Estratégia de CI/CD (Integração e Implantação Contínua) com GitHub Actions

A automação do ciclo de vida do desenvolvimento é fundamental para garantir a qualidade e a velocidade. Adotaremos uma estratégia de múltiplos workflows no GitHub Actions.

Princípios:

- **Automação Total:** Cada commit é um candidato a release, passando por um pipeline automatizado de validação.
- **Segurança no Pipeline:** Todos os segredos (credenciais de banco de dados, chaves de API, segredos de JWT) serão gerenciados via *GitHub Encrypted Secrets* e expostos ao pipeline apenas como variáveis de ambiente.
- **Consistência Ambiental:** Os ambientes de *Staging* e *Produção* serão provisionados com a mesma infraestrutura como código para garantir consistência.

1.1. Workflow de Validação (`on: pull_request`)

Este workflow atua como um portão de qualidade (quality gate) antes que o código seja mesclado à branch principal (`main`).

- **Gatilho:** Abertura ou atualização de um *Pull Request* para a branch `main` .
- **Objetivo:** Validar a integridade, qualidade e correção do código.
- **Passos:**
    1. **Checkout & Setup:** Configuração dos ambientes Java 21 e Node.js.
    2. **Validação do Backend:**
        - Executar testes unitários e de integração (`mvn test`). Os testes de integração utilizarão Testcontainers para instanciar um banco de dados PostgreSQL real, garantindo a compatibilidade.

    - Executar testes de arquitetura (`mvn verify`) com ArchUnit para garantir que as dependências entre camadas não foram violadas.
    - Compilar o artefato da aplicação (`mvn package`).
3. **Validação do Frontend:**
    - Instalar dependências (`npm install`).
    - Executar Linter e Formatter (`npm run lint`) para garantir a consistência do código.
    - Executar testes de componentes e integração (`npm run test`) com Vitest e React Testing Library.
    - Compilar os ativos estáticos (`npm run build`).

- **Regra de Proteção:** A mesclagem do *Pull Request* será bloqueada se qualquer um dos passos acima falhar.

1.2. Workflow de Implantação para Staging (`on: push, branches: [main]`)

Este workflow implanta a versão mais recente do código em um ambiente de homologação.

- **Gatilho:** Mesclagem de código na branch `main`.
- **Objetivo:** Disponibilizar uma versão funcional para testes de ponta a ponta (E2E) e validação de stakeholders.
- **Passos:**
    1. **Executar Validação:** Re-executar todos os passos do workflow de validação.
    2. **Build & Push da Imagem Docker:** Construir uma imagem Docker da aplicação backend e enviá-la para um registro de contêineres (ex: GitHub Container Registry).
    3. **Implantação no Render:** Acionar um *Deploy Hook* do Render via `curl`. O Render irá automaticamente baixar a imagem mais recente e reiniciar o serviço de Staging com zero downtime. As migrações de banco de dados do Flyway serão aplicadas automaticamente na inicialização da aplicação.
    4. **Executar Testes E2E:** Após a implantação, um job separado iniciará os testes de ponta a ponta (com Playwright) contra o ambiente de Staging.

5. **Notificação:** Enviar uma notificação para um canal do Slack informando o sucesso da implantação.

1.3. Workflow de Implantação para Produção (`on: release, types: [published]`)

Este workflow promove uma versão estável e testada para o ambiente de produção.

- **Gatilho:** Criação de uma nova *Release* no GitHub (ex: `v1.2.0`).
- **Objetivo:** Implantar uma nova versão para os usuários finais de forma controlada e segura.
- **Passos:**
  1. **Aprovação Manual:** O workflow será pausado aguardando a aprovação de um revisor definido (ex: Tech Lead), utilizando o ambiente de `production` do GitHub Actions, que exige aprovação.
  2. **Implantação no Render:** Após a aprovação, o *Deploy Hook* do serviço de Produção do Render é acionado.
  3. **Verificação de Saúde:** Um job de *smoke test* verifica se os principais endpoints da aplicação estão respondendo com status `200 OK` após a implantação.
  4. **Notificação:** Enviar uma notificação de alta prioridade sobre a implantação em produção.

2. Estratégia de Observabilidade e Monitoramento

A observabilidade é construída sobre três pilares: Logs, Métricas e Traces.

2.1. Dashboards

Criaremos dashboards específicos para monitorar diferentes aspectos do sistema em uma ferramenta como Datadog ou Grafana.

- **Dashboard de Saúde da API:**
  - **Métricas:** Taxa de requisições (RPM), taxa de erros (separada por 4xx e 5xx), latência (percentis p50, p90, p99), saúde da JVM (uso de heap, CPU).
  - **Objetivo:** Monitorar a saúde técnica e a performance da aplicação.

- **Dashboard de Uso de IA:**
  - **Métricas:** Latência das chamadas à API de IA por provedor, taxa de erros, tokens de entrada/saída por modelo, status dos *Circuit Breakers*.
  - **Objetivo:** Controlar custos, performance e a fiabilidade dos serviços de IA de terceiros.
- **Dashboard de Métricas de Negócio:**
  - **Métricas:** Novos cadastros, número de projetos criados, execuções do fluxo "proposta e aprovação" da IA, usuários ativos diários (DAU).
  - **Objetivo:** Fornecer visibilidade sobre a utilização do produto.

2.2. Estratégia de Alertas

- **P1 - Alertas Críticos (Notificação imediata para a equipe de plantão):**
  - Taxa de erros 5xx > 5% por mais de 5 minutos.
  - Latência p99 da API > 2 segundos.
  - Aplicação fora do ar (falha no health check).
- **P2 - Alertas de Atenção (Notificação em canal de Slack):**
  - Uso de Heap da JVM > 80%.
  - Pico incomum de erros 4xx (pode indicar um bug no cliente ou um ataque).
  - *Circuit Breaker* para um provedor de IA aberto.

3. Gestão de Migrações e Seeding de Dados

3.1. Migração de Schema com Flyway

A evolução do schema do banco de dados será gerenciada exclusivamente pelo Flyway. Todos os scripts DDL (Data Definition Language) serão versionados em

`src/main/resources/db/migration`, garantindo um processo de migração automatizado e repetível.

3.2. Estratégia de Seeding de Dados

O uso de `CommandLineRunner` é limitado. Adotaremos uma abordagem mais robusta:

- **Dados de Referência:** Para dados que devem existir em todos os ambientes (ex: planos de subscrição padrão), usaremos migrações repetíveis do Flyway (`R__*.sql`).
- **Dados de Teste (Ambientes não produtivos):** Utilizaremos um componente Spring ativado por perfil (`@Profile("!prod")`) para inserir dados de teste (usuários, workspaces) apenas em ambientes de desenvolvimento e staging.

4. Fluxos de Gestão de Usuários

As operações de gerenciamento de membros do Workspace serão expostas via API REST.

4.1. Convidar Membro para um Workspace

- **Endpoint:** `POST /api/v1/workspaces/{workspaceId}/invitations`
- **Corpo da Requisição:** `{ "email": "string", "role": "MEMBER" | "ADMIN" }`
- **Lógica do Caso de Uso:**
  1. O serviço de aplicação verifica se o usuário solicitante tem permissão (`OWNER` ou `ADMIN`) no workspace.
  2. Verifica se o e-mail já pertence a um membro para evitar duplicatas.
  3. Cria um registro de convite com um token seguro e data de expiração.
  4. Dispara o envio de um e-mail de convite contendo um link para aceitação.
- **Resposta:** `201 Created`

4.2. Aceitar Convite para um Workspace

- **Endpoint:** `POST /api/v1/invitations/accept`
- **Corpo da Requisição:** `{ "token": "string" }`
- **Lógica do Caso de Uso:**
  1. O usuário deve estar autenticado para aceitar um convite.
  2. O serviço valida o token (existência e data de expiração).
  3. Verifica se o e-mail do usuário autenticado corresponde ao e-mail do convite.

4. Adiciona o usuário à lista de membros do workspace com a função definida no convite.
5. Invalida o token do convite.

- **Resposta:** `200 OK`

4.3. Remover Membro de um Workspace

- **Endpoint:** `DELETE /api/v1/workspaces/{workspaceId}/members/{memberId}`
- **Lógica do Caso de Uso:**
  1. O serviço de aplicação verifica as permissões do solicitante. Um `ADMIN` não pode remover um `OWNER`, e o `OWNER` não pode ser removido se for o único membro com essa função .
  2. Invoca o método `workspace.removeMember(memberId)` no objeto de domínio, que contém as regras de negócio.
  3. Persiste o estado atualizado do agregado `Workspace`.
- **Resposta:** `204 No Content`

Ensino

Esta documentação demonstra que uma arquitetura de software robusta transcende o código e os padrões de design, estendendo-se para a **operacionalidade do sistema**. Tópicos como CI/CD, observabilidade e gestão de dados não são "adicionais", mas sim componentes integrais da arquitetura que garantem que o valor de negócio do software possa ser entregue de forma fiável, segura e contínua. Automatizar o caminho para a produção (CI/CD) e instrumentar o sistema para "perguntar" sobre seu estado (observabilidade) são investimentos que reduzem o risco e aumentam a velocidade de iteração a longo prazo.