DESAFIO TÉCNICO ALELO

DOCUMENTAÇÃO TÉCNICA

Autor: Silvio Ricardo Raposo Chagas

Versão: 1.00

ÍNDICE

1. Introdução 3
2. Considerações Iniciais 3
2.1 Objetivos do Projeto 3
3. Padrões Técnicos 3
3.1 Padrão de Comunicação 3
4. Web Service 3
4.1 Endpoints 4
5. Modelagem e dicionário de dados 5
5.1 Resumo 5
5.2 Modelagem 5
6. Configuração e Roteiro de Implantação 6
6.1 Requisitos de software 6
6.2 Procedimento de implantação 6
6.2.1 Publicação em servidor de aplicação 6
6.2.2 Publicação local e execução via JAR 7
6.2.3 Criação do schema e tabelas na base de dados 7

1. Introdução

Este documento tem por objetivo definir os critérios e especificações técnicas para utilização da API, conforme contexto apresentado no desafio, além da modelagem e dicionário de dados específico para a base de dados, entre outras informações necessárias para a implantação.

2. Considerações Iniciais

O projeto é uma ação proposta pela *Alelo*, a fim de avaliar os conhecimentos necessários na linguagem Java como parte do processo seletivo da empresa.

2.1 Objetivos do Projeto

Este projeto visa a disponibilização de *Web Service* com:

- Get/Post/Delete/Put;
- Testes unitários, usando banco em memória;
- Teste “mockando” o retorno de uma API de terceiros;
- Documentação via Swagger;

3. Padrões Técnicos

3.1 Padrão de Comunicação

O modelo de comunicação segue o padrão de Web Services definido pelo WS-I Basic Profile. A troca de mensagens entre o Web Service e o cliente segue o modelo REST e documento JSON.

4. Web Service

O mecanismo de utilização do Web Service segue as seguintes premissas:

a) Serão disponibilizados Web Services síncronos para os serviços;

b) O envio da solicitação e a obtenção do retorno serão realizados na mesma conexão através de um único método.

c) As URLs dos Web Services serão publicadas no Anexo I desta documentação.

d) c) O processo de utilização dos Web Services sempre é iniciado pelo interessado com o envio de uma mensagem nos padrões JSON.

e) A ocorrência de qualquer erro na validação dos dados recebidos interrompe o processo com o retorno HTTP de status 400 (Mal Formatado).

f) Padrão RESTF para as requisições e valores de retorno:

- 200 - Sucesso
- 201 - Registro criado, incluído com sucesso.
- 204 - Sem conteúdo para retornar
- 400 - Dados enviados em formato incorreto
- 404 - Registro não encontrado para a operação
- 409 - Conflito, já existe registro.

- 500 - Erro genérico no servidor, verificar aplicação

4.1 Endpoints

Usuário /usuarios [GET/POST/PUT/DELETE]

>[GET] Retorna lista de usuários

>[POST] Adiciona um usuário

```json
{
  "email": "email@email.com",
  "endereco": "ENDEREÇO DO CARINHA",
  "nome": "Carinha da Silva",
  "senha": "hahahhaha",
  "telefone": "9899999233"
}
```

>[PUT] /{id} Atualiza um usuário

```json
{
  "email": "email@email.com",
  "endereco": "Novo endereço",
  "senha": "novasenha"
}
```

>[DELETE] /{id} Apaga um usuário com base na id

Tarefas /tarefas [GET/POST/PUT/DELETE]

>[GET] Retorna lista de tarefas

>[POST] Adiciona uma tarefa

```json
{
  "descricao": "Tomar água",
  "usuario": {
    "id": 1
  }
}
```

>[PUT] /{id} Atualiza uma tarefa com base na id

```json
{
  "dataConcluida": "2021-01-20T20:15:45.093Z",
  "dataPrevista": "2021-01-20T20:16:45.093Z",
  "id": 1,
  "status": 1,
  "usuario": {
```

```
    "id": 1
  }
}
```

>[DELETE] /{id} Apaga uma tarefa com base na id

Listas /listas [GET/POST/PUT/DELETE]

>[GET] Retorna as listas

>[POST] Adiciona uma nova lista

```json
{
  "descricao": "Compras",
  "usuario": {
    "id": 1
  }
}
```

>[PUT] /{id} Atualiza uma lista com base na id

```json
{
  "descricao": "Sonhos",
  "usuario": {
    "id": 1
  }
}
```

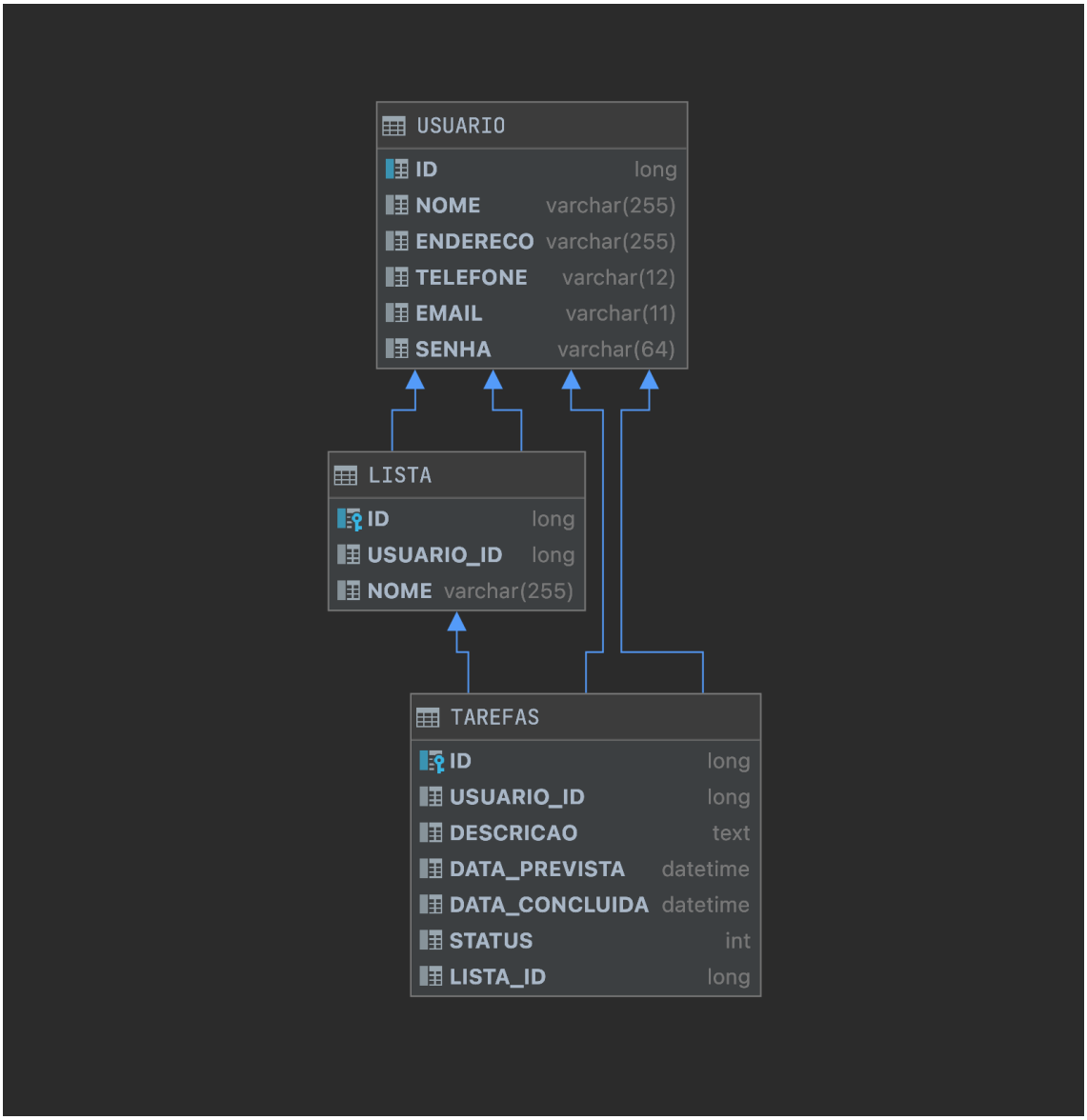
>[DELETE] /{id} Apaga uma tarefa com base na id

5. Modelagem e dicionário de dados

5.1 Resumo

A base de dados utilizada é H2 com persistência em arquivo local em ‘./local/Banco.db’.

5.2 Modelagem



6. Configuração e Roteiro de Implantação

6.1 Requisitos de software

- Java 8+

6.2 Procedimento de implantação

A aplicação pode ser implantada em servidor de aplicação, via pacote WAR ou executada diretamente via JAR. Todas as dependências são incluídas durante o processo automatizado de build.

6.2.1 Publicação em servidor de aplicação

6.2.1.1 Gerar o WAR do projeto

mvn compile war:war

6.2.2 Publicação local e execução via JAR

mvn package

6.2.3 Criação do schema e tabelas na base de dados

Schema.sql