

Projeto – Desenvolvedor Mobicare Core – Java Sênior - Daniel Ventura

Link da API em produção:

<http://ec2-18-231-160-253.sa-east-1.compute.amazonaws.com:8081/swagger-ui.html>

Repositório BitBucket:

<https://bitbucket.org/danielventura77>

O objetivo deste documento é descrever a arquitetura do projeto proposto, tendo em conta um cenário real de demanda de mercado, considerando requisitos de segurança, portabilidade, escalabilidade, qualidade de software e documentação. Além de demonstrar boas práticas de programação e evidenciar os cenários solicitados através de screenshots de tela.

1. Estrutura de pacotes da aplicação:

Package Explorer

- JavaSeniorDanielVentura [boot] [devtools]
 - src/main/java
 - br.com.mobicare
 - JavaSeniorDanielVenturaApplication.java **Classe de Start da Aplicação**
 - br.com.mobicare.adapter
 - WebConfigurerAdapter.java **Classe de configuração da qtd de registros por página numa consulta paginada no banco, quando esta não é informada na consulta.**
 - br.com.mobicare.docs
 - SwaggerConfig.java **Classe de configuração da interface web de documentação da API**
 - br.com.mobicare.endpoint
 - ColaboradorEndpoint.java
 - SetorEndpoint.java**Endpoints da API**
 - br.com.mobicare.error
 - ConstraintViolationDetails.java
 - ConstraintViolationException.java
 - ErrorDetails.java
 - MalformedJwtDetails.java
 - MalformedJwtException.java
 - ResourceNotFoundDetails.java
 - ResourceNotFoundException.java
 - ValidationErrorsDetails.java**Classes para manipulação de Exceptions com o objetivo de padronizar a saída de erro no Json de uma forma padronizada para o Frontend. Como por exemplo:**

```
1 {
2   "title": "Field Validation Error",
3   "status": 400,
4   "detail": "Field Validation Error",
5   "timestamp": 1528105160529,
6   "developerMessage": "org.springframework.web.bind.MethodArgumentNotValidException",
7   "field": "cpf,email",
8   "fieldMessage": "CPF inválido, Email inválido"
9 }
```
 - br.com.mobicare.handler
 - RestExceptionHandler.java **Classe central para handle de Exceptions do Spring Boot**
 - br.com.mobicare.model
 - AbstractEntity.java
 - ApplicationUser.java
 - Colaborador.java
 - Setor.java**Entity Beans da API**
 - br.com.mobicare.repository
 - ApplicationUserRepository.java
 - ColaboradorRepository.java
 - SetorRepository.java**Interfaces de acesso ao repositório (DAO)**
 - br.com.mobicare.security
 - JWTAuthenticationFilter.java
 - JWTAuthorizationFilter.java
 - SecurityConfig.java
 - SecurityConstants.java**Classes de configuração do Spring Security com token JWT (autenticação do tipo Bearer) - Filtros de Autenticação e Autorização com JWT**
 - br.com.mobicare.service
 - CustomUserDetailsService.java **Classe para manipulação de dados de autenticação do usuário no Spring Security**
 - br.com.mobicare.util
 - PasswordEncoder.java **Classe utilitária para encriptar a senha do usuário no banco de dados com o BCryptPasswordEncoder**
 - src/main/resources
 - scripts/MySQL
 - data.sql
 - schema.sql**Scripts DDL e DML para o banco de dados MySQL utilizado na API**
 - static
 - templates
 - application-dev.properties
 - application-hml.properties
 - application-prod.properties
 - application.properties
 - src/test/java
 - br.com.mobicare
 - ColaboradorEndpointTest.java
 - ColaboradorRepositoryTest.java
 - JavaSeniorDanielVenturaApplicationTests.java**- Classes de Testes Unitários com JUnit aplicadas as operações de CRUD do DAO. É utilizado o banco H2 em memória para o teste, garantindo a conservação de estado do banco após os testes.**
 - JRE System Library [JavaSE-1.8]
 - Maven Dependencies
 - src
 - target
 - mvnw
 - mvnw.cmd
 - pom.xml **Configuração de dependências do projeto, utilizando o Spring Boot 2.0.2 RELEASE**

Arquivos de configurações específicas para cada ambiente, como por exemplo: url de conexão com o banco, usuário e senha do banco.

A API contempla 3 perfis de build/deploy distintos:
- Desenvolvimento, Homologação e Produção

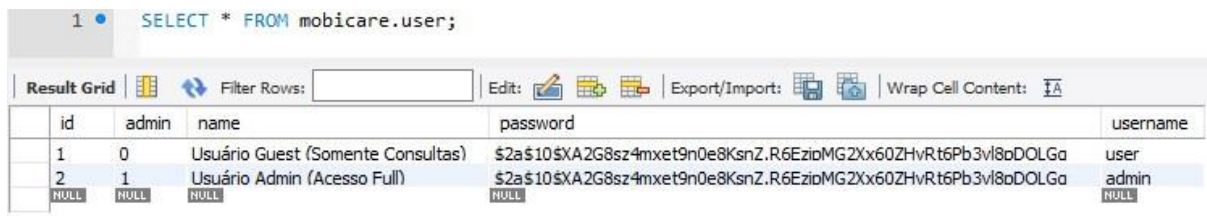
As configurações comuns a todos os ambientes se encontram no arquivo application.properties

- Classes de Testes de Integração sobre os endpoints, testando o status code de retorno. É utilizado o Mockito para mockar as operações DAO, garantindo a conservação do estado do banco após os testes.

2. Requisitos não funcionais

2.1. Segurança

Para atender a este requisito foi criada uma tabela **user** no banco de dados com os seguintes campos:



The screenshot shows a database query result for the 'user' table. The query is 'SELECT * FROM mobicare.user;'. The result grid displays two rows of data. The first row is for a 'user' with id 1, admin 0, name 'Usuário Guest (Somente Consultas)', and a long password. The second row is for an 'admin' with id 2, admin 1, name 'Usuário Admin (Acesso Full)', and the same long password. The 'password' field is highlighted in blue in both rows.

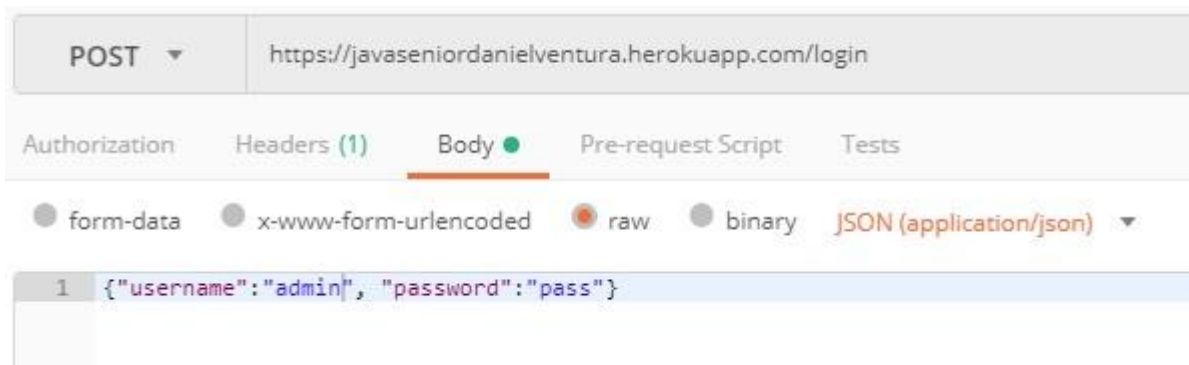
	id	admin	name	password	username
1	1	0	Usuário Guest (Somente Consultas)	\$2a\$10\$XA2G8sz4mxt9n0e8KsnZ.R6EzlpMG2Xx60ZHvRt6Pb3vI8oDOLGa	user
2	2	1	Usuário Admin (Acesso Full)	\$2a\$10\$XA2G8sz4mxt9n0e8KsnZ.R6EzlpMG2Xx60ZHvRt6Pb3vI8oDOLGa	admin

Não é possível executar os serviços nos endpoints da aplicação sem que seja feita a autenticação por um desses usuários. O usuário Guest, possui a **ROLE_USER** e desta forma está habilitado a utilizar os endpoints que possuam o padrão **"/**/protected/**"** no path. Dessa forma só estará habilitado a realizar consultas na API, mas não poderá alterar ou criar dados.

Já o usuário Admin, possui a **ROLE_USER** e a **ROLE_ADMIN** podendo acessar todos os endpoints da API, ou seja, com o padrão de nome **"/**/protected/**"** e **"/**/admin/**"**

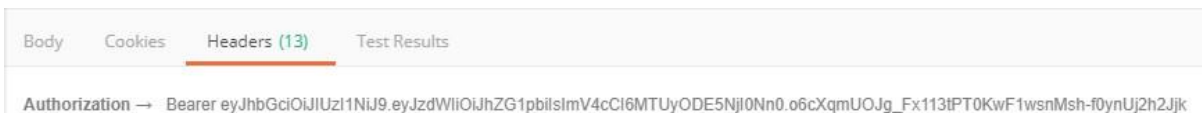
A autenticação é do tipo BEARER e é realizada com um **POST** no endpoint **"/login"** provido pelo próprio Spring Security. Recomendo utilizar o Postman para realizar o login.

No corpo da requisição deverá constar a seguinte informação:



OBS: A senha para os 2 usuários é: pass

Como resposta temos este Token JWT no header do response:



O mesmo tem data de validade de 1 dia. Dessa forma, durante este período, o mesmo poderá ser reutilizado sem a necessidade de ser regenerado. Uma vez expirado o acesso será negado, se fazendo necessário a geração de um novo token.

Caso o usuário utilize um token gerado com as credenciais **ROLE_USER** para acessar um path no padrão **"/**/admin/**"**, a API irá retornar status code 403- Forbidden.

2.2. Portabilidade

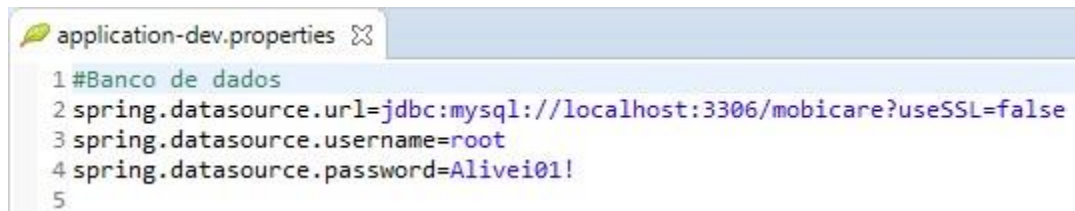
Com o objetivo de garantir a portabilidade da aplicação, foram criados 3 profiles para deploy. Os mesmos são designados por dev, hml e prod.

A aplicação é um jar executável com servidor de aplicação Tomcat embutido podendo ser iniciada através do comando:

```
java -Dspring.profiles.active=prod -jar JavaSeniorDanielVentura.jar
```

O argumento "prod" fornecido a JVM pela chave **spring.profiles.active** chaveia qual perfil será iniciado. Se o argumento não for fornecido na linha de comando, será utilizado o profile definido como default em **application.properties**.

O profile **dev** está configurado se conectar com uma instância do banco MySQL rodando local:

A screenshot of a code editor showing the 'application-dev.properties' file. The file contains five lines of configuration for a local MySQL database connection. Line 1 is a comment '#Banco de dados'. Line 2 sets 'spring.datasource.url=jdbc:mysql://localhost:3306/mobicare?useSSL=false'. Line 3 sets 'spring.datasource.username=root'. Line 4 sets 'spring.datasource.password=Alivei01!'. Line 5 is empty.

```
1 #Banco de dados
2 spring.datasource.url=jdbc:mysql://localhost:3306/mobicare?useSSL=false
3 spring.datasource.username=root
4 spring.datasource.password=Alivei01!
5
```

E o perfil **prod** está configurado para se conectar com uma instância do MySQL na Amazon RDS.

A screenshot of a code editor showing the 'application-prod.properties' file. The file contains five lines of configuration for an Amazon RDS MySQL instance. Line 1 is a comment '#Banco de dados'. Line 2 sets 'spring.datasource.url=jdbc:mysql://venturidb.clifppfb1rpw.sa-east-1.rds.amazonaws.com:3306/mobicare?useSSL=false'. Line 3 sets 'spring.datasource.username=venturi'. Line 4 sets 'spring.datasource.password=Alivei01!'. Line 5 is empty.

```
1 #Banco de dados
2 spring.datasource.url=jdbc:mysql://venturidb.clifppfb1rpw.sa-east-1.rds.amazonaws.com:3306/mobicare?useSSL=false
3 spring.datasource.username=venturi
4 spring.datasource.password=Alivei01!
5
```

2.3. Paginação no banco

Para garantir que a aplicação não tenha comprometimento de performance e principalmente não sobrecarregue os frontends, prejudicando a banda com a descarga excessiva de dados de consulta via JSON, foi adotada a seguinte alternativa:

Implementação de consulta paginada no banco de dados.

Dessa forma os clientes terão a possibilidade de informar na consulta de colaboradores os seguintes parâmetros:

- **size** que definirá a quantidade de registros de uma página.
- **page** que informará a API qual página da consulta será retornada.

Exemplo: /v1/protected/colaboradores/paginacao?page=0&size=10

Caso os parâmetros page e size não sejam informados na url, a aplicação retornará o size default igual a 5, configurado no application.properties dos seus respectivos profiles.

OBS: É possível utilizar também o parâmetro sort para ordenação em função de um dos fields da entidade retornada.

Por exemplo: sort=setor.nome,desc&sort=nome,asc

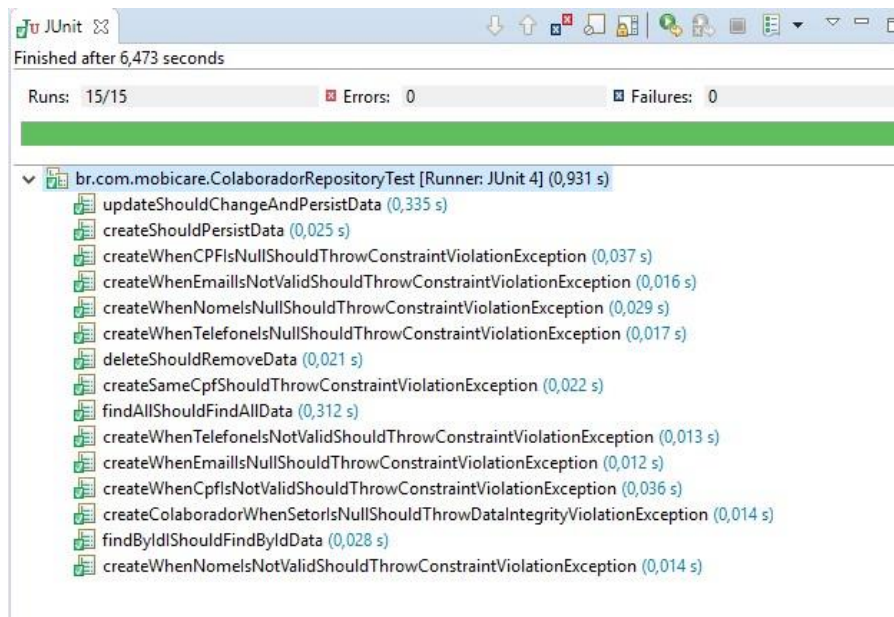
Neste caso ordena por nome do setor decrescente e por nome do colaborador ascendente.

2.4. Qualidade de Software

2.4.1. Realização de Testes Unitários com JUnit implementados na Classe ColaboradorRepositoryTest.

Aqui são testadas as operações de CRUD do DAO até o banco. O banco de dados “real” não participa do teste, pois é criada uma instância idêntica do banco em memória (Banco H2) e ao final do teste é feito o rollback da transação, fazendo com que os dados alterados em um teste, não interfira nos próximos testes do grupo.

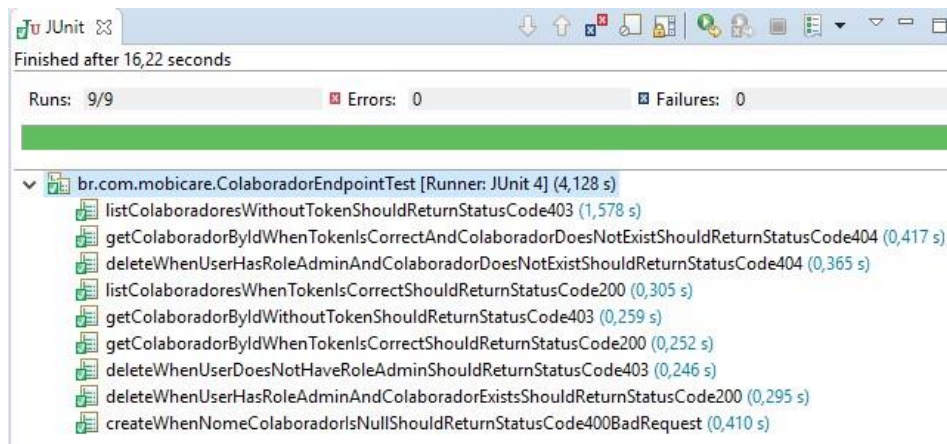
Segue abaixo a evidência de execução com sucesso dos testes unitários:



2.4.2. Realização de testes de integração com Junit e Mockito na Classe ColaboradorEndpointTest

Estes testes cobrem a implementação dentro dos métodos da classe Endpoint até as chamadas ao DAO que são mocadas pelo **Mockito**.

Segue abaixo a evidência de execução com sucesso dos testes de integração:



O objetivo é que em caso de falhas em algum destes testes, o ciclo de integração contínua seja quebrado até que a falha seja corrigida e o ciclo seja restabelecido liberando o build da API.

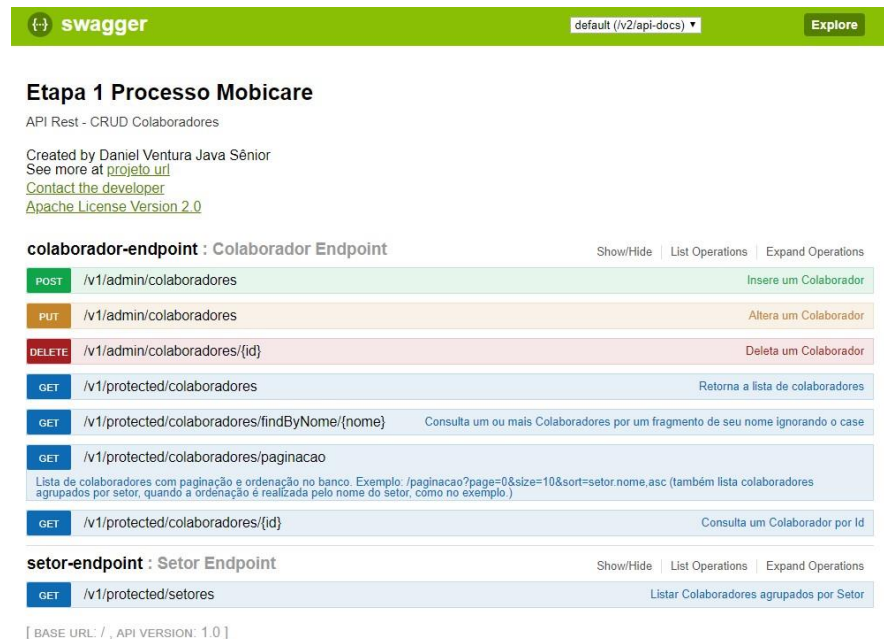
2.5. Documentação da API com Swagger-ui

Com o objetivo documentar e especificar para os clientes consumidores da API o detalhe da estrutura dos JSONs de entrada e saída, foi utilizado o Swagger-ui.

A vantagem dessa ferramenta está também em permitir a interação com a API de forma a consumir seus serviços, obtendo as respostas de forma prática e ágil, mesmo quando não se dispõe de um aplicativo cliente.

O único Endpoint que não pode ser acessado pelo Swagger-ui é o de login (/login) devido a este ser fornecido internamente pelo Spring Security.

Lista de Serviços:



Etapa 1 Processo Mobicare
API Rest - CRUD Colaboradores

Created by Daniel Ventura Java Sênior
See more at [projeto url](#)
[Contact the developer](#)
[Apache License Version 2.0](#)

colaborador-endpoint : Colaborador Endpoint

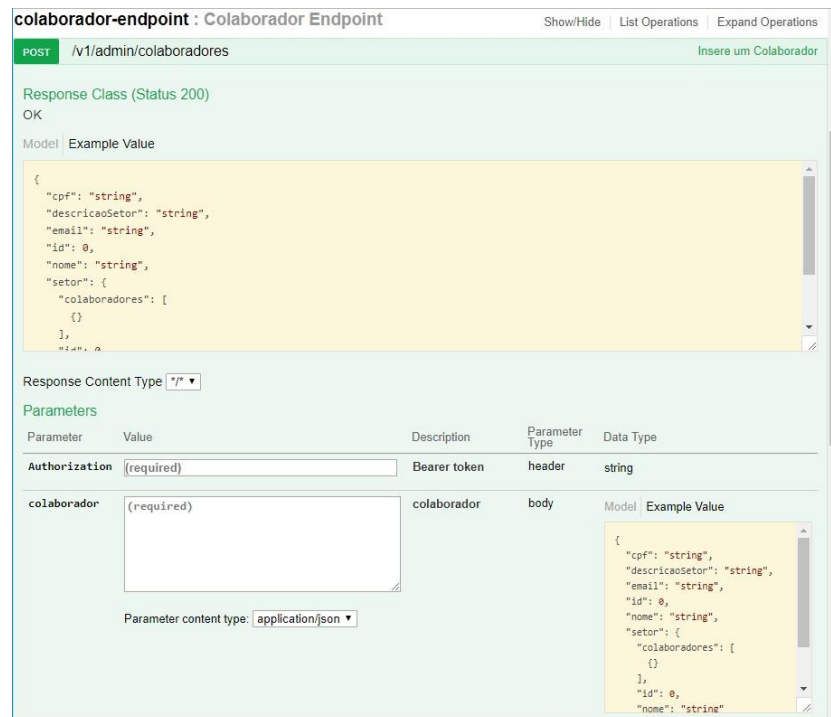
Method	Endpoint	Action
POST	/v1/admin/colaboradores	Inserir um Colaborador
PUT	/v1/admin/colaboradores	Alterar um Colaborador
DELETE	/v1/admin/colaboradores/{id}	Deletar um Colaborador
GET	/v1/protected/colaboradores	Retorna a lista de colaboradores
GET	/v1/protected/colaboradores/findByName/{nome}	Consulta um ou mais Colaboradores por um fragmento de seu nome ignorando o case
GET	/v1/protected/colaboradores/paginacao	Lista de colaboradores com paginação e ordenação no banco. Exemplo: /paginacao?page=0&size=10&sort=setor.nome.asc (também lista colaboradores agrupados por setor, quando a ordenação é realizada pelo nome do setor, como no exemplo.)
GET	/v1/protected/colaboradores/{id}	Consulta um Colaborador por id

setor-endpoint : Setor Endpoint

Method	Endpoint	Action
GET	/v1/protected/setores	Listar Colaboradores agrupados por Setor

[BASE URL: / , API VERSION: 1.0]

Entrada de dados para execução com exemplo de preenchimento informando os campos e tipos:



colaborador-endpoint : Colaborador Endpoint

POST /v1/admin/colaboradores Inserir um Colaborador

Response Class (Status 200)
OK

Model | Example Value

```
{
  "cpf": "string",
  "descricaoSetor": "string",
  "email": "string",
  "id": 0,
  "nome": "string",
  "setor": {
    "colaboradores": [
      {}
    ]
  },
  "id": 0,
  "nome": "string"
}
```

Response Content Type: */*

Parameters

Parameter	Value	Description	Parameter Type	Data Type
Authorization	(required)	Bearer token	header	string
colaborador	(required)	colaborador	body	Model Example Value

Parameter content type: application/json

Model | Example Value

```
{
  "cpf": "string",
  "descricaoSetor": "string",
  "email": "string",
  "id": 0,
  "nome": "string",
  "setor": {
    "colaboradores": [
      {}
    ]
  },
  "id": 0,
  "nome": "string"
}
```


3. Requisitos Obrigatórios

3.1. Os dados devem ser persistidos em qualquer fonte de dados. Ex.: banco de dados relacional, NoSQL, memória, arquivo de texto e etc.

Foi utilizado Banco de dados relacional MySQL na Amazon RDS.

3.2. Para atender uma solicitação de um setor da empresa será necessário desenvolver uma API REST para cadastro de colaboradores e essa API será consumida por diversos sistemas.

A Aplicação REST foi desenvolvida seguindo as melhores práticas e publicada para fins de demonstração e avaliação na Amazon AWS:

<http://ec2-18-231-160-253.sa-east-1.compute.amazonaws.com:8081/swagger-ui.html>

3.3. Essa API terá 4 funcionalidades:

3.3.1. Inserir colaborador

Evidência:

The screenshot displays a REST client interface with a POST request to `https://javaseniordanielventura.herokuapp.com/v1/admin/colaboradores`. The request body is a JSON object containing collaborator details. The response status is 201 Created, and the response body is a JSON object with an additional `descricaoSetor` field.

Request:

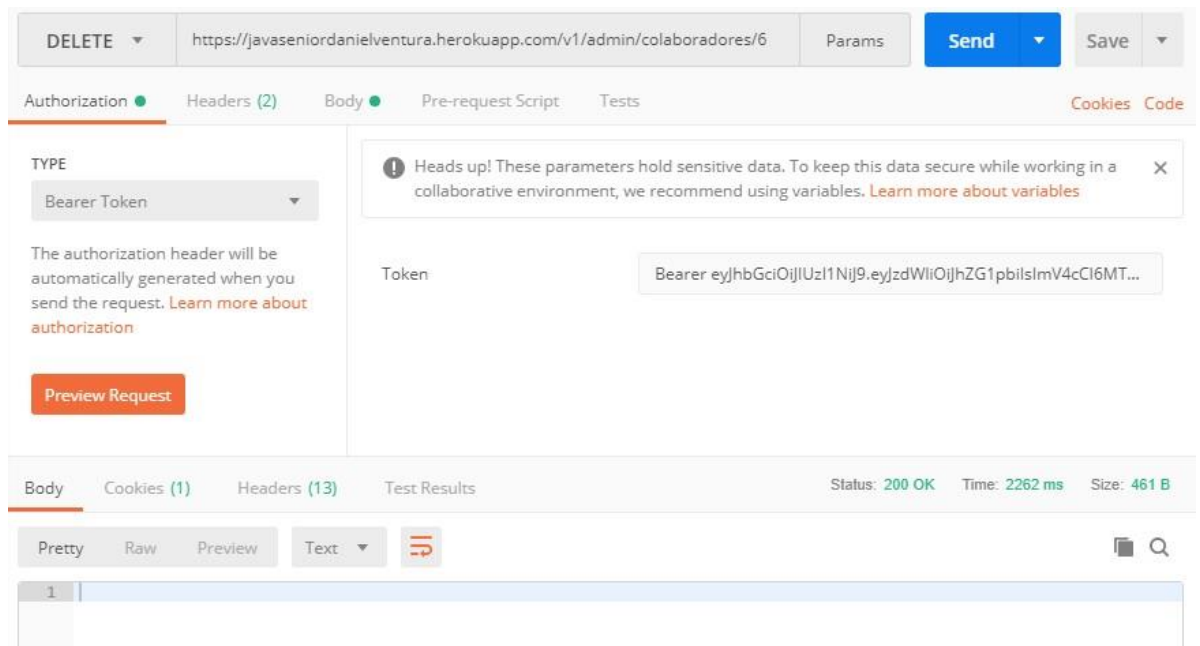
```
POST https://javaseniordanielventura.herokuapp.com/v1/admin/colaboradores
{
  "cpf": "27015442035",
  "nome": "Pedro Lira",
  "telefone": "(21) 97723-1123",
  "email": "pedro@gmail.com",
  "setor": {
    "id": 1
  }
}
```

Response:

```
201 Created
{
  "id": 7,
  "cpf": "27015442035",
  "nome": "Pedro Lira",
  "telefone": "(21) 97723-1123",
  "email": "pedro@gmail.com",
  "descricaoSetor": "1 - Setor A"
}
```

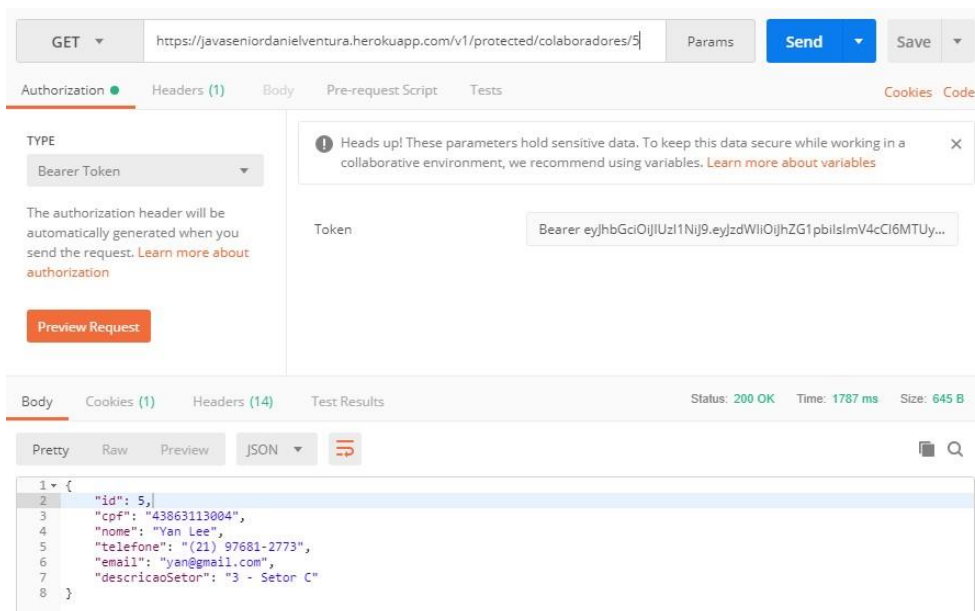
3.3.2. Remover colaborador

Evidência da remoção do colaborador de id 6:



3.3.3. Buscar um colaborador

Evidência de busca do colaborador de id 5:



3.3.4. Listar colaboradores agrupados por setor. Nesta última funcionalidade, deverão ser exibidos os campos nome e e-mail dos colaboradores para cada setor.

Segue a evidência:

GET https://javaseniordanielventura.herokuapp.com/v1/protected/setores Params Send

Body Cookies (1) Headers (14) Test Results Status: 200 OK Time: 2246 ms

Pretty Raw Preview JSON

```
1 [
2   {
3     "id": 1,
4     "nome": "Setor A",
5     "colaboradores": [
6       {
7         "nome": "Sérgio Novaes",
8         "email": "sergio@gmail.com",
9         "descricaoSetor": "1 - Setor A"
10      },
11      {
12        "nome": "Pedro Lira",
13        "email": "pedro@gmail.com",
14        "descricaoSetor": "1 - Setor A"
15      }
16    ]
17  },
18  {
19    "id": 2,
20    "nome": "Setor B",
21    "colaboradores": [
22      {
23        "nome": "Maria José",
24        "email": "mariajose@gmail.com",
25        "descricaoSetor": "2 - Setor B"
26      },
27      {
28        "nome": "Luiz Bertollo",
29        "email": "luiz@gmail.com",
30        "descricaoSetor": "2 - Setor B"
31      }
32    ]
33  },
34  {
35    "id": 3,
36    "nome": "Setor C",
37    "colaboradores": [
38      {
39        "nome": "Amanda Rineu",
40        "email": "amanda@gmail.com",
41        "descricaoSetor": "3 - Setor C"
42      },
43      {
44        "nome": "Yan Lee",
45        "email": "yan@gmail.com",
46        "descricaoSetor": "3 - Setor C"
47      }
48    ]
49  }
50 ]
```

Outra possibilidade para o agrupamento de colaboradores por setor é a busca paginada de colaboradores ordenada por nome do setor e por nome do colaborador, segue evidência:

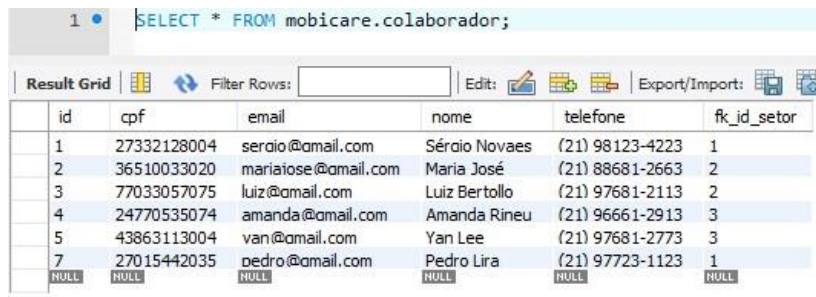
GET https://javaseniordanielventura.herokuapp.com/v1/protected/colaboradores/paginacao?page=0&size=10&sort=setor.nome,asc&sort=nome,asc

Pretty Raw Preview JSON

```
1 {
2   "content": [
3     {
4       "id": 7,
5       "cpf": "27015442035",
6       "nome": "Pedro Lira",
7       "telefone": "(21) 97723-1123",
8       "email": "pedro@gmail.com",
9       "descricaoSetor": "1 - Setor A"
10    },
11    {
12      "id": 1,
13      "cpf": "27332128004",
14      "nome": "Sérgio Novaes",
15      "telefone": "(21) 98123-4223",
16      "email": "sergio@gmail.com",
17      "descricaoSetor": "1 - Setor A"
18    },
19    {
20      "id": 3,
21      "cpf": "77033057075",
22      "nome": "Luiz Bertollo",
23      "telefone": "(21) 97681-2113",
24      "email": "luiz@gmail.com",
25      "descricaoSetor": "2 - Setor B"
26    },
27    {
28      "id": 2,
29      "cpf": "36510033020",
30      "nome": "Maria José",
31      "telefone": "(21) 88681-2663",
32      "email": "mariajose@gmail.com",
33      "descricaoSetor": "2 - Setor B"
34    },
35    {
36      "id": 4,
37      "cpf": "24770535074",
38      "nome": "Amanda Rineu",
39      "telefone": "(21) 96661-2913",
40      "email": "amanda@gmail.com",
41      "descricaoSetor": "3 - Setor C"
42    },
43    {
44      "id": 5,
45      "cpf": "43863113004",
46      "nome": "Yan Lee",
47      "telefone": "(21) 97681-2773",
48      "email": "yan@gmail.com",
49      "descricaoSetor": "3 - Setor C"
50    }
51  ],
52  "pageable": {
53    "sort": {
54      "sorted": true,
55      "unsorted": false
56    },
57    "pageSize": 10,
58    "pageNumber": 0,
59    "offset": 0,
60    "paged": true,
61    "unpaged": false
62  },
63  "last": true,
64  "totalElements": 6,
65  "totalPages": 1,
66  "first": true,
67  "sort": {
68    "sorted": true,
69    "unsorted": false
70  },
71  "numberOfElements": 6,
72  "size": 10,
73  "number": 0
74 }
```

3.3.5. Os atributos necessários para cadastrar um colaborador são: cpf, nome, telefone, e-mail.

Evidência:



The screenshot shows a database query interface with the SQL statement `SELECT * FROM mobicare.colaborador;`. Below the query, there is a 'Result Grid' showing the data for the 'colaborador' table. The table has columns: id, cpf, email, nome, telefone, and fk_id_setor. The data is as follows:

	id	cpf	email	nome	telefone	fk_id_setor
1	1	27332128004	sergio@gmail.com	Sérgio Novaes	(21) 98123-4223	1
2	2	36510033020	mariaiose@gmail.com	Maria José	(21) 88681-2663	2
3	3	77033057075	luiz@gmail.com	Luiz Bertollo	(21) 97681-2113	2
4	4	24770535074	amanda@gmail.com	Amanda Rineu	(21) 96661-2913	3
5	5	43863113004	van@gmail.com	Yan Lee	(21) 97681-2773	3
7	7	27015442035	pedro@gmail.com	Pedro Lira	(21) 97723-1123	1
		NULL	NULL	NULL	NULL	NULL

3.3.6. Um colaborador deve pertencer somente a um dos setores que já devem existir na tabela "setores", que possui um id e uma descrição.

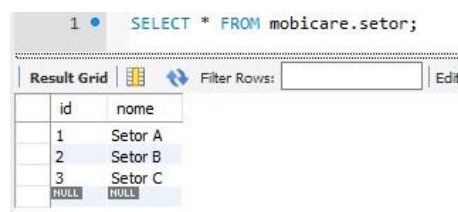
Mapeamento do Entity Setor no Entity Colaborador:

```
54
55 @ManyToOne
56 @JoinColumn(name = "fk_id_setor", nullable=false)
57 @JsonBackReference
58 private Setor setor;
```

Mapeamento do Entity Colaborador no Entity Setor:

```
24
25 @OneToMany(mappedBy = "setor", cascade = CascadeType.ALL)
26 @JsonManagedReference
27 private List<Colaborador> colaboradores;
```

Evidência da tabela Setor:



The screenshot shows a database query interface with the SQL statement `SELECT * FROM mobicare.setor;`. Below the query, there is a 'Result Grid' showing the data for the 'setor' table. The table has columns: id and nome. The data is as follows:

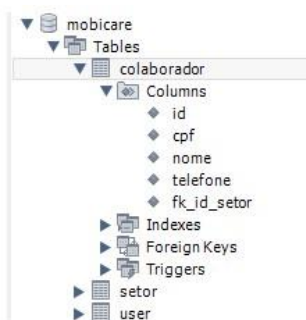
	id	nome
1	1	Setor A
2	2	Setor B
3	3	Setor C
	NULL	NULL

3.3.7. As tabelas devem ser criadas através de scripts, porém deve ser efetuada uma validação estrutural das tabelas existentes no banco no momento que a aplicação for iniciada.

Foi incluída a seguinte configuração ao JPA:

```
1 spring.profiles.active=dev
2
3 #Banco de dados
4 spring.datasource.tomcat.test-while-idle=true
5 spring.datasource.tomcat.validation-query=SELECT 1
6
7 spring.jpa.show-sql=true
8 spring.jpa.hibernate.ddl-auto=validate
9 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect
```

Para evidenciar o seu funcionamento, removi a coluna email da tabela de colaboradores:



Ao iniciar o servidor a seguinte exceção interrompeu o start do servidor, conforme esperado:

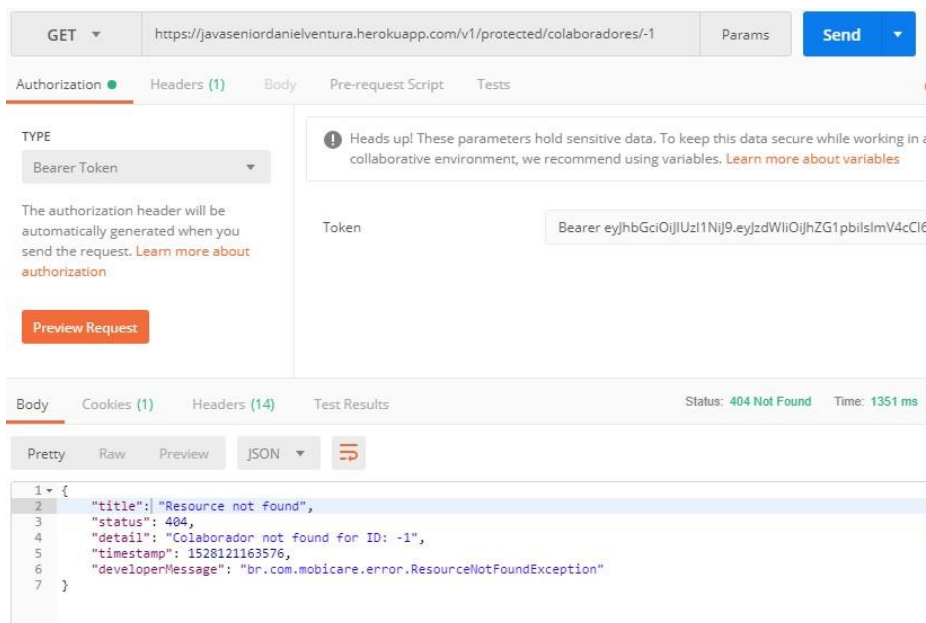
```
<terminated> JavaSeniorDanielVentura - JavaSeniorDanielVenturaApplication [Spring Boot App] C:\Program Files\Java\jdk1.8.0_172\bin\javaw.exe (4 de jun de 2018 02:57:18)
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.invokeInitMethods(AbstractAutowireCapableBeanFacto
    at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.initializeBean(AbstractAutowireCapableBeanFactory.
    21 common frames omitted
Caused by: org.hibernate.tool.schema.spi.SchemaManagementException: Schema-validation: missing column [email] in table [colaborador]
    at org.hibernate.tool.schema.internal.AbstractSchemaValidator.validateTable(AbstractSchemaValidator.java:136) ~[hibernate-core-5.2
```

Ou seja, evidenciamos que a referida configuração, realiza a validação estrutural das tabelas no Banco, impedindo que a aplicação seja levantada em produção com potenciais erros na DDL do schema.

3.3.8. A lista de códigos de Status HTTP deve seguir a seguinte regra:

- a. 201 - Recurso criado (**vide evidências acima**)
- b. 200 - Sucesso (**vide evidências acima**)
- c. 400 - Requisição inválida
- d. 404 - Recurso não encontrado

Segue a evidência da consulta de um colaborador com id = -1, para evidenciar o status code 404:



E abaixo uma evidência de requisição inválida, 400 bad request:

