Análise das Variáveis de Ambiente e Requisitos do Workflow

Este documento detalha a análise das variáveis de ambiente fornecidas e os requisitos implícitos para a construção de um fluxo de trabalho otimizado e seguro, integrando diversas plataformas e serviços.

1. Variáveis de Ambiente Fornecidas

As seguintes variáveis de ambiente foram fornecidas, categorizadas pela sua função principal:

1.1. Variáveis de Ambiente Frontend

- **`VITE_API_URL`**: `https://crset-backend-prod.railway.app`
    - **Propósito:** Define o endpoint da API de backend que o frontend irá consumir. Indica que o backend está alojado na Railway.app.
    - **Tipo:** URL de serviço.
    - **Sensibilidade:** Baixa (URL público).
- **`VITE_STRIPE_PUBLISHABLE_KEY`**: `pk_live_xxxxxxxxxxxxxxxx`
    - **Propósito:** Chave pública do Stripe, utilizada no frontend para inicializar o Stripe.js e criar tokens de pagamento. Esta chave é segura para ser exposta no lado do cliente.
    - **Tipo:** Chave de API pública.
    - **Sensibilidade:** Baixa (pode ser exposta no frontend).

1.2. Variáveis de Ambiente Backend

- **`STRIPE_SECRET_KEY`**: `sk_live_xxxxxxxxxxxxxxxx`

    - **Propósito:** Chave secreta do Stripe, utilizada no backend para processar transações, gerir subscrições, reembolsos e interagir com a API do Stripe de forma segura. **Esta chave é altamente sensível e nunca deve ser exposta no frontend.**
    - **Tipo:** Chave de API secreta.
    - **Sensibilidade:** Alta (deve ser mantida em segredo no backend).
- `JWT_SECRET`: `crset-super-secret-key`
    - **Propósito:** Chave secreta utilizada para assinar e verificar JSON Web Tokens (JWTs). Essencial para a segurança da autenticação e autorização no backend (e.g., NextAuth.js, Supabase Auth).
    - **Tipo:** Chave secreta.
    - **Sensibilidade:** Alta (deve ser mantida em segredo no backend).
- `SUPABASE_URL`: `https://xxxxxxxx.supabase.co`
    - **Propósito:** URL da instância do Supabase. Aponta para o serviço de base de dados e BaaS (Backend as a Service).
    - **Tipo:** URL de serviço.
    - **Sensibilidade:** Baixa (URL público).
- `SUPABASE_ANON_KEY`: `xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx`
    - **Propósito:** Chave pública (anon) do Supabase, utilizada para acesso inicial à base de dados e serviços do Supabase antes da autenticação do utilizador. Pode ser exposta no frontend, mas o acesso é limitado por Row Level Security (RLS).
    - **Tipo:** Chave de API pública.
    - **Sensibilidade:** Média (pode ser exposta no frontend, mas depende do RLS para segurança).
- `OPENAI_API_KEY`: `sk-crset-agent-mikexxxxxxxxxxxxxxxxxxxx`
    - **Propósito:** Chave de API para aceder aos serviços da OpenAI (e.g., GPT-4 para assistente IA). **Esta chave é altamente sensível e nunca deve ser exposta no frontend.**

    - **Tipo:** Chave de API secreta.
    - **Sensibilidade:** Alta (deve ser mantida em segredo no backend).

- `EMAILJS_SERVICE_ID`, `EMAILJS_TEMPLATE_ID`, `EMAILJS_PUBLIC_KEY`: `# (Cancelado)`
    - **Propósito:** Indicam que a integração com EmailJS foi cancelada ou não será utilizada neste workflow. Portanto, não serão consideradas para o fluxo de trabalho de comunicação por e-mail.
    - **Tipo:** N/A (cancelado).
    - **Sensibilidade:** N/A.

- `WHATSAPP_NUMBER`: `+351914423688`
    - **Propósito:** Número de telefone para comunicação via WhatsApp. Implica uma necessidade de integração de comunicação via WhatsApp, possivelmente para notificações ou suporte.
    - **Tipo:** Informação de contacto.
    - **Sensibilidade:** Baixa (número de contacto público).

- `FRONTEND_URL`: `https://crsetsolutions.com`
    - **Propósito:** URL da aplicação frontend em produção. Essencial para configurações de CORS, redirecionamentos de autenticação (e.g., NextAuth.js) e para referências em comunicações.
    - **Tipo:** URL de serviço.
    - **Sensibilidade:** Baixa (URL público).

2. Requisitos Implícitos do Workflow

Com base nas variáveis de ambiente fornecidas, os seguintes requisitos para o workflow podem ser inferidos:

2.1. Arquitetura e Deploy

- **Frontend:** Aplicação web (provavelmente Next.js ou React) que consome uma API de backend e interage com o Stripe para pagamentos.

- **Backend:** Uma API alojada na Railway.app que serve o frontend, processa pagamentos via Stripe, interage com o Supabase para base de dados e autenticação, e utiliza a OpenAI para funcionalidades de IA.
- **Deploy:** O frontend será provavelmente alojado em plataformas como Vercel ou Netlify, enquanto o backend já está na Railway.app. O deploy deve ser contínuo e automatizado.

2.2. Gestão de Dados e Autenticação

- **Base de Dados:** PostgreSQL gerido pelo Supabase, com a necessidade de configurar Row Level Security (RLS) para proteger os dados acedidos via `SUPABASE_ANON_KEY`.
- **Autenticação:** Sistema de autenticação baseado em JWT, provavelmente utilizando NextAuth.js ou similar, que se integra com o Supabase para gestão de utilizadores.

2.3. Funcionalidades Chave

- **Pagamentos:** Implementação de um fluxo de pagamentos seguro com Stripe, onde o frontend recolhe os dados do cartão (via `VITE_STRIPE_PUBLISHABLE_KEY`) e o backend processa a transação (via `STRIPE_SECRET_KEY`).
- **Inteligência Artificial:** Integração de funcionalidades de IA (e.g., chatbots, geração de conteúdo) no backend, utilizando a `OPENAI_API_KEY`.
- **Comunicação:** Necessidade de integração de comunicação via WhatsApp, possivelmente para notificações, alertas ou suporte ao cliente.

2.4. Segurança e Boas Práticas

- **Proteção de Segredos:** Todas as chaves secretas (`STRIPE_SECRET_KEY`, `JWT_SECRET`, `OPENAI_API_KEY`) devem ser armazenadas de forma segura como variáveis de ambiente nas plataformas de deploy (Railway.app, Vercel/Netlify) e nunca expostas no código fonte ou no frontend.
- **CORS:** Configuração adequada de Cross-Origin Resource Sharing (CORS) no backend para permitir requisições apenas do `FRONTEND_URL`.

- **HTTPS:** Assumir que todas as comunicações entre frontend e backend, e com serviços de terceiros, ocorrerão via HTTPS.
- **Princípio do Privilégio Mínimo:** Garantir que as permissões concedidas a cada chave ou token são as mínimas necessárias para a sua função.

3. Próximos Passos

Com esta análise, o próximo passo será desenhar a arquitetura detalhada do workflow, especificando como cada componente se integra e como as credenciais serão geridas em cada fase do ciclo de vida do desenvolvimento e deploy.

4. Arquitetura do Workflow Perfeito

Com base nas variáveis de ambiente e nos requisitos inferidos, a arquitetura do workflow perfeito será dividida em três camadas principais: Frontend, Backend e Serviços de Terceiros, interligadas por uma comunicação segura e eficiente. O objetivo é criar um sistema robusto, escalável e de fácil manutenção.

4.1. Visão Geral da Arquitetura

O workflow proposto adota uma arquitetura de microsserviços (ou pelo menos uma arquitetura distribuída) onde o frontend e o backend são desacoplados. O frontend, provavelmente uma Single Page Application (SPA) ou uma aplicação Next.js, interage com um backend centralizado que, por sua vez, comunica com diversos serviços de terceiros para funcionalidades específicas como base de dados, pagamentos e inteligência artificial. A comunicação entre os componentes é primariamente via HTTPS, garantindo a segurança dos dados em trânsito.

- **Uso:** Esta chave pública é utilizada no frontend para inicializar a biblioteca Stripe.js. Permite a recolha segura de informações de pagamento (como dados de cartão de crédito) diretamente do navegador do utilizador, tokenizando-as antes de as enviar para o backend. Os dados sensíveis do cartão nunca tocam nos servidores do frontend.
- **Melhores Práticas:**
    - **Exposição Segura:** É seguro expor esta chave no código do frontend, pois ela apenas permite a criação de tokens de pagamento e não a realização de

transações financeiras diretas.

- **Variável de Ambiente:** Deve ser injetada no frontend como uma variável de ambiente durante o processo de build/deploy (Vercel/Netlify).
- **PCI Compliance:** A utilização do Stripe.js ajuda a manter a conformidade com o PCI DSS (Payment Card Industry Data Security Standard), pois os dados sensíveis do cartão são manuseados diretamente pelo Stripe.

5.3. `STRIPE_SECRET_KEY` (Backend)

- **Uso:** Esta chave secreta é utilizada exclusivamente no backend para interagir com a API do Stripe para operações sensíveis, como criar cobranças, gerir subscrições, emitir reembolsos, verificar webhooks e aceder a informações detalhadas de clientes e transações.
- **Melhores Práticas:**
  - **Armazenamento Secreto: Nunca, em hipótese alguma, exponha esta chave no frontend ou a inclua no código fonte versionado (Git).** Deve ser armazenada como uma variável de ambiente secreta na plataforma de alojamento do backend (Railway.app).
  - **Princípio do Privilégio Mínimo:** A conta Stripe deve ter permissões restritas para esta chave, concedendo apenas o acesso necessário para as operações que o backend irá realizar.
  - **Rotação Regular:** Recomenda-se a rotação periódica desta chave para mitigar o risco de comprometimento.
  - **Webhooks:** Ao receber webhooks do Stripe, utilize o `Webhook Secret` (configurado no Stripe Dashboard e armazenado como variável de ambiente no backend) para verificar a autenticidade dos eventos, prevenindo ataques de falsificação.

5.4. `JWT_SECRET` (Backend)

- **Uso:** Esta chave secreta é fundamental para a segurança da autenticação e autorização. É usada pelo backend para assinar JSON Web Tokens (JWTs) quando um utilizador se autentica e para verificar a validade e integridade desses tokens em cada requisição subsequente a recursos protegidos.
- **Melhores Práticas:**

- **Armazenamento Secreto:** Tal como a `STRIPE_SECRET_KEY`, esta chave deve ser mantida **estritamente secreta** e armazenada como uma variável de ambiente na Railway.app. Nunca a inclua no código fonte.
- **Complexidade:** Utilize uma string longa, aleatória e complexa para o `JWT_SECRET` para dificultar ataques de força bruta.
- **Rotação:** Embora menos frequente que as chaves de API, a rotação periódica do `JWT_SECRET` pode ser considerada para aumentar a segurança, exigindo que os utilizadores se autentiquem novamente.
- **Expiração de Tokens:** Configure um tempo de expiração adequado para os JWTs para limitar a janela de oportunidade em caso de comprometimento.

5.5. `SUPABASE_URL` e `SUPABASE_ANON_KEY` (Backend e Frontend)

- **`SUPABASE_URL` (Backend e Frontend):**
  - **Uso:** O URL base para a sua instância do Supabase. É usado por ambas as camadas para se conectar aos serviços do Supabase (base de dados, autenticação, armazenamento).
  - **Melhores Práticas:** Pode ser exposto no frontend, pois é apenas um endereço de serviço.
- **`SUPABASE_ANON_KEY` (Frontend, com RLS; Backend para operações específicas):**
  - **Uso:** Esta é a

chave pública do Supabase. No frontend, é usada para inicializar o cliente Supabase e interagir com a base de dados e autenticação, mas o acesso é estritamente controlado por Row Level Security (RLS) configurado na base de dados. No backend, pode ser usada para operações que não exigem privilégios elevados, embora a `service_role` key (que é secreta) seja preferível para a maioria das operações de backend. * **Melhores Práticas:** * **RLS Essencial:** A segurança dos dados acedidos via `SUPABASE_ANON_KEY` depende **inteiramente** da correta configuração do Row Level Security (RLS) nas tabelas do Supabase. O RLS deve ser ativado e configurado para permitir que os utilizadores acedam apenas aos seus próprios dados ou a dados públicos. * **Não Expor `service_role` key:** A `service_role` key do Supabase (não fornecida nas variáveis de ambiente, mas existente) é uma chave secreta com

privilégios de superutilizador e **nunca deve ser exposta no frontend ou em qualquer código cliente**.

5.6. `OPENAI_API_KEY` (Backend)

- **Uso:** Esta chave secreta é utilizada no backend para autenticar e autorizar requisições à API da OpenAI, permitindo o acesso a modelos de linguagem (e.g., GPT-4) para funcionalidades de IA.
- **Melhores Práticas:**
    - **Armazenamento Secreto: Nunca exponha esta chave no frontend ou a inclua no código fonte versionado.** Deve ser armazenada como uma variável de ambiente secreta na plataforma de alojamento do backend (Railway.app).
    - **Princípio do Privilégio Mínimo:** Utilize esta chave apenas para as operações de IA necessárias. Se possível, considere a criação de chaves de API com permissões mais restritas na plataforma OpenAI, caso a sua arquitetura permita.
    - **Monitorização de Uso:** Monitorize o uso da API da OpenAI para detetar padrões incomuns ou potenciais abusos, pois o uso excessivo pode gerar custos significativos.

5.7. `WHATSAPP_NUMBER` (Backend/Configuração)

- **Uso:** Este número de telefone é o destino ou a origem das comunicações via WhatsApp. Será utilizado por um serviço de integração de WhatsApp (e.g., Twilio, Vonage, ou uma API de WhatsApp Business) para enviar mensagens programaticamente.
- **Melhores Práticas:**
    - **Armazenamento:** Pode ser armazenado como uma variável de ambiente no backend, pois não é uma credencial sensível por si só, mas faz parte da configuração de um serviço de comunicação.
    - **Gateway Seguro:** A integração com o WhatsApp deve ser feita através de um gateway oficial ou uma API de WhatsApp Business para garantir a conformidade e a segurança das mensagens.

5.8. `FRONTEND_URL` (Backend/Configuração)

- **Uso:** O URL público da aplicação frontend. É crucial para configurar as políticas de CORS no backend, para redirecionamentos de autenticação (e.g., após um login bem-sucedido) e para referências em e-mails ou outras comunicações.
- **Melhores Práticas:**
  - **Consistência:** Assegurar que este URL é consistente em todas as configurações (backend, provedores de autenticação, etc.).
  - **HTTPS:** Utilizar sempre `https://` para garantir a segurança das comunicações e redirecionamentos.

5.9. Credenciais Canceladas (EmailJS)

- **Uso:** As variáveis `EMAILJS_SERVICE_ID`, `EMAILJS_TEMPLATE_ID` e `EMAILJS_PUBLIC_KEY` foram marcadas como canceladas. Isto significa que o EmailJS não será utilizado para o envio de e-mails neste workflow. Caso a funcionalidade de e-mail seja necessária no futuro, uma alternativa (como SendGrid, Mailgun, ou um serviço de e-mail transacional via backend) deverá ser implementada.
- **Melhores Práticas:**
  - **Remoção:** Remover quaisquer referências a estas variáveis ou ao EmailJS no código para evitar confusão e garantir que não há tentativas de uso de credenciais desativadas.

6. Permissões Mínimas Necessárias por Serviço

Para cada serviço e credencial, é fundamental aplicar o princípio do privilégio mínimo, concedendo apenas as permissões estritamente necessárias para a sua operação. Isto reduz a superfície de ataque e minimiza o impacto de um potencial comprometimento de credenciais.

6.1. Stripe

- `VITE_STRIPE_PUBLISHABLE_KEY` (Frontend):
  - **Permissões:** Apenas para inicializar o Stripe.js e criar tokens de pagamento. Não deve ter permissão para criar cobranças ou aceder a dados

sensíveis de transações.

- `STRIPE_SECRET_KEY` **(Backend):**
  - **Permissões:** Acesso para criar e gerir clientes, criar e processar pagamentos (charges), gerir subscrições, emitir reembolsos e aceder a relatórios de transações. As permissões devem ser restritas ao mínimo necessário para as operações de negócio da aplicação.
  - **Webhooks:** O endpoint de webhook no backend deve ter permissão para receber e processar eventos específicos do Stripe (e.g., `checkout.session.completed`, `invoice.payment_succeeded`).

6.2. Supabase

- `SUPABASE_ANON_KEY` **(Frontend):**
  - **Permissões:** Acesso de leitura/escrita a tabelas específicas, conforme definido pelas políticas de Row Level Security (RLS). Por exemplo, um utilizador autenticado pode ter permissão para ler e escrever nos seus próprios registos, mas apenas ler dados públicos.

- **Backend (via `service_role` key ou credenciais de serviço):**
  - **Permissões:** Acesso total à base de dados (PostgreSQL) para operações de backend, como gestão de utilizadores, criação/atualização de dados sensíveis, e execução de funções de base de dados. Esta chave deve ser usada com extrema cautela e apenas em ambientes seguros.

6.3. OpenAI

- `OPENAI_API_KEY` **(Backend):**
  - **Permissões:** Acesso para fazer chamadas aos modelos de linguagem e/ou imagem específicos que a aplicação irá utilizar (e.g., `completions`, `chat/completions`). As permissões devem ser limitadas aos modelos e funcionalidades necessárias para evitar uso indevido ou custos excessivos.

6.4. Plataformas de Deploy (Vercel/Netlify, Railway.app)

- **Integração GitHub:** Permissão de leitura ao repositório GitHub para permitir o deploy contínuo.

- **Variáveis de Ambiente:** Permissão para configurar e gerir variáveis de ambiente (secrets) no painel de controlo da plataforma. Apenas utilizadores autorizados devem ter acesso a estas configurações.
- **Acesso a Logs:** Permissão para aceder aos logs da aplicação para monitorização e depuração.

Ao seguir estas diretrizes de uso e permissões mínimas, o workflow será mais seguro e resiliente a potenciais vulnerabilidades.

7. Plano de Implementação e Operação

Este plano descreve os passos práticos para implementar o workflow perfeito, desde a configuração inicial até à operação contínua e monitorização. O foco está na automação, segurança e escalabilidade.

7.1. Configuração Inicial dos Serviços

7.1.1. GitHub (Controlo de Versões)

1. **Criação de Repositórios:** Crie dois repositórios separados no GitHub: um para o frontend (Next.js/React) e outro para o backend (API na Railway.app).
2. **Configuração de Branches:** Adote uma estratégia de branching (e.g., Git Flow ou GitHub Flow) com branches `main` (produção) e `develop` (desenvolvimento).
3. **Proteção de Branches:** Configure regras de proteção para as branches `main` e `develop` para exigir revisões de código (pull requests) antes de qualquer merge.
4. **Segredos do GitHub Actions:** Para qualquer automação futura (e.g., testes, linting), utilize os segredos do GitHub para armazenar credenciais que possam ser necessárias durante os workflows de CI/CD, garantindo que nunca são expostas no código.

7.1.2. Supabase (Base de Dados e Autenticação)

1. **Criação de Projeto:** Crie um novo projeto no Supabase.
2. **Configuração da Base de Dados:**
    - Defina o esquema da base de dados (tabelas, colunas, relações) de acordo com os requisitos da aplicação.

    - Utilize migrações (e.g., via Prisma ORM ou Supabase CLI) para gerir as alterações no esquema da base de dados de forma controlada.
3. **Configuração de Autenticação:**
    - Ative os provedores de autenticação necessários (e.g., e-mail/password, Google OAuth).
    - Configure as URLs de redirecionamento para o `FRONTEND_URL` (`https://crsetsolutions.com`).
4. **Row Level Security (RLS): Essencial.** Implemente políticas de RLS em todas as tabelas para controlar o acesso aos dados. Isto garante que a `SUPABASE_ANON_KEY` (que é pública) só permita acesso autorizado e seguro aos dados.
5. **Obtenção de Credenciais:** Anote o `SUPABASE_URL` e a `SUPABASE_ANON_KEY` do painel de controlo do Supabase. Se necessário, gere uma `service_role` key para operações de backend com privilégios elevados (esta deve ser tratada como um segredo).

7.1.3. Stripe (Pagamentos)

1. **Criação de Conta:** Crie e configure a sua conta Stripe.
2. **Obtenção de Chaves API:** No painel de controlo do Stripe, obtenha a `Publishable Key` (para o frontend) e a `Secret Key` (para o backend).
3. **Configuração de Webhooks:** Crie um endpoint de webhook no Stripe que aponte para um URL público no seu backend (e.g., `https://crset-backend-prod.railway.app/stripe/webhook`). Configure o `Webhook Secret` e armazene-o como uma variável de ambiente no backend.
4. **Modo de Teste:** Utilize o modo de teste do Stripe para desenvolvimento e testes, usando chaves de teste e cartões de teste fornecidos pelo Stripe.

7.1.4. OpenAI (Inteligência Artificial)

1. **Criação de Conta:** Crie uma conta na OpenAI.
2. **Obtenção de Chave API:** Gere uma `API Key` no painel de controlo da OpenAI. Trate-a como um segredo de alta sensibilidade.
3. **Monitorização de Uso:** Configure alertas de uso e limites de gastos na OpenAI para evitar custos inesperados.

7.1.5. Gateway de WhatsApp (e.g., Twilio, Vonage, ou Solução Customizada)

1. **Seleção e Configuração:** Escolha um provedor de API de WhatsApp (se ainda não tiver um) e configure-o de acordo com a sua documentação. Isto geralmente envolve a verificação do seu número de telefone (`WHATSAPP_NUMBER`) e a obtenção de credenciais de API (Account SID, Auth Token, etc.).
2. **Integração no Backend:** Implemente a lógica de envio de mensagens no backend utilizando as credenciais do provedor de WhatsApp.

7.2. Configuração de Variáveis de Ambiente nas Plataformas de Deploy

7.2.1. Frontend (Vercel / Netlify)

1. **Conexão com GitHub:** Conecte o seu repositório GitHub do frontend à Vercel/Netlify.
2. **Configuração de Variáveis de Ambiente:** No painel de controlo da Vercel/Netlify, adicione as seguintes variáveis de ambiente para os ambientes de desenvolvimento e produção:
   - `VITE_API_URL`
   - `VITE_STRIPE_PUBLISHABLE_KEY`
   - `FRONTEND_URL`
3. **Build Command:** Configure o comando de build (e.g., `npm run build` ou `yarn build`).
4. **Output Directory:** Especifique o diretório de output da build (e.g., `dist` ou `.next`).

7.2.2. Backend (Railway.app)

1. **Conexão com GitHub:** Conecte o seu repositório GitHub do backend à Railway.app.
2. **Configuração de Variáveis de Ambiente:** No painel de controlo da Railway.app, adicione as seguintes variáveis de ambiente para os ambientes de desenvolvimento e produção. **Estas devem ser tratadas como segredos e nunca expostas no código:**

- `STRIPE_SECRET_KEY`
- `JWT_SECRET`
- `SUPABASE_URL`
- `SUPABASE_ANON_KEY` (embora pública, é boa prática mantê-la aqui para consistência)
- `OPENAI_API_KEY`
- `WHATSAPP_NUMBER`
- `FRONTEND_URL` (para configuração de CORS e redirecionamentos)
- `STRIPE_WEBHOOK_SECRET` (se aplicável)
- Credenciais do Gateway de WhatsApp (se aplicável)

3. **Build Command:** Configure o comando de build (e.g., `npm install && npm run build`).
4. **Start Command:** Configure o comando para iniciar a aplicação (e.g., `npm run start`).
5. **Porta:** Assegure que a aplicação backend está a escutar na porta correta (geralmente `PORT` fornecida pelo ambiente da Railway.app).

7.3. Processo de Deploy Contínuo (CI/CD)

1. **Integração GitHub:** Ambas as plataformas (Vercel/Netlify e Railway.app) devem estar integradas com os respetivos repositórios GitHub.
2. **Triggers de Deploy:** Configure para que um deploy automático seja acionado sempre que houver um push para a branch `main` (para produção) ou `develop` (para staging/desenvolvimento).
3. **Testes Automatizados:** Implemente testes unitários, de integração e end-to-end no seu código. Configure workflows no GitHub Actions para executar estes testes antes de permitir o merge para as branches protegidas. Isto garante que apenas código testado e funcional é implementado.
4. **Rollbacks:** Familiarize-se com as funcionalidades de rollback das plataformas de deploy. Em caso de problemas após um deploy, deve ser possível reverter rapidamente para uma versão anterior estável.

7.4. Monitorização e Resolução de Problemas

7.4.1. Monitorização

- **Logs:** Monitorize os logs da aplicação em ambas as plataformas (Vercel/Netlify para frontend, Railway.app para backend). Configure sistemas de agregação de logs (e.g., Logtail, Datadog) se a escala justificar.
- **Performance:** Utilize ferramentas de monitorização de performance de aplicações (APM) para acompanhar métricas como tempo de resposta, erros, uso de CPU/memória. As próprias plataformas de deploy oferecem dashboards básicos.
- **Health Checks:** Implemente endpoints de health check no backend (e.g., `/health`) que as plataformas de deploy podem usar para verificar a disponibilidade do serviço.
- **Alertas:** Configure alertas para erros críticos, picos de tráfego incomuns ou falhas de serviço, notificando a equipa responsável.

7.4.2. Resolução de Problemas

- **Logs Detalhados:** Em caso de erro, aceda aos logs detalhados para identificar a causa raiz. Utilize ferramentas de depuração (debuggers) no ambiente de desenvolvimento.
- **Ambientes de Staging:** Mantenha um ambiente de staging (desenvolvimento) que replique o ambiente de produção para testar correções antes de as implementar em produção.
- **Comunicação entre Serviços:** Verifique a conectividade e a comunicação entre o frontend, backend e serviços de terceiros. Erros de CORS, variáveis de ambiente incorretas ou chaves API inválidas são causas comuns.
- **Documentação:** Mantenha uma documentação atualizada sobre a arquitetura, configuração e procedimentos de resolução de problemas comuns.

Este plano de implementação e operação fornece um roteiro claro para construir e manter um workflow eficiente e seguro, aproveitando ao máximo as capacidades de cada plataforma.