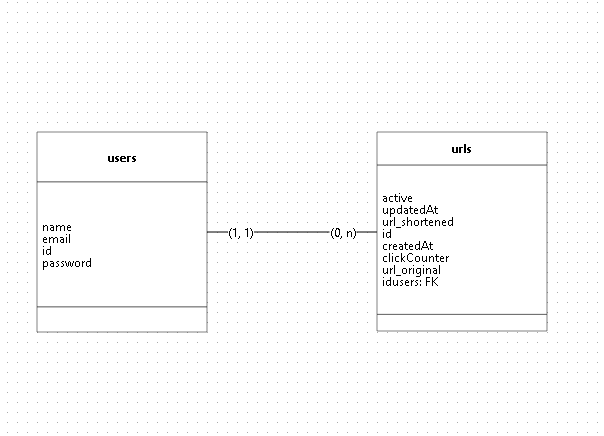
1. **Documentação Técnica:**
   1. URL Shortener API - API de encurtamento de URLs.
   2. O URL\_SHORTENER\_API é uma API REST desenvolvida com NestJS que permite o encurtamento e gerenciamento de URLs, com autenticação de usuários. A aplicação é containerizada com Docker e utiliza PostgreSQL como banco de dados. A persistência é gerenciada com Prisma ORM.
2. **Tecnologias utilizadas:**
   1. Principais tecnologias e frameworks:
      1. **Backend:** Node.Js, Nest.Js, TypeScript
      2. **Banco de dados:** PostgreSQL
      3. **ORM:** Prisma
      4. **Outras ferramentas:** Docker, Jest, Git, Swagger, Prometheus, Grafana, Br Modelo Web (modelagem de banco de dados).
3. **Arquitetura da aplicação:**
   1. Arquitetura Backend — NestJS com TypeScript (Modular por Domínio)
      1. A arquitetura adotada foi a modular com separação por domínios funcionais, utilizando NestJS com TypeScript. Essa abordagem foi escolhida por sua escalabilidade, organização clara e por promover um baixo acoplamento entre os módulos. A estrutura segue os princípios SOLID — principalmente o da responsabilidade única — e se beneficia fortemente do sistema de injeção de dependências do NestJS. Isso torna o código mais testável, reutilizável e de fácil manutenção, permitindo que novas funcionalidades sejam implementadas de forma coesa e sustentável à medida que a aplicação evolui.
4. **Instruções para rodar o projeto (setup):**
   1. **Passo a passo para rodar localmente:**
      1. Clone o repositório do GitHub:
         1. git clone <https://github.com/Carloshpjacinto/url_shortener_api.git>
      2. Acesse a pastas da aplicação e instale as dependências com o seguinte comando:
         1. npm install
      3. Rode os seguintes comandos no bash para remover os caracteres de quebra de linha do Windows do arquivo entrypoint.sh localmente:
         1. sed -i 's/\r$//' entrypoint.sh
         2. chmod +x entrypoint.sh
      4. Crie a variavel de ambiente JWT\_SECRET no arquivo .env. O arquivo .env.example mostra os nomes utilizados nas variáveis do projeto.
      5. Executar a aplicação em Docker::
         1. npm run dev
            1. As migrations são executadas automaticamente após o Docker terminar de subir o banco de dados.
            2. O Docker está executando o PostgreSQL, PgAdmin, Prometheus, Grafana e a Aplicação.
5. **Funcionalidades principais:**
   1. Abaixo estão listadas as principais funcionalidades implementadas na aplicação**:**
      1. **Cadastro de usuário:**  
          Permite o registro de novos usuários.
      2. **Login de usuário:**  
          Autenticação do usuário com verificação de credenciais.
      3. **Autenticação via JWT:**  
          Utilização de JSON Web Tokens para proteger rotas e garantir o acesso seguro às funcionalidades da aplicação.
      4. **Encurtamento de URLs:**  
          Permite que usuários, autenticados ou não, encurtem URLs.
      5. **Editar URLs encurtadas:**  
          Permite que o usuário atualize URLs encurtadas em sua conta.
      6. **Deletar URLs encurtadas:**  
          Funcionalidade que permite ao usuário deletar URLs criadas em sua conta.
6. **Principais Endpoints da API:**

|  |  |  |
| --- | --- | --- |
| **Método** | **Rota** | **Descrição** |
| POST | /auth/register | Registra os dados básicos de um novo usuário. |
| POST | /auth/login/ | Realiza a autenticação e o login do usuário. |
| GET | /auth/profile | Retorna os dados do usuário autenticado. |
| POST | /auth/url | Realiza o encurtamento de uma URL. |
| GET | /auth/redirect | Realiza a busca da URL original buscando pela URL encurtada. |
| DELETE | /auth/url | Deleta a URL encurtada na conta do usuário. |
| GET | /auth/url | Busca todas as URLs encurtadas na conta do usuário. |
| PATCH | /auth/url | Atualiza a URL encurtada presente na conta do usuário. |

Todos os endpoints (exceto register e login) requerem autenticação via token JWT no header da requisição.

1. **Estrutura do banco de dados:**
   1. **Diagrama do banco de dados:**



* 1. **Tabela user:**
     1. **Id →** Número de identificação única do registro do usuário (chave primária.
     2. **name** **→** nome do usuário, esse campo é do tipo string.
     3. **email** **→** e-mail do usuário para envio de confirmação de transferência, campo do tipo string.
     4. **password →** senha do usuário, esse campo é do tipo string.
  2. **Tabela urls:**
     1. **id** **→** Número de identificação única da URL (chave primária).
     2. **active →** Booleano de verificação da existência da URL.
     3. **updatedAt →** Data de atualização da URL.
     4. **url\_shortened** **→** valor da URL encurtada.
     5. **createdAt →** Data da criação da URL encurtada.
     6. **clickCounter →** Número de clicks dados nas URL depois de redirecionadas
     7. **url\_original** **→** Valor da orginal da URL.
     8. **userId →** ID do usuário autenticado que encurtou a URL.

1. **Como rodar teste**
   1. Os testes automatizados foram feitos com **Jest** por ser uma ferramenta rápida, simples de configurar e amplamente usada na comunidade JavaScript/TypeScript. Para executá-los com cobertura de código, utilize o comando: npm run test:cov
2. **Construção dos dashboards em Grafana**
   1. Configurar conexões no Grafana com os Data Sources (Aplicação e PostgreSQL):
      1. Acesse o menu Connections > Data sources.
         1. Aplicação (Nest.js com Prometheus):
            1. Clique em Add data source e selecione Prometheus.  
               No campo Prometheus server URL, insira: [http://host.docker.internal:9090](http://host.docker.internal:9090/)Clique em Save & Test para validar a conexão.
         2. Banco de Dados (PostgreSQL):
            1. Clique em Add data source e selecione PostgreSQL.  
               Preencha os seguintes campos:  
               Host: postgres  
               Database: url\_shortener  
               User: postgres  
               Password: docker  
               TLS/SSL Mode: disable  
               Clique em Save & Test para validar a conexão.
   2. O painel com os dashboards pode ser importado por meio do arquivo JSON presente no projeto: dashboard.grafana.json
   3. Configuração do Painel de Métricas da Aplicação:
      1. **Usuários Cadastrados:**

**Data source:** grafana-postgresql-datasource (PostgreSQL)

**Tabela:** users

**Operação:** COUNT

**Coluna:** id

Clique em **"Run query"**

* + 1. **URLs Ativas:**

**Data source:** grafana-postgresql-datasource (PostgreSQL)

**Tabela:** urls

**Operação:** COUNT

**Coluna:** active

Ative o **"Filter"**:

Clique em **"+"** e selecione: active == Yes

Clique em **"Run query"**

* + 1. **URLs Desativadas:**

**Data source:** grafana-postgresql-datasource (PostgreSQL)

**Tabela:** urls

**Operação:** COUNT

**Coluna:** active

Ative o **"Filter"**:

Clique em **"+"** e selecione: active == No

Clique em **"Run query"**

* + 1. **CPU (Aplicação e PostgreSQL):**

**Data source:** prometheus

**Métrica:** process\_cpu\_seconds\_total

Clique em **"Run query"**

* + 1. **Memória Física (Aplicação e PostgreSQL):**

**Data source:** prometheus

**Métrica:** process\_resident\_memory\_bytes

Clique em **"Run query"**

* + 1. **Memória Virtual (Aplicação e PostgreSQL):**

**Data source:** prometheus

**Métrica:** process\_virtual\_memory\_bytes

Clique em **"Run query"**

* + 1. **Handles Ativos (Aplicação):**

**Data source:** prometheus

**Métrica:** nodejs\_active\_handles\_total

Clique em **"Run query"**

* + 1. **Picos de Lentidão (Aplicação):**

**Data source:** prometheus

**Métrica:** nodejs\_eventloop\_lag\_p99\_seconds

Clique em **"Run query"**

1. **Possíveis melhorias**
   1. **Rate limit por IP ou por usuário**
   2. **Expiração automática das URLs**
   3. **Integração com Redis para cache de redirecionamentos**