



Residência em Software

Residência em Tecnologia da Informação e Comunicação

SpringBoot: Dados em Requisições API

Professor:

Alvaro Degas Coelho



INSTITUIÇÃO EXECUTORA



COORDENADORA



APOIO



Para Começar

- Anotações que usamos até agora
 - `@Controller`: indica para o SpringBoot que uma classe implementa um Controller
 - Ela será associada a endpoints
 - `@RequestMapping`: indica para o SpringBoot que um método está sendo mapeado para um tipo específico de requisição
 - Normalmente o endpoint mapeado é indicado em seguida
 - `@ResponseBody`: indica que a saída do método já é a saída a ser devolvida no endpoint solicitado

Sobre a Arquitetura

- A classe RedeSocialApplication
 - Inicia a aplicação
 - TomCat, programas de Backend, etc.
- As classes dentro da package controller
 - São os Controllers. Serão associados aos Endpoints (entre outras coisas)
 - anotação @Controller
- Endpoint
 - Posição válida dentro do site (no caso, localhost:8080)
 - Exemplo: Localhost:8080/listausuarios

Umas coisas a melhorar

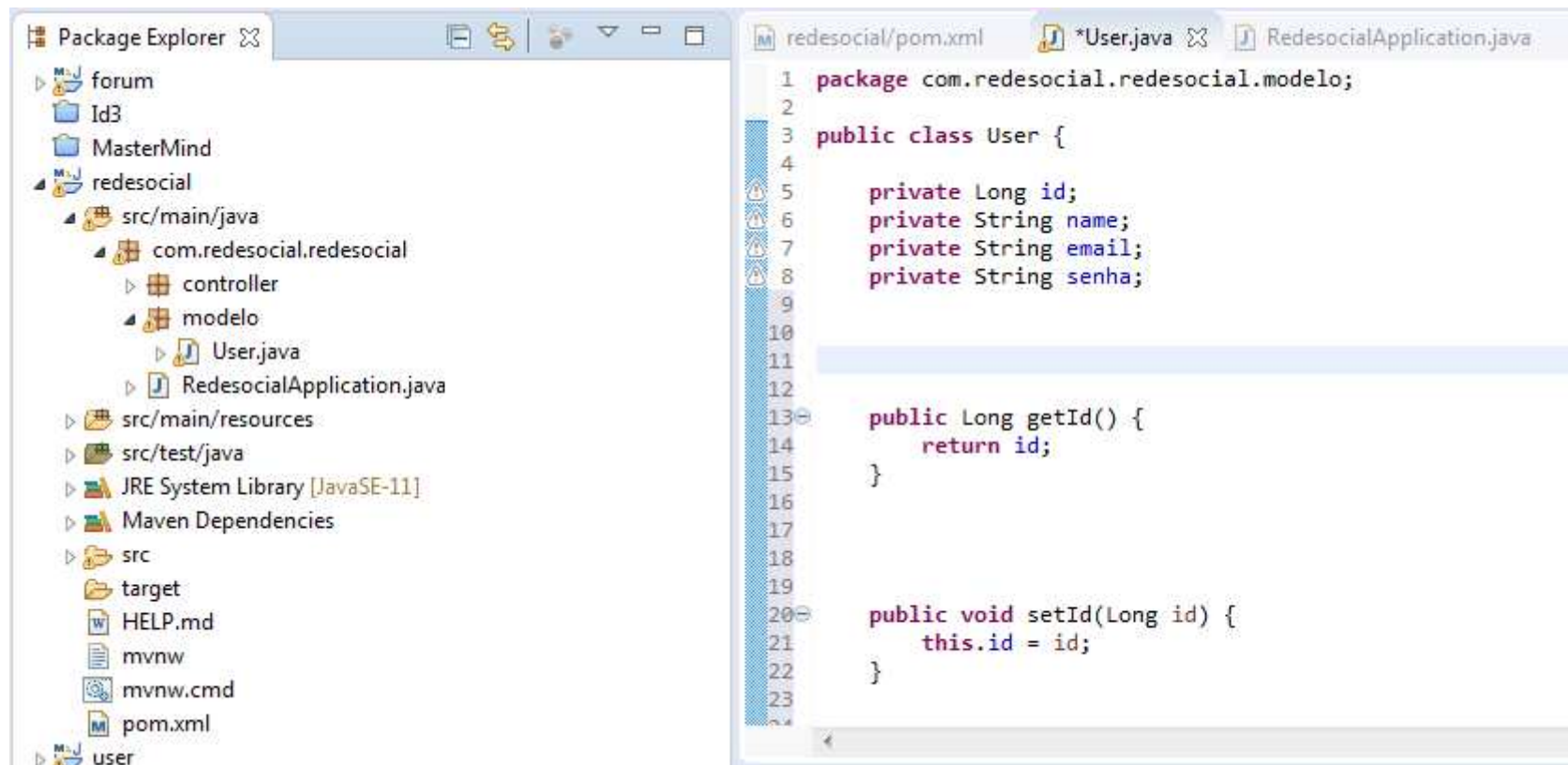
- Nosso Controller alô mundo tem limitações
 - Obviamente
- Em casos gerais um controller não apresenta uma página
 - Devolve dados e/ou condições finais (confirmar, negar, exceções, etc.)
 - Rest
- Uma classe pode ser definida como um Controller Rest
 - Todos os métodos serão rest

O projeto

- Vamos começar a rede social
 - Backend
- Saupor que temos um FrontEnd “pronto”
 - CRUD do Usuário
- Vamos tratar o BackEnd
 - Iniciando com o Usuário

Criar uma classe Usuario

- Dentro de uma package modelo
 - “irmã” da package controller
- Id (Long), Nome, email e senha
- Construtor, Getters & Setters



The screenshot shows an IDE with the Package Explorer on the left and the Source Editor on the right. The Package Explorer displays the project structure, including the 'redesocial' package and its sub-packages 'controller' and 'modelo'. The 'User.java' file is highlighted in the 'modelo' package. The Source Editor shows the following code:

```
1 package com.redesocial.redesocial.modelo;
2
3 public class User {
4
5     private Long id;
6     private String name;
7     private String email;
8     private String senha;
9
10
11
12
13     public Long getId() {
14         return id;
15     }
16
17
18
19
20     public void setId(Long id) {
21         this.id = id;
22     }
23
24 }
```

User não é Entity

- Ainda
- Estamos criando a classe para ilustrar a geração de um dado a partir de um EndPoint
- “Fazer de conta” que há uma consulta
 - Retornar uma Lista

Um Endpoint

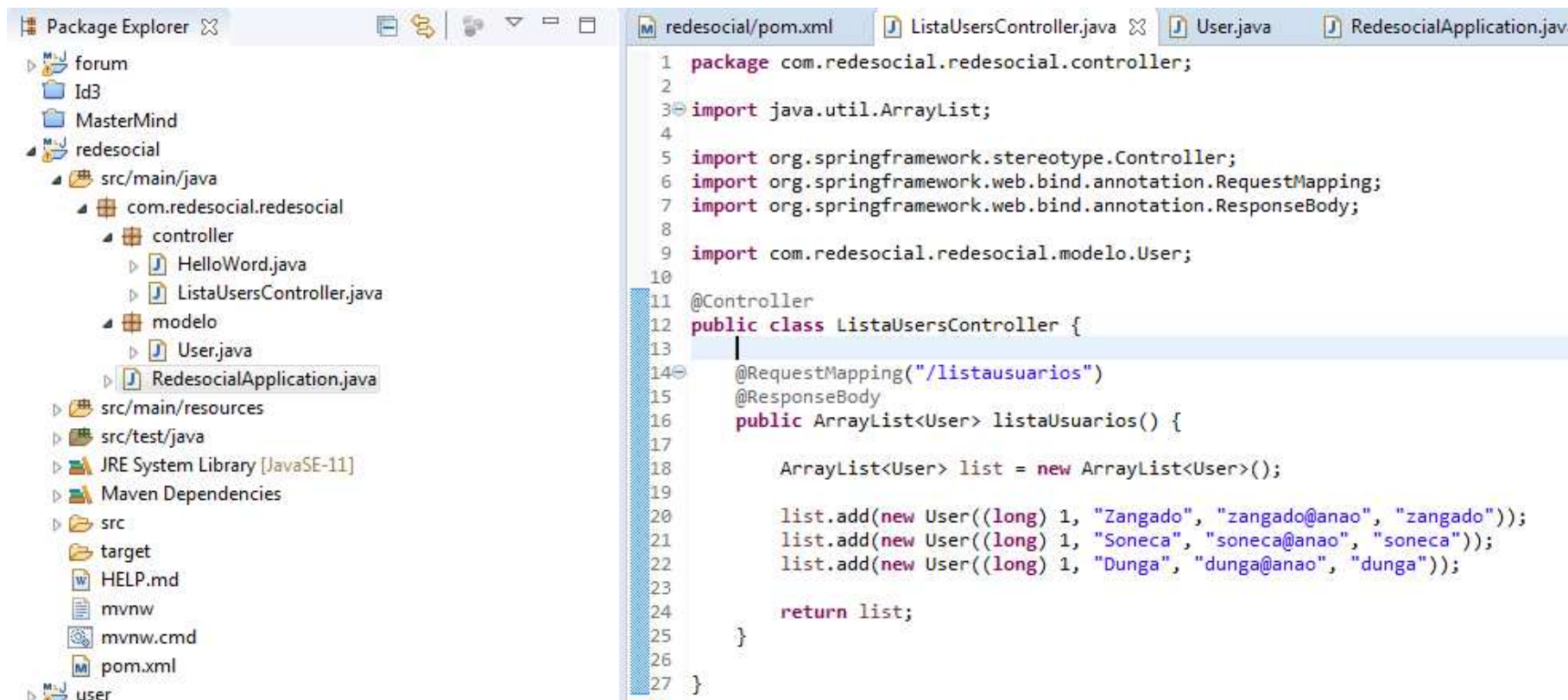
- Para retornar uma lista de usuários
 - Responderá em localhost:8080/listausuarios
 - Um novo controller!
 - Olhe pro Controller Alo Mundo!!!!
 - @Controller
- Um método dentro do Controller
 - Retorna um ArrayList de Usuários!
 - Crie o ArrayList (use sua criatividade)
 - @RequestMapping("/listausuarios")
 - @ResponseBody



Residência
em Software

Façam seus Controllers

Endpoint para /listausuarios



The screenshot displays an IDE with the Package Explorer on the left and the code editor on the right. The Package Explorer shows the project structure for 'redesocial', including packages like 'controller' and 'modelo'. The code editor shows the implementation of the 'ListaUsersController' class, which is annotated with '@Controller' and '@RequestMapping("/listausuarios")'. The 'listaUsuarios()' method returns an 'ArrayList<User>' containing three sample users: 'Zangado', 'Soneca', and 'Dunga'.

```
1 package com.redesocial.redesocial.controller;
2
3 import java.util.ArrayList;
4
5 import org.springframework.stereotype.Controller;
6 import org.springframework.web.bind.annotation.RequestMapping;
7 import org.springframework.web.bind.annotation.ResponseBody;
8
9 import com.redesocial.redesocial.modelo.User;
10
11 @Controller
12 public class ListaUsersController {
13
14     @RequestMapping("/listausuarios")
15     @ResponseBody
16     public ArrayList<User> listaUsuarios() {
17
18         ArrayList<User> list = new ArrayList<User>();
19
20         list.add(new User((long) 1, "Zangado", "zangado@anao", "zangado"));
21         list.add(new User((long) 1, "Soneca", "soneca@anao", "soneca"));
22         list.add(new User((long) 1, "Dunga", "dunga@anao", "dunga"));
23
24         return list;
25     }
26 }
27 }
```

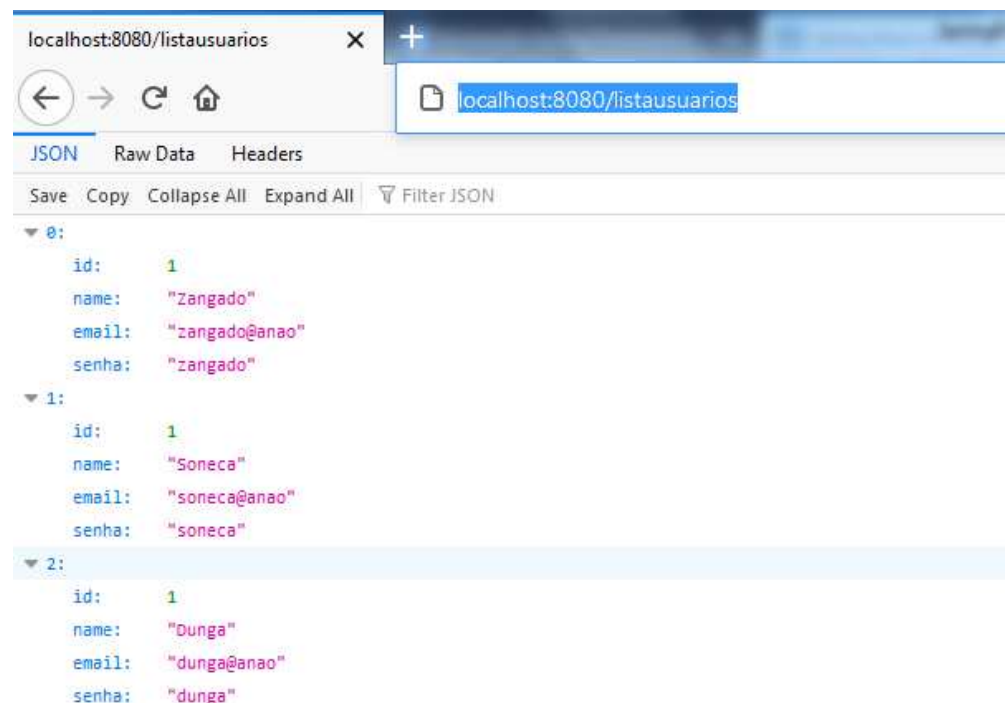


Residência
em Software

Execute seu projeto
Verifique localhost:8080/listausuarios

Endpoint para /listausuarios

- Formato json (Springboot usa a biblioteca Jackson)



```
localhost:8080/listausuarios
JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON
0:
  id: 1
  name: "Zangado"
  email: "zangado@anao"
  senha: "zangado"
1:
  id: 1
  name: "Soneca"
  email: "soneca@anao"
  senha: "soneca"
2:
  id: 1
  name: "Dunga"
  email: "dunga@anao"
  senha: "dunga"
```

Sobre a Arquitetura

- A classe RedeSocialApplication
 - Inicia a aplicação
 - TomCat, programas de Backend, etc.
- As classes dentro da package controller
 - São os Controllers. Serão associados aos Endpoints (entre outras coisas)
 - anotação @Controller
- Endpoint
 - Posição válida dentro do site (no caso, localhost:8080)
 - Localhost:8080/listausuarios

Umas coisas a melhorar

- **@ResponseBody**
 - Devolver o dado ao invés de navegar para uma página (Api Rest)
- Sempre será assim (no nosso caso)
- **@RestController**
 - Controller do tipo Rest (não devolve página)
 - Tudo que devolve é ResponseBody

```
1 package com.redesocial.redesocial.controller;  
2  
3 import java.util.ArrayList;  
4  
5 import org.springframework.web.bind.annotation.RequestMapping;  
6 import org.springframework.web.bind.annotation.RestController;  
7  
8 import com.redesocial.redesocial.modelo.User;  
9  
10 @RestController  
11 public class ListaUsersController {  
12  
13     @RequestMapping("/listausuarios")  
14     public ArrayList<User> listaUsuarios() {  
15  
16         ArrayList<User> list = new ArrayList<User>();  
17  
18         list.add(new User((long) 1, "Zangado", "zangado@anao", "zangado"));  
19         list.add(new User((long) 1, "Soneca", "soneca@anao", "soneca"));  
20         list.add(new User((long) 1, "Dunga", "dunga@anao", "dunga"));  
21  
22         return list;  
23     }  
24 }  
25 }
```


@RestController

- Usualmente um Controller vai ser do tipo REST
 - Associado a endpoints
 - Devolvendo dados
- Por padrão, o SpringBoot formata os dados utilizando JSON
 - Embute as classes necessárias para isso

Um incômodo

- É necessário parar a execução e reiniciar todas as vezes que houver alterações
- Muitas vezes esquecemos
 - Isto atrapalha muito o processo de desenvolvimento
- Uma ideia: a cada arquivo salvo no projeto a aplicação automaticamente reiniciar


DevTools



dependência devtools

Imagens Vídeos Shopping Notícias Livros Maps Voos Finanças

Aproximadamente 31.200 resultados (0,23 segundos)

 Spring
<https://spring.io> › 2015/06/17 › d... - Traduzir esta página

DevTools in Spring Boot 1.3

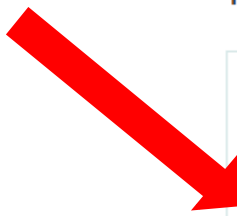
17 de jun. de 2015 — **DevTools** in Spring Boot 1.3 · Property Defaults · Automatic Restart · LiveReload · Remote Debug Tunneling · Remote Update and Restart · Video Preview.

DevTools in Spring Boot 1.3

ENGINEERING | PHIL WEBB | JUNE 17, 2015 | 39 COMMENTS

Spring Boot 1.3 will ship with a brand new module called `spring-boot-devtools`. The development-time experience when working on Spring Boot applications.

To use the module you simply need to add it as a dependency in your Maven POM:

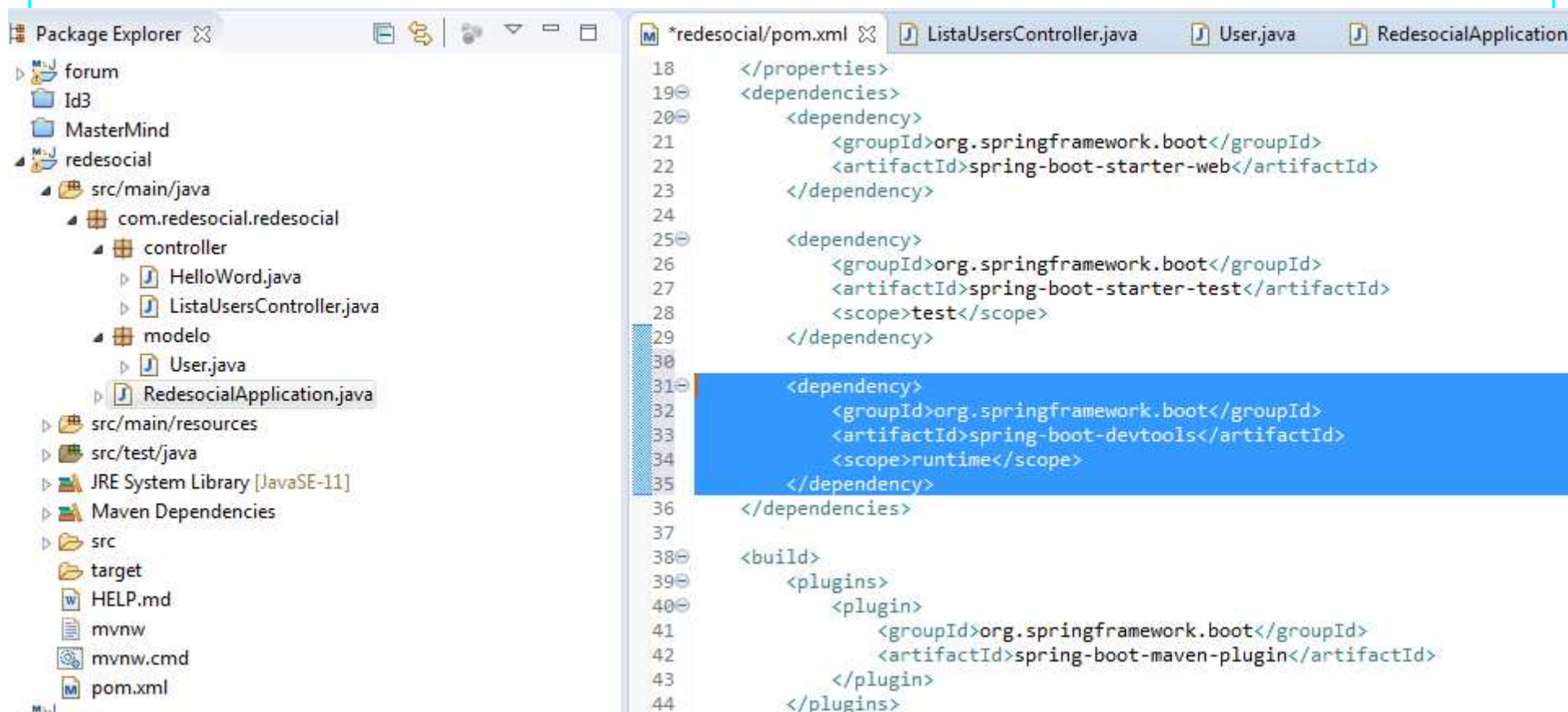


```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
  </dependency>
</dependencies>
```

**Ou... Baixe o arquivo com
configurações SpringBoot de nosso
drive**

Devtools

- Um módulo que reinicia “automaticamente” a aplicação
 - Quando se salva algum arquivo no projeto
- Nova dependência no arquivo pom.xml



The screenshot shows an IDE with the Package Explorer on the left and the Maven POM file editor on the right. The Package Explorer shows the project structure for 'redesocial', including the 'src/main/java' directory with the 'com.redesocial.redesocial' package. The POM file editor shows the 'dependencies' section with three dependencies: 'spring-boot-starter-web', 'spring-boot-starter-test', and 'spring-boot-devtools'. The 'spring-boot-devtools' dependency is highlighted in blue.

```
18     </properties>
19     <dependencies>
20         <dependency>
21             <groupId>org.springframework.boot</groupId>
22             <artifactId>spring-boot-starter-web</artifactId>
23         </dependency>
24
25         <dependency>
26             <groupId>org.springframework.boot</groupId>
27             <artifactId>spring-boot-starter-test</artifactId>
28             <scope>test</scope>
29         </dependency>
30
31         <dependency>
32             <groupId>org.springframework.boot</groupId>
33             <artifactId>spring-boot-devtools</artifactId>
34             <scope>runtime</scope>
35         </dependency>
36     </dependencies>
37
38     <build>
39         <plugins>
40             <plugin>
41                 <groupId>org.springframework.boot</groupId>
42                 <artifactId>spring-boot-maven-plugin</artifactId>
43             </plugin>
44         </plugins>
```

DTO

- Já falamos disso
 - JPQL
- Data Transfer Object
- Não é conveniente transportar um objeto do model para a view
 - Dados desnecessários, dados privados, etc.
- DTO: um objeto definido para a transporte dos dados pertinentes

UserDTO

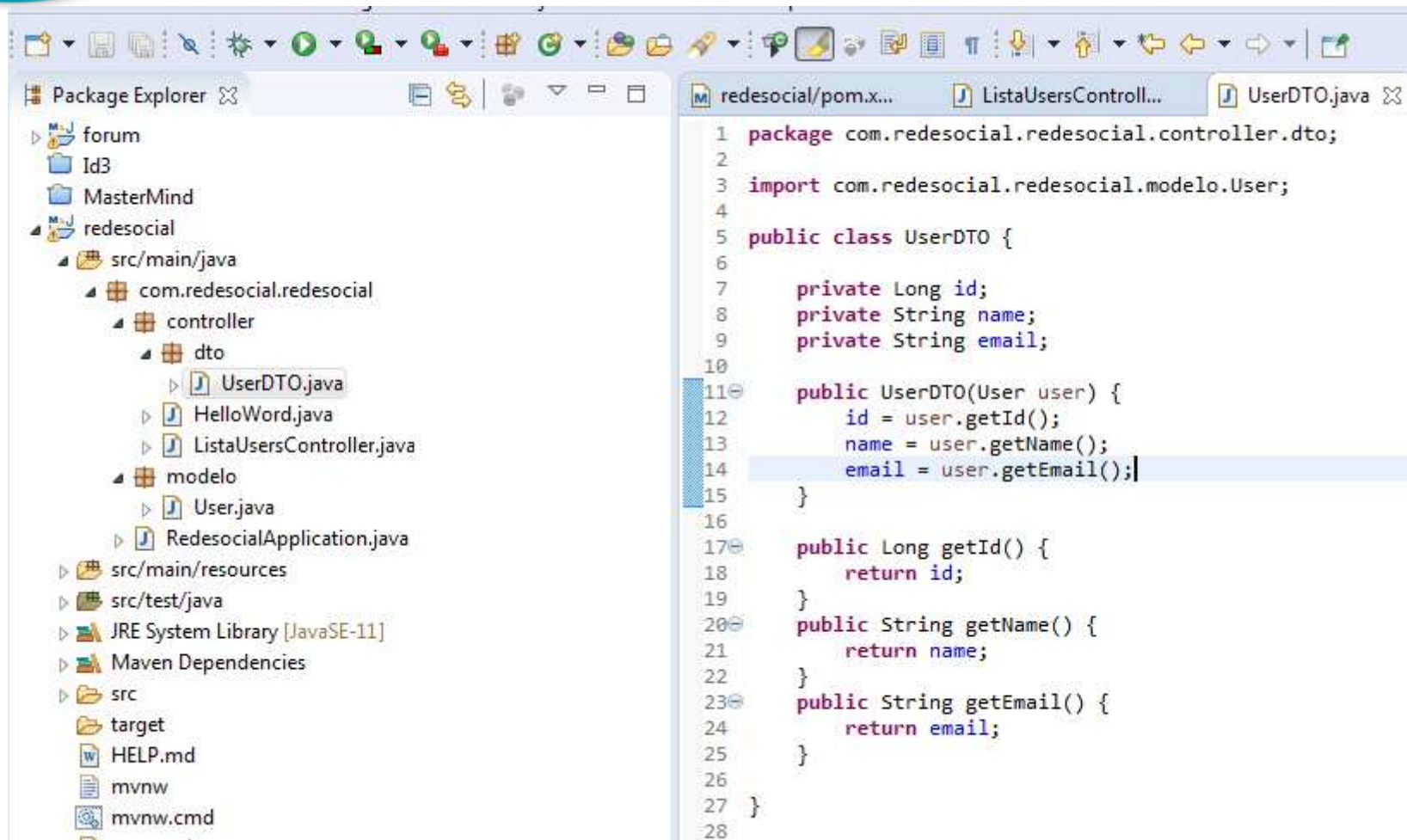
- Package DTO
 - Dentro da package controller (DTO só tem sentido neste contexto)
- Apenas atributos pertinentes
 - Id, Nome, email
- Getters (porque não precisa dos setters?)
- Construtor
 - Pode receber um User como parâmetro
 - Encapsulamento e desacoplamento
 - Cria um UserDTO com os atributos do User
- Controller ListaUsers
 - Precisa ser modificado
 - Lista de UserDTO ao invés de lista de User



Residência
em Software

Façam seu DTO Adequem seus Controllers

UserDTO



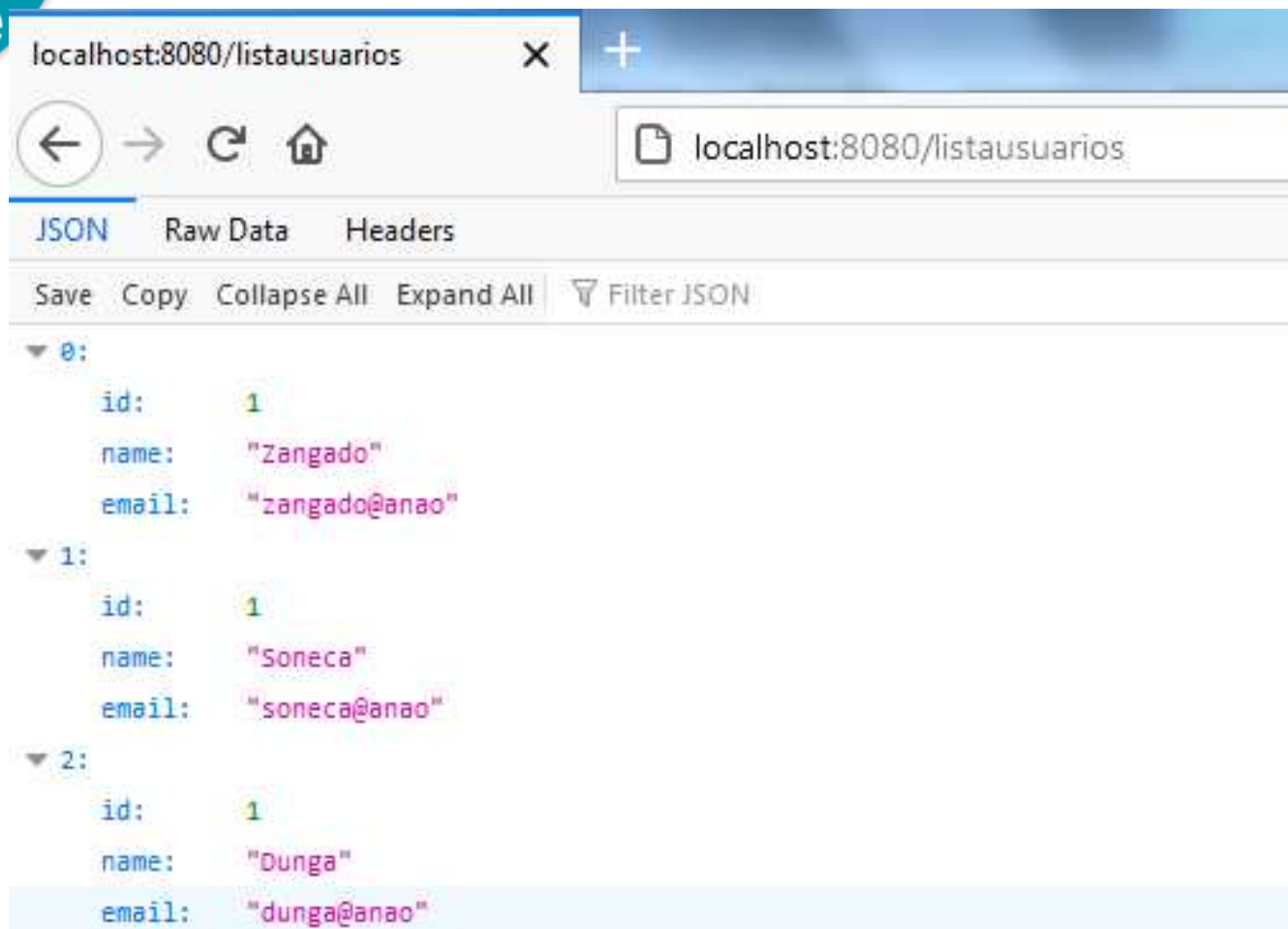
```
1 package com.redesocial.redesocial.controller.dto;
2
3 import com.redesocial.redesocial.modelo.User;
4
5 public class UserDTO {
6
7     private Long id;
8     private String name;
9     private String email;
10
11     public UserDTO(User user) {
12         id = user.getId();
13         name = user.getName();
14         email = user.getEmail();
15     }
16
17     public Long getId() {
18         return id;
19     }
20     public String getName() {
21         return name;
22     }
23     public String getEmail() {
24         return email;
25     }
26
27 }
28
```


Controller ListaUsers

```
redesocial/pom.x...  ListaUsersControll...  UserDTO.java  User.java  RedesocialApplica...

1 package com.redesocial.redesocial.controller;
2
3 import java.util.ArrayList;
4
5 import org.springframework.web.bind.annotation.RequestMapping;
6 import org.springframework.web.bind.annotation.RestController;
7
8 import com.redesocial.redesocial.controller.dto.UserDTO;
9 import com.redesocial.redesocial.modelo.User;
10
11 @RestController
12 public class ListaUsersController {
13
14     @RequestMapping("/listausuarios")
15     public ArrayList<UserDTO> listaUsuarios() {
16
17         ArrayList<UserDTO> list = new ArrayList<UserDTO>();
18
19         list.add(new UserDTO(new User((long) 1, "Zangado", "zangado@anao", "zangado")));
20         list.add(new UserDTO(new User((long) 1, "Soneca", "soneca@anao", "soneca")));
21         list.add(new UserDTO(new User((long) 1, "Dunga", "dunga@anao", "dunga")));
22
23         return list;
24     }
25
26 }
27
```

localhost:8080/listausuarios



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/listausuarios'. The browser's developer tools are open, showing the 'JSON' tab. The response is a JSON array with three user objects. Each object has 'id', 'name', and 'email' properties. The first user is 'Zangado', the second is 'Soneca', and the third is 'Dunga'. The third user object is highlighted in blue.

```
{
  "0": {
    "id": 1,
    "name": "Zangado",
    "email": "zangado@anao"
  },
  "1": {
    "id": 1,
    "name": "Soneca",
    "email": "soneca@anao"
  },
  "2": {
    "id": 1,
    "name": "Dunga",
    "email": "dunga@anao"
  }
}
```

Um Controller “vivo”

- Um Controller é um objeto
- Pode possuir atributos
- Ter comportamentos em função dos atributos
- Exercício
 - Faça um pequeno Controller que tenha um atributo privado
 - Faça um método que retorne este atributo

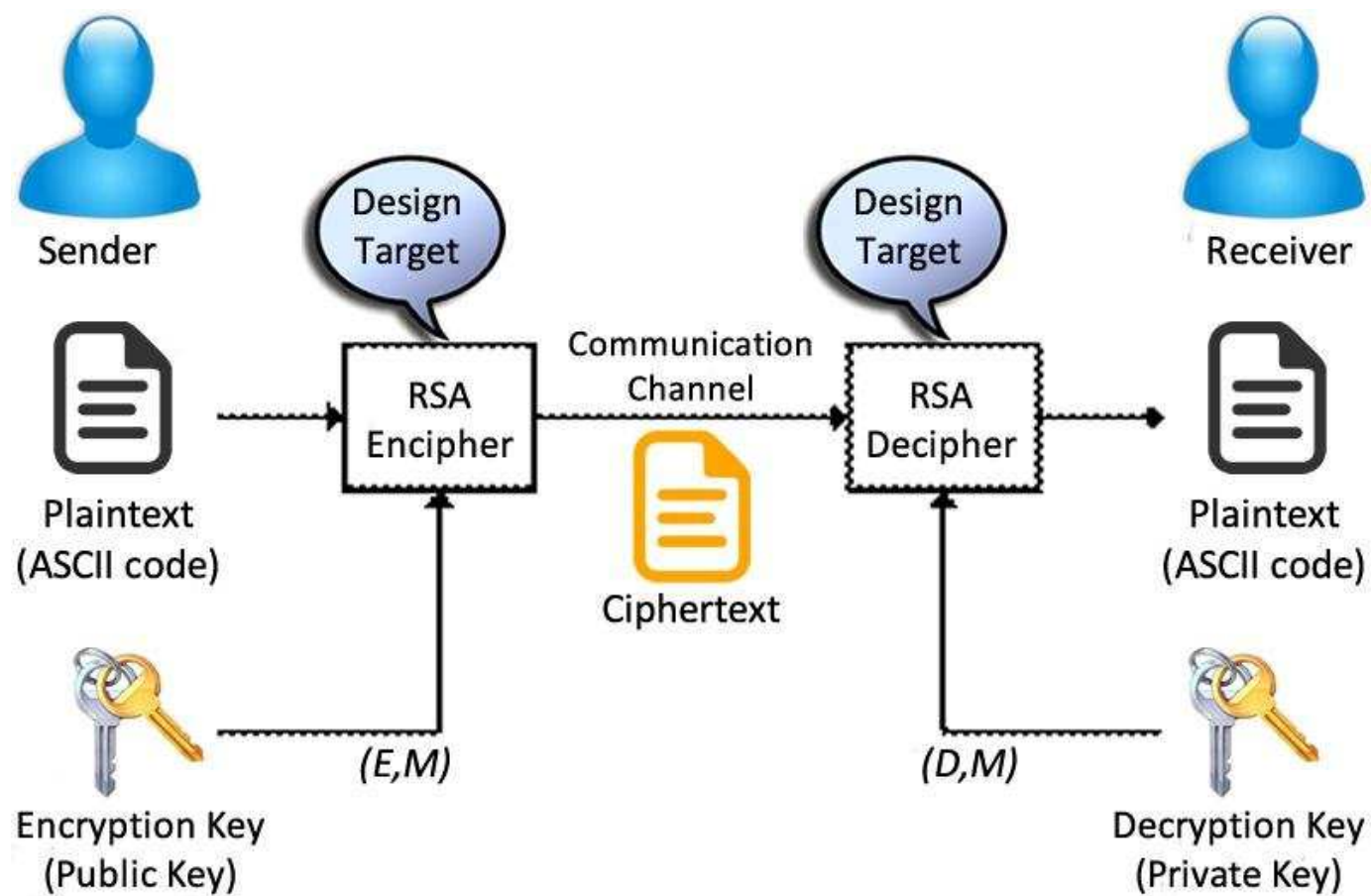
Pequeno desafio

- Crie um gerador de números aleatórios
 - Endpoint: /aleatorio
- Crie um gerador de números em sequência
 - Endpoint: /sequencia
- Crie um gerador de números primos em sequência
 - Endpoint: /sequenciaprimos

Grande desafio

- RSA
- Algoritmo de criptografia
- Virtualmente inquebrável
- Chave pública e chave privada
- Comunicação perfeitamente segura

RSA



As chaves RSA

- Chave (pública e privada)
 - Um número-chave e um módulo
- O módulo é comum às duas chaves
- $M = P * Q$
 - Onde
 - P e Q são números primos gigantes
 - P e Q são usados para gerar os números-chave




Quebrar o RSA

- “Fácil”
- Descobrir P e Q
- Basta fatorar M ($M=P*Q$)
- Custo computacional gigante


Seu desafio

- Disponibilizar um gerador de chaves RSA
- Endpoint: /gerachavesrsa
- Vai precisar criar duas classes de processamento
 - Key.java : que implementa o processamento das chaves
 - RSAGenerator.java: que implementa o código RSA
 - Interessa-nos os métodos getPrivateKey e getPublicKey

Executando

 Introdução  aiforno elétrico  Nova aba

JSON Dados brutos Cabeçalhos

Salvar Copiar Recolher tudo Expandir tudo  Filtrar JSON

▼ privateKey:

- component: 1.1758434494916765e+38
- modulus: 1.4949327105343248e+38

▼ publicKey:

- component: 18211198766471470000
- modulus: 1.4949327105343248e+38

Baixe as classes Key.java e
RSAGenerator.java do nosso drive!
Arquivo RSA.zip