📝 Relatório de Progresso – Desafio Loomi

Desenvolvedora: Elany Peixoto
Repositório: github.com/develany/loomi-bank-microservices
Trello: https://trello.com/invite/b/68f6a00ae5a8228a9e1d57f9/ATTI2ec22c9c77120ff00bf2ca9726b1c2f1D0A7D2DE/desafio-loomi

🔗 Plataforma de gestão de atividades

Utilizei o **Trello** (link do board) para organizar e acompanhar o progresso das tarefas do desafio.
O board foi estruturado nas seguintes colunas:

- **Backlog:** escopo e ideias iniciais do desafio;
- **To Do:** tarefas priorizadas para início;
- **In Progress:** itens em desenvolvimento ativo;
- **Review/Testes:** código em fase de validação e ajustes;
- **Done:** entregas concluídas e testadas.

⚙️ Organização das demandas e atividades

As atividades foram divididas entre os dois microsserviços propostos:

- **users-service:** responsável pela gestão de clientes e dados bancários.
- **transactions-service:** responsável por gerenciar transferências entre usuários.

Cada microsserviço teve tarefas específicas:

1. Estrutura inicial (NestJS, Docker, PostgreSQL);
2. Implementação dos endpoints mínimos;
3. Comunicação via RabbitMQ;
4. Testes unitários e de integração;
5. Documentação (Swagger e README);
6. Boas práticas de segurança e validação.

🧭 Prioridade das entregas

A priorização foi feita com base em dependências técnicas:

1. **Setup do ambiente** com Docker Compose, PostgreSQL e RabbitMQ;
2. **Users Service** (base para validação de transações);

3. **Transactions Service** (integração com users-service);
4. **Comunicação assíncrona** via mensageria (RabbitMQ);
5. **Testes e documentação**;
6. **Revisões e ajustes finais de código.**

🚧 Principais dificuldades enfrentadas

- Ajuste da **comunicação entre microsserviços via RabbitMQ** (filas, mensagens e serialização).
- **Sincronização entre containers** no Docker Compose.
- **Configuração do TypeORM** com múltiplas conexões PostgreSQL.
- **Limitação de tempo**, priorizando entregas estáveis e código limpo.

💡 O que faria diferente com mais tempo

- Implementaria **autenticação e autorização completas** via JWT.
- Criaria uma **API Gateway** para centralizar chamadas entre serviços.
- Adicionaria **testes end-to-end** (Jest + Supertest).
- Automatizaria o **deploy na AWS (EC2 + GitHub Actions)**.
- Integraria **monitoramento e logs estruturados** (ex: Prometheus, Datadog).

🧩 Fluxo de Git

- **Branch principal:** `develop`
- **Estrutura de branches:**
  - `feature/users-service`
  - `feature/transactions-service`
  - `feature/rabbitmq-integration`
  - `feature/tests`
- Utilização do **Gitflow**, com commits descritivos e Pull Requests revisados antes do merge.

🤖 Uso de Ferramentas de IA

O uso de IA foi feito de forma estratégica, com foco em produtividade, qualidade e revisão técnica.

- **ChatGPT (OpenAI):**
  Utilizado para estruturar a base dos microsserviços, definir padrões de comunicação

entre serviços, ajustar configurações do Docker Compose e refinar o código conforme boas práticas do NestJS.

- **Claude (Anthropic):**
  Aplicado na **geração e revisão de código**, especialmente para refatorar trechos complexos, revisar padrões de design e sugerir melhorias de legibilidade e modularidade.
- **GitHub Copilot:**
  Usado para **autocompletar funções simples**, validações e auxiliares de configuração.

➡️ Todo o código sugerido por IA foi **revisado manualmente**, garantindo **autoria, autenticidade e coerência** com o padrão do projeto.