🚀 Referral System Vortex

Sistema de indicação desenvolvido como desafio técnico para seleção do Vortex Unifor. Uma plataforma completa que permite aos usuários se cadastrarem, indicarem outros usuários através de códigos de referência e acumularem pontos por cada indicação realizada.

📋 Índice

📖 Sobre o Projeto

O **Referral System Vortex** é uma aplicação full-stack que implementa um sistema de indicação (referral) completo. O projeto foi desenvolvido utilizando uma arquitetura monorepo com Turborepo, permitindo o gerenciamento eficiente de múltiplos pacotes e aplicações em um único repositório.

O que o sistema faz?

- **Cadastro de Usuários**: Permite que novos usuários se registrem na plataforma
- **Sistema de Indicação**: Cada usuário recebe um código único de referência
- **Pontuação**: Usuários ganham pontos quando outros se cadastram usando seu código
- **Autenticação**: Sistema seguro de login com JWT
- **Validação de Dados**: Validação robusta tanto no frontend quanto no backend

✨ Funcionalidades

🔐 Autenticação e Autorização

- Cadastro de novos usuários com validação de dados
- Login seguro com geração de token JWT
- Middleware de autenticação para rotas protegidas
- Criptografia de senhas com bcrypt

🎯 Sistema de Referência

- Geração automática de código único de referência para cada usuário
- Cadastro através de código de indicação (opcional)
- Sistema de pontuação para quem indica
- Validação de códigos de referência

📊 Perfil do Usuário

- Visualização de informações do usuário autenticado
- Exibição da pontuação acumulada
- Acesso ao código pessoal de referência

🎨 Interface do Usuário

- Interface moderna e responsiva
- Formulários de cadastro e login intuitivos
- Validação de formulários em tempo real
- Feedback visual de erros e sucessos

🛠 Tecnologias Utilizadas

Backend (API)

Node.js + TypeScript

Escolhi Node.js com TypeScript para garantir type-safety e melhor manutenibilidade do código. TypeScript previne erros em tempo de desenvolvimento e melhora a experiência do desenvolvedor com autocompletion.

Express.js

Framework minimalista e flexível para criação da API REST. Sua simplicidade e vasta comunidade tornam o desenvolvimento mais ágil.

Prisma ORM

ORM moderno que facilita interações com o banco de dados. Escolhi Prisma por:

- Type-safety automático
- Migrations simplificadas
- Excelente developer experience
- Query builder intuitivo

PostgreSQL

Banco de dados relacional robusto e confiável, ideal para um sistema que requer integridade de dados e relacionamentos complexos.

JWT (jsonwebtoken)

Para autenticação stateless e segura, permitindo escalabilidade horizontal da aplicação.

Bcrypt

Biblioteca para hash de senhas, garantindo que as credenciais dos usuários sejam armazenadas de forma segura.

Zod

Biblioteca de validação e parsing de schemas TypeScript-first, garantindo validação consistente de dados.

Frontend (Web)

React 19

Biblioteca JavaScript moderna para construção de interfaces. React oferece:

- Componentização eficiente
- Virtual DOM para performance
- Vasto ecossistema de bibliotecas
- Comunidade ativa

TypeScript

Garante type-safety também no frontend, reduzindo bugs e melhorando a manutenibilidade.

Vite

Build tool extremamente rápido que melhora significativamente a experiência de desenvolvimento com Hot Module Replacement (HMR) instantâneo.

React Router

Solução de roteamento para Single Page Applications (SPA), permitindo navegação fluida entre páginas.

Arquitetura e Ferramentas

Turborepo

Gerenciador de monorepo que otimiza builds e execução de tarefas. Escolhi Turborepo por:

- Cache inteligente de builds
- Execução paralela de tarefas
- Configuração simples
- Excelente performance

pnpm

Gerenciador de pacotes eficiente que economiza espaço em disco e acelera instalações através de hard links.

Monorepo Structure

Organizei o projeto em monorepo para:

- **apps/api**: Backend da aplicação
- **apps/web**: Frontend da aplicação
- **packages/schemas**: Schemas de validação compartilhados (Zod)

- **packages/tsconfig**: Configurações TypeScript compartilhadas

Esta estrutura permite:

- Reutilização de código entre projetos
- Versionamento unificado
- Deploys independentes
- Melhor organização do código

🏗️ Arquitetura do Projeto

O projeto segue uma arquitetura em camadas no backend:

```
API (Backend)
├── Controllers     → Manipulação de requisições HTTP
├── Services        → Lógica de negócio
├── Repository      → Acesso aos dados (Prisma)
├── Middlewares     → Autenticação e tratamento de erros
├── Types           → Definições de tipos TypeScript
└── Prisma          → Schema e migrations do banco
```

O frontend segue uma estrutura componentizada:

```
Web (Frontend)
├── Components      → Componentes reutilizáveis
├── Pages           → Páginas da aplicação
├── Routes          → Configuração de rotas
├── Helpers         → Funções auxiliares
├── Types           → Definições de tipos
└── Styles          → Estilos CSS
```

📦 Pré-requisitos

Antes de começar, você precisará ter instalado em sua máquina:

- **Node.js** (versão 18 ou superior)
- **pnpm** (versão 10 ou superior)
- **PostgreSQL** (versão 12 ou superior)
- **Git**

Instalando o pnpm

```
npm install -g pnpm@10.0.0
```

🚀 Como Executar o Projeto

1. Clone o repositório

```
git clone https://github.com/seu-usuario/Refarral-System-Vortex.git
cd Refarral-System-Vortex
```

2. Instale as dependências

```
pnpm install
```

3. Configure as variáveis de ambiente

Crie um arquivo `.env` na pasta `apps/api/` com as seguintes variáveis:

```
DATABASE_URL="postgresql://usuario:senha@localhost:5432/referral_db"
JWT_SECRET="seu_secret_jwt_aqui"
PORT=3000
```

Importante: Substitua `usuario`, `senha` e configure sua string de conexão do PostgreSQL.

4. Configure o banco de dados

Execute as migrations do Prisma:

```
cd apps/api
pnpm run setup
```

Este comando irá:

- Gerar o Prisma Client
- Executar as migrations no banco de dados

5. Execute o projeto em modo de desenvolvimento

Volte para a raiz do projeto e execute:

```
cd ../..
pnpm dev
```

Este comando irá iniciar simultaneamente:

- **API (Backend)**: http://localhost:3000
- **Web (Frontend)**: http://localhost:5173

6. Acesse a aplicação

Abra seu navegador e acesse:

- **Frontend**: http://localhost:5173
- **API**: http://localhost:3000

Comandos Úteis

Modo de desenvolvimento (raiz do projeto)

```
pnpm dev          # Inicia API e Web em modo desenvolvimento
```

API (apps/api)

```
pnpm dev                    # Inicia a API em modo watch
pnpm run prisma:studio      # Abre o Prisma Studio (GUI do banco)
pnpm run prisma:generate    # Gera o Prisma Client
pnpm run prisma:migrate:dev # Executa novas migrations
pnpm build                  # Compila o TypeScript
```

Web (apps/web)

```
pnpm dev      # Inicia o servidor de desenvolvimento
pnpm build    # Cria build de produção
pnpm preview  # Preview do build de produção
pnpm lint     # Executa o linter
```

🤖 Colaboração com IA

Como Usei Ferramentas de IA

Durante o desenvolvimento deste projeto, utilizei intensivamente o **GitHub Copilot** e o **ChatGPT** como assistentes de desenvolvimento. Aqui está um detalhamento de como essas ferramentas foram aplicadas:

1. Arquitetura e Estruturação do Projeto

- **O que fiz**: Consultei a IA para decidir sobre a melhor estrutura de monorepo
- **Como ajudou**: A IA sugeriu o uso de Turborepo e pnpm, explicando os benefícios de cada ferramenta
- **Resultado**: Consegui implementar uma arquitetura escalável e organizada desde o início

2. Implementação de Padrões de Projeto

- **O que fiz**: Solicitei sugestões sobre padrões para organizar o backend

- **Como ajudou**: A IA recomendou a separação em camadas (Controller → Service → Repository)
- **Aprendizado**: Entendi melhor os princípios SOLID e separação de responsabilidades

3. Geração de Código Boilerplate

- **O que fiz**: Usei Copilot para gerar código repetitivo
- **Partes do projeto**:
  - Estrutura inicial de controllers e services
  - Configuração do Prisma Schema
  - Setup de middlewares
  - Tipos TypeScript
- **Como ajudou**: Economizou tempo significativo em tarefas repetitivas
- **Aprendizado**: Aprendi a revisar e adaptar o código gerado, não apenas aceitá-lo cegamente

4. Validação com Zod

- **O que fiz**: Pedi exemplos de schemas de validação com Zod
- **Como ajudou**: A IA forneceu patterns robustos de validação incluindo regex para senhas
- **Resultado**: Implementei validações complexas de forma elegante e type-safe

5. Tratamento de Erros

- **O que fiz**: Consultei sobre melhores práticas para error handling em Express
- **Como ajudou**: A IA sugeriu criar classes customizadas de erro e middleware centralizado
- **Aprendizado**: Entendi a importância de um sistema de erros consistente e bem estruturado

6. Sistema de Autenticação JWT

- **O que fiz**: Pedi orientação sobre implementação segura de JWT
- **Como ajudou**: A IA explicou:
  - Como gerar e validar tokens
  - Onde armazenar o secret
  - Como criar middleware de autenticação
- **Aprendizado**: Aprendi sobre segurança e boas práticas em autenticação

7. Otimização de Queries do Prisma

- **O que fiz**: Questionei sobre as melhores formas de estruturar queries
- **Como ajudou**: A IA sugeriu o uso de `select` e relacionamentos eficientes
- **Resultado**: Queries mais performáticas e otimizadas

8. Frontend com React

- **O que fiz**: Usei Copilot para gerar componentes React
- **Partes do projeto**:
  - Formulários de login e registro
  - Componentes reutilizáveis (Button, Input)
  - Configuração de rotas

- **Aprendizado**: Aprendi patterns modernos de React como custom hooks e composition

9. Debugging e Resolução de Problemas

- **O que fiz**: Quando encontrava erros, copiava a mensagem e consultava a IA
- **Como ajudou**: A IA identificava rapidamente a causa e sugeria soluções
- **Exemplo**: Problemas com CORS, configuração de Prisma, types do TypeScript
- **Resultado**: Resolução muito mais rápida de bugs

10. Documentação

- **O que fiz**: Pedi ajuda para estruturar este README
- **Como ajudou**: A IA sugeriu seções importantes e formatação adequada
- **Resultado**: Documentação profissional e completa

O Que Aprendi com a Interação com IA

🎯 Aprendizados Técnicos

1. **Arquitetura de Software**: Compreendi melhor como estruturar projetos em monorepo e separação de concerns
2. **TypeScript Avançado**: Aprendi sobre tipos complexos, generics e type inference
3. **Boas Práticas**: Absorvi patterns e convenções da indústria
4. **Segurança**: Entendi conceitos de autenticação, autorização e proteção de dados

💡 Aprendizados sobre Uso de IA

1. **Prompts Específicos**: Aprendi que quanto mais específico e contextualizado o prompt, melhor a resposta
2. **Revisão Crítica**: Nem todo código gerado está perfeito - é essencial revisar e adaptar
3. **Complementaridade**: IA é excelente para acelerar, mas o conhecimento humano é essencial para decisões arquiteturais
4. **Aprendizado Ativo**: Usar IA não significa copiar e colar - é uma ferramenta de ensino quando usada corretamente

🚀 Impacto na Produtividade

- **Tempo Economizado**: Estimativa de 40-50% de redução no tempo de desenvolvimento
- **Qualidade do Código**: Melhor estruturação e aderência a padrões
- **Curva de Aprendizado**: Acelerei meu aprendizado de novas tecnologias (Prisma, Turborepo)
- **Foco em Lógica**: Pude focar mais na lógica de negócio do que em código boilerplate

⚠ Desafios e Cuidados

1. **Dependência**: É importante não se tornar dependente - entender o que está sendo gerado
2. **Contexto**: A IA pode não ter contexto completo do projeto, então decisões finais são humanas
3. **Atualização**: Algumas sugestões podem usar versões antigas de bibliotecas
4. **Segurança**: Código sensível (secrets, senhas) requer atenção extra

Conclusão sobre IA

O uso de IA no desenvolvimento foi **transformador**. Não substituiu meu papel como desenvolvedor, mas atuou como um **pair programmer incansável**, sempre disponível para tirar dúvidas, sugerir melhorias e acelerar tarefas repetitivas.

A chave é usar IA como uma **ferramenta de ampliação de capacidades**, não como substituto do pensamento crítico e criativo. O resultado foi um projeto melhor estruturado, desenvolvido mais rapidamente, mas com total compreensão de cada linha de código implementada.

👨‍💻 Autor

João Victor Barreto

Desenvolvido como parte do desafio técnico para seleção do Vortex Unifor.

⭐ Se este projeto foi útil, considere dar uma estrela no repositório!