginas Legais e de Pagamento**: termos.html e privacidade.html (páginas legais), contratacao.html, sucesso.html e cancelado.html (partes do fluxo de assinatura).

**4.2. Arquivos de Estilo CSS (A Aparência Visual)**

Estes arquivos definem as cores, fontes e o layout das páginas HTML.

* **styles.css**: O arquivo de estilo principal para as páginas públicas (inicial, login, cadastro).
* **dashboard-lovable.css**: O arquivo de estilo específico para o painel do usuário logado, garantindo uma aparência consistente e profissional em todas as ferramentas internas.

**4.3. Arquivos JavaScript (A Interatividade e a Lógica)**

Estes são os "cérebros" do nosso site, que o tornam dinâmico e funcional.

* **Lógica Central e de Comunicação:**
  + **main-app.js**: O coração da aplicação quando o usuário está logado. Ele gerencia o estado de autenticação, carrega os dados do usuário do Firestore, calcula as métricas do dashboard e controla o acesso (verificação de trial).
  + **auth.js**: Responsável por todo o processo de autenticação: gerencia os formulários de login e cadastro, cria novos usuários e os redireciona pela plataforma.
  + **api.js**: Funciona como o "carteiro" da aplicação. Sua única função é fazer as chamadas para o nosso backend em Python, enviando pedidos e recebendo as respostas da IA.
* **Lógica Específica de Cada Página:**
  + cronograma-page.js, exercicios-page.js, discursivas-page.js, dicas-page.js, perfil-page.js, contratacao.js: Cada um desses arquivos contém a lógica interativa para sua respectiva página HTML. Eles capturam os dados dos formulários, chamam o api.js e exibem os resultados na tela.
  + **Perfil-page.js:** Este arquivo também é responsável por exibir o status da assinatura do usuário (ex: "Premium") e fornecer um link para o portal de gerenciamento de assinaturas da Stripe.
  + **cronograma-page.js**: Eles capturam os dados dos formulários, chamam o api.js e exibem os resultados na tela. Especificamente, este arquivo é responsável por renderizar o cronograma em um formato de tabela semanal e contém a lógica para exportar o plano de estudos exibido para um arquivo Excel (.xlsx).
* **Configuração:**
  + **firebase-config.js**: Inicializa a conexão com os serviços do Firebase (banco de dados e autenticação) usando as chaves públicas do projeto.
  + **stripe-config.js**: Armazena a chave publicável da Stripe, necessária para o frontend iniciar o processo de checkout.

**4.4. Backend e Arquivos de Servidor**

* **app.py**: O motor da nossa plataforma. É um servidor web construído com Flask (Python) que define todas as nossas rotas de API (ex: /gerar-plano-estudos, /stripe-webhook). Ele recebe as requisições do frontend, processa os dados, se comunica com as APIs da OpenAI e da Stripe, e envia as respostas de volta.

**4.5. Configuração do Projeto e Deploy**

* **requirements.txt**: A "lista de compras" do nosso backend. Define todas as bibliotecas Python que precisam ser instaladas para que o app.py funcione (Flask, OpenAI, Stripe, etc.).
* **.env**: Um arquivo local (que não vai para o GitHub) onde guardamos nossas chaves secretas durante o desenvolvimento no seu computador.
* **.gitignore**: Um arquivo de regras que diz ao Git quais arquivos e pastas ignorar (como .env e serviceAccountKey.json), protegendo nossas chaves secretas e mantendo o repositório limpo.
* **build.sh** e **Procfile**: São "manuais de instrução" para a plataforma de hospedagem Render, dizendo a ela como instalar (build.sh) e como iniciar (Procfile) nosso backend em Python.