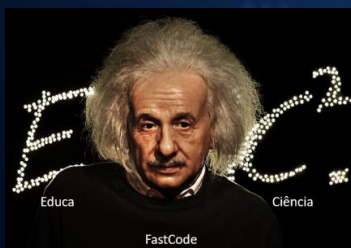


Java: Rest de maneira descomplicada

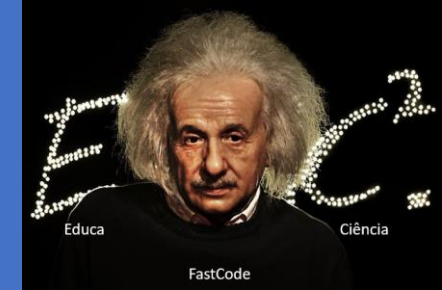
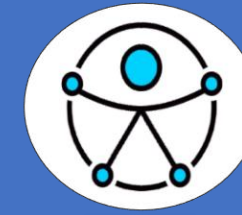


2022

Fábio Perucello



Palestrante



LIVE

Java-Conceito e Fundamento de Maneira Descomplicada

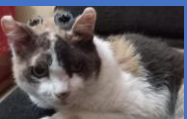
31.08.2022 • 21H

www.explorandoti.com.br/eventos

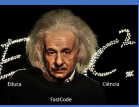
Fábio Perucello

ODM - Operational Decision Management -
Developer (IBM) | Instrutor Java (Evolua
Sumaré)

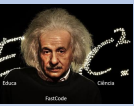
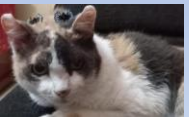
EXPLORANDO TI



Fábio Perucello



Java: Descomplicando Rest



Java: Descomplicando Rest

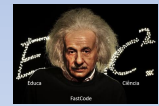
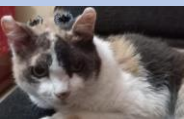
Vamos entender primeiro o que significa a palavra REST:

REST -> Representational State Transfer , ou simplesmente Transferência de Estado Representacional, foi criado por Roy Fielding.

Vamos além , podemos simplesmente dizer que é um “estilo de arquitetura de software” muito usado hoje em dia , até porque a maior virtude quando dizemos REST se define em dois pontos:

1) *Criar serviços Web*

2) *Facilitar a Integração deste serviço junto aos “Sistemas”*

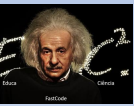
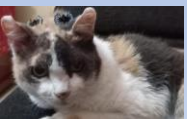


Java: Descomplicando Rest

Dizemos assim que nossa arquitetura de desenvolvimento é baseada num conceito “REST”, onde ao “criarmos” serviços , onde eles deverão retornar dados , normalmente em “xml” ou “json” e para isso , a Arquitetura REST se baseia-se por protocolos HTTP , protocolos estes que tem como principal objetivo “fazer integração” com outros Sistemas (SOAP, WSDL)

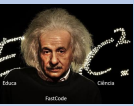
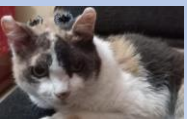
Protocolos:

- ✓ **GET** – consultar
- ✓ **POST** – Inserir
- ✓ **PUT** – Atualizar (também veremos com PATCH)
- ✓ **DELETE** – Deletar

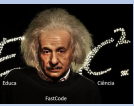
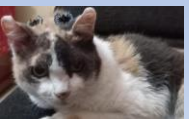
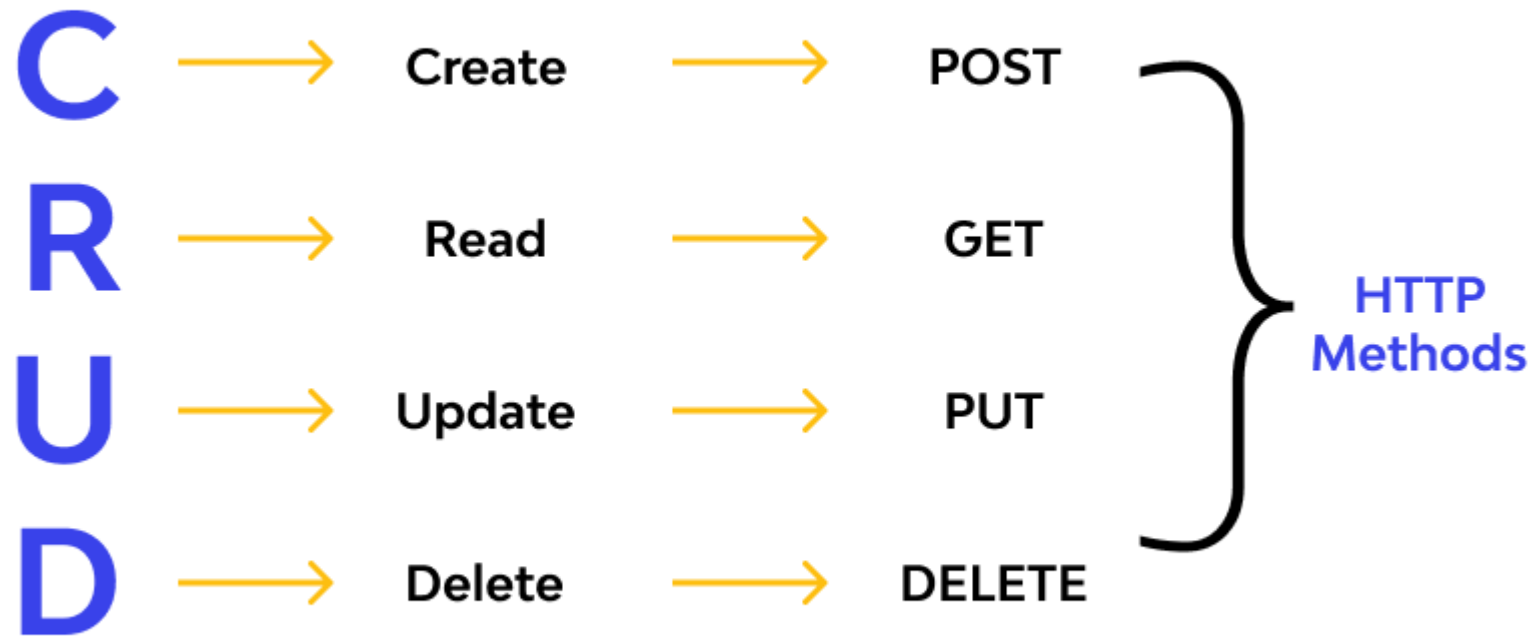


Java: Descomplicando Rest

Método	Função
GET	Recuperar qualquer informação especificada pela URI
HEAD	Idêntico ao GET, porém contendo apenas o cabeçalho em sua resposta
POST	Adiciona um novo subordinado ao recurso especificado pelo URI
PUT	Solicita armazenamento da entidade na URI especificada
DELETE	Apaga o recurso especificado
TRACE	Retorna a própria requisição para fins de auditoria
OPTIONS	Retorna os métodos disponível em determinada URI
CONNECT	Utilizado para converter conexões HTTP em túneis para hosts remotos
PATCH	Atualiza parcialmente o recurso especificado



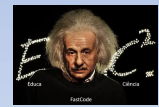
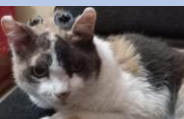
Java: Descomplicando Rest



Java: Descomplicando Rest

Principais Características REST:

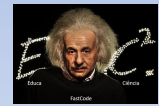
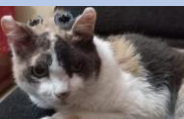
- a) Cada requisição de serviço deve retornar de forma independente e “não” deve ter estado.
- b) Operações Padronizadas -> GET , POST, PUT , DELETE
- c) Utiliza-se de uma “**uri**” (Uniform Resource Identifier) ou seja , uma sintaxe universal para identificar os recursos (também conhecido como endpoint).
- d) Utilização de tipos de conteúdo para solicitar um conteúdo (Request e Response) -> xml ou json



Java: Descomplicando Rest

Protocolo HTTP -> desenvolvido em 1989 por Tim Berners-Lee, o Hypertext Transfer Protocol.

- a) É um protocolo de camada de aplicação amplamente utilizado na web para sistemas distribuídos em hipermídia.
- b) HTTP carrega como requisito o emprego de um protocolo de camada de transporte resiliente a erros e inconsistências da rede onde a opção mais comum para o caso é o TCP.
- c) HTTP não se limita à definição do formato de requisição e resposta, mas também define uma série de conceitos essenciais para sua implantação.



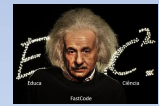
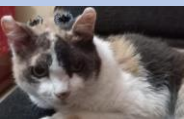
Java: Descomplicando Rest

Protocolo HTTP -> A sessão HTTP compreende todo ciclo de processamento, usualmente dividido em três etapas:

- ✓ *Uma conexão de camada de transporte é estabelecida entre cliente e servidor.*
- ✓ *O cliente envia uma requisição e aguarda resposta.*
- ✓ *Após processar a requisição, o servidor envia uma resposta contendo os dados requisitados e um código de status*

Neste processo estão envolvidos os dois tipos de mensagens suportados pelo protocolo:

1. **Request** -> Requisição
2. **Response** -> Resposta

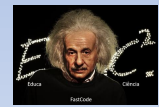
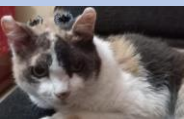


Java: Descomplicando Rest

Principais Características REST:

Utiliza-se de uma “uri” (Uniform Resource Identifier) ou seja , uma sintaxe universal para identificar os recursos (também conhecido como endpoint).

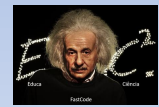
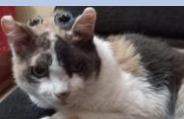
- **URI** – uniform resource identifier (identificador uniforme do recurso -> disponibilizado por um servidor web, tem um nome associado de forma que os clientes consigam acessar os recursos de interesse.
- **URL** – uniform resource locator (localizador uniforme do recurso) é o subtipo mais comum de URI.



Java: Descomplicando Rest

3 Regrinhas Básicas:

- 1) A primeira é responsável por descrever o protocolo utilizado para acesso ao recurso:
 - `http://` ou `https://`
- 2) A segunda o nome do servidor acessado:
 - `www.google.com`
- 3) A terceira todo o texto após o nome do servidor e identifica localmente o recurso:
 - `/produtos/descricao`

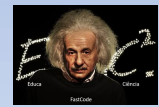
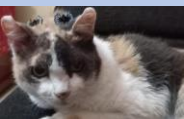


Java: Descomplicando Rest

O acrônimo API é a abreviação de Application Programming Interface, que significa:

✓ “Interface de Programação de Aplicações”

Representa um conjunto de rotinas e padrões estabelecidos e documentados para que uma determinada aplicação de software tenha autorização para utilizar as funcionalidades sem precisar conhecer todo o processo desta implementação.



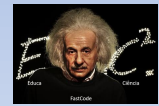
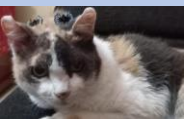
Java: Descomplicando Rest

Podemos dizer que quando pensamos em URL , certamente associamos ao acesso à alguma coisa , certo?

Exemplo, aquela pergunta básica, me passa a “URL” para acessar determinado produto, ou , me passa o “endpoint” !

Podemos dizer também que “aquela” URL que você acessa determinado site ou informação que esteja hospedada em um determinado servidor em resumo essa URL é nosso “recurso”.

É tarefa fundamental de toda API receber e processar requisições, respondendo-as de alguma forma. Para entender melhor como isso é feito em APIs RESTful devemos aprender sobre a semântica dos verbos HTTP.



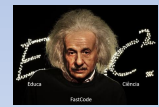
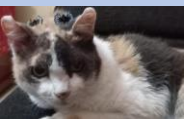
Java: Descomplicando Rest

Quando começamos a desenvolver – ou consumir - nossos primeiros serviços RESTful, a primeira coisa que precisamos entender é o papel dos verbos HTTP.

A ideia geral é a seguinte: seu serviço vai prover uma url base e os verbos HTTP vão indicar qual ação está sendo requisitada pelo consumidor do serviço.

Por exemplo, considerando a URL `www.dominio.com/rest/produto/`, se enviarmos para ela uma requisição HTTP utilizando o verbo GET, provavelmente obteremos como resultado uma listagem de registros (dos produtos).

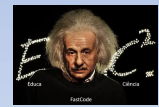
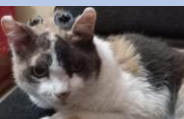
Por outro lado, se utilizarmos o verbo POST, provavelmente estaremos tentando adicionar um novo produto, cujos dados serão enviados no corpo da requisição.



Java: Descomplicando Rest

Note que não passamos nenhuma conexão com banco de dados , apenas uma URL que, depois de devidamente desenvolvida , teremos a segurança de ter os dados acessados.

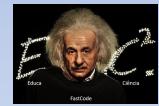
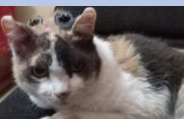
O interessante , é que podemos controlar os níveis de acesso , como usuário e senha , token de acesso , etc ... Mas isso fica para uma live mais técnica , nesse momento estamos conhecendo melhor o que é REST.

The logo consists of the text "{ REST:API }" in a bold, blue, sans-serif font. The text is centered within a white rectangular box.

Java: Descomplicando Rest

Vale ressaltar , que o Java é extremamente poderoso para essa finalidade , no entanto, o Java disponibiliza uma IDE para que podemos desenvolver nossa API -> mais conhecida como STS - Spring Tool Suite.

“O Spring Tool Suíte é uma IDE baseada em Eclipse que dá algumas facilidades para trabalhos com o Spring no geral. Uma das coisas legais é que ele nos ajuda a criar projetos com Spring Boot. Mas, o STS não é pré-requisito para criação de projetos com Spring Boot.”

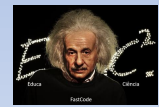
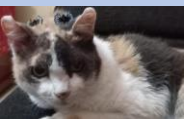


Java: Descomplicando Rest

Hoje em dia, está cada vez mais comum termos aplicações que funcionam online, em navegadores ou dispositivos móveis. Essas aplicações tem por objetivo consumir informação por meio de interfaces que implementam uma série rotinas e padrões que chamamos de API.

Podemos resumir ainda mais , onde API é um conjunto de padrões e regras muito bem documentadas para que uma aplicação “**tal 1**” possa utilizar funcionalidades da aplicação “**tal 2**”.

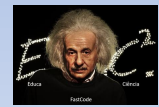
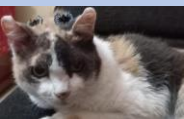
Vamos tentar explicar melhor , a Evolua tem um e-commerce , desenvolvedores que trabalham na solução tem como objetivo criar a dinâmica da loja, como criar, atualizar, deletar os produtos internamente e mostrar os produtos para os clientes. Essas funcionalidades podem ser criadas em uma aplicação do lado do servidor como se fosse uma API, de forma que o site do e-commerce possa usar essas informações.



Java: Descomplicando Rest

Atualmente o protocolo de comunicação na Web é o HTTP.

Ele funciona como um protocolo de requisição-resposta em um modelo que chamamos de cliente-servidor, onde no exemplo anterior da Loja da Evolua “e-commerce”, o navegador que é usado para acessar o site seria o cliente e o computador ou máquina virtual em algum serviço de cloud em que a API está hospedada é o servidor, assim, nosso cliente manda uma requisição HTTP para o servidor e o servidor, com recursos e conteúdos próprios, retorna uma mensagem de resposta para o cliente sobre o curso !



Java: Descomplicando Rest

Baseado nesses métodos que falamos (GET , PUT , POST , DELETE) , podemos afirmar que o servidor deve processar cada uma das requisições e retornar uma resposta adequada.

O conteúdo da resposta pode estar no formato XML, JSON, YAML, texto, etc..

Podemos classificar essa resposta em 5 grupos ou 5 retornos mas eu chamo de “famílias”:

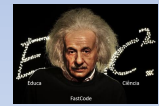
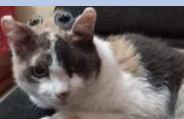
1XX — Informações Gerais

2XX — Sucesso

3XX — Redirecionamento

4XX — Erro no cliente

5XX — Erro no servidor

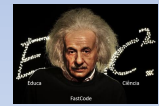
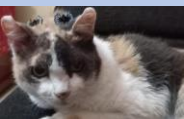


Java: Descomplicando Rest

Recordando que REST é uma abstração da arquitetura , é um estilo de arquitetura de software que define uma série de restrições para a criação de web services (serviços Web), ou seja, **restringe como seus componentes devem interagir entre si.**

** Esse termo foi introduzido e definido por Roy Fielding em sua tese de doutorado no final dos anos 90 e início dos anos 2000.*

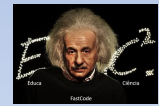
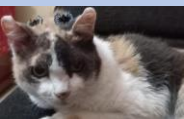
Fielding definiu que os princípios REST que eram conhecidos como “**modelo de objeto HTTP**” e passaram a ser utilizados no projeto dos padrões HTTP 1.1 e URI (Uniform Resource Identifiers), dessa forma, podemos dizer que em sua semântica, o REST utiliza-se métodos HTTP.



Java: Descomplicando Rest

RESTful -> Podemos afirmar que uma API é RESTful, desde que garantimos que implementação da API está de acordo com essa arquitetura REST.

Conceitualmente, nos serviços RESTful, tanto os dados quanto as funcionalidades são considerados recursos e ficam acessíveis aos clientes através da utilização de URIs.



Java: Descomplicando Rest

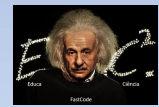
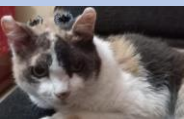
Artigo interessante que compartilho:

<http://www.ricardoluis.com/wp-content/uploads/2015/08/Artigo-WebServices-em-REST.pdf>

Lembrando que também como EducaCiência temos diversos artigos que abordamos esse tema e até projetos didáticos.

<https://github.com/perucello/Artigos-EducaCiencia> FastCode

<https://github.com/perucello?tab=repositories>



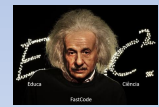
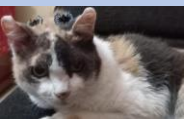
Java: Descomplicando Rest

Basicamente, **API** é um conjunto de padrões ou rotinas de uma determinada linguagem de programação que permite a construção de aplicativos/Sistemas.

Podemos dizer de uma forma simples que API é a MATRIX dos aplicativos enfim, uma interface que “roda” ou “executa” por trás de tudo.

Em um resumo simples, a API funciona através de comunicação de diversos códigos onde tem por finalidade definir o comportamento específico de cada objeto relacionado a sua interface.

A API tende a realizar interação com diversas funções como por exemplo simples um site, como por exemplo a busca de imagens, notícias, artigos, etc.



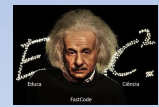
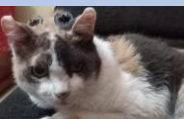
Java: Descomplicando Rest

De uma maneira geral, a API é composta por uma série de funções acessíveis por intermédio da Linguagem de Programação e que permite utilizar de característica do software menos evidentes como um exemplo, um Sistema Operacional, onde há diversas funções acopladas no funcionamento que permite o programador criar Janelas, conceder acesso à arquivos, etc.

API Web - É um conjunto de interfaces no contexto de desenvolvimento Web, uma API é um conjunto definido de mensagens de requisição e resposta HTTP, geralmente expresso nos formatos XML ou JSON.

-> A chamada Web 2.0 vem abandonando o modelo de serviços SOAP em favor da técnica REST.

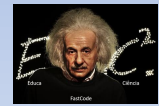
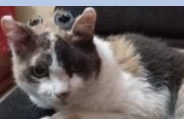
REST - acrônimo de Representational State Transfer, e tem como objetivo primário a definição de características fundamentais para a construção de aplicações Web seguindo boas práticas.



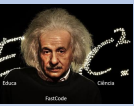
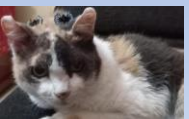
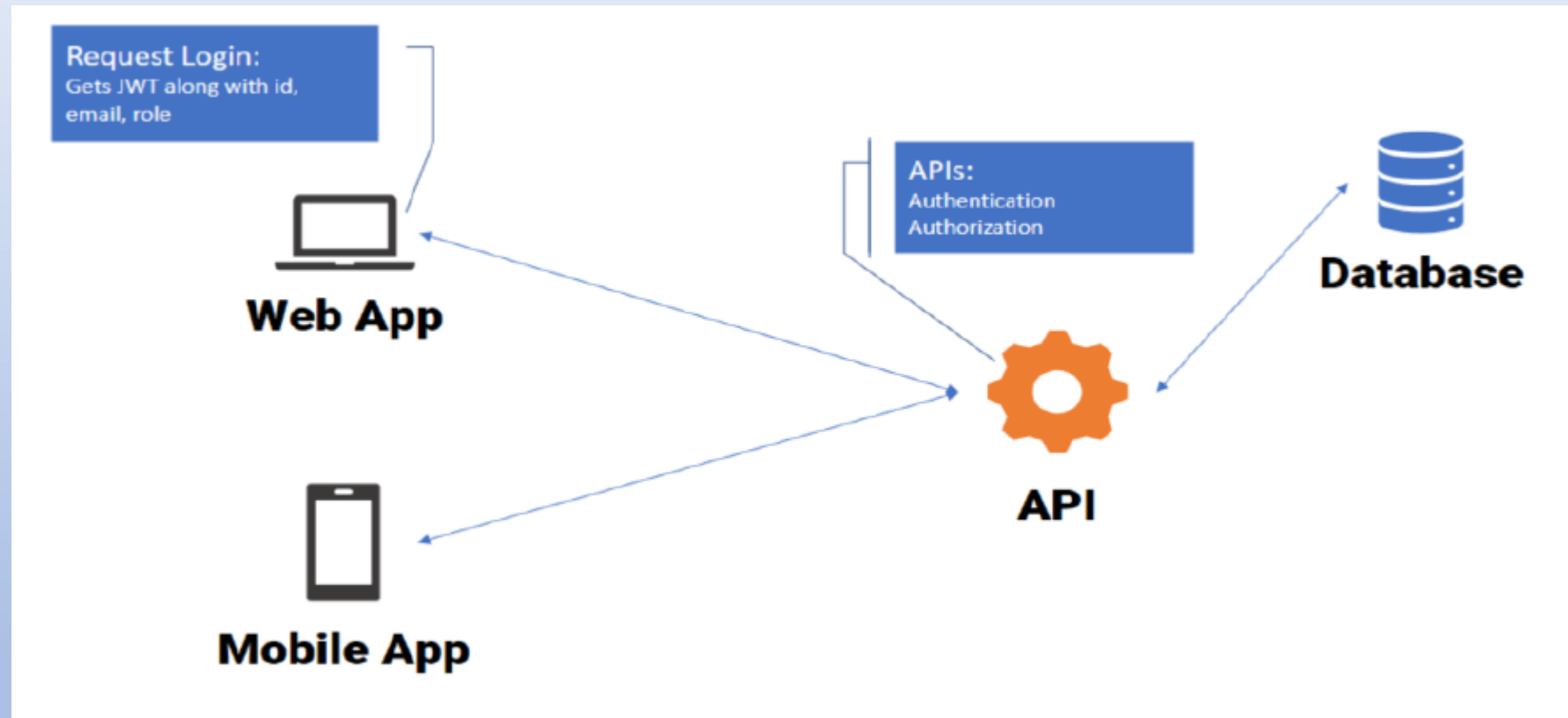
Java: Descomplicando Rest

Exemplo de conceito REST

- I. Você entra com um endereço em seu navegador (Chrome, Firefox, Edge, etc).*
- II. Seu navegador estabelece uma conexão TCP/IP com o servidor de destino e envia uma requisição GET HTTP com o endereço que você digitou.*
- III. O servidor interpreta sua requisição e de acordo com o que foi solicitado, uma resposta HTTP é retornada ao seu navegador.*
- IV. A resposta retornada pode ser de sucesso, contendo alguma representação em formato HTML, ou pode ser algum erro, como por exemplo o famoso 404 Not Found, que indica que o endereço/recurso que você solicitou não pôde ser encontrado.*
- V. Em caso de sucesso, o seu navegador interpreta o HTML e você consegue navegar pela página renderizada.*

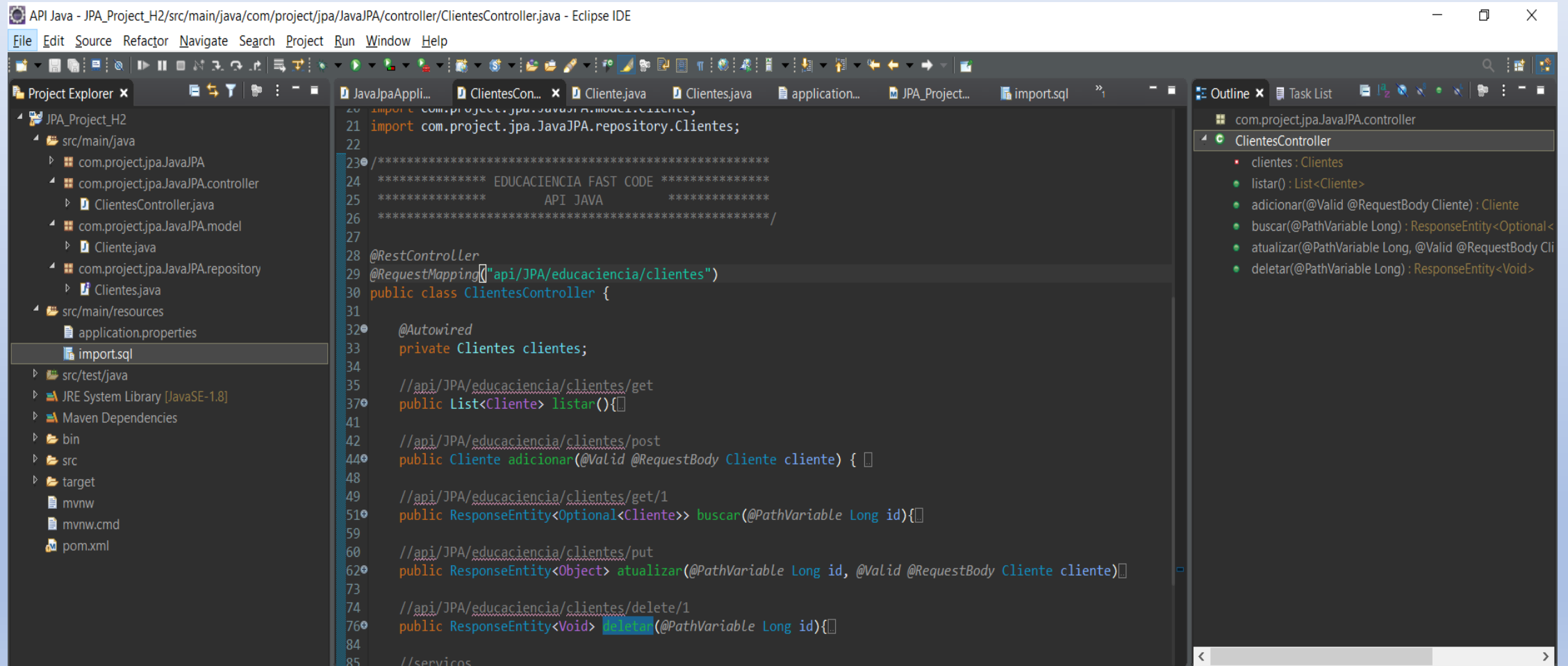


Java: Descomplicando Rest



Java: Descomplicando Rest

Exemplo -> Criamos uma API em Java

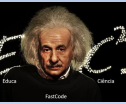


The screenshot displays the Eclipse IDE interface for a Java project named 'JPA_Project_H2'. The main editor shows the file 'ClientesController.java' with the following code:

```
20 import com.project.jpa.javaJPA.model.Cliente;  
21 import com.project.jpa.javaJPA.repository.Clientes;  
22  
23 /***** EDUCACIENCIA FAST CODE *****/  
24 ***** API JAVA *****  
25 *****  
26  
27  
28 @RestController  
29 @RequestMapping("api/JPA/educaciencia/clientes")  
30 public class ClientesController {  
31  
32     @Autowired  
33     private Clientes clientes;  
34  
35     //api/JPA/educaciencia/clientes/get  
36     public List<Cliente> listar(){}  
37  
38  
39     //api/JPA/educaciencia/clientes/post  
40     public Cliente adicionar(@Valid @RequestBody Cliente cliente) {  
41  
42  
43  
44  
45  
46  
47  
48  
49     //api/JPA/educaciencia/clientes/get/1  
50     public ResponseEntity<Optional<Cliente>> buscar(@PathVariable Long id){  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60     //api/JPA/educaciencia/clientes/put  
61     public ResponseEntity<Object> atualizar(@PathVariable Long id, @Valid @RequestBody Cliente cliente){  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74     //api/JPA/educaciencia/clientes/delete/1  
75     public ResponseEntity<Void> deletar(@PathVariable Long id){  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85     //services
```

The Project Explorer on the left shows the project structure, including the 'com.project.jpa.javaJPA.controller' package containing 'ClientesController.java'. The Outline on the right lists the methods of the 'ClientesController' class:

- clientes : Clientes
- listar() : List<Cliente>
- adicionar(@Valid @RequestBody Cliente) : Cliente
- buscar(@PathVariable Long) : ResponseEntity<Optional<Cliente>>
- atualizar(@PathVariable Long, @Valid @RequestBody Cliente) : ResponseEntity<Object>
- deletar(@PathVariable Long) : ResponseEntity<Void>



Java: Descomplicando Rest

Exemplo -> Consumimos os dados via Postman - Get

GET <http://localhost:8080/api/JPA/educaciencia/clientes/get> Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

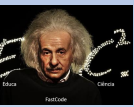
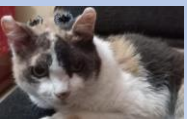
KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 1490 ms Size: 451 B Save Response

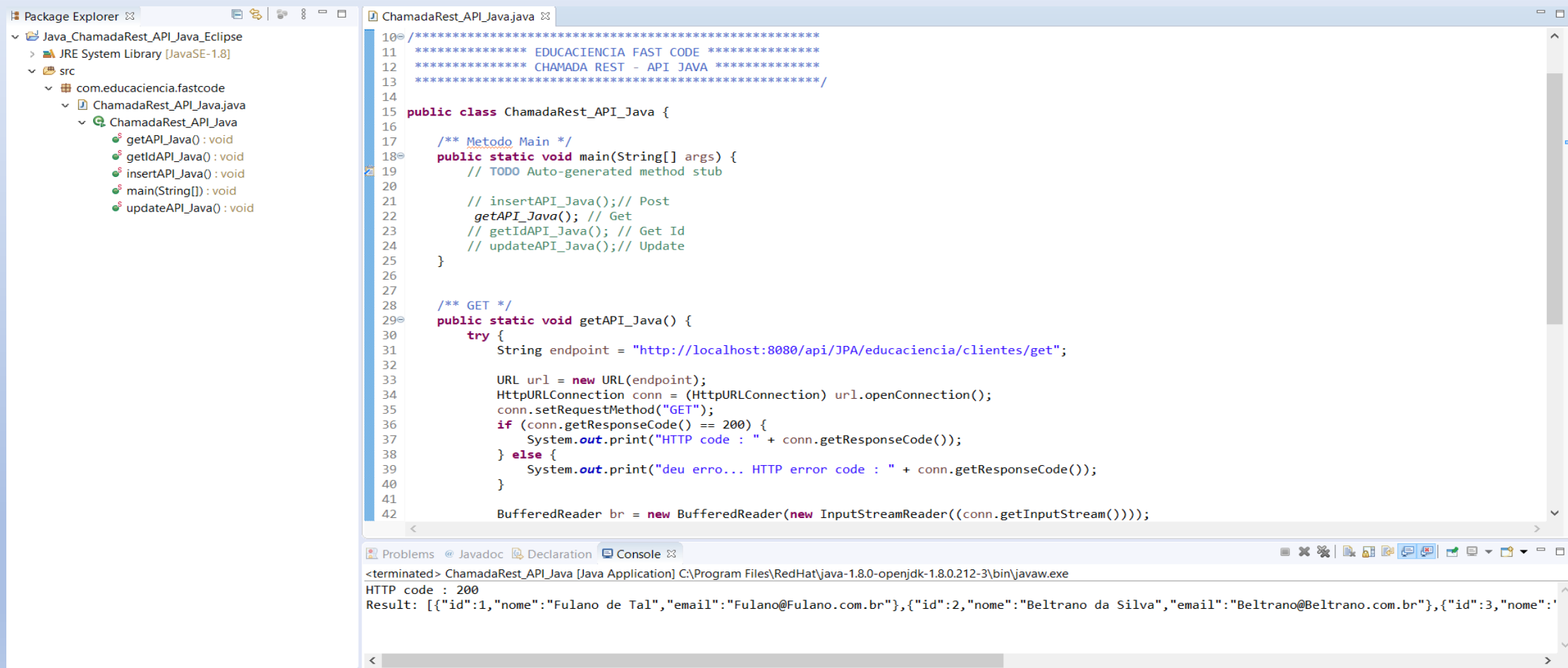
Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "id": 1,
4     "nome": "Fulano de Tal",
5     "email": "Fulano@Fulano.com.br"
6   },
7   {
8     "id": 2,
9     "nome": "Beltrano da Silva",
10    "email": "Beltrano@Beltrano.com.br"
11  },
12  {
13    "id": 3,
14    "nome": "Ciclano Souza",
15    "email": "Ciclano@Ciclano.com.br"
16  },
17  {
18    "id": 4,
19    "nome": "EducaCiencia FastCode",
20    "email": "educaciencia_fastcode@outlook.com.br"
21  }
22 }
```



Java: Descomplicando Rest

Nossa API está no ar , vamos consumir por uma aplicação Java

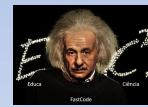


The screenshot shows the Eclipse IDE with a project named 'Java_ChamadaRest_API_Java_Eclipse'. The Package Explorer on the left shows the package structure: 'com.educaciencia.fastcode' containing 'ChamadaRest_API_Java.java'. The main editor displays the source code of 'ChamadaRest_API_Java.java', which includes a main method and a 'getAPI_Java()' method. The main method calls 'getAPI_Java()' and prints the result. The 'getAPI_Java()' method uses 'URLConnection' to make a GET request to 'http://localhost:8080/api/JPA/educaciencia/clientes/get'. The Console at the bottom shows the output of the application, indicating a successful GET request with a 200 status code and a JSON array of client data.

```
10 //*****  
11 //***** EDUCACIENCIA FAST CODE *****  
12 //***** CHAMADA REST - API JAVA *****  
13 //*****  
14  
15 public class ChamadaRest_API_Java {  
16  
17     /** Metodo Main */  
18     public static void main(String[] args) {  
19         // TODO Auto-generated method stub  
20  
21         // insertAPI_Java(); // Post  
22         getAPI_Java(); // Get  
23         // getIdAPI_Java(); // Get Id  
24         // updateAPI_Java(); // Update  
25     }  
26  
27  
28     /** GET */  
29     public static void getAPI_Java() {  
30         try {  
31             String endpoint = "http://localhost:8080/api/JPA/educaciencia/clientes/get";  
32  
33             URL url = new URL(endpoint);  
34             HttpURLConnection conn = (HttpURLConnection) url.openConnection();  
35             conn.setRequestMethod("GET");  
36             if (conn.getResponseCode() == 200) {  
37                 System.out.print("HTTP code : " + conn.getResponseCode());  
38             } else {  
39                 System.out.print("deu erro... HTTP error code : " + conn.getResponseCode());  
40             }  
41  
42             BufferedReader br = new BufferedReader(new InputStreamReader((conn.getInputStream())));
```

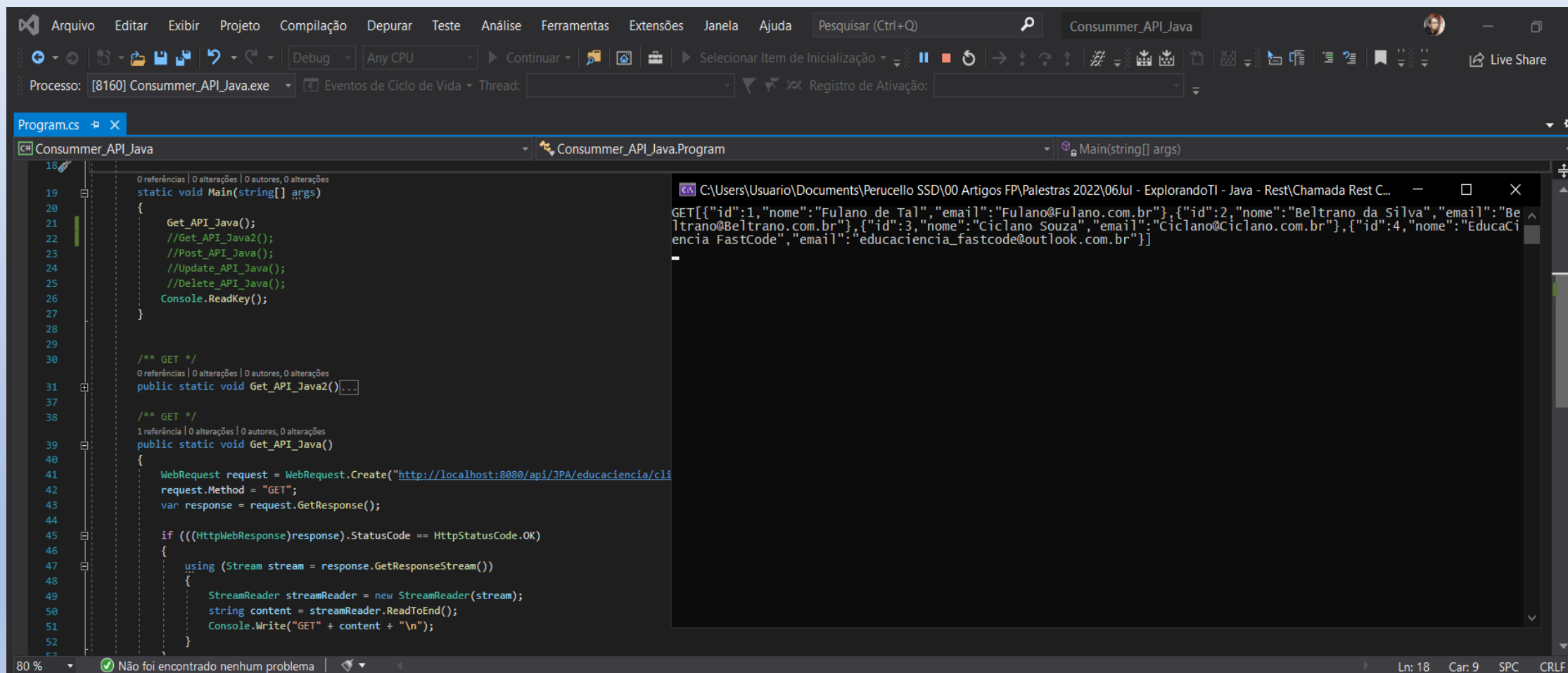
Console Output:

```
<terminated> ChamadaRest_API_Java [Java Application] C:\Program Files\RedHat\java-1.8.0-openjdk-1.8.0.212-3\bin\javaw.exe  
HTTP code : 200  
Result: [{"id":1,"nome":"Fulano de Tal","email":"Fulano@Fulano.com.br"}, {"id":2,"nome":"Beltrano da Silva","email":"Beltrano@Beltrano.com.br"}, {"id":3,"nome":":
```



Java: Descomplicando Rest

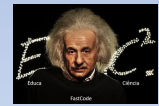
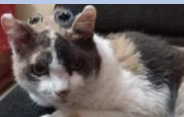
Nossa API está no ar , vamos consumir por uma aplicação C#



The screenshot displays the Visual Studio IDE with a C# application named 'Consumer_API_Java'. The code is in 'Program.cs' and shows a 'Main' method that calls 'Get_API_Java()' and 'Get_API_Java2()'. The 'Get_API_Java2()' method uses 'WebRequest' to perform a GET request to 'http://localhost:8080/api/JPA/educaciencia/cli'. The response is read as a stream and written to the console. A separate window shows the JSON response of the GET request, which is an array of four objects representing users.

```
18
19 static void Main(string[] args)
20 {
21     Get_API_Java();
22     //Get_API_Java2();
23     //Post_API_Java();
24     //Update_API_Java();
25     //Delete_API_Java();
26     Console.ReadKey();
27 }
28
29
30 /** GET */
31 0 referências | 0 alterações | 0 autores, 0 alterações
32 public static void Get_API_Java2()...
33
34 /** GET */
35 1 referência | 0 alterações | 0 autores, 0 alterações
36 public static void Get_API_Java()
37 {
38     WebRequest request = WebRequest.Create("http://localhost:8080/api/JPA/educaciencia/cli");
39     request.Method = "GET";
40     var response = request.GetResponse();
41
42     if (((HttpWebResponse)response).StatusCode == HttpStatusCode.OK)
43     {
44         using (Stream stream = response.GetResponseStream())
45         {
46             StreamReader streamReader = new StreamReader(stream);
47             string content = streamReader.ReadToEnd();
48             Console.WriteLine("GET" + content + "\n");
49         }
50     }
51 }
52
53
```

GET[{"id":1,"nome":"Fulano de Tal","email":"Fulano@Fulano.com.br"}, {"id":2,"nome":"Beltrano da Silva","email":"Beltrano@Beltrano.com.br"}, {"id":3,"nome":"Ciclano Souza","email":"Ciclano@Ciclano.com.br"}, {"id":4,"nome":"Educaciencia FastCode","email":"educaciencia_fastcode@outlook.com.br"}]

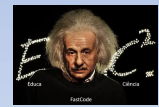
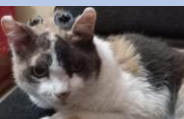


Java: Descomplicando Rest

Ainda tem mais, atualmente para facilitar o processo , algumas empresas aderiram à documentações online , no caso , vamos mostrar o SWAGGER.

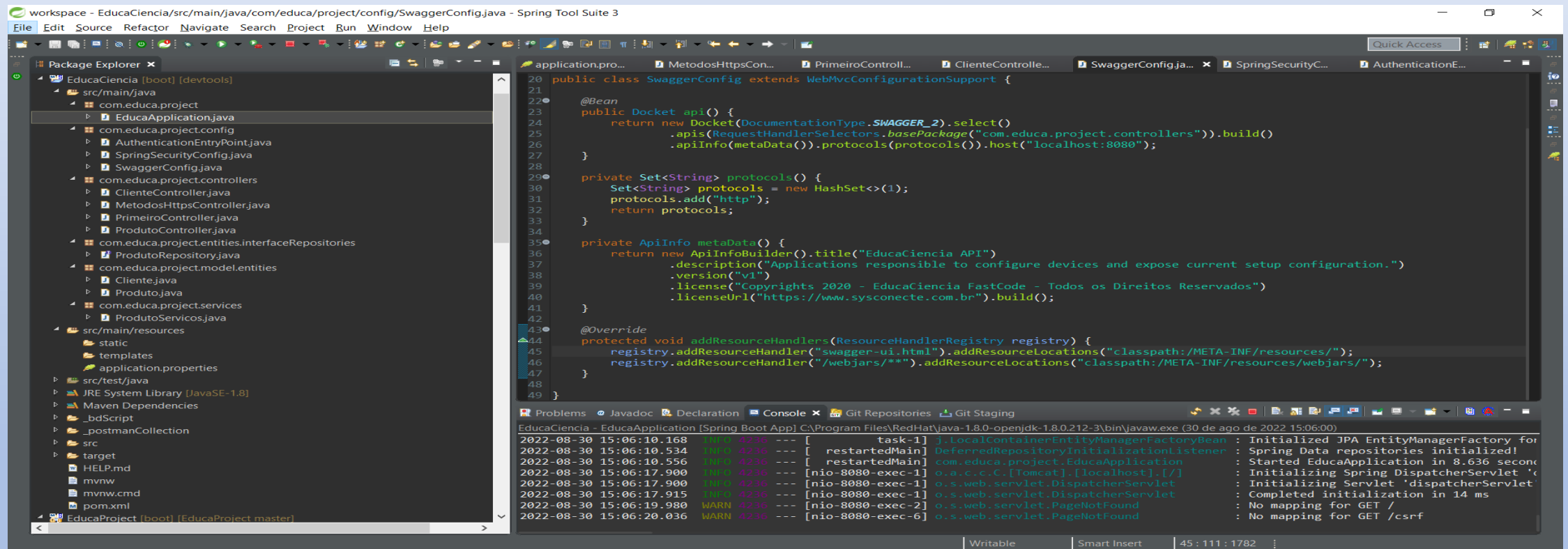
O Swagger é um framework composto por diversas ferramentas que, independente da linguagem, auxilia a descrição, consumo e visualização de serviços de uma API REST.

A interface do usuário do Swagger permite que qualquer pessoa - seja sua equipe de desenvolvimento ou seus consumidores finais - visualize e interaja com os recursos da API sem ter nenhuma lógica de implementação em vigor. Ele é gerado automaticamente a partir de sua especificação OpenAPI (anteriormente conhecida como Swagger), com a documentação visual facilitando a implementação de back-end e o consumo do lado do cliente. <https://swagger.io/tools/swagger-ui/>



Java: Descomplicando Rest

Neste exemplo, estamos iniciando uma aplicação Java SpringBoot onde configuramos a documentação SWAGGER.



```
workspace - EducaCiencia/src/main/java/com/educa/project/config/SwaggerConfig.java - Spring Tool Suite 3
File Edit Source Refactor Navigate Search Project Run Window Help

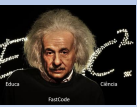
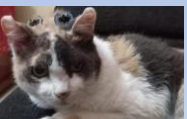
Package Explorer
EducaCiencia [boot] [devtools]
  src/main/java
    com.educa.project
      EducaApplication.java
    com.educa.project.config
      AuthenticationEntryPoint.java
      SpringSecurityConfig.java
      SwaggerConfig.java
    com.educa.project.controllers
      ClienteController.java
      MetodosHttpsController.java
      PrimeiroController.java
      ProdutoController.java
    com.educa.project.entities.interfaces
      Repositories
    com.educa.project.model.entities
      Cliente.java
      Produto.java
    com.educa.project.services
      ProdutoServicos.java
  src/main/resources
    static
    templates
    application.properties
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  _bdScript
  _postmanCollection
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
  EducaProject [boot] [EducaProject master]

application.pro... MetodosHttpsCon... PrimeiroControll... ClienteControll... SwaggerConfig.ja... SpringSecurityC... AuthenticationE...

20 public class SwaggerConfig extends WebMvcConfigurationSupport {
21
22 @Bean
23 public Docket api() {
24     return new Docket(DocumentationType.SWAGGER_2).select()
25         .apis(RequestHandlerSelectors.basePackage("com.educa.project.controllers")).build()
26         .apiInfo(metaData()).protocols(protocols()).host("localhost:8080");
27 }
28
29 private Set<String> protocols() {
30     Set<String> protocols = new HashSet<>(1);
31     protocols.add("http");
32     return protocols;
33 }
34
35 private ApiInfo metaData() {
36     return new ApiInfoBuilder().title("EducaCiencia API")
37         .description("Applications responsible to configure devices and expose current setup configuration.")
38         .version("v1")
39         .license("Copyrights 2020 - EducaCiencia FastCode - Todos os Direitos Reservados")
40         .licenseUrl("https://www.sysconecte.com.br").build();
41 }
42
43 @Override
44 protected void addResourceHandlers(ResourceHandlerRegistry registry) {
45     registry.addResourceHandler("swagger-ui.html").addResourceLocations("classpath:/META-INF/resources/");
46     registry.addResourceHandler("/webjars/**").addResourceLocations("classpath:/META-INF/resources/webjars/");
47 }
48
49 }
```

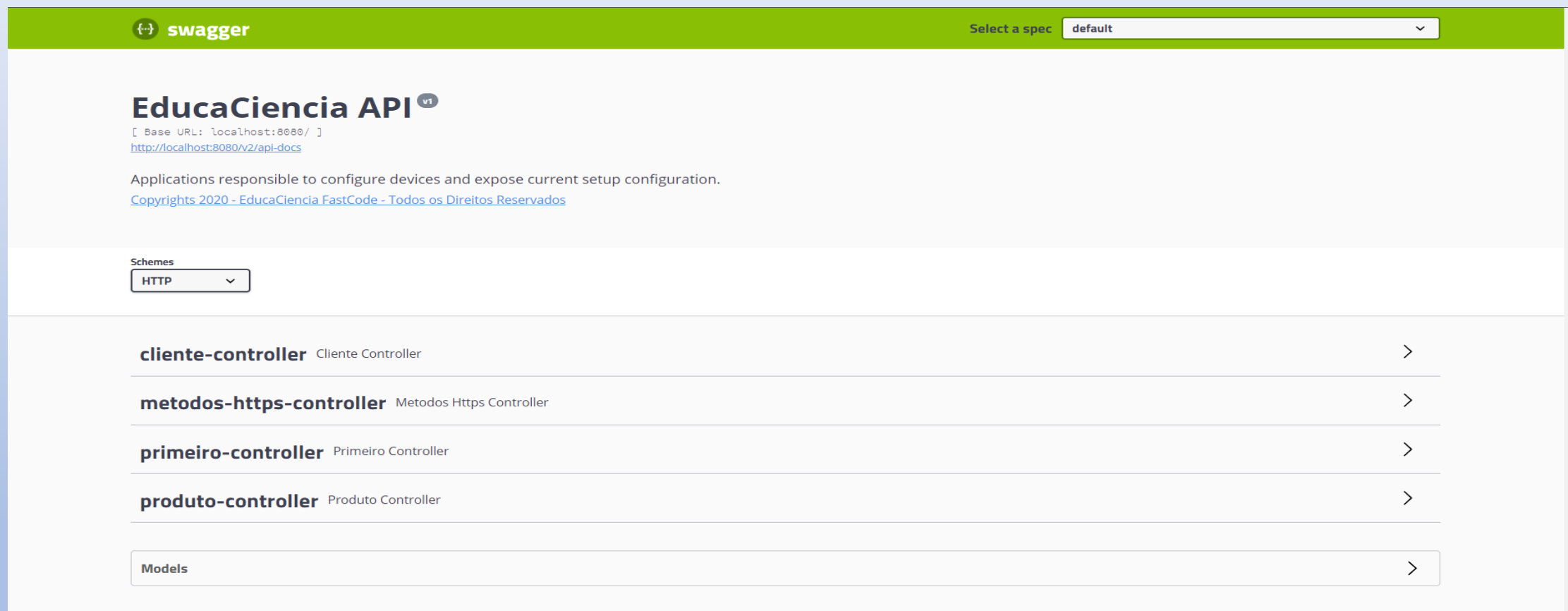
```
2022-08-30 15:06:10.168 INFO 4236 --- [task-1] J.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for
2022-08-30 15:06:10.534 INFO 4236 --- [task-1] DeferredRepositoryInitializationListener : Spring Data repositories initialized!
2022-08-30 15:06:10.556 INFO 4236 --- [task-1] restartedMain com.educa.project.EducaApplication : Started EducaApplication in 8.636 seconds
2022-08-30 15:06:17.900 INFO 4236 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2022-08-30 15:06:17.900 INFO 4236 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2022-08-30 15:06:17.915 INFO 4236 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 14 ms
2022-08-30 15:06:19.980 WARN 4236 --- [nio-8080-exec-2] o.s.web.servlet.PageNotFound : No mapping for GET /
2022-08-30 15:06:20.036 WARN 4236 --- [nio-8080-exec-6] o.s.web.servlet.PageNotFound : No mapping for GET /csrf
```

Fonte: Curso de Java Evolua Sumare



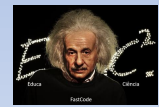
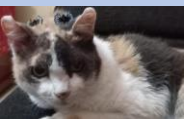
Java: Descomplicando Rest

Documentação SWAGGER - <http://localhost:8080/swagger-ui.html>



The screenshot shows the Swagger UI interface for the 'EducaCiencia API v1'. The top bar is green with the Swagger logo and a 'Select a spec' dropdown set to 'default'. The main content area has a title 'EducaCiencia API v1' with a version badge. Below the title, it shows the base URL '[Base URL: localhost:8080/]' and a link to 'http://localhost:8080/v2/api-docs'. A description states: 'Applications responsible to configure devices and expose current setup configuration.' and a copyright notice: 'Copyrights 2020 - EducaCiencia FastCode - Todos os Direitos Reservados'. Under the 'Schemes' section, 'HTTP' is selected. A list of controllers is displayed: 'cliente-controller' (Cliente Controller), 'metodos-https-controller' (Metodos Https Controller), 'primeiro-controller' (Primeiro Controller), and 'produto-controller' (Produto Controller). Each controller has a right-pointing arrow. At the bottom, there is a 'Models' section with a right-pointing arrow.

Fonte: Curso de Java Evolua Sumare



Java: Descomplicando Rest

Documentação SWAGGER - <http://localhost:8080/swagger-ui.html>

EducaCiencia API
[Base URL: localhost:8080/]
<http://localhost:8080/v2/api-docs>
Applications responsible to configure devices and expose current setup configuration.
[Copyrights 2020 - EducaCiencia FastCode - Todos os Direitos Reservados](#)

Schemes
HTTP

cliente-controller Cliente Controller
metodos-https-controller Metodos Https Controller
primeiro-controller Primeiro Controller
produto-controller Produto Controller

GET /api/produtos findAll
POST /api/produtos novoProduto
GET /api/produtos/{id} findById
PUT /api/produtos/{id} UpdateProduto
DELETE /api/produtos/{id} DeleteProduto

Models

produto-controller Produto Controller

GET /api/produtos findAll

Parameters
No parameters

Execute Clear

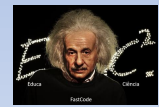
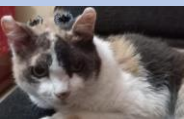
Responses
Response content type */*

Curl
curl -X GET "http://localhost:8080/api/produtos" -H "accept: */*"

Request URL
http://localhost:8080/api/produtos

Server response
Code Details
200
Response body
[
 {
 "id": 2,
 "nome": "teste",
 "preco": 600,
 "desconto": 0.4
 },
 {
 "id": 3,
 "nome": "Celular",
 "preco": 0,
 "desconto": 0
 },
 {
 "id": 4,
 "nome": "Celular",
 "preco": 0.15,
 "desconto": 0.15
 },
 {
 "id": 6,
 "nome": "Celular",
 "preco": 0.15,
 "desconto": 0.15
 }
]

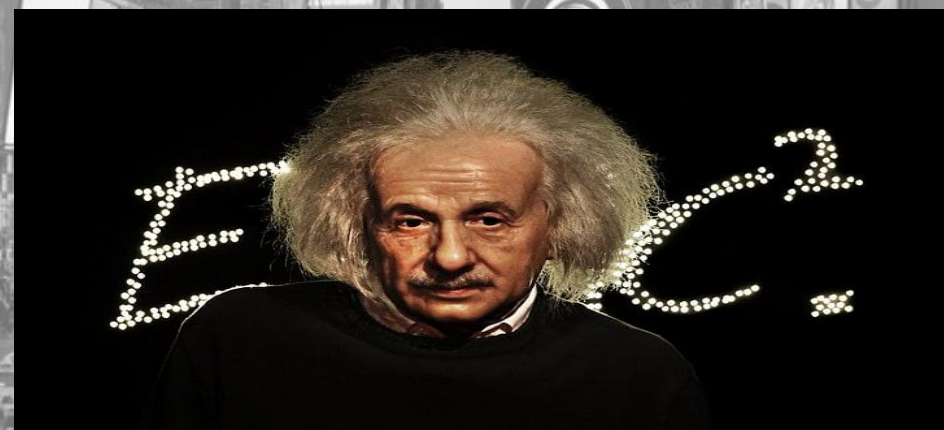
Fonte: Curso de Java Evolua Sumare



AGRADECIMENTOS aos Apoiadores!!!



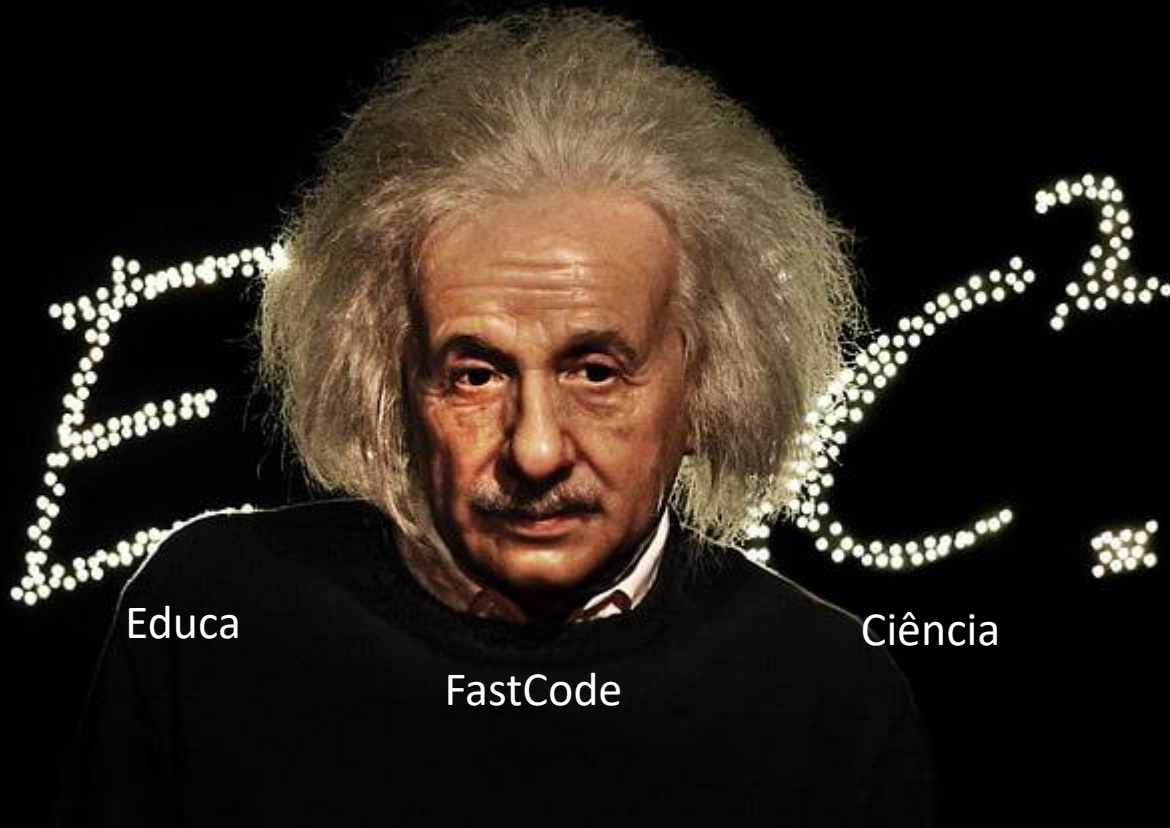
EXPLORANDO TI



SysConecte
Sistemas inteligentes para seu negócio

Fábio Perucello





Educa

FastCode

Ciência

EMAIL

EDUCACIENCIA-FASTCODE@OUTLOOK.COM.BR

EXPLORANDO TI

[HTTPS://WWW.EXPLORANDOTI.COM.BR/](https://www.explorandoti.com.br/)

[HTTPS://WWW.YOUTUBE.COM/EXPLORANDOTI](https://www.youtube.com/explorandoti)

GITHUB

[HTTPS://GITHUB.COM/PERUCELLO](https://github.com/perucello)

ARTIGOS

[HTTPS://GITHUB.COM/PERUCELLO/ARTIGOS-EDUCACIENCIA FASTCODE](https://github.com/perucello/artigos-educaciencia-fastcode)

LIVES

[HTTPS://WWW.YOUTUBE.COM/PLAYLIST?LIST=PLG_DRIR73C6OGRAJXXQD0XJ677LU_ZFGM](https://www.youtube.com/playlist?list=PLG_DRIR73C6OGRAJXXQD0XJ677LU_ZFGM)

Fábio Perucello

