PROJETO CRUD - BACKEND

SUMÁRIO

OBJETIVO

Criar uma API RESTful com com as funcionalidades de CRUD. Criar um front end intuitivo, sincronizado com o backend, para realizar as operações de visualização, inserção, edição e deleção dos dados.

DEPENDÊNCIAS

O projeto será implementado conforme abaixo:

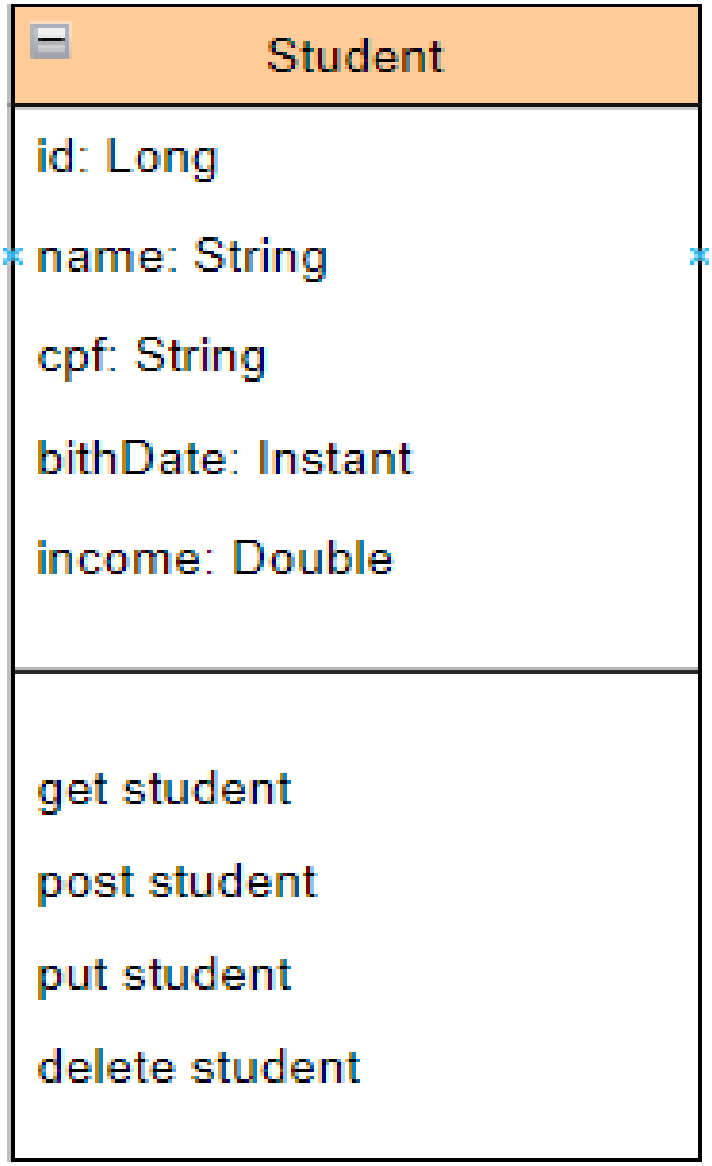
- **IDE**: Spring Tool Suite 4 (STS)
- **Banco de Dados**: H2
- **Gerenciador de Dependências**: Maven
- **Linguagem**: Java
- **Versionamento**: GitHub
- **Testes da API**: Postman
- **Documentação:** Swagger OpenApi

CASO DE USO

Uma Universidade precisa cadastrar os seus alunos, conforme segue abaixo:

- NOME
- CPF
- DATA DE NASCIMENTO
- RENDA

DIAGRAMA DE CLASSE



TESTES A SEREM REALIZADOS NO POSTMAN

BUSCA PAGINADA DE ALUNOS

- GET /students?page=0&sort=name,asc

BUSCA DE ALUNO POR ID

- GET /students/1

INSERIR NOVO ALUNO

- POST /students

```
{
  "name": "Auricelio Freitas",
  "cpf": "12345678901",
  "birthDate": "1982-08-28T10:30:00Z",
  "income": 15089.0
}
```

ATUALIZAR ALUNO

- PUT /students/1

```
{
  "name": "Auricelio Moreira",
  "cpf": "12345678901",
  "birthDate": "1982-08-28T10:30:00Z",
  "income": 15089.0
}
```

DELETAR ALUNO

- DELETE /students/1

IMPLEMENTAR O BACKEND

DEFINIR O WORKSPACE PARA O PROJETO

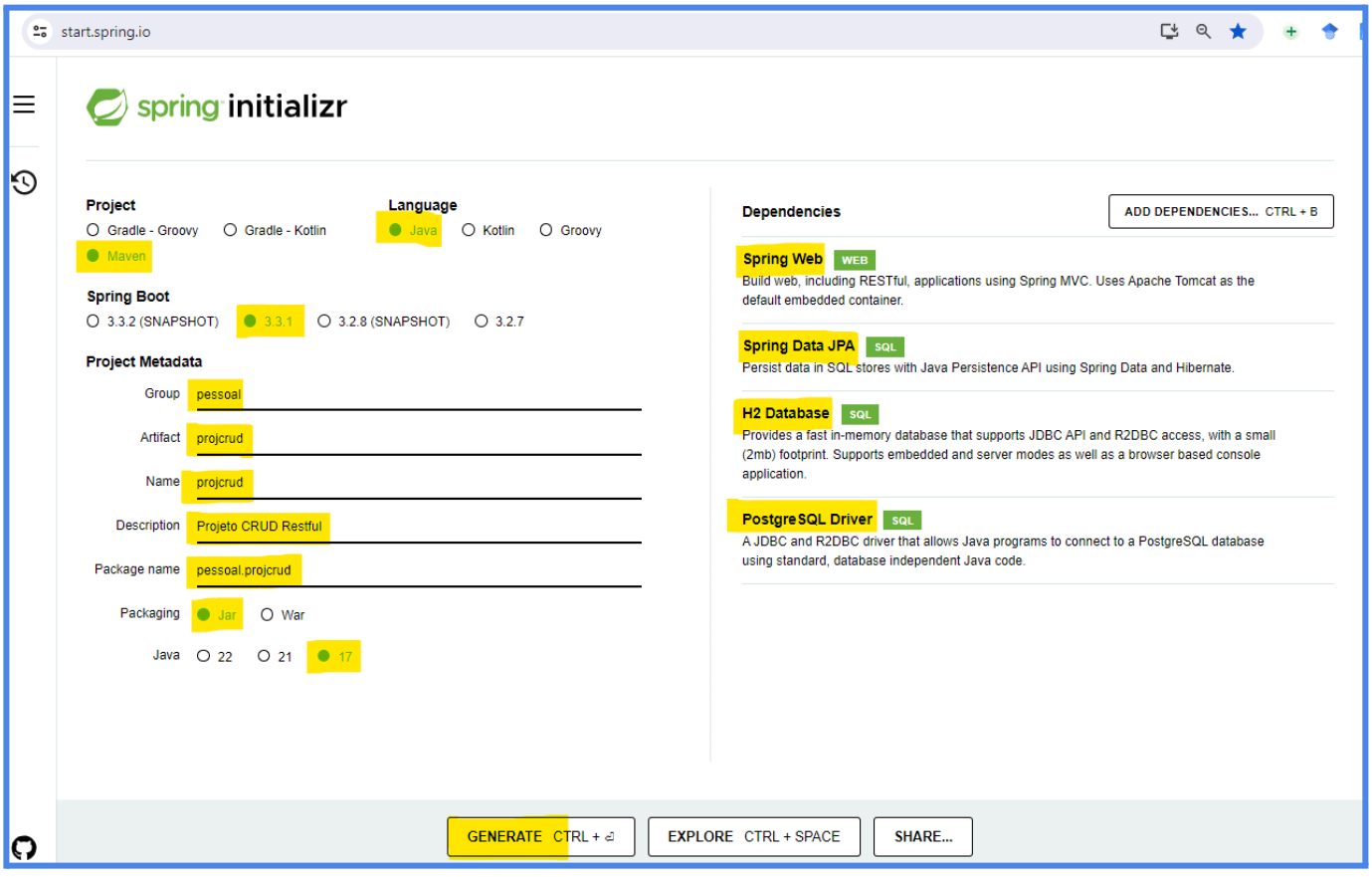
- C:\PROJETOS\ProjCrudBackFront

CRIAR O PROJETO COM O INITIALIZR

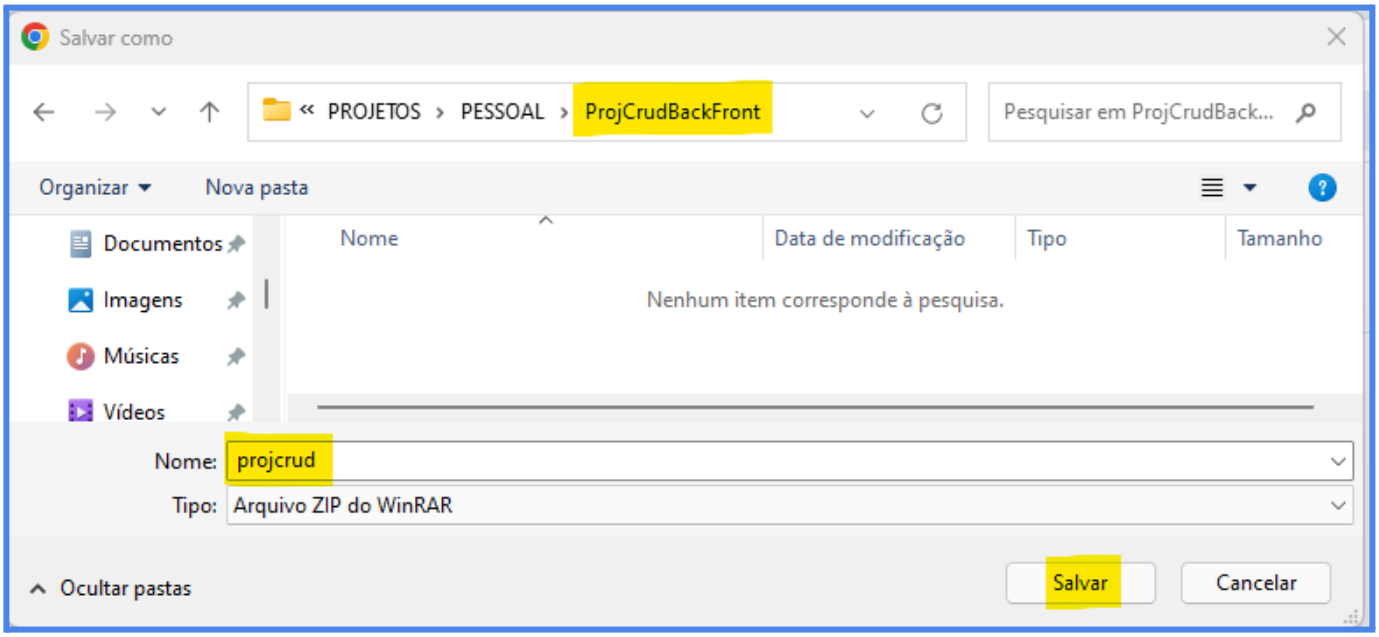
ACESSAR A URL

- https://start.spring.io/

GERAR O PROJETO

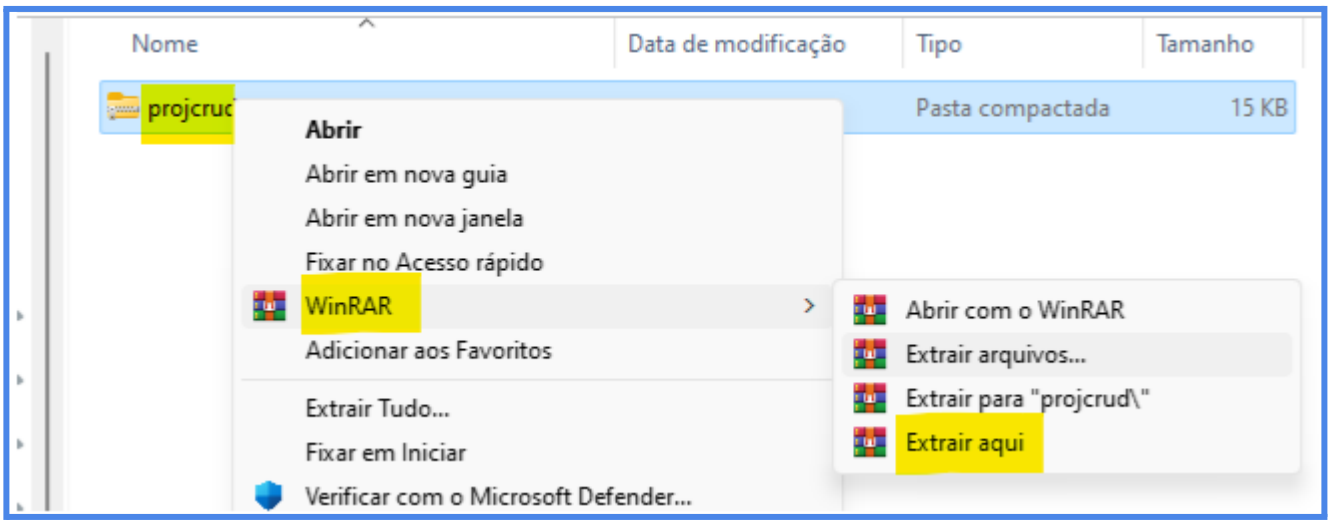


Salvar dentro do workspace

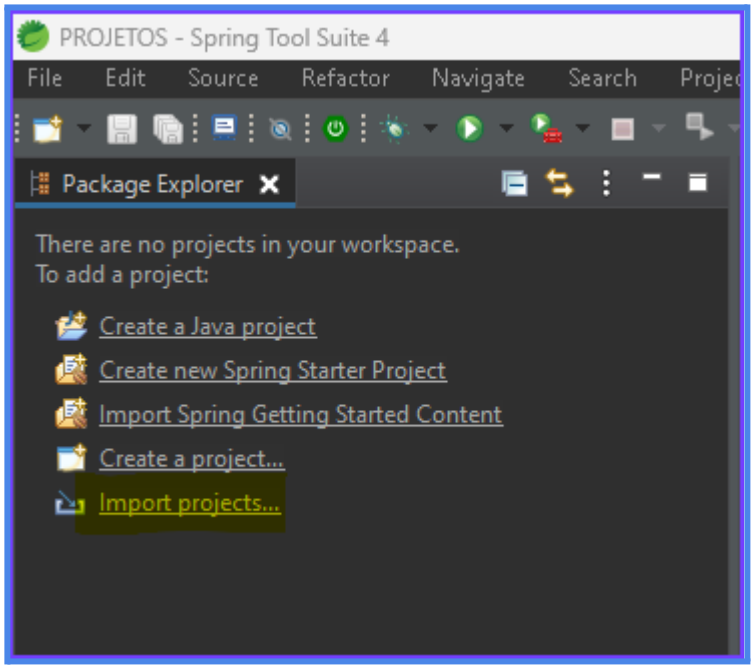


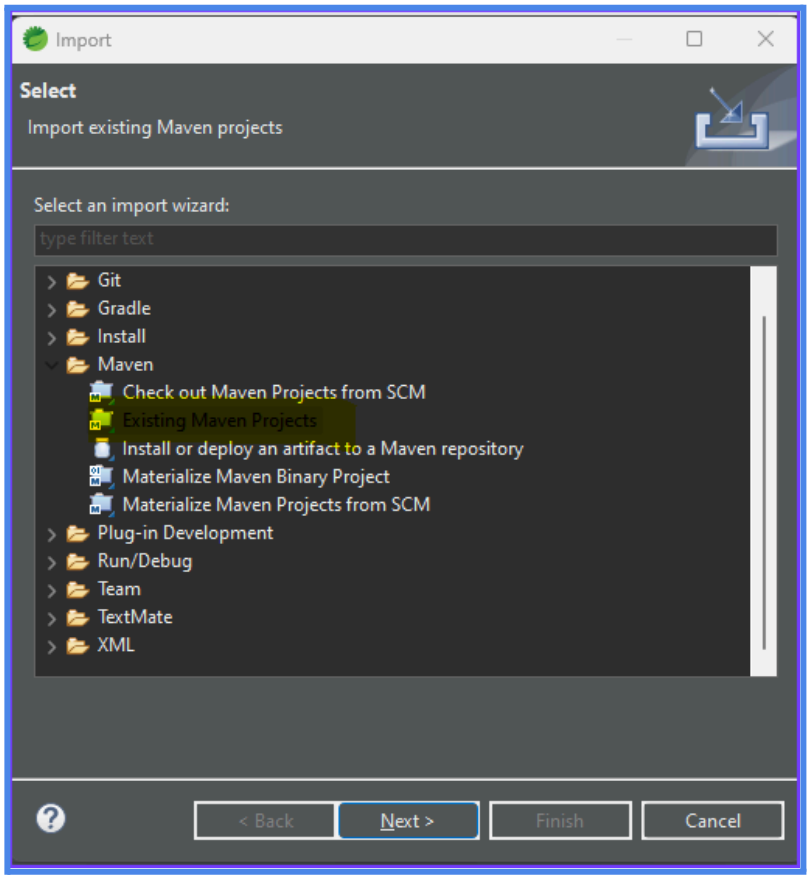
Descompactar e Renomear

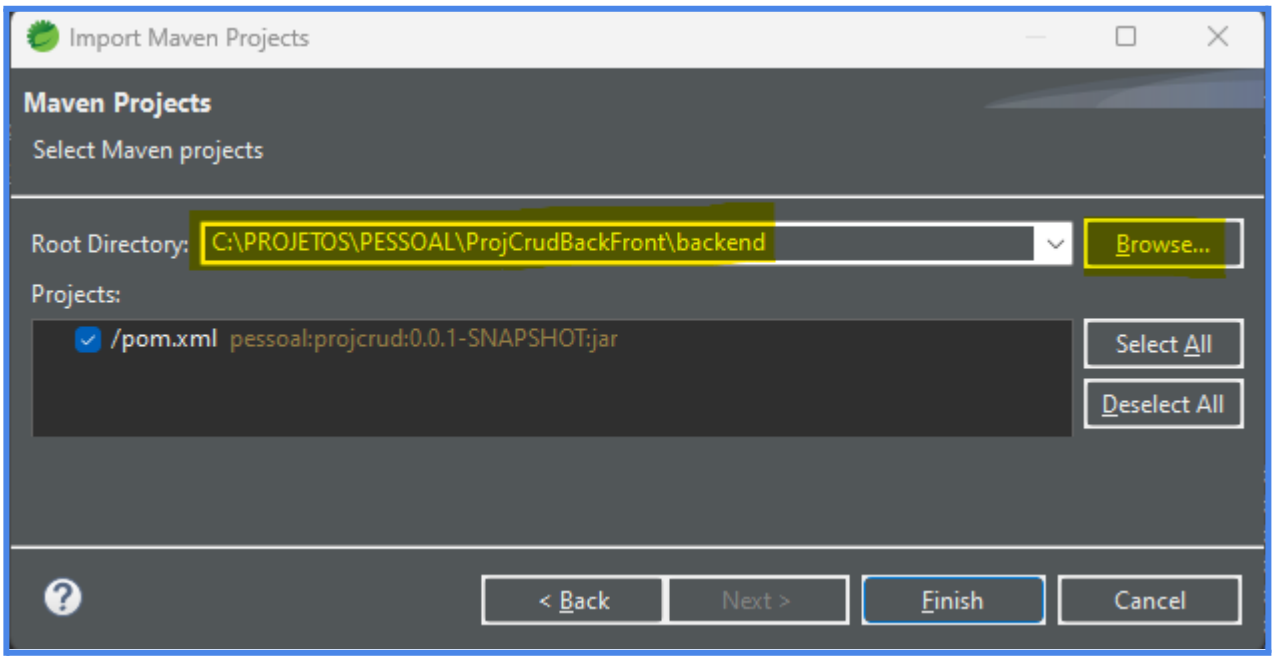
- backend



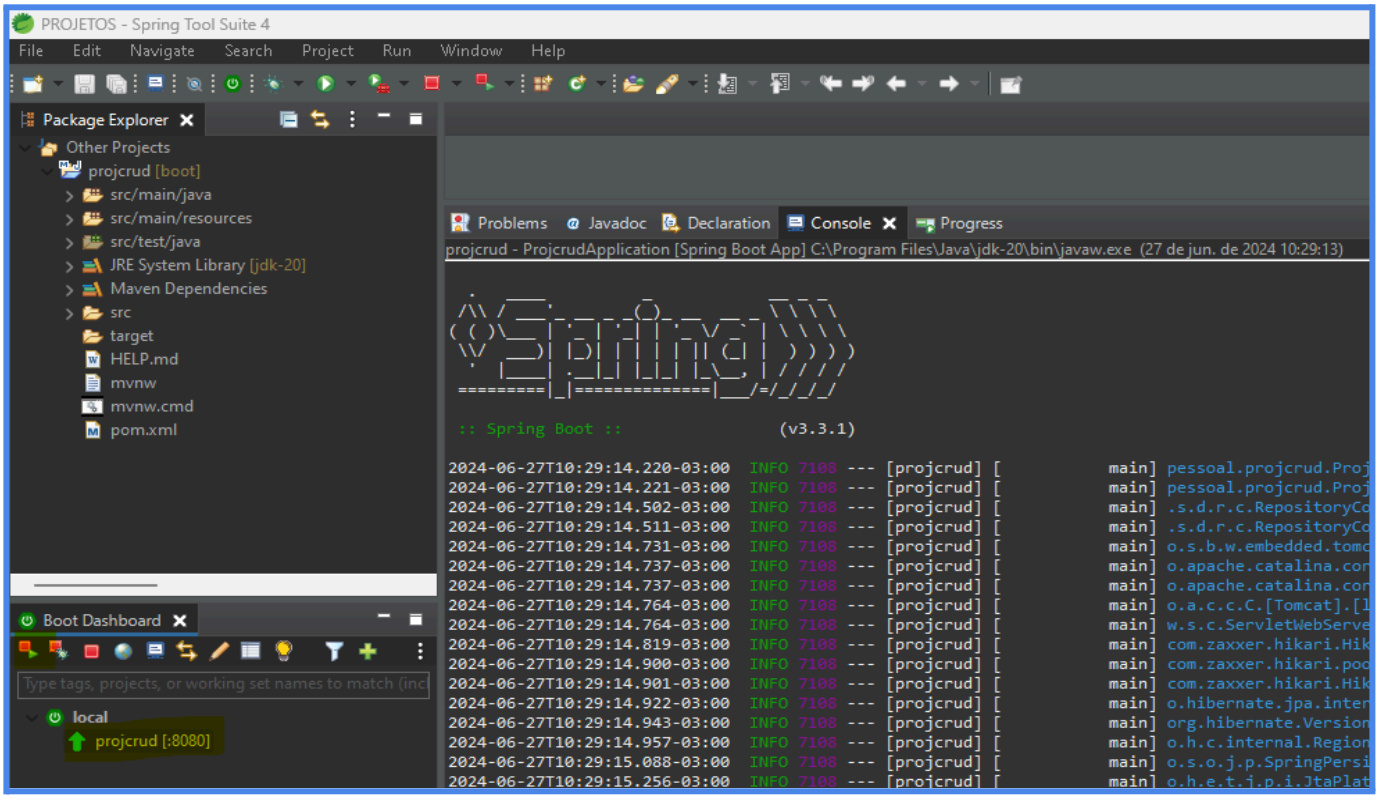
IMPORTAR O PROJETO PARA O STS





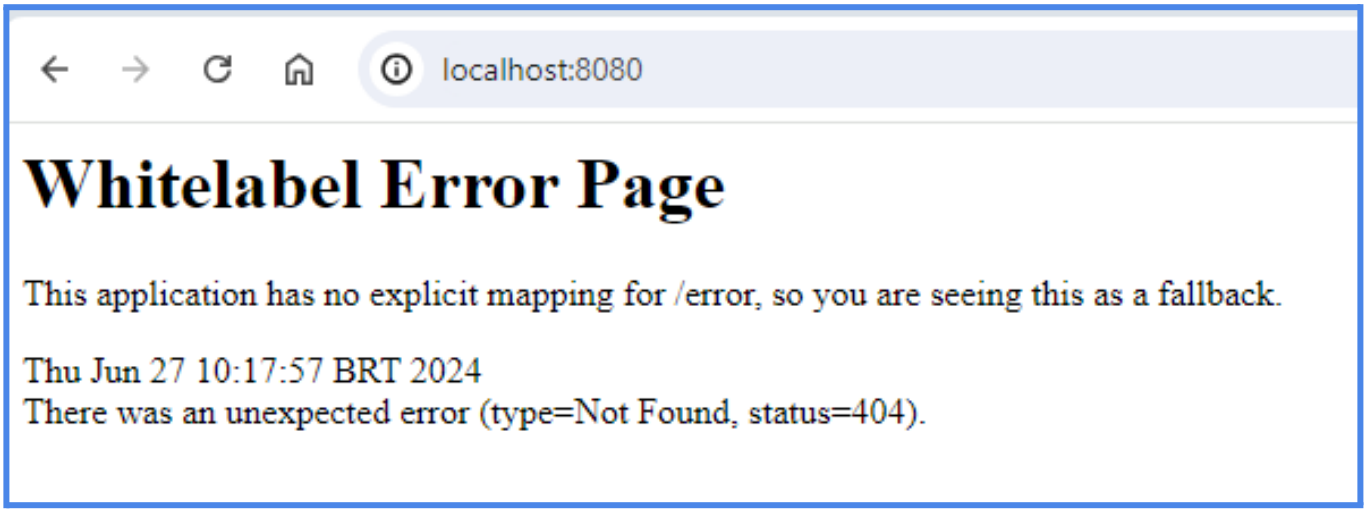


- **Rodar o projeto**



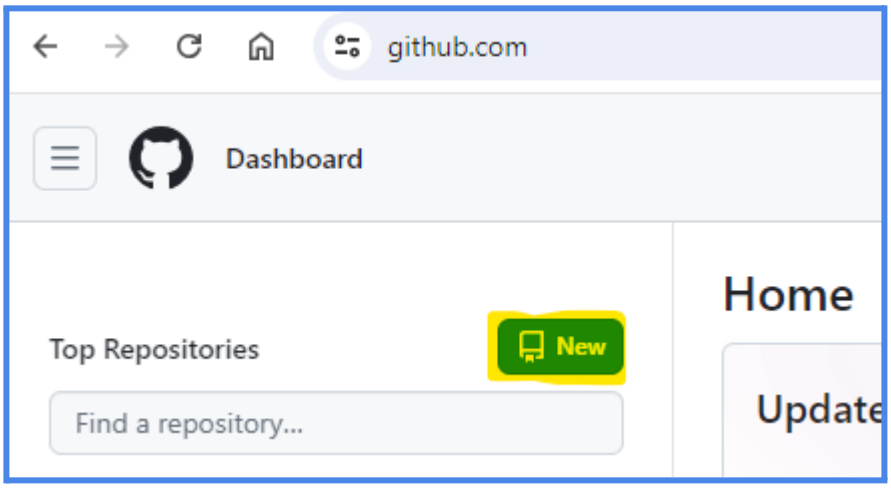
TESTAR NO BROWSER

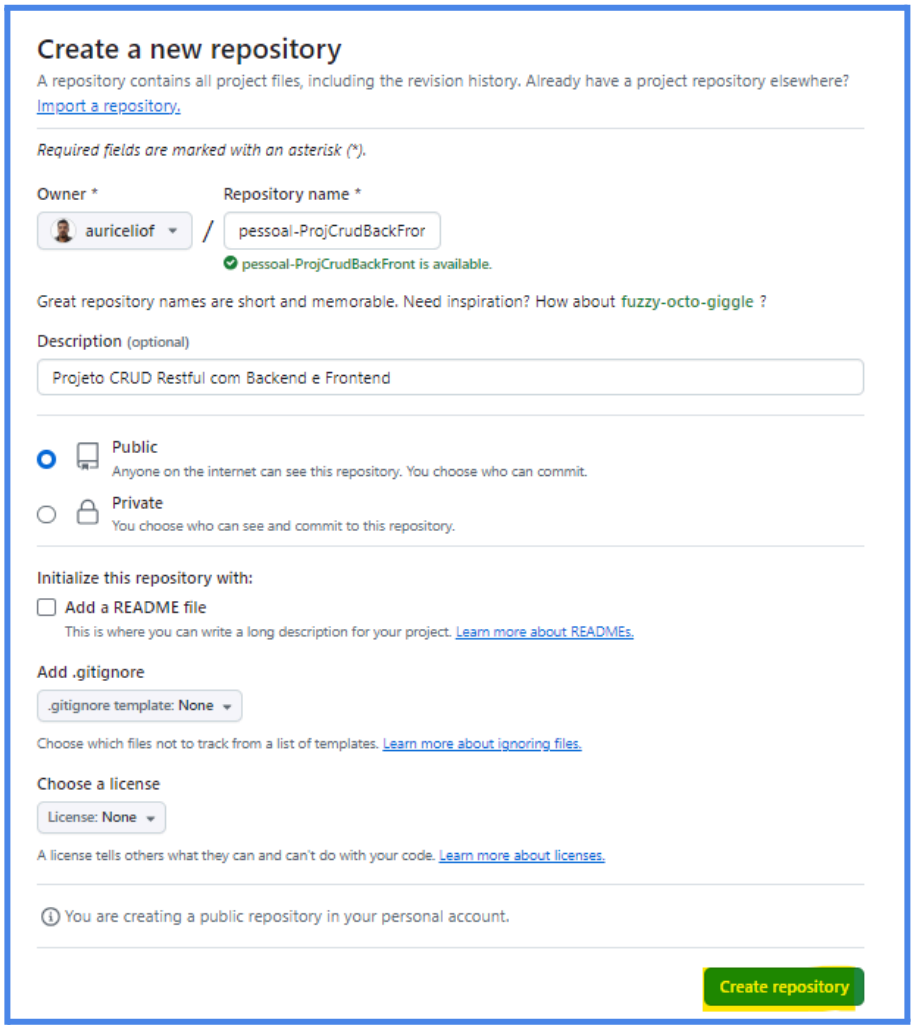
- **URL:** http://localhost:8080/

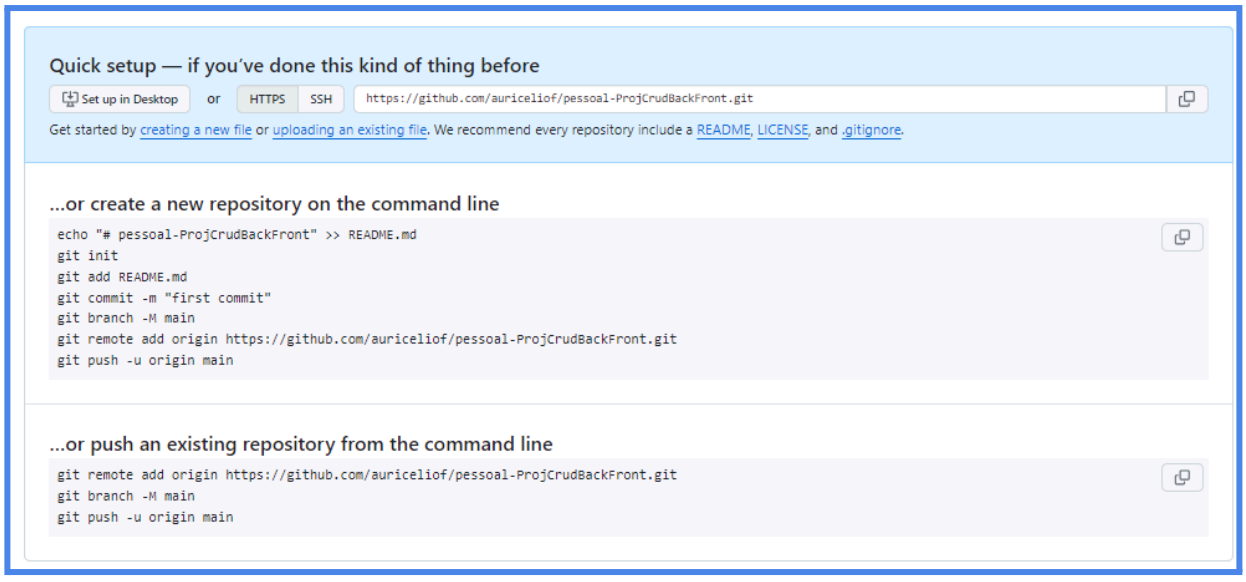


VERSIONAMENTO

Criar o projeto no Github



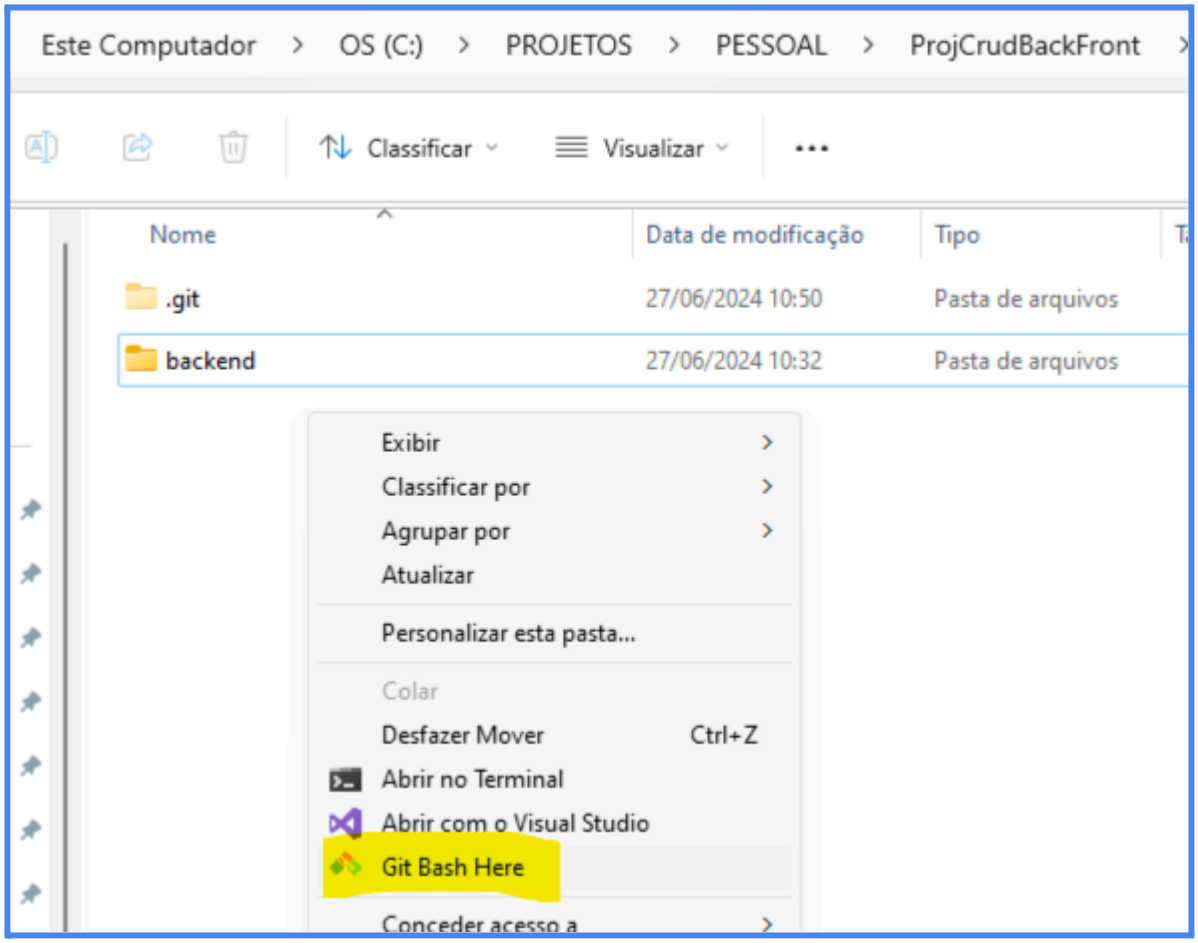

Create a new repository
A repository contains all project files, including the revision history. Already have a project repository elsewhere?
Import a repository.
Required fields are marked with an asterisk (*).
Owner *
Repository name *
auriceliof
pessoal-ProjCrudBackFror
pessoal-ProjCrudBackFront is available.
Great repository names are short and memorable. Need inspiration? How about fuzzy-octo-giggle ?
Description (optional)
Projeto CRUD Restful com Backend e Frontend
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:
Add a README file
This is where you can write a long description for your project. Learn more about READMEs.
Add .gitignore
.gitignore template: None
Choose which files not to track from a list of templates. Learn more about ignoring files.
Choose a license
License: None
A license tells others what they can and can't do with your code. Learn more about licenses.
You are creating a public repository in your personal account.
Create repository


Quick setup — if you've done this kind of thing before
Set up in Desktop
or
HTTPS
SSH
https://github.com/auriceliof/pessoal-ProjCrudBackFront.git
Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.
...or create a new repository on the command line
echo "# pessoal-ProjCrudBackFront" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/auriceliof/pessoal-ProjCrudBackFront.git
git push -u origin main
...or push an existing repository from the command line
git remote add origin https://github.com/auriceliof/pessoal-ProjCrudBackFront.git
git branch -M main
git push -u origin main

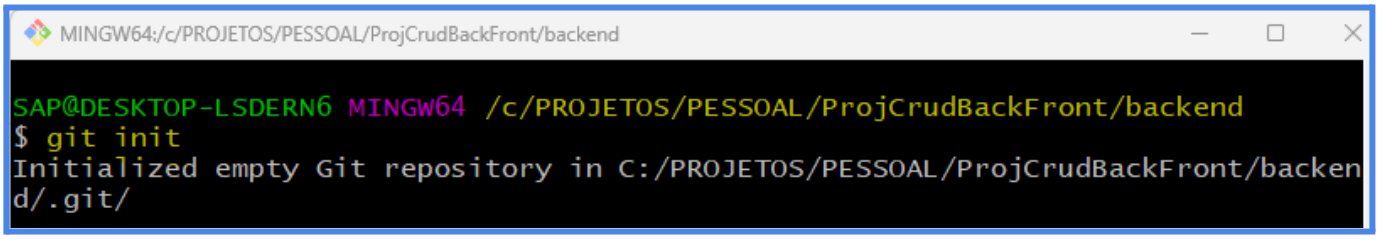
Sincronizar com o projeto local via git

NOTA: O Git deve estar instalado no computador. Caso não tenha ainda, baixe a versão mais atual e instale, dando next até o final.

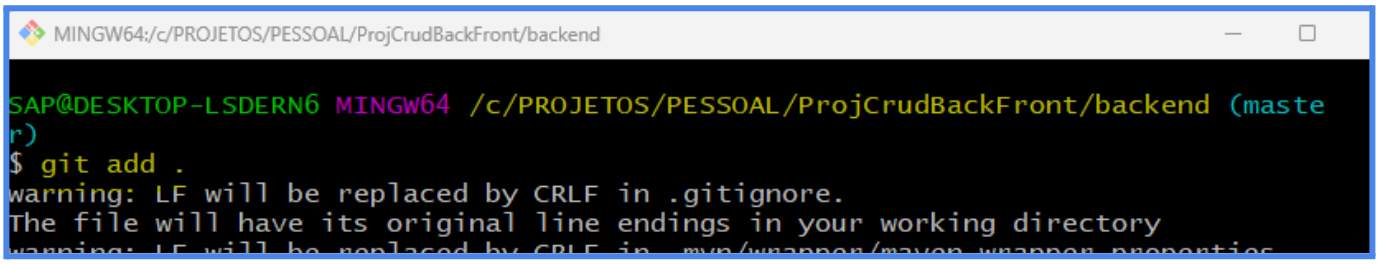
- **Botão direito no projeto e clicar em "Open Git bash here"**



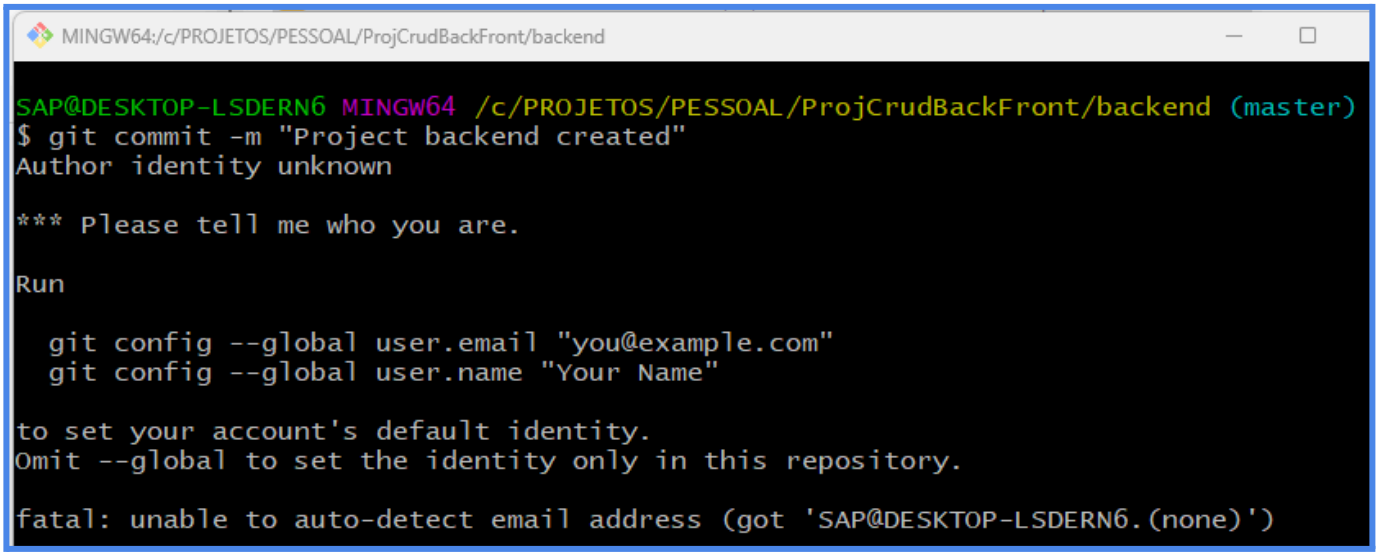
- Efetuar os comando abaixo
    - git init



    - git add .

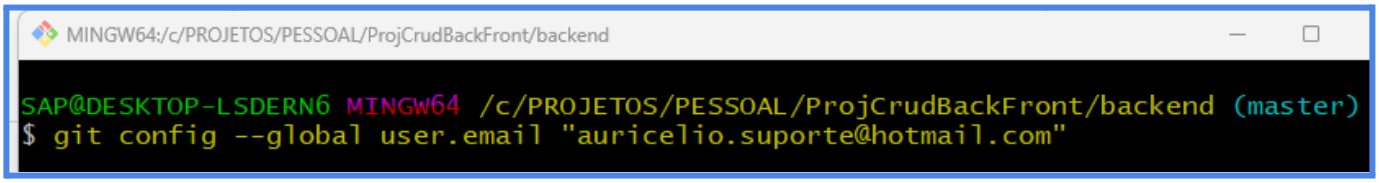


    - git commit -m "Project backend created"

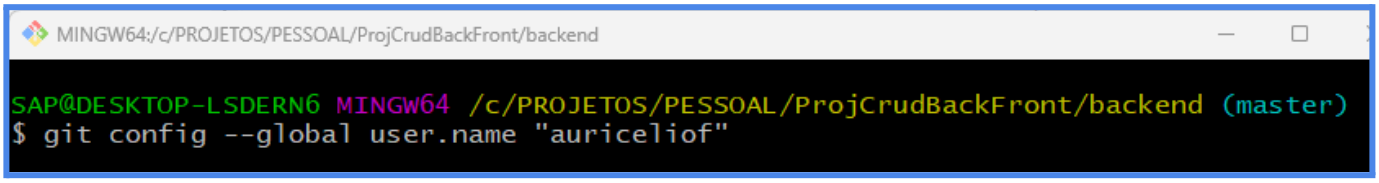


NOTA: Caso seja a primeira vez que sincronize no github no computador atual, irá pedir para informar o email e o nome da conta GitHub. Na próxima vez isso não irá aparecer.

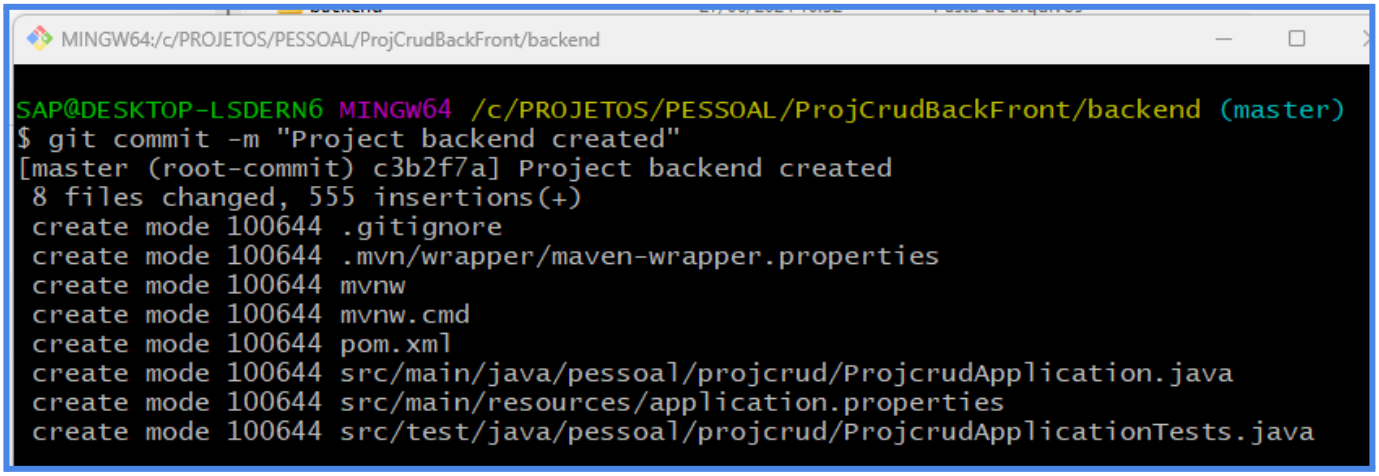
- git config --global user.email "auricelio.suporte@hotmail.com"



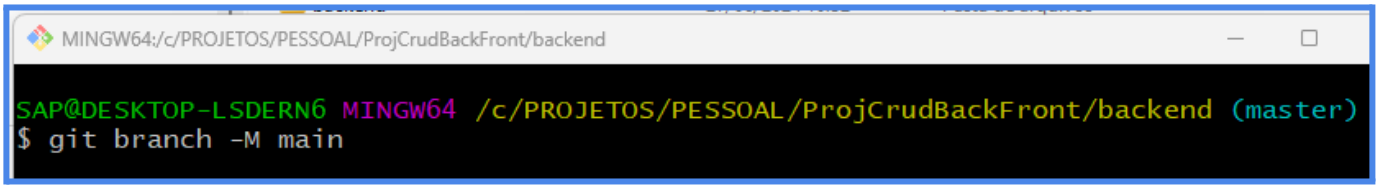
- git config --global user.name "auriceliof"



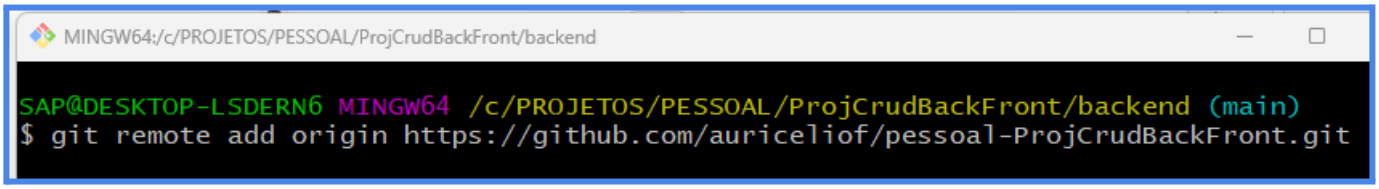
- git commit -m "Project backend created"



- git branch -M main

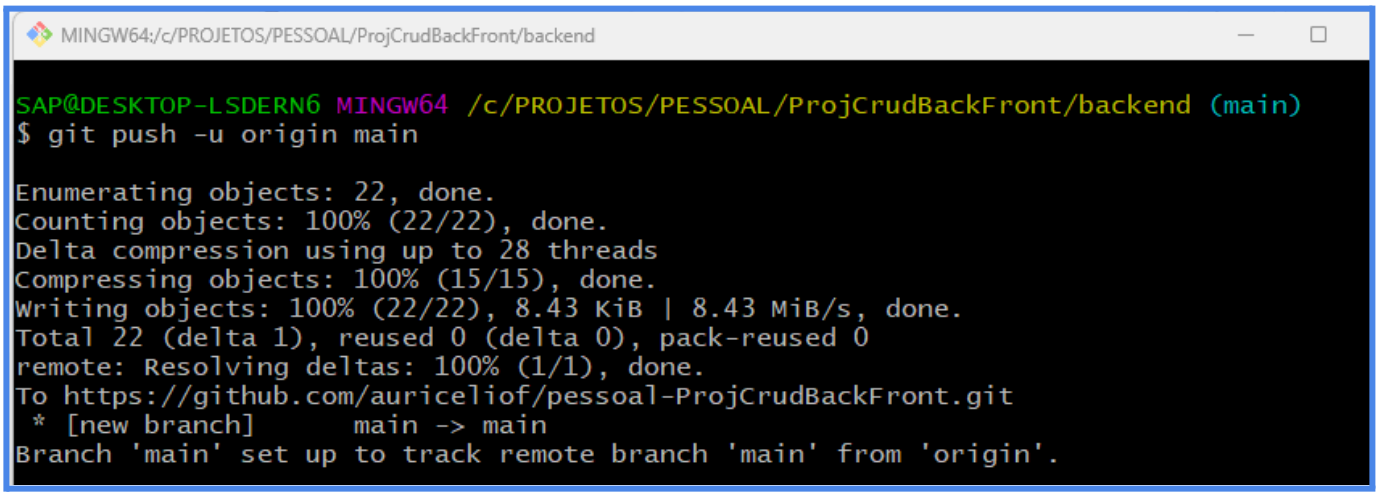


- git remote add origin https://github.com/auriceliof/pessoal-ProjCrudBackFront.git

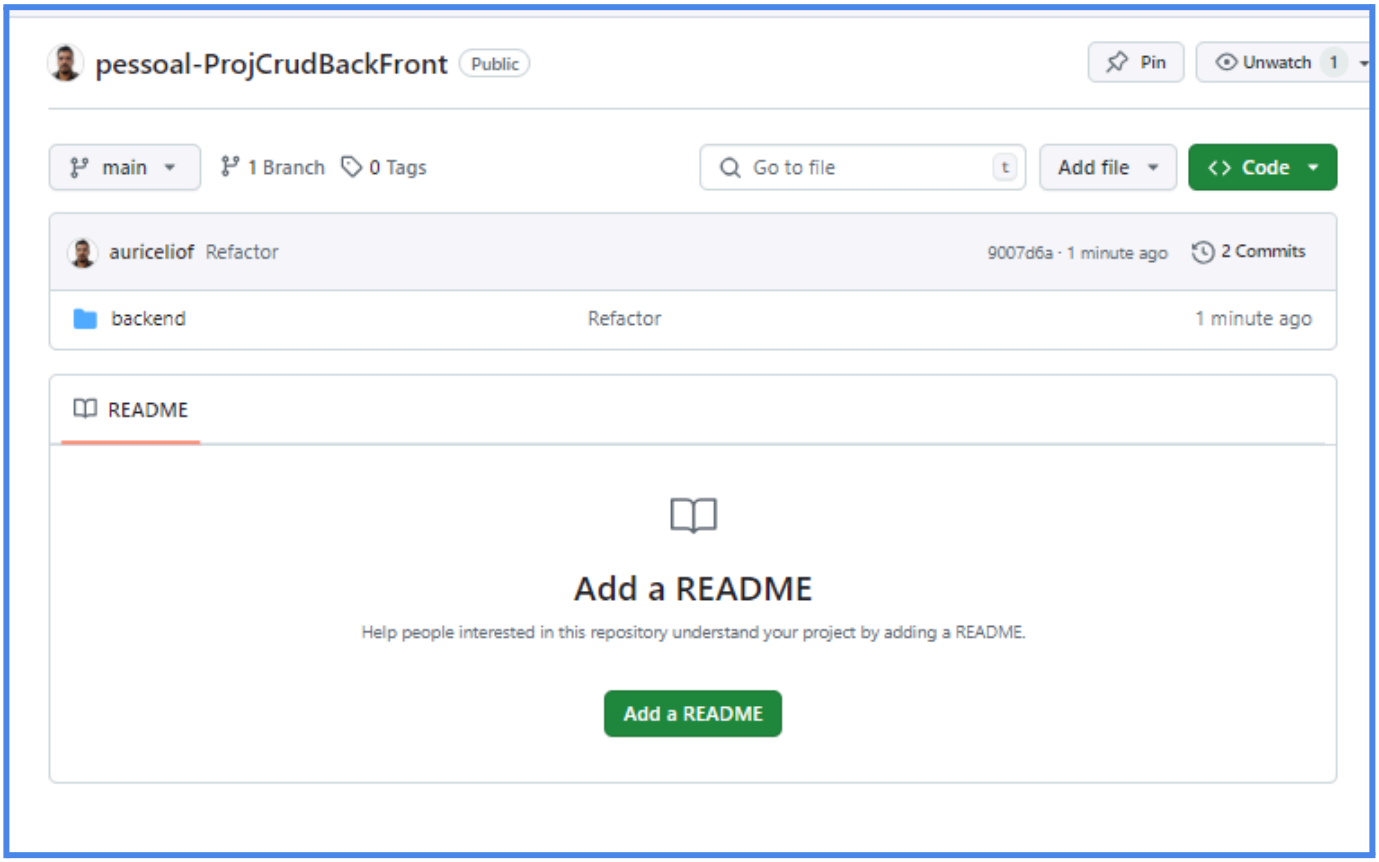


- git push -u origin main

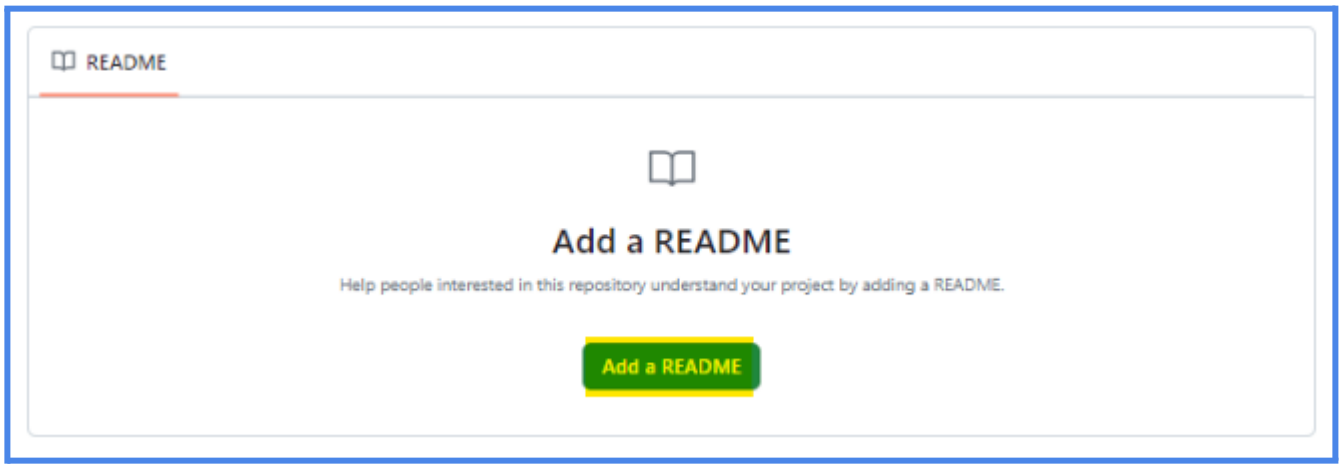
NOTA: Caso seja a primeira vez que sincronize no github no computador atual, irá pedir para realizar a autenticação no GitHub. Na próxima vez isso não irá aparecer.

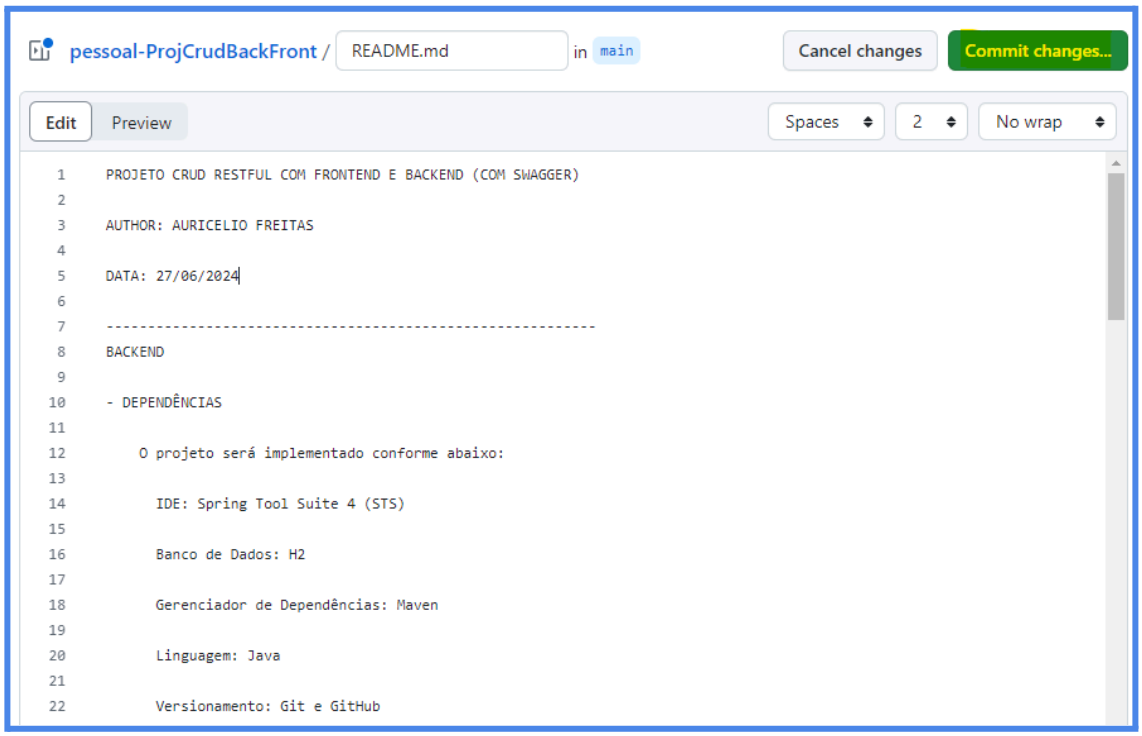


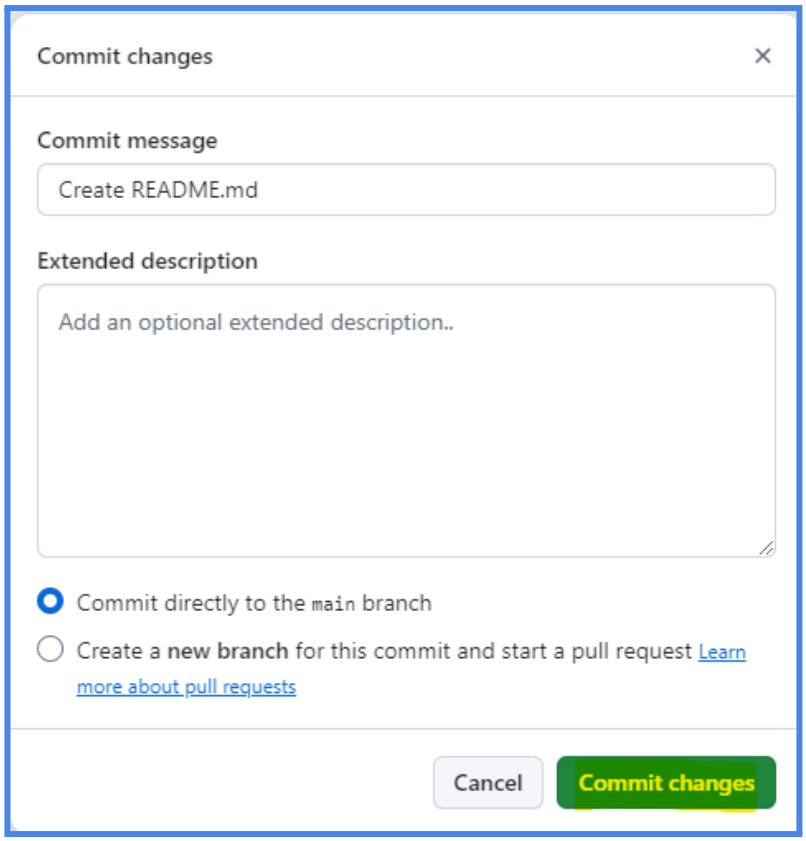
- Visualizar no GitHub (F5)

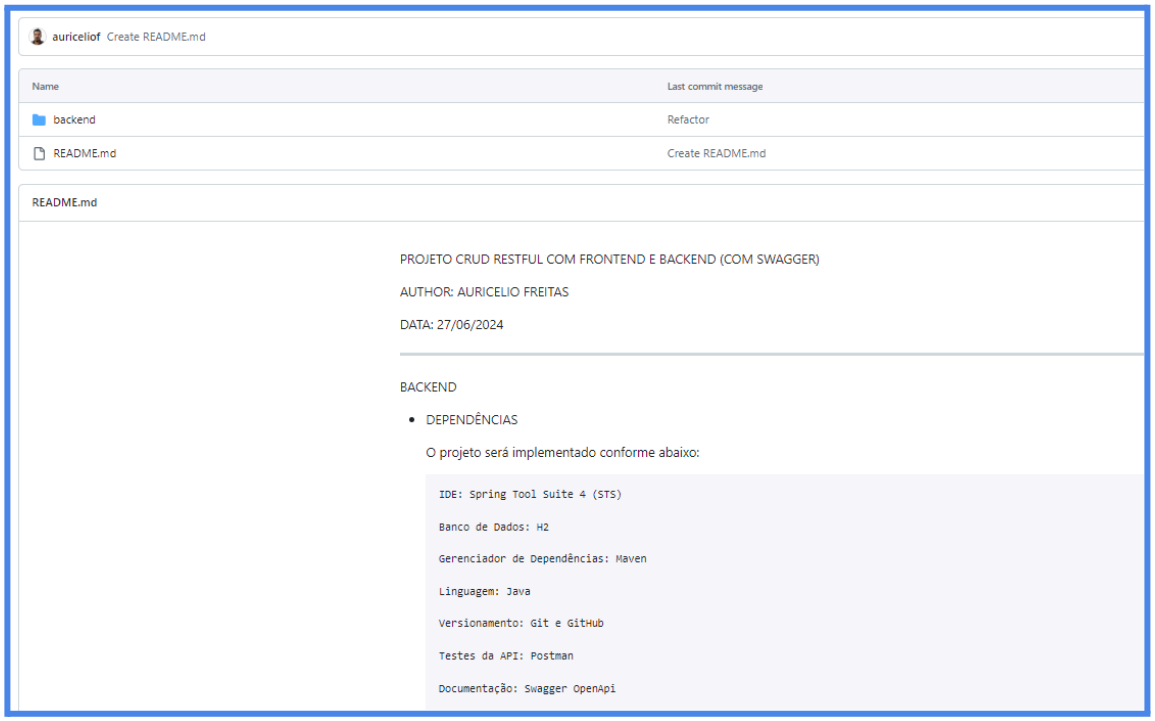


Adicionar o README ao projeto



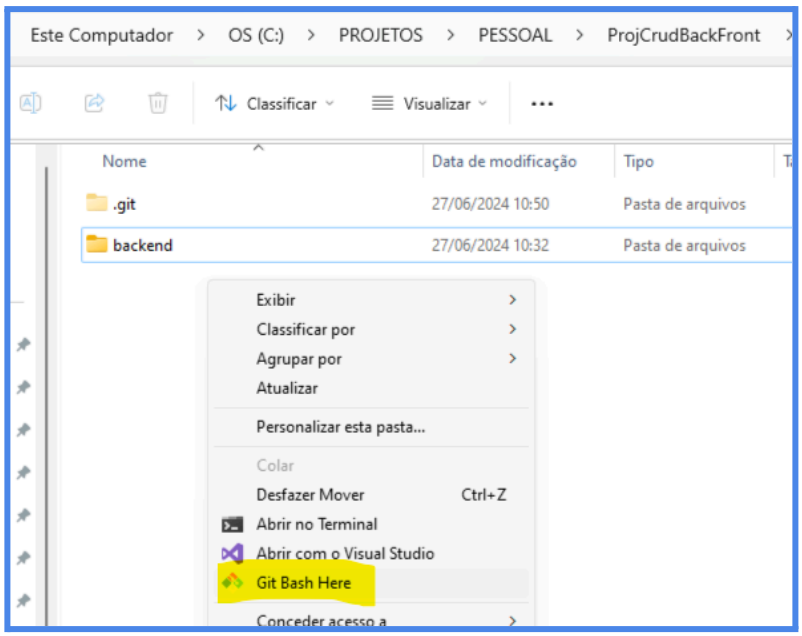

pessoal-ProjCrudBackFront / README.md in main
Cancel changes
Commit changes...
Edit
Preview
Spaces
2
No wrap
1 PROJETO CRUD RESTFUL COM FRONTEND E BACKEND (COM SWAGGER)
2
3 AUTHOR: AURICELIO FREITAS
4
5 DATA: 27/06/2024
6
7 ----------
8 BACKEND
9
10 - DEPENDÊNCIAS
11
12 O projeto será implementado conforme abaixo:
13
14 IDE: Spring Tool Suite 4 (STS)
15
16 Banco de Dados: H2
17
18 Gerenciador de Dependências: Maven
19
20 Linguagem: Java
21
22 Versionamento: Git e GitHub


Commit changes
Commit message
Create README.md
Extended description
Add an optional extended description..
Commit directly to the main branch
Create a new branch for this commit and start a pull request Learn more about pull requests
Cancel
Commit changes



Atualizar o projeto local

- **Botão direito no projeto e clicar em "Open Git bash here"**



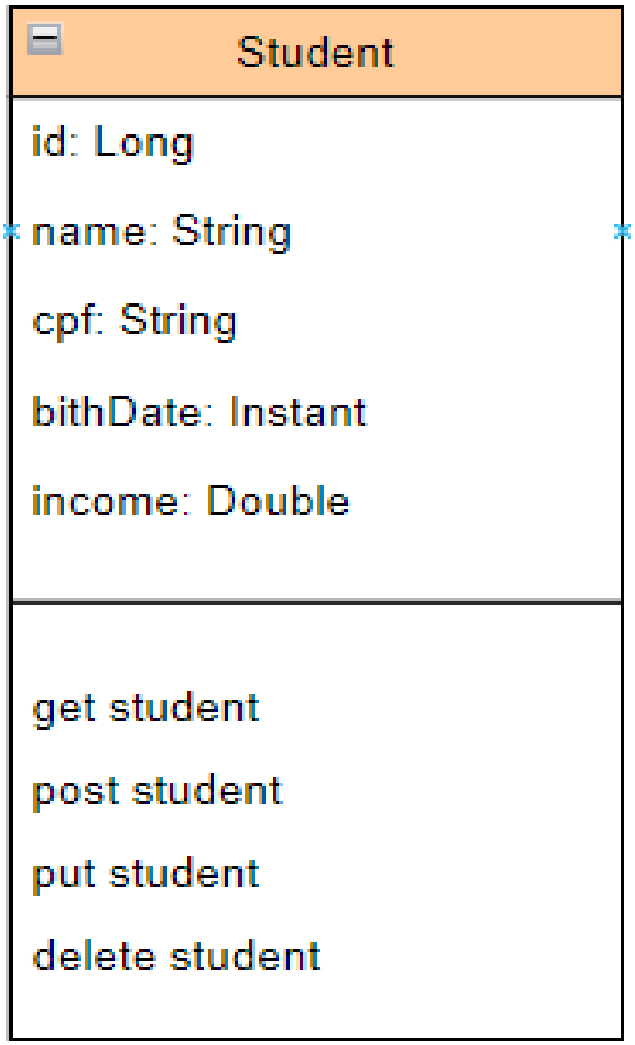
- git pull

```
MINGW64:/c/PROJETOS/PESSOAL/ProjCrudBackFront

SAP@DESKTOP-LSDERN6 MINGW64 /c/PROJETOS/PESSOAL/ProjCrudBackFront (main)
$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 2.10 KiB | 358.00 KiB/s, done.
From https://github.com/auriceliof/pessoal-ProjCrudBackFront
   9007d6a..37e69d1  main       -> origin/main
Updating 9007d6a..37e69d1
Fast-forward
 README.md | 144 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 1 file changed, 144 insertions(+)
 create mode 100644 README.md
```

INICIAR O DESENVOLVIMENTO DO PROJETO

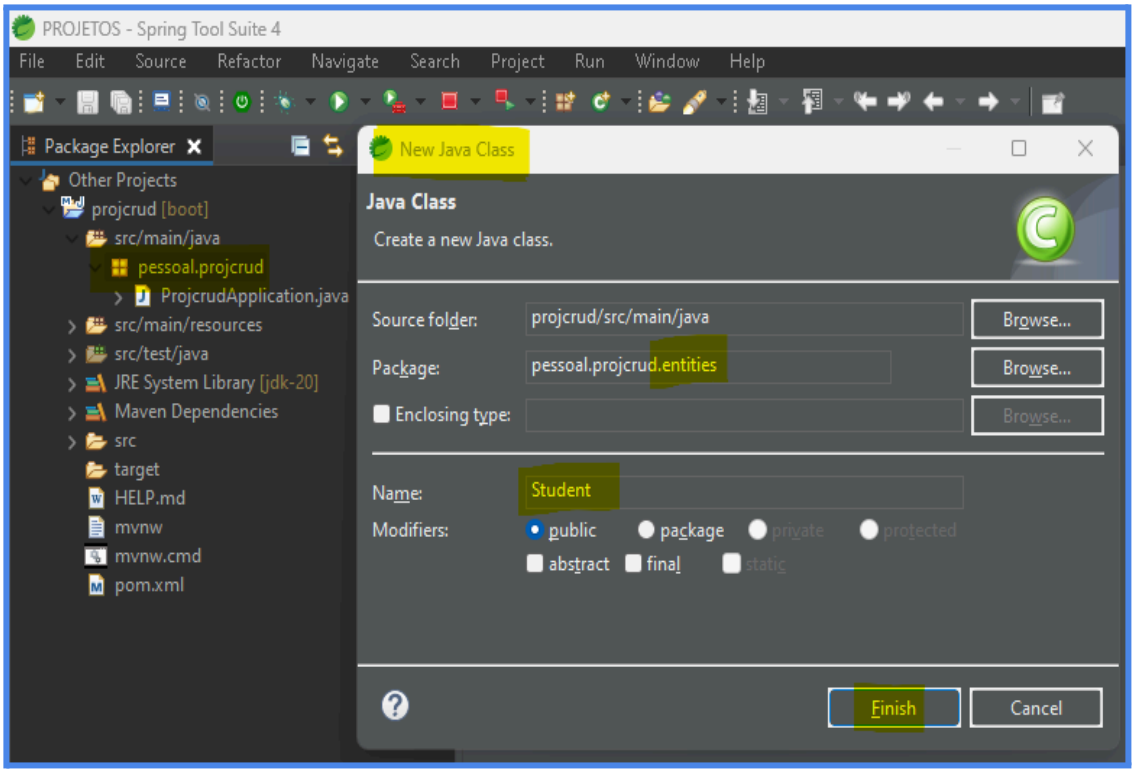
STUDENT_CLASSE

DIAGRAMA DE CLASSE

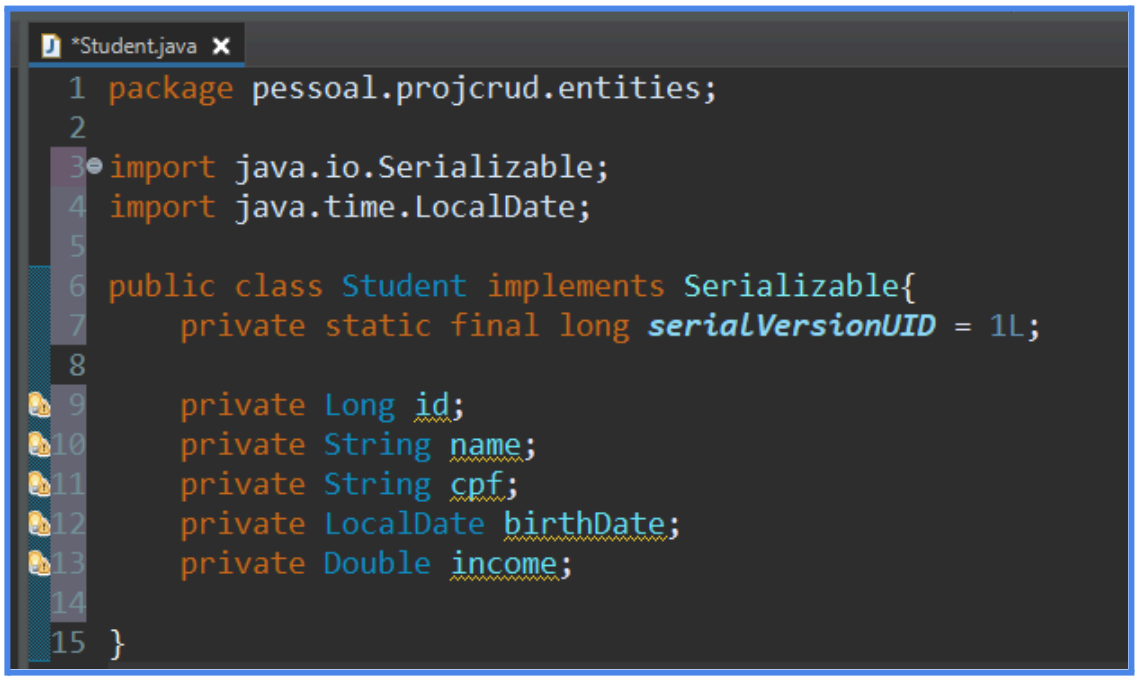


IMPLEMENTAR A ESTRUTURA E A CLASSE STUDENT NO STS

Criar a estrutura e classe



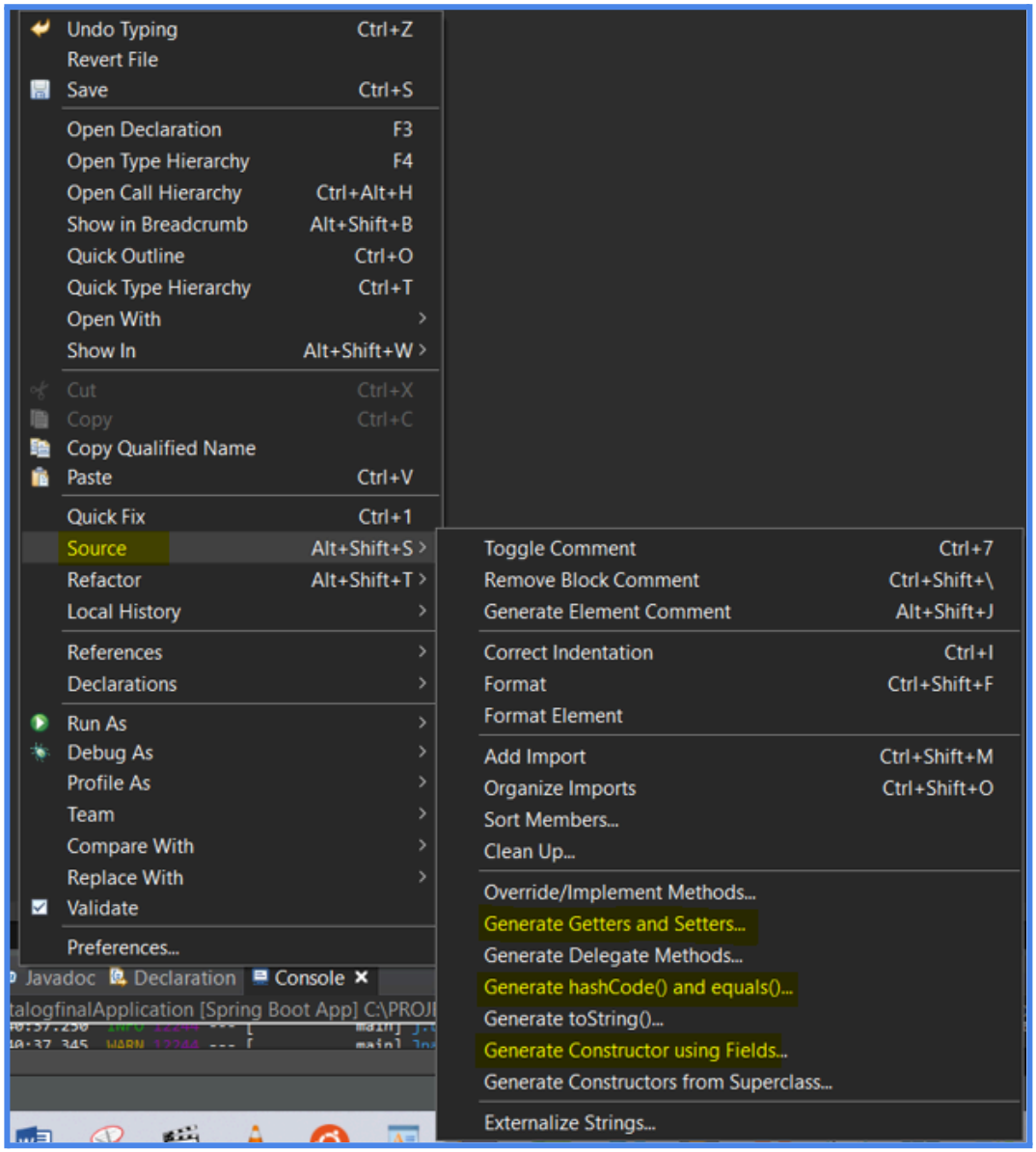
Definir o Serializable e os atributos da Classe



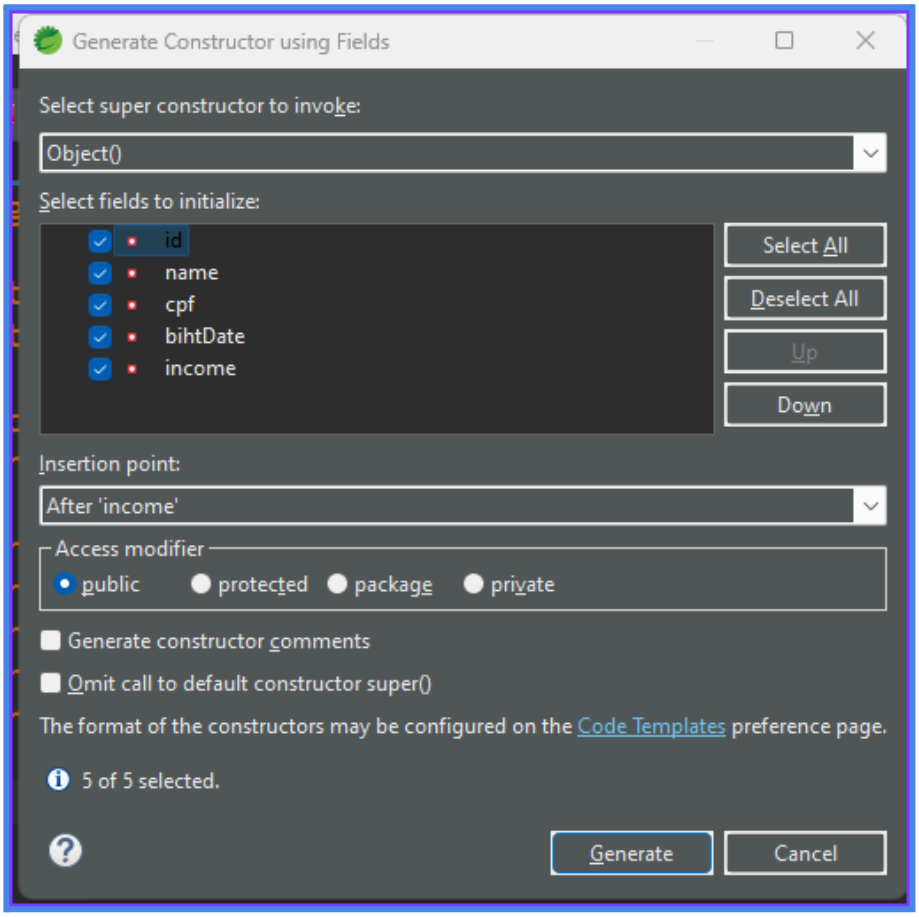
DICA:

Para criar os construtores, getters and setters e o hashcode, proceder conforme segue:

- *Clicar com o botão direito aonde quer inserir*
  - *Source*
    - *"Escolher o método que deseja implementar"*
      - *Selecionar os itens*
        - *Clicar em: Generate*



Criar o construtor

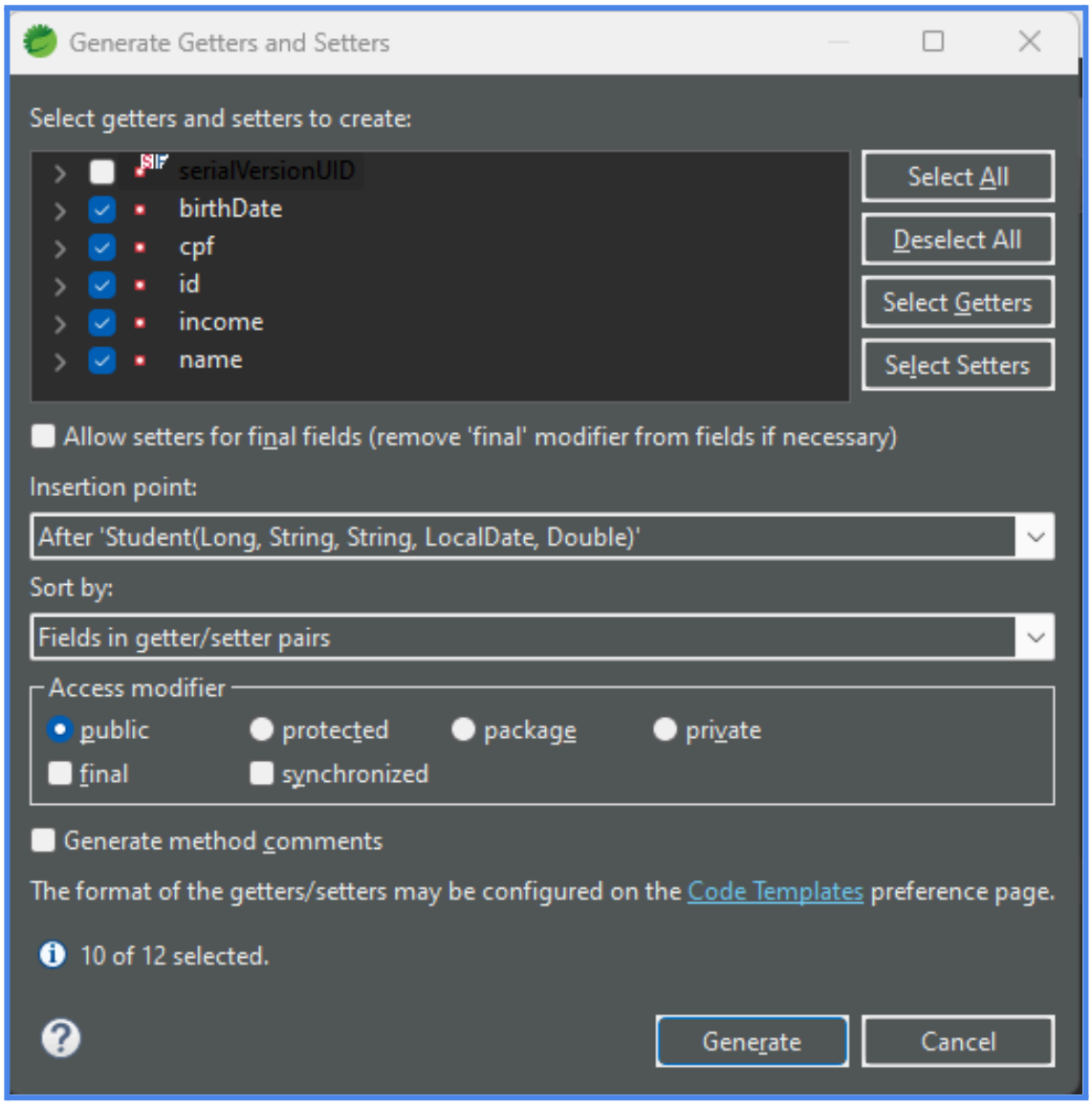


*Student.java

```
14
15      public Student() {
16
17      }
18
19      public Student(Long id, String name, String cpf, LocalDate birthDate, Double income) {
20          super();
21          this.id = id;
22          this.name = name;
23          this.cpf = cpf;
24          this.birthDate = birthDate;
25          this.income = income;
26      }
27  }
```

NOTA: Por melhores práticas, também criaremos um construtor vazio.

Criar os Getters and Setters

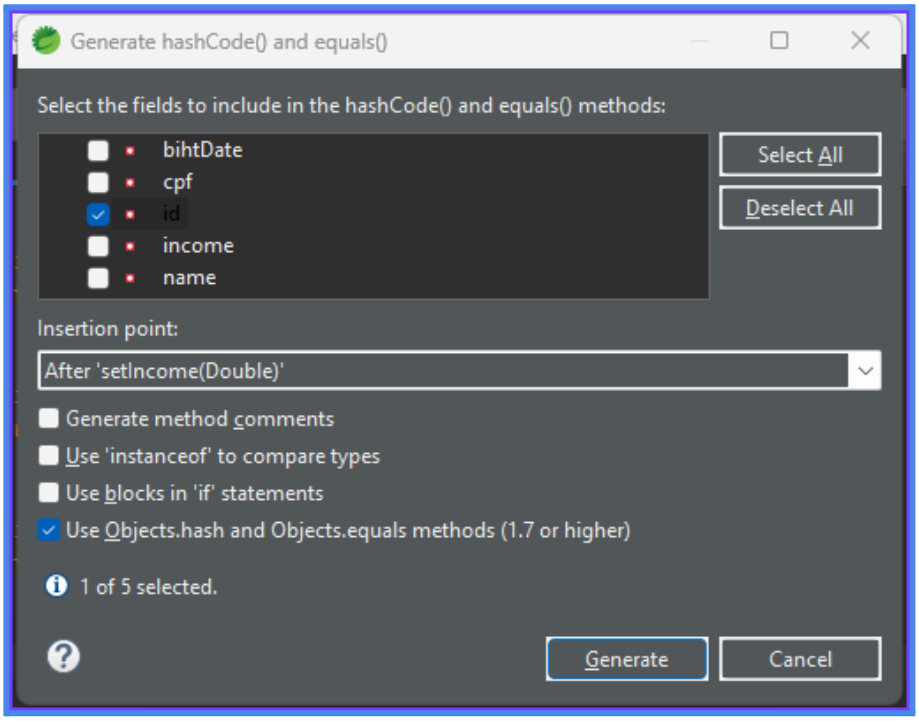


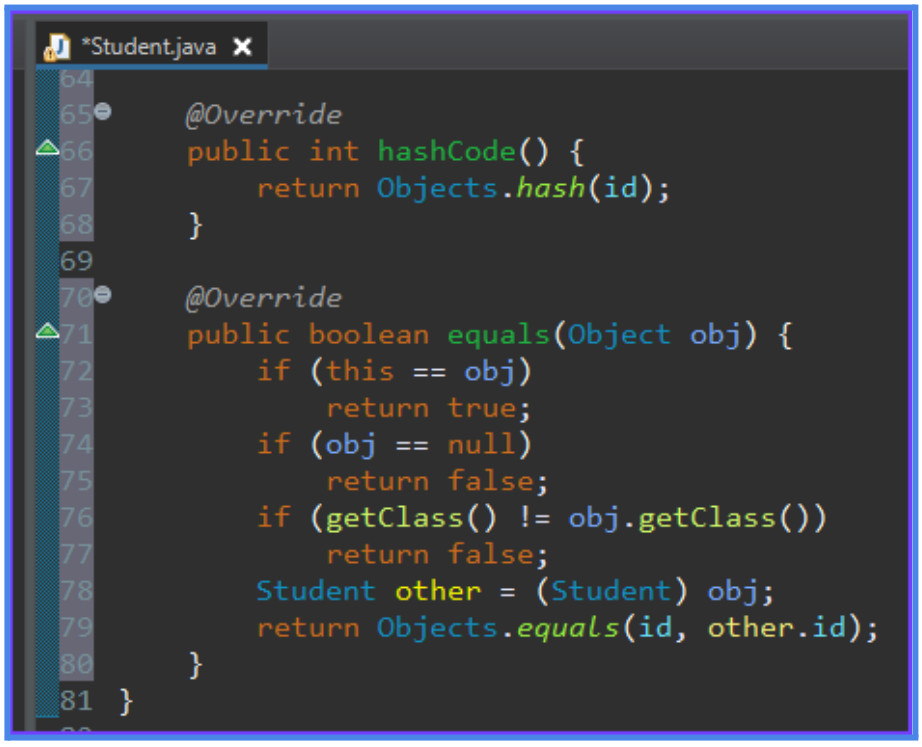
*Student.java

```
    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 LocalDate getBirthDate() {
        return birthDate;
    }

    public void setBirthDate(LocalDate birthDate) {
        this.birthDate = birthDate;
    }

    public Double getIncome() {
        return income;
    }

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

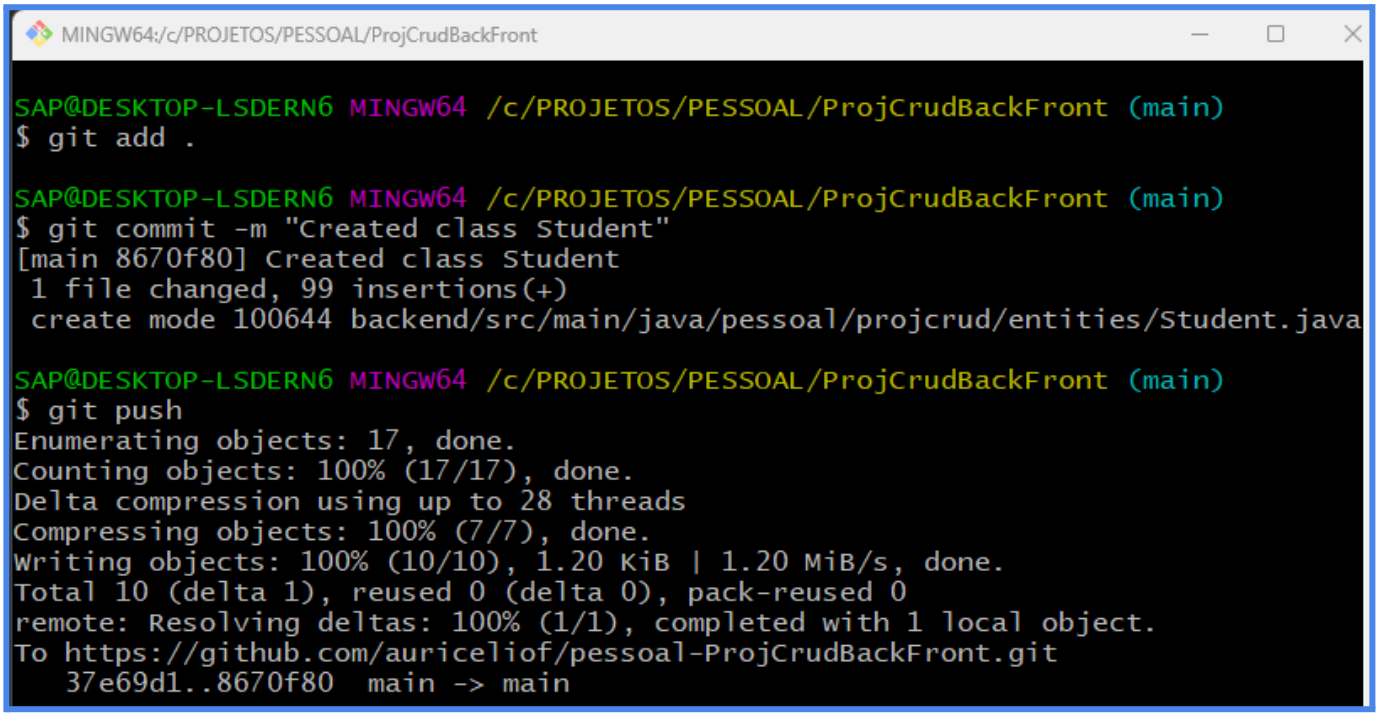
Criar o hashCode and equals

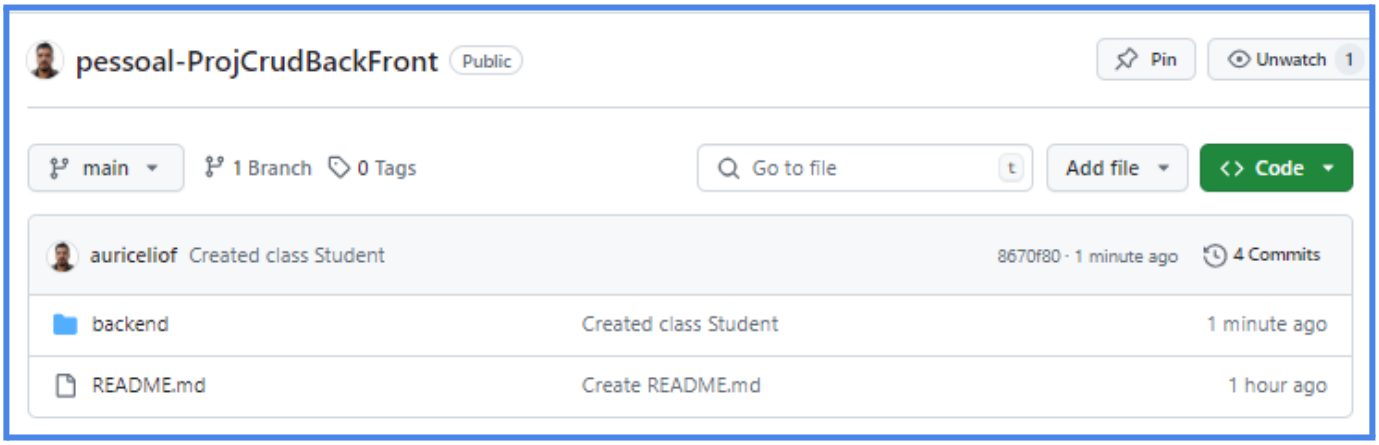




Github-2

- “Git bash here” no diretório do projeto
  - git add backend
  - git commit -m “Created class Student”
  - git push

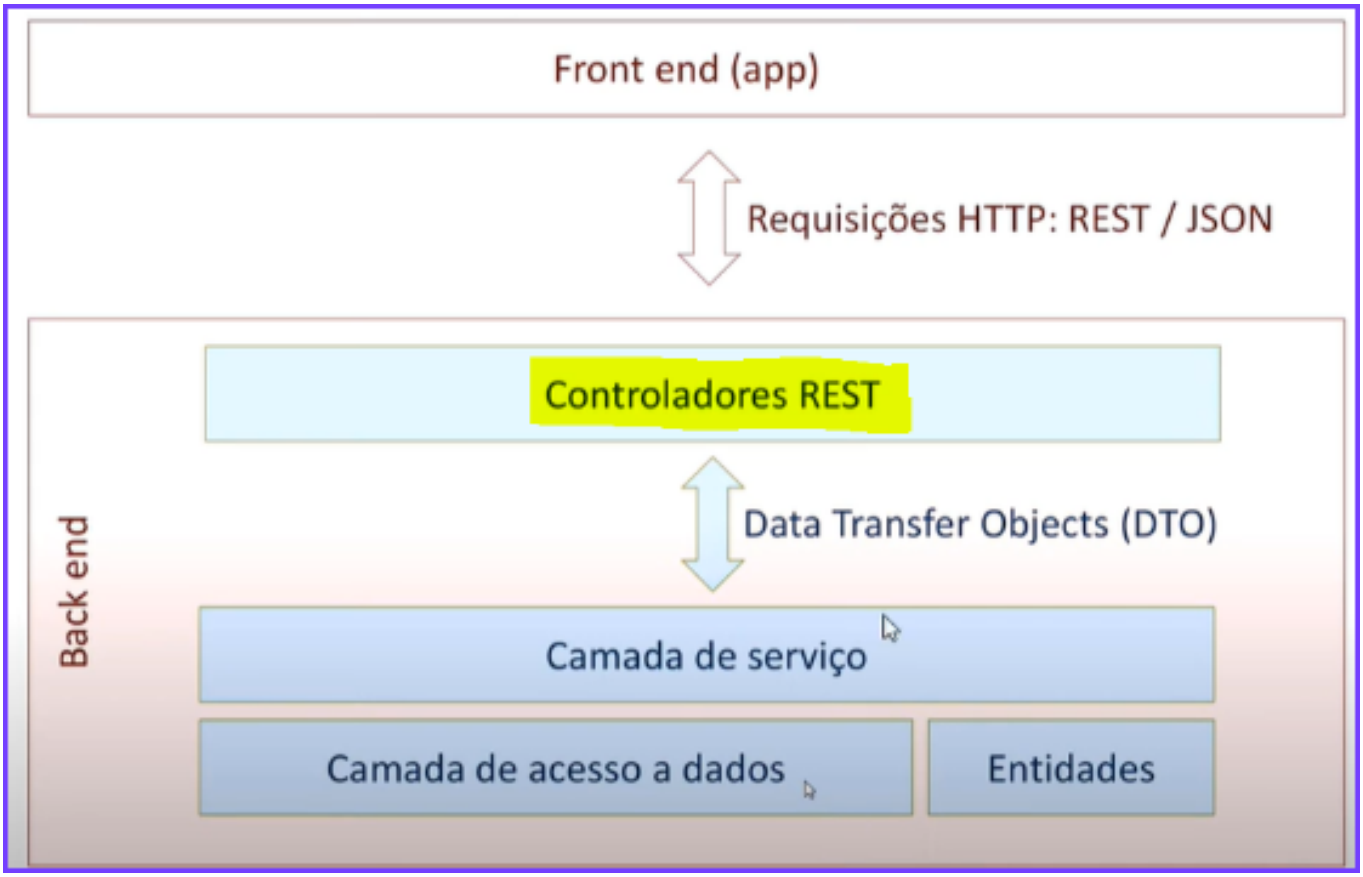




STUDENT_CONTROLLER

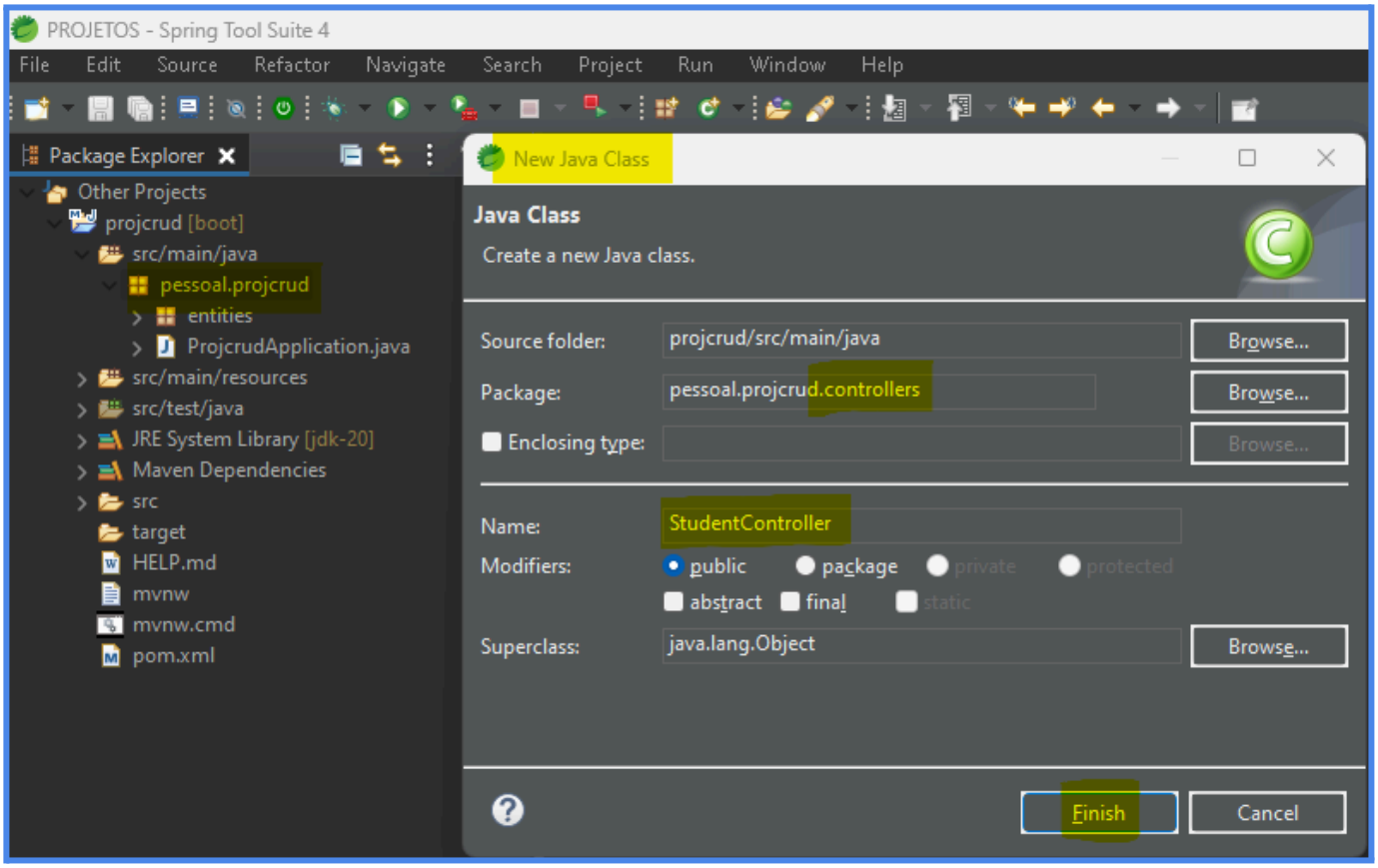
- O controlador é quem gerencia as requisições, podendo ser chamado de Controller ou Resource. É onde implantamos nossos endpoints.

CONCEITUAL



IMPLEMENTAR A ESTRUTURA

Criar a estrutura e o recurso StudentController



Implementar as Notações Rest

StudentController.java

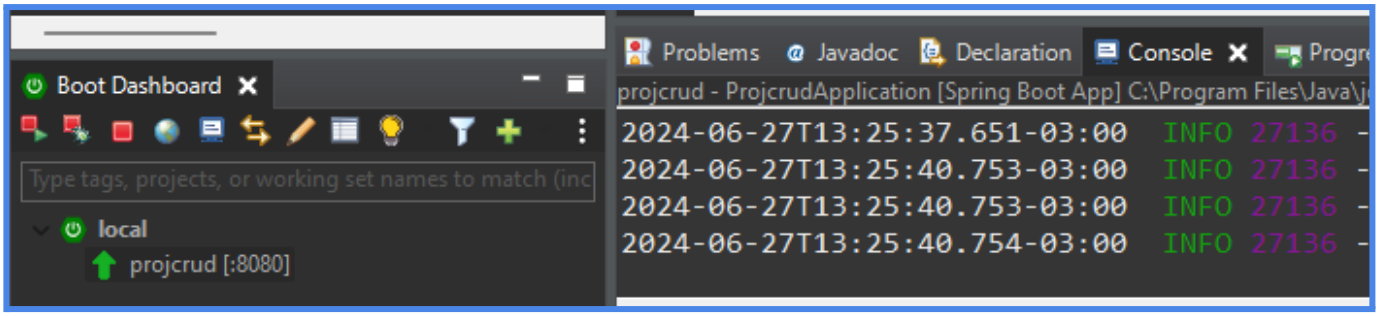
```java
package pessoal.projcrud.controllers;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping(value = "/students")
public class StudentController {

}
```

Criar o Endpoint findAll para teste

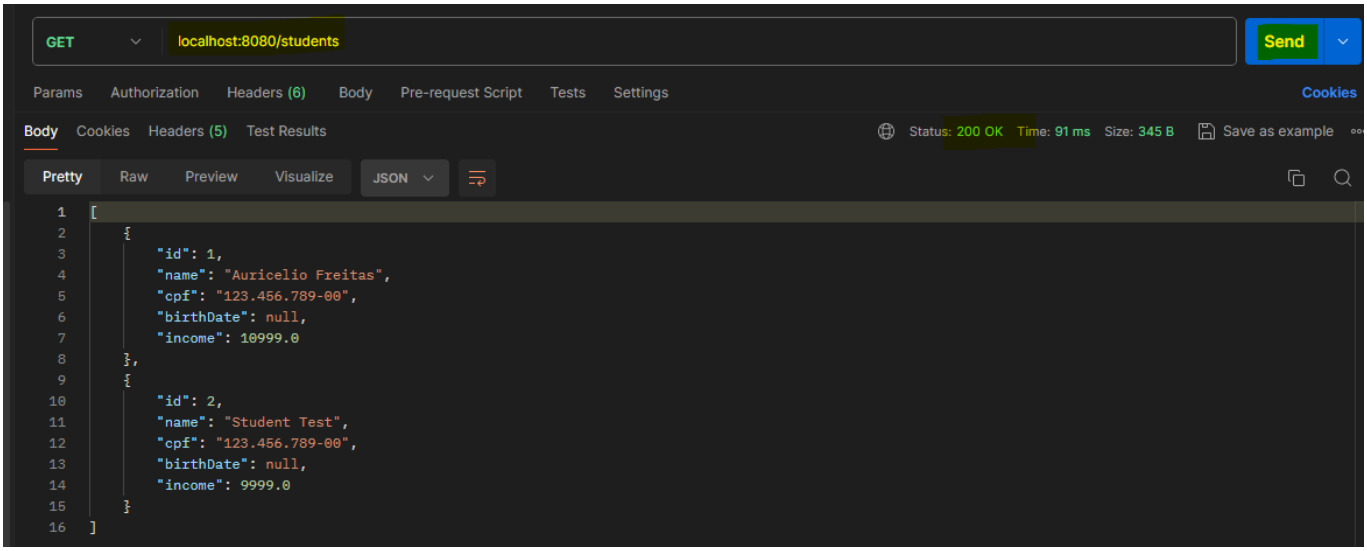
StudentController.java

```java
package pessoal.projcrud.controllers;

import java.util.ArrayList;
import java.util.List;

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 pessoal.projcrud.entities.Student;

@RestController
@RequestMapping(value = "/students")
public class StudentController {

    @GetMapping
    public  ResponseEntity<List<Student>> findAll() {
        List<Student> list = new ArrayList<>();

            list.add(new Student(1L, "Auricelio Freitas", "123.456.789-00", null, 10999.0));
            list.add(new Student(2L, "Student Test", "123.456.789-00", null, 9999.0));

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

Rodar o projeto

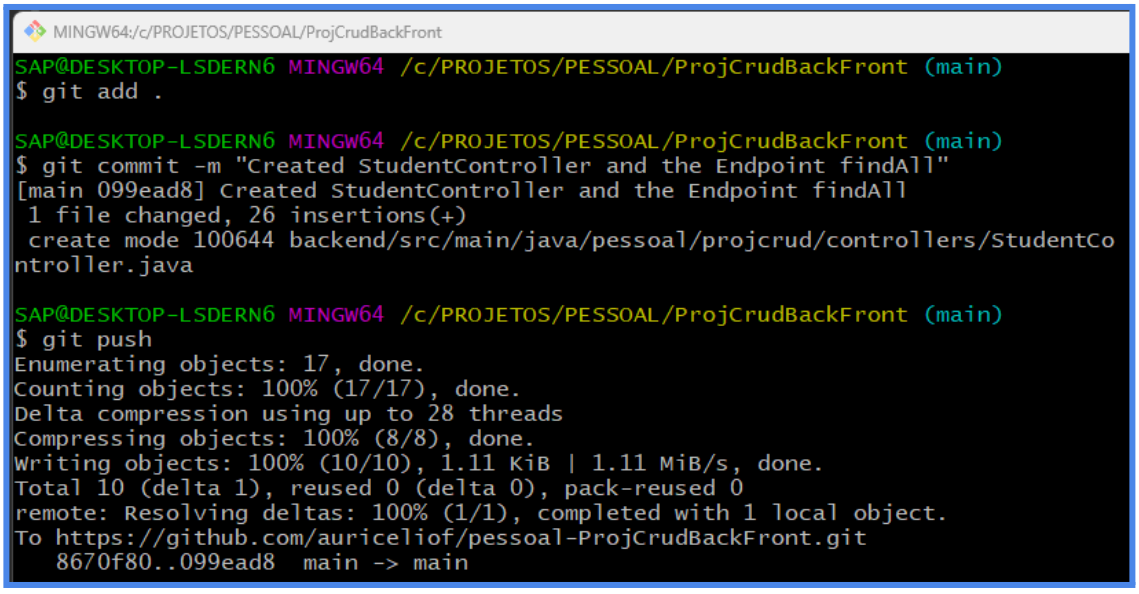


Testar com o Postman



Github-3

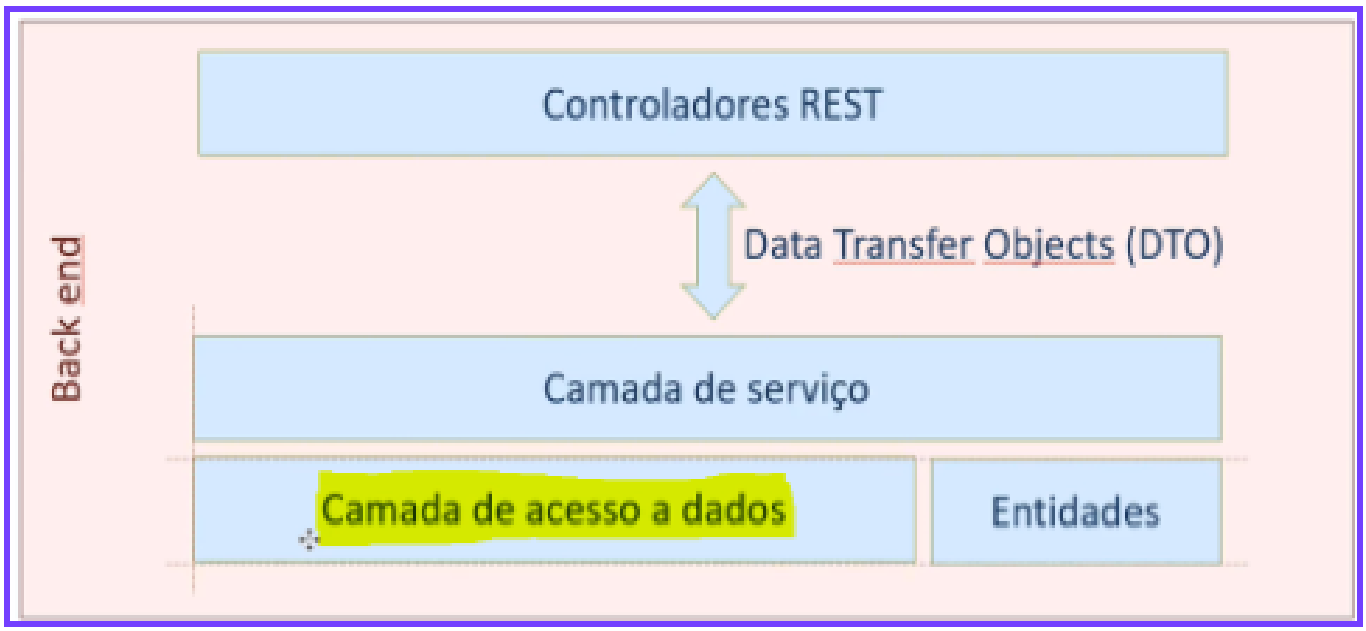
- “Git bash here” no diretório do projeto
  - git add .
  - git commit -m “Created StudentController and the Endpoint findAll”
  - git push



STUDENT_REPOSITORY

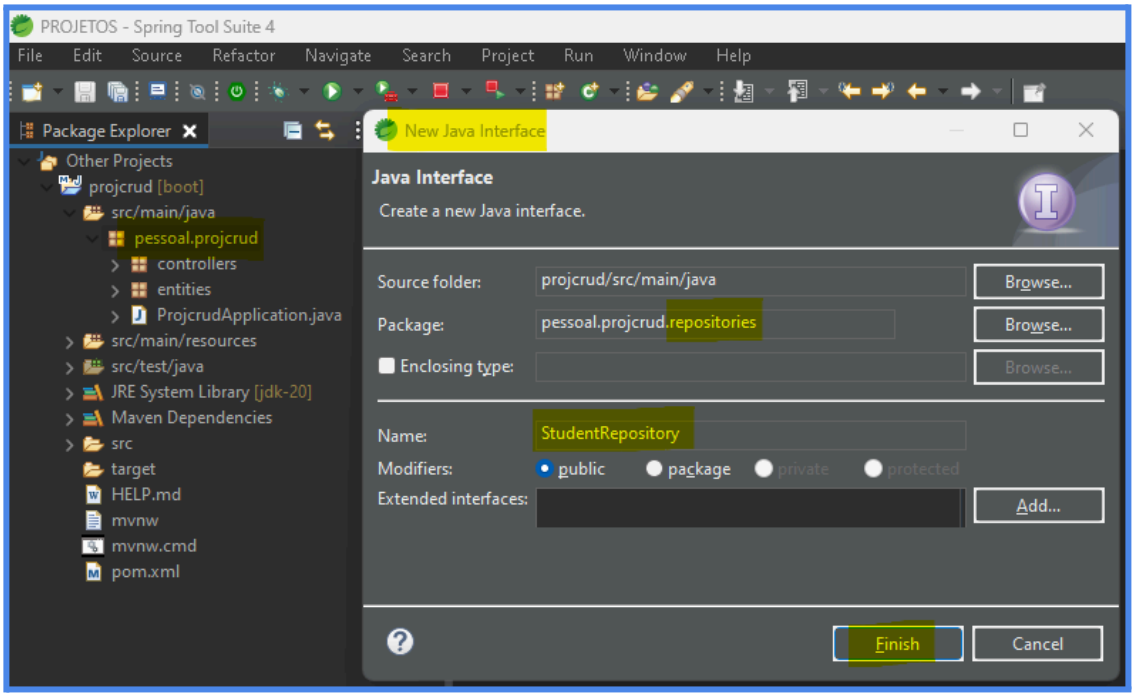
- Iremos implementar a “Camada de acesso a dados”, chamada de Repositories. É a camada responsável pelo acesso ao banco.

CONCEITUAL

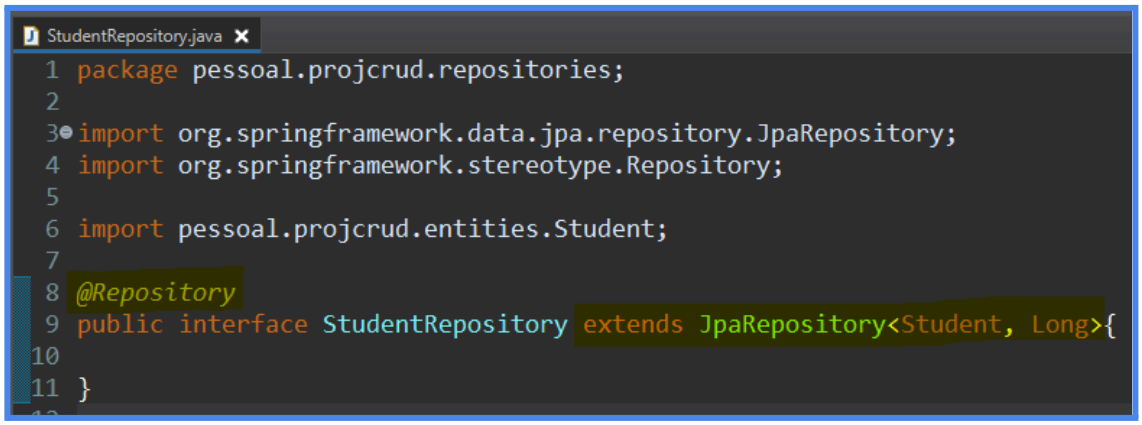


IMPLEMENTAR A ESTRUTURA E O STUDENT_REPOSITORY

Criar a estrutura e a interface de acesso ao banco



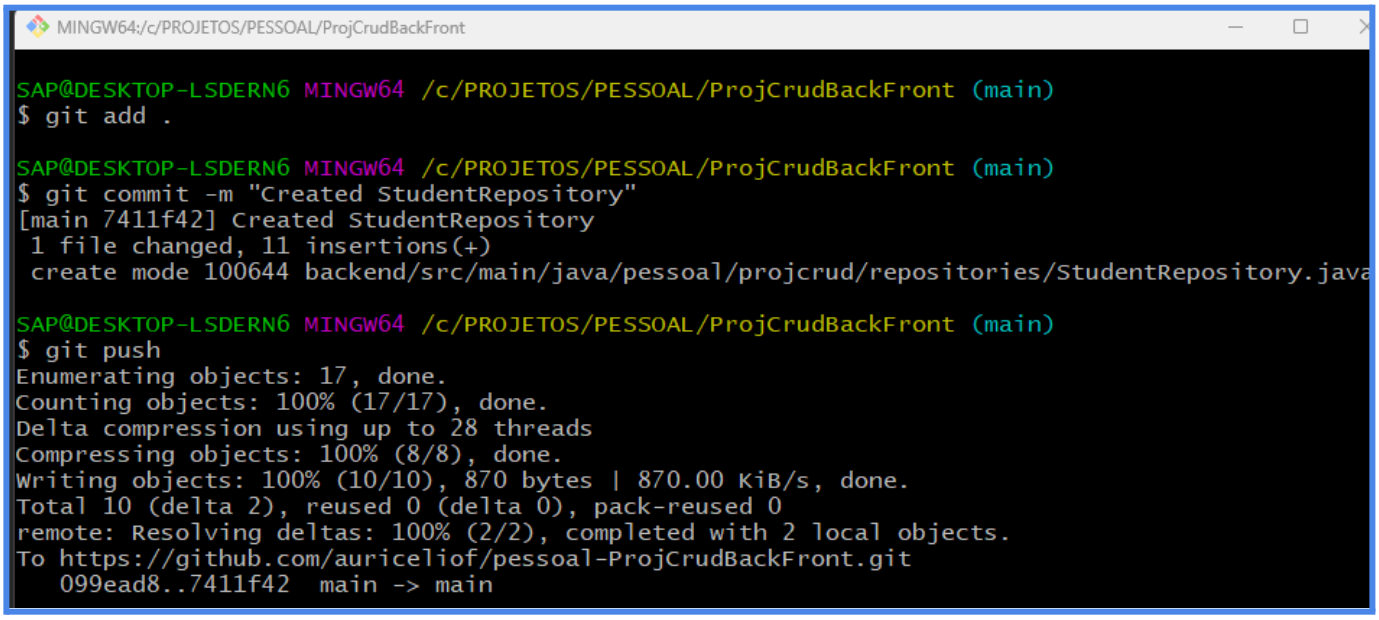
Implementar a notação e estender a JPA



NOTA: Apenas ao estender o JpaRepository, o spring já nos fornece vários métodos de manipulação aos dados com o banco.

Github-4

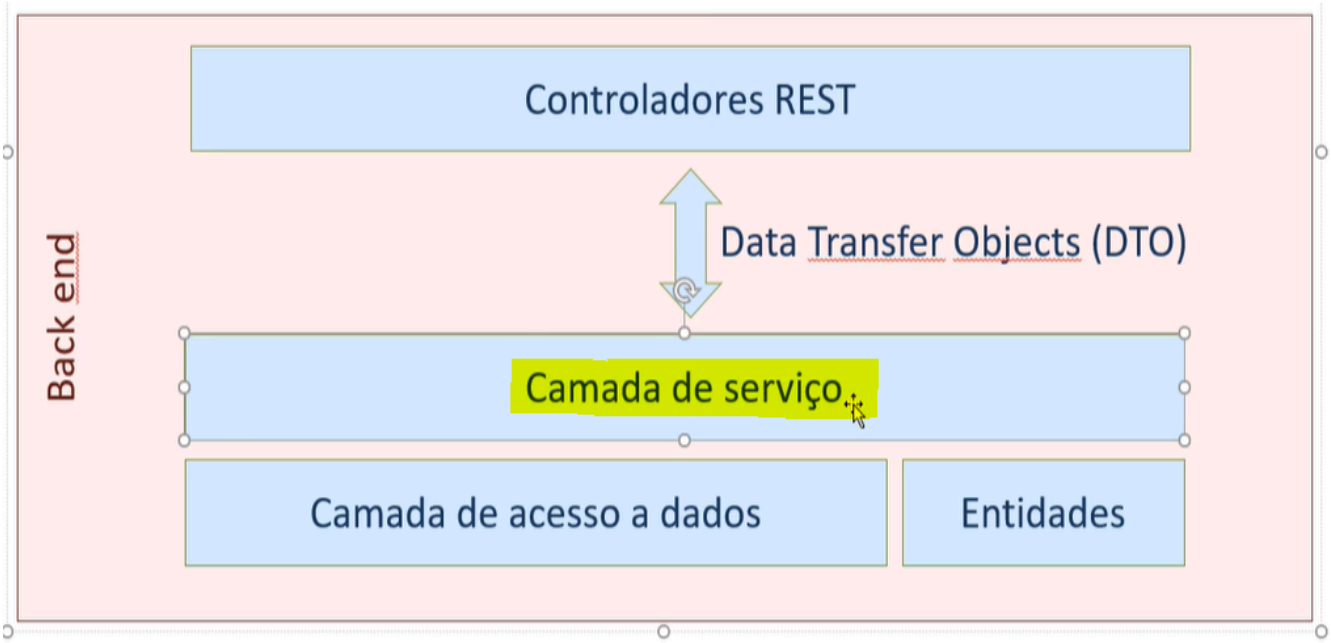
- “Git bash here” no diretório do projeto
  - git add backend
  - git commit -m “Created StudentRepository”
  - git push



STUDENT_SERVICE

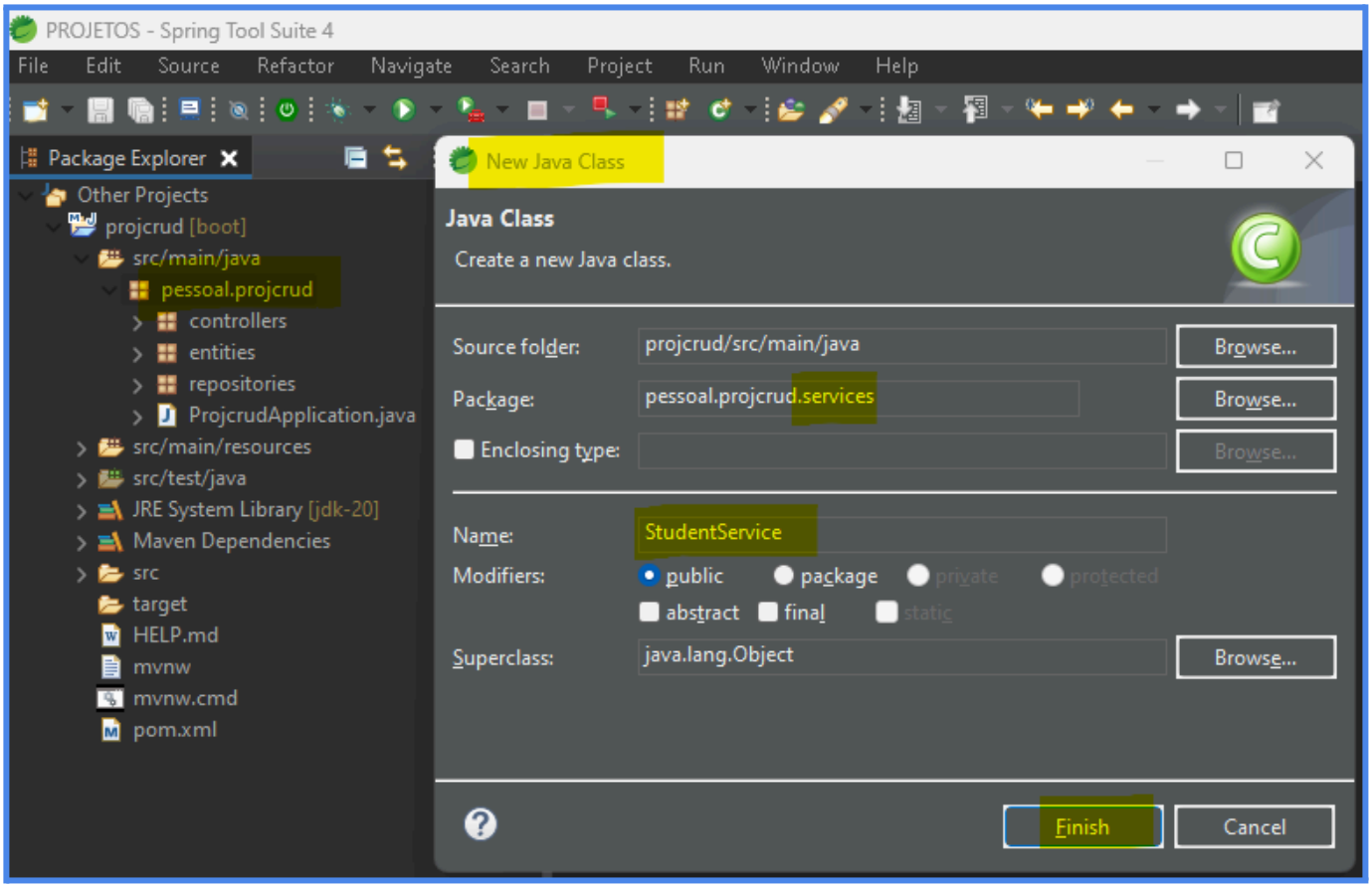
- Iremos implementar a “Camada de serviço”, onde concentramos as regras de negócio e toda a lógica do projeto.

CONCEITUAL



IMPLEMENTAR A ESTRUTURA E O STUDENT_SERVICE

Criar a estrutura e a classe de serviço



Implementar a lógica para o endpoint findAll

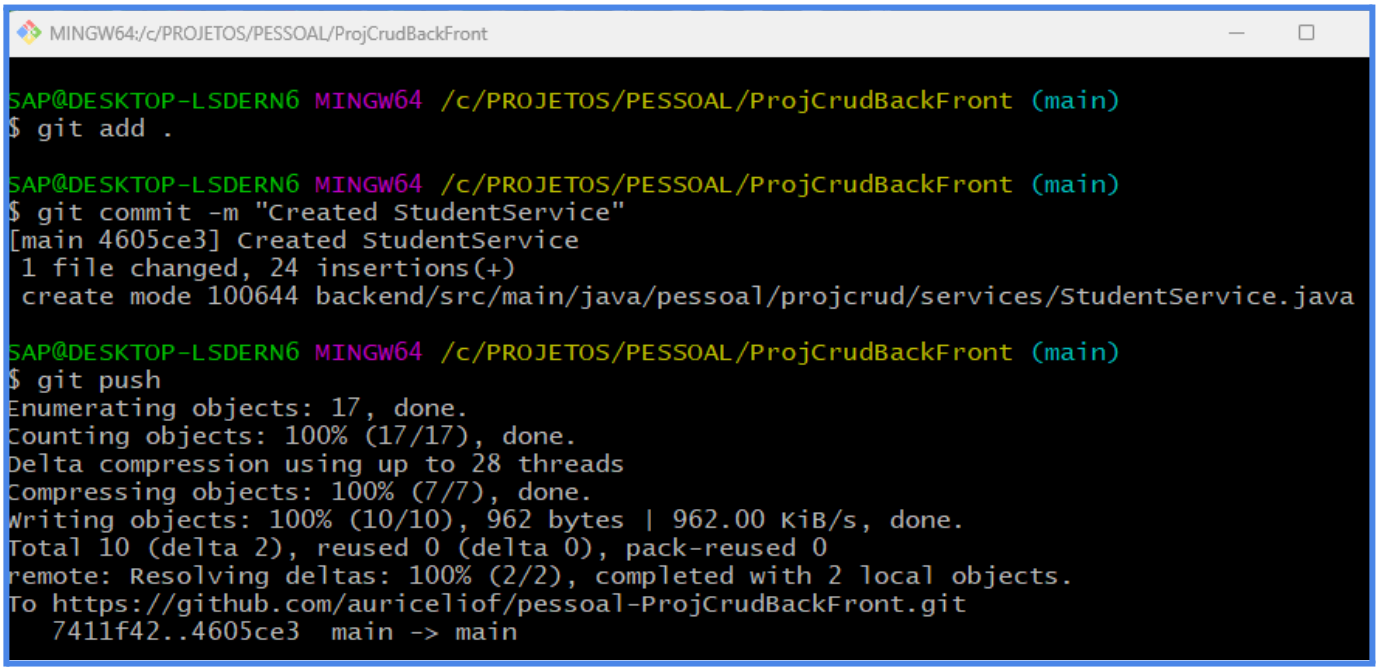
StudentService.java

```java
package pessoal.projcrud.services;

import java.util.List;

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

import pessoal.projcrud.entities.Student;
import pessoal.projcrud.repositories.StudentRepository;

@Service
public class StudentService {

    @Autowired
    private StudentRepository repository;

    @Transactional(readOnly = true)
    public List<Student> findAll() {

        return repository.findAll();
    }
}
```

NOTA: O "@Transactional", garante a integridade da transação de um método junto ao banco. No caso de pesquisa, utilizar o "readOnly" para evitar o locking desnecessário ao banco.

Github-5

- “Git bash here” no diretório do projeto
  - git add backend
  - git commit -m “Created StudentService”
  - git push



INTEGRAÇÃO COM O BANCO

AJUSTAR AS CAMADAS

Implementar o StudentController

StudentController.java

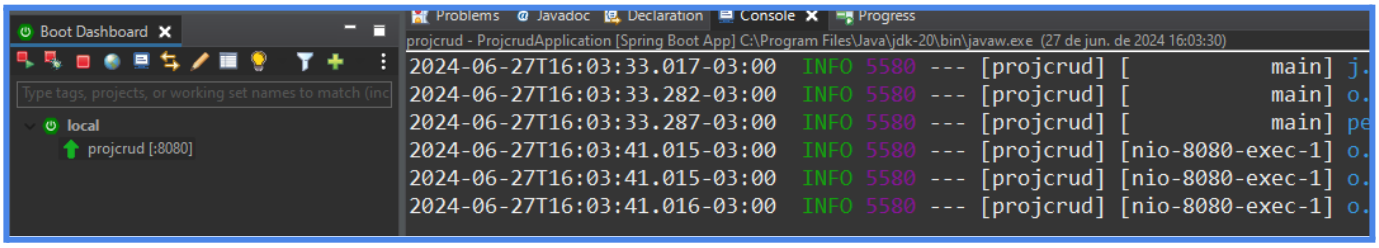
```java
package pessoal.projcrud.controllers;

import java.util.List;

@RestController
@RequestMapping(value = "/students")
public class StudentController {

    @Autowired
    private StudentService service;

    @GetMapping
    public  ResponseEntity<List<Student>> findAll() {

        List<Student> list = service.findAll();

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

Implementar a classe Student

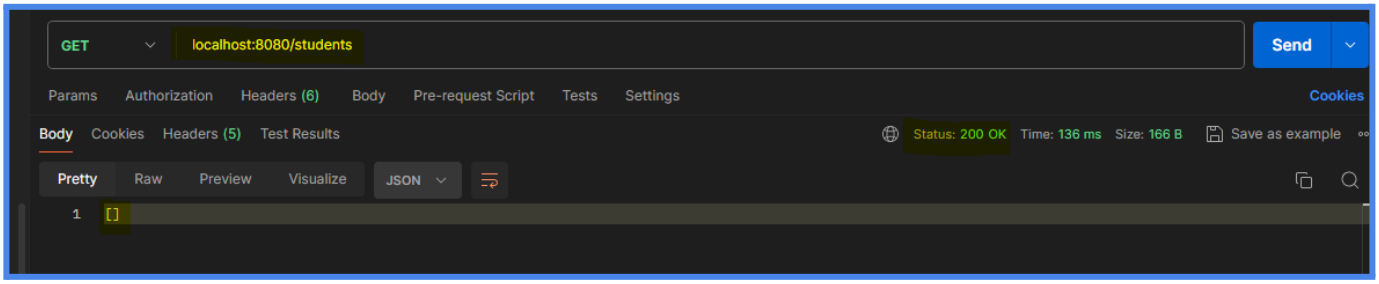
```java
@Entity
@Table(name = "tb_student")
public class Student implements Serializable{
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String cpf;

    @Column(columnDefinition = "TIMESTAMP WITH TIME ZONE")
    private LocalDate birthDate;
    private Double income;
```

NOTA:**. Ao instanciar as notações em questão, sempre escolher o pacote referente à especificação **"jakarta.persistence".

Rodar o projeto



Testar com o Postman



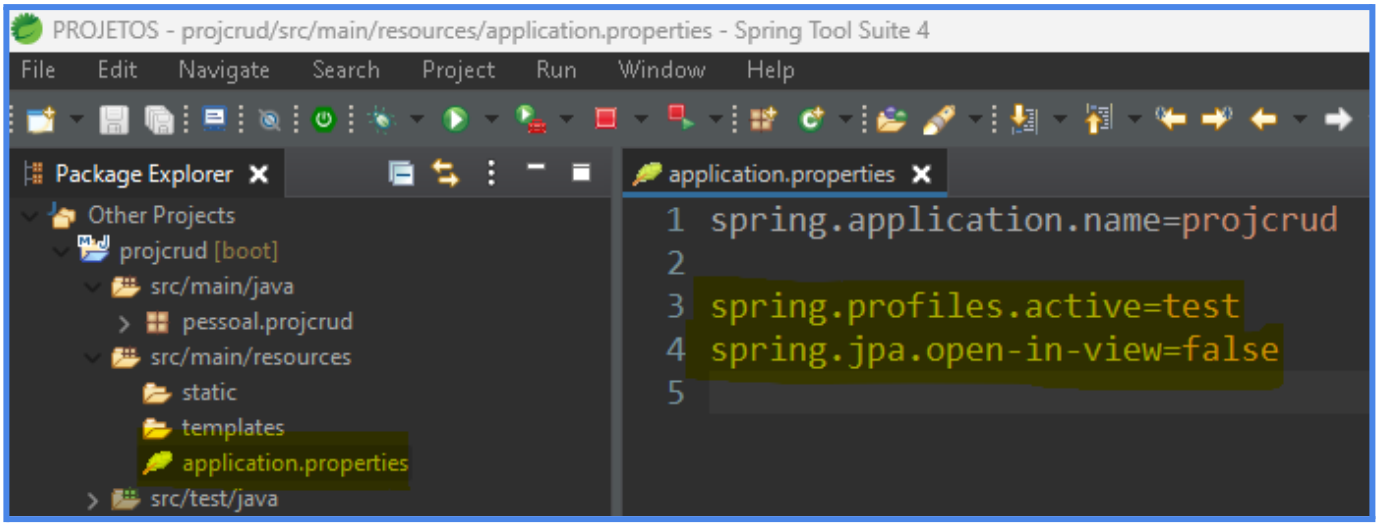
NOTA: Como não configuramos o banco ainda, nos é retornada, apenas, uma lista vazia.

BANCO H2

Configurar o perfil de teste no application.properties

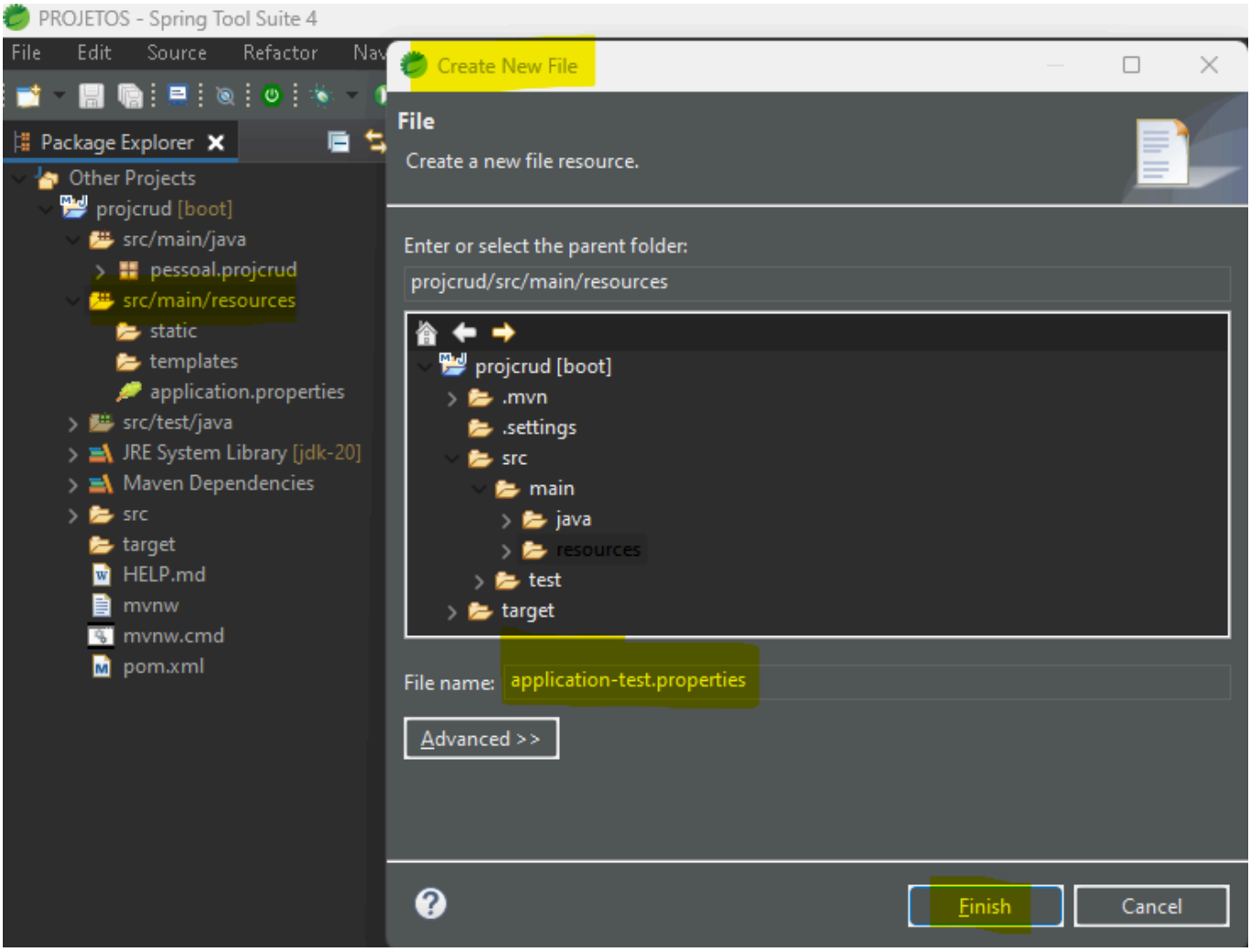
Perfil de teste

```
spring.profiles.active=test
spring.jpa.open-in-view=false
```



***NOTA**: O "spring.jpa.open-in-view", faz com que as transações ao banco com JPA sejam encerradas na camada de Serviço. Não passando para a camada de Controle.*

Criar o arquivo de configuração application-test.properties



Implementar o application-test.properties

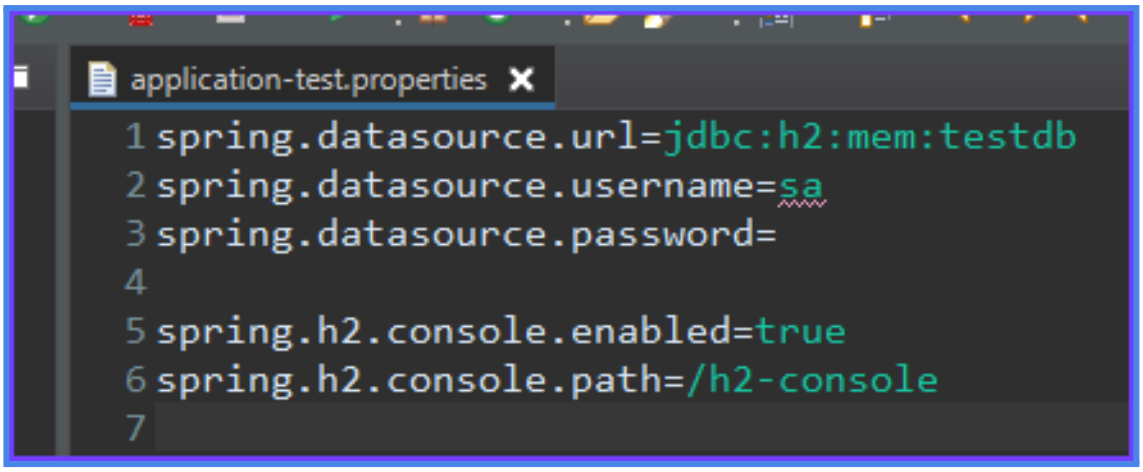
spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.username=sa

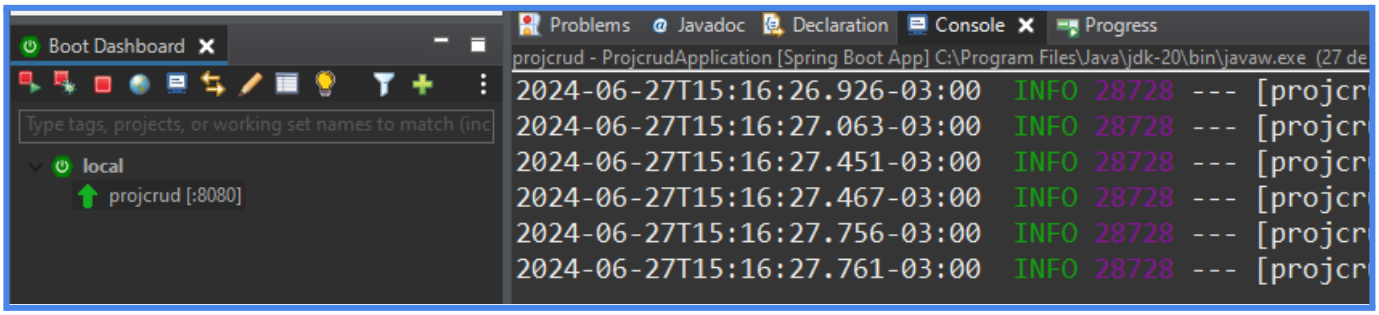
spring.datasource.password=

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console

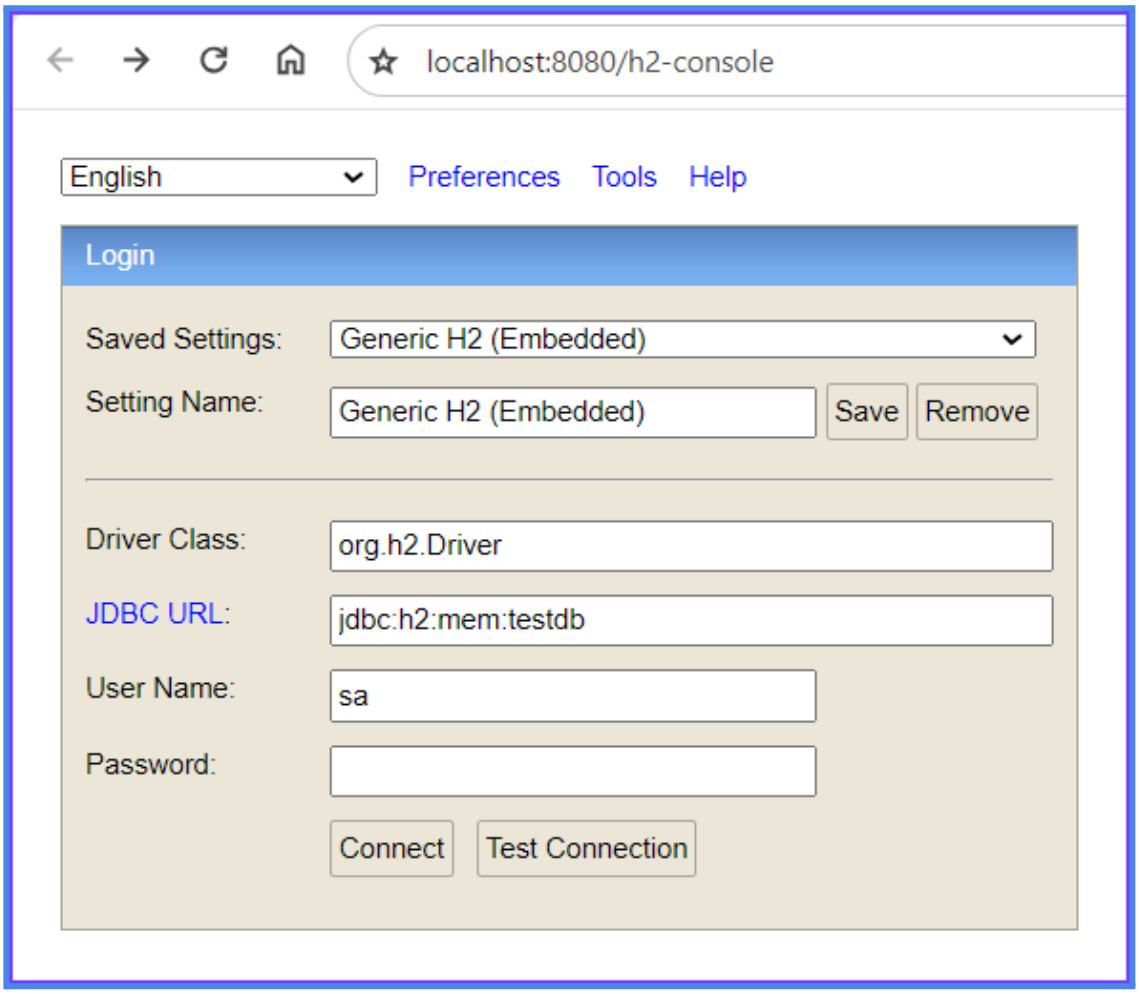


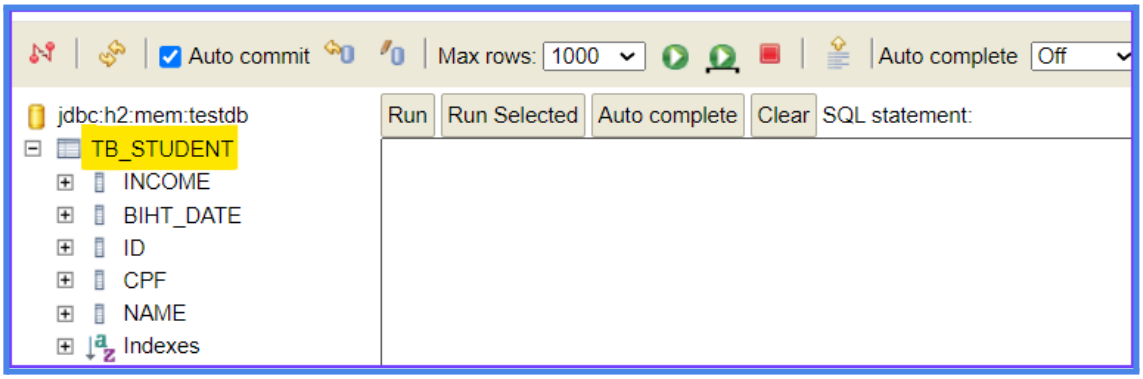
Rodar o projeto



Acessar o banco H2, via web

- http://localhost:8080/h2-console





Teste de inserção

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

```
INSERT INTO TB_STUDENT (NAME) VALUES ('Auricelio')
```

```
INSERT INTO TB_STUDENT (NAME) VALUES ('Auricelio');
Update count: 1
(3 ms)
```

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

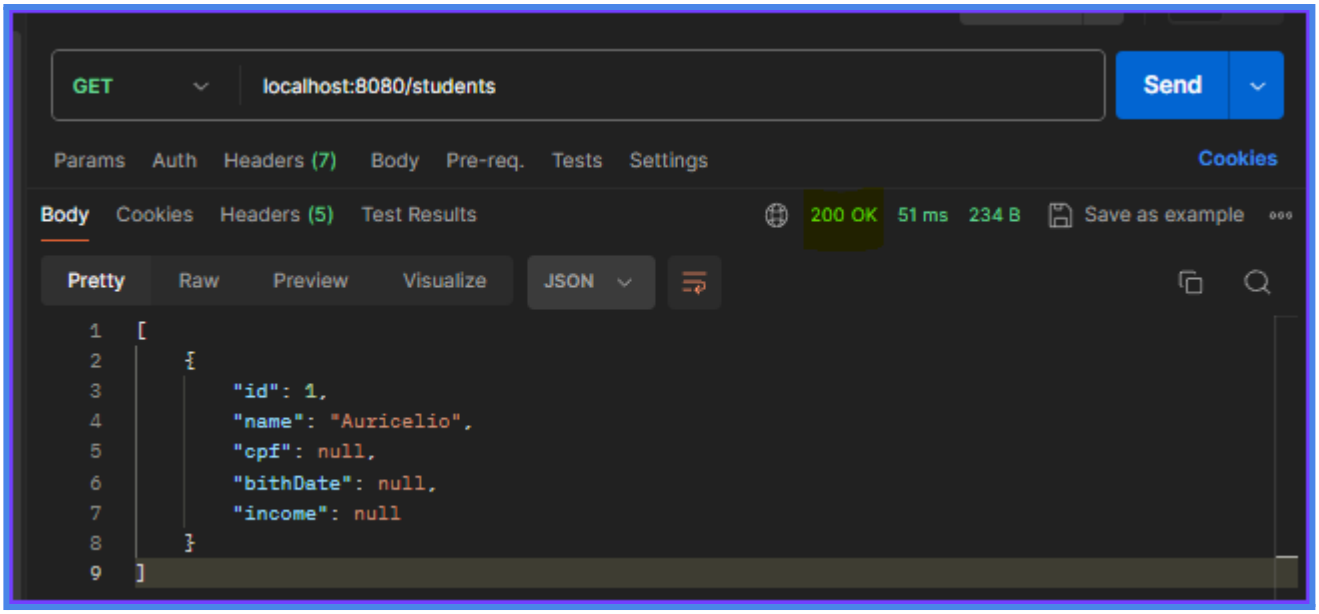
```
SELECT * FROM TB_STUDENT
```

```
SELECT * FROM TB_STUDENT;
```

INCOME	BITH_DATE	ID	CPF	NAME
null	*null*	1	*null*	Auricelio

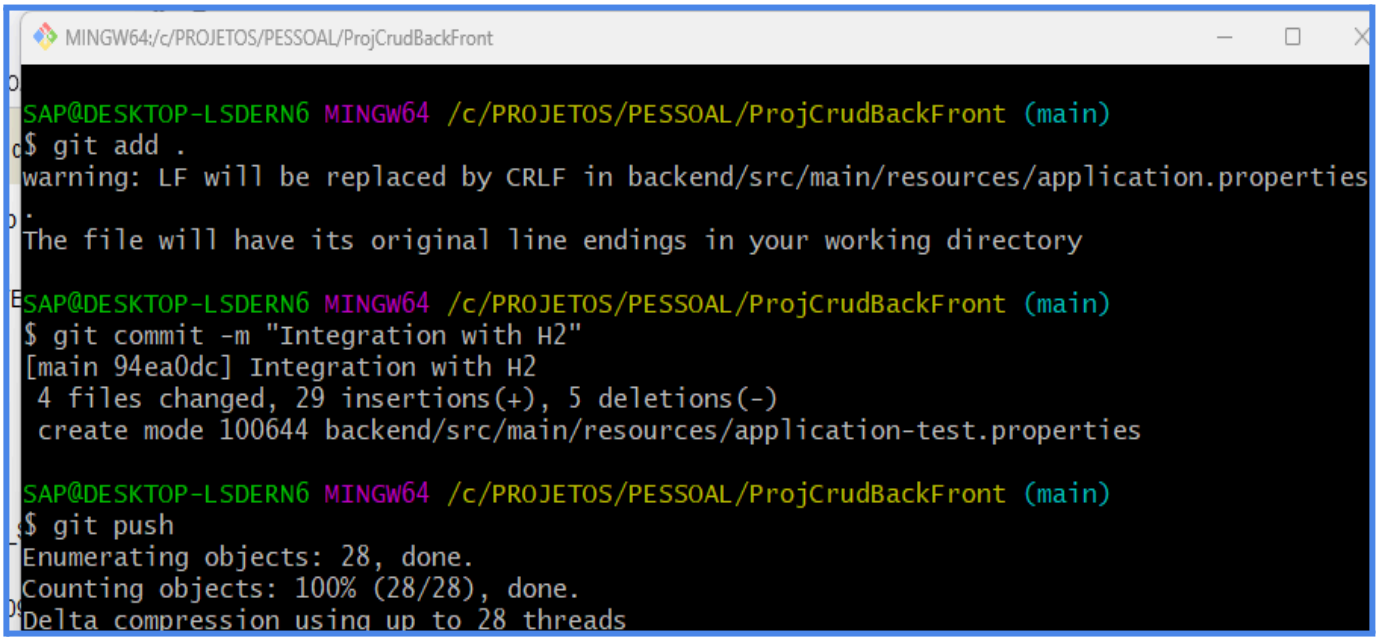
(1 row, 3 ms)

Testar no postman



Github-6

- “Git bash here” no diretório do projeto
  - git add .
  - git commit -m “Integration with H2”
  - git push

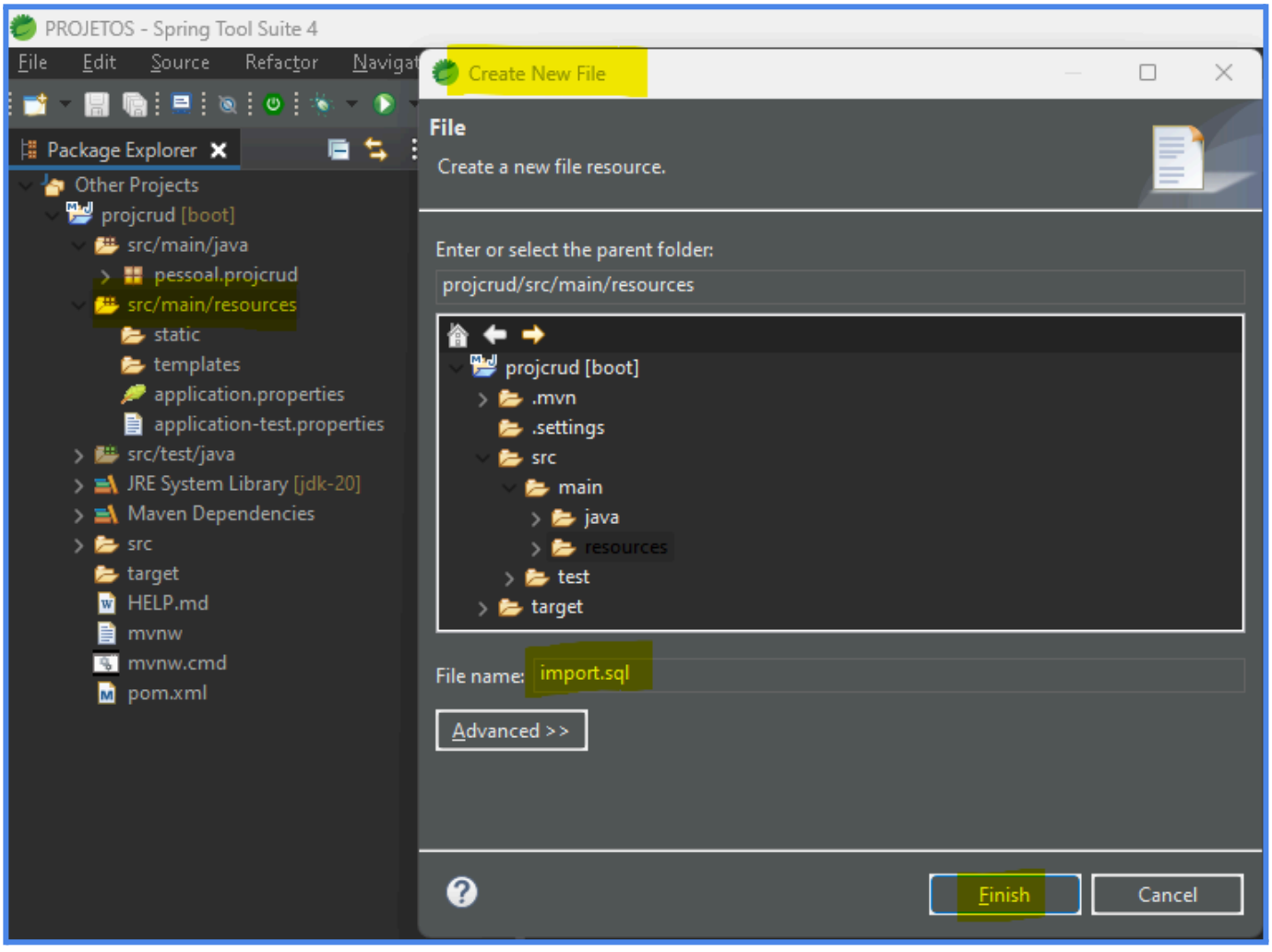


SEEDING DA BASE DE DADOS

- É o termo utilizado para dar uma carga inicial de dados ao banco, a fim de conseguirmos realizar testes com dados. Toda vez que o sistema reiniciar a carga inicial é gerada novamente.

IMPLEMENTAR A CARGA PARA O BANCO

Criar o import.sql, no src/main/resources

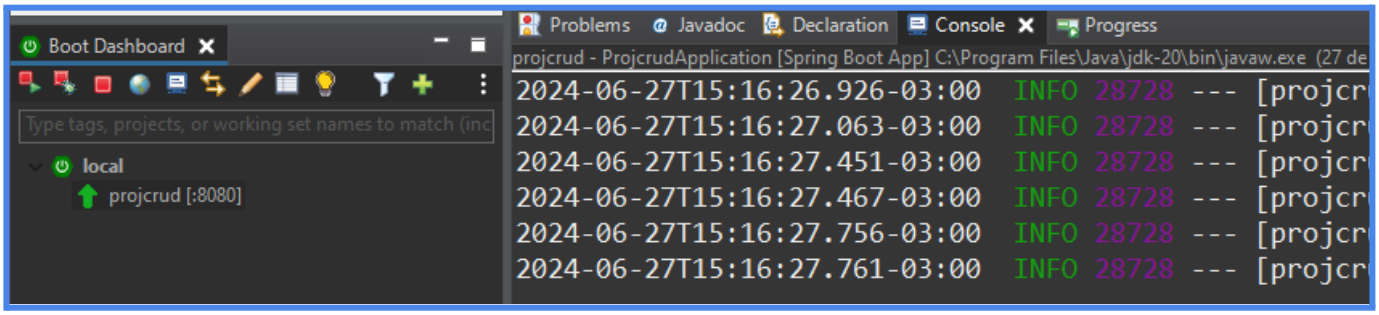


***NOTA**: Nas versões antigas do STS, usava-se o termo data.sql. A partir da versão STS 3, passou-se a utilizar a nomenclatura import.sql.*

Implementar os inserts para a carga inicial

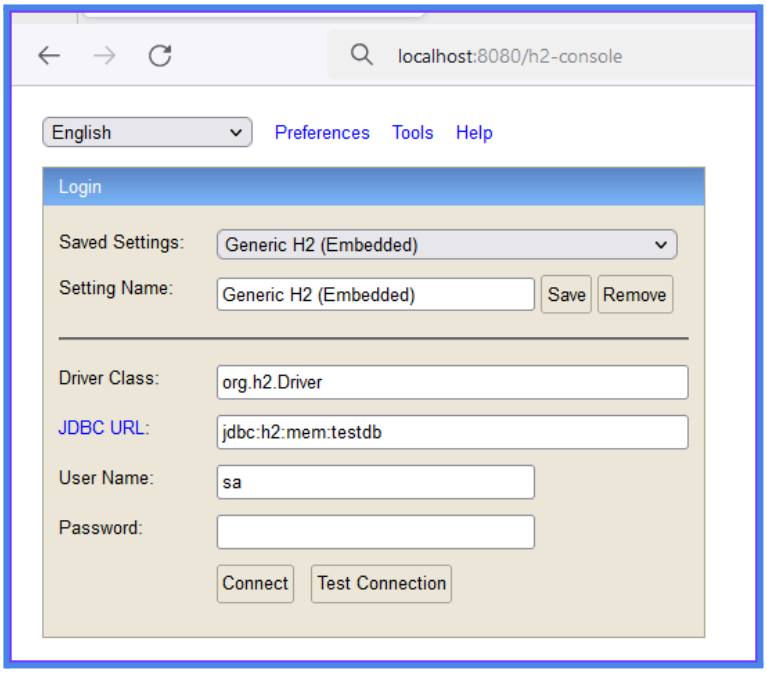
```
INSERT INTO tb_student (name, cpf, birth_date, income) VALUES ('Auricelio Freitas',
'000.111.222-33', TIMESTAMP WITH TIME ZONE '1982-08-28T10:30:00Z', 10700.5);
INSERT INTO tb_student (name, cpf, birth_date, income) VALUES ('Miguel Soares',
'000.111.222-33', TIMESTAMP WITH TIME ZONE '1989-02-10T10:30:00Z', 20700.5);
INSERT INTO tb_student (name, cpf, birth_date, income) VALUES ('Matheus Marques',
'000.111.222-33', TIMESTAMP WITH TIME ZONE '1992-11-23T10:30:00Z', 10700.5);
INSERT INTO tb_student (name, cpf, birth_date, income) VALUES ('Rafael Sales',
'000.111.222-33', TIMESTAMP WITH TIME ZONE '1988-10-13T10:30:00Z', 10700.5);
INSERT INTO tb_student (name, cpf, birth_date, income) VALUES ('Bruno Freitas',
'000.111.222-33', TIMESTAMP WITH TIME ZONE '1987-04-22T10:30:00Z', 9700.5);
INSERT INTO tb_student (name, cpf, birth_date, income) VALUES ('Romulo Araujo',
'000.111.222-33', TIMESTAMP WITH TIME ZONE '1991-01-09T10:30:00Z', 8700.5);
INSERT INTO tb_student (name, cpf, birth_date, income) VALUES ('Ryan Sousa',
'000.111.222-33', TIMESTAMP WITH TIME ZONE '1994-03-26T10:30:00Z', 7700.5);
INSERT INTO tb_student (name, cpf, birth_date, income) VALUES ('test1', '000.111.222-33',
null, 1024.0);
```

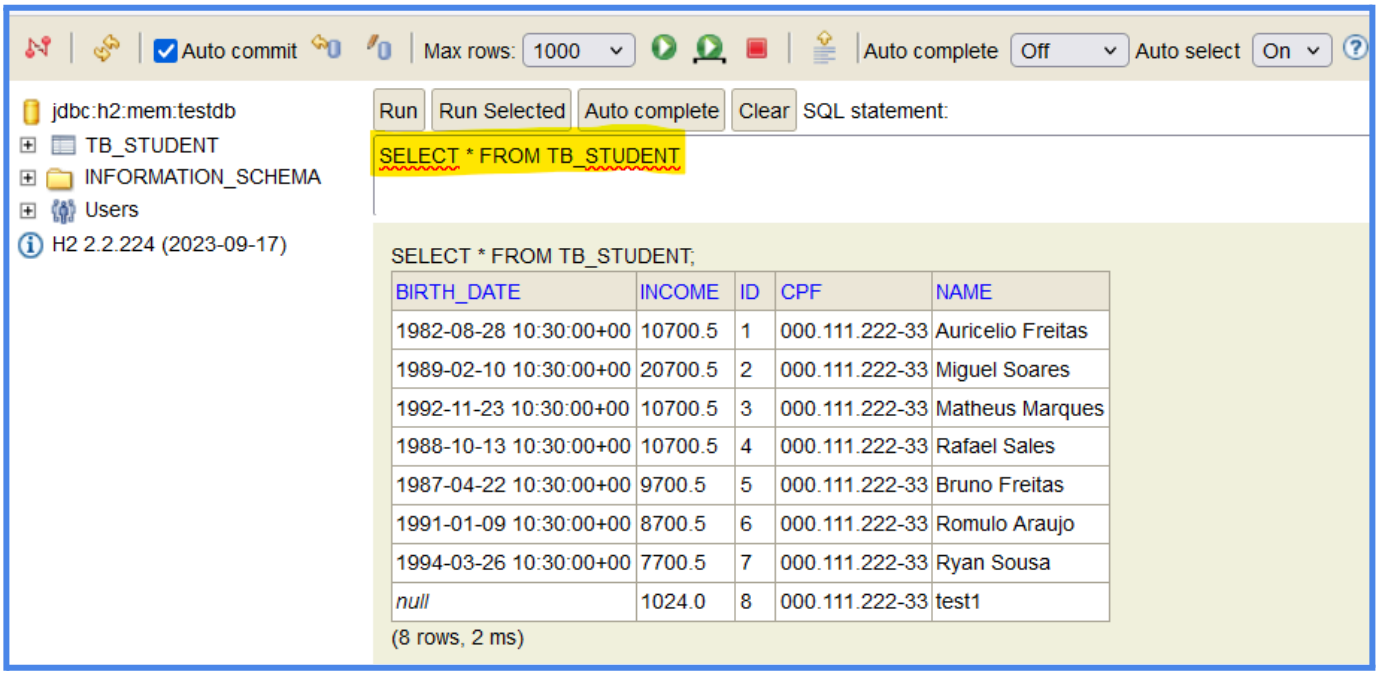
Rodar o projeto



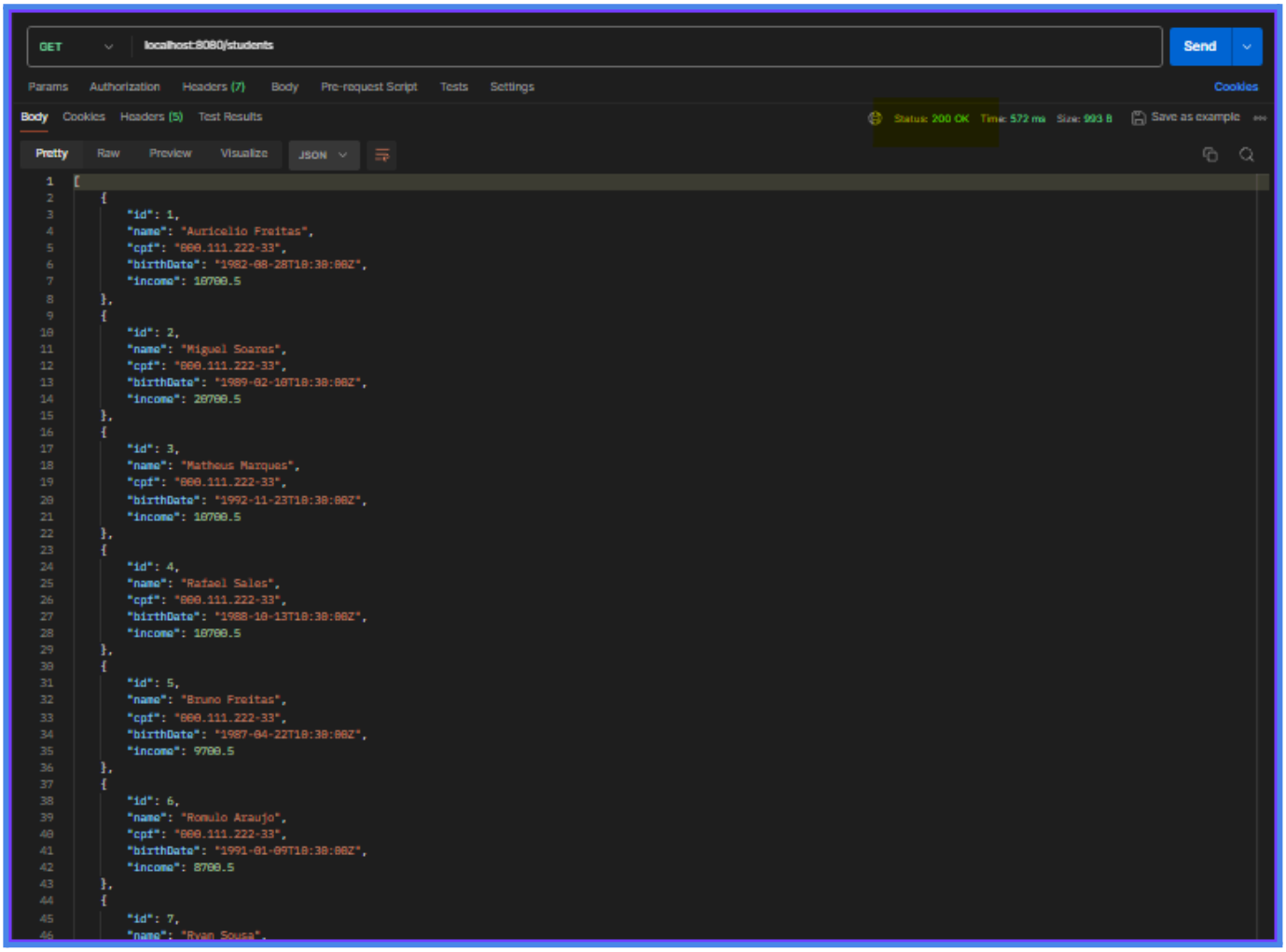
TESTAR A CARGA

Testar no banco H2



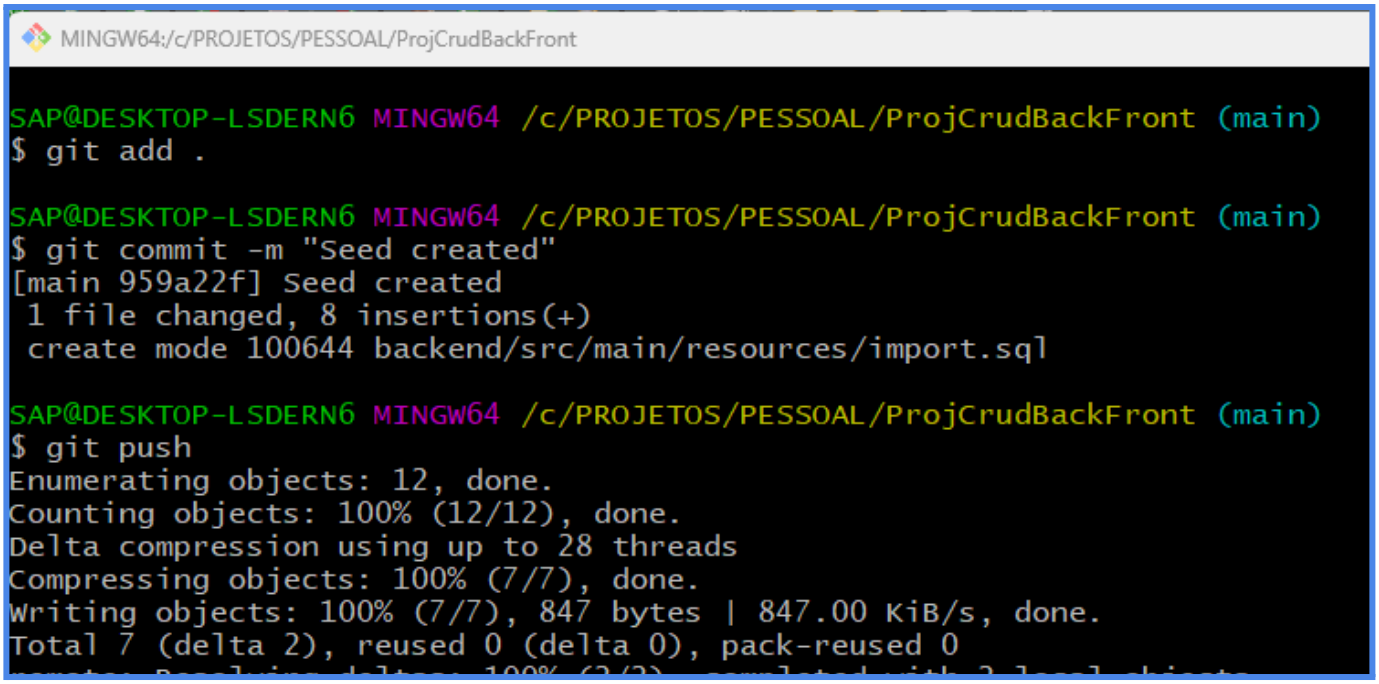


Testar no Postman



Github-7

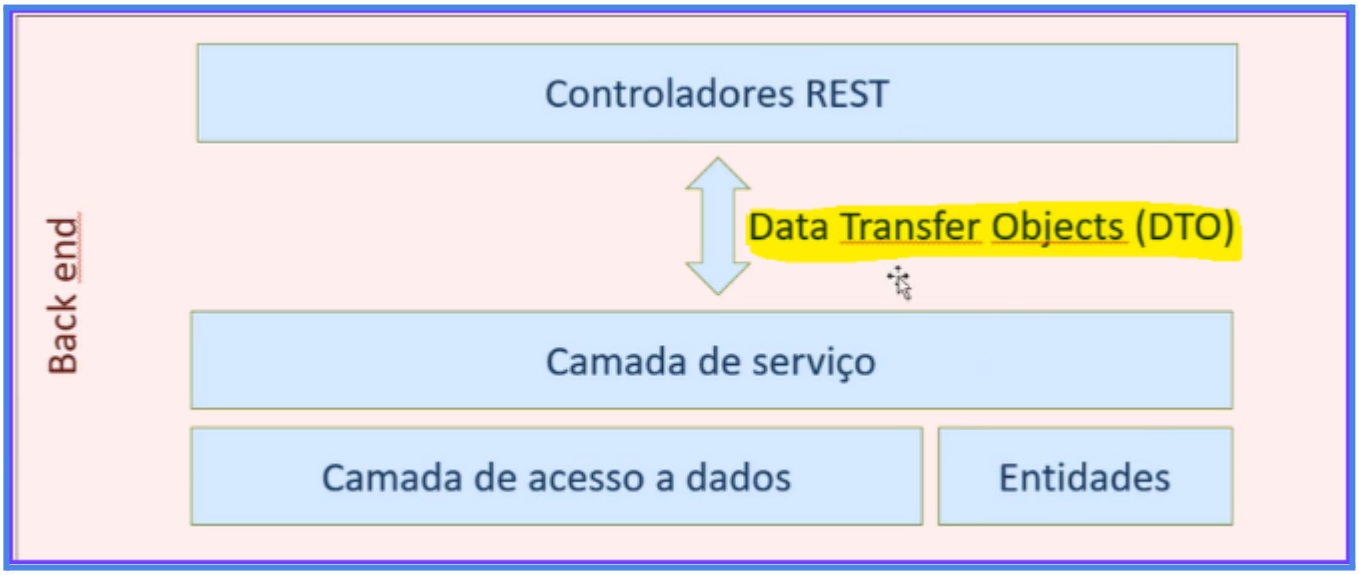
- “Git bash here” no diretório do projeto
    - git add backend
    - git commit -m “Seed created”
    - git push



DTO

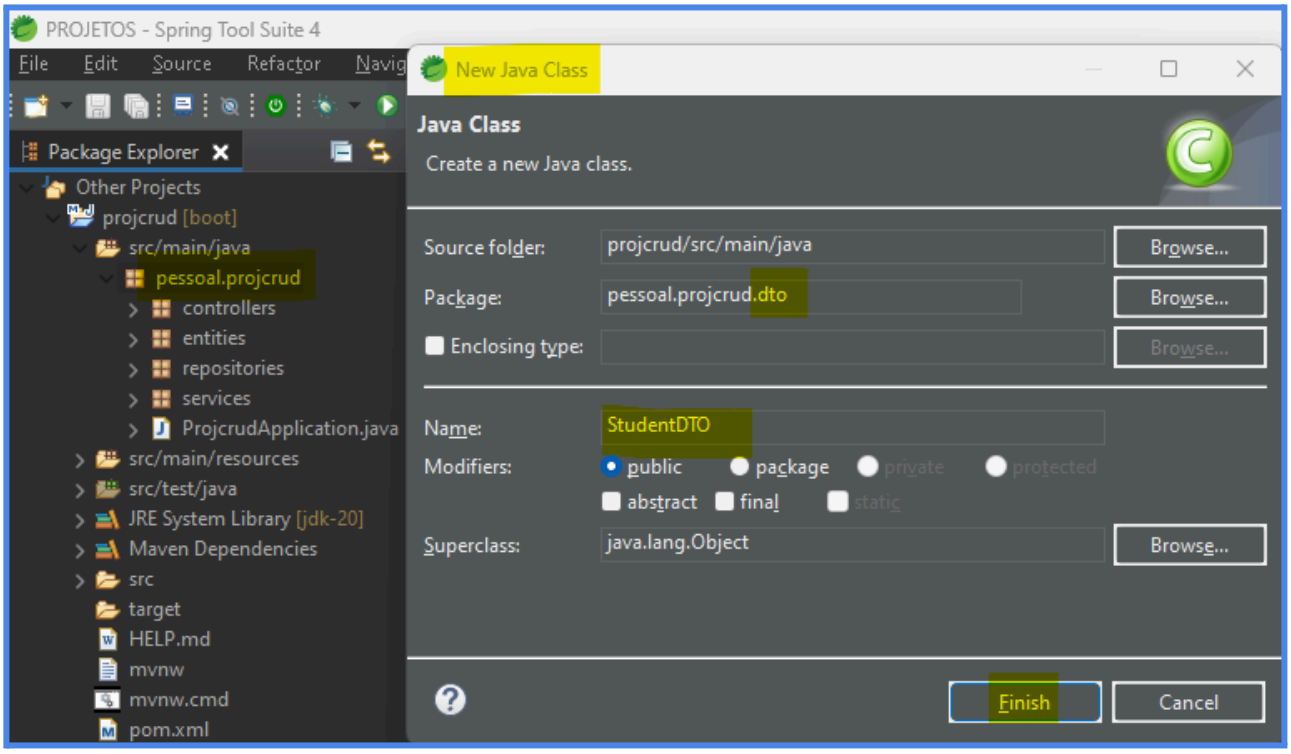
- DTO é um objeto que serve, apenas, para transferência de dados.
- Não tem relação com a JPA.
- O Controlador Rest não tem integração direta com a Entidade
- A comunicação entre o Controlador e o Serviço é feito por meio do DTO
- Podemos controlar quais dados serão entregues para a API

CONCEITUAL



IMPLEMENTAR A ESTRUTURA E O STUDENT_DTO

Criar a estrutura e a classe DTO



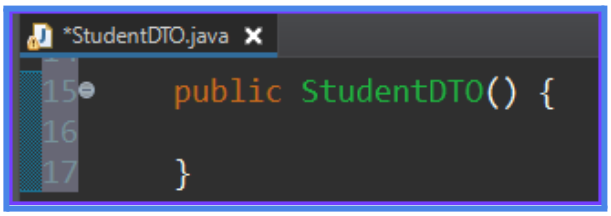
Implementar o Serializable e os mesmos atributos da Classe Student

*StudentDTO.java

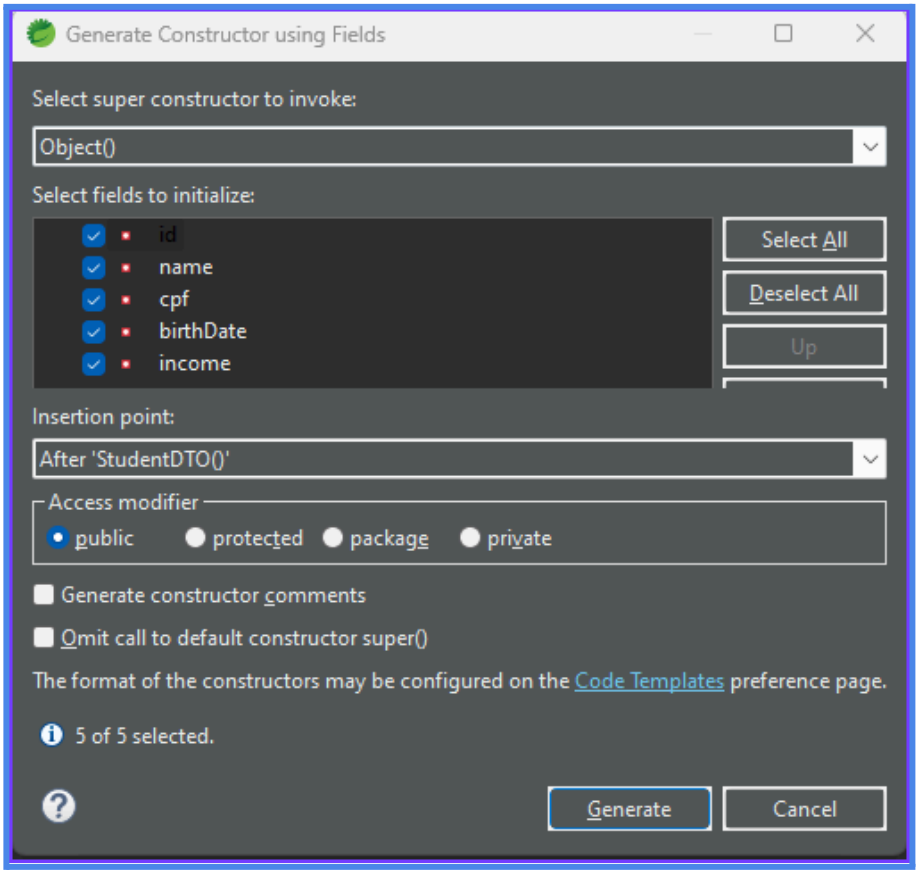
```
6  public class StudentDTO implements Serializable {
7      private static final long serialVersionUID = 1L;
8
9      private Long id;
10     private String name;
11     private String cpf;
12     private Instant birthDate;
13     private Double income;
```

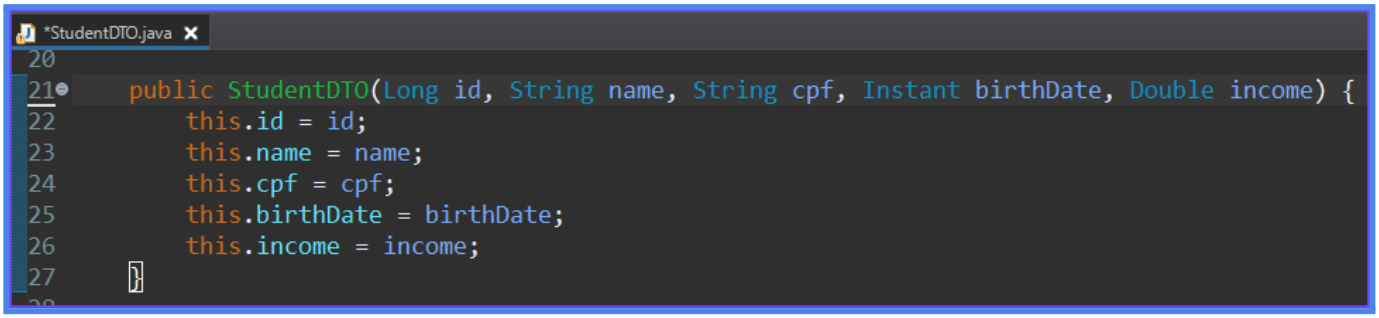
Implementar os construtores

Vazio

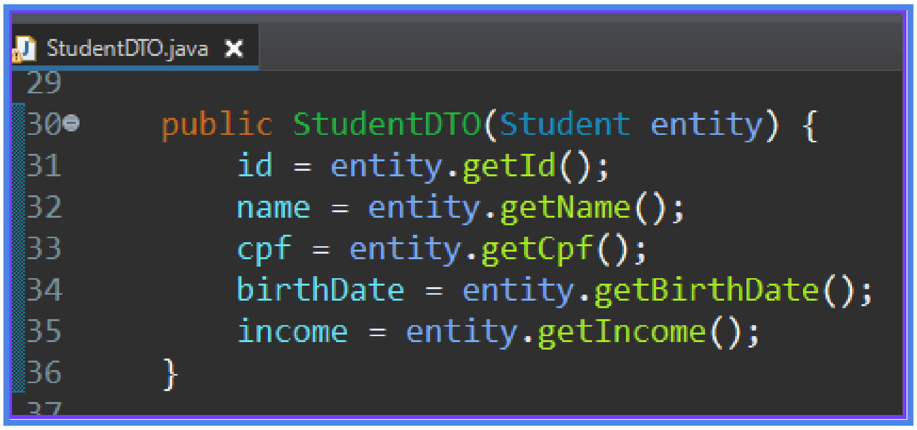


de Classe



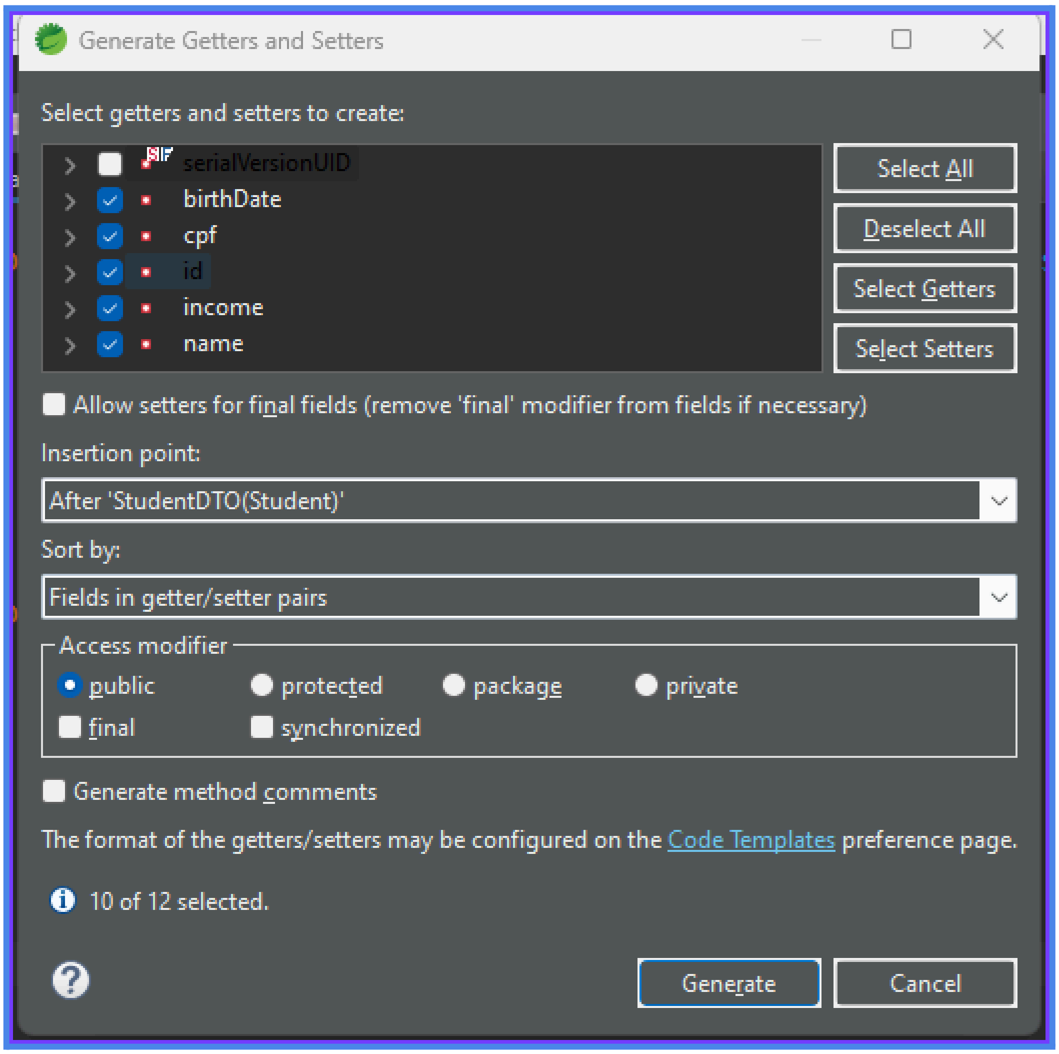


De Entidade



```java
StudentDTO.java
29
30      public StudentDTO(Student entity) {
31          id = entity.getId();
32          name = entity.getName();
33          cpf = entity.getCpf();
34          birthDate = entity.getBirthDate();
35          income = entity.getIncome();
36      }
```

Implementar os Getters and Setters



*StudentDTO.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 LocalDate getBirthDate() {
        return birthDate;
    }

    public void setBirthDate(LocalDate birthDate) {
        this.birthDate = birthDate;
    }

    public Double getIncome() {
        return income;
    }

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

REALIZAR OS AJUSTES PARA O DTO

Implementar o DTO na classe StudentService

StudentService.java

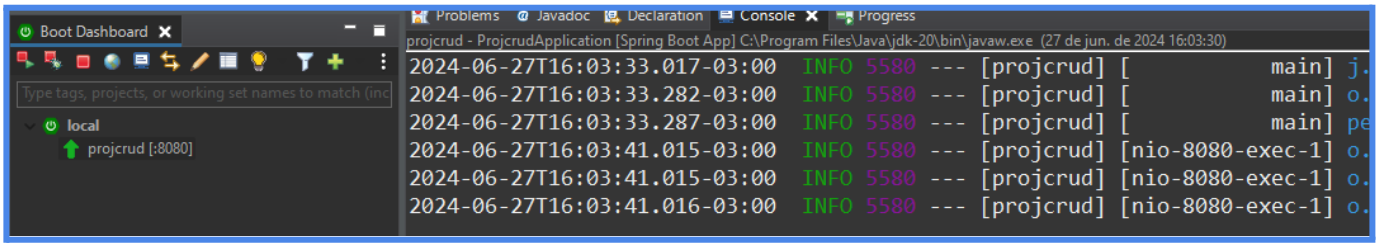
```java
package pessoal.projcrud.services;

import java.util.List;

@Service
public class StudentService {

    @Autowired
    private StudentRepository repository;

    @Transactional(readOnly = true)
    public List<StudentDTO> findAll() {

        List<Student> list = repository.findAll();

        return list.stream().map(x -> new StudentDTO(x)).collect(Collectors.toList());
    }
}
```

Implementar o DTO na classe StudentController

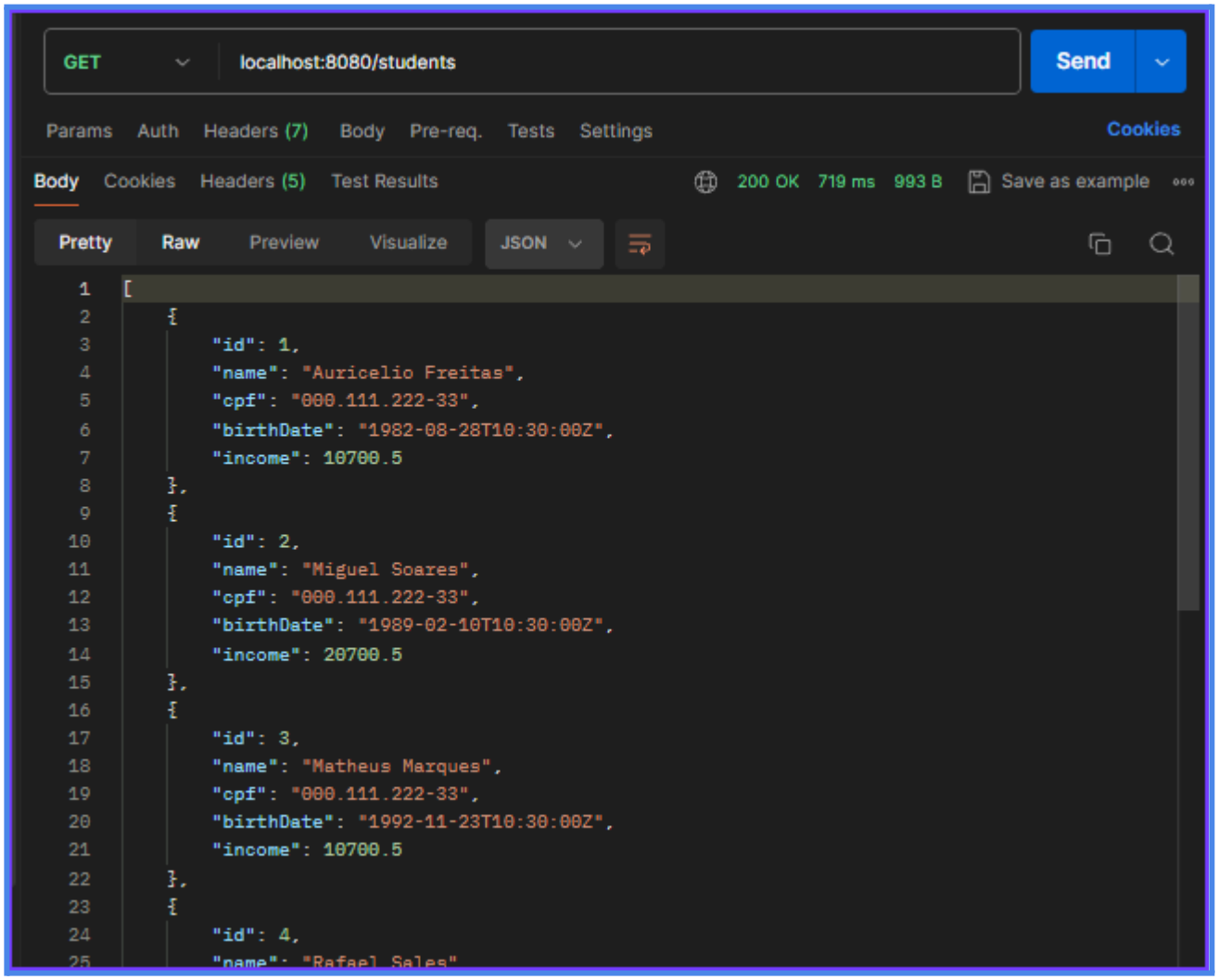
StudentController.java

```java
package pessoal.projcrud.controllers;

import java.util.List;

@RestController
@RequestMapping(value = "/students")
public class StudentController {

    @Autowired
    private StudentService service;

    @GetMapping
    public  ResponseEntity<List<StudentDTO>> findAll() {

        List<StudentDTO> list = service.findAll();

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

Rodar o projeto

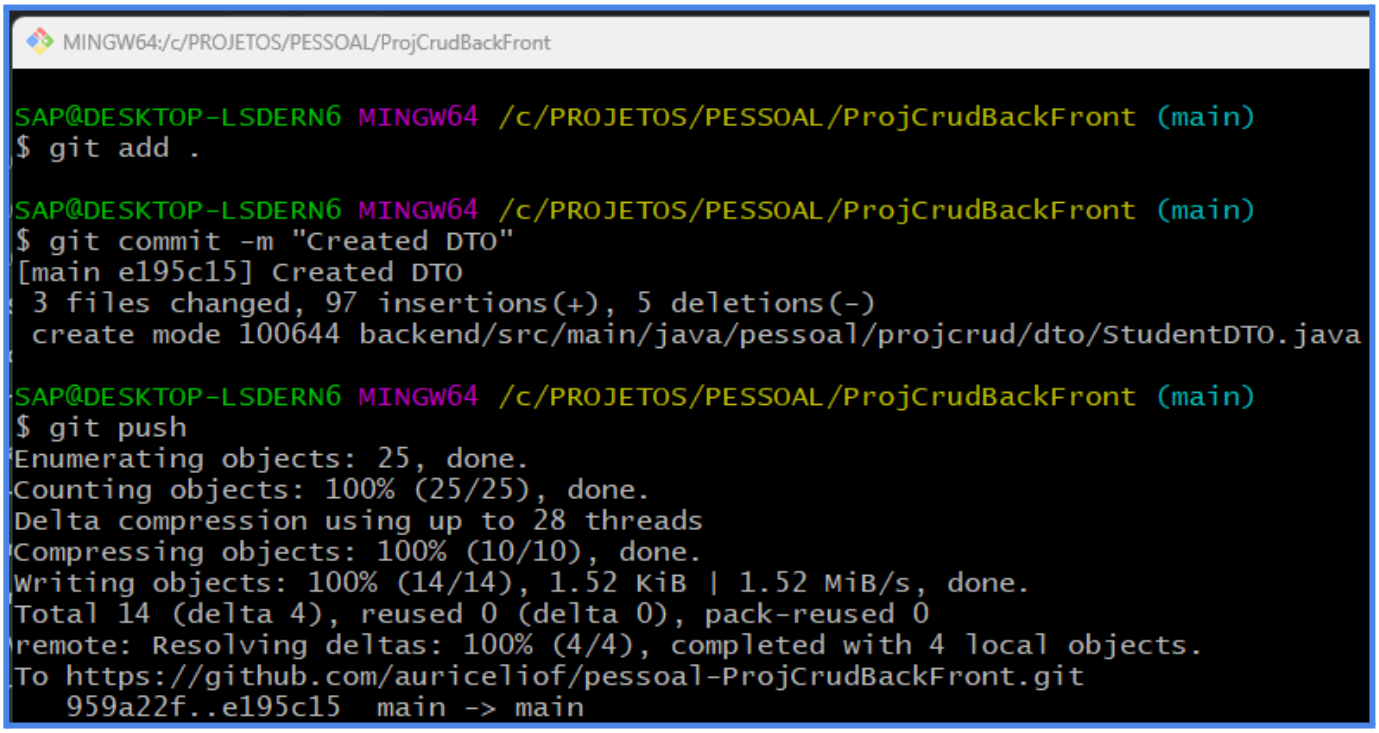


TESTAR COM O POSTMAN



Github-8

- “Git bash here” no diretório do projeto
  - git add backend
  - git commit -m “Created DTO”
  - git push



ENDPOINT: FIND_BY_ID

BUSCAR ALUNOS POR ID COM GET

Implementar busca por Id, no StudentController

```java
*StudentController.java
29
30    @GetMapping(value = "/{id}")
31    public ResponseEntity<StudentDTO> findById(@PathVariable Long id) {
32
33        StudentDTO dto = service.findById(id);
34
35        return ResponseEntity.ok().body(dto);
36    }
37 }
```

Implementar o método findById, no StudentService

DICA: *No StudentController, clicando sobre o método e em sobre o Create method …, será criado um método default no StudentService, onde podemos implementá-lo em seguida.*

```java
*StudentController.java
29
30    @GetMapping(value = "/{id}")
31    public ResponseEntity<StudentDTO> findById(@PathVariable Long id) {
32
33        StudentDTO dto = service.findById(id);
34
35        return ResponseEntity.ok(
36    }
37 }
38
```

The method findById(Long) is undefined for the type StudentService
2 quick fixes available:
Create method 'findById(Long)' in type 'StudentService'
Add cast to 'service'

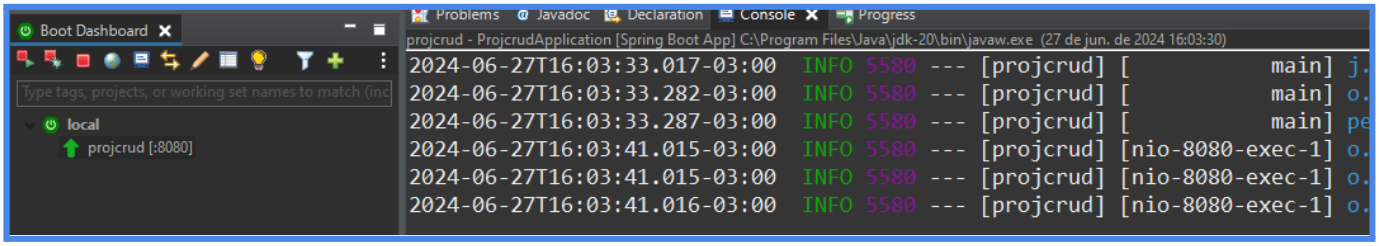
*StudentService.java

```
27
28      public StudentDTO findById(Long id) {
29          // TODO Auto-generated method stub
30          return null;
31      }
```

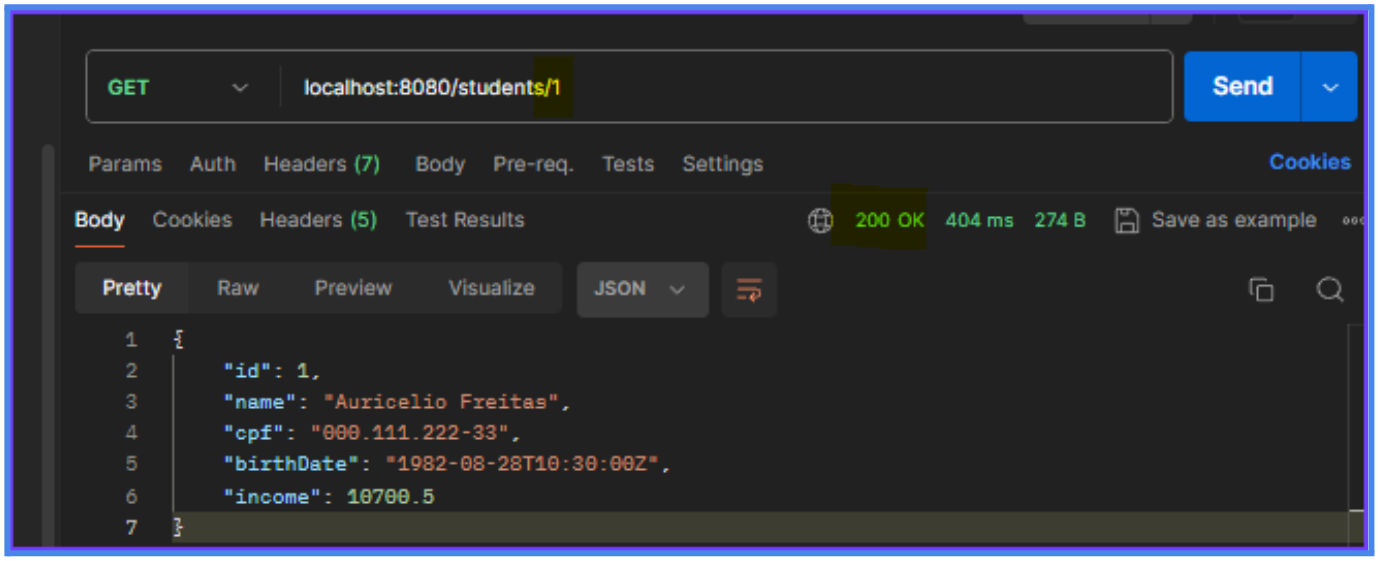
StudentService.java

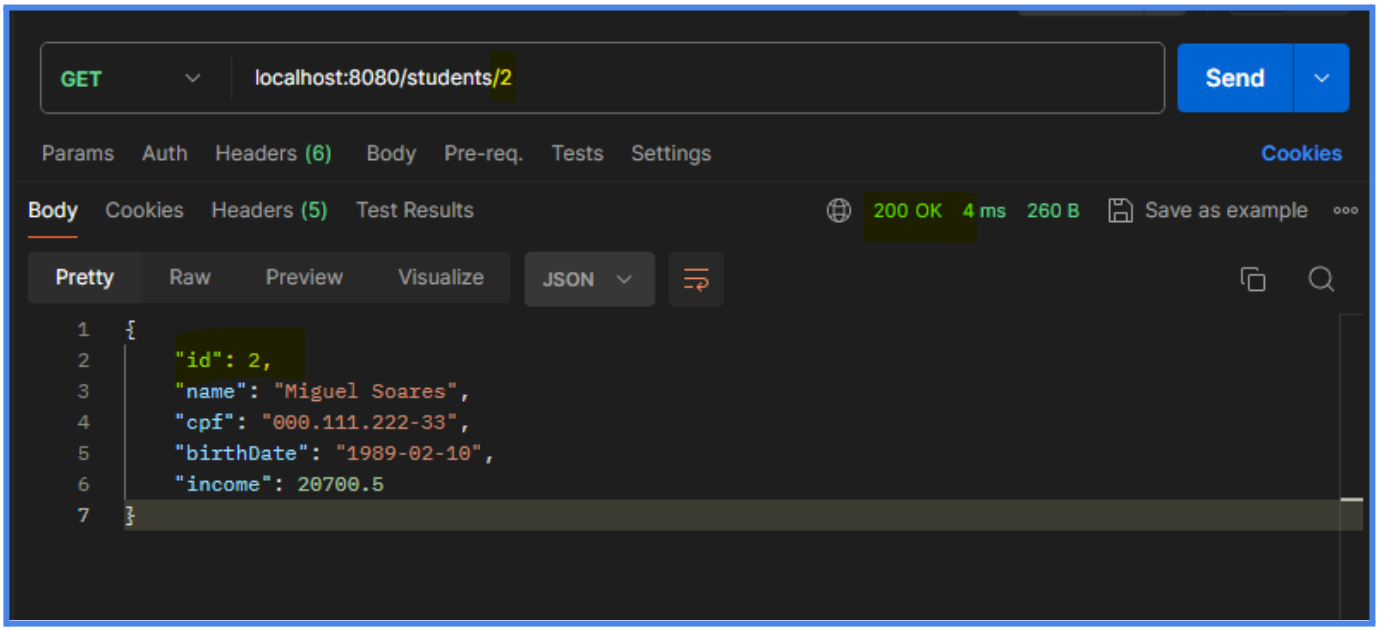
```
28
29      @Transactional
30      public StudentDTO findById(Long id) {
31
32          Optional<Student> obj = repository.findById(id);
33          Student entity = obj.get();
34
35          return new StudentDTO(entity);
36      }
37  }
```

Rodar o projeto



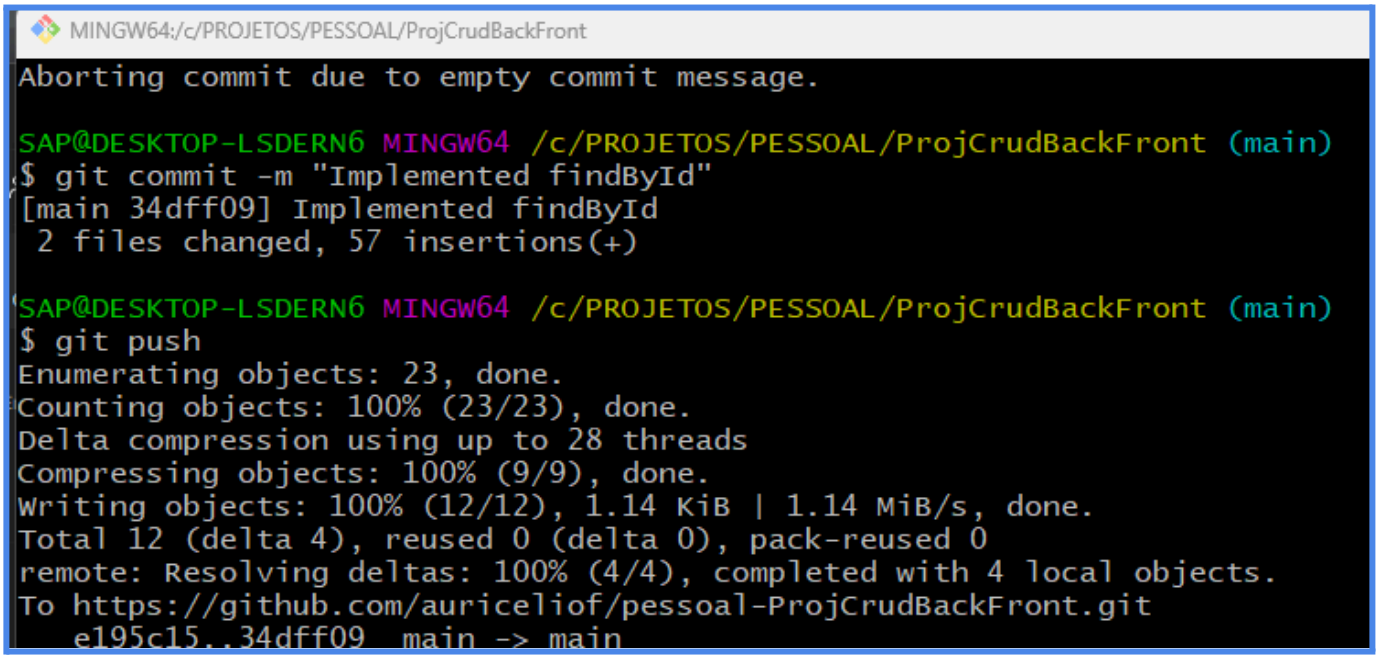
TESTAR NO POSTMAN





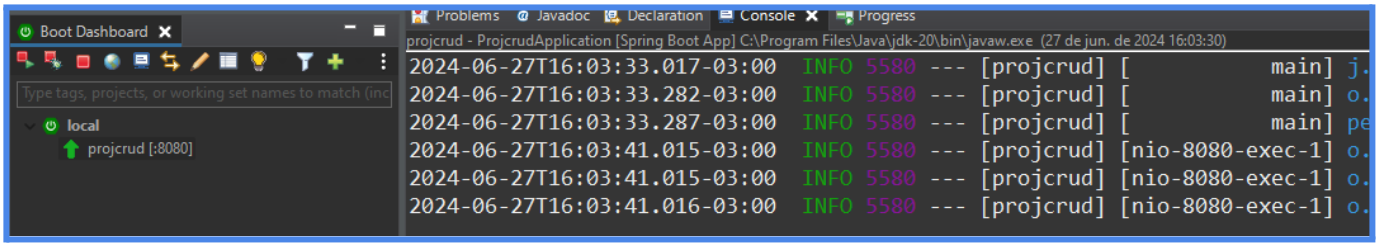
Github-9

- “Git bash here” no diretório do projeto
  - git add backend
  - git commit -m “Implemented findById”
  - git push

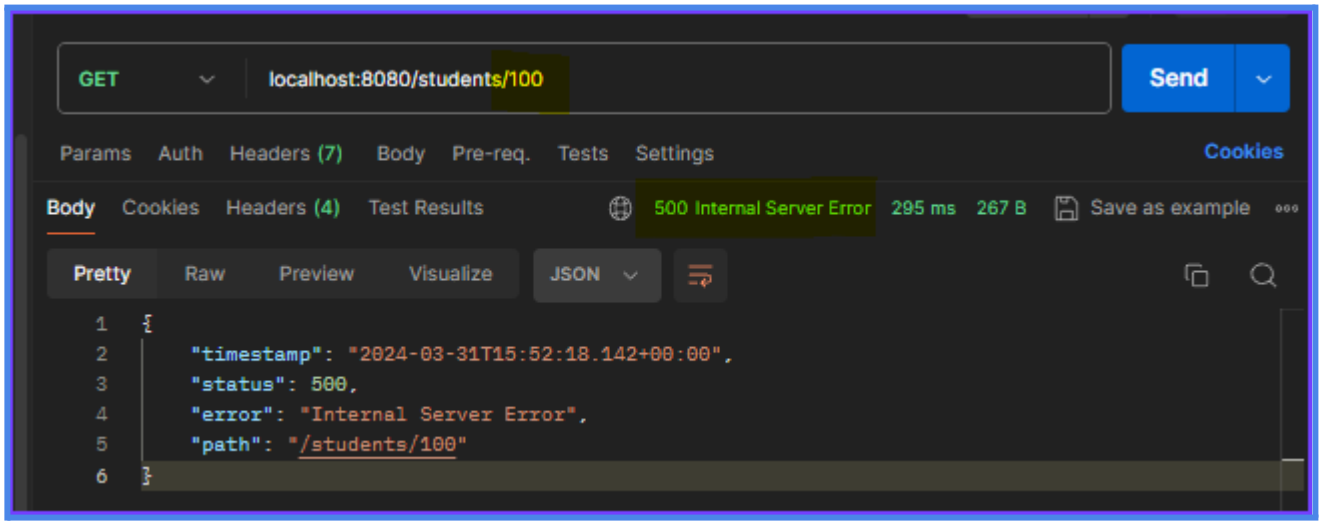


TRATAMENTO DE EXCEÇÕES PARA O FIND_BY_ID

Rodar o projeto

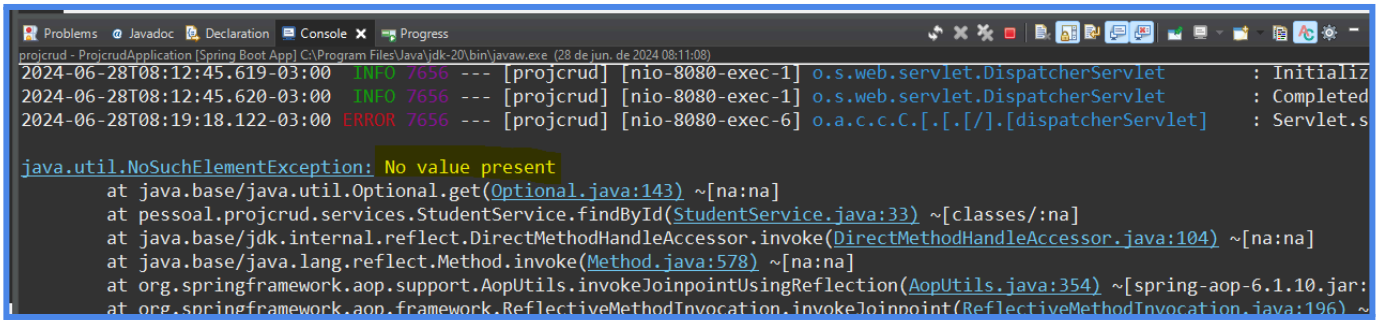


Simular erro no Postman

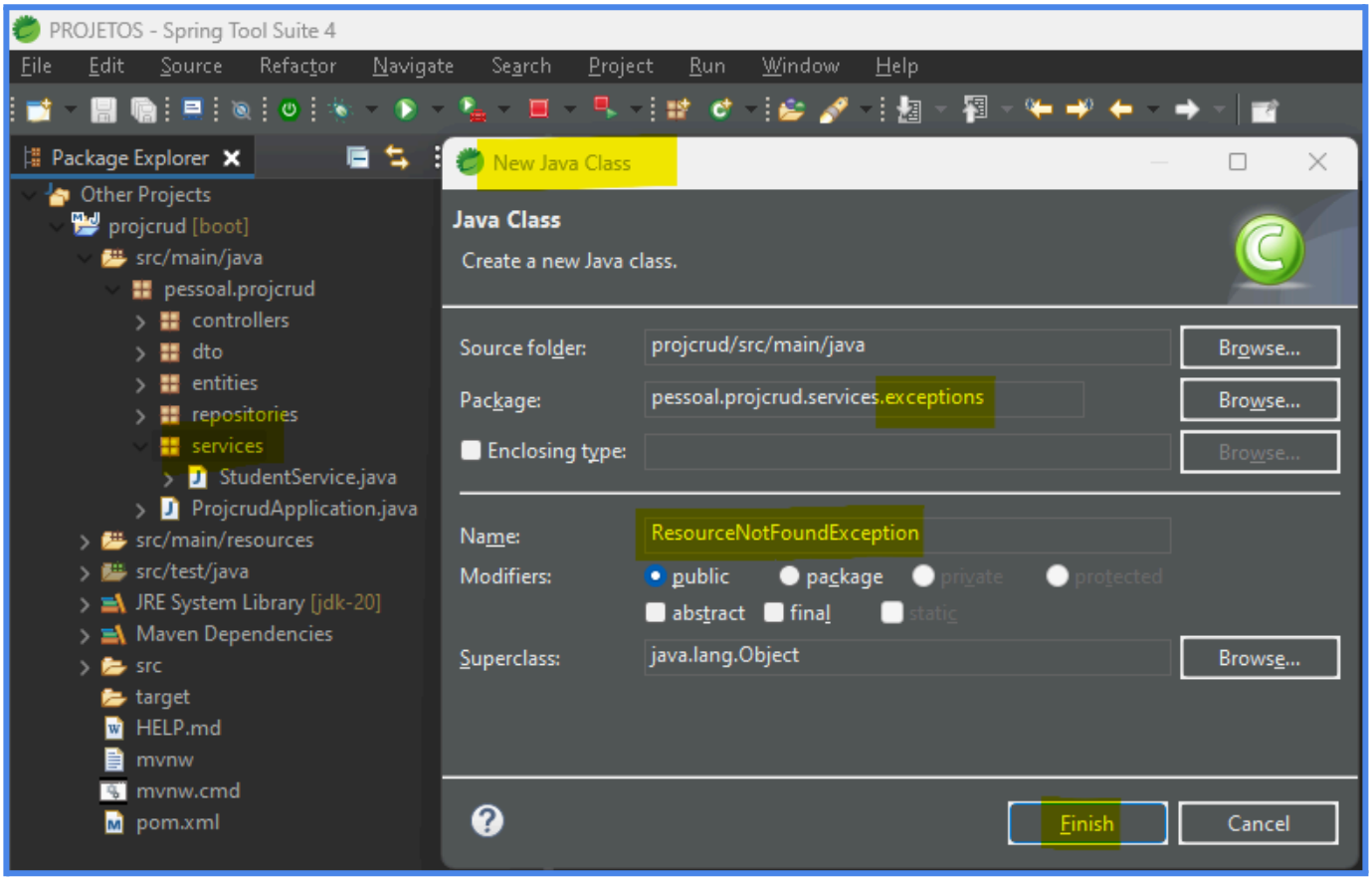


NOTA: Quando realizamos uma busca por um ID que não existe, é retornado o erro 500. Iremos tratá-lo a seguir.

Verificar o erro no console



Criar a estrutura de exceções no service



Implementar o ResourceNotFoundException

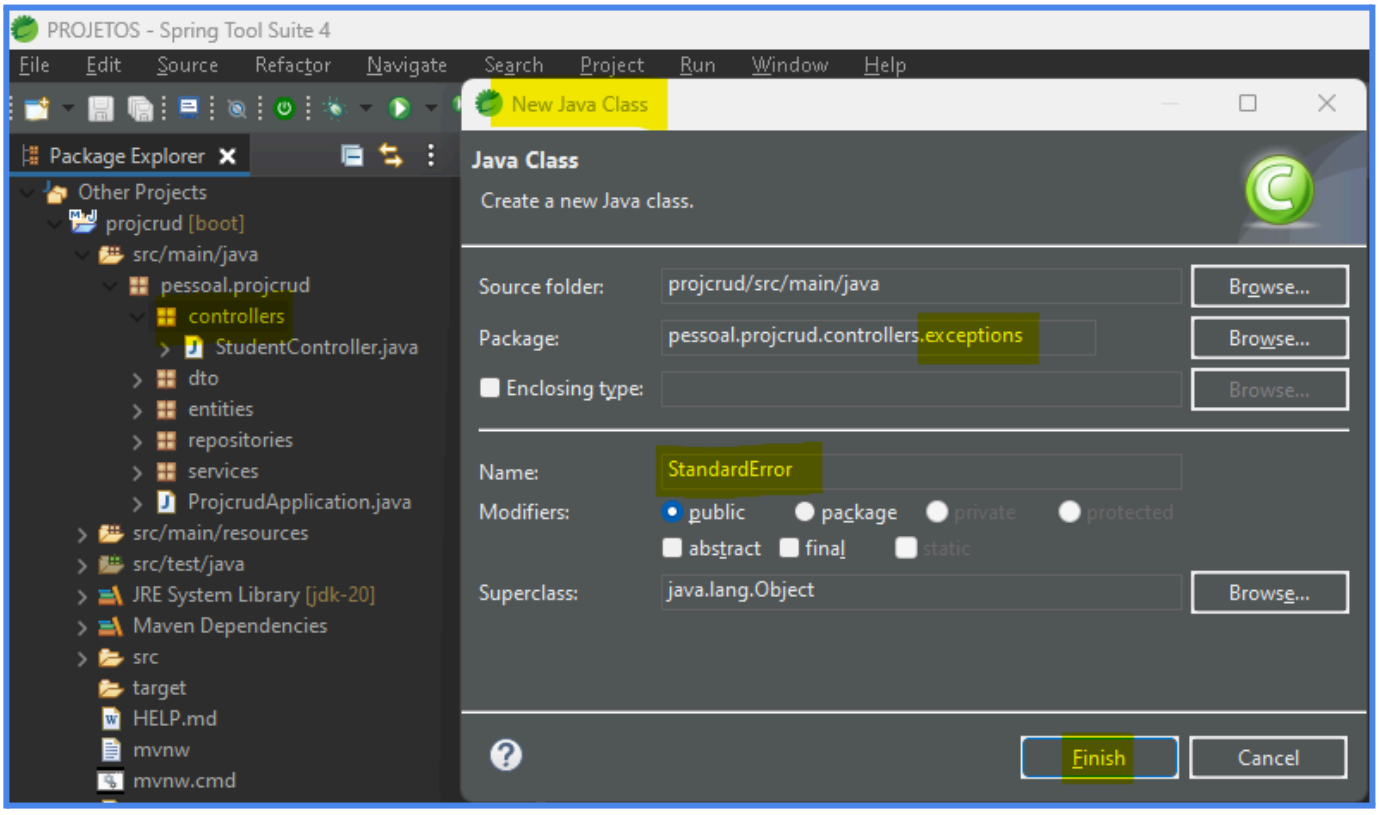
ResourceNotFoundException.java

```java
package pessoal.projcrud.services.exceptions;

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

    public ResourceNotFoundException(String msg) {
        super(msg);
    }
}
```

Implementar a exceção no findById, do StudentService

```
*StudentService.java
29
30      @Transactional(readOnly = true)
31      public StudentDTO findById(Long id) {
32
33          Optional<Student> obj = repository.findById(id);
34          Student entity = obj.orElseThrow(() -> new ResourceNotFoundException("Entity not found"));
35
36          return new StudentDTO(entity);
37      }
```

Criar a estrutura de exceções e uma classe personalizada no controller

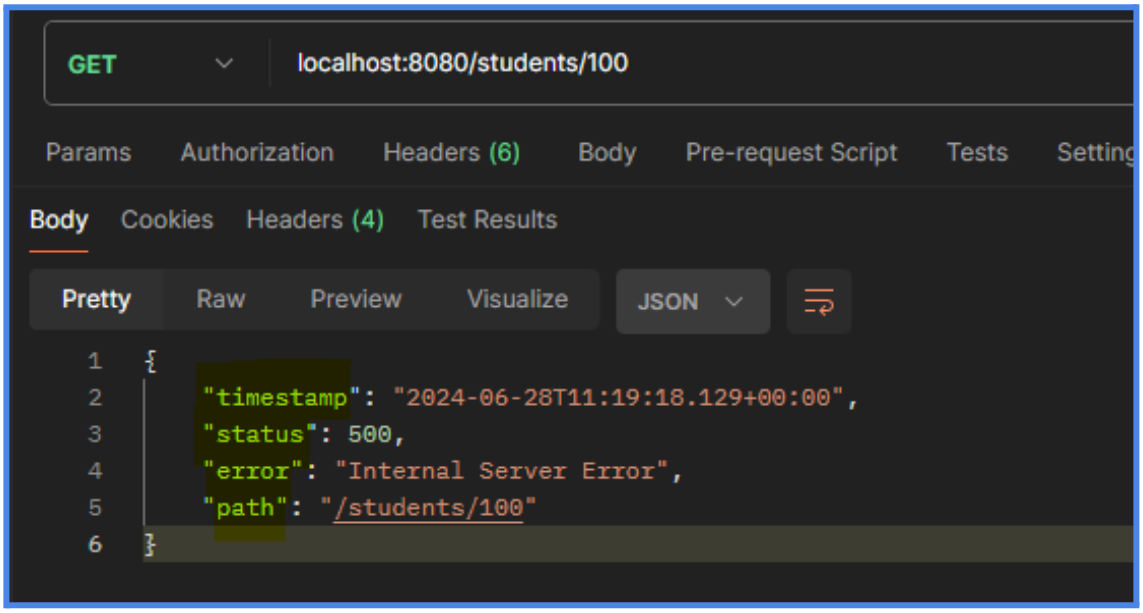


Implementar o Serializable e os atributos do StandardError conforme erro visto anteriormente

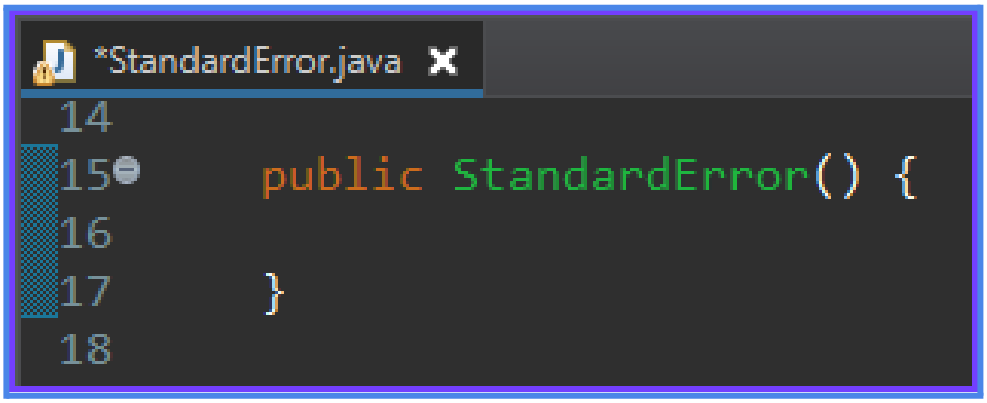
StandardError.java

```java
package pessoal.projcrud.controllers.exceptions;

import java.io.Serializable;
import java.time.Instant;

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

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

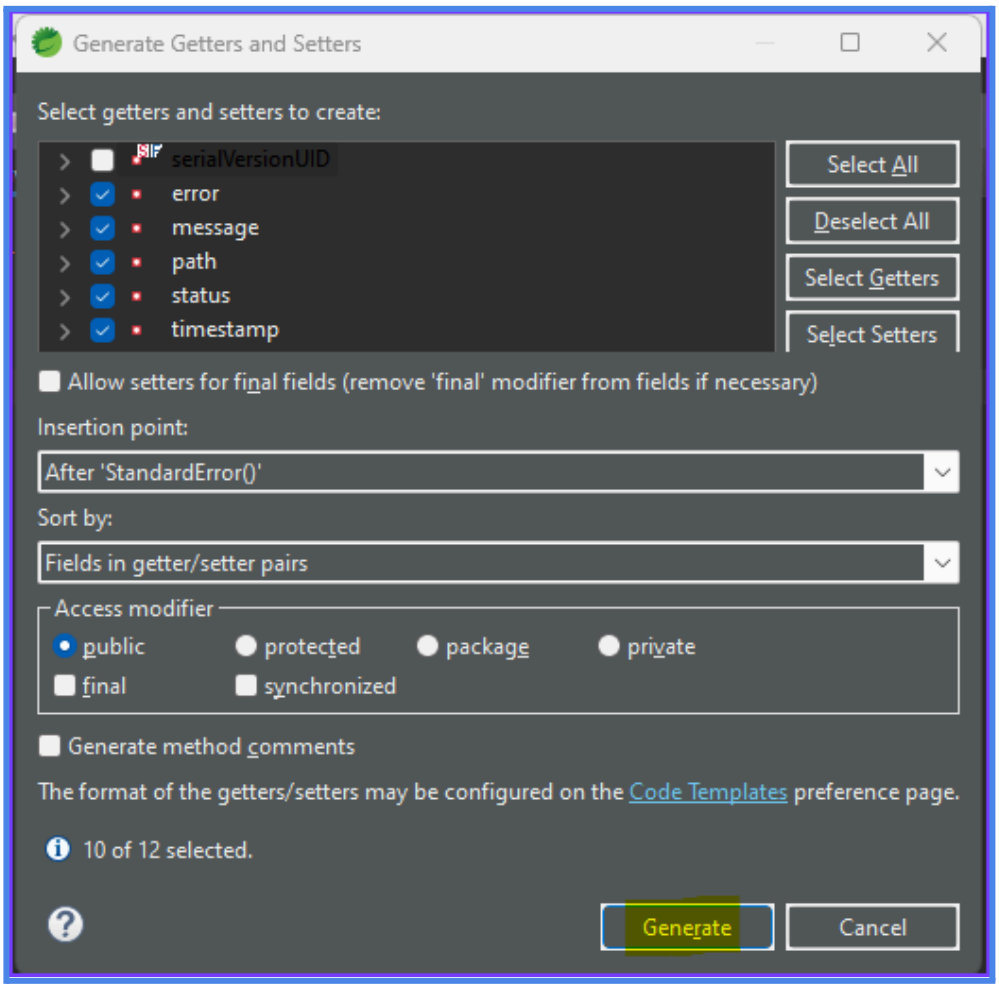
***NOTA**: Devemos definir os mesmos atributos mostrados no teste de erro do postman.*



Implementar um construtor vazio



Implementar os Getters and Setters

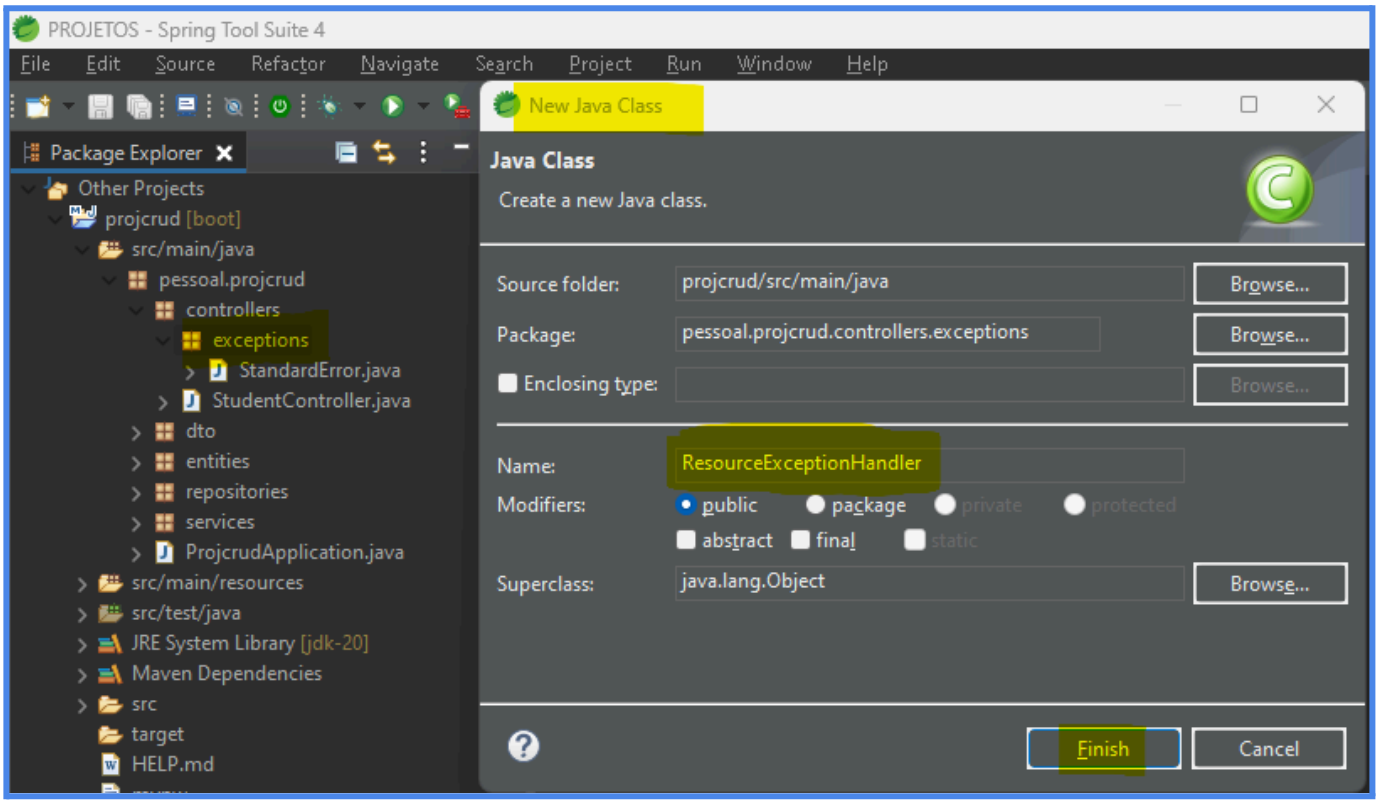


StandardError.java

```
18
19      public Instant getTimestamp() {
20          return timestamp;
21      }
22
23      public void setTimestamp(Instant timestamp) {
24          this.timestamp = timestamp;
25      }
26
27      public Integer getStatus() {
28          return status;
29      }
30
31      public void setStatus(Integer status) {
32          this.status = status;
33      }
34
35      public String getError() {
36          return error;
37      }
38
39      public void setError(String error) {
40          this.error = error;
41      }
```

```
43      public String getMessage() {
44          return message;
45      }
46
47      public void setMessage(String message) {
48          this.message = message;
49      }
50
51      public String getPath() {
52          return path;
53      }
54
55      public void setPath(String path) {
56          this.path = path;
57      }
58
59  }
```

Criar um controller advice para manipular a exceção

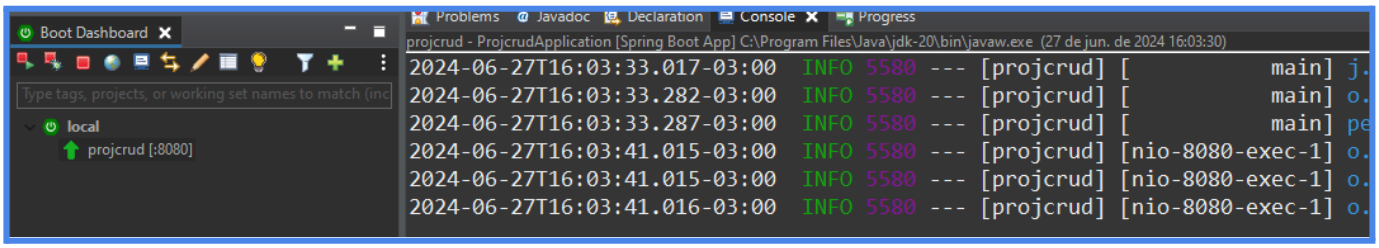


Implementar o ResourceExceptionHandler

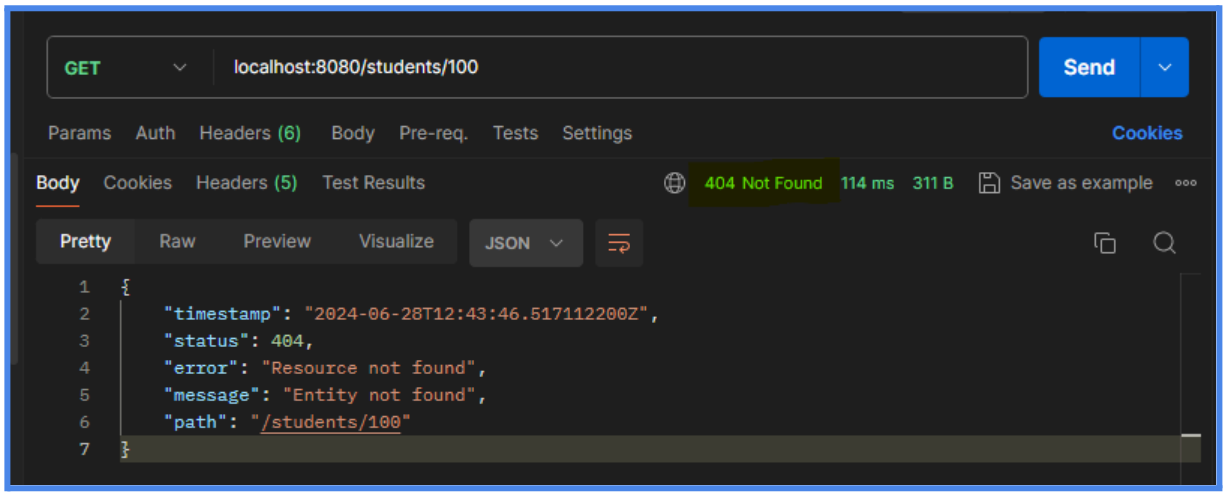
ResourceExceptionHandler.java

```java
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 jakarta.servlet.http.HttpServletRequest;
import pessoal.projcrud.services.exceptions.ResourceNotFoundException;

@ControllerAdvice
public class ResourceExceptionHandler {

    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<StandardError> resourceNotFound(ResourceNotFoundException e, HttpServletRequest request) {

        HttpStatus status = HttpStatus.NOT_FOUND;

        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);
    }
}
```

Rodar o projeto

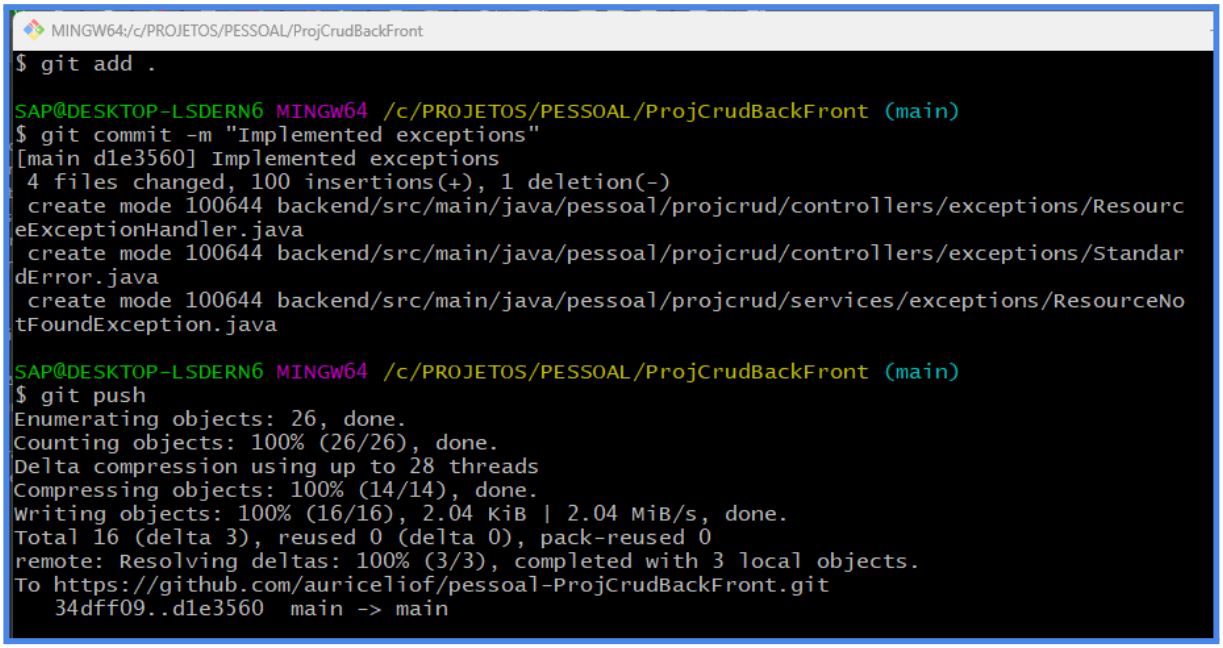


TESTAR NO POSTMAN



Github-10

- “Git bash here” no diretório do projeto
  - git add backend
  - git commit -m “Implemented exception for findById”
  - git push



PAGINAÇÃO

AJUSTAR O FIND_ALL PARA BUSCA PAGINADA

Implementar a busca paginada, no StudentController

StudentController.java

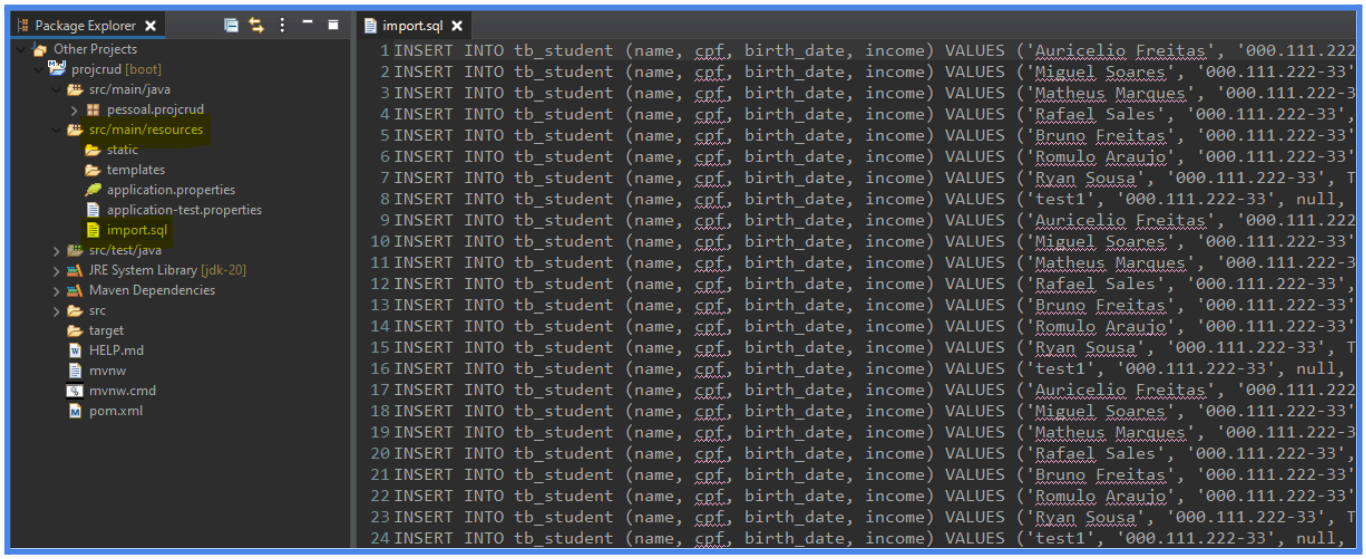
```java
package pessoal.projcrud.controllers;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import pessoal.projcrud.dto.StudentDTO;
import pessoal.projcrud.services.StudentService;

@RestController
@RequestMapping(value = "/students")
public class StudentController {

    @Autowired
    private StudentService service;

    @GetMapping
    public  ResponseEntity<Page<StudentDTO>> findAll(Pageable pageable) {

        Page<StudentDTO> list = service.findAllPaged(pageable);

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

Ajustar a busca paginada, no StudentService

StudentService.java

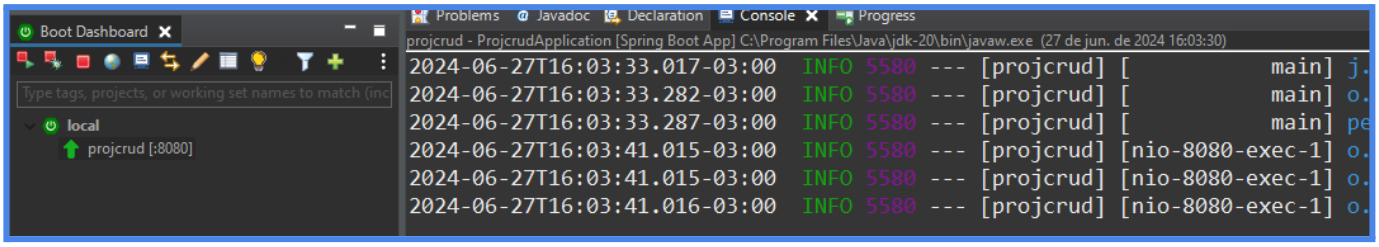
```java
package pessoal.projcrud.services;

import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import pessoal.projcrud.dto.StudentDTO;
import pessoal.projcrud.entities.Student;
import pessoal.projcrud.repositories.StudentRepository;
import pessoal.projcrud.services.exceptions.ResourceNotFoundException;

@Service
public class StudentService {

    @Autowired
    private StudentRepository repository;

    @Transactional(readOnly = true)
    public Page<StudentDTO> findAllPaged(Pageable pageable) {

        Page<Student> list = repository.findAll(pageable);

        return list.map(x -> new StudentDTO(x));
    }

```

Expandir o seed do banco para teste de paginação



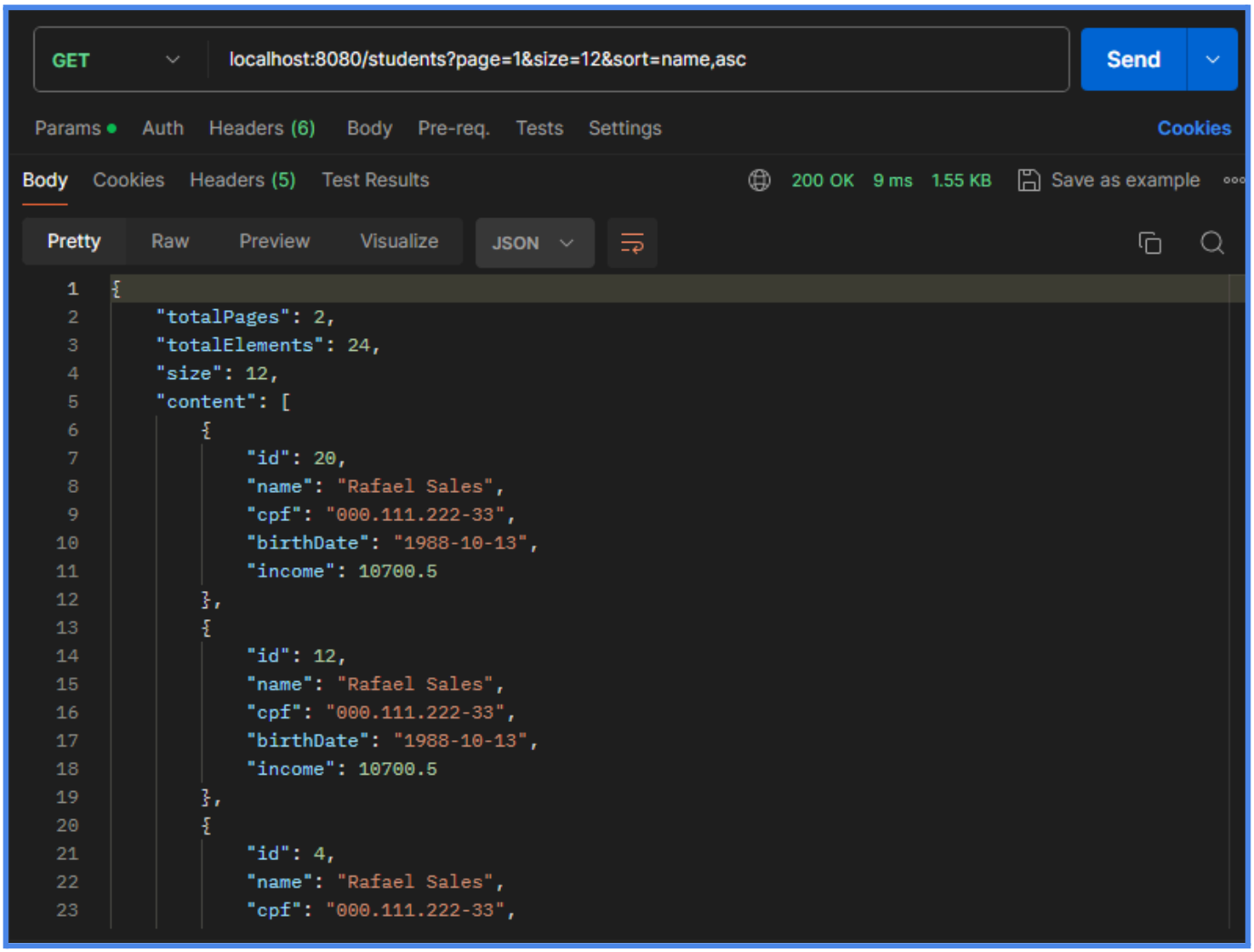
***NOTA**: Neste caso, apenas replicar os existentes (copiar, colar).*

Rodar o projeto



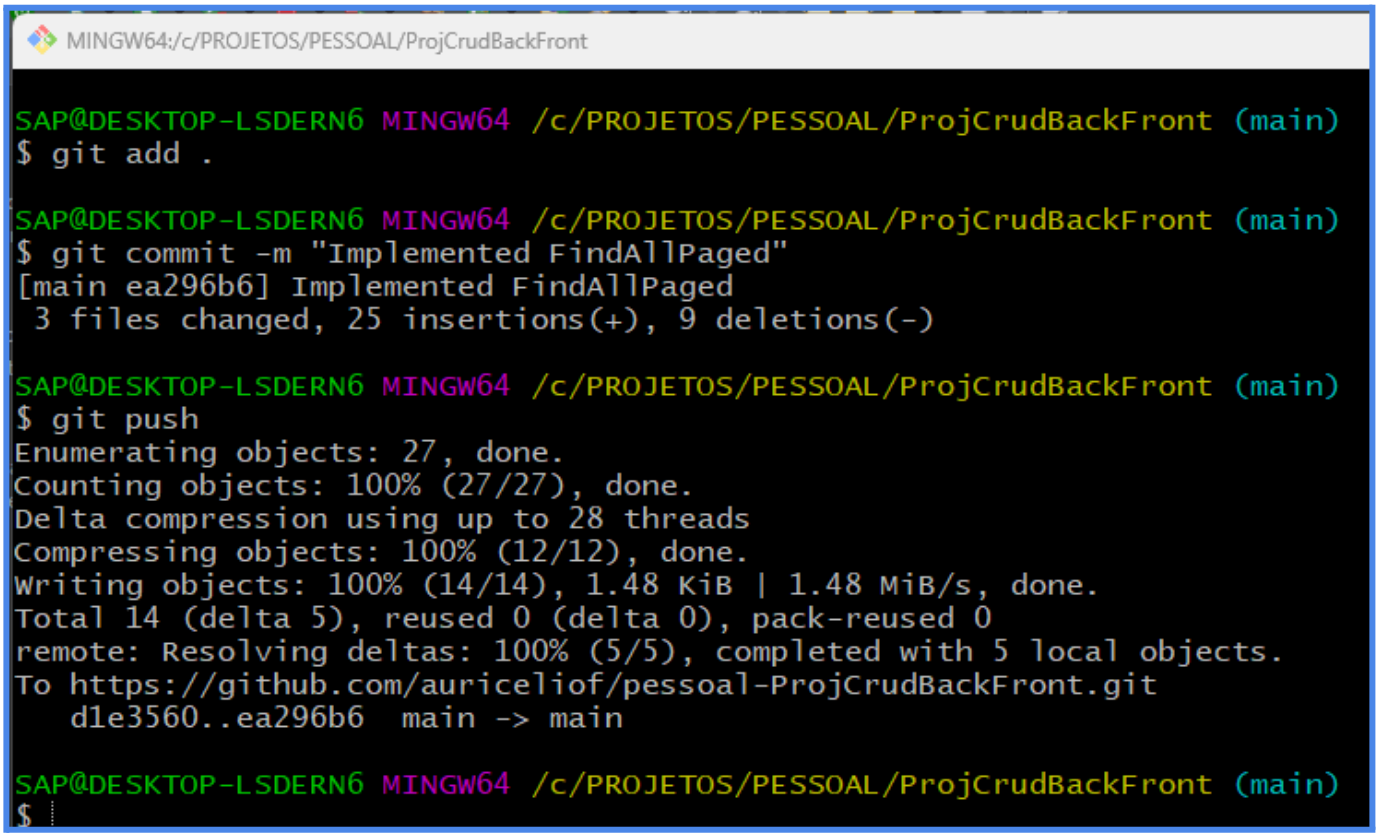
TESTAR NO POSTMAN

- *GET: localhost:8080/students?page=0&size=12&sort=name,asc*



Github-11

- “Git bash here” no diretório do projeto
  - git add backend
  - git commit -m “Implemented FindAllPaged”
  - git push



ENDPOINT - INSERT

INSERIR NOVO ALUNO COM POST

Implementar o insert, no StudentController

StudentController.java

```
40      @PostMapping
41      public ResponseEntity<StudentDTO> insert(@RequestBody StudentDTO dto) {
42
43          dto = service.insert(dto);
44
45          return ResponseEntity.ok().body(dto);
46      }
47  }
```

Implementar a metodologia REST ao método

URI uri = ServletUriComponentsBuilder.fromCurrentRequest().path("{/id}").buildAndExpand(dto.getId()).toUri();

StudentController.java

```
42
43      @PostMapping
44      public ResponseEntity<StudentDTO> insert(@RequestBody StudentDTO dto) {
45
46          dto = service.insert(dto);
47
48          URI uri = ServletUriComponentsBuilder.fromCurrentRequest().path("{/id}")
49                  .buildAndExpand(dto.getId()).toUri();
50
51          return ResponseEntity.created(uri).body(dto);
52      }
53  }
```

***OBS**: Importar o URI do "java.net"*

***NOTA**: Implantamos o caminho no Header da requisição e corrigimos o retorno de 200 (padrão) para 201 (recomendação REST), com o created.*

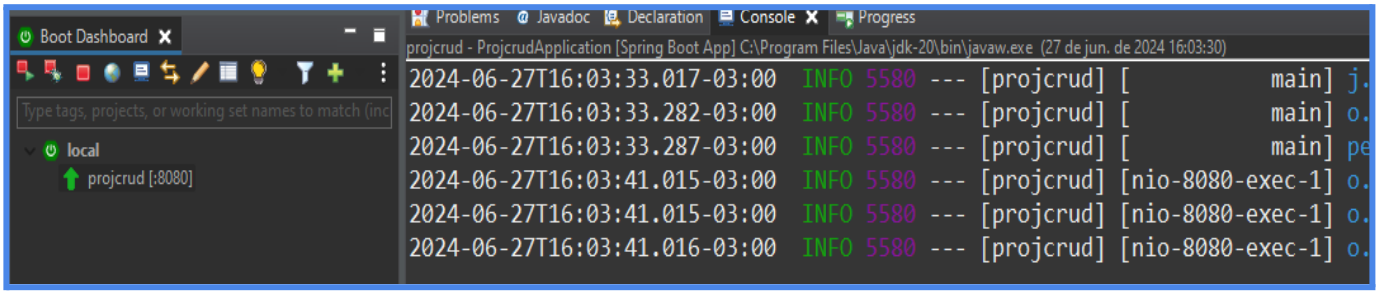
Implementar o método insert, convertendo o DTO para uma entidade, no StudentService

StudentService.java

```
38
39      @Transactional
40      public StudentDTO insert(StudentDTO dto) {
41
42          Student entity = new Student();
43
44              entity.setName(dto.getName());
45              entity.setCpf(dto.getCpf());
46              entity.setBirthDate(dto.getBirthDate());
47              entity.setIncome(dto.getIncome());
48
49          entity = repository.save(entity);
50
51          return new StudentDTO(entity);
52      }
53  }
```

***NOTA**: Não colocar o atributo ID, pois é o banco que irá autoincrementar.*

Rodar o projeto

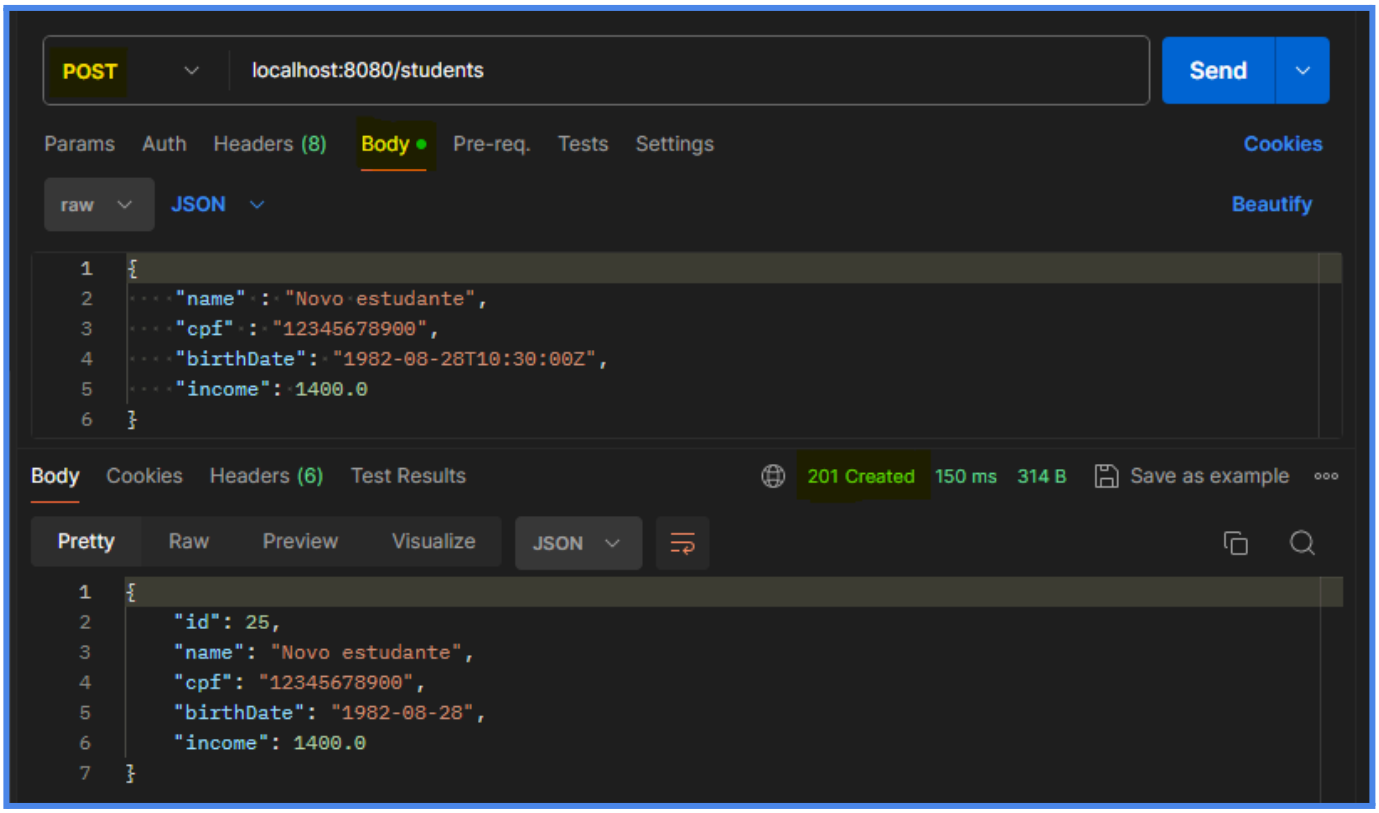


TESTAR NO POSTMAN

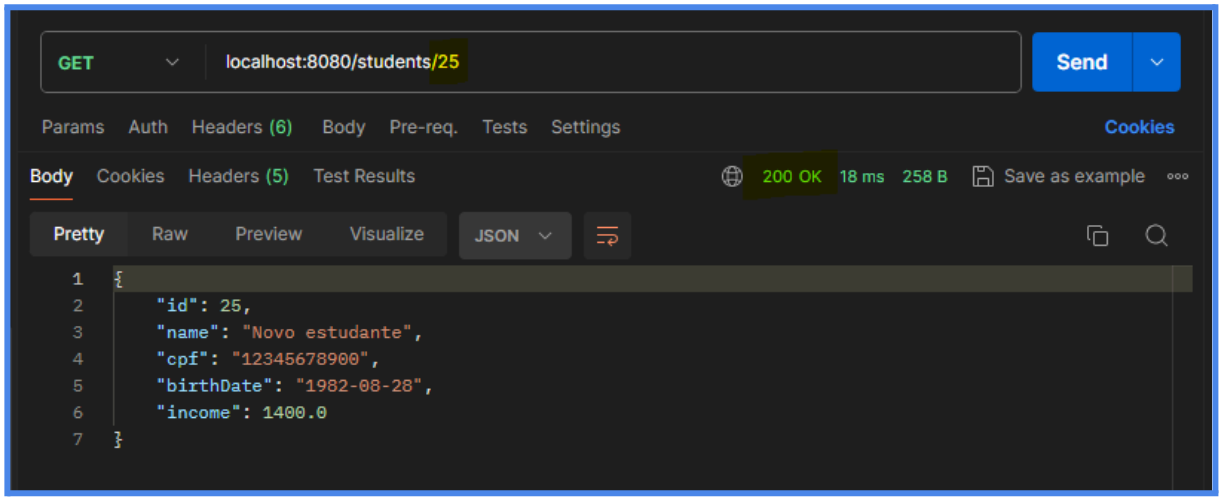
- *POST: localhost:8080/students*
  - *Body:*

```
{
   "name" : "Novo aluno",
   "cpf" : "123.456.789-00",
   "birthDate" : "2005-01-12T10:28:00Z",
   "income" : 1920.0
}
```

Inserir

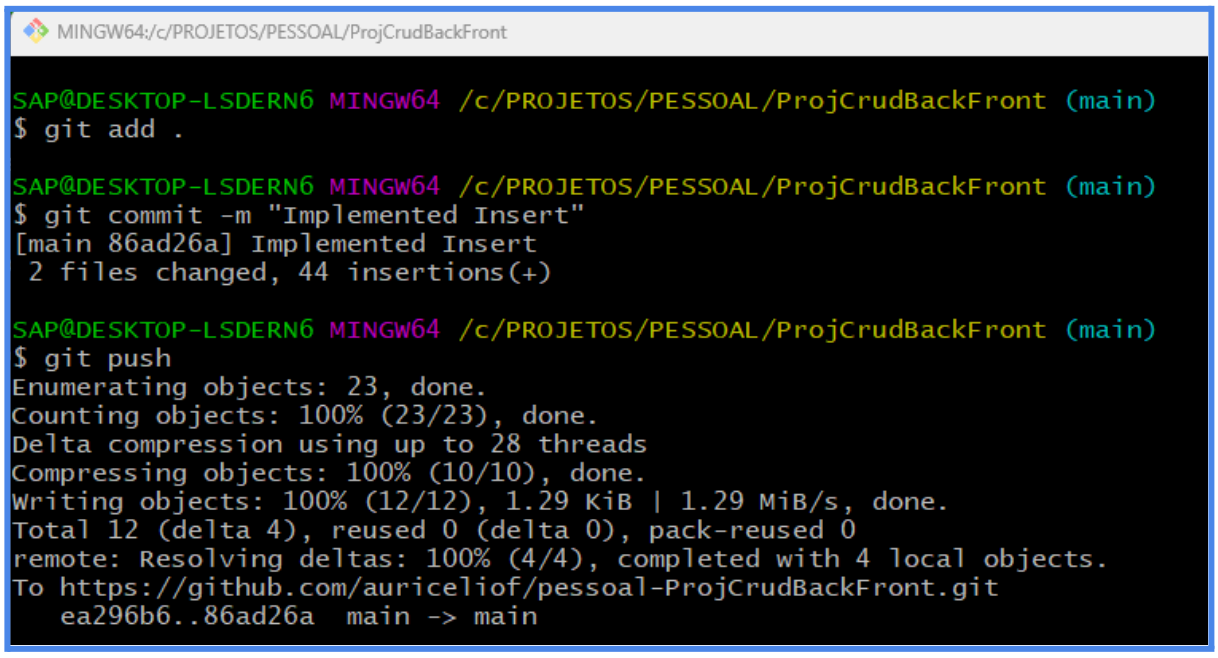


Buscar por Id



Github-12

- “Git bash here” no diretório do projeto
  - git add backend
  - git commit -m “Implemented Insert”
  - git push



ENDPOINT - UPDATE

ATUALIZAR ALUNO COM PUT

Implementar o update, no StudentController

StudentController.java

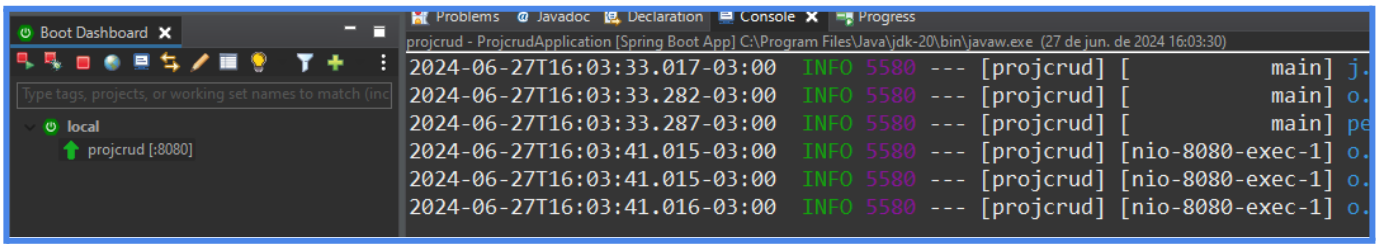
```
54
55    @PutMapping(value = "/{id}")
56    public ResponseEntity<StudentDTO> update(@PathVariable Long id, @RequestBody StudentDTO dto) {
57
58        dto = service.update(id, dto);
59
60        return ResponseEntity.ok().body(dto);
61    }
62 }
```

Implementar o método update, no StudentService

StudentService.java

```
53
54    public StudentDTO update(Long id, StudentDTO dto) {
55
56        Student entity = repository.getReferenceById(id);
57
58            entity.setName(dto.getName());
59            entity.setCpf(dto.getCpf());
60            entity.setBirthDate(dto.getBirthDate());
61            entity.setIncome(dto.getIncome());
62
63        entity = repository.save(entity);
64
65        return new StudentDTO(entity);
66    }
67 }
68
```

Rodar o projeto

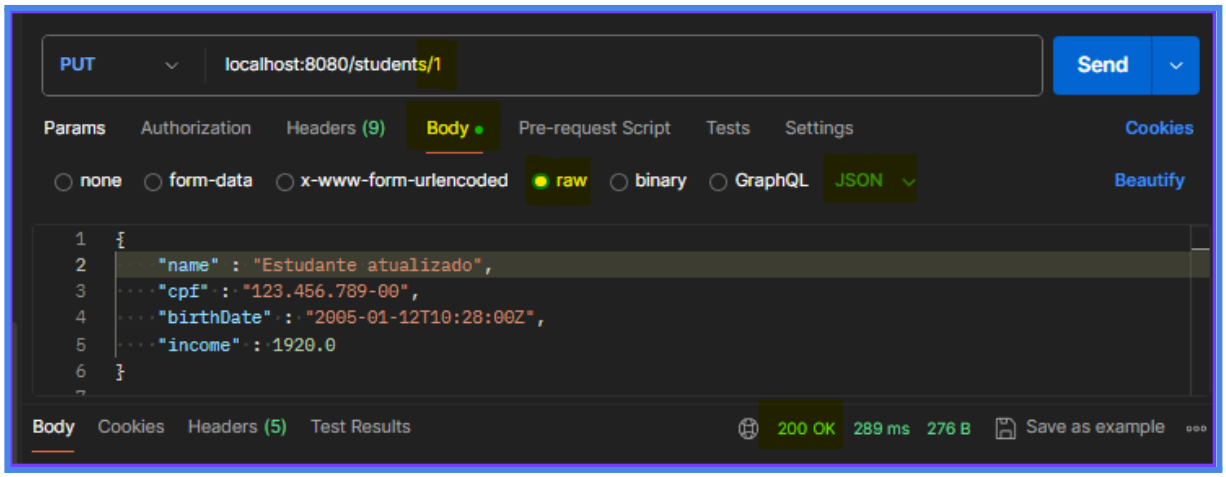


TESTAR NO POSTMAN

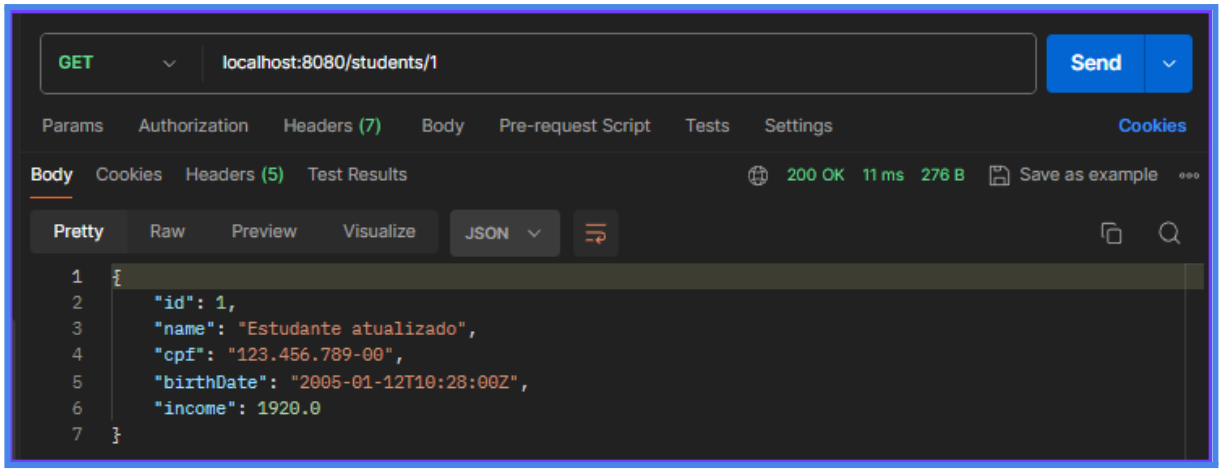
- *PUT: localhost:8080/students/1*
  - *Body:*

```
{
   "name" : "Aluno atualizado",
   "cpf" : "123.456.789-00",
   "birthDate" : "2005-01-12T10:28:00Z",
   "income" : 1920.0
}
```

Update



Busca por ID



TRATAMENTO DE ERRO PARA O UPDATE

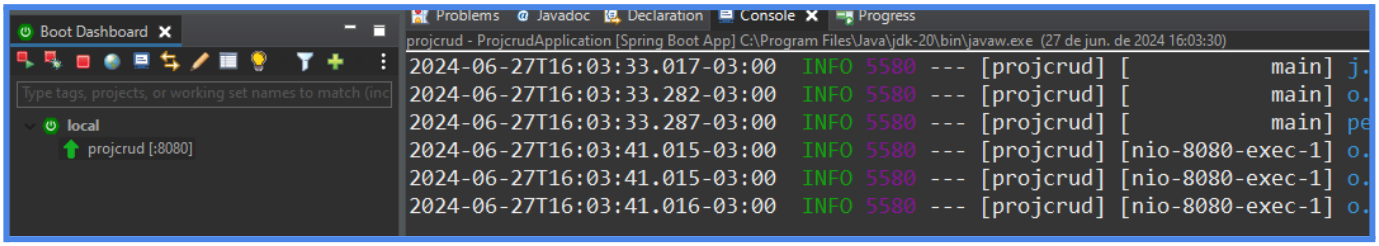
Implementar o tratamento para ID Não encontrado

StudentService.java

```
55      @Transactional
56      public StudentDTO update(Long id, StudentDTO dto) {
57          try {
58              Student entity = repository.getReferenceById(id);
59
60              entity.setName(dto.getName());
61              entity.setCpf(dto.getCpf());
62              entity.setBirthDate(dto.getBirthDate());
63              entity.setIncome(dto.getIncome());
64
65              entity = repository.save(entity);
66
67              return new StudentDTO(entity);
68          }
69          catch (EntityNotFoundException e) {
70              throw new ResourceNotFoundException("ID not found: " + id);
71          }
72      }
73  }
```

NOTA: Devemos colocar o método update num bloco “Try Catch”, pois ao atualizar um ID, este pode não existir.

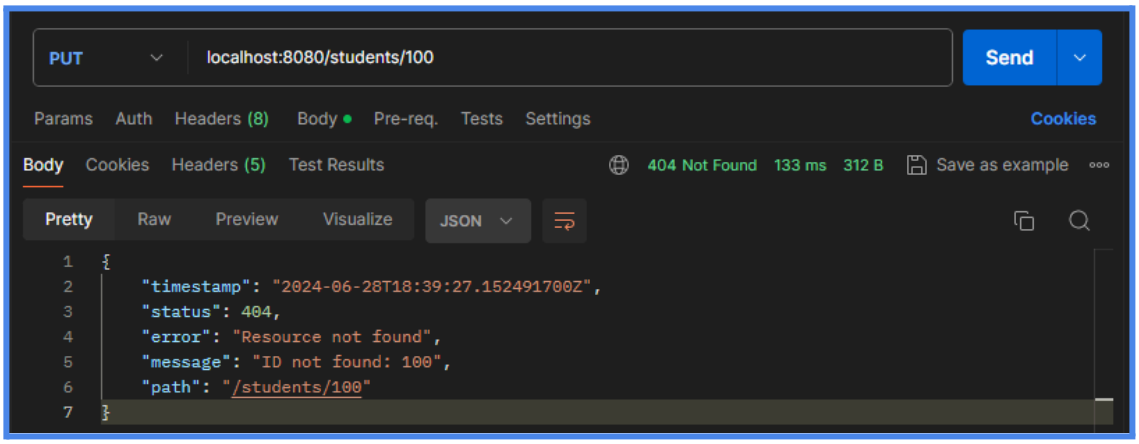
Rodar o projeto



TESTAR NO POSTMAN

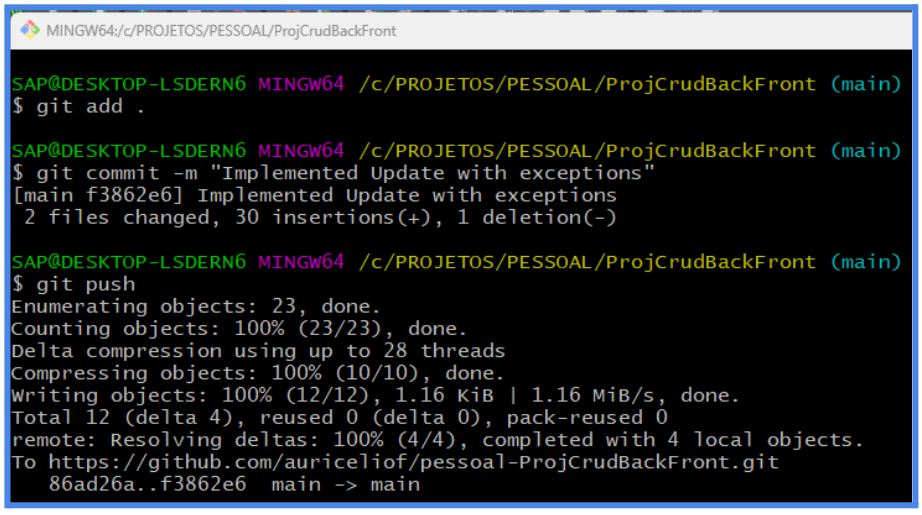
- *PUT: localhost:8080/students/100*

Update



Github-13

- “Git bash here” no diretório do projeto
  - git add backend
  - git commit -m “Implemented Update with exceptions”
  - git push



ENDPOINT - DELETE

DELETAR UM ALUNO COM O MÉTODO REST DELETE

Implementar o update, no StudentController

StudentController.java

```
63
64      @DeleteMapping(value = "/{id}")
65      public ResponseEntity<StudentDTO> delete(@PathVariable Long id) {
66
67          service.delete(id);
68
69          return ResponseEntity.noContent().build();
70      }
71  }
72
```

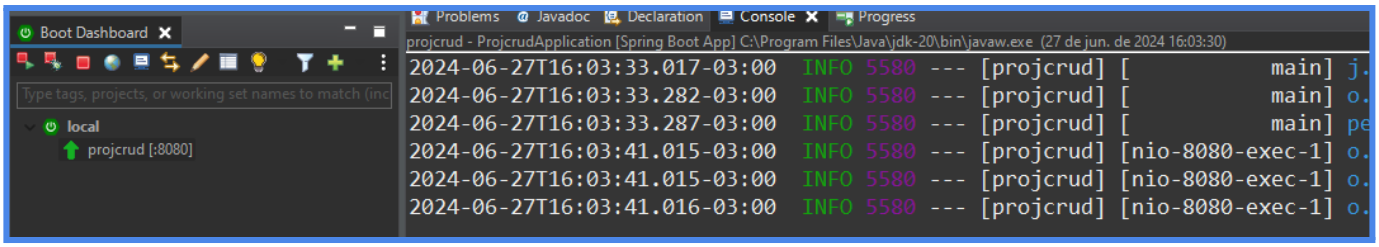
Implementar o método update, no StudentService

StudentService.java

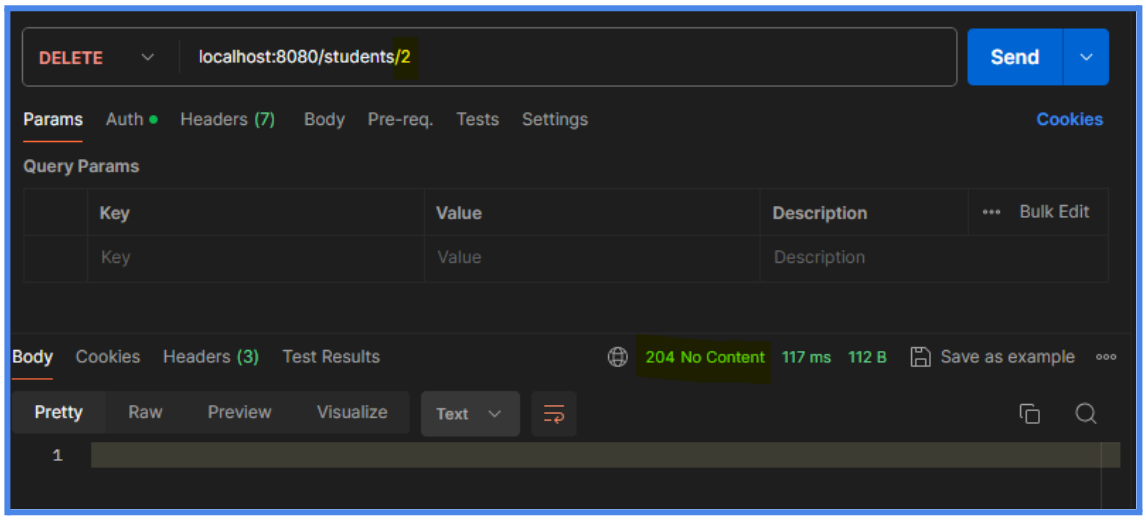
```
73
74      public void delete(Long id) {
75
76          repository.deleteById(id);
77      }
78  }
79
```

***NOTA**: No método delete não colocamos o @Transactional.*

Rodar o projeto



TESTAR NO POSTMAN



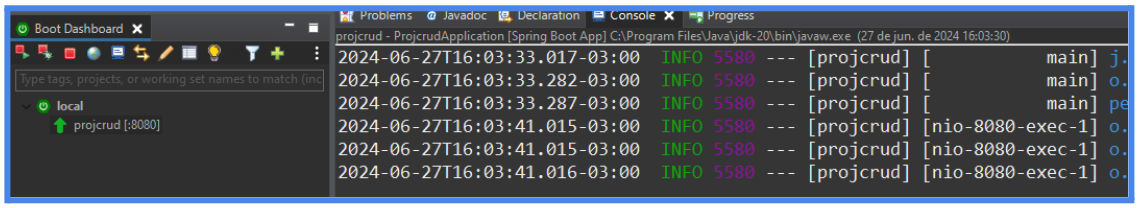
TRATAMENTO DE ERRO DO DELETE

Implementar o tratamento para ID Não encontrado

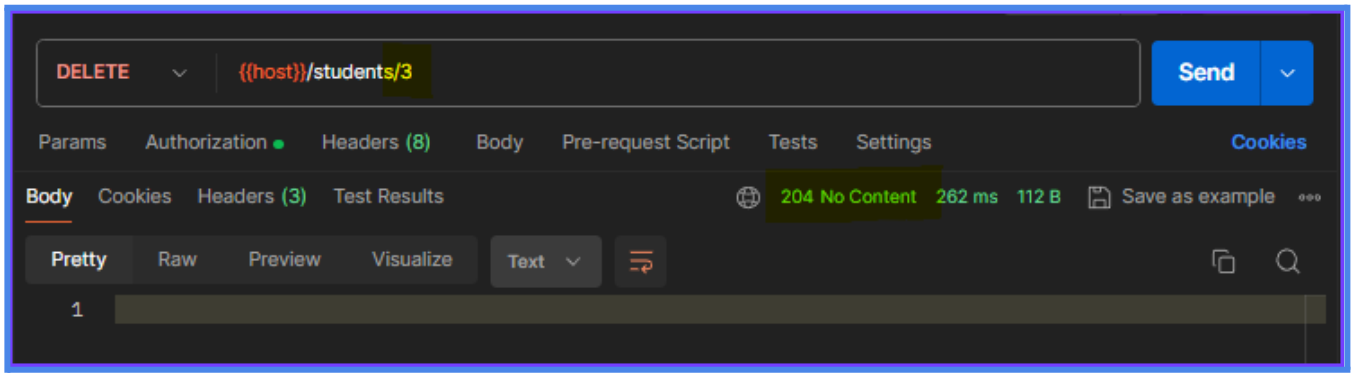
***NOTA**: Devemos colocar o método delete num bloco "Try Catch", pois ao deletar um ID, este pode não existir.*

***OBS**: Neste projeto, não iremos implementar o tratamento de erro para "integridade referencial", pois a classe student não se relaciona com nenhuma outra, ao menos por enquanto.*

Rodar o projeto

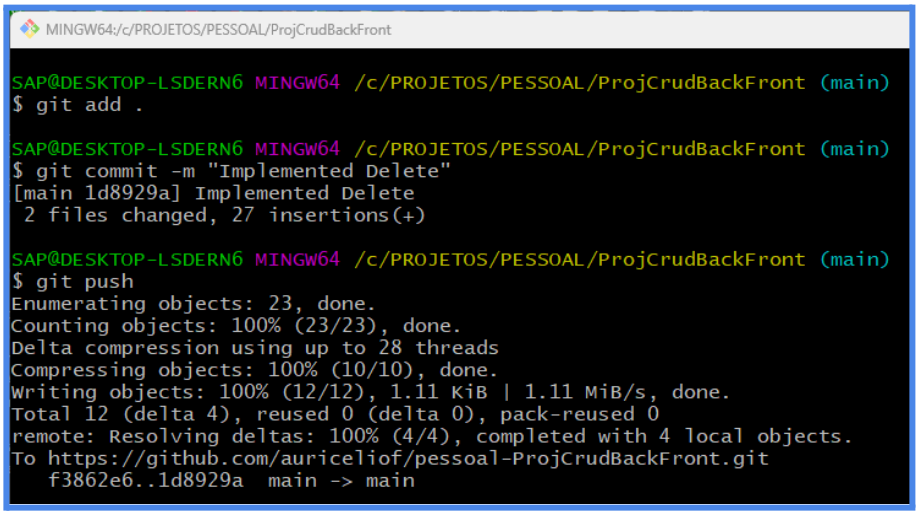


TESTAR NO POSTMAN



Github-14

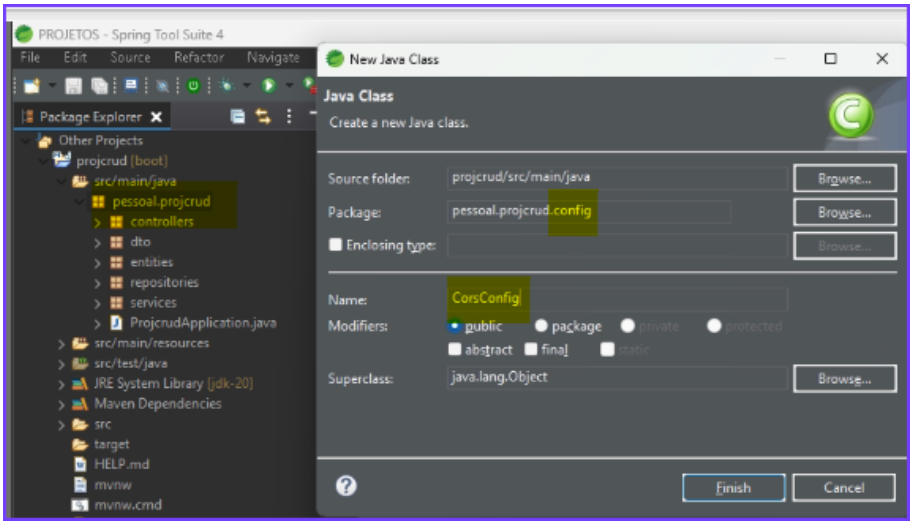
- “Git bash here” no diretório do projeto
  - git add backend
  - git commit -m “Implemented Delete”
  - git push



LIBERAÇÃO CORS PARA FRONTEND

IMPLEMENTAR O CONFIG

Criar o CorsConfig



Implementar o CorsConfig

CorsConfig.java

```java
package pessoal.projcrud.config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import org.springframework.web.filter.CorsFilter;

@Configuration
public class CorsConfig {

    @Bean
    public CorsFilter corsFilter() {
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        CorsConfiguration config = new CorsConfiguration();

        // Permitir solicitações de qualquer origem (use * para permitir de qualquer origem)
        config.addAllowedOrigin("*");

        // Permitir solicitações GET, POST, PUT, DELETE e OPTIONS
        config.addAllowedMethod("*");

        // Permitir os cabecalhos específicos que seu aplicativo React envia
        config.addAllowedHeader("*");

        source.registerCorsConfiguration("/**", config);
        return new CorsFilter(source);
    }
}
```

Github-15

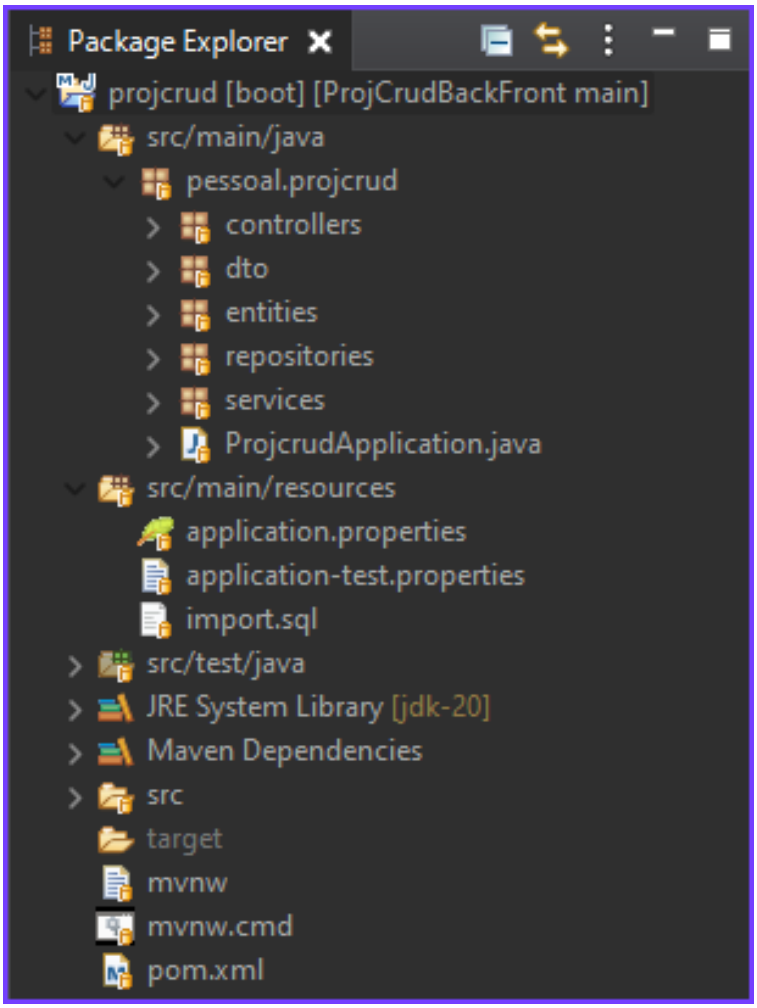
- “Git bash here” no diretório do projeto
    - git add backend
    - git commit -m “Implemented CORS”
    - git push

IMPLEMENTAR O SWAGGER

O objetivo deste capítulo é implementar o swagger no Spring Tool Suite 4.

O Swagger é uma ferramenta amplamente utilizada para criar, visualizar e manter a documentação de APIs de maneira automatizada.

PROJETO



NOTA: Neste momento, o projeto já deve estar em funcionamento, com todos os endpoints respondendo.

IMPLEMENTAÇÃO DO SWAGGER

DEPENDÊNCIA MAVEN

```
<dependency>
        <groupId>org.springdoc</groupId>
        <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
        <version>2.0.2</version>
</dependency>
```

projcrud/pom.xml

```
57          <dependency>
58              <groupId>org.springdoc</groupId>
59              <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
60              <version>2.0.2</version>
61          </dependency>
62      </dependencies>
```

IMPLEMENTAR O MAIN PRINCIPAL

```
@OpenAPIDefinition(info = @Info(title = "Projeto Swagger", version = "1", description = "Documentação para os Endpoints do Projeto CRUD"))
```

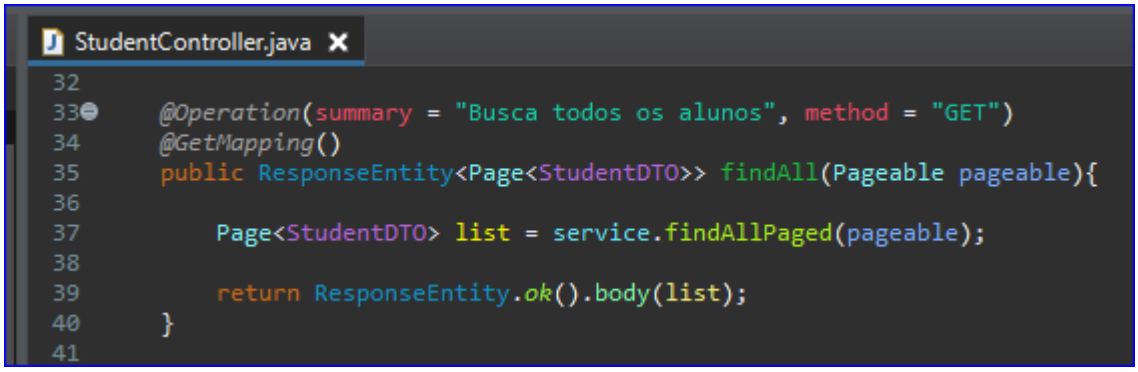
ProjcrudApplication.java

```
1  package pessoal.projcrud;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6  import io.swagger.v3.oas.annotations.OpenAPIDefinition;
7  import io.swagger.v3.oas.annotations.info.Info;
8
9  @SpringBootApplication
10 @OpenAPIDefinition(info = @Info(title = "Projeto Swagger", version = "1", description = "Documentação para os Endpoints do Projeto CRUD"))
11 public class ProjcrudApplication {
12
13     public static void main(String[] args) {
14         SpringApplication.run(ProjcrudApplication.class, args);
15     }
16
17 }
18
```

IMPLEMENTAR O CONTROLLER

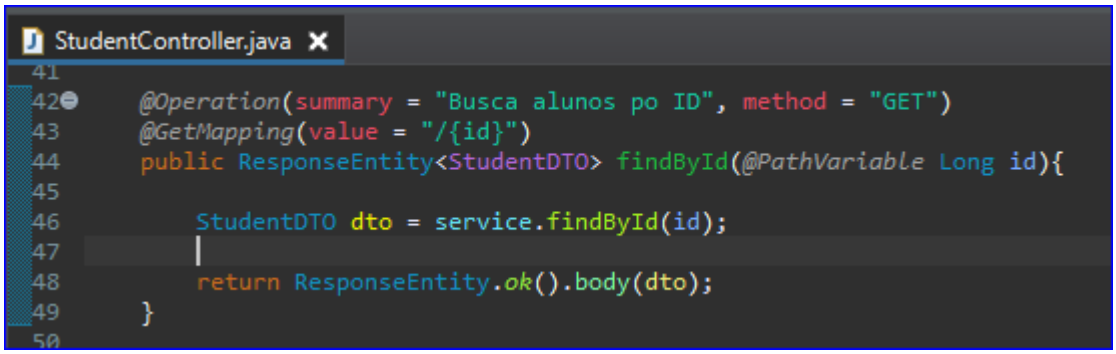
FindAll

@Operation(summary = "Busca todos os alunos", method = "GET")



FindById

@Operation(summary = "Busca alunos po ID", method = "GET")

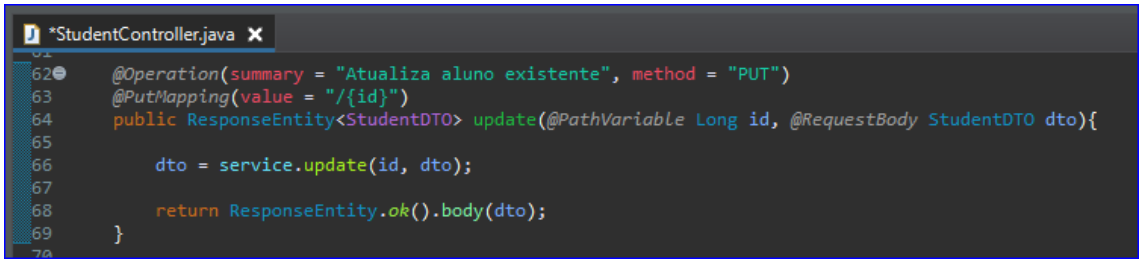


Insert

@Operation(summary = "Cadastra novos alunos", method = "POST")

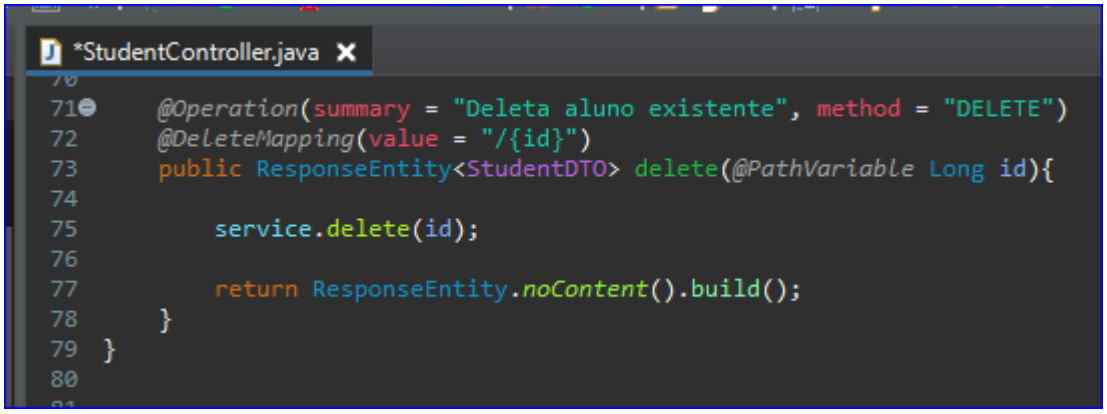
Update

@Operation(summary = "Atualiza aluno existente", method = "PUT")



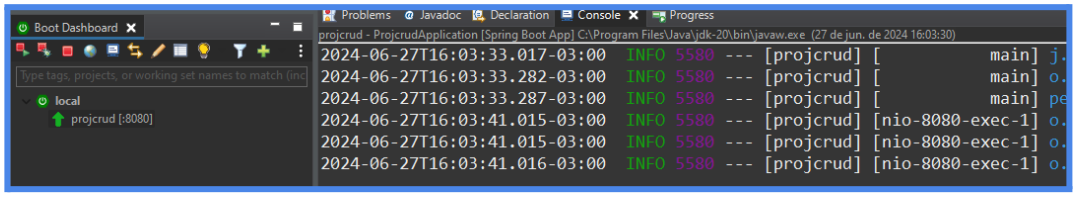
Delete

@Operation(summary = "Deleta aluno existente", method = "DELETE")



ACESSAR O SWAGGER

Rodar o projeto

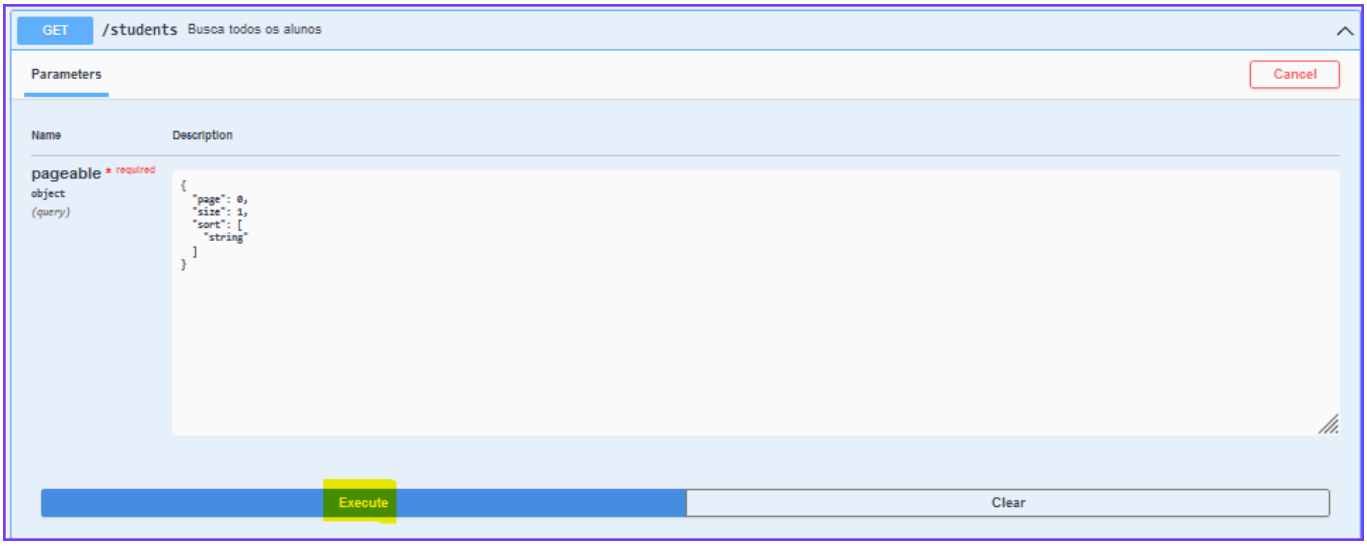


Acesso online

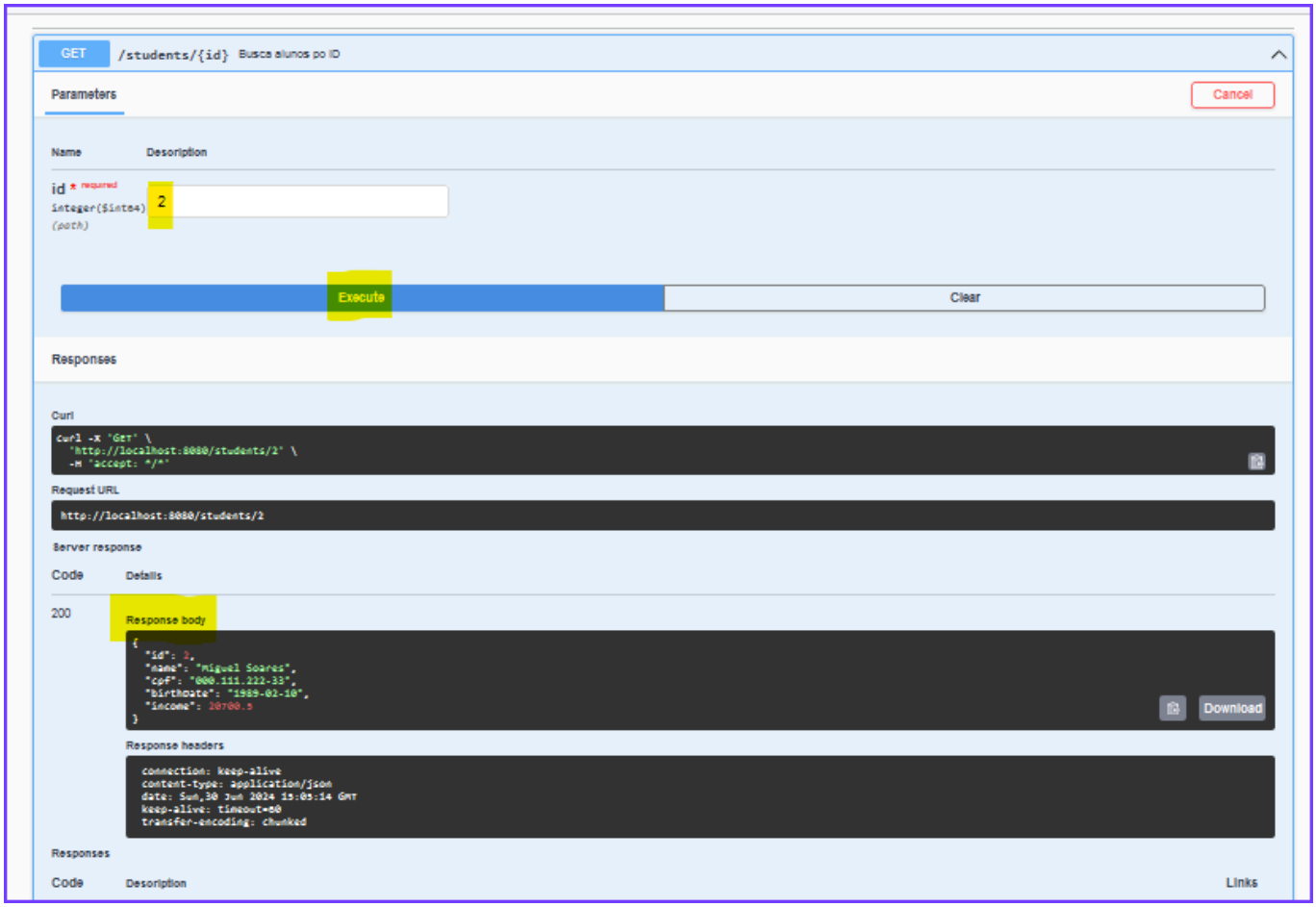
- http://localhost:8080/swagger-ui/index.html#

Testar o Swagger

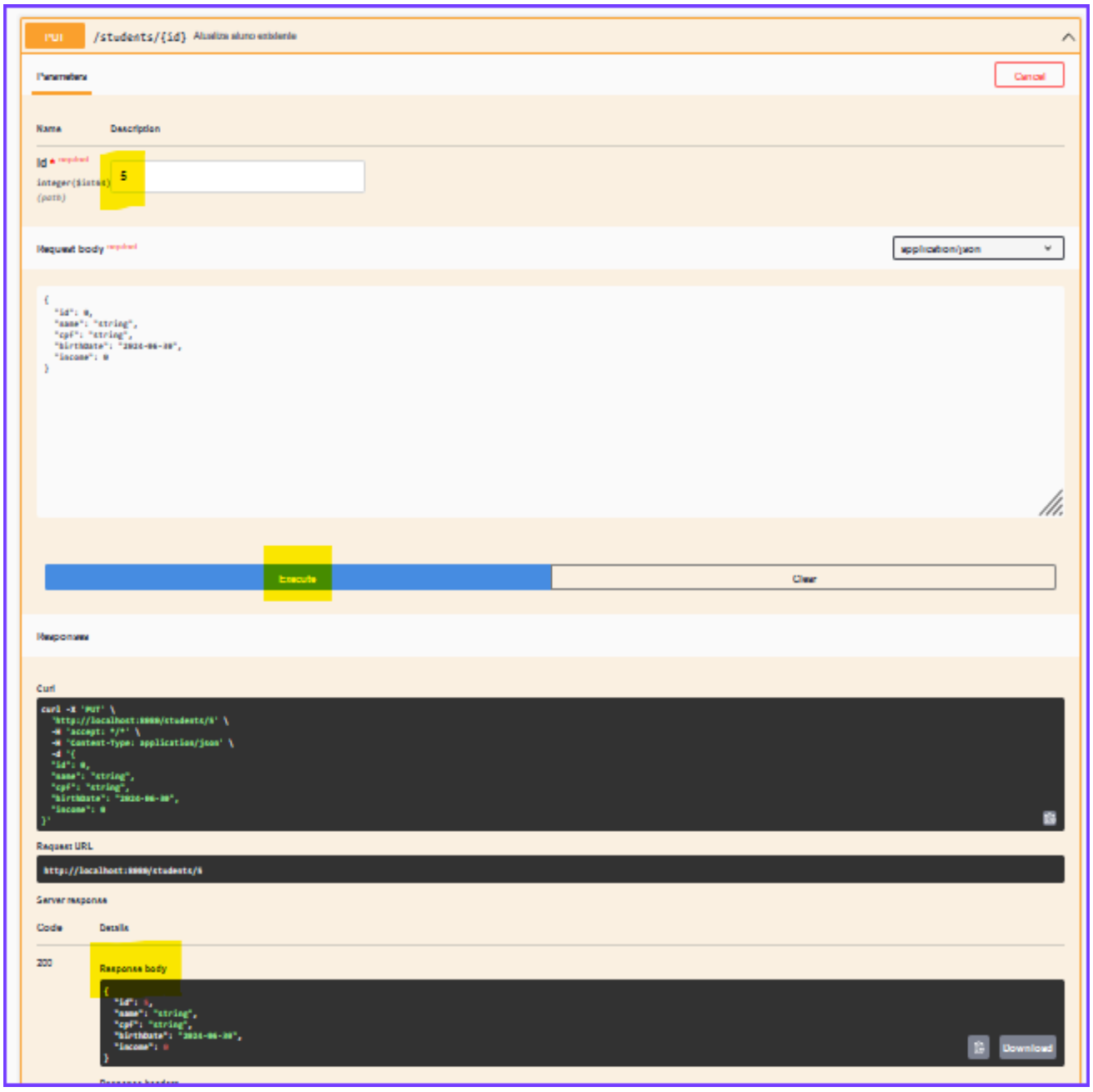
Buscar todos



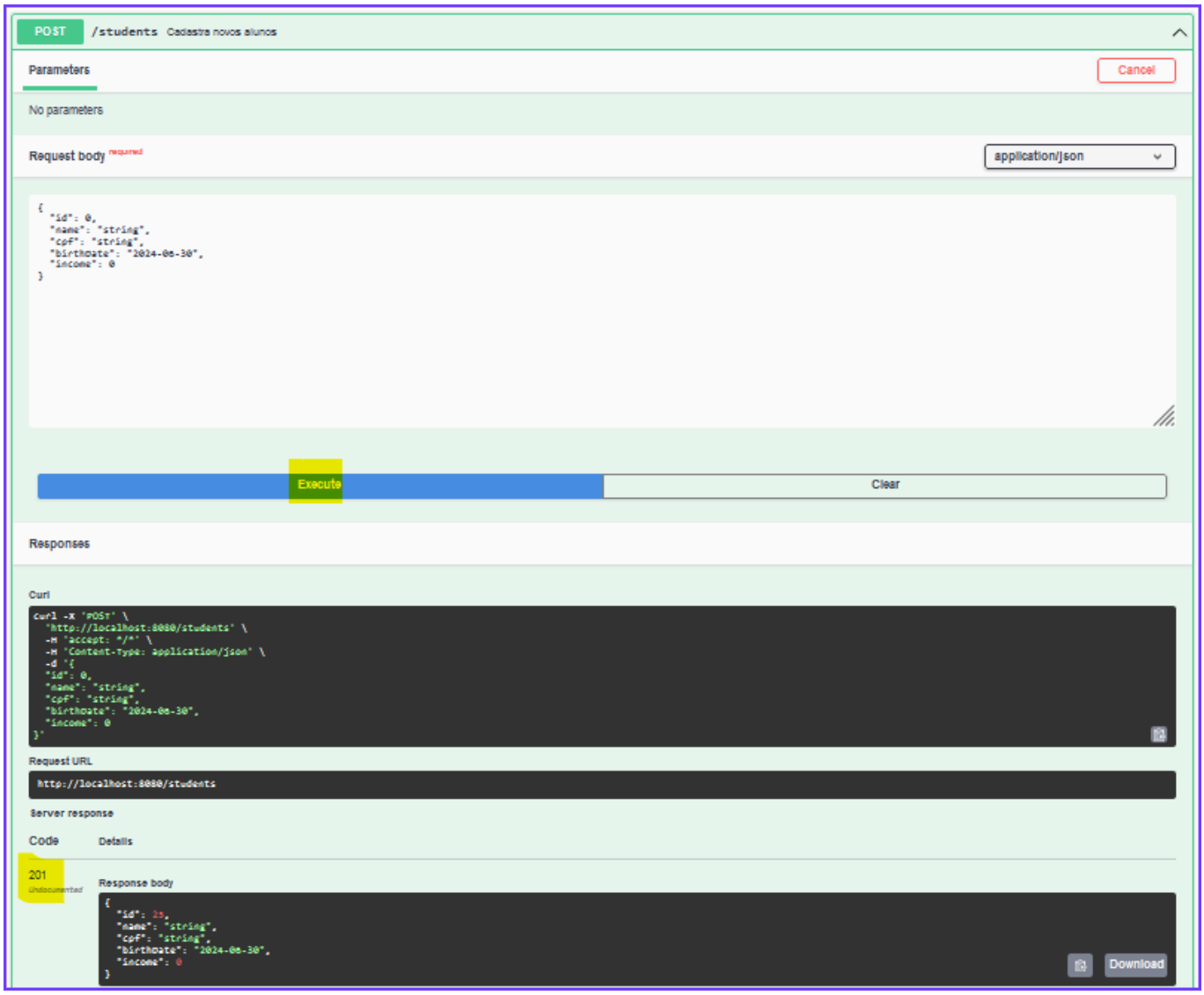
Busca por ID


GET
/students/{id}
Busca alunos po ID
Parameters
Cancel
Name
Description
id * required
integer($int64)
(path)
2
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://localhost:8080/students/2' \
-H 'accept: */*'
Request URL
http://localhost:8080/students/2
Server response
Code
Details
200
Response body
{
"id": 2,
"name": "Miguel Soares",
"cpf": "000.111.222-33",
"birthdate": "1989-02-10",
"income": 20700.5
}
Download
Response headers
connection: keep-alive
content-type: application/json
date: Sun,30 Jun 2024 15:05:14 GMT
keep-alive: timeout=60
transfer-encoding: chunked
Responses
Code
Description
Links

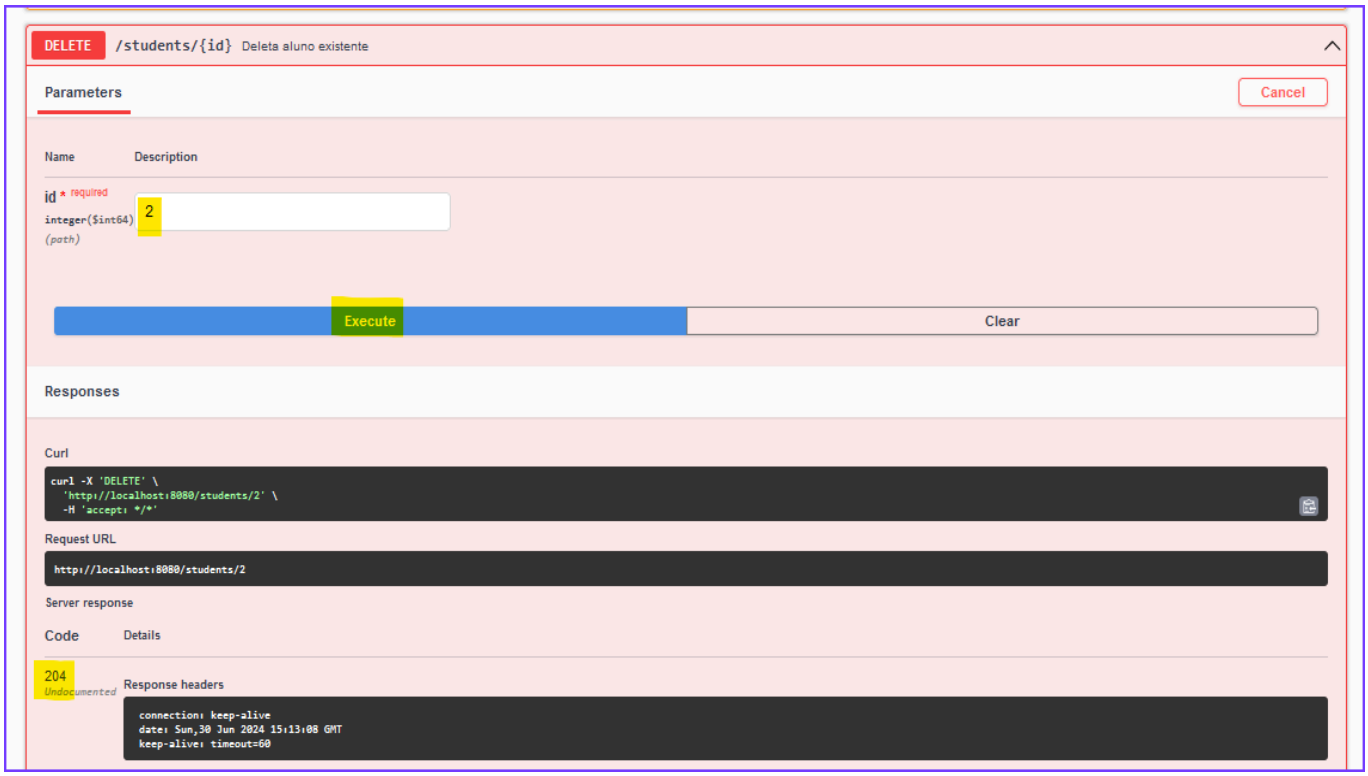
Atualizar


PUT
/students/{id}
Cancel
Name
Description
id
5
Request body
application/json
{
"id": 0,
"name": "string",
"cpf": "string",
"birthDate": "2026-06-08",
"income": 0
}
Execute
Clear
Responses
Curl
curl -X 'PUT' \
'http://localhost:8080/students/5' \
-H 'accept: */*' \
-H 'content-type: application/json' \
-d '{
"id": 0,
"name": "string",
"cpf": "string",
"birthDate": "2026-06-08",
"income": 0
}'
Request URL
http://localhost:8080/students/5
Server response
Code
Details
200
Response body
{
"id": 0,
"name": "string",
"cpf": "string",
"birthDate": "2026-06-08",
"income": 0
}
Download

Inserir


POST /students Cadastra novos alunos
Parameters
Cancel
No parameters
Request body required
application/json
{
"id": 0,
"name": "string",
"cpf": "string",
"birthdate": "2024-06-30",
"income": 0
}
Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://localhost:8080/students' \
-H 'accept: */*' \
-H 'Content-Type: application/json' \
-d '{
"id": 0,
"name": "string",
"cpf": "string",
"birthdate": "2024-06-30",
"income": 0
}'
Request URL
http://localhost:8080/students
Server response
Code
Details
201
Response body
{
"id": 15,
"name": "string",
"cpf": "string",
"birthdate": "2024-06-30",
"income": 0
}
Download

Deletar



Github-16

- “Git bash here” no diretório do projeto
  - git add backend
  - git commit -m “Implemented SWAGGER”
  - git push

Fim