**FATEC PROF. JESSEN VIDAL**

**CENTRO PAULA SOUZA**

Grupo Átomo

**Documentação Técnica: Sistema de Gestão de Parceiros**

**São José dos Campos**

**2024**

Grupo Átomo

**Documentação Técnica: Sistema de Gestão de Parceiros**

Documentação Técnica do Sistema de Gestão de Parceiros, desenvolvido pelo grupo Átomo durante o 5º período do curso de Análise e Desenvolvimento de Sistemas da FATEC de São José dos Campos em 2024.

**São José dos Campos**

**2024**

**SUMÁRIO**

[**1** **VISÃO GERAL DO PROJETO** 3](#_Toc169360988)

[**1.1** **Descrição do projeto** 3](#_Toc169360989)

[**2** **REQUISITOS** 3](#_Toc169360990)

[**2.1** **Requisitos Funcionais** 3](#_Toc169360991)

[**2.2** **Requisitos Não Funcionais** 3](#_Toc169360992)

[**3 ARQUITETURA DO SISTEMA** 4](#_Toc169360993)

[**3.1 Business Process Model and Notation - BPMN** 4](#_Toc169360994)

[**3.2 Fluxo de Dados** 5](#_Toc169360995)

[**3.3 Modelagem do Banco de Dados** 5](#_Toc169360996)

[**4 DESIGN DA APLICAÇÃO** 6](#_Toc169360997)

[**4.1 Estrutura de Pastas** 6](#_Toc169360998)

[**4.2 Bibliotecas e Frameworks** 7](#_Toc169360999)

[**5 GUIA DE CONFIGURAÇÃO E INSTALAÇÃO** 8](#_Toc169361000)

[**5.1 Pré-requisitos** 8](#_Toc169361001)

[**5.2 Passo a Passo de Instalação** 8](#_Toc169361002)

[**5.3 Configuração do Projeto** 8](#_Toc169361003)

[**6 Desenvolvimento** 9](#_Toc169361004)

[**6.1 Guia de Estilo de Código** 9](#_Toc169361005)

[**6.2 Fluxo de Trabalho do Git** 9](#_Toc169361006)

[**7 REFERÊNCIAS E RECURSOS ADICIONAIS** 9](#_Toc169361007)

[**7.1 Documentação de APIs** 9](#_Toc169361008)

[**ANEXOS** 12](#_Toc169361009)

# VISÃO GERAL DO PROJETO

## Descrição do projeto

O projeto em questão visa desenvolver um sistema de gestão de parceiros para a Oracle, especialmente projetado para operar em plataformas móveis. Este sistema tem como objetivo principal simplificar e otimizar a gestão dos parceiros da empresa, permitindo o cadastro eficiente de novos parceiros, atualizações contínuas de informações em tempo real, acompanhamento do desenvolvimento de conhecimento dos parceiros, e a geração de relatórios estratégicos. Ao resolver a complexidade e dispersão de informações enfrentadas atualmente, a Oracle busca melhorar a eficiência operacional, fortalecer a colaboração com seus parceiros e potencializar decisões estratégicas fundamentadas, posicionando-se de maneira mais competitiva no mercado.

# REQUISITOS

## Requisitos Funcionais

Os requisitos funcionais do sistema de gestão de parceiros para a Oracle incluem uma interface intuitiva para o cadastro de novos parceiros, permitindo a inserção eficiente de informações detalhadas. Além disso, é necessário desenvolver uma interface dedicada ao acompanhamento e desenvolvimento contínuo dos parceiros, facilitando a gestão de atualizações e progressos individuais. Um Dashboard dinâmico de indicadores de desempenho será implementado para oferecer uma visão panorâmica das métricas-chave relacionadas aos parceiros, como produtividade, satisfação e eficiência operacional. Além disso, um Relatório Analítico permitirá a análise do desempenho dos parceiros por produto implementado, proporcionando insights estratégicos essenciais para a tomada de decisões informadas e aprimoramento contínuo das parcerias estratégicas da Oracle.

## Requisitos Não Funcionais

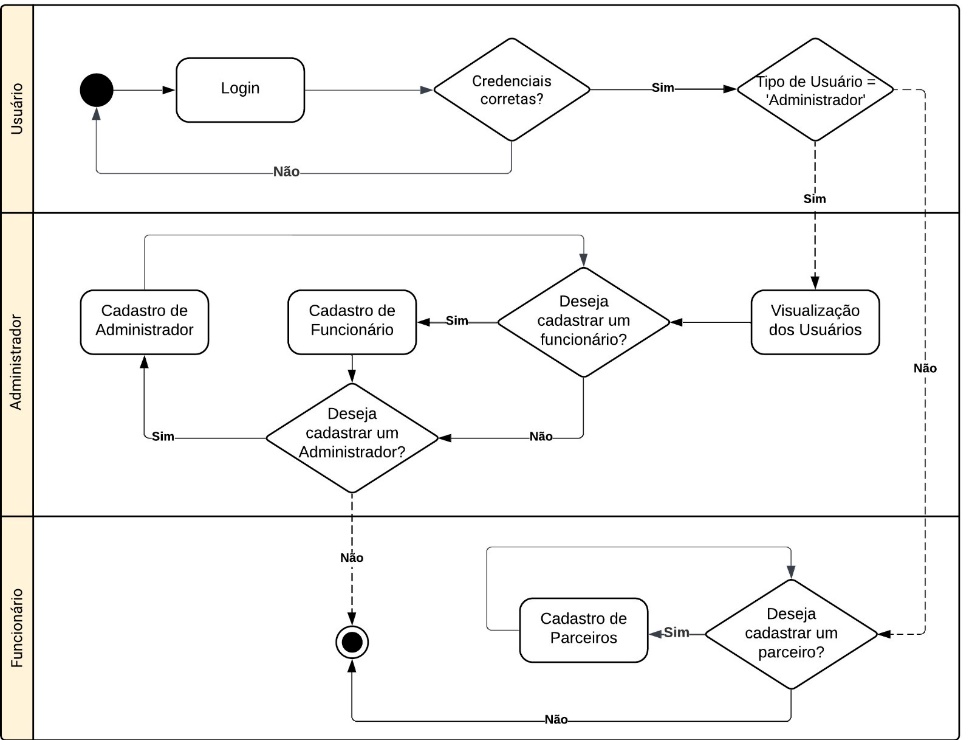
Os requisitos não funcionais essenciais para o sistema de gestão de parceiros da Oracle incluem a elaboração de um Manual do Usuário, fornecendo orientações claras e detalhadas desde o processo inicial de cadastro de parceiros até a interpretação dos relatórios analíticos. Além disso, é crucial a criação de uma Documentação Técnica detalhada que descreva a estruturação completa do sistema, abrangendo a arquitetura geral, fluxos de dados e decisões de design implementadas. A Documentação da API deve ser igualmente detalhada, oferecendo informações precisas sobre endpoints, métodos de requisição, parâmetros aceitos e exemplos práticos de integração para facilitar a interação do sistema com outras plataformas. Por fim, a Modelagem de Banco de Dados ou Arquivo de Dados deve ser cuidadosamente elaborada, apresentando um modelo claro da estrutura de dados utilizada, com tabelas, relações e atributos necessários para armazenar e gerenciar eficientemente as informações dos parceiros, garantindo segurança e performance do sistema.

# ARQUITETURA DO SISTEMA

## Business Process Model and Notation - BPMN

Para representar de forma visual as funcionalidades do sistema, foi elaborado um Business Process Model and Notation (Figura 1), ou BPMN, onde é possível mapear e identificar cada etapa do processo do sistema.

Figura 1 - BPMN



Fonte: autoria própria.

## Fluxo de Dados

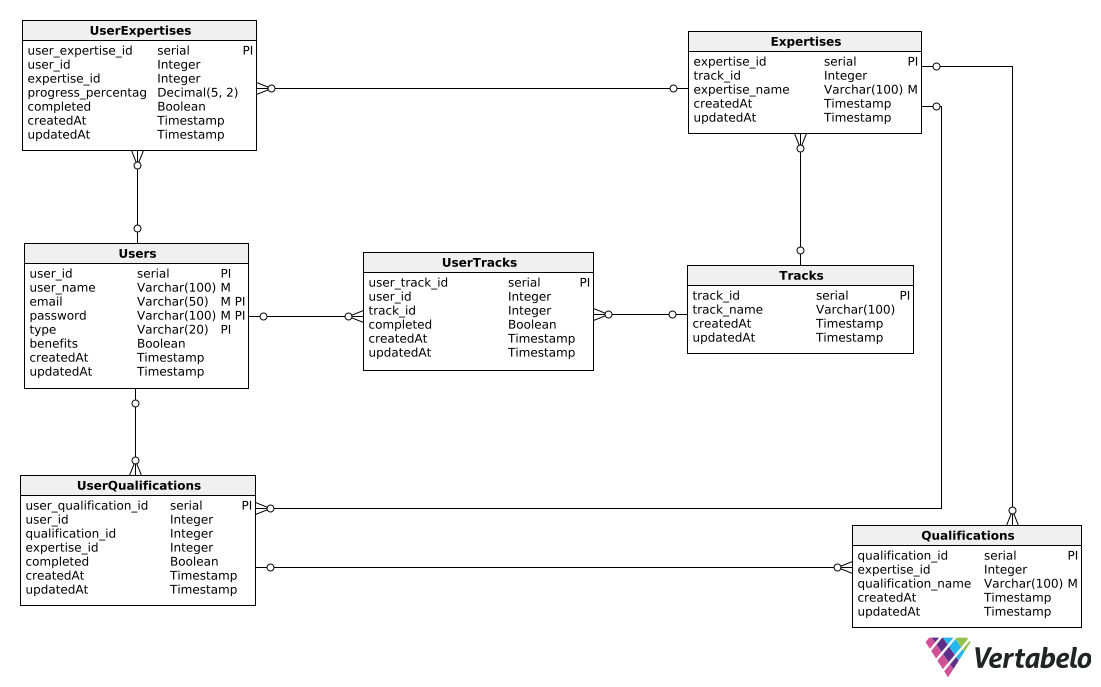
O fluxo de dados da aplicação tem início com a execução do aplicativo via framework Expo, que renderiza as telas e componentes navegáveis essenciais para o funcionamento do sistema. A partir dessa base, o aplicativo faz requisições a APIs externas utilizando bibliotecas como Axios, e hooks do React como useEffect e useState, para buscar dados necessários para preencher campos, gráficos e tabelas dentro da interface do usuário. Essas requisições são fundamentais para garantir que os dados exibidos estejam sempre atualizados e relevantes para o usuário final.

No lado do backend, as APIs são desenvolvidas utilizando bibliotecas e frameworks como Axios, CORS, Express e PG, e hospedada no Serviço em Nuvem da AWS. Essa configuração permite que o servidor execute operações de consulta, inserção, exclusão e modificação dos dados armazenados no banco de dados. As consultas ao banco de dados são realizadas de maneira eficiente, garantindo a integridade e a performance do sistema, facilitando a manipulação e o gerenciamento dos dados conforme as necessidades da aplicação.

## Modelagem do Banco de Dados

A Figura 2 representa de forma esquemática o modelo conceitual do banco de dados utilizado para o desenvolvimento do backend da aplicação.

Figura 2 - Modelagem do Banco de Dados



Fonte: autoria própria.

# DESIGN DA APLICAÇÃO

## Estrutura de Pastas

O projeto foi organizado de forma a facilitar o desenvolvimento e manutenção do sistema, onde cada diretório possui seu propósito. Foram separadas em:

* ‘front\_oracle/’: Pasta dedicada aos componentes que compõe a parte visual do projeto.
  + ‘assets/’: Armazena imagens e ícones usados pelo Expo na construção do aplicativo.
  + ‘src/’: Pasta principal para a construção das interfaces.
    - ‘assets/’: Contém as imagens de uso geral da aplicação.
    - ‘components/’: Contém as pastas de arquivos dos componentes específicos e gerais usados para compor as interfaces.
    - ‘connection/’: Contém o arquivo que faz a conexão com o servidor backend.
    - ‘routes/’: Contém os arquivos de rotas de navegação do aplicativo.
    - ‘styles/’: Contém pastas e arquivos de estilos personalizados pré-definidos, como fontes, cores, tamanhos e sombras.
    - ‘utils/’: Contém arquivos de uso geral, como mocks de listas.
    - ‘views/’: Contém as pastas e arquivos principais das interfaces.
  + ‘App.js’: Responsável por renderizar a aplicação como um todo.
  + Arquivos gerais do node: Arquivos como package.json e gitgnore, para configuração de ambiente.
* ‘back\_oracle’:
  + ‘src/’: Pasta principal para a construção das interfaces.
    - ‘controllers/’: Pasta que contém os arquivos controllers do backend responsáveis por gerenciar as consultas no banco.
    - ‘db/’: Pasta que contém os arquivos de conexão com os bancos de dados.
    - ‘models/’: Pasta que contém os arquivos models do backend, responsáveis por realizar as consultas no banco.
    - ‘routes/’: Pasta que contém os arquivos de rotas do backend, responsáveis por definir o caminho onde a requisição será feita.
  + ‘.env’: Arquivo de configuração de variáveis de ambiente.
  + ‘App.js’: Arquivo para execução geral do servidor backend.
  + Arquivos gerais do node: Arquivos como package.json e gitgnore, para configuração de ambiente.

## Bibliotecas e Frameworks

Para o desenvolvimento do projeto, foram utilizados bibliotecas e frameworks para realização de funções e construção de interfaces. Foram elas:

* Para o frontend:
  + "@expo/vector-icons": Biblioteca de ícones.
  + "@react-native-async-storage/async-storage": Biblioteca para armazenamento temporário de informações.
  + "react-navigation ": Biblioteca para navegação.
  + "axios": Biblioteca para conexão com servidores.
  + "expo": Framework para depuração e deploy da aplicação.
  + "expo-sharing": Biblioteca para compartilhamento interno de dados e informações na aplicação.
  + "expo-status-bar": Biblioteca para gerenciar a barra de status do sistema.
  + "react": Framework de hooks e elementos visuais.
  + "react-native": Framework principal para construção do aplicativo.
  + "xlsx": Framework para exportação de dados em planilha do excel.
* Para o backend:
  + "bcryptjs": Biblioteca para encriptação de senhas.
  + "body-parser": Biblioteca para análise de corpo das requisições.
  + "cors": Biblioteca que permite requisições de várias origens.
  + "dotenv": Biblioteca para uso de variáveis de ambiente.
  + "express": Biblioteca para roteamento e outras funções que facilitam o desenvolvimento.
  + "jsonwebtoken": Biblioteca que gera tokens para validação.
  + "nodemailer": Biblioteca para envio de e-mails.
  + "mongodb": Framework que permite interação com bancos de dados MongoDB
  + "mongoose": Biblioteca que permite modelagem de dados MongoDB.
  + "pg": Framework que permite interação com bancos de dados PostgreSQL.

# GUIA DE CONFIGURAÇÃO E INSTALAÇÃO

## Pré-requisitos

Para a instalação e configuração do aplicativo, é necessário que o usuário tenha disponibilidade ao Node.js, npm/yarn, Android Studio, Expo Go (em algum dispositivo móvel), PostgreSQL e MongoDB.

## Passo a Passo de Instalação

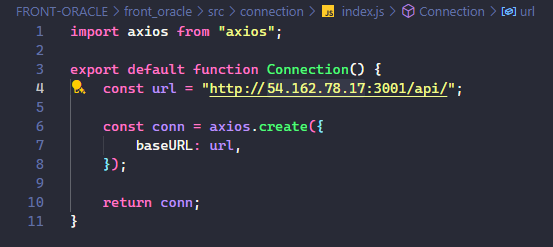
Dentro da pasta “front\_oracle”, o usuário deve executar o comando “npm install” para instalar todas as dependências do projeto, e para iniciá-lo, deve executar o comando ‘npx expo start –tunnel’.

Dentro da pasta “back\_oracle”, o usuário deve executar o mesmo comando “npm install”, e para iniciar o servidor, executar o comando “node App.js”.

## Configuração do Projeto

No frontend, a configuração deve ser feita em “FRONT-ORACLE\front\_oracle\src\connection”, alterando o IP da conexão pelo IP do servidor onde o backend está sendo executado, assim como vemos na Figura 3.

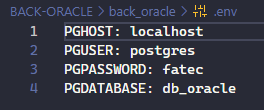
Figura 3 - Configuração de Conexão

****

Fonte: autoria própria.

Para o backend, é necessário realizar a configuração das variáveis de ambiente, adicionando um arquivo “.env” da aplicação. O arquivo deve ser estruturado como mostra a Figura 4.

Figura 4 - Variáveis de ambiente

****

Fonte: autoria própria.

# Desenvolvimento

## Guia de Estilo de Código

Para facilitar o desenvolvimento e manutenção do código, o projeto foi estruturado seguindo o padrão model, controller e routes para o back, e views, components, styles e routes para o front. Cada pasta com um propósito diferente, facilitando a reutilização de componentes e diminuindo a quantidade de código.

## Fluxo de Trabalho do Git

Para o Git, foi aplicado o fluxo de trabalho por branches, onde cada usuário era responsável por realizar suas tarefas e criar “commits” em sua respectiva branch, abrindo um “Pull Request” em cada sprint para que suas alterações fossem incluídas na branch principal.

Também foi aplicado um padrão de commits para melhor organização e versionamento.

# REFERÊNCIAS E RECURSOS ADICIONAIS

## Documentação de APIs

A conexão entre frontend e backend é feita através de endpoints mantidos pelo servidor, e cada endpoint exige informações específicas para seu funcionamento. Nessa aplicação, foram utilizados endpoints para criação, edição, remoção e visualização de dados. São eles, e suas exigências assim como seus retornos:

* ‘/api’ 🡪 Rota base da aplicação. Antecede todas as próximas rotas.
  + post(‘/login’) 🡪 Rota de login do usuário. Seu body exige email e senha, e caso essas informações estejam corretas, realiza o login e retorna os dados do usuário logado em sua “response”.
  + post(‘/registerPartner’) 🡪 Rota de cadastro de parceiros. Seu body exige nome, email e senha, e retorna os dados do usuário criado caso haja sucesso.
  + post(‘/selectExpertise’) 🡪 Rota para retornar expertises do parceiro selecionado. Seu body exige o id do usuário em questão.
  + get(‘/listarExpertises/:track\_id’) 🡪 Rota para retornar a lista de todas as expertises das tracks. A url exige o id da track especificada.
  + get(‘/expertises/:expertise\_id/users/:user\_id/qualifications’) 🡪 Rota para retornar a lista de todos os qualificadores das expertises de um usuário específico. A url exige o id da expertise e o id do usuário.
  + post(‘/userQualifications’) 🡪 Rota para concluir o qualificador escolhido do parceiro selecionado. Seu body exige o id do usuário e o id do qualificador.
  + post(‘/userTracks’) 🡪 Rota para concluir a track escolhida do parceiro selecionado. Seu body exige o id do usuário e o id da track.
  + post(‘/userExpertises’) 🡪 Rota para concluir a expertise escolhida do parceiro selecionado. Seu body exige o id do usuário e o id da expertise.
  + post(‘/tracksById/:user\_id’) 🡪 Rota para retornar as tracks concluídas do parceiro selecionado. A url exige o id do usuário em questão como parâmetro.
  + post(‘/cadastro’) 🡪 Rota de cadastro de consultores e administradores. Seu body exige nome, email, senha, tipo e benefícios, e retorna uma mensagem de erro ou de sucesso.
  + get(‘/listUsers’) 🡪 Rota para retornar a lista de todos os usuários registrados.
  + put(‘/partners/:id’) 🡪 Rota para atualizar usuário. A url exige o id do usuário a ser atualizado, e seu body exige um objeto com nome e email. Retorna mensagem de erro ou sucesso.
  + delete(‘/partners/:id’) 🡪 Rota para deletar parceiro. A url exige o id do usuário a ser deletado, e seu body exige o tipo do usuário que realizará a operação. Retorna mensagem de erro ou sucesso.
  + delete(‘/partners/:id’) 🡪 Rota para deletar parceiro. A url exige o id do usuário a ser deletado, e seu body exige o tipo do usuário que realizará a operação. Retorna mensagem de erro ou sucesso.
  + get(‘/partner/:userId’) 🡪 Seu body recebe o id de algum usuário, e retorna seus dados.
  + delete(‘/users/:id’) 🡪 Rota para deletar usuário. A url exige o id do usuário a ser deletado. Retorna mensagem de erro ou sucesso.
  + post(‘/password-reset-request’) 🡪 Rota para enviar um email de requisição de recuperação de senha. Seu body exige um endereço de email, e retorna uma mensagem de erro ou sucesso.
  + post(‘/reset-password’) 🡪 Rota para alterar senha após requisição de recuperação. Seu body exige o token de validação enviado no email e uma nova senha, e retorna uma mensagem de erro ou sucesso.
  + post(‘expertiseProgressByUserAndTrack’) 🡪 Rota para retornar o percentual de progressão das expertises de uma track referente à um usuário. Seu body exige o id do usuário e da track.
  + get(‘/track’) 🡪 Retorna a lista das tracks disponíveis.
  + put(‘/users/:userId/password’) 🡪 Atualiza a senha do usuário com base no id passado na url. Seu body exige a senha antiga e a nova senha, e retorna uma mensagem de erro ou sucesso.

# ANEXOS

* GitHub do projeto

Para mais informações, como cronogramas, organização, arquivos, reviews e informações da equipe, acesse o GitHub do grupo.

GitHub: <https://github.com/atomofatec/API-ORACLE>