PROJETO CHAMAGRO

1. Descrição do Projeto

ChamAgro

2. Objetivo

O presente projeto visa aprimorar a comunicação entre o Instituto de Assistência Técnica e Extensão Rural – IATER e os agricultores do estado de Roraima, facilitando a troca de informações sobre o controle de pragas. Com isso, busca-se proteger o meio ambiente e aumentar a produtividade agrícola. A plataforma proposta visa reduzir o tempo de resposta entre a detecção de pragas e a adoção de medidas corretivas, permitindo que o IATER, e, quando necessário, a Agência de Defesa Agropecuária de Roraima (ADERR), possam atuar de forma rápida e eficaz.

2. Tecnologias Utilizadas

a) *Frontend*

Next.js: Framework React que fornece uma interface rápida e amigável ao SEO.

Next Auth: Gerencia autenticação por meio de vários provedores, como e-mail/senha.

React Query: Gerencia a busca de dados e o gerenciamento de estado, proporcionando uma experiência de usuário fluida.

React Hook Form, Zod e Yup: Utilizados para validação confiável e contínua de formulários.

Shadcn UI Components: Fornece componentes de interface de usuário consistentes e acessíveis.

Tailwind CSS: Framework CSS utilizado para estilizar a interface de forma responsiva e eficiente.

Framer Motion: Biblioteca para animações e transições suaves, aprimorando a experiência do usuário.

Lucide React Icons: Biblioteca de ícones para melhorar a estética e a usabilidade do aplicativo.

Date-fns: Biblioteca para manipulação de datas, essencial para recursos de calendário e agendamento.

JavaScript: Linguagem base usada tanto no frontend quanto no backend.

b) *Backend*

Node.js: Ambiente de execução JavaScript no servidor, para construção da API.

Express: Framework para Node.js que organiza a estrutura do backend, rotas e requisições.

Prisma ORM: ORM para facilitar a manipulação de dados no PostgreSQL de forma segura.

c) *Banco de Dados*

PostgreSQL: Sistema de banco de dados relacional para armazenar informações sobre usuários, pragas, notificações e conversas.

Geolocalização e Mapas

Google Maps API / Mapbox: Exibe áreas afetadas em um mapa, auxiliando na análise de regiões impactadas.

Notificações em Tempo Real

WebSockets (via Socket.IO): Comunicação em tempo real, possibilitando envio de notificações push e chat entre IATE e agricultores.

Armazenamento de Imagens

Google Firebase / Appwrite: Plataformas para armazenamento seguro de imagens enviadas pelos agricultores, como fotos de áreas afetadas.

Essas ferramentas combinadas permitem uma aplicação robusta, segura e de fácil utilização para os usuários.

3. Dependências Necessárias

a) *Backend*

Node.js: v20.17.0
Express: v4.18.3
Prisma ORM: v5.22.0
Socket.IO: v4.7.2

b) *Frontend*

Next.js: v14.0.0
Next Auth: v4.22.1
React Query: v5.0.0
React Hook Form: v7.45.0
Zod: v3.22.2
Yup: v1.1.0
Shadcn UI Components: v0.1.0
Tailwind CSS: v3.3.2
Framer Motion: v10.0.0
Lucide React Icons: v0.236.0
Date-fns: v2.30.0

Observação: Certifique-se de usar as versões recomendadas para evitar problemas de compatibilidade.

4. Variáveis de Ambientes

PORT=5000

DB_PRODUCTION=false

NODE_ENV=production

POSTGRES_URL="postgres://default:bU4mrpoklF3G@ep-royal-math-a4c615f1.us-east-1.aws.neon.tech:5432/verceldb?sslmode=require"

DB_LOCAL_URL="postgres://postgres:admin@localhost:5432/chamagro_db"

JWT_SALTROUND = 10

JWT_SECRET="2NmORAkTBKcDpxFOMSQMPdKWKqZ4tQq+9XY2jBnB8ywYrd3NTY9YZn8LHSM="

JWT_ACCESS_EXPIRATION_MINUTES=30

JWT_REFRESH_EXPIRATION_DAYS=30

JWT_RESET_PASSWORD_EXPIRATION_MINUTES=10

JWT_VERIFY_EMAIL_EXPIRATION_MINUTES=10

SMTP_HOST=smtp.gmail.com

SMTP_PORT=587

SMTP_USERNAME=devacc2010@gmail.com

SMTP_PASSWORD=fbddxayxoqodzlqr

EMAIL_FROM=devacc2010@gmail.com

OAUTH_CLIENT_ID=145765227010-gldu312hatrd0i0ov6sjm104hon8spah.apps.googleusercontent.com

OAUTH_CLIENT_SECRET=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

OAUTH_REDIRECT_URI=https://developers.google.com/oauthplayground

OAUTH_REFRESH_TOKEN=1//049VwR7plRAKBCgYIARAAGAQSNwF-L9IrdYnABKkLVZy6joTk9YbsBJYduc_8hE_XZEgiDX6rCTJ2TFbd_5A_NjKb-alpLJjUG-M

OAUTH_USER = devacc2010@gmail.com

ENCRYPTION_SECRET_KEY="6893686b45c188f931a4c3ff71fc70261ec504c978b54d0cd14f63a88f516ffd"

APPWRITE_ENDPOINT=https://cloud.appwrite.io/v1

APPWRITE_PROJECT_ID=66ed3ffa003ab4e401de

APPWRITE_API_KEY=standard_d1a45c4a1f1027cc5e0cea1c12149628e8a70328fa90b30a77d9b07fb471a8547718c27f2e1c549810f8e403efddbeb380746092ae00ea253fa4492a41946ad46c375ec0bf2bac0b53b3946629a0a9cc11447cb3785e1142f8b7337274183f264cad771a2f3ac50606185eb3c1499b8e790c3a31fb8974389844e44bd313c2e7

APPWRITE_BUCKET_ID=66ed4078000ce1f08a1f

5. Como rodar a Aplicação

5.1 Pré-requisitos

Antes de iniciar, verificar se possui:

Node.js (versão X.X.X ou superior)
NPM ou Yarn (para gerenciar pacotes)
PostgreSQL (banco local ou em nuvem configurado)
Firebase / Appwrite (configurações e conta de armazenamento)

5.2 Instalação

Clone o repositório:

1) git clone [URL do repositório]
2) Instale as dependências:
3) cd [diretório do projeto]
4) npm install # ou yarn install

Configure as variáveis de ambiente: Crie um arquivo. *env* no diretório raiz e adicione as chaves necessárias, como credenciais do banco de dados e chaves de autenticação. Certifique-se de incluir o. *env* no arquivo. *gitignore* para proteger suas credenciais sensíveis:

DATABASE_URL="postgresql://usuario:senha@localhost:5432/chamagro"
GOOGLE_MAPS_API_KEY="sua-chave-google-maps"
FIREBASE_API_KEY="sua-chave-firebase"

Construa o projeto:

npm run build
Inicie o servidor: Para um ambiente de produção, execute:
npm start
Para um ambiente de desenvolvimento, execute:
npm run dev

Por padrão, a aplicação é executada em http://localhost:3000, mas você pode configurar a porta usando a variável de ambiente PORT no arquivo. env.

6. Documentação de Validação com Zod

Esta documentação descreve os esquemas de validação utilizados para autenticação de usuários, incluindo o registro, login, verificação de código e redefinição de senha, todos definidos com a biblioteca Zod.

1. signUpSchema - Esquema de Registro de Usuário

O signUpSchema valida os dados necessários para o registro de um novo usuário. As regras de validação para cada campo são:

email: O campo deve ser uma string no formato de e-mail. Caso contrário, a mensagem "O email é inválido" será exibida.

password: O campo deve ser uma string com no mínimo 6 caracteres e no máximo 12 caracteres. Caso contrário, as mensagens de erro serão:

"A senha deve ter no mínimo 6 caracteres" se o valor for muito curto.
"A senha deve ter no máximo 12 caracteres" se o valor for muito longo.

confirmPassword: Deve seguir as mesmas regras do campo password, com a mesma validação de tamanho.

firstName: O nome deve ser uma string com pelo menos 3 caracteres e no máximo 100 caracteres. Se o valor for muito curto, exibe a mensagem "O nome deve ter pelo menos 3 caracteres". Se for muito longo, exibe "O nome deve ter no máximo 100 caracteres".

lastName: Mesmas regras de firstName para o sobrenome.

role: Este campo deve ser um valor de enumeração, aceitando apenas os valores "DEFAULT", "PRODUTOR", "MERCHANT" e "TECHNICIAN". Caso um valor inválido seja fornecido, a mensagem "Tipo de função do usuário é inválido" será exibida.

Regra adicional:

Verificação de senhas: O campo confirmPassword deve ser igual ao campo password. Se as senhas não coincidirem, a mensagem de erro "As senhas não coincidem" será exibida.

Tipo Inferido:

export type SignUpSchema = z.infer<typeof signUpSchema>

2. signInSchema - Esquema de Login de Usuário

O signInSchema valida os dados de entrada para login.
email: O campo deve ser uma string (não há verificação adicional de formato no esquema).
password: O campo deve ser uma string e é obrigatório. A mensagem de erro "A senha é obrigatória" será exibida caso o campo esteja vazio.

Tipo Inferido:

export type SignInSchema = z.infer<typeof signInSchema>

3. authSchema - Esquema de Autenticação

O authSchema valida o campo user_email, que deve ser um e-mail válido.
user_email: Deve ser uma string no formato de e-mail. Se o e-mail não for válido, a mensagem "Deve informar um email válido" será exibida.

Tipo Inferido:

export type AuthSchema = z.infer<typeof authSchema>

4. verificationCodeSchema - Esquema de Verificação de Código

O verificationCodeSchema valida os dados para a verificação do código enviado ao usuário.

email: O campo deve ser uma string no formato de e-mail. Caso contrário, exibe a mensagem "Deve informar um email válido".

code: O campo deve ser uma string com pelo menos 6 caracteres. Caso contrário, exibe a mensagem "O código deve ter no mínimo 6 caracteres".

Tipo Inferido:

export type VerificationCodeSchema = z.infer<typeof verificationCodeSchema>

5. passwordResetSchema - Esquema de Redefinição de Senha

O passwordResetSchema valida os dados necessários para a redefinição de senha.

email: O campo deve ser uma string no formato de e-mail, exibindo a mensagem "Deve informar um email válido" caso o formato seja inválido.

code: O campo deve ser uma string com pelo menos 6 caracteres. Exibe a mensagem "O código deve ter no mínimo 6 caracteres" caso o código seja muito curto.

password: O campo deve ser uma string com no mínimo 6 caracteres e no máximo 12 caracteres. Caso contrário, exibe as mensagens de erro:

"A senha deve ter no mínimo 6 caracteres" se for muito curta.

"A senha deve ter no máximo 12 caracteres" se for muito longa.

confirmPassword: Deve seguir as mesmas regras do campo password.

Regra adicional:

Verificação de senhas: O campo confirmPassword deve ser igual ao campo password. Se não forem iguais, a mensagem "As senhas não coincidem" será exibida.

Tipo Inferido:

export type PasswordResetSchema = z.infer<typeof passwordResetSchema>

Tipos Inferidos de Todos os Esquemas

Cada um dos esquemas definidos tem um tipo inferido associado que pode ser utilizado para tipar funções que interagem com esses dados:

export type SignUpSchema = z.infer<typeof signUpSchema>

export type SignInSchema = z.infer<typeof signInSchema>

export type VerificationCodeSchema = z.infer<typeof verificationCodeSchema>

export type AuthSchema = z.infer<typeof authSchema>

export type PasswordResetSchema = z.infer<typeof passwordResetSchema>

Esses tipos ajudam a garantir que os dados manipulados no código estejam de acordo com os formatos e as regras definidas, proporcionando maior segurança e previsibilidade.

7. Documentação de Rotas de Autenticação

Esta documentação descreve as rotas da API relacionadas à autenticação de usuários, incluindo registro, login, redefinição de senha e verificação de token. As rotas são construídas usando o framework Express e fazem uso de middleware de validação para garantir a conformidade dos dados recebidos com os esquemas definidos.

- Rota 1: Registro de Usuário

URL: /

Método: POST

Descrição: Realiza o registro de um novo usuário.

Validação: registerSchema

Controle: registerController

Campos Requeridos:

email: E-mail do usuário.

password: Senha do usuário.

confirmPassword: Confirmação da senha.

firstName: Nome do usuário.

lastName: Sobrenome do usuário.

role: Função do usuário (DEFAULT, PRODUTOR, MERCHANT, TECHNICIAN).

Resposta Esperada: Confirmação do sucesso do registro ou mensagens de erro relacionadas à validação.

- Rota 2: Verificar Credenciais do Usuário por E-mail

URL: /check-credential/:email

Método: GET

Descrição: Verifica se as credenciais (e-mail) do usuário já estão registradas.

Validação: credentialCheckSchema

Controle: checkUserCredentialByEmailController

Parâmetros:

email: E-mail do usuário para verificar as credenciais.

Resposta Esperada: Confirmação de se o e-mail está registrado ou não no sistema.

- Rota 3: Solicitar Redefinição de Senha

URL: /token/request-password-reset/:email

Método: GET

Descrição: Solicita a redefinição de senha através do envio de um e-mail de verificação.

Validação: credentialCheckSchema

Controle: requestPasswordResetController

Parâmetros:

email: E-mail do usuário para o qual a solicitação de redefinição de senha será enviada.

Resposta Esperada: E-mail enviado ou erro relacionado ao e-mail.

- Rota 4: Redefinir Senha de Conta

URL: /account/reset-password

Método: POST

Descrição: Permite a redefinição de senha do usuário.

Validação: passwordResetSchema

Controle: resetPasswordController

Campos Requeridos:

email: E-mail do usuário.

code: Código de verificação recebido no e-mail.

password: Nova senha do usuário.

confirmPassword: Confirmação da nova senha.

Resposta Esperada: Confirmação de sucesso ou erro de validação.

- Rota 5: Verificar Token de Autenticação

URL: /verify-token

Método: POST

Descrição: Verifica a validade de um token de autenticação.

Validação: verifyTokenSchema

Controle: verifyTokenController

Campos Requeridos:

token: Token de autenticação a ser verificado.

Resposta Esperada: Confirmação da validade do token ou erro.

- Rota 6: Verificar Token de Redefinição de Senha

URL: /verify-password-token

Método: POST

Descrição: Verifica a validade do token enviado para redefinição de senha.

Validação: verifyTokenSchema

Controle: verifyPasswordResetRequestController

Campos Requeridos:

token: Token de redefinição de senha a ser verificado.

Resposta Esperada: Confirmação de validade do token ou erro.

- Rota 7: Login de Usuário

URL: /login

Método: POST

Descrição: Realiza o login do usuário.

Validação: loginSchema

Controle: loginController

Campos Requeridos:

email: E-mail do usuário.

password: Senha do usuário.

Resposta Esperada: Token de autenticação ou erro de validação.

- Rota 8: Login com NextAuth

URL: /user/:email

Método: GET

Descrição: Realiza o login com NextAuth utilizando o e-mail do usuário.

Validação: credentialCheckSchema

Controle: loginWithNextAuthController

Parâmetros:

email: E-mail do usuário para login com NextAuth.

Resposta Esperada: Token de autenticação ou erro.

Esquemas de Validação Utilizados

registerSchema: Valida os dados para o registro de um novo usuário.

credentialCheckSchema: Valida o e-mail do usuário para verificar sua existência ou seu status.

passwordResetSchema: Valida os dados necessários para redefinir a senha do usuário.

loginSchema: Valida os dados para o login de um usuário.

verifyTokenSchema: Valida o token para autenticação e redefinição de senha.

Resumo das Funções de Controle

registerController: Controla o registro de um novo usuário.

checkUserCredentialByEmailController: Verifica as credenciais do usuário por e-mail.

requestPasswordResetController: Solicita a redefinição de senha.

resetPasswordController: Realiza a redefinição de senha.

verifyTokenController: Verifica a validade de um token de autenticação.

verifyPasswordResetRequestController: Verifica a validade do token de redefinição de senha.

loginController: Controla o processo de login.

loginWithNextAuthController: Realiza o login utilizando NextAuth com base no e-mail do usuário

8. Documentação de Rotas de Autenticação

Esta documentação descreve as rotas da API relacionadas à autenticação de usuários, incluindo registro, login, redefinição de senha e verificação de token. As rotas são construídas usando o framework Express e fazem uso de middleware de validação para garantir a conformidade dos dados recebidos com os esquemas definidos.

- Rota 1: Registro de Usuário

URL: /

Método: POST

Descrição: Realiza o registro de um novo usuário.

Validação: registerSchema

Controle: registerController

Campos Requeridos:

email: E-mail do usuário.

password: Senha do usuário.

confirmPassword: Confirmação da senha.

firstName: Nome do usuário.

lastName: Sobrenome do usuário.

role: Função do usuário (DEFAULT, PRODUTOR, MERCHANT, TECHNICIAN).

Resposta Esperada: Confirmação do sucesso do registro ou mensagens de erro relacionadas à validação.

- Rota 2: Verificar Credenciais do Usuário por E-mail

URL: /check-credential/:email

Método: GET

Descrição: Verifica se as credenciais (e-mail) do usuário já estão registradas.

Validação: credentialCheckSchema

Controle: checkUserCredentialByEmailController

Parâmetros:

email: E-mail do usuário para verificar as credenciais.

Resposta Esperada: Confirmação de se o e-mail está registrado ou não no sistema.

- Rota 3: Solicitar Redefinição de Senha

URL: /token/request-password-reset/:email

Método: GET

Descrição: Solicita a redefinição de senha através do envio de um e-mail de verificação.

Validação: credentialCheckSchema

Controle: requestPasswordResetController

Parâmetros:

email: E-mail do usuário para o qual a solicitação de redefinição de senha será enviada.

Resposta Esperada: E-mail enviado ou erro relacionado ao e-mail.

- Rota 4: Redefinir Senha de Conta

URL: /account/reset-password

Método: POST

Descrição: Permite a redefinição de senha do usuário.

Validação: passwordResetSchema

Controle: resetPasswordController

Campos Requeridos:

email: E-mail do usuário.

code: Código de verificação recebido no e-mail.

password: Nova senha do usuário.

confirmPassword: Confirmação da nova senha.

Resposta Esperada: Confirmação de sucesso ou erro de validação.

- Rota 5: Verificar Token de Autenticação

URL: /verify-token

Método: POST

Descrição: Verifica a validade de um token de autenticação.

Validação: verifyTokenSchema

Controle: verifyTokenController

Campos Requeridos:

token: Token de autenticação a ser verificado.

Resposta Esperada: Confirmação da validade do token ou erro.

- Rota 6: Verificar Token de Redefinição de Senha

URL: /verify-password-token

Método: POST

Descrição: Verifica a validade do token enviado para redefinição de senha.

Validação: verifyTokenSchema

Controle: verifyPasswordResetRequestController

Campos Requeridos:

token: Token de redefinição de senha a ser verificado.

Resposta Esperada: Confirmação de validade do token ou erro.

- Rota 7: Login de Usuário

URL: /login

Método: POST

Descrição: Realiza o login do usuário.

Validação: loginSchema

Controle: loginController

Campos Requeridos:

email: E-mail do usuário.

password: Senha do usuário.

Resposta Esperada: Token de autenticação ou erro de validação.

- Rota 8: Login com NextAuth

URL: /user/:email

Método: GET

Descrição: Realiza o login com NextAuth utilizando o e-mail do usuário.

Validação: credentialCheckSchema

Controle: loginWithNextAuthController

Parâmetros:

email: E-mail do usuário para login com NextAuth.

Resposta Esperada: Token de autenticação ou erro.

9. Esquemas de Validação Utilizados

registerSchema: Valida os dados para o registro de um novo usuário.
credentialCheckSchema: Valida o e-mail do usuário para verificar sua existência ou seu status.
passwordResetSchema: Valida os dados necessários para redefinir a senha do usuário.
loginSchema: Valida os dados para o login de um usuário.
verifyTokenSchema: Valida o token para autenticação e redefinição de senha.

10. Resumo das Funções de Controle

registerController: Controla o registro de um novo usuário.
checkUserCredentialByEmailController: Verifica as credenciais do usuário por e-mail.
requestPasswordResetController: Solicita a redefinição de senha.
resetPasswordController: Realiza a redefinição de senha.
verifyTokenController: Verifica a validade de um token de autenticação.
verifyPasswordResetRequestController: Verifica a validade do token de redefinição de senha.
loginController: Controla o processo de login.
loginWithNextAuthController: Realiza o login utilizando NextAuth com base no e-mail do usuário.

11. Documentação de Rotas de Usuário

Esta documentação descreve as rotas responsáveis pela gestão de usuários no sistema. As rotas permitem a criação, atualização, deleção e consulta de informações de usuários, bem como a atualização de fotos de perfil. As rotas são gerenciadas utilizando o framework Express.

- Rota 1: Criar e Consultar Usuários

URL: /

Métodos:

POST: Criação de um novo usuário.

GET: Listagem de todos os usuários.

Descrição:

POST: Permite registrar um novo usuário no sistema.

GET: Retorna uma lista de usuários. A busca pode ser filtrada conforme os parâmetros definidos no schema de consulta.

Requisição POST:

Corpo da Requisição: Um objeto JSON com os dados necessários para criar o usuário.

```
{
 "email": "example@example.com",
 "name": "John Doe",
 "password": "password123"
}
```

Resposta Esperada:

```
{
 "status": "success",
 "message": "Usuário criado com sucesso",
 "data": { ...userData }
}
```

Requisição GET:

Parâmetros de Consulta: Aceita filtros para consulta de usuários, como limites de paginação e parâmetros de pesquisa.

Resposta Esperada:

```
{
 "status": "success",
 "message": "Lista de usuários",
 "data": [ ...userArray ]
}
```

- Rota 2: Consultar, Atualizar e Deletar Usuários Específicos

URL: /:userId

Métodos:

GET: Consulta os dados de um usuário específico pelo ID.

PATCH: Atualiza as informações de um usuário específico.
DELETE: Deleta um usuário específico.

Descrição:

GET: Permite consultar as informações de um usuário com base no ID fornecido.
PATCH: Permite atualizar as informações do usuário com base no ID fornecido. A atualização pode incluir dados como nome, e-mail, etc.
DELETE: Exclui um usuário do sistema.

Requisição PATCH:

Corpo da Requisição: Um objeto JSON contendo os dados a serem atualizados.

```
{
 "name": "John Updated",
 "email": "newemail@example.com"
}
```

Resposta Esperada:

```
{
 "status": "success",
 "message": "Usuário atualizado com sucesso",
 "data": { ...updatedUserData }
}
```

Requisição DELETE:

Corpo da Requisição: Nenhum.

Resposta Esperada:

```
{
 "status": "success",
 "message": "Usuário deletado com sucesso",
 "data": null
}
```

- Rota 3: Atualizar Foto de Perfil de Usuário

URL: /user/:userId
Método: POST

Descrição: Permite atualizar a foto de perfil de um usuário.
Requisição:
Corpo da Requisição: Um arquivo de imagem (geralmente enviado como multipart/form-data).

Resposta Esperada:

```
{
 "status": "success",
 "message": "Foto de perfil atualizada com sucesso",
 "data": { ...updatedUserData }
}
```

- Rota 4: Consultar Usuário por E-mail

URL: /email/:email

Método: GET
Descrição: Permite consultar as informações de um usuário específico pelo seu e-mail.
Requisição:
Parâmetro de URL: O e-mail do usuário.
Resposta Esperada:

```
{
 "status": "success",
 "message": "Usuário encontrado",
 "data": { ...userData }
}
```

Validação e Schemas

As rotas utilizam schemas de validação baseados no Zod para garantir que os dados de entrada e as consultas sejam válidos. A validação é aplicada por meio do middleware validate, que garante que as entradas estejam de acordo com os seguintes esquemas:

userRegistrationSchema: Usado para validar os dados ao criar um novo usuário.
queryUserEmailSchema: Usado para validar a consulta de um usuário pelo e-mail.
queryUserSchema: Usado para validar a consulta de um usuário pelo ID.
queryUsersSchema: Usado para validar a consulta de uma lista de usuários.
updateUserSchema: Usado para validar a atualização de dados de um usuário.

Exemplo de Resposta de Sucesso Padrão

As respostas para todas as rotas seguem o seguinte formato:

```
{
 "status": "success",
 "message": "Descrição da operação",
 "data": <dados retornados ou null>
}
```

Exemplo de resposta de sucesso:

```
{
 "status": "success",
 "message": "Usuário criado com sucesso",
 "data": {
  "id": 1,
  "email": "example@example.com",
  "name": "John Doe"
 }
}
```

Erros

Quando ocorre um erro em qualquer uma das rotas, a resposta segue o formato de erro padrão:

```
{
 "status": "error",
 "message": "Descrição do erro",
 "data": null
}
```

Esta documentação descreve as rotas de usuário de forma detalhada, abrangendo os métodos HTTP, parâmetros necessários, e exemplos de respostas para uma melhor compreensão da utilização das APIs