03

Obtendo coordenadas do aluno

Transcrição

Obter as coordenadas de latitude e longitude de um aluno baseado em seu endereço não é uma operação simples para o banco de dados, e sim para o consumo de um serviço, e por isso criaremos um novo pacote com a classe `GeolocalizacaoService`:

```
package br.com.alura.escolalura.service;

@Service
public class GeolocalizacaoService {
  public List<Double> obterLatELongPor(Contato contato){
    return null;
  }
}
```

Já criamos o método `obterLatELongPor`, que tem como parâmetro um objeto `contato`, o qual deve retornar uma lista. Perceba também que a classe está anotada como `@Service` para deixar claro ao Spring do que se trata a classe.

O próximo passo é criar um objeto de contexto que configure a chave de acesso à API do Google Maps utilizando a classe `GeioApiContext`. Com o contexto, poderemos criar um objeto de *request* pela classe `GeocodingApi`, que nos permite indicar um endereço pelo qual queremos fazer uma pesquisa.

```
GeoApiContext context = new GeoApiContext().setApiKey("AIzaSyA03zKo_Ey2LXS6y9SS12t9Fq4ZIA29lOk")
GeocodingApiRequest request = GeocodingApi.newRequest(context).address(contato.getEndereco());
```

Estamos pesquisando o endereço do aluno por meio do Google Maps, porém essa operação pode não ser instantânea, portanto utilizaremos o método `await` do objeto `request` para aguardar pela resposta. Com os possíveis resultados, selecionaremos o primeiro.

```
GeocodingResult[] results = request.await();
GeocodingResult resultado = results[0];
```

Cada resultado individual desta pesquisa por endereço possui um atributo chamado `geometry`, que por sua vez possui outro, `location`, a partir do qual poderemos saber a longitude e latitude do aluno.

```
Geometry geometry = resultado.geometry;
LatLng location = geometry.location;
```

A latitude e longitude presentes no objeto `location` podem ser utilizadas na criação da lista de coordenadas, retornando esta como resultado da execução do método `obterLatELongPor`. O código completo deste método se encontra abaixo:

```
@Service
public class GeolocalizacaoService {

  public List<Double> obterLatELongPor(Contato contato) throws ApiException, InterruptedExceptio
    GeoApiContext context = new GeoApiContext().setApiKey("AIzaSyA03zKo_Ey2LXS6y9SS12t9Fq4ZIA29
    GeocodingApiRequest request = GeocodingApi.newRequest(context).address(contato.getEndereco(

    GeocodingResult[] results = request.await();
    GeocodingResult resultado = results[0];

    Geometry geometry = resultado.geometry;
    LatLng location = geometry.location;

    return Arrays.asList(location.lat, location.lng);
  }

}
```

Concluindo esta classe, poderemos utilizá-la no *controller* de alunos para que, ao salvarmos um deles, possamos salvar também suas coordenadas. Primeiro injetamos o objeto `geolocalizacaoService` :

```
@Controller
public class AlunoController {
  // código omitido

  @Autowired
  private GeolocalizacaoService GeolocalizacaoService;
}
```

E utilizamos o método `obterLatELongPor` no método `salvar` da seguinte forma:

```
@PostMapping("/aluno/salvar")
public String salvar(@ModelAttribute Aluno aluno){
  try {
    List<Double> latELong = geolocalizacaoService.obterLatELongPor(aluno.getContato());
    aluno.getContato().setCoordinates(latELong);
    repository.salvar(aluno);
  } catch (Exception e) {
    System.out.println("Endereco nao localizado");
    e.printStackTrace();
  }

  System.out.println(aluno);
  return "redirect:/";
}
```

O que falta para podermos testar tudo que fizemos até aqui é atualizar nosso *codec* para tratar das questões de contato do aluno. Já fizemos isto antes com outros atributos, e para sermos mais práticos, já temos o código pronto. No método `encode` da classe `AlunoCodec` , adicionaremos:

```
public void encode(BsonWriter writer, Aluno aluno, EncoderContext encoder) {
  //código omitido
  Contato contato = aluno.getContato();

  List<Double> coordinates = new ArrayList<Double>();
  for(Double location : contato.getCoordinates()){
    coordinates.add(location);
  }

  document.put("contato", new Document()
    .append("endereco" , contato.getEndereco())
    .append("coordinates", coordinates)
    .append("type", contato.getType()));

  //código omitido
}
```

No método `decode` teremos a adição:

```
public Aluno decode(BsonReader reader, DecoderContext decoder) {

  //código omitido

  Document contato = (Document) document.get("contato");
  if (contato != null) {
    String endereco = contato.getString("contato");
    List<Double> coordinates = (List<Double>) contato.get("coordinates");
    aluno.setContato(new Contato(endereco, coordinates));
  }

  //código omitido
}
```

Com o *codec* devidamente atualizado para tratar dos campos de contato do aluno, poderemos testar nossa aplicação. Cadastraremos um novo aluno chamado José, com data de nascimento em 22 de Maio, cursando Administração, e que tem como endereço a Rua Vergueiro, 3000.

Verificando se os dados do José foram salvos com sucesso no banco de dados, teremos:

```
{
  "_id" : ObjectId("59a4f87ba8a81321a4b23c7c"),
  "nome" : "Jose",
  "data_nascimento" : ISODate("2017-05-22T03:00:00Z"),
  "curso" : {
    "nome" : "Administração"
  },
  "habilidades" : [ ],
  "notas" : [ ],
  "contato" : {
    "endereco" : "Rua Vergueiro, 3000",
    "coordinates" : [
      -23.5872307,
      -46.6339501
```

```
      ],
      "type" : "Point"
    }
  }
```

As coordenadas foram buscadas e salvas em nosso documento de aluno. Para testar se elas realmente funcionam, poderemos usar o site latlong.net (http://latlong.net). No campo `Place Name` poderemos colar as coordenadas de latitude e longitude separadas por vírgula. Teste!

> Fizemos muitas consultas ao banco de dados utilizando o terminal, porém existem ferramentas visuais para verificar o banco de forma mais interessante, como o Robomongo (https://robomongo.org/download), conhecida por Robo 3T.

Um último detalhe que podemos adicionar ao nosso cadastro de alunos é o *autocomplete* de endereços. Com esta funcionalidade, iniciando-se a digitação do endereço, o próprio Maps pode sugerir o endereço completo.

Para fazermos isto realizaremos uma nova chamada para a API fornecendo nossa chave de autenticação e informando a função JavaScript a ser executada, para indicar o *autocomplete*. Primeiramente a chamada para a API, que irá no formulário de cadastro de aluno, o `cadastrar.html` .

```
<script src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&amp;libraries=places&amp;
```

É preciso indicar a chave da API, e no final da chamada teremos um parâmetro de *callback* indicando que a função a ser executada é a `autoComplete` . Criaremos essa função no arquivo `googlemaps.js` , na pasta `resources/static/js/` . O código desta função se encontra abaixo:

```
function autoComplete(){
  var input = document.getElementById('endereco');
  autocomplete = new google.maps.places.Autocomplete(input);
}
```

Como último passo, precisaremos adicionar o carregamento deste *script* na página de cadastro de aluno:

```
<script type="text/javascript" src="../js/googlemaps.js"></script>
```

O código completo da página de cadastro após estas adições fica da seguinte forma:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" >
<head>
  <meta charset="UTF-8" />
  <link type="text/css" rel="stylesheet" href="../materialize/css/materialize.min.css" media="s
  <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet" />
  <title>EscolAlura</title>
  <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.1.min.js"></script>
  <script type="text/javascript" src="../materialize/js/inicializar.js"></script>
  <script type="text/javascript" src="../js/googlemaps.js"></script>
</head>
```

```
<body class="grey lighten-3">
  <div id="formularioEdicao" class="container">
    <h3 class="main-title center">Cadastrar Aluno</h3>
    <div class="row">
      <form class="col s12" action="#" th:action="@{/aluno/salvar}" th:object="${aluno}" method
        <div class="section">
          <h5>Dados Básicos</h5>
          <div class="row">
            <div class="input-field col s12">
              <input id="nome" type="text" th:field="*{nome}" />
              <label for="nome">Nome</label>
            </div>
          </div>
          <div class="row">
            <div class="input-field col s12">
              <input id="dataNascimento" type="date" class="datepicker" th:field="*{dataNascime
              <label for="dataNascimento">Dt. Nascimento</label>
            </div>
          </div>
          <div class="row">
            <div class="input-field col s12">
              <input id="curso" type="text" class="validate" th:field="*{curso.nome}"  />
              <label for="curso">Curso</label>
            </div>
          </div>
          <div class="row">
            <div class="input-field col s12">
              <input id="endereco" type="text" class="validate" th:field="*{contato.endereco}"
              <label for="endereco">Endereço</label>
            </div>
          </div>


        </div> <!-- Fim SECTION Dados Basicos -->

        <div class="row">
          <div class="input-field col s12 center">
            <button class="btn waves-effect waves-light" type="submit" name="action">Salvar Alu
          </div>
        </div>
      </form>
    </div>

  </div> <!-- Fim do formulario de edicao -->
  <script src="https://maps.googleapis.com/maps/api/js?key=AIzaSyA03zKo_Ey2LXS6y9SS12t9Fq4ZIA29
  <script type="text/javascript" src="../materialize/js/materialize.min.js"></script>
</body>
</html>
```

Vamos testar o cadastro de alunos novamente, nos atentando ao fato de que o endereço é sugerido pela própria API, facilitando o aluno ou o responsável por ele. A imagem abaixo ilustra o funcionamento do *autocomplete*: