

**UNIKANBAN: SISTEMA KANBAN DESENVOLVIDO
COM ARQUITETURA MVC**

Alain Diego dos Passos

Fernanda Amaral Santos

Jairo Marcos do Nascimento Santos Filho

Luan Gustavo França Ricardo

Naiara Daros Duarte

Sumário

1. Introdução.....	4
2. Tecnologias Utilizadas.....	5
3. Back-end.....	6
3.1 Estrutura de Pastas do Backend.....	6
3.2 Backend: Módulo config/.....	8
3.2.1 dbConnect.js.....	8
3.2.2 jsonSecret.js.....	8
3.3 Backend: Módulo controllers/.....	8
3.3.1 AuthController.....	8
3.3.2 ColunaController.....	8
3.3.3 QuadroController.....	8
3.3.4 TarefaController.....	9
3.3.5 UsuarioController.....	9
3.4 Backend: Módulo dtos/.....	10
3.4.1 ColunaDto.....	10
3.4.2 QuadroDto.....	10
3.4.3 TarefaDto.....	10
3.4.4 UsuarioDto.....	10
3.5 Backend: Módulo middleware/.....	10
3.5.1 authMiddleware.js.....	10
3.6 Backend: Módulo models/.....	11
3.6.1 Coluna.js.....	11
3.6.2 Quadro.js.....	11
3.6.3 Tarefa.js.....	11
3.6.4 Usuario.js.....	12
3.6.5 index.js – Relacionamentos.....	12
3.7 Backend: Módulo repositories/.....	12
3.7.1 BaseRepository.js.....	12
3.7.2 ColunaRepository.js.....	13
3.7.3 QuadroRepository.js.....	13
3.7.4 TarefaRepository.js.....	13
3.7.5 UsuarioRepository.js.....	13
3.8 Backend: Módulo routes/.....	14
3.8.1 authRoutes.js.....	14
3.8.2 colunasRoutes.js.....	14
3.8.3 quadrosRoutes.js.....	14
3.8.4 tarefaRoutes.js.....	15
3.8.5 usuarioRoutes.js.....	15
3.9 Backend: Módulo services/.....	15
3.9.1 AuthService.....	15
3.9.2 ColunaService.....	15
3.9.3 QuadroService.....	16
3.9.4 TarefaService.....	16

3.9.5 UsuarioService.....	16
3.10 Backend: Arquivo app.js.....	16
3.11 Backend: Arquivo server.js.....	17
3.12 Backend: Arquivo .gitignore.....	17
3.13 Backend: Arquivo database.sqlite.....	18
3.14 Backend: Arquivo package.json.....	18
3.15 Backend: Arquivo package-lock.json.....	19
4. Front-end.....	19
4.1 Estrutura de Pastas do Frontend:.....	19
4.2 Frontend: Módulo components/.....	20
4.2.1 CardTarefa.js.....	20
4.2.2 ModalNovaTarefa.js.....	20
4.2.3 ModalTarefa.js.....	21
4.2.4 Navbar.js.....	21
4.2.5 PrivateRoute.js.....	21
4.3 Frontend: Módulo pages/.....	22
4.3.1 Login.js.....	22
4.3.2 QuadroPage.js.....	22
4.3.3 TarefaPage.js.....	22
4.3.4 UsuarioPage.js.....	23
4.4 Frontend: App.js.....	23
4.5 Frontend: index.js.....	24
4.6 Frontend: .gitignore (Frontend).....	24
4.7 Frontend: pasta public/.....	24
4.8 package.json.....	25
4.9 package-lock.json.....	25
5. Requisitos Funcionais.....	26
6. Telas do UniKanban.....	40
6.1 Login e Cadastro:.....	40
6.2 Quadros.....	40
6.3 Kanban:.....	41
6.4 Usuário:.....	41

1. Introdução

Este documento apresenta a estrutura e as funcionalidades do **UniKanban**, uma aplicação web desenvolvida para gerenciamento de tarefas no estilo Kanban. O sistema é dividido em duas partes principais: **backend** (API RESTful) e **frontend** (interface desenvolvida com React), sendo construído com foco em boas práticas de arquitetura de software, como separação de responsabilidades, modularização e facilidade de manutenção.

Inicialmente, serão detalhadas as tecnologias utilizadas em cada parte do sistema, explicando onde e como foram aplicadas. Em seguida, o documento descreve a estrutura de pastas e arquivos que compõem o projeto, facilitando o entendimento do seu funcionamento interno.

Também são apresentados os **requisitos funcionais** com os respectivos endpoints REST implementados na API, acompanhados de descrições e exemplos práticos de uso. Por fim, o documento inclui as telas desenvolvidas no frontend para ilustrar a experiência do usuário na aplicação.

Para implementar o projeto localmente, acesse o repositório oficial: <https://github.com/LuanGFRicardo/kanban>. Informações adicionais estão disponíveis no arquivo README.md.

2. Tecnologias Utilizadas

O projeto UniKanban foi desenvolvido utilizando um conjunto moderno de tecnologias voltadas à criação de aplicações web completas e seguras. No backend, a base foi construída com Node.js e Express.js, oferecendo uma estrutura leve e eficiente para criar APIs RESTful. O gerenciamento dos dados ficou por conta do Sequelize ORM, que facilita o mapeamento objeto-relacional com o banco de dados SQLite, escolhido por sua leveza durante o desenvolvimento. Para garantir a segurança no controle de acesso, foi implementada autenticação baseada em JWT (JSON Web Token), protegendo rotas sensíveis e associando ações a usuários autenticados.

No frontend, o sistema conta com React.js como biblioteca principal para a criação de interfaces interativas. A navegação entre páginas é feita com React Router DOM, enquanto as requisições à API são tratadas com Axios, garantindo uma comunicação fluida entre cliente e servidor. A biblioteca React-Bootstrap foi integrada para fornecer componentes estilizados e responsivos, acelerando o desenvolvimento da interface. Além disso, foi adotado CSS modularizado, o que permite uma melhor organização dos estilos e evita conflitos entre componentes. Cada uma dessas tecnologias foi cuidadosamente inserida nas respectivas camadas do projeto, promovendo uma estrutura bem organizada, escalável e fácil de manter.

3. Back-end

3.1 Estrutura de Pastas do Backend

src/

|— config/

| |— dbConnect.js // Conexão com o banco de dados

| |— jsonSecret.js // Segredo usado para geração de tokens JWT

|— controllers/

| |— authController.js // Controle de autenticação (login)

| |— colunasController.js // Controle das colunas

| |— quadrosController.js // Controle dos quadros

| |— tarefasController.js // Controle das tarefas

| |— usuariosController.js // Controle dos usuários

|— dtos/

| |— colunaDTO.js // Data Transfer Object de Coluna

| |— quadroDTO.js // DTO de Quadro

| |— tarefaDTO.js // DTO de Tarefa

| |— usuarioDTO.js // DTO de Usuário

|— middleware/

| |— authMiddleware.js // Middleware de autenticação para proteger rotas

|— models/

| |— Coluna.js // Modelo da tabela de colunas

| |— Quadro.js // Modelo da tabela de quadros

| |— Tarefa.js // Modelo da tabela de tarefas

| |— Usuario.js // Modelo da tabela de usuários

| |— index.js // Associação entre os modelos

```
|— repositories/
|   |— baseRepository.js    // Repositório base reutilizável
|   |— colunaRepository.js  // Repositório para colunas
|   |— quadroRepository.js  // Repositório para quadros
|   |— tarefaRepository.js  // Repositório para tarefas
|   └─ usuarioRepository.js // Repositório para usuários
|— routes/
|   |— authRoutes.js        // Rotas de autenticação
|   |— colunasRoutes.js     // Rotas de colunas
|   |— quadrosRoutes.js     // Rotas de quadros
|   |— tarefaRoutes.js      // Rotas de tarefas
|   └─ usuarioRoutes.js     // Rotas de usuários
|— services/
|   |— authService.js       // Lógica de autenticação
|   |— colunaService.js     // Lógica de negócio das colunas
|   |— quadroService.js     // Lógica de negócio dos quadros
|   |— tarefaService.js     // Lógica de negócio das tarefas
|   └─ usuarioService.js    // Lógica de negócio dos usuários
|— app.js                  // Configuração principal da aplicação
└─ server.js               // Inicialização do servidor

.gitignore                 // Arquivos e pastas ignorados pelo Git
database.sqlite            // Banco de dados SQLite
package.json               // Dependências e scripts do projeto
package-lock.json          // Versões exatas das dependências
```

3.2 Backend: Módulo config/

3.2.1 dbConnect.js

Responsável por configurar a conexão com o banco de dados utilizando Sequelize.

3.2.2 jsonSecret.js

Armazena a chave secreta utilizada para geração e validação de JWTs.

3.3 Backend: Módulo controllers/

3.3.1 AuthController

Responsável por autenticar usuários via email e senha, retornando o token JWT ao final.

- login(req, res) – Recebe email e senha e retorna um JWT válido ao usuário autenticado.

3.3.2 ColunaController

Gerencia todas as operações CRUD relacionadas às colunas de um quadro Kanban.

- createColuna(req, res) – Cria nova coluna associada a um quadro.
- getAllColunas(req, res) – Lista todas as colunas existentes.
- getColunaById(req, res) – Busca uma coluna pelo ID.
- updateColuna(req, res) – Atualiza uma coluna existente.
- deleteColuna(req, res) – Remove uma coluna pelo ID.
- searchColunaByNome(req, res) – Busca colunas por nome.

3.3.3 QuadroController

Responsável pelas operações dos quadros vinculados a usuários autenticados.

- createQuadro(req, res) – Cria um novo quadro ligado ao ID do usuário extraído do token.
- getAllQuadros(req, res) – Lista todos os quadros de um usuário.

- getQuadroById(req, res) – Retorna quadro específico pelo ID.
- updateQuadro(req, res) – Atualiza os dados de um quadro.
- deleteQuadro(req, res) – Exclui quadro específico.
- searchQuadroByNome(req, res) – Busca quadros por nome.

3.3.4 TarefaController

Gerencia tarefas vinculadas a colunas e quadros. Permite criar, buscar, atualizar, excluir e filtrar tarefas por título.

- createTarefa(req, res) – Cria nova tarefa com validações de coluna e quadro.
- getAllTarefas(req, res) – Lista todas as tarefas do sistema.
- getTarefaById(req, res) – Busca tarefa específica.
- updateTarefa(req, res) – Atualiza dados da tarefa.
- deleteTarefa(req, res) – Exclui uma tarefa existente.
- searchTarefaByTitulo(req, res) – Busca tarefas com título correspondente.

3.3.5 UsuarioController

Controlador responsável pelas operações CRUD de usuários do sistema.

- createUsuario(req, res) – Cria novo usuário.
- getAllUsuarios(req, res) – Lista todos os usuários cadastrados.
- getUsuarioById(req, res) – Retorna usuário específico.
- updateUsuario(req, res) – Atualiza os dados do usuário.
- deleteUsuario(req, res) – Remove usuário do sistema.
- searchUsuarioByNome(req, res) – Busca usuários por nome.

3.4 Backend: Módulo dtos/

Os DTOs (Data Transfer Objects) são responsáveis por estruturar e transferir dados entre as camadas da aplicação, garantindo encapsulamento e controle sobre o que é exposto nas respostas da API.

3.4.1 ColunaDto

Representa a estrutura de dados de uma Coluna para entrada (via fromRequest) e saída (via constructor).

3.4.2 QuadroDto

DTO utilizado para transferir os dados de um quadro.

3.4.3 TarefaDto

Define os dados utilizados na manipulação de tarefas.

3.4.4 UsuarioDto

Representa um usuário do sistema com os dados básicos.

3.5 Backend: Módulo middleware/

O módulo middleware/ contém funções intermediárias que tratam a autenticação e autorização nas rotas protegidas.

3.5.1 authMiddleware.js

Middleware responsável por validar o token JWT enviado no cabeçalho Authorization.

Responsabilidades:

- Verifica se o token está presente e tem formato Bearer <token>.
- Valida o token com a chave secreta.
- Anexa userId e userEmail ao objeto req se o token for válido.
- Retorna erro 401 se o token estiver ausente ou inválido.

Utilizado para proteger rotas sensíveis que exigem autenticação.

3.6 Backend: Módulo models/

Os modelos representam as entidades do sistema e são definidos usando Sequelize. Cada arquivo define a estrutura de uma tabela no banco de dados SQLite, e os relacionamentos são organizados no index.js.

3.6.1 Coluna.js

Define a tabela Coluna com os campos:

- id: chave primária auto-incremento
- nome: texto obrigatório
- ordem: número que representa a posição da coluna
- quadroid: chave estrangeira que referencia o quadro ao qual pertence

3.6.2 Quadro.js

Define a tabela Quadro com os campos:

- id: chave primária auto-incremento
- nome: texto obrigatório
- usuarioid: chave estrangeira obrigatória para o dono do quadro
- Timestamps ativados (createdAt, updatedAt)

3.6.3 Tarefa.js

Define a tabela Tarefa, que representa uma tarefa dentro de uma coluna de um quadro:

- titulo: texto obrigatório
- descricao: texto opcional
- status: texto opcional (poderia ser tratado como ENUM)
- quadroid: chave estrangeira obrigatória
- colunaid: chave estrangeira obrigatória
- Timestamps ativados (createdAt, updatedAt)

3.6.4 Usuario.js

Define a tabela Usuario, que representa os usuários do sistema:

- nome: texto obrigatório
- email: texto obrigatório e único
- senha: texto obrigatório

3.6.5 index.js – Relacionamentos

Define os relacionamentos entre os modelos:

- Coluna.hasMany(Tarefa) e Tarefa.belongsTo(Coluna) → uma coluna contém várias tarefas
- Quadro.hasMany(Coluna) e Coluna.belongsTo(Quadro) → um quadro contém várias colunas
- Usuario.hasMany(Quadro) e Quadro.belongsTo(Usuario) → um usuário possui vários quadros

Essas definições garantem que o Sequelize sincronize as tabelas corretamente e propaguem ações como delete e update em cascata (CASCADE).

3.7 Backend: Módulo repositories/

Os repositórios encapsulam a lógica de acesso aos dados utilizando Sequelize, oferecendo uma interface reutilizável e testável para as operações com o banco. Cada entidade possui seu próprio repositório específico, herdando funcionalidades comuns da BaseRepository.

3.7.1 BaseRepository.js

Classe genérica que centraliza os métodos CRUD utilizados por todos os repositórios:

- create(data) – Cria um novo registro.
- findById(id) – Retorna um item pelo seu ID.

- `findAll(filter = {})` – Lista todos os itens, com possibilidade de aplicar filtros.
- `update(id, data)` – Atualiza os dados de um item pelo ID.
- `delete(id)` – Remove um item pelo ID.
- `search(query)` – Busca registros com base em uma cláusula where.

3.7.2 ColunaRepository.js

Especialização da `BaseRepository` para o modelo Coluna. Possui método adicional:

- `searchByNome(nome)` – Retorna colunas cujo nome contenha o texto informado (busca parcial com LIKE).

3.7.3 QuadroRepository.js

Extensão da `BaseRepository` para lidar com quadros. Inclui:

- `findByUsuarioid(usuarioid)` – Retorna todos os quadros de um determinado usuário.

3.7.4 TarefaRepository.js

Repositório especializado para tarefas, com filtros úteis para listagens específicas:

- `searchByTitulo(titulo)` – Retorna tarefas cujo título contenha o texto informado.
- `searchByStatus(status)` – Filtra tarefas com status semelhante ao informado.

3.7.5 UsuarioRepository.js

Repositório para a entidade `Usuario`, com funcionalidade específica de busca:

- `searchByNome(nome)` – Lista usuários cujo nome corresponda parcialmente ao termo buscado.

3.8 Backend: Módulo routes/

As rotas são responsáveis por definir os caminhos da API e vincular as requisições HTTP aos respectivos controladores. Cada entidade possui seu próprio arquivo de rotas. Algumas rotas são protegidas com middleware de autenticação JWT (authMiddleware), garantindo que apenas usuários autenticados possam acessá-las.

3.8.1 authRoutes.js

Rota dedicada à autenticação de usuários.

- POST /auth/login – Realiza o login com email e senha, retornando um token JWT se as credenciais forem válidas.

3.8.2 colunasRoutes.js

Rotas relacionadas à entidade Coluna.

- POST / – Cria uma nova coluna.
- GET / – Lista todas as colunas.
- GET /:id – Retorna uma coluna específica pelo ID.
- PUT /:id – Atualiza os dados de uma coluna.
- DELETE /:id – Remove uma coluna.
- GET /search/:nome – Busca colunas pelo nome.

3.8.3 quadrosRoutes.js

Rotas protegidas por autenticação relacionadas aos quadros de um usuário.

- POST / – Cria um novo quadro vinculado ao usuário autenticado.
- GET /search/:nome – Busca quadros pelo nome.
- GET / – Lista todos os quadros do usuário autenticado.
- GET /:id – Retorna um quadro específico.
- PUT /:id – Atualiza os dados de um quadro.
- DELETE /:id – Remove um quadro.

Todas as rotas estão protegidas com `authMiddleware`.

3.8.4 tarefaRoutes.js

Define os endpoints para gerenciar tarefas.

- POST / – Cria uma nova tarefa.
- GET / – Lista todas as tarefas.
- GET /:id – Retorna uma tarefa pelo ID.
- PUT /:id – Atualiza os dados de uma tarefa.
- DELETE /:id – Exclui uma tarefa.
- GET /search/:titulo – Busca tarefas por título.

3.8.5 usuarioRoutes.js

Rotas voltadas ao gerenciamento de usuários, com exceção da criação (aberta).

- POST / – Cria um novo usuário.
- GET / – Lista todos os usuários (protegida).
- GET /:id – Retorna um usuário específico (protegida).
- PUT /:id – Atualiza os dados de um usuário (protegida).
- DELETE /:id – Remove um usuário (protegida).
- GET /search/:nome – Busca usuários por nome (protegida).

3.9 Backend: Módulo services/

3.9.1 AuthService

Executa a autenticação de usuários com email e senha. Utiliza `bcryptjs` para comparação de senhas e `jsonwebtoken` para geração de tokens.

3.9.2 ColunaService

Contém a lógica de negócio associada às colunas, como validação da existência de quadros e manipulação de dados via repositório.

3.9.3 QuadroService

Garante que um quadro seja criado apenas se o usuário existir. Gerencia quadros por ID ou por usuário.

3.9.4 TarefaService

Valida a existência de quadros e colunas antes de criar ou atualizar uma tarefa. Permite buscas por título.

3.9.5 UsuarioService

Executa operações relacionadas a usuários, incluindo criptografia de senhas antes do armazenamento.

Segue a seção da documentação referente ao arquivo app.js:

3.10 Backend: Arquivo app.js

O arquivo app.js é o ponto central de configuração da aplicação backend. Nele são inicializados os middlewares globais, a conexão com o banco de dados e as rotas principais da API.

Principais responsabilidades:

- **Importação de bibliotecas essenciais:** como express, cors, dotenv, e o ORM sequelize.
- **Carregamento automático de modelos:** via models/index.js, o que garante que os relacionamentos entre entidades sejam registrados antes do uso das rotas.
- **Configuração de middlewares globais:**
 - express.json() para leitura de requisições com corpo JSON.
 - cors() para permitir acesso à API de origens diferentes (Cross-Origin Resource Sharing).
- **Sincronização do banco de dados** com os modelos Sequelize usando db.sync().
- **Registro das rotas** do sistema, organizadas por entidade:

- /api/tarefas → Rotas de tarefas
- /api/colunas → Rotas de colunas
- /api/quadros → Rotas de quadros
- /api/usuarios → Rotas de usuários
- /api/auth/login → Autenticação

Este arquivo exporta a instância configurada do Express (app) para ser utilizada no arquivo server.js, que efetivamente sobe o servidor.

3.11 Backend: Arquivo server.js

O arquivo server.js é o ponto de entrada da aplicação backend. Sua principal responsabilidade é iniciar o servidor da API e escutá-lo em uma porta específica.

Principais responsabilidades:

- **Importar a aplicação configurada** no app.js, que já está com todos os middlewares, rotas e conexões de banco preparados.
- **Definir a porta de escuta** da aplicação, no caso, 3000.
- **Inicializar o servidor** utilizando app.listen(PORT, callback), exibindo uma mensagem no console assim que a aplicação estiver pronta para receber requisições.

3.12 Backend: Arquivo .gitignore

O .gitignore é utilizado para informar ao Git quais arquivos e diretórios devem ser ignorados no versionamento. No projeto **UniKanban**, foram definidos os seguintes itens:

Itens ignorados:

- **node_modules/**
Diretório que contém todas as dependências do projeto instaladas via

NPM. Como ele pode ser reconstruído com `npm install`, não deve ser versionado.

- **database.sqlite**

Arquivo do banco de dados local, utilizado para testes e desenvolvimento. Pode conter dados sensíveis ou irrelevantes para o controle de versão.

- **Arquivos de log:**

- `npm-debug.log*`
- `yarn-debug.log*`
- `yarn-error.log*`

Esses arquivos são gerados automaticamente em casos de erro de execução e não precisam ser salvos no repositório.

- **.DS_Store**

Arquivo oculto gerado automaticamente por sistemas macOS, que não possui utilidade para o projeto.

3.13 Backend: Arquivo database.sqlite

O arquivo `database.sqlite` é o banco de dados local da aplicação, gerado automaticamente pelo Sequelize.

Características:

- Armazena as tabelas e dados definidos nos modelos (`models/`).
- É ideal para desenvolvimento e testes por ser leve e fácil de configurar.
- Não é versionado no Git (está listado no `.gitignore`).

3.14 Backend: Arquivo package.json

O `package.json` é o manifesto da aplicação Node.js. Ele contém:

- **Nome e versão do projeto**
- **Scripts úteis** (como `start`, `dev`)

- **Dependências e devDependencies** instaladas
- **Configuração de entrada** (ex: main: server.js)

3.15 Backend: Arquivo package-lock.json

Esse arquivo é gerado automaticamente pelo NPM ao instalar pacotes. Ele registra:

- Versões exatas das dependências e subdependências
- Hashes para verificação de integridade
- Garantia de consistência entre ambientes

Ele não deve ser editado manualmente. Deve ser versionado junto com o projeto.

4. Front-end

4.1 Estrutura de Pastas do Frontend:

kanban-frontend/

```

├── public/
|   ├── favicon.ico    // Ícone da aba do navegador
|   ├── index.html     // HTML principal onde o React será montado
|   ├── logo192.png    // Logo padrão (usado em PWA)
|   ├── logo512.png    // Logo maior (PWA e instalação)
|   ├── manifest.json  // Configuração de Progressive Web App
|   └── robots.txt     // Instruções para indexação de robôs (SEO)
├── src/
|   ├── components/
|   |   ├── CardTarefa.js    // Componente visual de um card de tarefa
|   |   └── ModalNovaTarefa.js // Modal para criar uma nova tarefa

```

```
| | └─ ModalTarefa.js    // Modal para visualizar/editar tarefa
| | └─ Navbar.js        // Barra de navegação principal
| | └─ PrivateRoute.js  // Componente que protege rotas privadas
| └─ pages/
| | └─ Login.js         // Tela de login
| | └─ QuadroPage.js    // Página que exibe os quadros
| | └─ tarefaPage.js    // Página com tarefas organizadas por coluna
| | └─ UsuarioPage.js   // Página de gerenciamento de usuários
| └─ App.js             // Componente raiz com as rotas do app
└─ index.js             // Ponto de entrada da aplicação React
└─ .gitignore           // Arquivos/pastas ignorados pelo Git
└─ package.json         // Dependências e scripts do projeto
└─ package-lock.json    // Versões exatas das dependências
```

4.2 Frontend: Módulo components/

O diretório components/ centraliza todos os componentes reutilizáveis da interface do sistema, responsáveis por layout, comportamento visual e controle de fluxo. São usados em múltiplas páginas.

4.2.1 CardTarefa.js

Componente visual para exibição individual de uma tarefa em forma de card.

Responsabilidades:

- Apresentar o título, descrição e status da tarefa.
- Detectar cliques para abrir o modal de detalhes ou edição.

4.2.2 ModalNovaTarefa.js

Modal utilizado para cadastrar uma nova tarefa.

Responsabilidades:

- Recebe título, descrição e status da tarefa via formulário.
- Reseta os campos ao fechar.
- Dispara onCreate ao submeter.

Campos:

- Título (input)
- Descrição (textarea)
- Status (select com valores como *Aguardando*, *Concluído*, etc.)

4.2.3 ModalTarefa.js

Modal para visualizar e editar uma tarefa existente.

Responsabilidades:

- Mostra os dados da tarefa selecionada.
- Permite edição de título, descrição e status.
- Dispara onSave ou onDelete conforme ação do usuário.

4.2.4 Navbar.js

Barra de navegação presente nas páginas autenticadas.

Funcionalidades:

- Botão de voltar (quando onVoltar é passado)
- Nome da aplicação com link para /quadros
- Link de perfil direcionando para /usuario

Utiliza componentes do React Bootstrap e ícones do react-bootstrap-icons.

4.2.5 PrivateRoute.js

Componente que protege rotas privadas. Permite acesso apenas se houver token JWT válido salvo no localStorage.

Funcionamento:

- Se existir token, renderiza o componente filho.
- Caso contrário, redireciona para /login.

4.3 Frontend: Módulo pages/

As páginas representam as views principais do sistema. Cada página é responsável por orquestrar os componentes, interagir com a API e manter o estado da aplicação.

4.3.1 Login.js

Página que permite:

- **Realizar login** com email e senha
- **Cadastrar um novo usuário**

Funcionalidades:

- Armazena o token JWT no localStorage após login bem-sucedido
- Exibe mensagens de sucesso/erro com feedback visual
- Redireciona o usuário para a página de quadros após login

4.3.2 QuadroPage.js

Página principal que exibe todos os **quadros** do usuário autenticado.

Funcionalidades:

- Busca, cria, edita e remove quadros
- Exibe quadros em cards clicáveis
- Ao clicar em um quadro, redireciona para TarefaPage com as tarefas correspondentes
- Usa AppNavbar no topo

4.3.3 TarefaPage.js

Página de gerenciamento das tarefas dentro de um quadro, organizada em **colunas estilo Kanban**.

Funcionalidades:

- Adiciona, edita, deleta tarefas e colunas
- Arrasta tarefas entre colunas com react-beautiful-dnd
- Utiliza os modais ModalNovaTarefa e ModalTarefa para manipulação dos dados
- Permite edição online do nome das colunas
- Exibe feedback em tempo real com mensagens

4.3.4 UsuarioPage.js

Página de perfil do usuário.

Funcionalidades:

- Carrega dados do usuário via jwtDecode
- Permite editar nome, email e senha
- Realiza atualização no banco com autenticação JWT
- Usa AppNavbar com botão de voltar

4.4 Frontend: App.js

Arquivo responsável por configurar as rotas principais da aplicação usando React Router DOM.

Funcionalidades:

- Encapsula as rotas dentro de um Router
- Redireciona a rota raiz / para /login
- Define as rotas públicas:
 - /login: tela de autenticação
 - /quadros: lista de quadros do usuário

- /usuario: dados do perfil
- Define rota protegida:
 - /tarefas: acessível apenas se o usuário estiver autenticado (PrivateRoute)

4.5 Frontend: index.js

Ponto de entrada da aplicação React. Aqui ocorre a montagem da aplicação na árvore DOM do HTML.

Responsabilidades:

- Importa React e ReactDOM
- Aplica o CSS do Bootstrap para estilização global
- Renderiza o componente `<App />` dentro do elemento com ID root do index.html (localizado na pasta public)
- Utiliza `React.StrictMode` para destacar problemas de ciclo de vida e práticas inseguras

4.6 Frontend: .gitignore (Frontend)

Define os arquivos e diretórios que devem ser ignorados pelo Git durante o versionamento.

4.7 Frontend: pasta public/

A pasta `public/` contém os arquivos estáticos e de configuração base da aplicação React:

- index.html: ponto de montagem da aplicação React (div id="root")
- favicon.ico: ícone da aba do navegador
- logo192.png e logo512.png: ícones para Progressive Web Apps

- manifest.json: configurações do PWA
- robots.txt: instruções para indexação de mecanismos de busca

4.8 package.json

Arquivo central de configuração do projeto React:

- Lista dependências (react, react-dom, react-scripts, axios, react-bootstrap, etc.)
- Define os scripts de execução:
 - start: inicia o servidor de desenvolvimento
 - build: gera versão de produção
 - test: roda os testes (se configurados)
- Armazena metadados do projeto como nome, versão e descrição

4.9 package-lock.json

Arquivo gerado automaticamente ao instalar dependências com npm. Garante que todos os pacotes sejam instalados nas versões exatas, assegurando consistência entre ambientes.

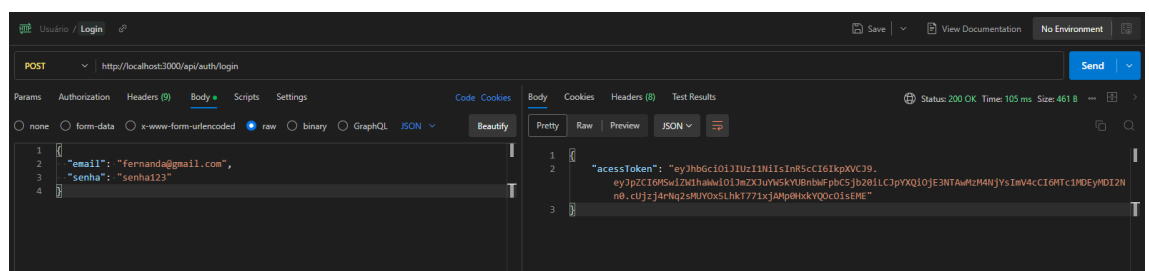
5. Requisitos Funcionais

A seguir estão descritos os **endpoints REST** implementados na aplicação UniKanban:

Autenticação

- **POST /auth/login** – *Login de usuário*

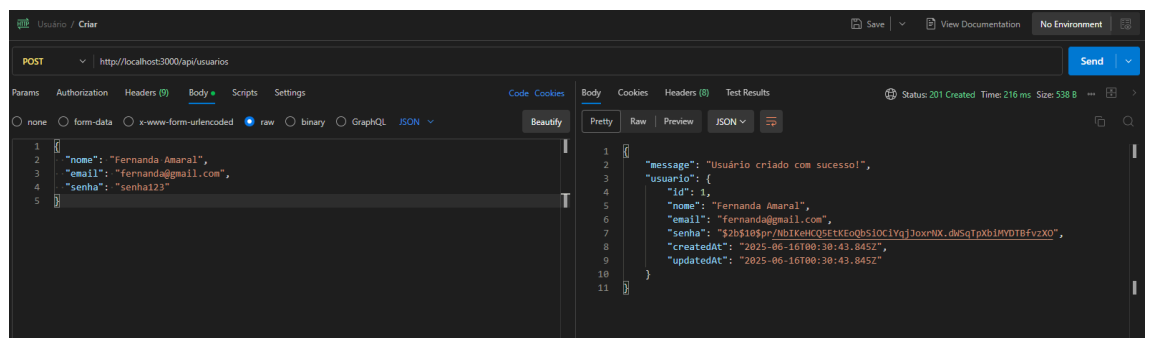
Permite que um usuário se autentique informando email e senha. Em caso de sucesso, retorna um **token JWT** que deve ser utilizado nas próximas requisições protegidas via cabeçalho Authorization: Bearer {token}. Exemplo:



Usuários

- **POST /usuarios** – Criação de usuários

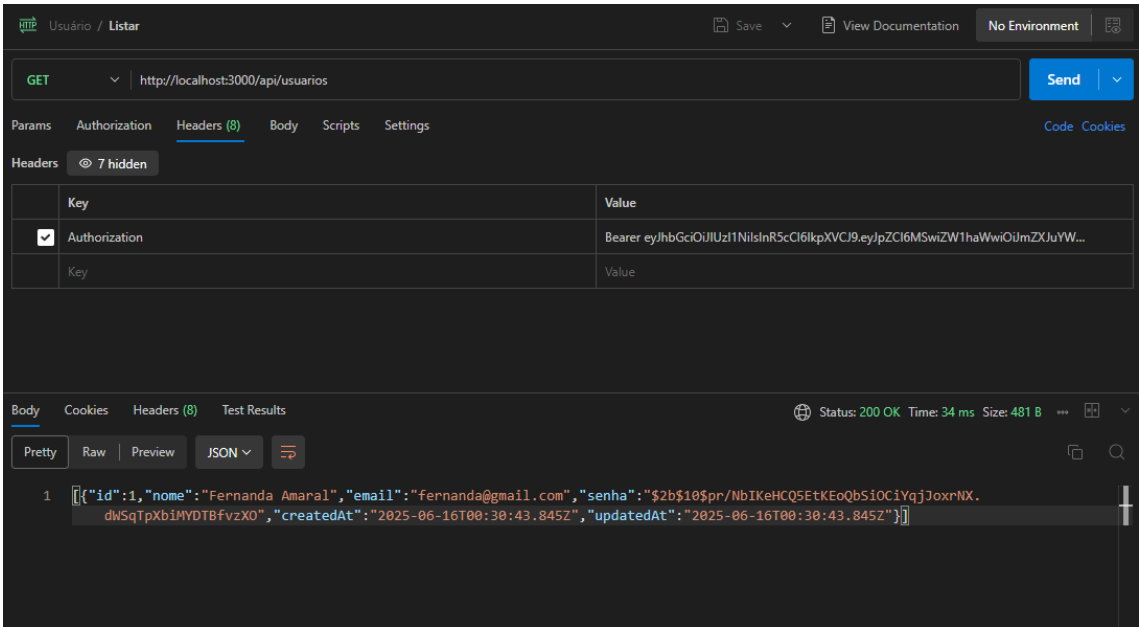
Permite que novos usuários se cadastrem informando nome, email e senha. Exemplo:



- **GET /usuarios** – *Listagem de usuários*

Este endpoint retorna uma lista com todos os usuários cadastrados no sistema. Por questões de segurança, o acesso é **restrito a usuários autenticados**. Para realizar a requisição, é necessário incluir um **token JWT** válido no cabeçalho da requisição. Esse token é obtido ao realizar login. Cabeçalho (Header) necessário:

Key	Value
Authorization	Bearer {seu_token_jwt_aqui}

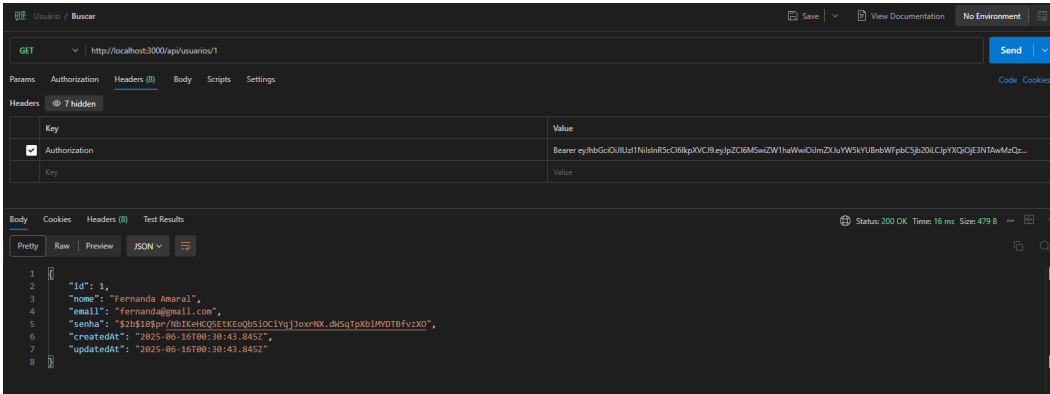


- **GET /usuarios/:id – Detalhes do usuário**

Retorna as informações de um usuário específico a partir de seu ID. Por questões de segurança, o acesso é **restrito a usuários autenticados**. Para realizar a requisição, é necessário incluir um **token JWT** válido no cabeçalho da requisição. Esse token é obtido ao realizar login.

Cabeçalho (Header) necessário:

Key	Value
Authorization	Bearer {seu_token_jwt_aqui}

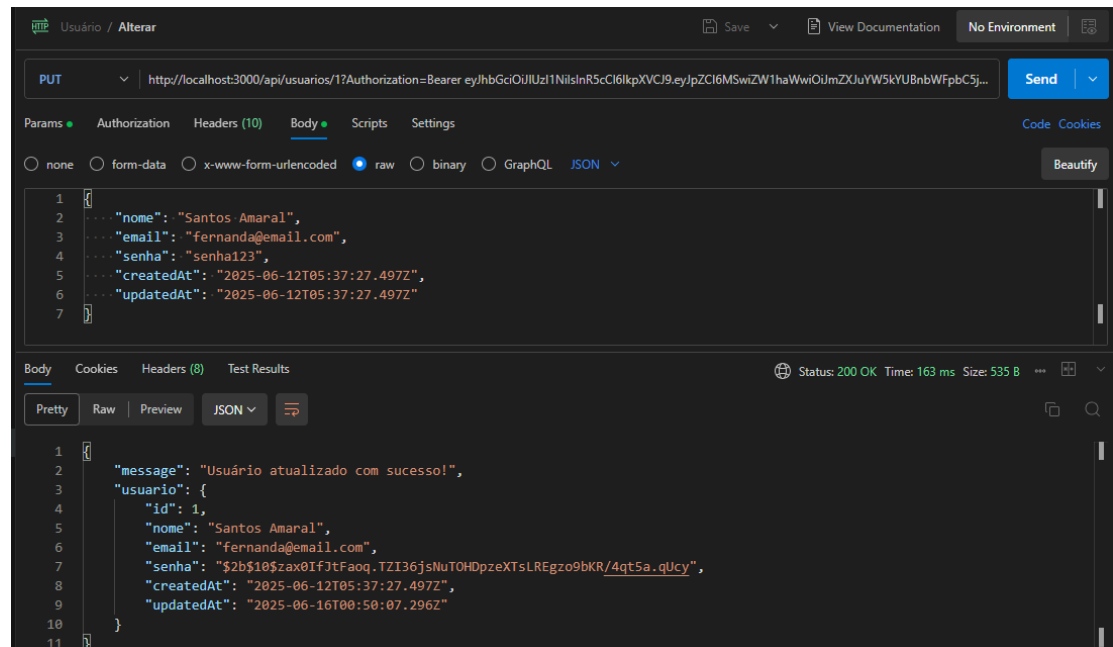


- **PUT /usuarios/:id** – *Atualização de dados*

Permite a edição dos dados de um usuário autenticado. Por questões de segurança, o acesso é **restrito a usuários autenticados**. Para realizar a requisição, é necessário incluir um **token JWT** válido no cabeçalho da requisição. Esse token é obtido ao realizar login. Cabeçalho (Header) necessário:

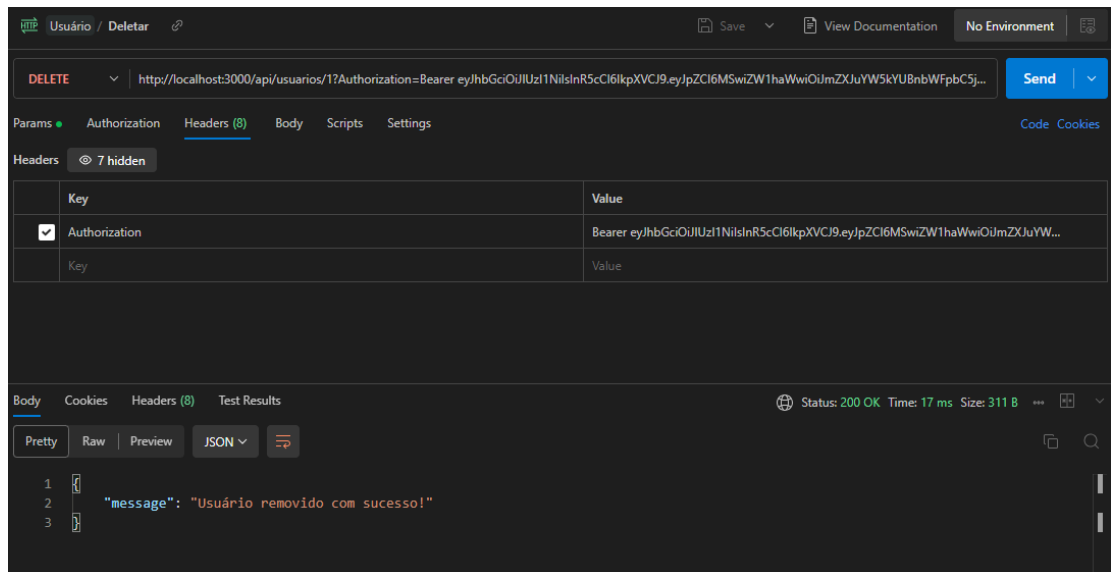
Key	Value
Authorization	Bearer {seu_token_jwt_aqui}

Inserir o JSON:



- **DELETE /usuarios/:id** – *Remoção de usuário*

Remove um usuário do sistema com base no seu ID. Por questões de segurança, o acesso é **restrito a usuários autenticados**. Para realizar a requisição, é necessário incluir um **token JWT** válido no cabeçalho da requisição. Esse token é obtido ao realizar login. Cabeçalho (Header) necessário:



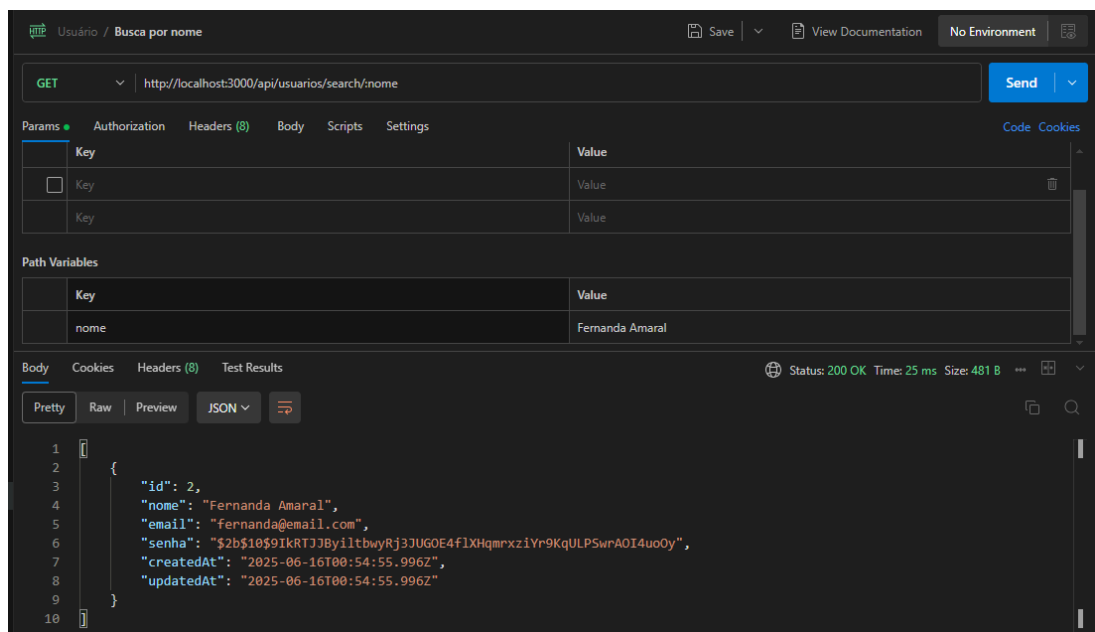
- **GET /usuarios/search/:nome** – *Busca de usuários por nome*

Permite buscar usuários a partir de um termo contido no nome. Por questões de segurança, o acesso é **restrito a usuários autenticados**.

Para realizar a requisição, é necessário incluir um **token JWT** válido no cabeçalho da requisição. Esse token é obtido ao realizar login.

Cabeçalho (Header) necessário:

Key	Value
Authorization	Bearer {seu_token_jwt_aqui}



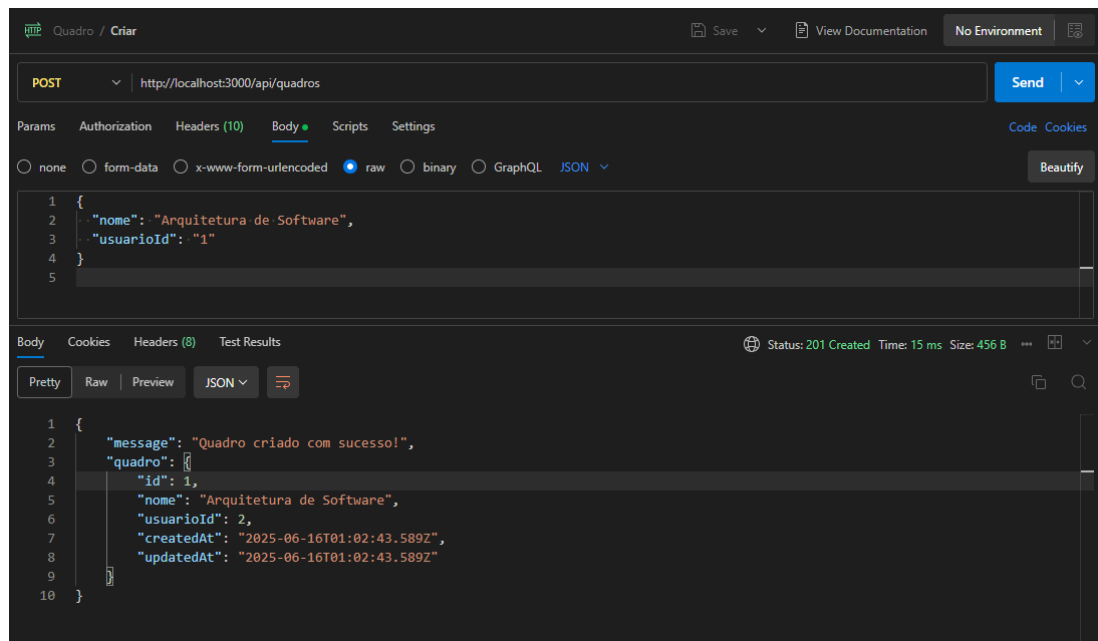
Quadros

- **POST /quadros** – Criação de quadros

Permite que usuários autenticados criem quadros pessoais. Por questões de segurança, o acesso é **restrito a usuários autenticados**. Para realizar a requisição, é necessário incluir um **token JWT** válido no cabeçalho da requisição. Esse token é obtido ao realizar login. Cabeçalho (Header) necessário:

Key	Value
Authorization	Bearer {seu_token_jwt_aqui}

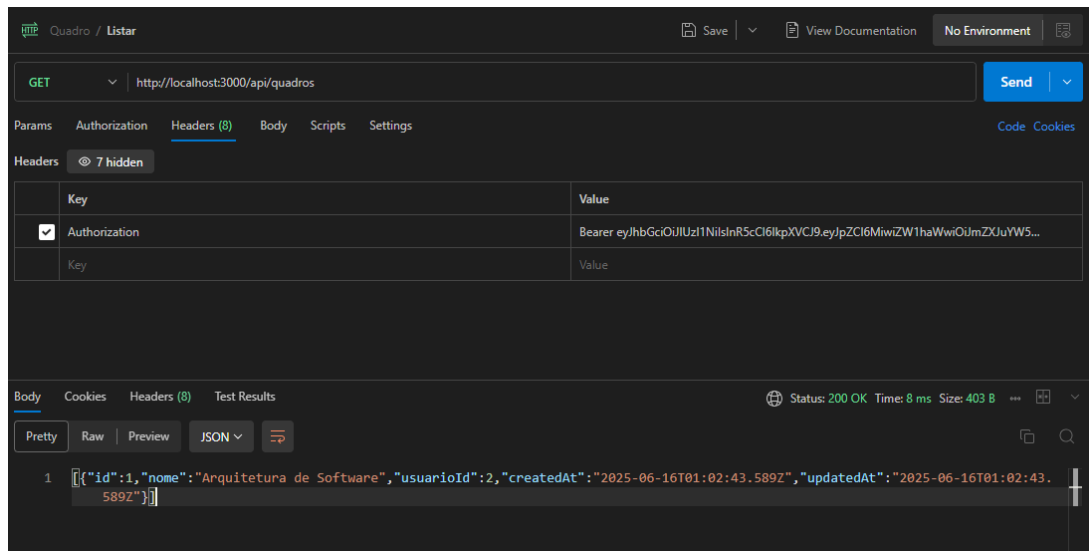
Inserir o JSON:



- **GET /quadros** – Listagem de quadros do usuário

Retorna todos os quadros criados pelo usuário autenticado. Por questões de segurança, o acesso é **restrito a usuários autenticados**. Para realizar a requisição, é necessário incluir um **token JWT** válido no cabeçalho da requisição. Esse token é obtido ao realizar login. Cabeçalho (Header) necessário:

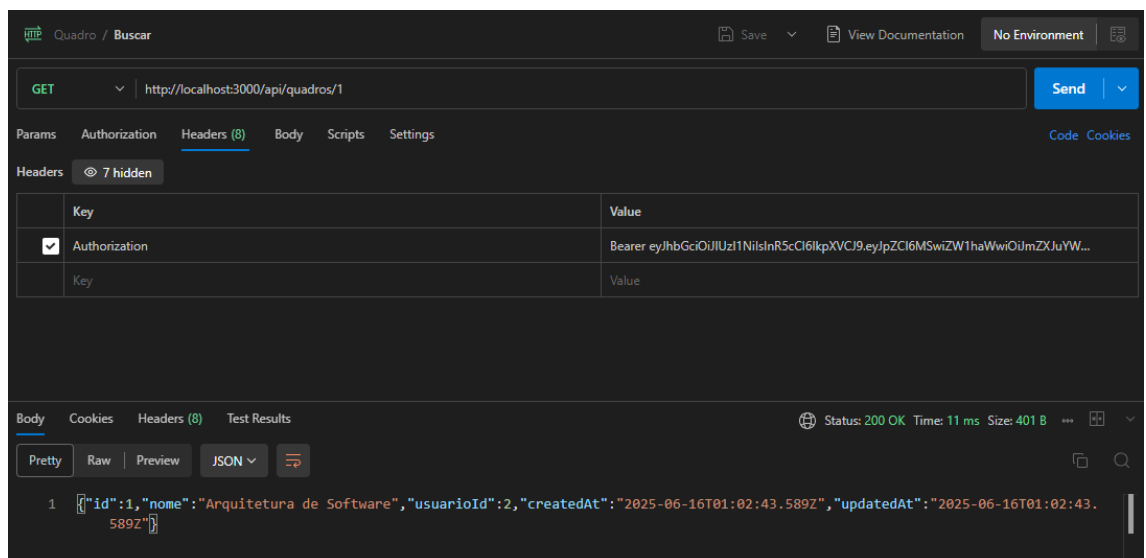
Key	Value
Authorization	Bearer {seu_token_jwt_aqui}



- **GET /quadros/:id – Visualização de quadro específico**

Retorna os dados de um quadro com base em seu ID. Por questões de segurança, o acesso é **restrito a usuários autenticados**. Para realizar a requisição, é necessário incluir um **token JWT** válido no cabeçalho da requisição. Esse token é obtido ao realizar login. Cabeçalho (Header) necessário:

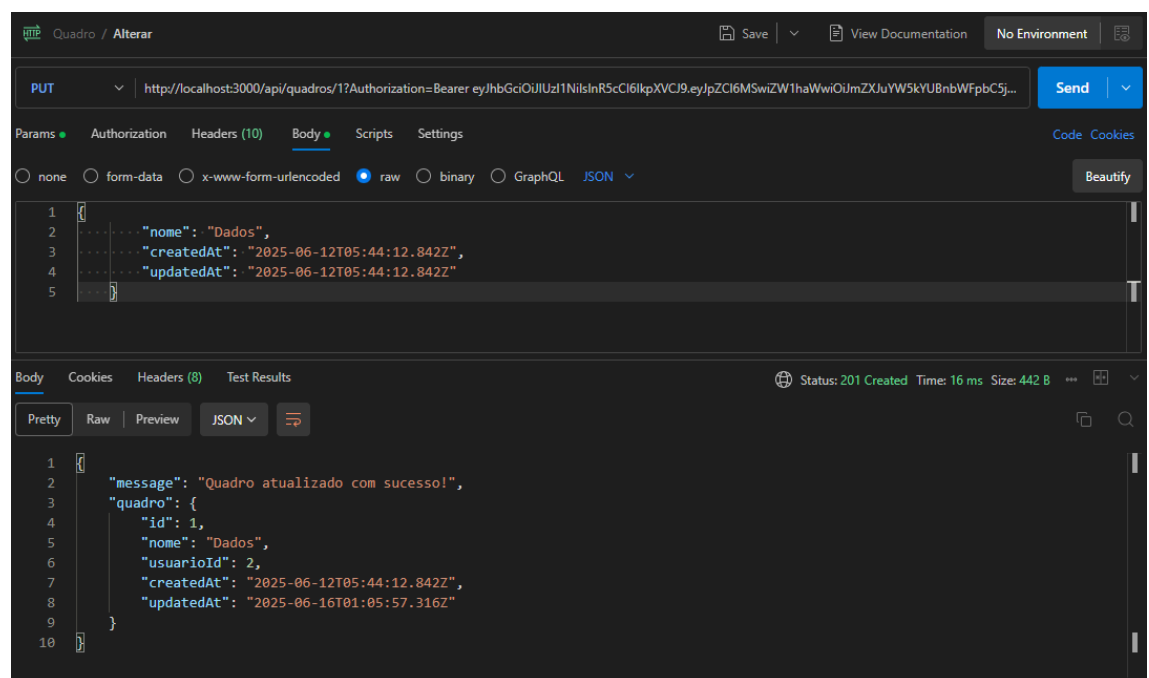
Key	Value
Authorization	Bearer {seu_token_jwt_aqui}



- **PUT /quadros/:id** – Edição de quadros

Permite que o usuário edite o nome de um quadro. Por questões de segurança, o acesso é **restrito a usuários autenticados**. Para realizar a requisição, é necessário incluir um **token JWT** válido no cabeçalho da requisição. Esse token é obtido ao realizar login. Cabeçalho (Header) necessário:

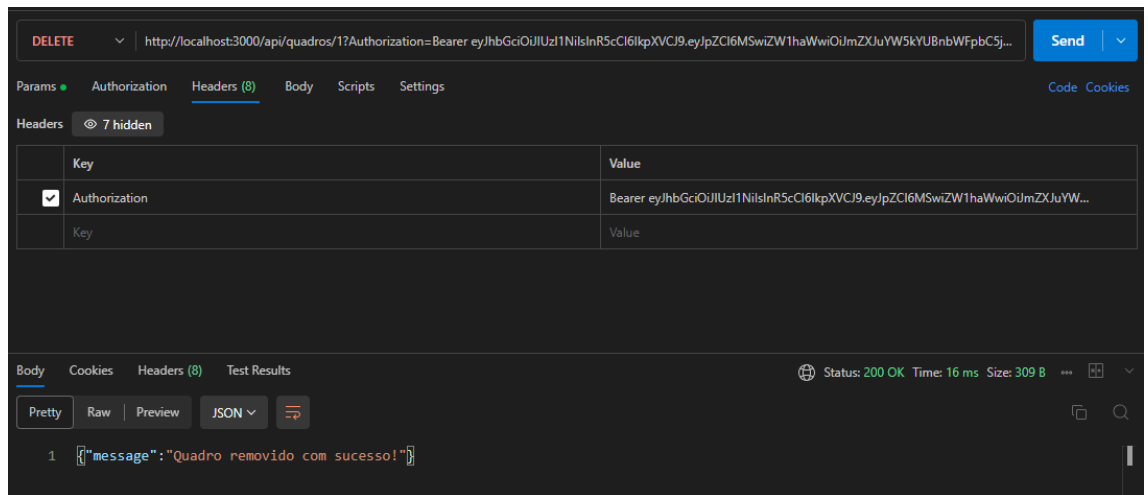
Key	Value
Authorization	Bearer {seu_token_jwt_aqui}



- **DELETE /quadros/:id** – *Exclusão de quadros*

Remove um quadro do sistema e todas suas colunas e tarefas associadas. Por questões de segurança, o acesso é **restrito a usuários autenticados**. Para realizar a requisição, é necessário incluir um **token JWT** válido no cabeçalho da requisição. Esse token é obtido ao realizar login. Cabeçalho (Header) necessário:

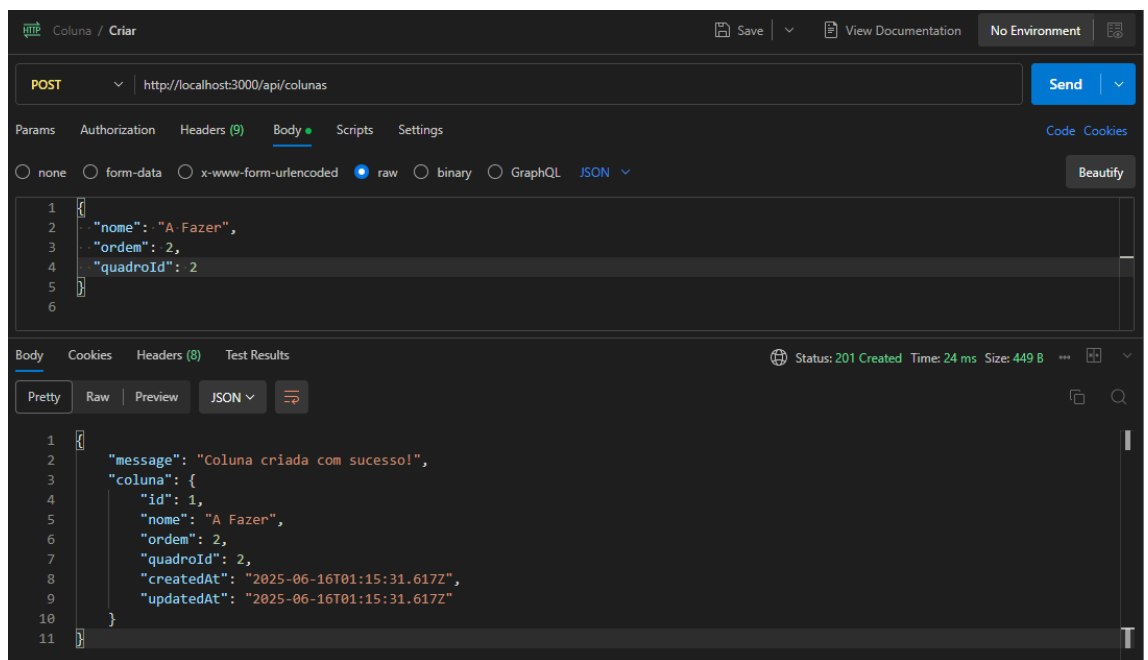
Key	Value
Authorization	Bearer {seu_token_jwt_aqui}



Colunas

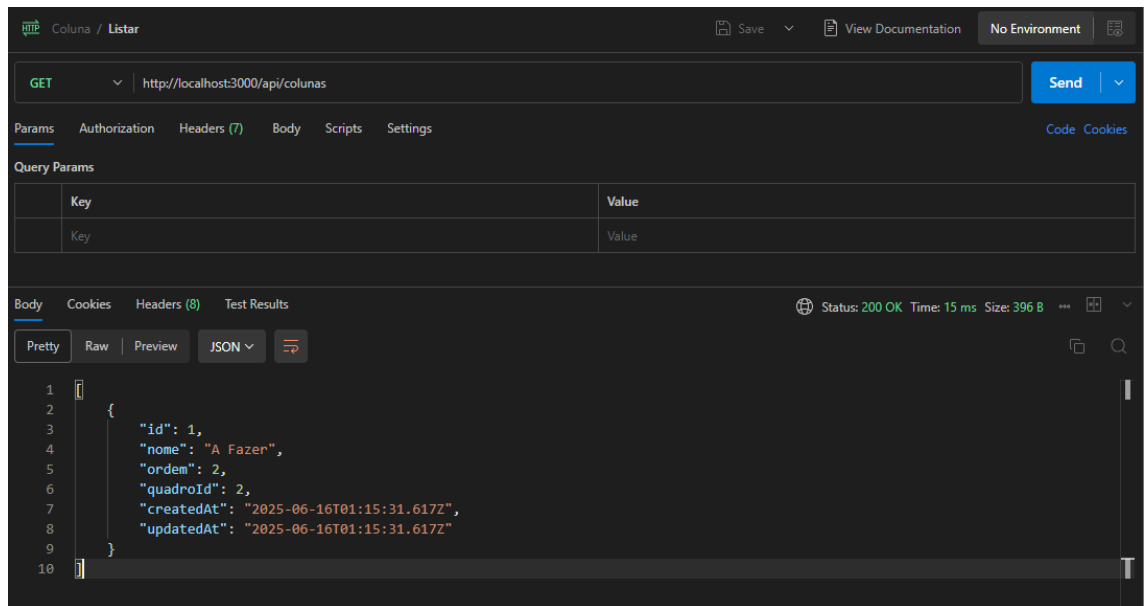
- **POST /colunas** – *Criação de colunas*

Permite adicionar uma nova coluna a um quadro, com nome e ordem de exibição. O id do quadro precisa estar cadastrado.



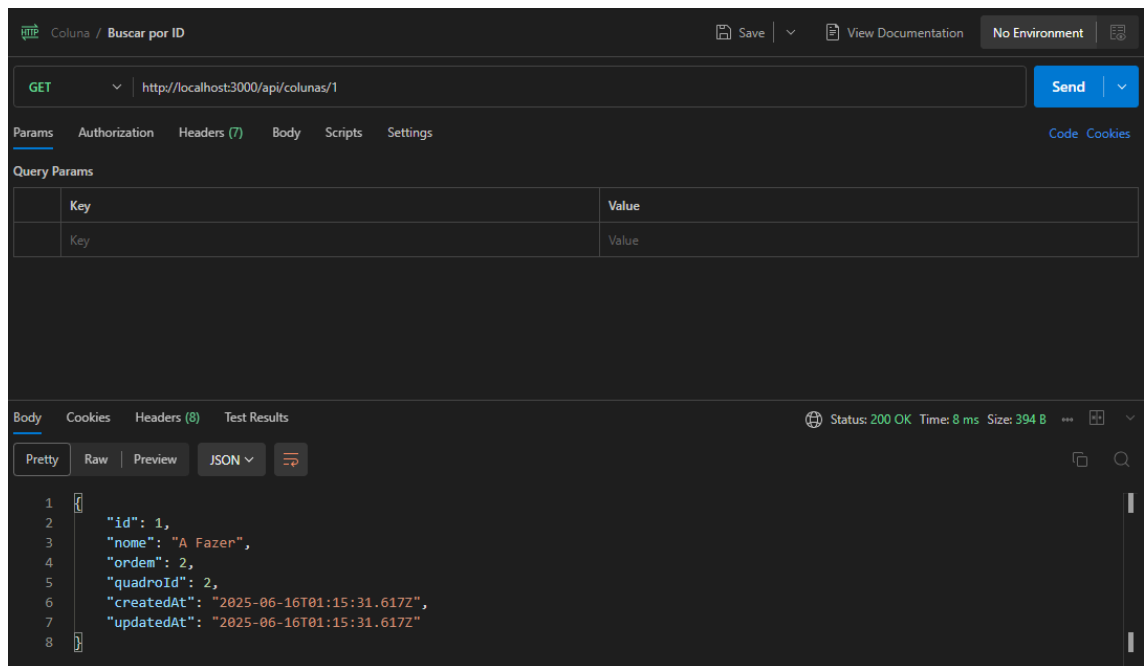
- **GET /colunas** – *Listagem de todas as colunas*

Retorna todas as colunas criadas, podendo ser filtradas por quadro no front-end.



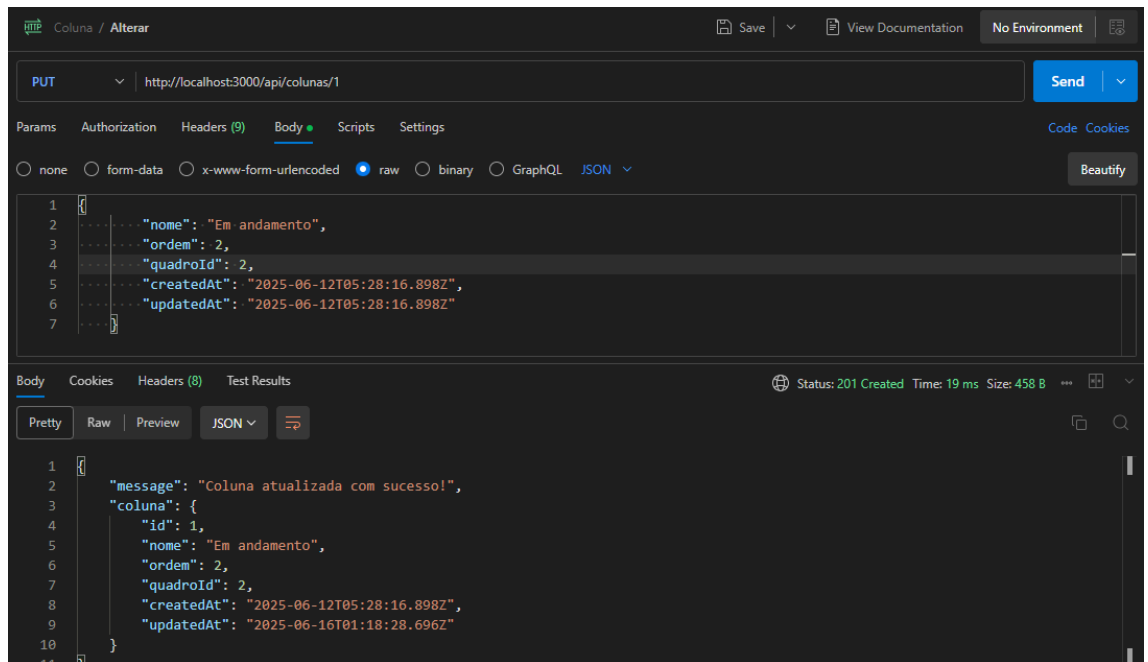
- **GET /colunas/:id – Detalhes de uma coluna**

Retorna os dados de uma coluna específica com base em seu ID.



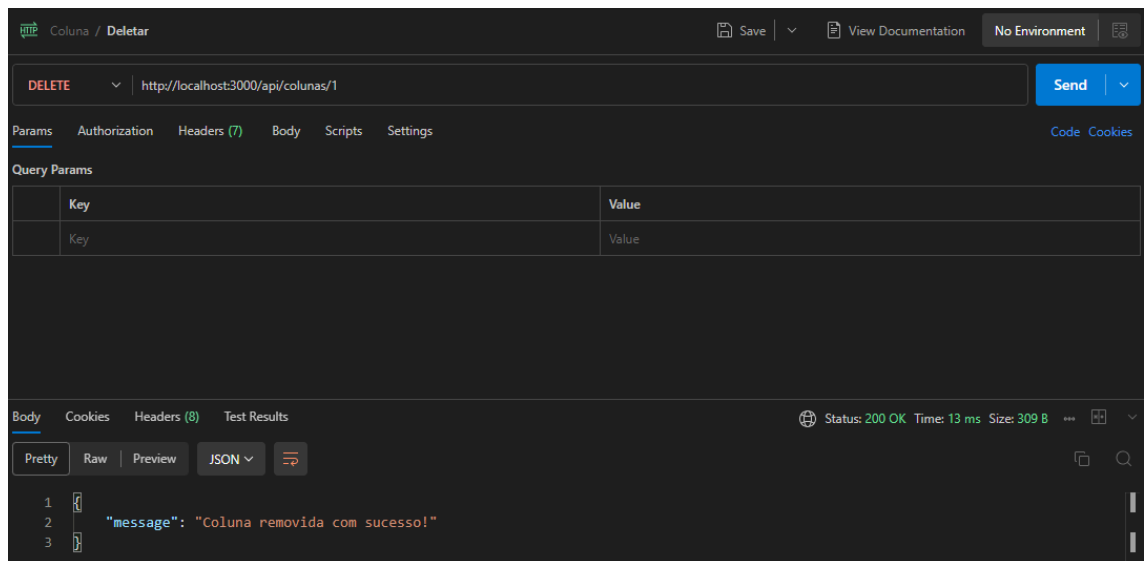
- **PUT /colunas/:id – Atualização de colunas**

Permite editar o nome ou ordem de uma coluna existente.



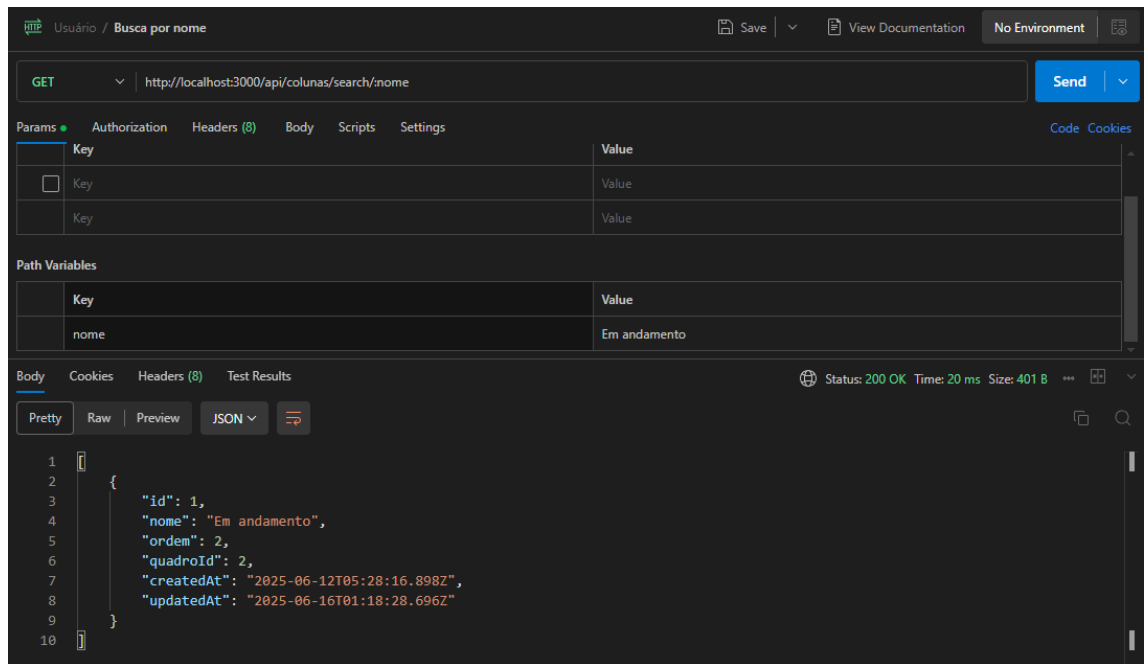
- **DELETE /colunas/:id – Remoção de colunas**

Remove a coluna e todas as tarefas relacionadas a ela.



- **GET /colunas/search/:nome – Busca de colunas por nome**

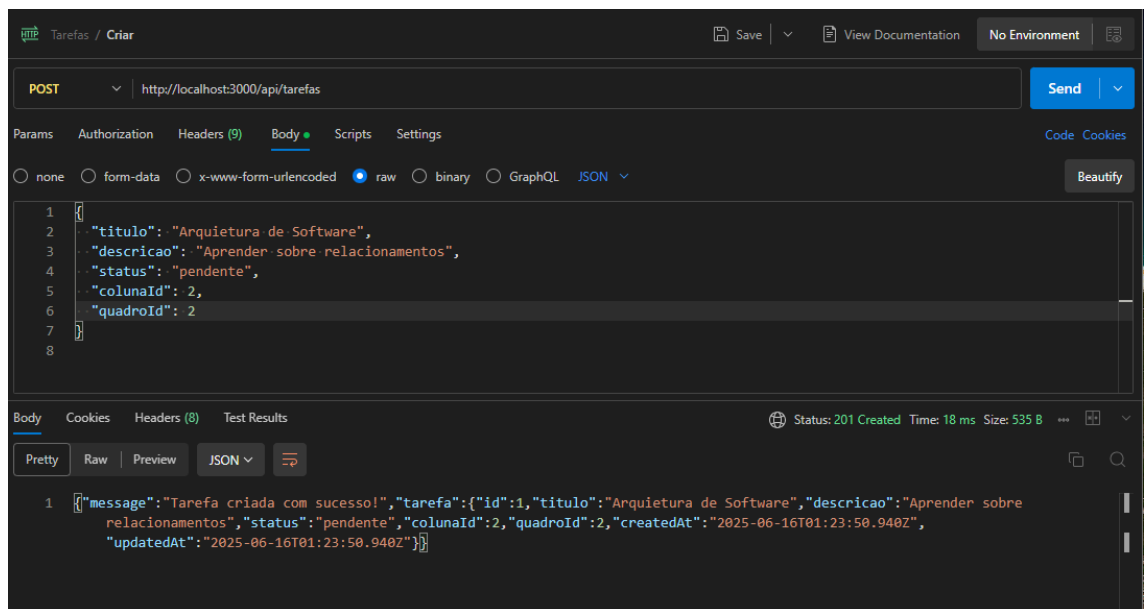
Permite filtrar colunas a partir de um termo no nome.



Tarefas

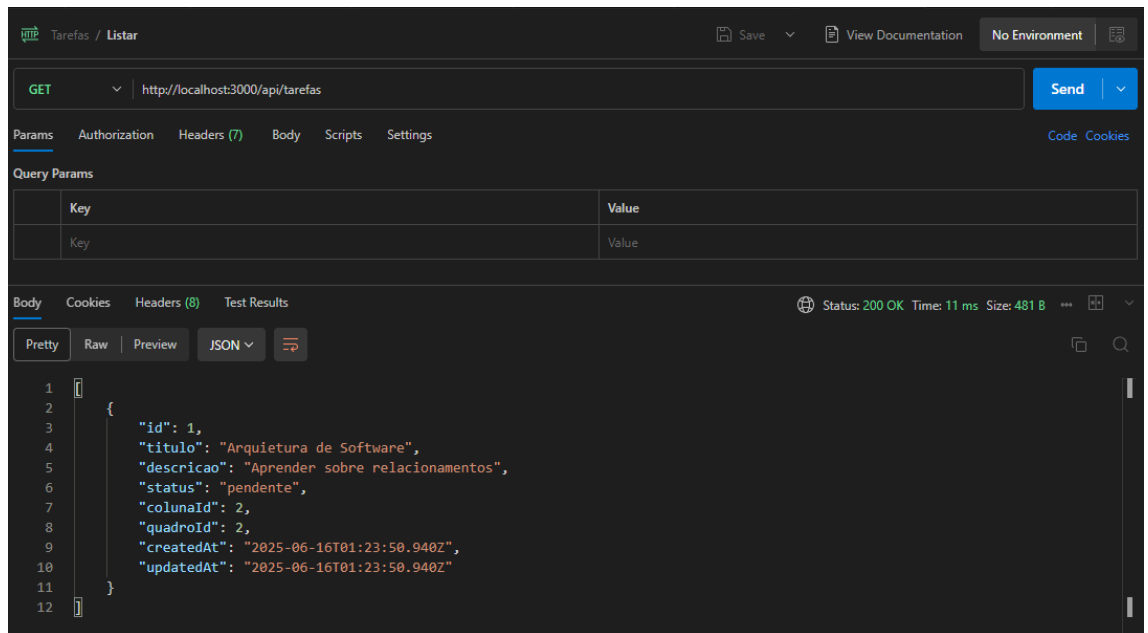
- **POST /tarefas – Criação de tarefas**

Permite que o usuário crie uma tarefa vinculada a uma coluna e quadro específicos. O id do quadro e da coluna precisam estar cadastrados.



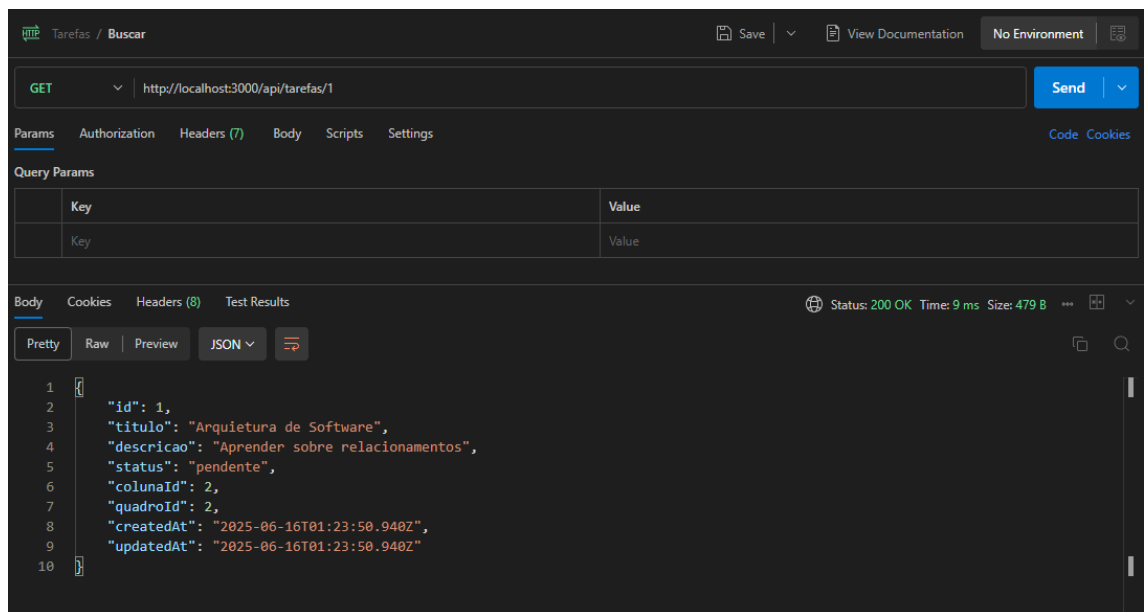
- **GET /tarefas – Listagem de todas as tarefas**

Retorna todas as tarefas existentes no sistema



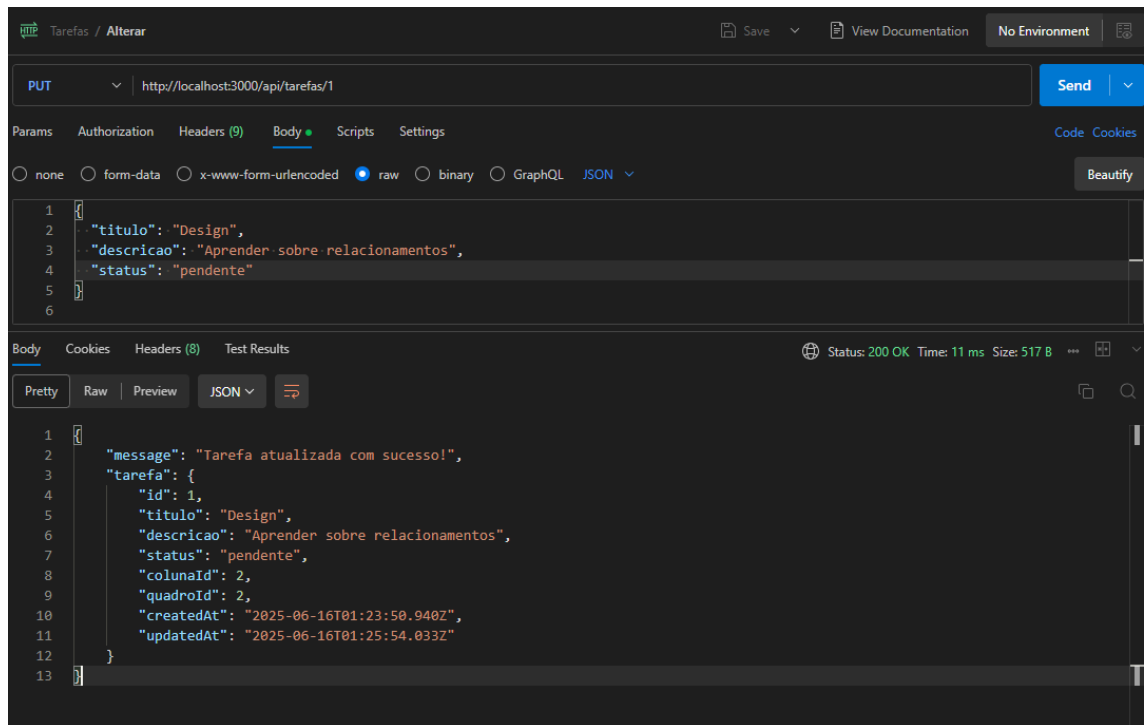
- **GET /tarefas/:id – Detalhes da tarefa**

Retorna as informações completas de uma tarefa específica.

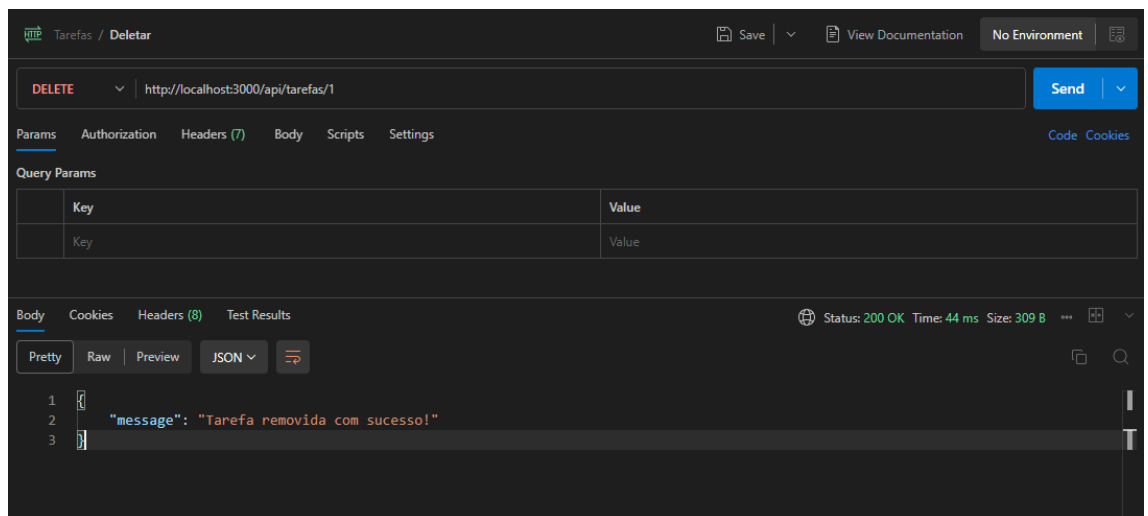


- **PUT /tarefas/:id – Edição de tarefas**

Permite alterar título, descrição e status de uma tarefa existente.



- **DELETE /tarefas/:id** – *Remoção de tarefa*
Exclui uma tarefa com base no ID informado.



- **GET /tarefas/search/:titulo** – *Busca de tarefas por título*
Permite buscar tarefas usando parte do texto do título como filtro.

Tarefas / Busca por nome

SaveView DocumentationNo Environment

GEThttp://localhost:3000/api/tarefas/search/nomeSend

ParamsAuthorizationHeaders (8)BodyScriptsSettingsCodeCookies

Query Params

	Key	Value
<input type="checkbox"/>	Key	Value
	Key	Value

Path Variables

	Key	Value
	nome	Arquitectura de Software

BodyCookiesHeaders (8)Test ResultsStatus: 200 OK Time: 7 ms Size: 481 B

PrettyRawPreviewJSON

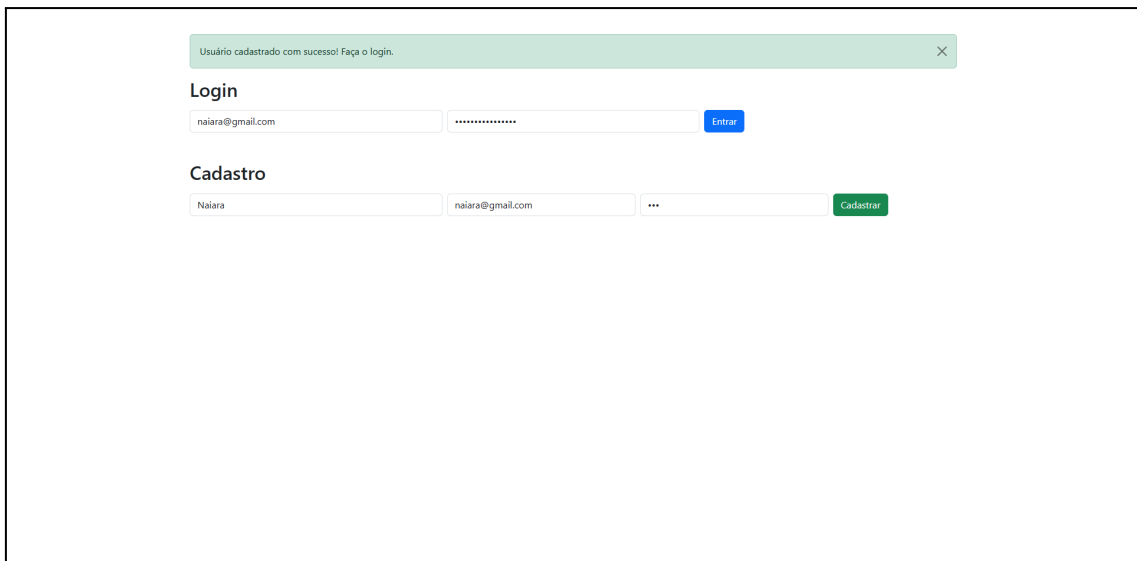
```
1  {
2
3    "id": 2,
4    "titulo": "Arquitectura de Software",
5    "descricao": "Aprender sobre relacionamentos",
6    "status": "pendente",
7    "colunaId": 2,
8    "quadroId": 2,
9    "createdAt": "2025-06-16T01:27:38.185Z",
10   "updatedAt": "2025-06-16T01:27:38.185Z"
11 }
12
```

6. Telas do UniKanban

Esta seção apresenta as principais telas da aplicação UniKanban, evidenciando o fluxo do usuário desde o login até a visualização e organização de tarefas nos quadros.

6.1 Login e Cadastro:

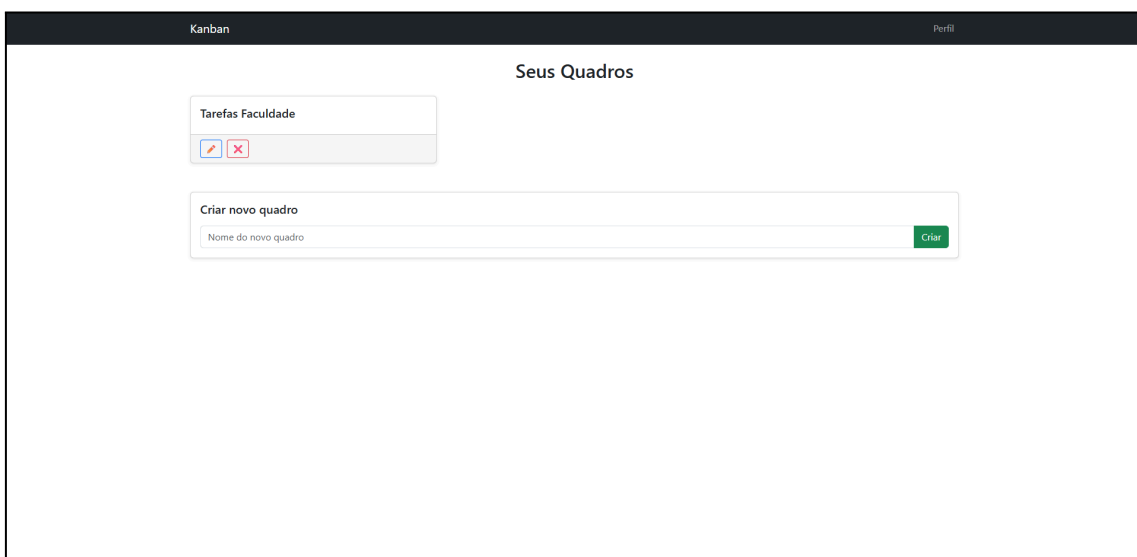
Tela inicial da aplicação. O usuário pode realizar login com e-mail e senha já cadastrados ou preencher os campos para criar uma nova conta. O formulário é simples, com campos validados e feedback visual em caso de sucesso ou erro.



A imagem mostra a interface de login e cadastro da aplicação UniKanban. No topo, há uma barra de notificação verde com o texto "Usuário cadastrado com sucesso! Faça o login." e um ícone de fechar. Abaixo, há duas seções: "Login" e "Cadastro". A seção "Login" possui dois campos de entrada: um para o e-mail (contendo "naiara@gmail.com") e outro para a senha (com pontos para ocultar), seguidos por um botão azul "Entrar". A seção "Cadastro" possui três campos de entrada: um para o nome (contendo "Naiara"), um para o e-mail (contendo "naiara@gmail.com") e um para a senha (com pontos para ocultar), seguidos por um botão verde "Cadastrar".

6.2 Quadros

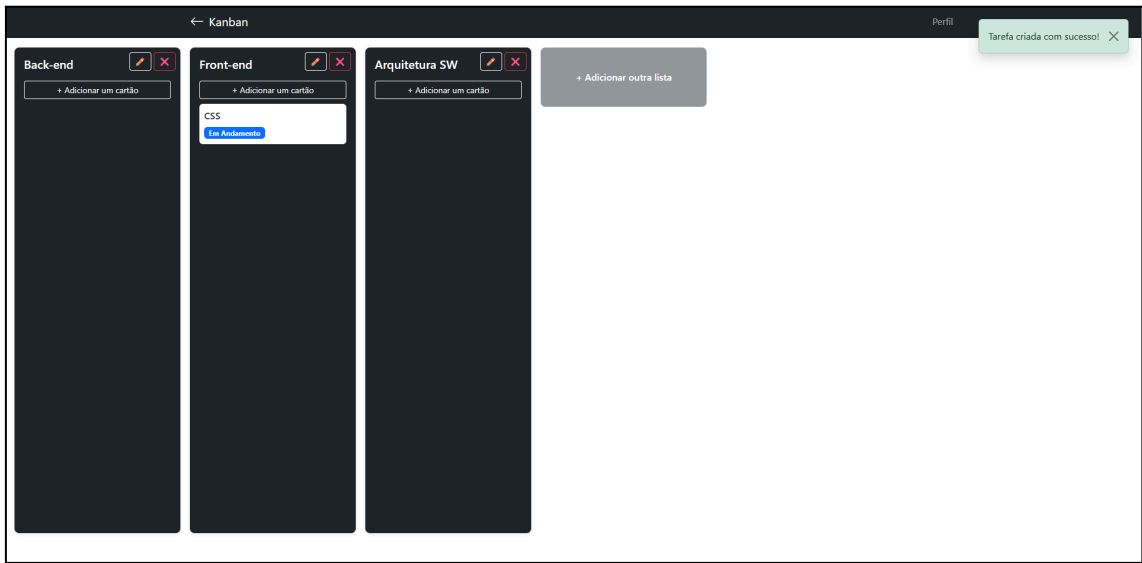
Após o login, o usuário é direcionado para a tela de seleção e criação de quadros. Aqui é possível visualizar os quadros existentes e adicionar novos quadros de tarefas, que serão utilizados na organização das colunas e tarefas.



A imagem mostra a interface de quadros da aplicação UniKanban. No topo, há uma barra de navegação com o texto "Kanban" e "Perfil". Abaixo, há uma seção intitulada "Seus Quadros". Nesta seção, há um card "Tarefas Faculdade" com dois ícones (um de adicionar e um de remover). Abaixo do card, há um formulário "Criar novo quadro" com um campo de entrada para o "Nome do novo quadro" e um botão verde "Criar".

6.3 Kanban:

Esta é a tela principal da aplicação, onde as tarefas são organizadas visualmente no estilo Kanban.



6.4 Usuário:

Tela de perfil do usuário, onde é possível visualizar as informações básicas da conta e editar seus dados.

