Class 1 - How to Create a Spring Boot Project

- Enter in this web site to generate your Maven Project:

  https://start.spring.io/

- Put the type of Project (Maven), the language (Java) and Spring boot version (2.3.7)

Project

● Maven Project ○ Gradle Project

Language

● Java ○ Kotlin ○ Groovy

Spring Boot

○ 2.5.0 (SNAPSHOT) ○ 2.4.2 (SNAPSHOT) ○ 2.4.1 ○ 2.3.8 (SNAPSHOT)
● 2.3.7

Project Metadata

Group	com.restcrudbasics
Artifact	client
Name	client
Description	A basic CRUD REST using Spring Boot
Package name	com.restcrudbasics.client
Packaging	● Jar ○ War
Java	○ 15 ● 11 ○ 8

- Put the name of the project, the packaging (jar) and Java version (11 LTS)
- Add the dependencies of the Project. In this case, a simple Maven Project have Spring Web dependency

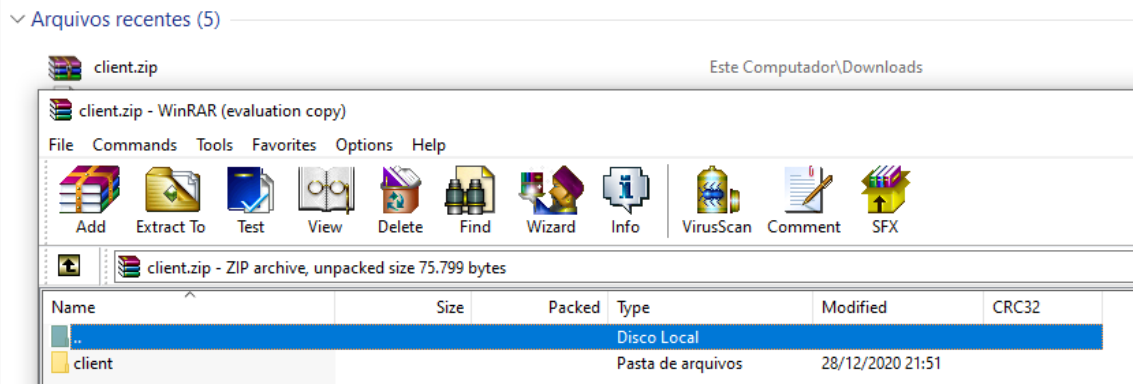
Dependencies ADD DEPENDENCIES... CTRL + B

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

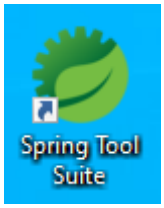
- Click in generate, to generate the file

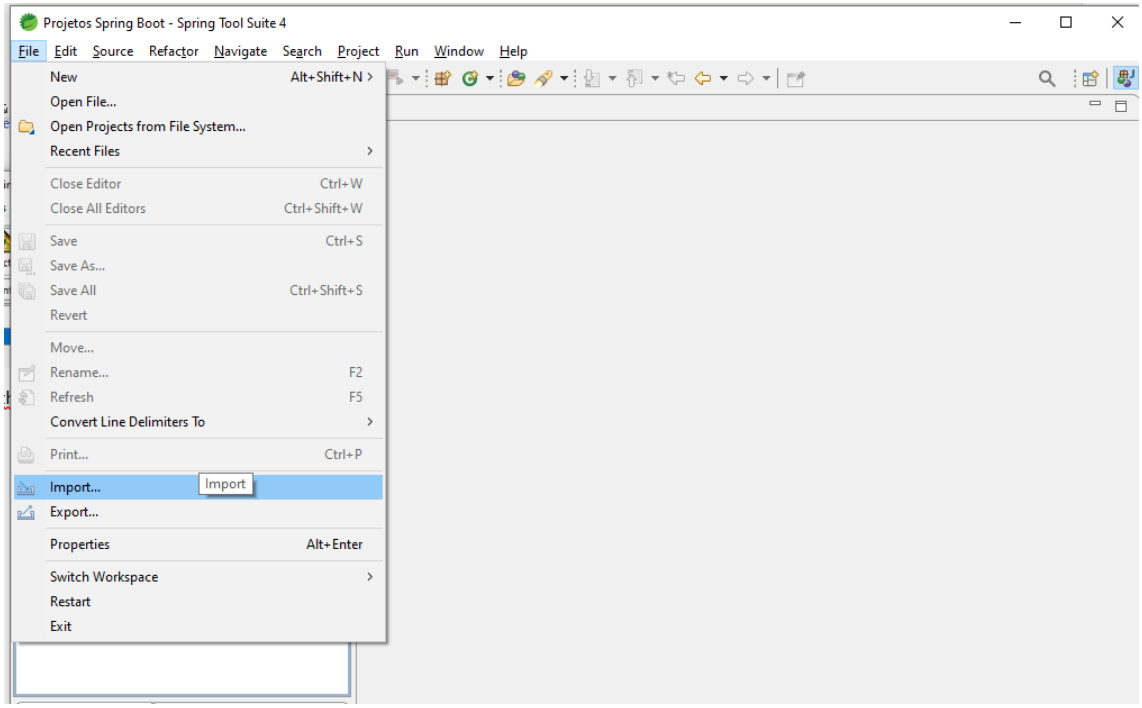
GENERATE CTRL + ⏎ | EXPLORE CTRL + SPACE | SHARE...

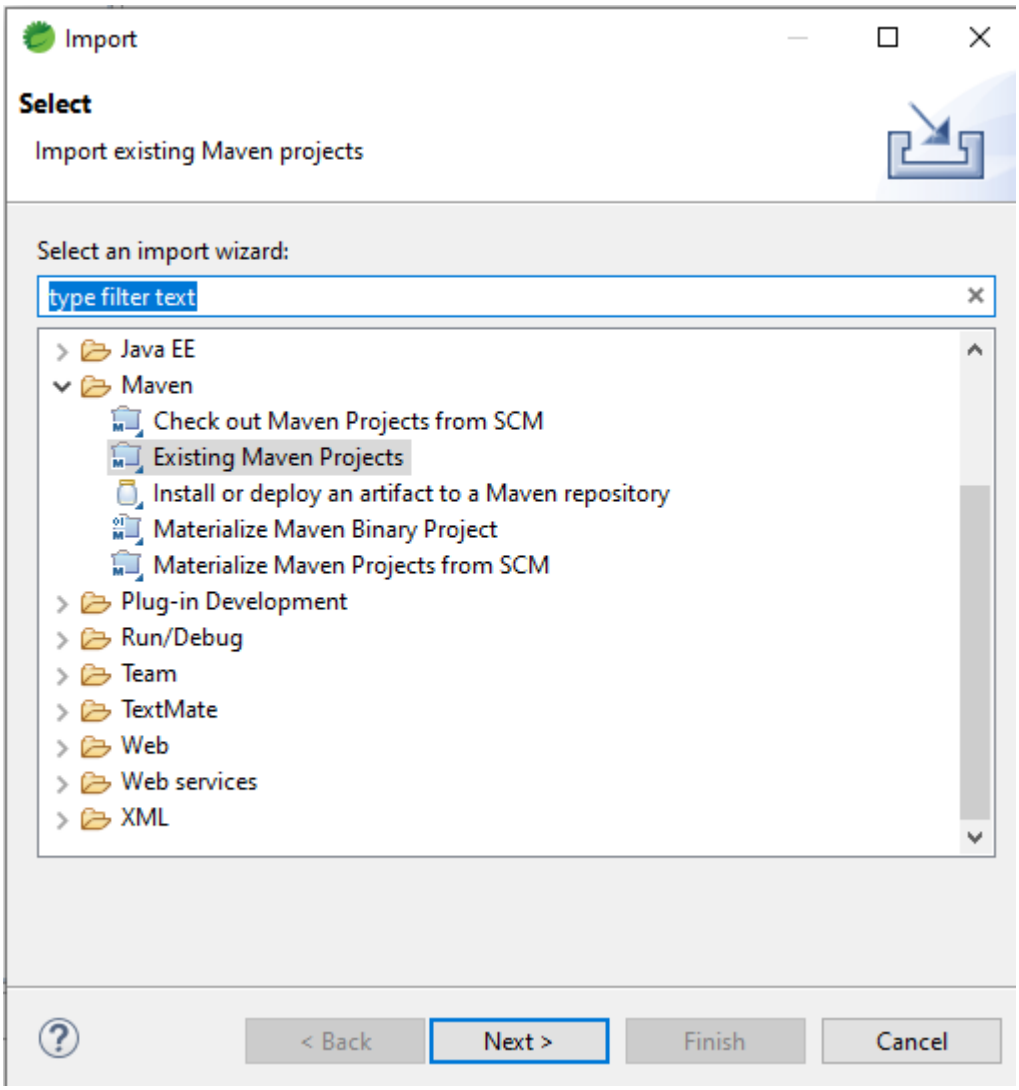
- Unzip the file and put in the desire workspace

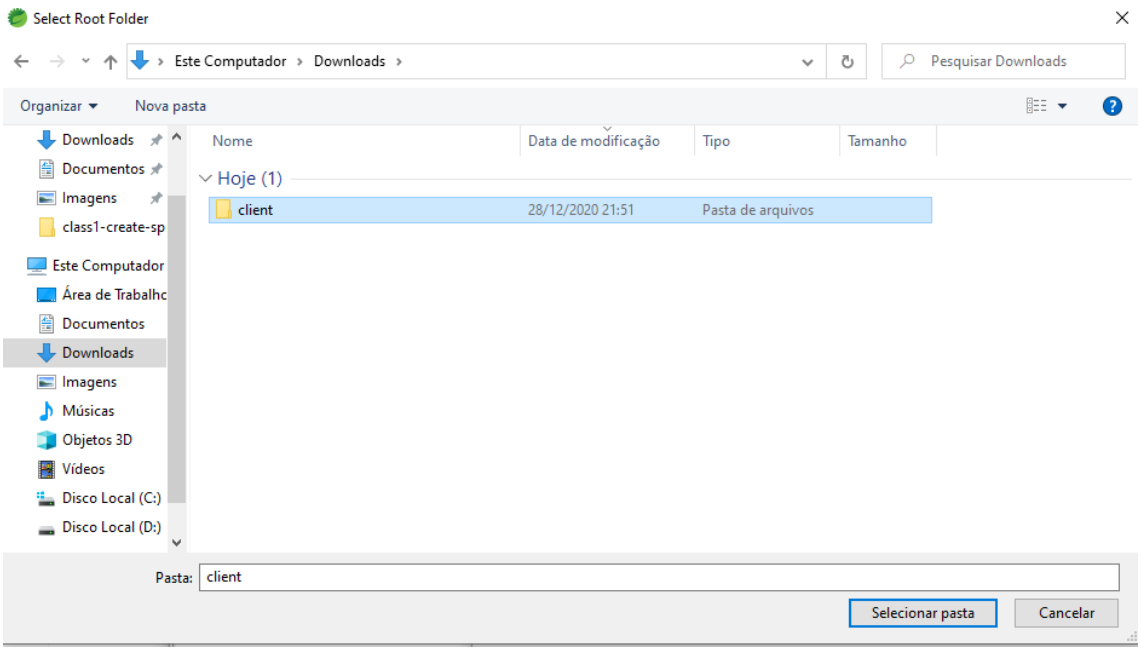


- Open the Spring tool Suite and open the folder as existing maven Project

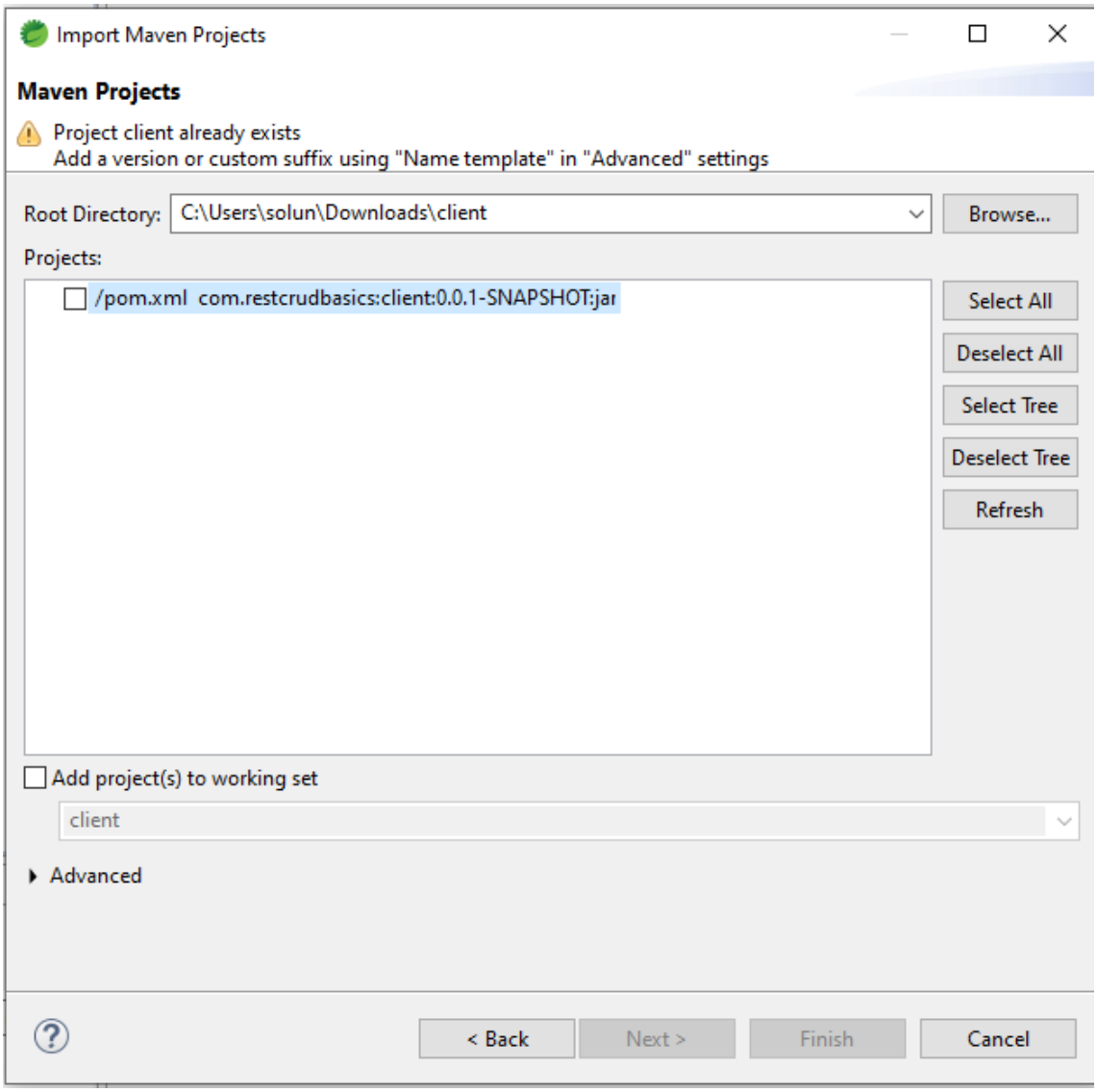




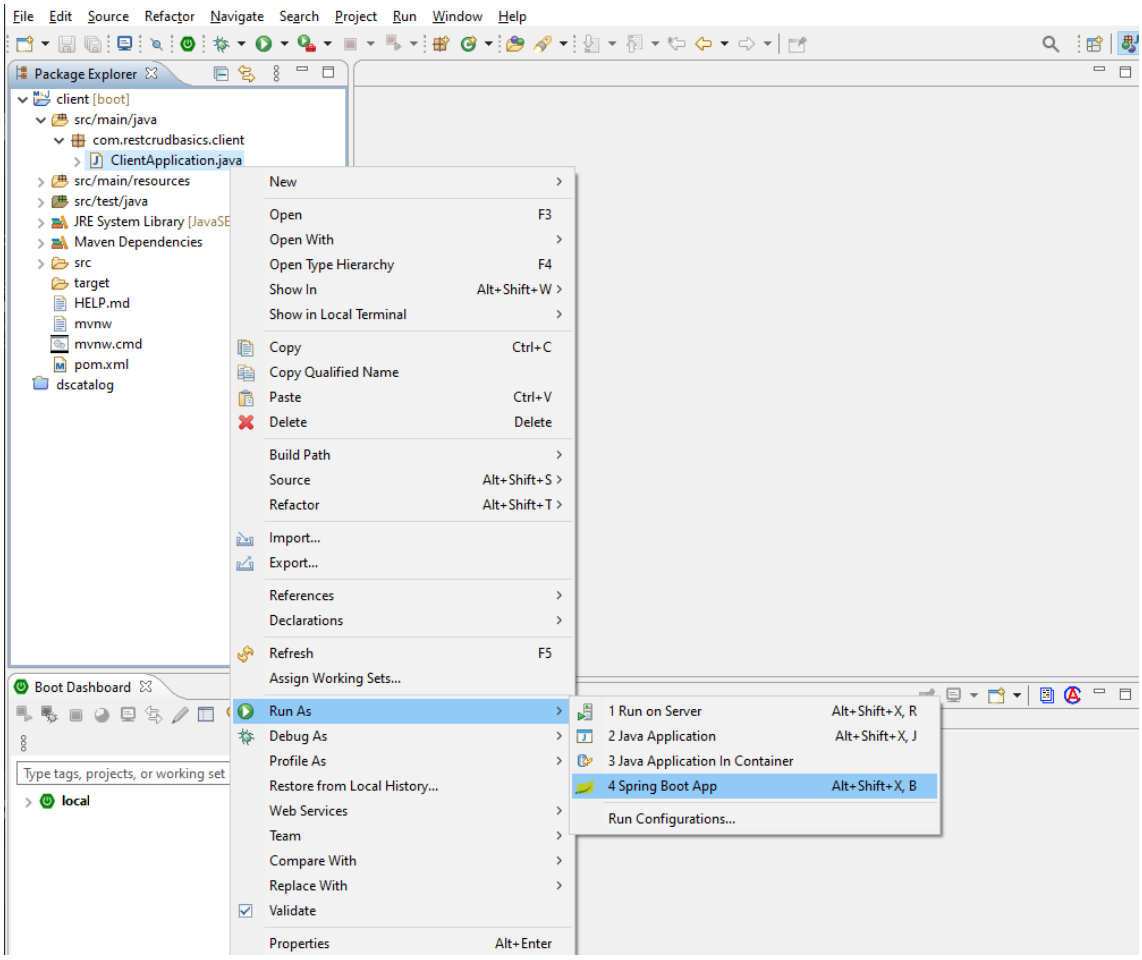

Import
Select
Import existing Maven projects
Select an import wizard:
type filter text
Java EE
Maven
Check out Maven Projects from SCM
Existing Maven Projects
Install or deploy an artifact to a Maven repository
Materialize Maven Binary Project
Materialize Maven Projects from SCM
Plug-in Development
Run/Debug
Team
TextMate
Web
Web services
XML
< Back
Next >
Finish
Cancel

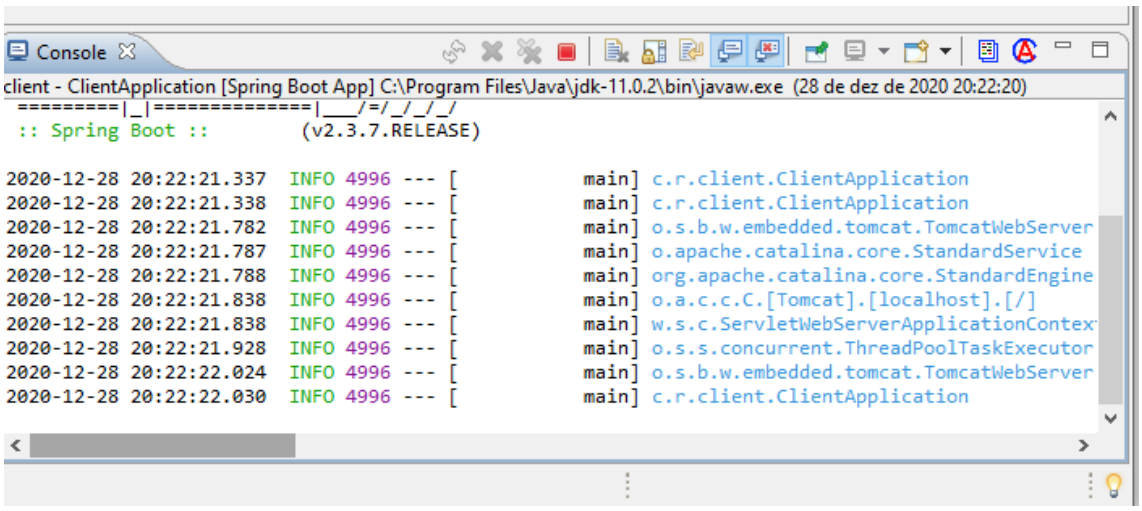

Select Root Folder
Este Computador › Downloads ›
Pesquisar Downloads
Organizar
Nova pasta
Downloads
Documentos
Imagens
class1-create-sp
Este Computador
Área de Trabalho
Documentos
Downloads
Imagens
Músicas
Objetos 3D
Vídeos
Disco Local (C:)
Disco Local (D:)
Nome
Data de modificação
Tipo
Tamanho
Hoje (1)
client
28/12/2020 21:51
Pasta de arquivos
Pasta:
client
Selecionar pasta
Cancelar

- The STS will detect the pom.xml file



- After STS download the dependencies, run as spring boot app

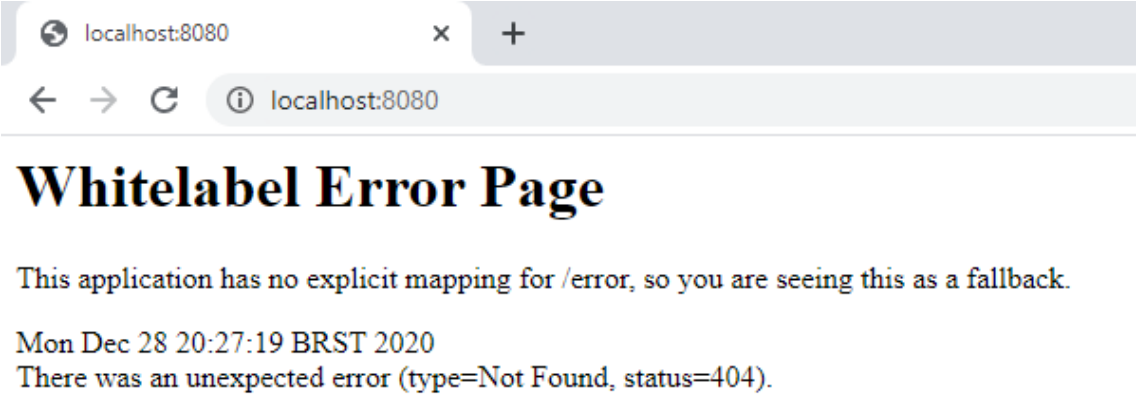




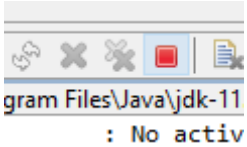
- The STS have a web container that name is Apache Tomcat. For standard, a STS application will run in the web port 8080

```
: Tomcat initialized with port(s): 8080 (http)
```

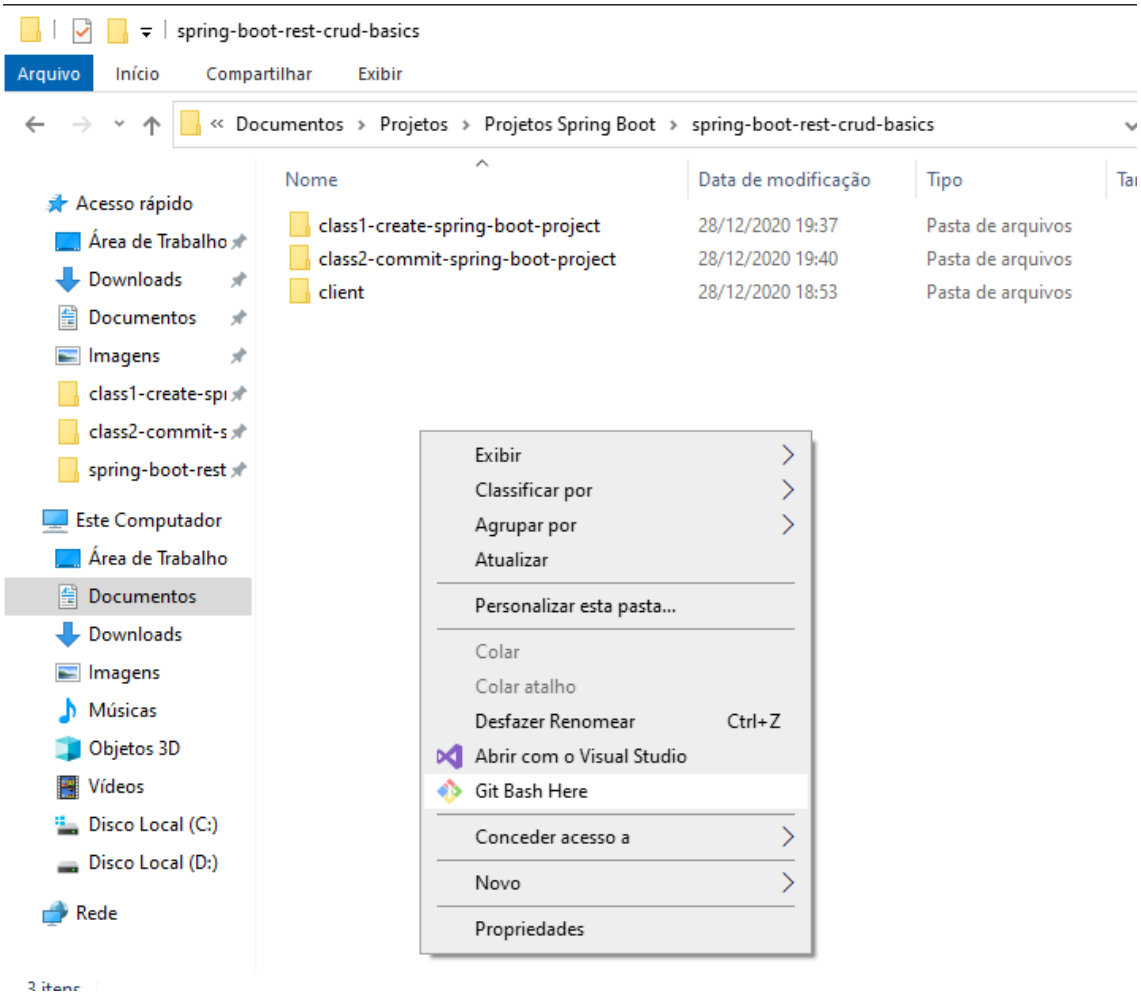
- The Apache Tomcat is the database were we gonna make the tests
- To test if everything is ok, go in your web search and put localhost:8080
- If everything is ok is gone appear the message below



- Stop the application and make a commit in github

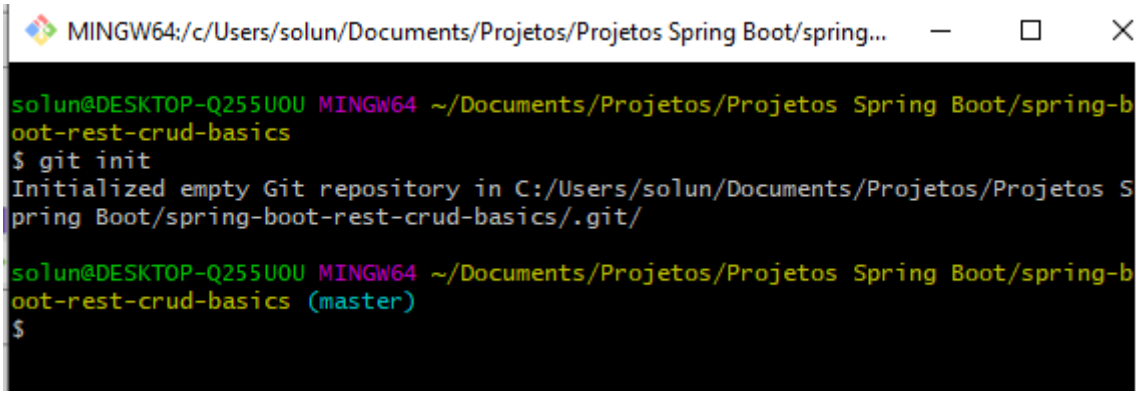


- Go in the main folder of the Project and put Git Bash Here



- Enter git init to create your repositor

$ git init



- If your user is not register in git in your computer, put the command below
- $ git config --global user.name “Your User Name GitHub”
- $ git config --global user.email “Your email GitHub”

```
[redacted] MINGW64 /c/[redacted] (master)
$ git config --global user.name "[redacted]ves"

[redacted] MINGW64 /c/[redacted] (master)
$ git config --global user.email "[redacted]@gmail.com"
```

- Enter in GitHub and create a new repositor

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Repository template

Start your repository with a template repository's contents.

No template

Owner * / **Repository name** *

vitorstabile / spring-boot-rest-crud-basics

Great repository names are spring-boot-rest-crud-basics is available. n? How about laughing-octo-guide?

Description (optional)

Public
Anyone on the internet can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

- [ ] **Add a README file**
This is where you can write a long description for your project. Learn more.

- [ ] **Add .gitignore**
Choose which files not to track from a list of templates. Learn more.

- [ ] **Choose a license**
A license tells others what they can and can't do with your code. Learn more.

Create repository

- Associate your repository with github

```
$ git remote add origin https://github.com/vitorstabile/spring-boot-rest-crud-basics.git
```

```
solun@DESKTOP-Q255U0U MINGW64 ~/Documents/Projetos/Projetos Spring Boot/spring-boot-rest-crud-basics
$ git init
Initialized empty Git repository in C:/Users/solun/Documents/Projetos/Projetos Spring Boot/spring-boot-rest-crud-basics/.git/

solun@DESKTOP-Q255U0U MINGW64 ~/Documents/Projetos/Projetos Spring Boot/spring-boot-rest-crud-basics (master)
$ git remote add origin https://github.com/vitorstabile/spring-boot-rest-crud-basics.git

solun@DESKTOP-Q255U0U MINGW64 ~/Documents/Projetos/Projetos Spring Boot/spring-boot-rest-crud-basics (master)
$ |
```

- Enter the command git status to see what you change

$ git status

```
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        class1-create-spring-boot-project/
        class2-commit-spring-boot-project/
        client/
```

- Add the changes to stage

$ git add .

```
$ git add .
warning: LF will be replaced by CRLF in backend/.gitignore.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in backend/.mvn/wrapper/MavenWrappe
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in backend/.mvn/wrapper/maven-wrapp
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in backend/mvnw.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in backend/mvnw.cmd.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in backend/pom.xml.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in backend/src/main/java/com/devsup
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in backend/src/main/resources/appli
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in backend/src/test/java/com/devsup
.
The file will have its original line endings in your working directory
```

- Enter git status to see what is in your stage

$ git status

```
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   backend/.gitignore
        new file:   backend/.mvn/wrapper/MavenWrapperDownloader.java
        new file:   backend/.mvn/wrapper/maven-wrapper.jar
        new file:   backend/.mvn/wrapper/maven-wrapper.properties
        new file:   backend/mvnw
        new file:   backend/mvnw.cmd
        new file:   backend/pom.xml
        new file:   backend/src/main/java/com/devsuperior/dscatalog/DscatalogApplication.java
        new file:   backend/src/main/resources/application.properties
        new file:   backend/src/test/java/com/devsuperior/dscatalog/DscatalogApplicationTests.java
```

- Enter the commit

$ git commit -m “Project created”

```
$ git commit -m "Project created"
```

- Enter the command to send to github

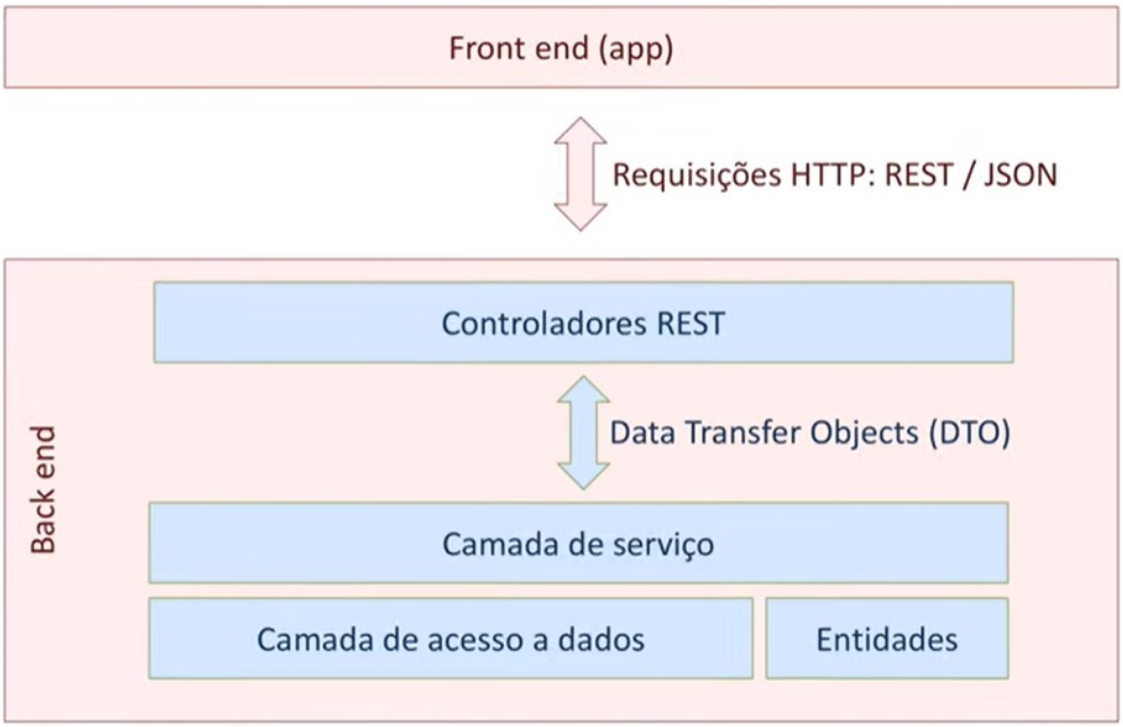
$ git push -u origin master

```
$ git push -u origin master
```

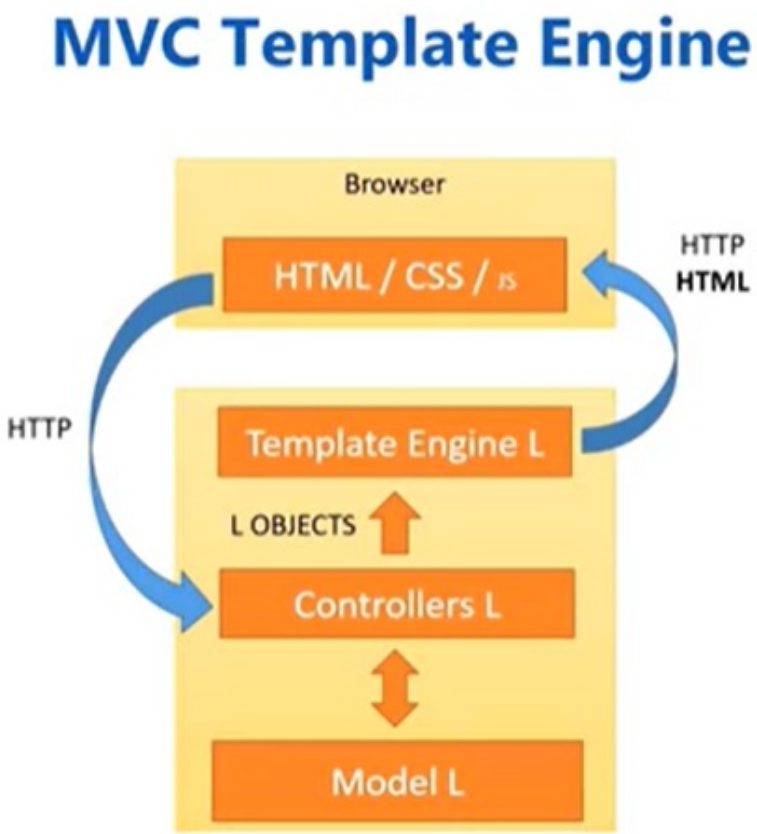
```
Enumerating objects: 27, done.
Counting objects: 100% (27/27), done.
Delta compression using up to 8 threads
Compressing objects: 100% (17/17), done.
Writing objects: 100% (27/27), 52.51 KiB | 8.75 MiB/s, done.
Total 27 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/acenelio/dscatalog-bootcamp-devsuperior.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

- See if the commit is in your github repositor

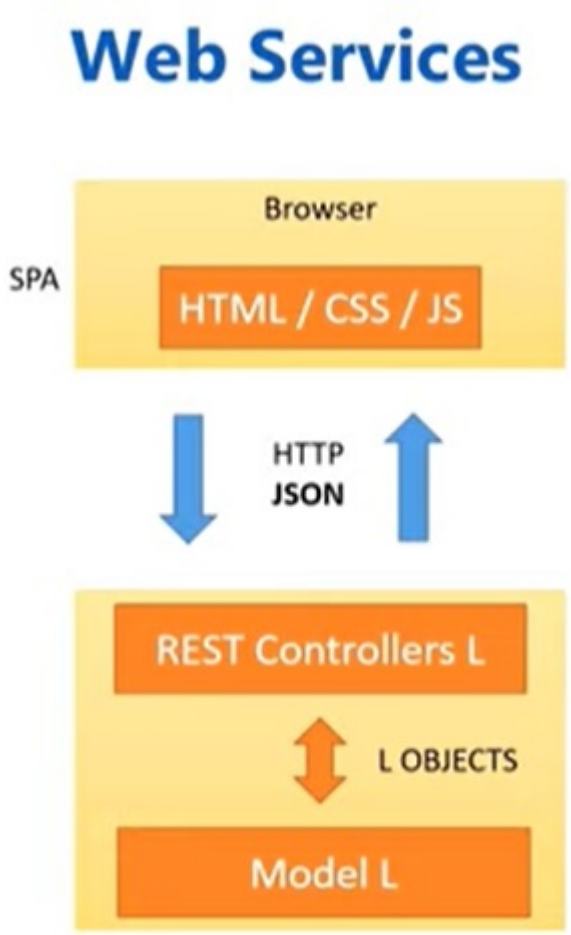
- Layered Architecture Example – REST (Representational State Transfer) Pattern



- MVC template engine example – The view of the system is built with your backend language with the Template Engine (Java, C# and PHP).



- Web Services – SPA (Single Page Application) is most used. The view of the system is built with HTML, CSS and JS, separately from the backend.
- REST – Use JSON, HTML and XML as data format.
- SOAP – Use only XML as data format.



- A route without REST controller

```
SEM REST:

https://seudominio.com/clientesSalvar [POST]

https://seudominio.com/clientesDeletar [POST]

https://seudominio.com/clientesBuscar?nome=Ana [GET]
```

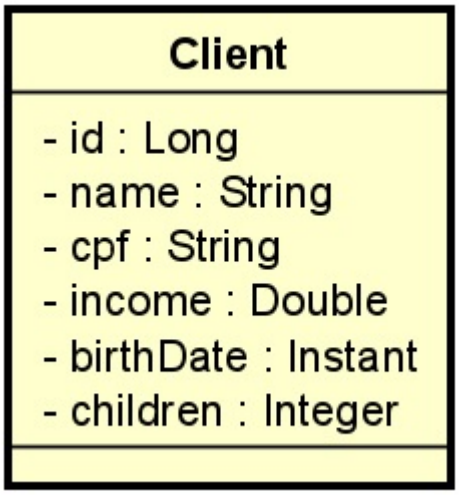
- A route with REST controller
- POST – INSERT a new object
- DELETE – Delete a object
- GET – Retrieve a object
- PUT – Update a object

```
COM REST:

https://seudominio.com/clientes [POST] (NÃO idempotente)
    { "name" : "Joao" }

https://seudominio.com/clientes/5 [DELETE] (idempotente)
https://seudominio.com/clientes/5 [GET] (idempotente)
https://seudominio.com/clientes/5 [PUT] (idempotente)
    { "name" : "Joao" }
```

- A Example of CRUD

```
CRUD - Create Retrieve (todos paginado / por id) Update Delete
```

/fl·ăă n π /2yÕŜLJúdzl·f a2ŘŜf ¦ a [



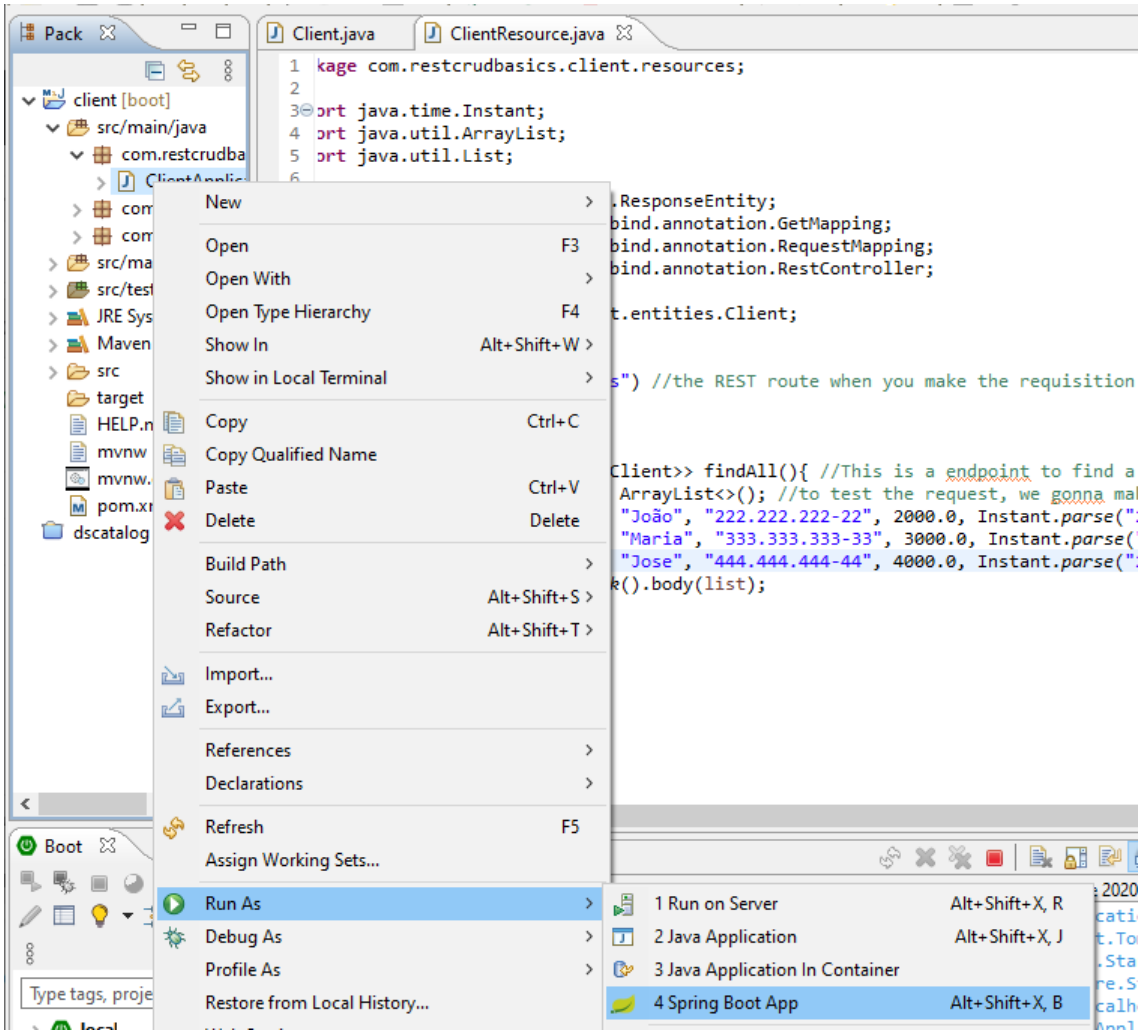
/ fl·ăă p π L Y LJfŜY Ŝyŭ 9yŭʌŭè

- Implement the basic entity
    - Basic Attributes
    - Associations (Instantiate collections)
    - Constructors
    - Getters & Setters (collections: only get)
    - hashCode & equals
    - Serializable

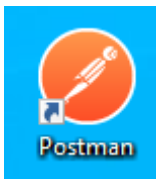
- After implemente the resource, you have to make the test

```
@GetMapping //endpoint GET
public ResponseEntity<List<Client>> findAll(){ //This is a endpoint to find a client by id
    List<Client> list = new ArrayList<>(); //to test the request, we gonna make a list
    list.add(new Client(1L, "João", "222.222.222-22", 2000.0, Instant.parse("2017-02-03T11:35:30.00Z"), 2));
    list.add(new Client(1L, "Maria", "333.333.333-33", 3000.0, Instant.parse("2017-02-03T11:35:30.00Z"), 3));
    list.add(new Client(1L, "Jose", "444.444.444-44", 4000.0, Instant.parse("2017-02-03T11:35:30.00Z"), 4));
    return ResponseEntity.ok().body(list);
}
```

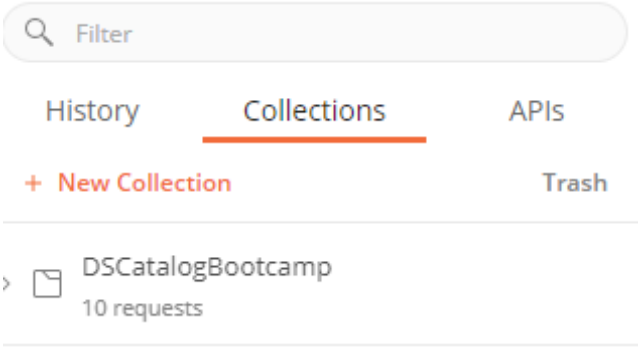
- To do this, run the spring boot app



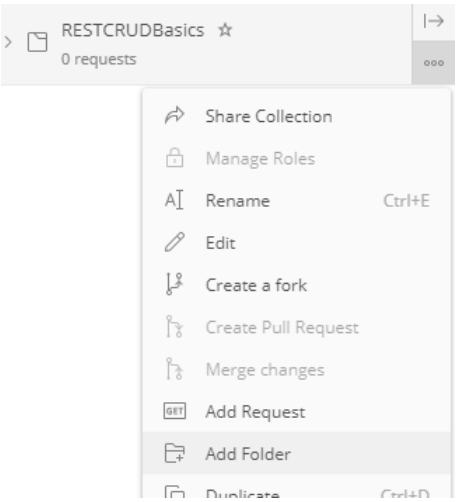
- Enter in the POSTMAN app

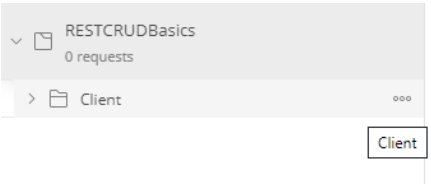


- Go in collection and make a new collection for your requisitions

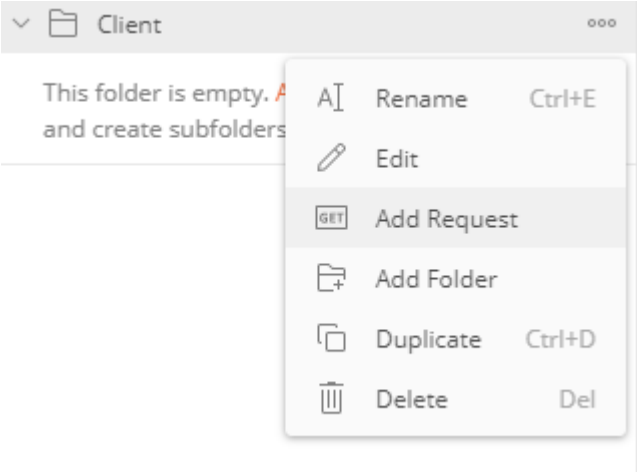


- Put a name and make a new folder with the entity name

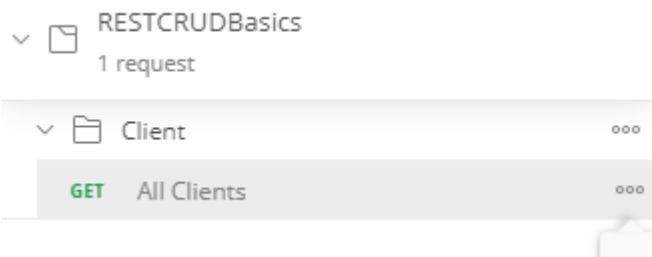




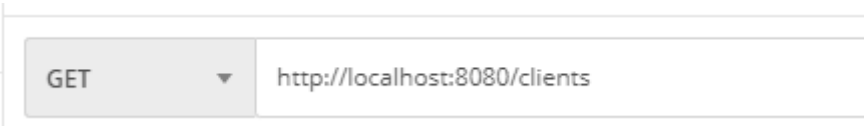
- Go in the folder and make a nem request



- Put a name in the request



- Now, make the request. In this case, to find all clientes



- If everything is ok, your request is shown in JSON format

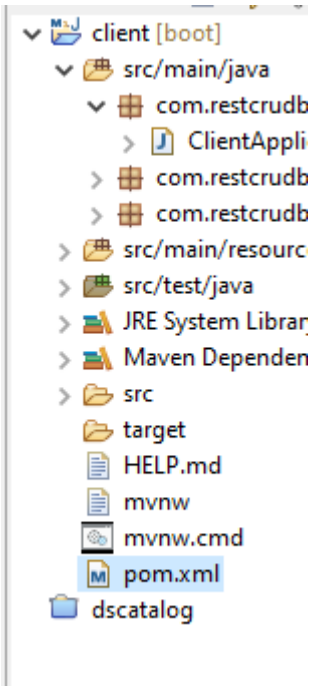

```
    "id": 1,
    "name": "João",
    "cpf": "222.222.222-22",
    "income": 2000.0,
    "birthDate": "2017-02-03T11:35:30Z",
    "children": 2
},
{
    "id": 1,
    "name": "Maria",
    "cpf": "333.333.333-33",
    "income": 3000.0,
    "birthDate": "2017-02-03T11:35:30Z",
    "children": 3
},
{
    "id": 1,
    "name": "Jose",
    "cpf": "444.444.444-44",
    "income": 4000.0,
    "birthDate": "2017-02-03T11:35:30Z",
    "children": 4
```

/fl·ăă τ π a l·ØŜy˝ 5ŜLJŜyŘŜyŎΛŜă óW t !Σ l HΣ t2ăűƎNJŜ{v[l·yŘ {LJNΛyƎ {ŜŎdzNΛűè

- In this maven dependencies below, you have the JPA,H2 database, PostgreSQL, springboot validation and spring boot security

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
</dependency>

<dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
</dependency>

<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-validation</artifactId>
</dependency>

<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

- Open the pom.xml file in STS



- Search for the dependencies

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

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
        <exclusions>
            <exclusion>
                <groupId>org.junit.vintage</groupId>
                <artifactId>junit-vintage-engine</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
</dependencies>
```

- Now, you have to put the dependencies between the <dependencies> statemant

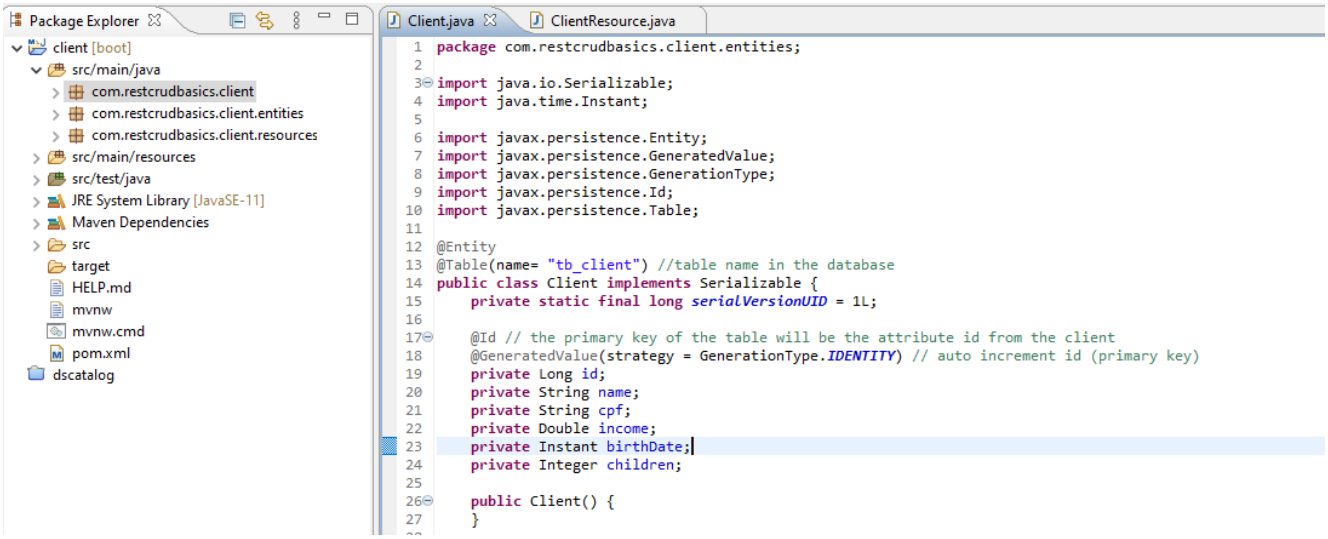
Client.java | ClientResource.java | *client/pom.xml

```
22     <dependencies>
23         <dependency>
24             <groupId>org.springframework.boot</groupId>
25             <artifactId>spring-boot-starter-web</artifactId>
26         </dependency>
27
28         <dependency>
29             <groupId>org.springframework.boot</groupId>
30             <artifactId>spring-boot-starter-data-jpa</artifactId>
31         </dependency>
32
33         <dependency>
34             <groupId>com.h2database</groupId>
35             <artifactId>h2</artifactId>
36             <scope>runtime</scope>
37         </dependency>
38
39         <dependency>
40             <groupId>org.postgresql</groupId>
41             <artifactId>postgresql</artifactId>
42             <scope>runtime</scope>
43         </dependency>
44
45         <dependency>
46             <groupId>org.springframework.boot</groupId>
47             <artifactId>spring-boot-starter-validation</artifactId>
48         </dependency>
49 <!--
50         <dependency>
51             <groupId>org.springframework.boot</groupId>
52             <artifactId>spring-boot-starter-security</artifactId>
53         </dependency>
54 -->
55         <dependency>
56             <groupId>org.springframework.boot</groupId>
57             <artifactId>spring-boot-starter-test</artifactId>
58             <scope>test</scope>
```

- When you save, the STS will make the download of the dependencies.

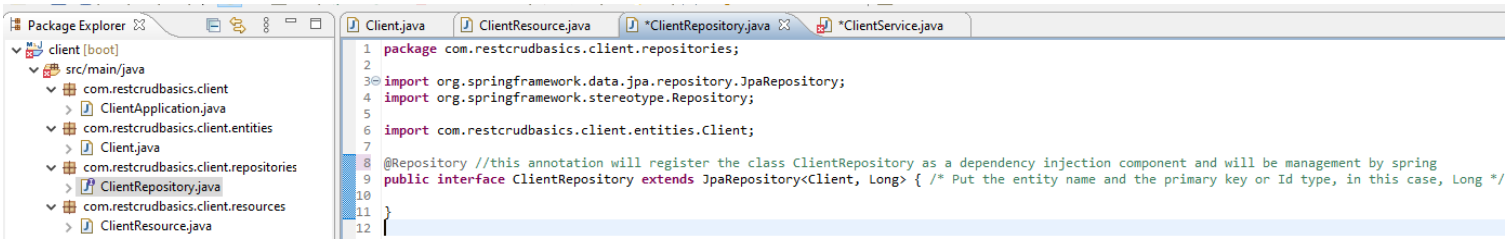
/fl-ăă y π LYLJtŜY Ŝyŭ hw a

- Basic ORM (Object Relational Mapping) with JPA in spring boot



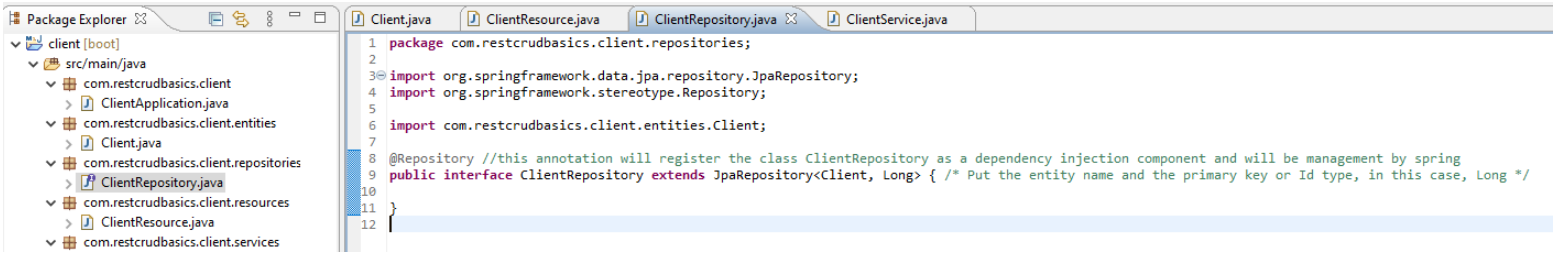
/ fl-ăă ф π LYLJtŜY Ŝyŭ wŜLJ2ăλŭ2Nè [l-èŜN

- Basic Repository implementation

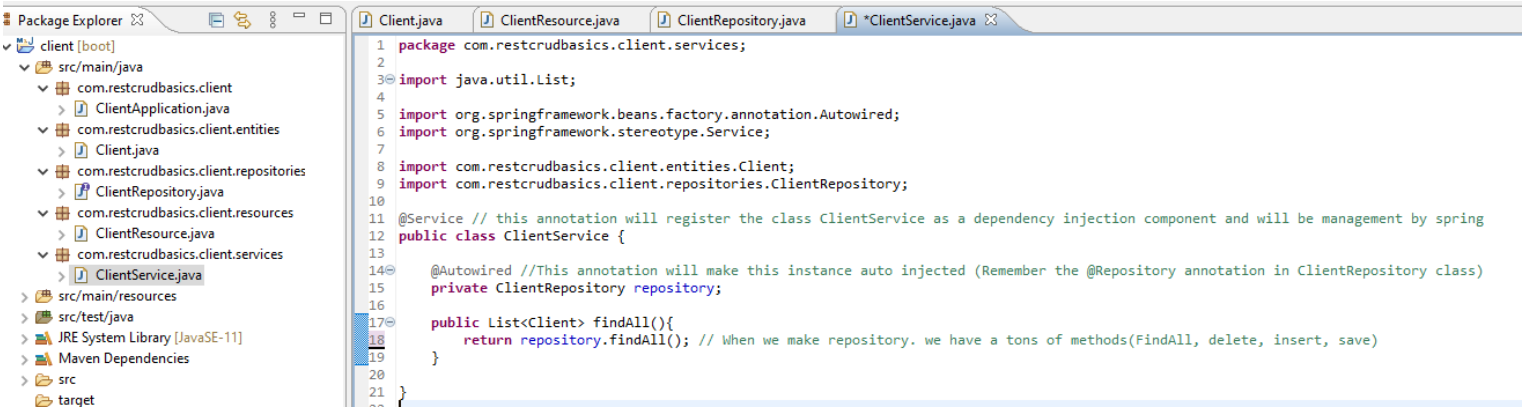


/fl·ăă мл π LYLJfŜYŜyŭ {ŜNØΛŎŜ [l·èŜN

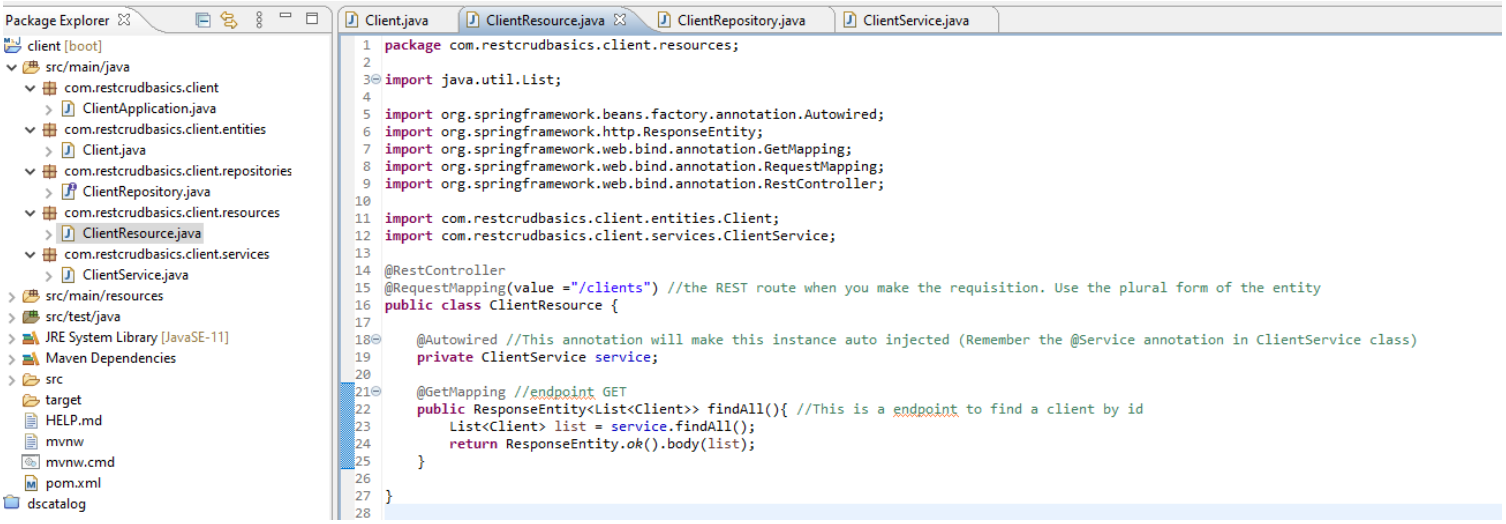
- First, make the dependency injection in the ClientRepository class with the specific annotation @Repository



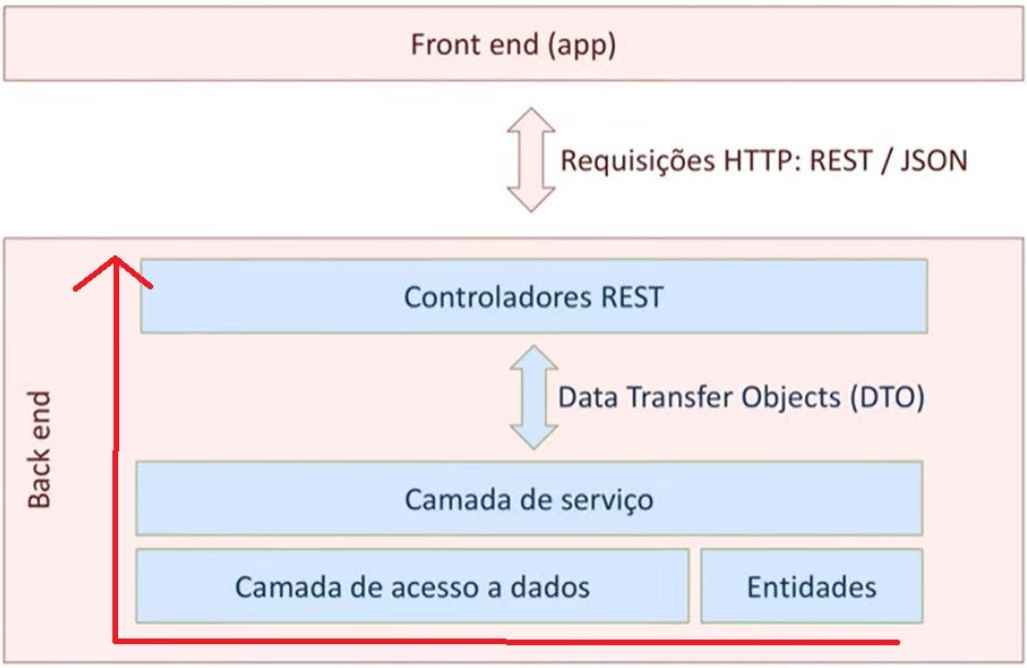
- Now, the ClientRepository class is a dependency injection componente management by spring
- Now, implemente the service Layer



- Now, make the depency injection in the ClientResource



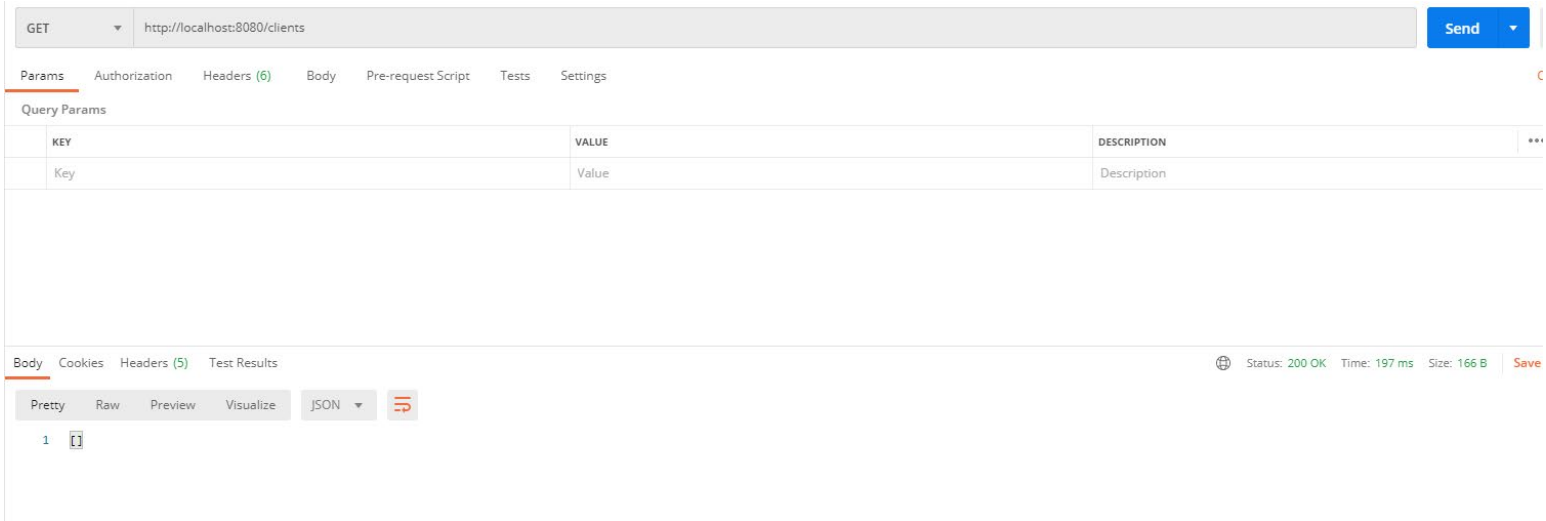
- With this, we are following the architecture layer above



- Now, is time for the REST requisition test

/fl·ăă M1 π Requisition Test in Postman

- Now, run the spring boot app, and make the request /clients in POSTAN



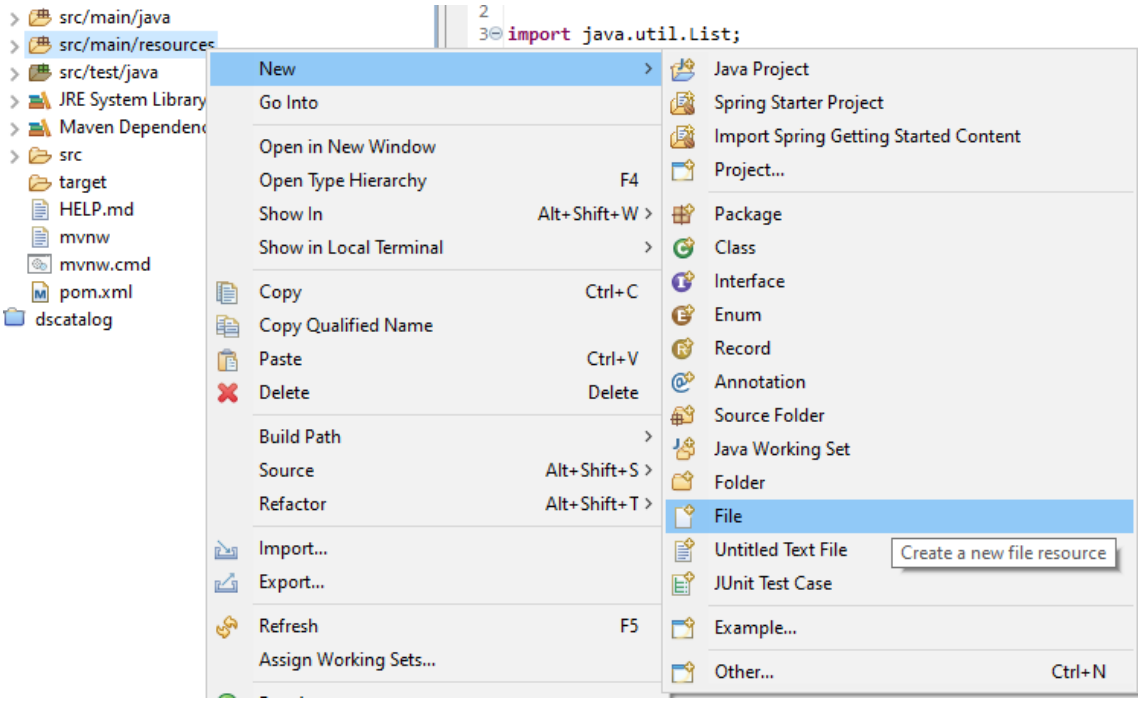
- When we look to the body of the return of the method findAll for the ResponseEntity, we see that is to return a List

```
@GetMapping //endpoint GET
public ResponseEntity<List<Client>> findAll(){ //This is a endpoint to find a client by id
    List<Client> list = service.findAll();
    return ResponseEntity.ok().body(list);
}
```

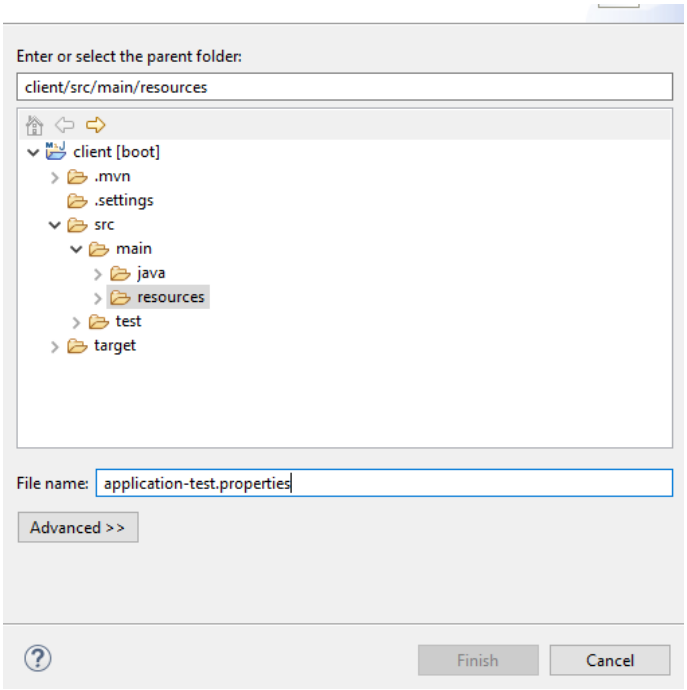
- In the Request, we see that the Json return a empty List

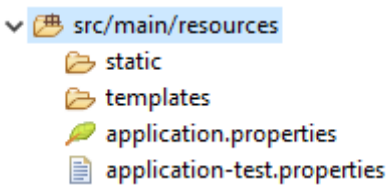
/fl·ăă MH π /2yƒƛƎdzℕŜ I H 5l·ŭl·ōl·ăŜ ól·LJLJƭƛŎl·ŭƛ2yϕLJℕ2LJŜℕŭƛŜăú

- Now, we gonna make the H2 database configuration properties
- Go in Resources and make a new file



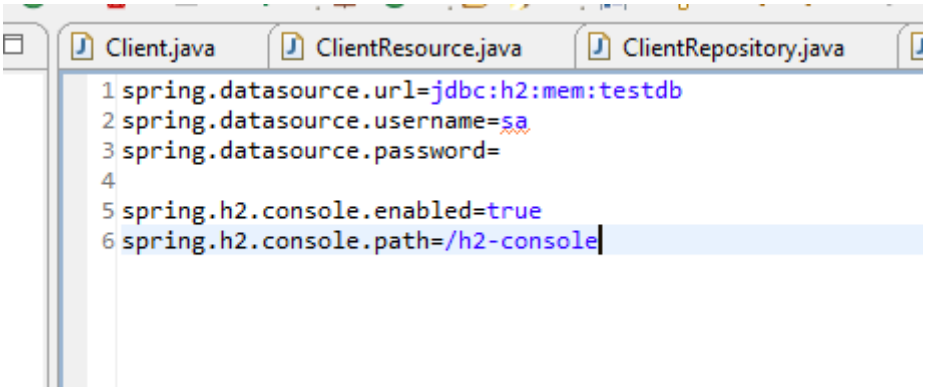
- Make a file with the name application-test.properties



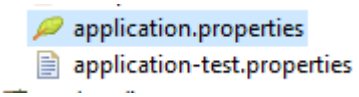


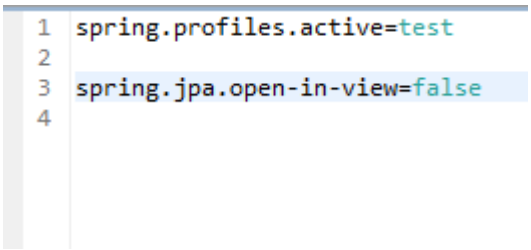
- Now put the H2 database configuration

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=

spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```



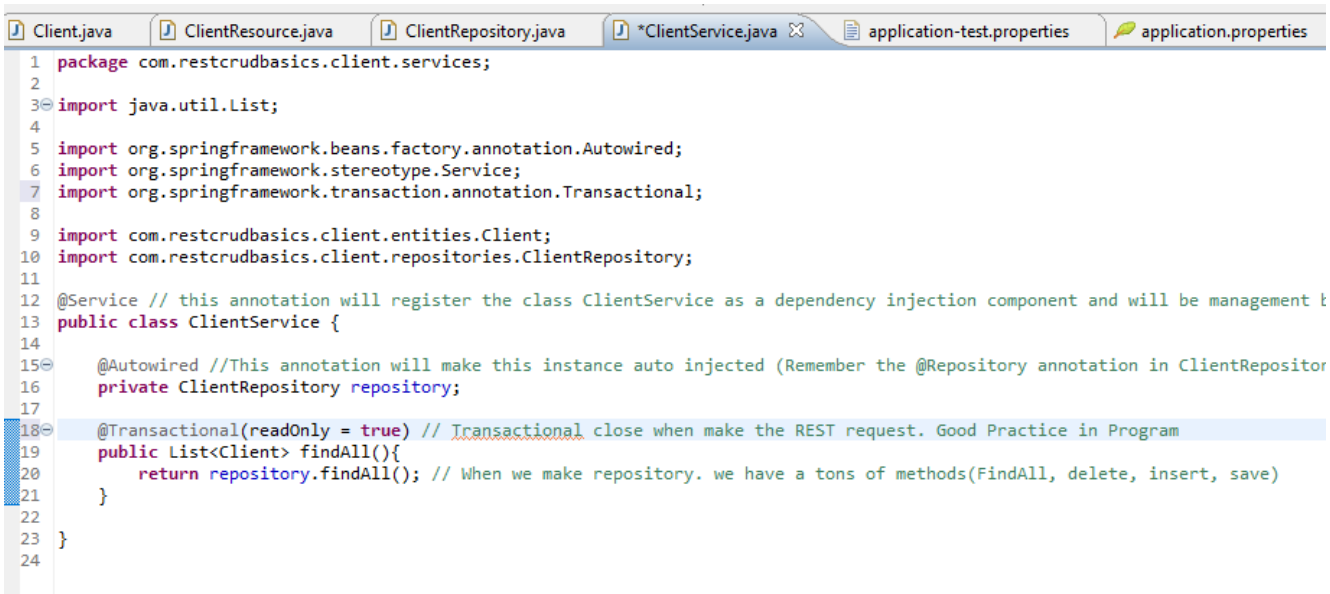
- Now, go to the file application.properties and put the configuration below





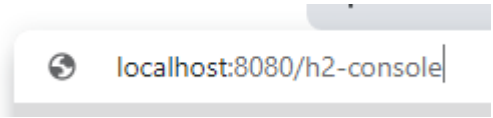

```
spring.profiles.active=test

spring.jpa.open-in-view=false
```

- This is a profile test. We gonna use the H2 database as database to make test's. Now, enter the database H2. In a application, we use 3 profiles - Test, Development and Production.
  Test - Normally, use H2 as database.
  Development - The same database used in Production or client. Normally, PostgreSQL
  Production - The final test.
- The Propert view = false is to close the transactions (Service, Repository) to not be open when make a REST request.
- To make this, use the annotation @Transactional in the service method

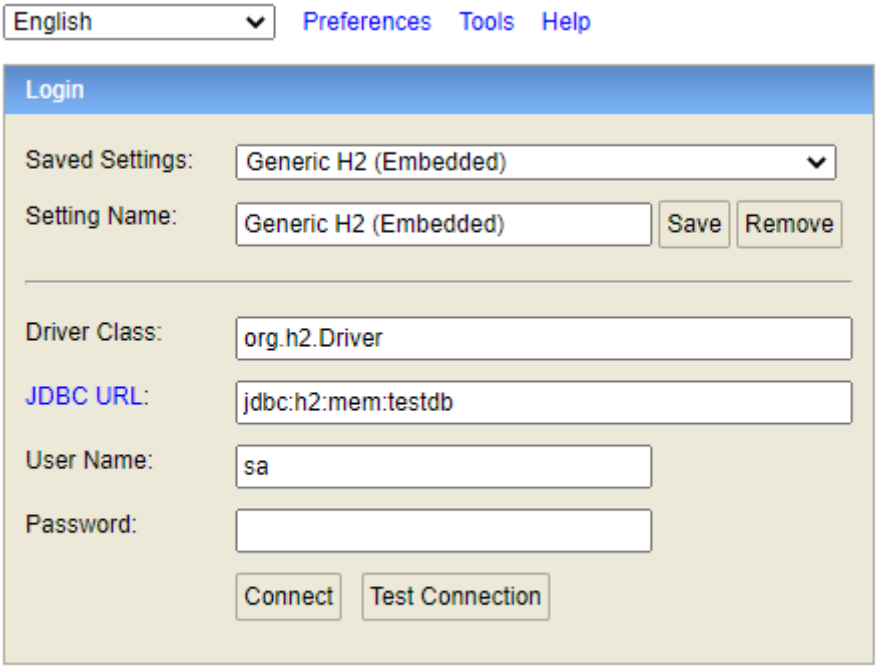


/fl·ăă M3 π Enter H2 Console

- Now, to enter in the H2, run the spring app and acess the H2



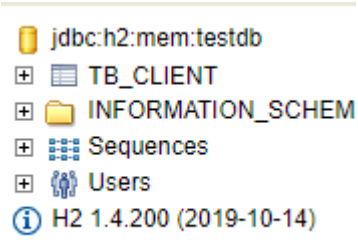
- Is gonna appear the H2 console login



- This match all configuration test that you made

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=

spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

- Go in connect and you will see the created table



- If you enter, you gonna make a select and see the colums that you created

Run | Run Selected | Auto complete | Clear | SQL statement:

```
SELECT * FROM TB_CLIENT
```

SELECT * FROM TB_CLIENT;

ID	BIRTH_DATE	CHILDREN	CPF	INCOME	NAME

(no rows, 3 ms)

Edit

- The database exist. You can make test's like insert a argument and make a request in POSTMAN

Run | Run Selected | Auto complete | Clear | SQL statement:

```
INSERT INTO TB_CLIENT (NAME) VALUES ('JOÃO');
```

INSERT INTO TB_CLIENT (NAME) VALUES ('JOÃO');
Update count: 1
(1 ms)

GET http://localhost:8080/clients

Params Authorization Headers (6) Body

Query Params

	KEY
	Key

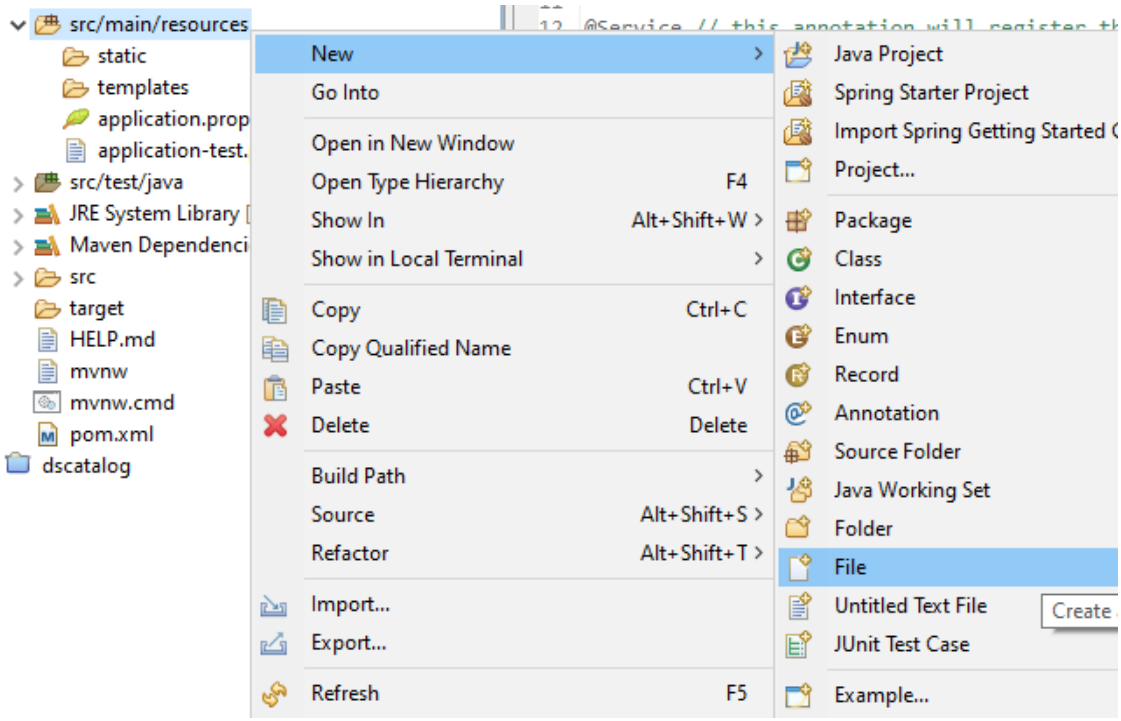
Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

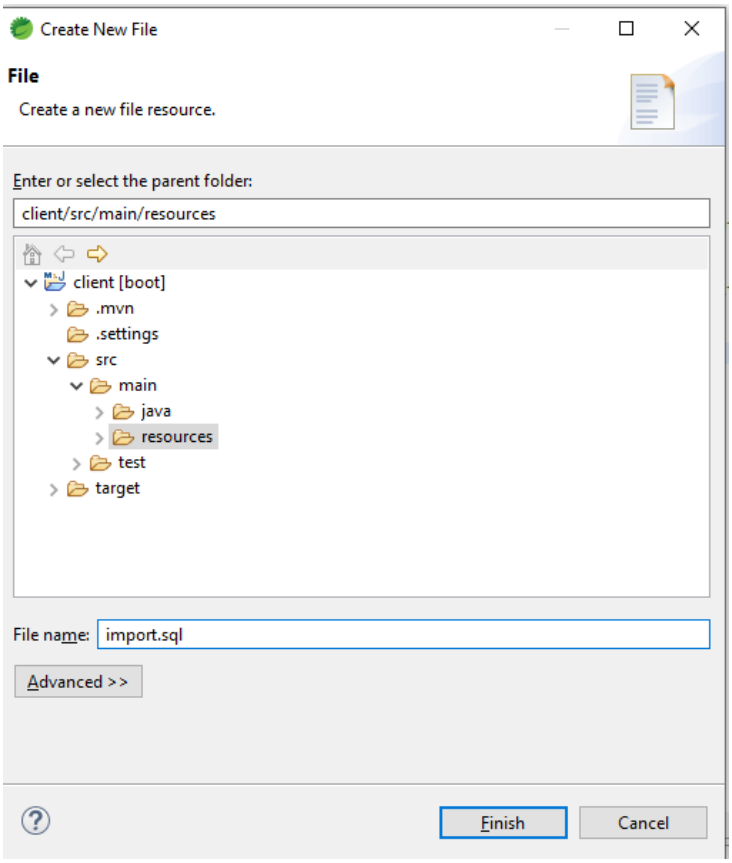

```
[
    {
        "id": 1,
        "name": "JOÃO",
        "cpf": null,
        "income": null,
        "birthDate": null,
        "chldren": null
    }
]
```

/fl·ăă M4 π Seeding H2 Database (import.sql)

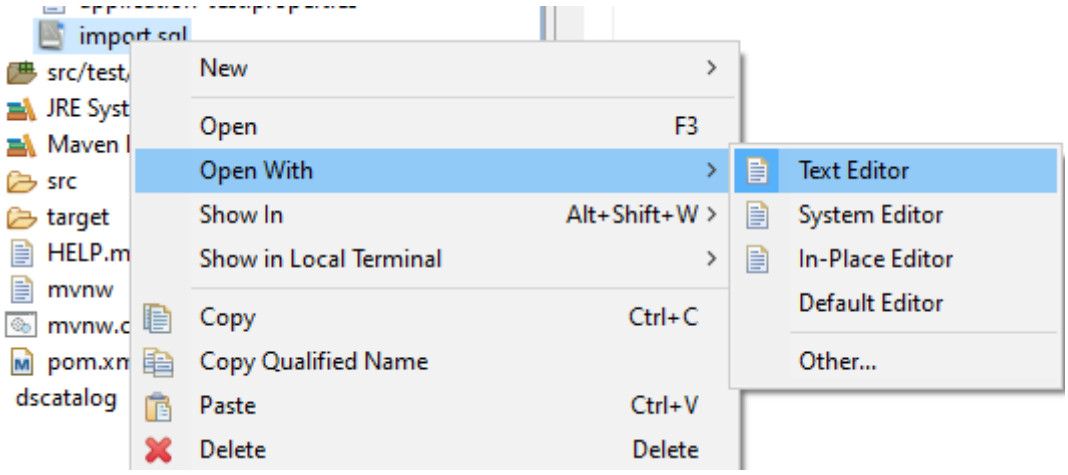
- Now, we gonna make a seeding of the database H2
- Go in Resources and make a new file



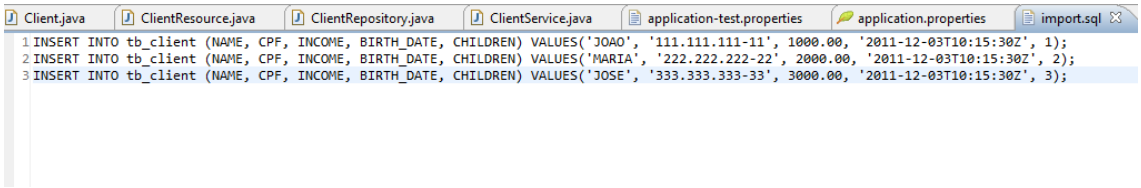
- Make a file import.sql



- Open with text editor STS



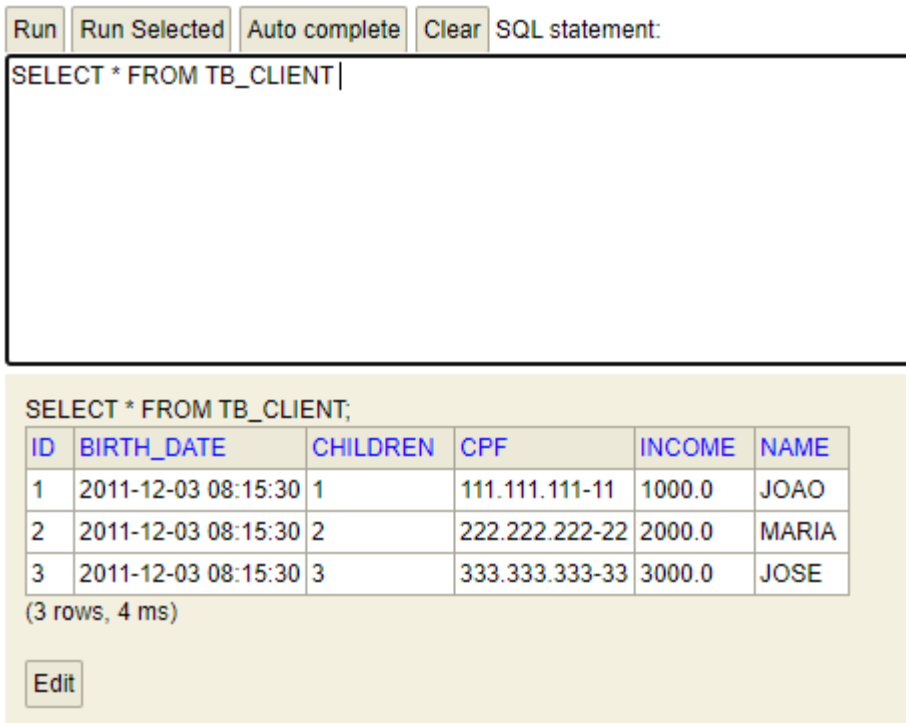
- Now, make the SQL to insert the seeding



- The columns name have to be the same name of the columns in H2

ID	BIRTH_DATE	CHILDREN	CPF	INCOME	NAME

- Run a Select to test



- Now you can test in POSTMAN

All Clients

GET http://localhost:8080/clients

Params Authorization Headers (6) Body Pre-reque

Query Params

KEY
Key

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON


```
2   {
3       "id": 1,
4       "name": "JOAO",
5       "cpf": "111.111.111-11",
6       "income": 1000.0,
7       "birthDate": "2011-12-03T10:15:30Z",
8       "chldren": 1
9   },
10  {
11      "id": 2,
12      "name": "MARIA",
13      "cpf": "222.222.222-22",
14      "income": 2000.0,
15      "birthDate": "2011-12-03T10:15:30Z",
16      "chldren": 2
17  },
18  {
19      "id": 3,
20      "name": "JOSE",
21      "cpf": "333.333.333-33",
22      "income": 3000.0,
23      "birthDate": "2011-12-03T10:15:30Z",
```

/fl-ăă mp π LYLJfŜYŜyŭ 5¢h ó5l-ŭl- ¢₦l-yãŦŜ₦ hōʔŜŎŭŭ

- Implement the basic atributes to DTO
  - Basic Attributes
  - Associations (Instantiate collections)
  - Constructors
  - Getters & Setters (collections: only get)
  - Serializable

nt.java | ClientResource.java | ClientRepository.java | ClientService.java | application-test.properties | application.properties | import.sql | *ClientDTO.java

```java
package com.restcrudbasics.client.dto;

import java.time.Instant;

public class ClientDTO {

    private Long id;
    private String name;
    private String cpf;
    private Double income;
    private Instant birthDate;
    private Integer children;

    public ClientDTO() {
    }

    public ClientDTO(Long id, String name, String cpf, Double income, Instant birthDate, Integer children) {
        this.id = id;
        this.name = name;
        this.cpf = cpf;
        this.income = income;
        this.birthDate = birthDate;
        this.children = children;
    }
```

```java
public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getCpf() {
    return cpf;
}

public void setCpf(String cpf) {
    this.cpf = cpf;
}

public Double getIncome() {
    return income;
}

public void setIncome(Double income) {
    this.income = income;
}

public Instant getBirthDate() {
    return birthDate;
}
```

- Now, make a constructor that receive the entity Client

```java
public ClientDTO(Client client) {
    this.id = client.getId();
    this.name = client.getName();
    this.cpf = client.getCpf();
    this.income = client.getIncome();
    this.birthDate = client.getBirthDate();
    this.children = client.getChildren();
}
```

- Now, change the Service
- The repository, work just with Entity, not DTO. You have to modif the code to convert Entity to DTO

```java
@Transactional(readOnly = true)
public List<CategoryDTO> findAll() {
    List<Category> list = repository.findAll();

    List<CategoryDTO> listDto = new ArrayList<>();
    for (Category cat : list) {
        listDto.add(new CategoryDTO(cat));
    }

    return listDto;
}
```

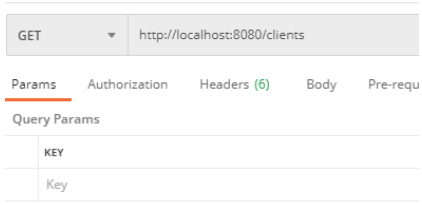
- We can use a lambda expression.

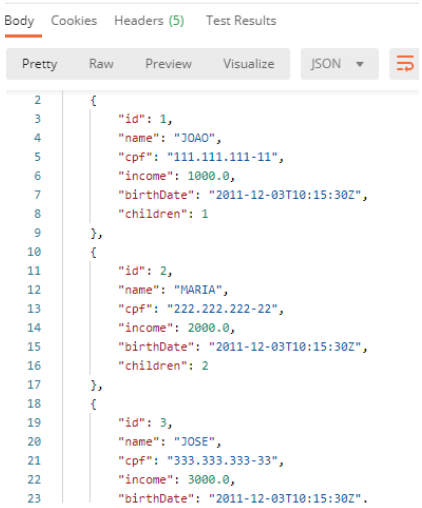
```java
package com.restcrudbasics.client.services;

import java.util.List;
import java.util.stream.Collectors;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.restcrudbasics.client.dto.ClientDTO;
import com.restcrudbasics.client.repositories.ClientRepository;
import com.restcrudbasics.client.entities.Client;

@Service // this annotation will register the class ClientService as a dependency injection component and will be management by spring
public class ClientService {

    @Autowired //This annotation will make this instance auto injected (Remember the @Repository annotation in ClientRepository class)
    private ClientRepository repository;

    @Transactional(readOnly = true) // Transactional close when make the REST request. Good Practice in Program
    public List<ClientDTO> findAll(){
        List<Client> list = repository.findAll(); // We have to convert this Client list to ClientDTO list.
        return list.stream().map(x -> new ClientDTO(x)).collect(Collectors.toList()); // Use Functional program
    }

}
```

- Now, change the resource

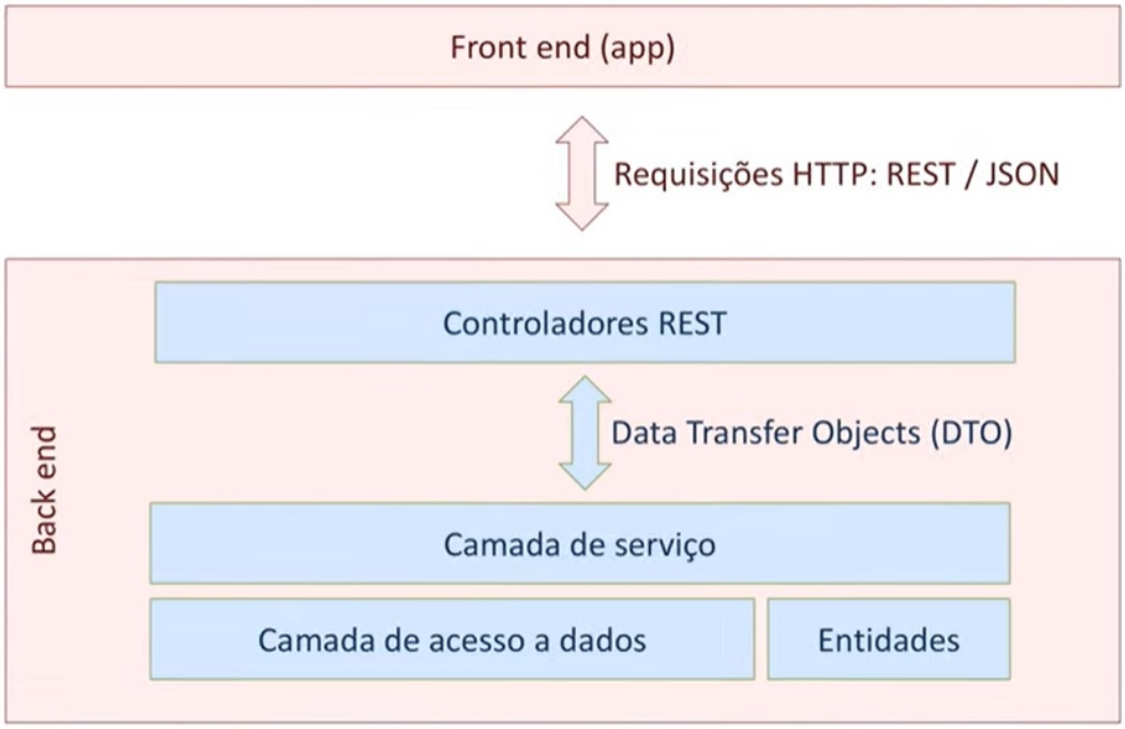
```java
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.restcrudbasics.client.dto.ClientDTO;
import com.restcrudbasics.client.services.ClientService;

@RestController
@RequestMapping(value ="/clients") //the REST route when you make the requisition. Use the plural form of the entity
public class ClientResource {

    @Autowired //This annotation will make this instance auto injected (Remember the @Service annotation in ClientService class)
    private ClientService service;

    @GetMapping //endpoint GET
    public ResponseEntity<List<ClientDTO>> findAll(){ //This is a endpoint to find a client by id
        List<ClientDTO> list = service.findAll();
        return ResponseEntity.ok().body(list);
    }

}
```

- Make a test



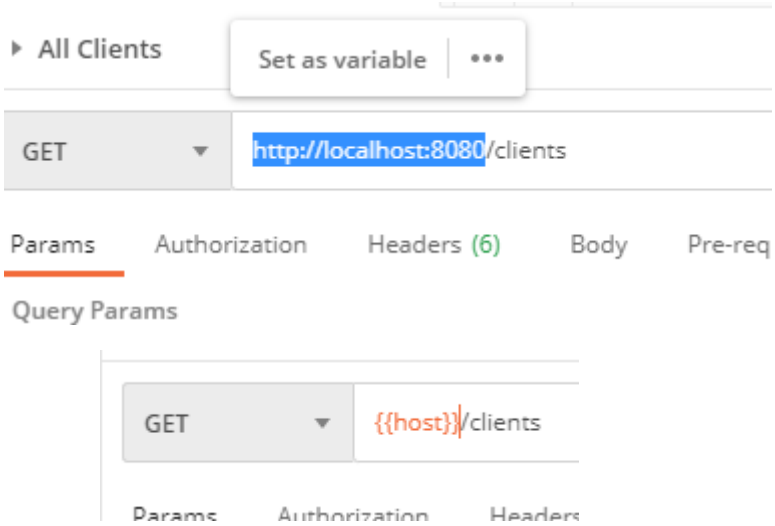


- Now, we implement all the web service architecture

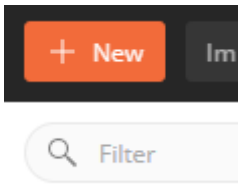


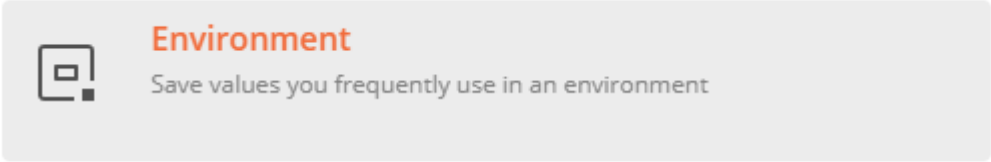
/fl·ăă м6 π Configure Environment in Postman

- Enter in postman and change the address http://localhost:8080/clients to {{host}}/clients

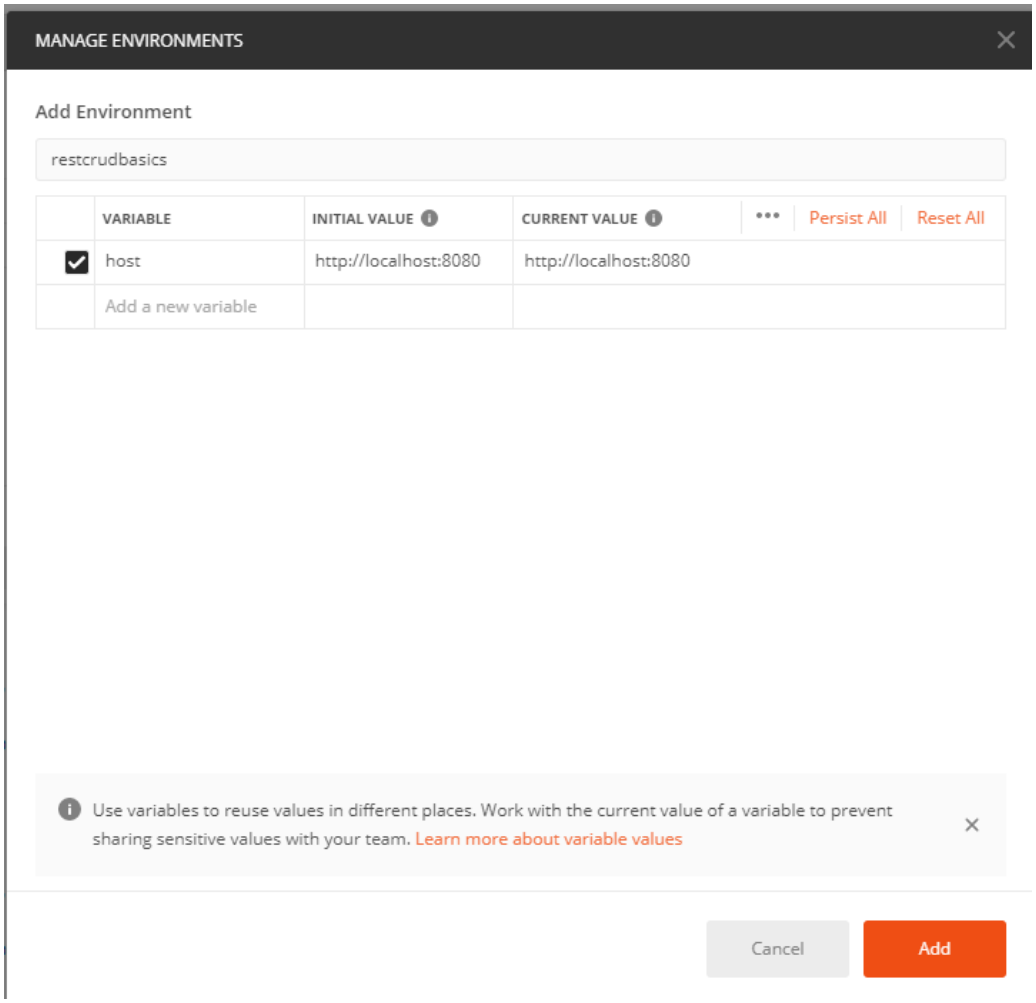


- Go in new and create a new environment

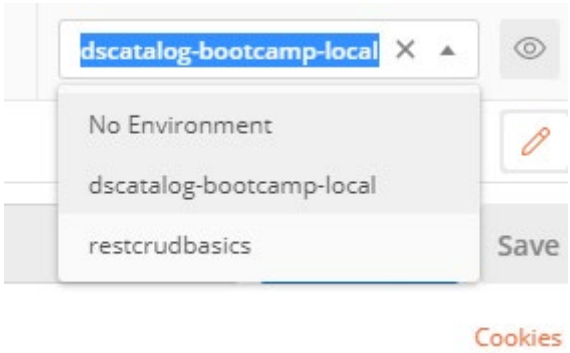




- Put a name and in the variable, put host and the address



- Now, change the environment



/fl·ăă M7 π Implement GetById (GET)

- First, we have to change the route. In this case, we have to put the /id in the annotation value.
- After that, we have to put the annotation @PathVariable in the id of the method to equal the id route in JSON request.

```
@GetMapping(value ="/{id}") //endpoint GET
public ResponseEntity<ClientDTO> findById(@PathVariable Long id){ //This is a endpoint to a client by id
    ClientDTO dto = service.findById(id);
    return ResponseEntity.ok().body(dto);
}
```

- Now, implement the findById() service method
- When we put the repository, we have a method findById that return a optional object. We have to convert him.

```
@Transactional(readOnly = true)
public ClientDTO findById(Long id) {
    Optional<Client> obj = repository.findById(id);
    Client entity = obj.get();
    return new ClientDTO(entity);
}
```

- But, if we put a wrong id in the request we get the 500 error. We have to treat this error

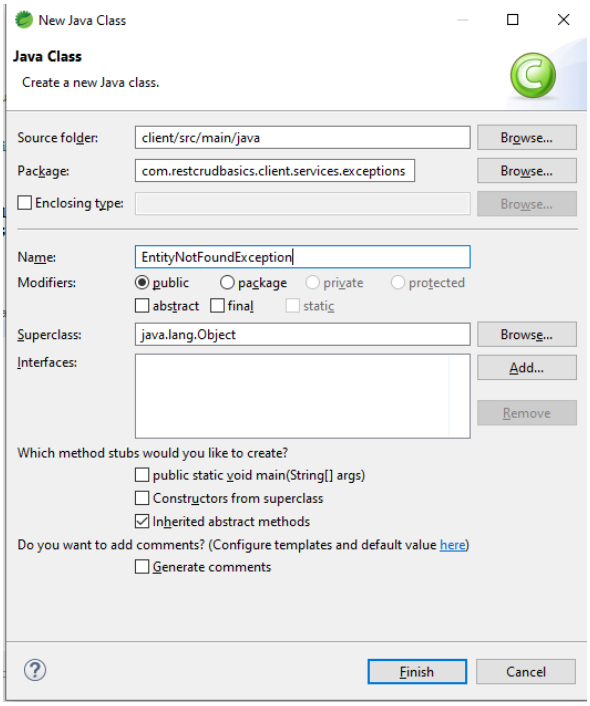
Body Cookies Headers (4) Test Results Status: 500 Internal Server Error

Pretty Raw Preview Visualize JSON

```
1 {
2     "timestamp": "2021-01-14T23:41:29.736+00:00",
3     "status": 500,
4     "error": "Internal Server Error",
5     "message": "",
6     "path": "/clients/5"
7 }
```

/fl·ăă M8 π Implement Exception to GetById (GET)

- When we need to pass a argument like a id in a request, we have to treat the errors if we put a argument that is not find. This kind of error we treat with exceptions.
- First, we need to create the exceptions package



- Implement the Exception

```
package com.restcrudbasics.client.services.exceptions;

public class EntityNotFoundException extends RuntimeException {
    private static final long serialVersionUID = 1L;

    public EntityNotFoundException(String msg) {
        super(msg);
    }

}
```

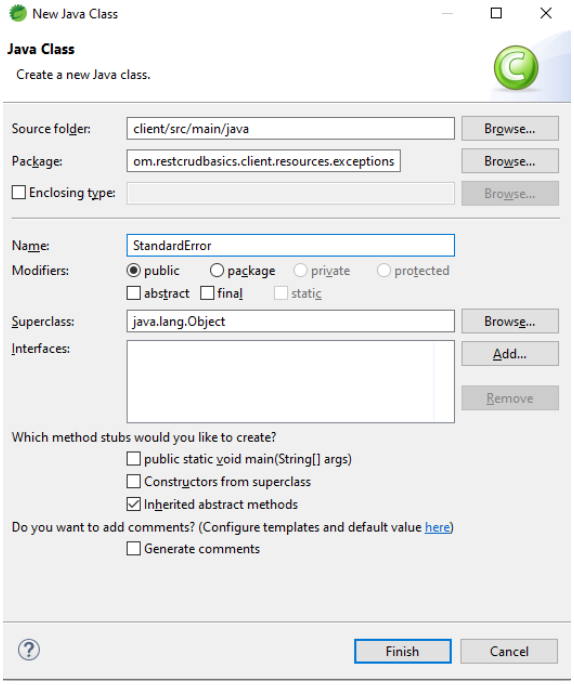
- Now, we need to implement the try-catch method in the service entity

```
@Transactional(readOnly = true)
public ClientDTO findById(Long id) {
    Optional<Client> obj = repository.findById(id);
    Client entity = obj.orElseThrow(() -> new EntityNotFoundException("Entity not found"));
    return new ClientDTO(entity);
}
```

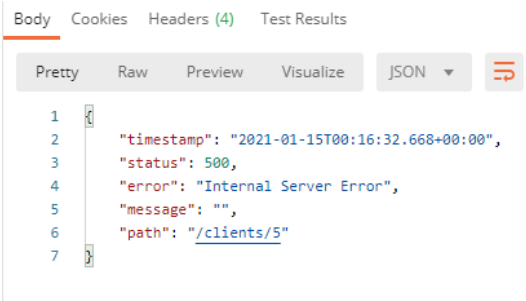
- If we run the app, we keep getting the 500 error, but, the log in STS how the custom exception that we create.

```
client - ClientApplication [Spring Boot App]
2021-01-14 22:16:32.662 ERROR 4100 --- [nio-8080-exec-2] o.a.c.c.C.[.[.[/].[dispatcherServlet]    : Servlet.service()

com.restcrudbasics.client.services.exceptions.EntityNotFoundException: Entity not found
        at com.restcrudbasics.client.services.ClientService.lambda$1(ClientService.java:31) ~[classes/:na]
        at java.base/java.util.Optional.orElseThrow(Optional.java:408) ~[na:na]
        at com.restcrudbasics.client.services.ClientService.findById(ClientService.java:31) ~[classes/:na]
```

- Now, we have to return a 404 response when the error occurs.
- For this, we have to implement the error in the resource, to show the error.
- Create a new class exception in the layer resource.



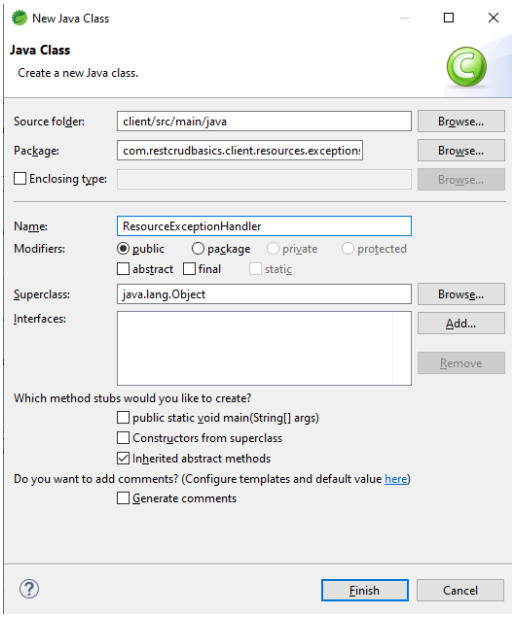
- This class have the same properties of the JSON error



- The implement of error class

```java
package com.restcrudbasics.client.resources.exceptions;

import java.time.Instant;

public class StandardError {
    private static final long serialVersionUID = 1L;

    private Instant timestamp;
    private Integer status;
    private String error;
    private String message;
    private String path;

    public StandardError() {

    }

    public Instant getTimestamp() {
        return timestamp;
    }

    public void setTimestamp(Instant timestamp) {
        this.timestamp = timestamp;
    }

    public Integer getStatus() {
        return status;
    }

    public void setStatus(Integer status) {
        this.status = status;
    }

    public String getError() {
        return error;
    }
```

- To not treat everyone Resource Method that throws an exception, we create a Advice Controller to economy and make a clean code.
- Let's create the class



- Now, we need to implement the Handler of the exception to show the customize treated error in the http response.

ClientResource.java | ClientService.java | ClientDTO.java | EntityNotFoundException.java | StandardError.java | ClientRepository.java | ResourceExceptionHandler.java

```java
package com.restcrudbasics.client.resources.exceptions;

import java.time.Instant;

import javax.servlet.http.HttpServletRequest;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

import com.restcrudbasics.client.services.exceptions.EntityNotFoundException;

@ControllerAdvice // intercept the error that occur in the resource layer
public class ResourceExceptionHandler {

    @ExceptionHandler(EntityNotFoundException.class) // every time a error EntityNotFoundException occur, the error will be treat with the ResponseEntity Method
    public ResponseEntity<StandardError> entityNotFound(EntityNotFoundException e, HttpServletRequest request){ //this method recive the exception and the information of the Http request
        StandardError err = new StandardError();
        err.setTimestamp(Instant.now());
        err.setStatus(HttpStatus.NOT_FOUND.value());
        err.setError("Resource not found");
        err.setMessage(e.getMessage());
        err.setPath(request.getRequestURI());
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(err);
    }

}
```

- Now, when the error occurs, the http response will show the treated error and 404 error status

Body | Cookies | Headers (5) | Test Results — Status: 404 Not Found

Pretty | Raw | Preview | Visualize | JSON

```
{
    "timestamp": "2021-01-15T01:23:17.872144900Z",
    "status": 404,
    "error": "Resource not found",
    "message": "Entity not found",
    "path": "/clients/5"
}
```

/fl·åå M9 π Implement Insert (POST)

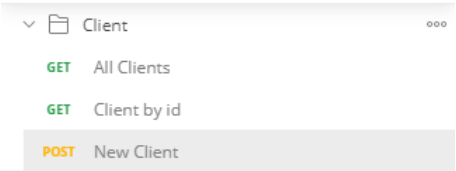
- Now, we will implement the POST method
- In this method, we will insert a new object
- We will create a method that return a 201 response and a message of “resource created” with a URI

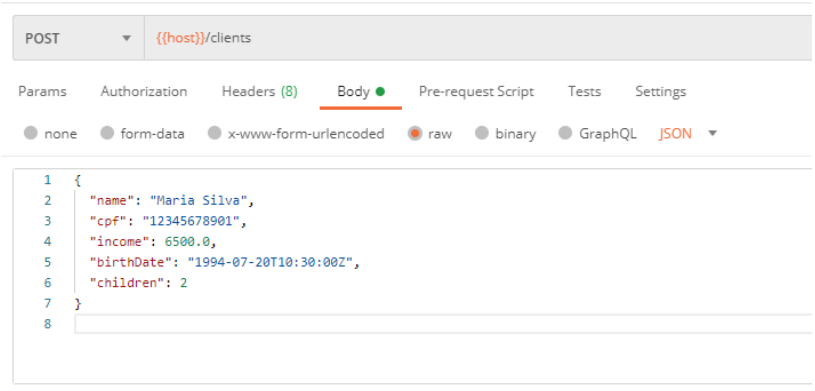
```
    @PostMapping
    public ResponseEntity<ClientDTO> insert(@RequestBody ClientDTO dto){
        dto = service.insert(dto);
        URI uri = ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}")
                .buildAndExpand(dto.getId()).toUri();
        return ResponseEntity.created(uri).body(dto);
    }
}
```

- Lets create the insert service

```
@Transactional
public ClientDTO insert(ClientDTO dto) {
    Client entity = new Client();
    entity.setName(dto.getName());
    entity.setCpf(dto.getCpf());
    entity.setIncome(dto.getIncome());
    entity.setBirthDate(dto.getBirthDate());
    entity.setChildren(dto.getChildren());
    entity = repository.save(entity);
    return new ClientDTO(entity);
}
```

- Now, to test, we have to create a body in JSON Request
- Go in Body, Raw, JSON and elaborate the insert in JSON format





- Give the status 201, so, is everything ok
- If we see the headers, we are able to see the location of the new object

Body Cookies Headers (6) Test Results Status: 201 Created

Pretty Raw Preview Visualize JSON


```
{
    "id": 4,
    "name": "Maria Silva",
    "cpf": "12345678901",
    "income": 6500.0,
    "birthDate": "1994-07-20T10:30:00Z",
    "children": 2
}
```


Body Cookies Headers (6) Test Results

KEY	VALUE
Location	http://localhost:8080/clients/4
Content-Type	application/json
Transfer-Encoding	chunked
Date	Fri, 15 Jan 2021 02:03:06 GMT
Keep-Alive	timeout=60
Connection	keep-alive

Class 20 - Implement Update (PUT)

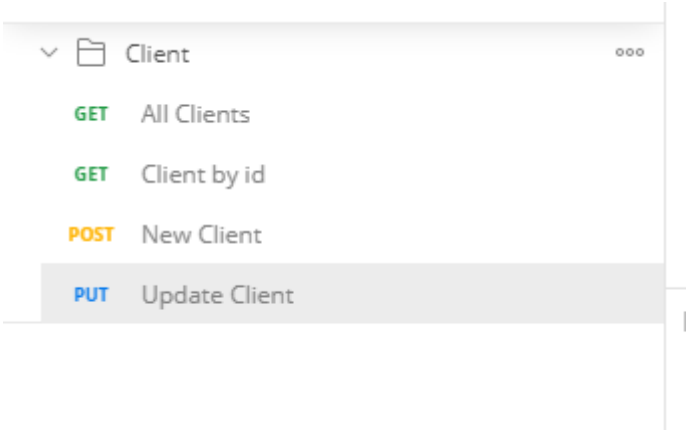
- Implement Put method

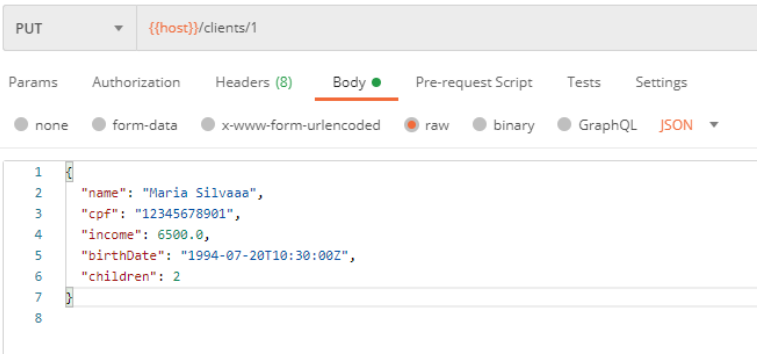
```
    @PutMapping(value ="/{id}")
    public ResponseEntity<ClientDTO> update(@PathVariable Long id, @RequestBody ClientDTO dto){
        dto = service.update(id, dto);
        return ResponseEntity.ok().body(dto);
    }
}
```

- Implement update service
- If we put the findById in service, we will access the database twice (To find and to save). To not access twice the database we use the getOne method.
- Be careful, we have a mistake in the name of the exception

```
@Transactional
public ClientDTO update(Long id, ClientDTO dto) {
    try {
        Client entity = repository.getOne(id);
        entity.setName(dto.getName());
        entity.setCpf(dto.getCpf());
        entity.setIncome(dto.getIncome());
        entity.setBirthDate(dto.getBirthDate());
        entity.setChildren(dto.getChildren());
        entity = repository.save(entity);
        return new ClientDTO(entity);
    }
    catch(EntityNotFoundException e){
        throw new ResourceNotFoundException("Id not found " + id);
    }

}
```

- Lets Test in Postman





Body Cookies Headers (5) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON


```
{
    "id": 1,
    "name": "Maria Silvaaa",
    "cpf": "12345678901",
    "income": 6500.0,
    "birthDate": "1994-07-20T10:30:00Z",
    "children": 2
}
```

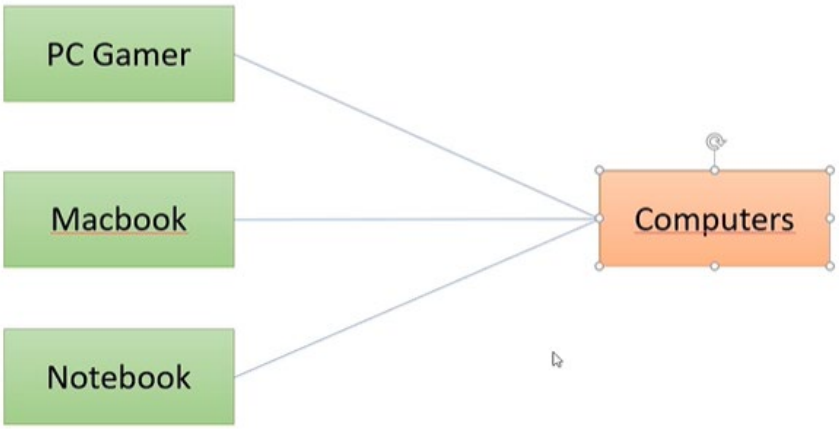

- Give the 200 response. Is ok

Class 21 - Implement Delete (Delete)

- Implement Delete Resource layer

```
@DeleteMapping(value ="/{id}")
public ResponseEntity<ClientDTO> delete(@PathVariable Long id){
    service.delete(id);
    return ResponseEntity.noContent().build();
}
```

- Now, implement service layer.
- When we implement delete, we can throw two errors. One error if we put a wrong id and another error if we delete a object that is primary key from another object in the database. In this case, we just have the Client entity. If the entity has a relation with another entity, we have to treat this error. Is a DataIntegrityViolationException.




```
public void delete(Long id) {
    try {
        repository.deleteById(id);
    }
    catch (EmptyResultDataAccessException e) {
        throw new ResourceNotFoundException("Id not found " + id);
    }
    catch (DataIntegrityViolationException e) {
        throw new DatabaseException("Integrity violation");
    }
```

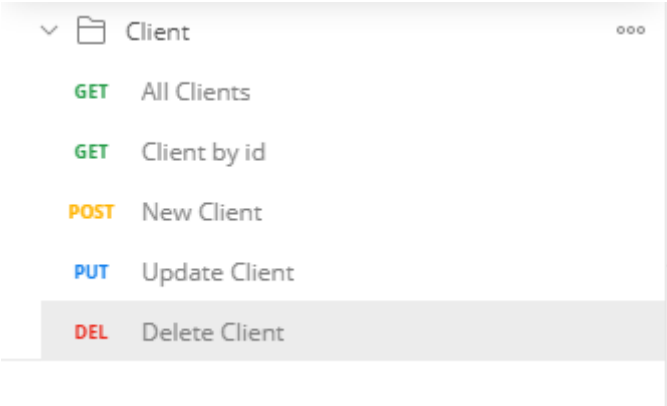
- Create a new Exception class

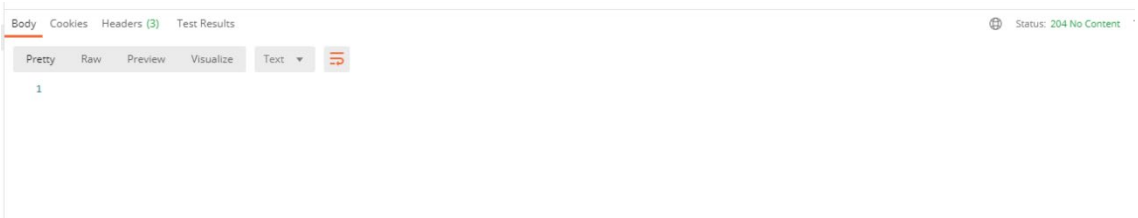
```java
package com.devsuperior.dscatalog.services.exceptions;

public class DatabaseException extends RuntimeException {
    private static final long serialVersionUID = 1L;

    public DatabaseException(String msg) {
        super(msg);
    }
}

```

- Now, we gonna pass the response of the error if this occur

```java
@ExceptionHandler(ResourceNotFoundException.class)
public ResponseEntity<StandardError> entityNotFound(ResourceNotFoundException e, HttpServletRequest reque
    HttpStatus status = HttpStatus.BAD_REQUEST;
    StandardError err = new StandardError();
    err.setTimestamp(Instant.now());
    err.setStatus(status.value());
    err.setError("Resource not found");
    err.setMessage(e.getMessage());
    err.setPath(request.getRequestURI());
    return ResponseEntity.status(status).body(err);
}

@ExceptionHandler(DatabaseException.class)
public ResponseEntity<StandardError> database(DatabaseException e, HttpServletRequest request) {
    HttpStatus status = HttpStatus.BAD_REQUEST;
    StandardError err = new StandardError();
    err.setTimestamp(Instant.now());
    err.setStatus(status.value());
    err.setError("Database exception");
    err.setMessage(e.getMessage());
    err.setPath(request.getRequestURI());
    return ResponseEntity.status(status).body(err);
```

- Let test





- Give the response 204. Ok

Class HH - Implement Auditing Data

- If we want to implement auditing data when we create a new object or update a object, we can implement methods in the entity and create columns in the entity table.
- First, implement the attributes in the entity

```
@Id // the primary key of the table will be the attribute id from the client
@GeneratedValue(strategy = GenerationType.IDENTITY) // auto increment id (primary key)
private Long id;
private String name;
private String cpf;
private Double income;
private Instant birthDate;
private Integer children;

private Instant createdAt;

private Instant updatedAt;
```

- Now, we have to put a annotation in the attributes to inform that we want store this data in the UTC form.

```
@Id // the primary key of the table will be the attribute id from the client
@GeneratedValue(strategy = GenerationType.IDENTITY) // auto increment id (primary key)
private Long id;
private String name;
private String cpf;
private Double income;
private Instant birthDate;
private Integer children;

@Column(columnDefinition = "TIMESTAMP WITHOUT TIME ZONE")
private Instant createdAt;

@Column(columnDefinition = "TIMESTAMP WITHOUT TIME ZONE")
private Instant updatedAt;

public Client() {
}
```

- Now, we have to create the getters and setters methods. We will not create the setters methods, because we don't want to set methods for the auditing data, just get.

```
public void setChildren(Integer chldren) {
    this.children = chldren;
}

public Instant getCreatedAt() {
    return createdAt;
}

public Instant getUpdatedAt() {
    return updatedAt;
}
```

- To get the instant time when create or update, we will create a method to store the instant time.

```java
public Instant getCreatedAt() {
    return createdAt;
}

public Instant getUpdatedAt() {
    return updatedAt;
}

public void prePersist() {
    createdAt = Instant.now();
}

public void preUpdate() {
    updatedAt = Instant.now();
}
```

- To make this method return the instant, we put a annotation.

```java
public Instant getCreatedAt() {
    return createdAt;
}

public Instant getUpdatedAt() {
    return updatedAt;
}

@PrePersist
public void prePersist() {
    createdAt = Instant.now();
}

@PreUpdate
public void preUpdate() {
    updatedAt = Instant.now();
}
```

- We don’t create the attributes in the DTO package, because we don’t want to inform to the user the instant data.

- When we run the app, the tables was created

TB_CLIENT
- ID
- BIRTH_DATE
- CHILDREN
- CPF
- CREATED_AT
- INCOME
- NAME
- UPDATED_AT

- Let’s test the Update and Create

POST {{host}}/clients

Params | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings

Query Params

KEY	VALUE
Key	Value

Body | Cookies | Headers (6) | Test Results

Pretty | Raw | Preview | Visualize | JSON


```
{
    "id": 4,
    "name": "Maria Silva",
    "cpf": "12345678901",
    "income": 6500.0,
    "birthDate": "1994-07-20T10:30:00Z",
    "children": 2
}
```

```
SELECT * FROM TB_CLIENT
```

SELECT * FROM TB_CLIENT;

ID	BIRTH_DATE	CHILDREN	CPF	CREATED_AT	INCOME	NAME	UPDATED_AT
1	2011-12-03 08:15:30	1	111.111.111-11	*null*	1000.0	JOAO	*null*
2	2011-12-03 08:15:30	2	222.222.222-22	*null*	2000.0	MARIA	*null*
3	2011-12-03 08:15:30	3	333.333.333-33	*null*	3000.0	JOSE	*null*
4	1994-07-20 07:30:00	2	12345678901	2021-01-24 18:54:26.806998	6500.0	Maria Silva	*null*

(4 rows, 4 ms)

GET All Clients | POST New Client | PUT Update Client

Update Client

PUT {{host}}/clients/1

Params | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings

Query Params

KEY	VALUE
Key	Value

Body | Cookies | Headers (5) | Test Results

Pretty | Raw | Preview | Visualize | JSON


```
{
    "id": 1,
    "name": "Maria Silvaaa",
    "cpf": "12345678901",
    "income": 6500.0,
    "birthDate": "1994-07-20T10:30:00Z",
    "children": 2
}
```


SELECT * FROM TB_CLIENT;

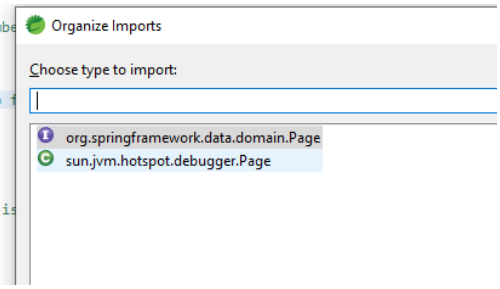
ID	BIRTH_DATE	CHILDREN	CPF	CREATED_AT	INCOME	NAME	UPDATED_AT
1	1994-07-20 07:30:00	2	12345678901	*null*	6500.0	Maria Silvaaa	2021-01-24 18:55:53.081488
2	2011-12-03 08:15:30	2	222.222.222-22	*null*	2000.0	MARIA	*null*
3	2011-12-03 08:15:30	3	333.333.333-33	*null*	3000.0	JOSE	*null*
4	1994-07-20 07:30:00	2	12345678901	2021-01-24 18:54:26.806998	6500.0	Maria Silva	*null*

(4 rows, 1 ms)

Class 23 - Implement Pagination

- Now, we will implement the pagination.
- First, we will modify the return of the HTTP request in the resource entity GET from List to Page.

```
@Autowired //This annotation will make this instance auto injected (Remembe
private ClientService service;

@GetMapping //endpoint GET
public ResponseEntity<Page<ClientDTO>> findAll(){ //This is a endpoint to f
    List<ClientDTO> list = service.findAll();
    return ResponseEntity.ok().body(list);
}

@GetMapping(value ="/{id}") //endpoint GET
public ResponseEntity<ClientDTO> findById(@PathVariable Long id){ //This is
    ClientDTO dto = service.findById(id);
    return ResponseEntity.ok().body(dto);
}
```



- When we make a pagination, we can pass some attributes to the request like this.

```
@RequestParam(value = "page", defaultValue = "0") Integer page,
@RequestParam(value = "linesPerPage", defaultValue = "12") Integer linesPerPage,
@RequestParam(value = "orderBy", defaultValue = "name") String orderBy,
@RequestParam(value = "direction", defaultValue = "DESC") String direction
){
```

- The annotation @RequestParam is optional in the request. If we want a parameter obligate, we put the annotation @PathVariable.
- Page = The first exhibition page. linesPerPage = Number of pages we want. orderBy = The form we want to order. Direction = if will be Descending or ascending.
- Put the annotation in the body of the Getall method

```
@GetMapping //endpoint GET
public ResponseEntity<Page<ClientDTO>> findAll(

        @RequestParam(value = "page", defaultValue = "0") Integer page,
        @RequestParam(value = "linesPerPage", defaultValue = "12") Integer linesPerPage,
        @RequestParam(value = "orderBy", defaultValue = "moment") String orderBy,
        @RequestParam(value = "direction", defaultValue = "DESC") String direction
        ){

            //This is a endpoint to find all clients
    List<ClientDTO> list = service.findAll();
    return ResponseEntity.ok().body(list);
}
```

- Now, we will change the get method to return a page and instant a object in spring that is like a page object.

```
@GetMapping //endpoint GET
public ResponseEntity<Page<ClientDTO>> findAll(

        @RequestParam(value = "page", defaultValue = "0") Integer page,
        @RequestParam(value = "linesPerPage", defaultValue = "12") Integer linesPerPage,
        @RequestParam(value = "orderBy", defaultValue = "moment") String orderBy,
        @RequestParam(value = "direction", defaultValue = "DESC") String direction
        ){

    PageRequest pageRequest = PageRequest.of(page, linesPerPage, Direction.valueOf(direction), orderBy);

            //This is a endpoint to find all clients
    Page<ClientDTO> list = service.findAllPaged(pageRequest);

    return ResponseEntity.ok().body(list);
}
```

- Now, we have to change the findAll service.
- The findAllPaged will receive a PageRequest
- The repository method already have a findAll method receiving a PageRequest

```
@Transactional(readOnly = true) // Transactional close when make the REST request. Good Practice in Program
public Page<ClientDTO> findAllPaged(PageRequest pageRequest){
    Page<Client> list = repository.findAll(pageRequest); // We have to convert this Client list to ClientDTO list.
    return list.map(x -> new ClientDTO(x)); // Use Functional program
}
```

- Now, let test


```
{
    "content": [
        {
            "id": 34,
            "name": "ROSE",
            "cpf": "444.444.444-44",
            "income": 4000.0,
            "birthDate": "1993-12-03T10:15:30Z",
            "children": 4
        },
        {
            "id": 14,
            "name": "ROSE",
            "cpf": "444.444.444-44",
            "income": 4000.0,
            "birthDate": "1993-12-03T10:15:30Z",
            "children": 4
        },
        {
            "id": 44,
            "name": "ROSE",
            "cpf": "444.444.444-44",
```


- When we make the request GetAll, the response is a content, with the form of Descending for client name. Is this case, the page 0 have the name Rose.
- We can alter the path of the request, like the page, linesPerPage, direction and order by

```
/clients?page=0&linesPerPage=6&direction=ASC&orderBy=name
```

```
{
    "content": [
        {
            "id": 37,
            "name": "BRUNO",
            "cpf": "777.777.777-77",
            "income": 7000.0,
            "birthDate": "1996-12-03T10:15:30Z",
            "children": 7
        },
        {
            "id": 7,
            "name": "BRUNO",
            "cpf": "777.777.777-77",
            "income": 7000.0,
            "birthDate": "1996-12-03T10:15:30Z",
            "children": 7
        },
        {
            "id": 17,
            "name": "BRUNO",
            "cpf": "777.777.777-77",
```


/clients?page=0&linesPerPage=6&direction=ASC&orderBy=birthdate


```
            "name": "JOAO",
            "cpf": "111.111.111-11",
            "income": 1000.0,
            "birthDate": "1990-12-03T10:15:30Z",
            "children": 1
        },
        {
            "id": 1,
            "name": "JOAO",
            "cpf": "111.111.111-11",
            "income": 1000.0,
            "birthDate": "1990-12-03T10:15:30Z",
            "children": 1
        },
        {
            "id": 2,
            "name": "MARIA",
            "cpf": "222.222.222-22",
            "income": 2000.0,
            "birthDate": "1991-12-03T10:15:30Z",
            "children": 2
```


/clients?page=0&linesPerPage=6&direction=ASC&orderBy=cpf

```
        "income": 10000.0,
        "birthDate": "1999-12-03T10:15:30Z",
        "children": 10
    },
    {
        "id": 40,
        "name": "ROBERTA",
        "cpf": "101.101.101-10",
        "income": 10000.0,
        "birthDate": "1999-12-03T10:15:30Z",
        "children": 10
    },
    {
        "id": 21,
        "name": "JOAO",
        "cpf": "111.111.111-11",
        "income": 1000.0,
        "birthDate": "1990-12-03T10:15:30Z",
        "children": 1
    }
```