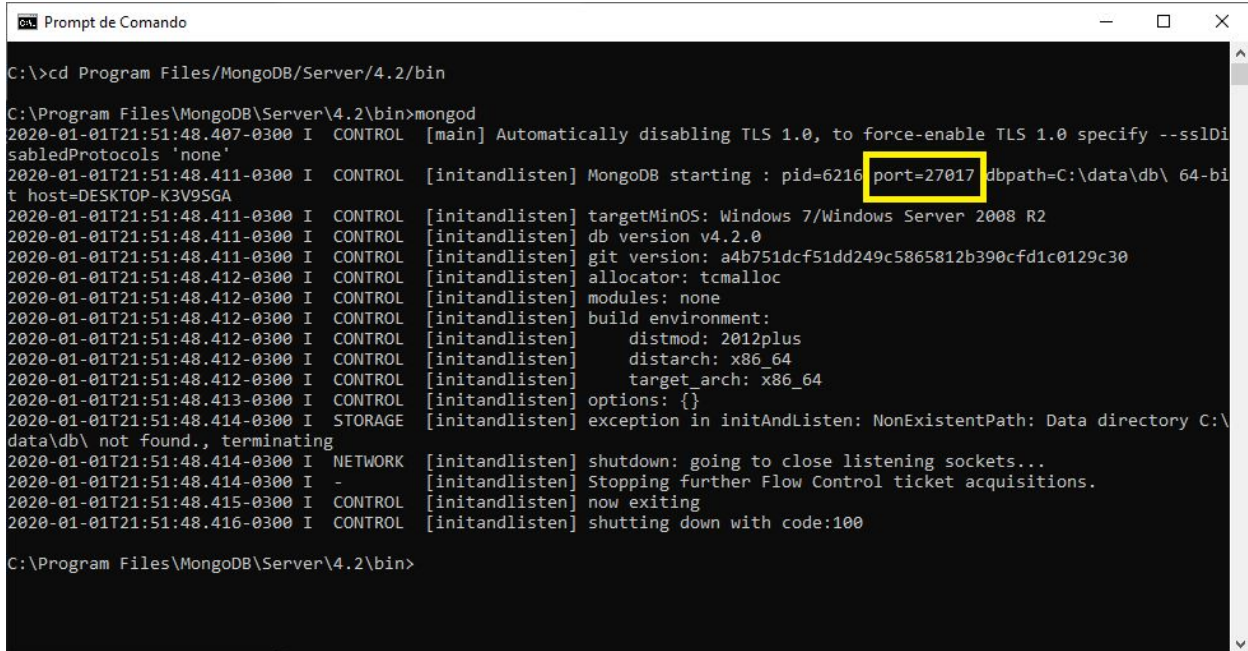


1 - Primeiramente, devemos nos certificar de que o MongoDB esteja instalado na máquina pelo terminal e para isso digite o comando “mongod”, que é o servidor do MongoDB, apontando para o diretório bin onde foi instalado (não se esqueça que após a instalação, configurar o path). Repare na porta que estiver sendo executado;



```
C:\>cd Program Files/MongoDB/Server/4.2/bin

C:\Program Files\MongoDB\Server\4.2\bin>mongod
2020-01-01T21:51:48.407-0300 I CONTROL [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'
2020-01-01T21:51:48.411-0300 I CONTROL [initandlisten] MongoDB starting : pid=6216 port=27017 dbpath=C:\data\db\ 64-bit host=DESKTOP-K3V9SGA
2020-01-01T21:51:48.411-0300 I CONTROL [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2020-01-01T21:51:48.411-0300 I CONTROL [initandlisten] db version v4.2.0
2020-01-01T21:51:48.411-0300 I CONTROL [initandlisten] git version: a4b751dcf51dd249c5865812b390cfd1c0129c30
2020-01-01T21:51:48.412-0300 I CONTROL [initandlisten] allocator: tcmalloc
2020-01-01T21:51:48.412-0300 I CONTROL [initandlisten] modules: none
2020-01-01T21:51:48.412-0300 I CONTROL [initandlisten] build environment:
2020-01-01T21:51:48.412-0300 I CONTROL [initandlisten] distmod: 2012plus
2020-01-01T21:51:48.412-0300 I CONTROL [initandlisten] distarch: x86_64
2020-01-01T21:51:48.412-0300 I CONTROL [initandlisten] target_arch: x86_64
2020-01-01T21:51:48.413-0300 I CONTROL [initandlisten] options: {}
2020-01-01T21:51:48.414-0300 I STORAGE [initandlisten] exception in initAndListen: NonExistentPath: Data directory C:\data\db\ not found., terminating
2020-01-01T21:51:48.414-0300 I NETWORK [initandlisten] shutdown: going to close listening sockets...
2020-01-01T21:51:48.414-0300 I - [initandlisten] Stopping further Flow Control ticket acquisitions.
2020-01-01T21:51:48.415-0300 I CONTROL [initandlisten] now exiting
2020-01-01T21:51:48.416-0300 I CONTROL [initandlisten] shutting down with code:100

C:\Program Files\MongoDB\Server\4.2\bin>
```

Figura 1

2 - Criar um novo projeto Spring no STS:

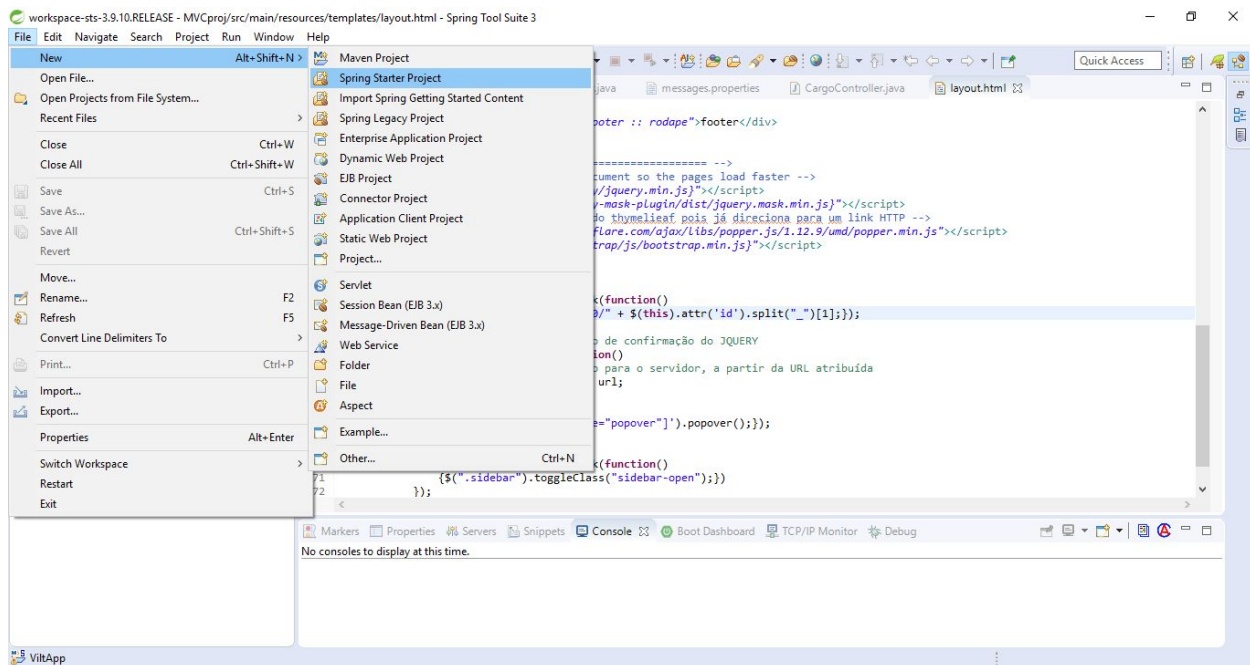
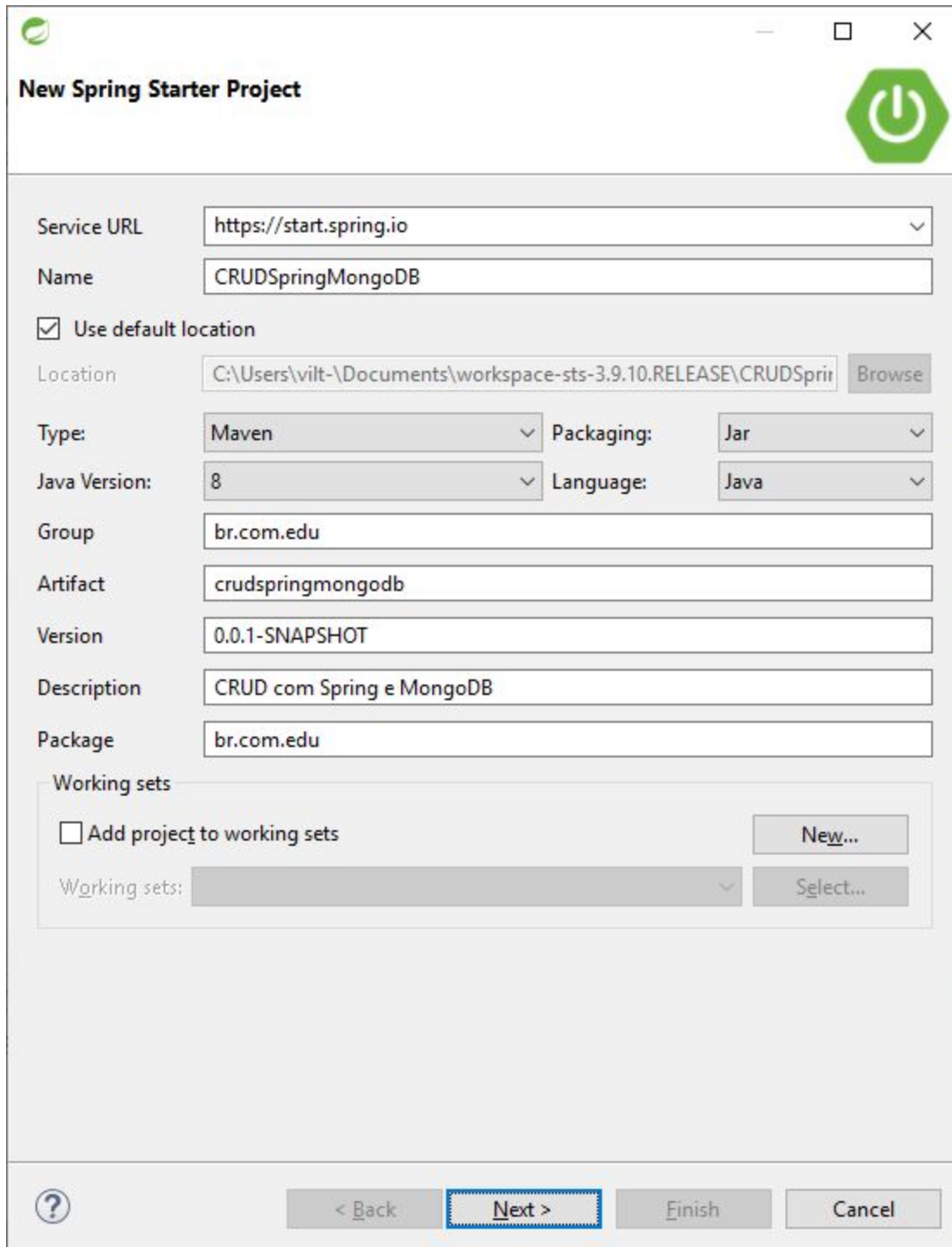


Figura 2



The image shows a 'New Spring Starter Project' dialog box. At the top, there is a title bar with a green icon on the left and standard window controls on the right. Below the title bar, the text 'New Spring Starter Project' is displayed on the left, and a large green power button icon is on the right. The main area contains several input fields and checkboxes. The 'Service URL' is set to 'https://start.spring.io'. The 'Name' field contains 'CRUDSpringMongoDB'. There is a checked checkbox for 'Use default location'. The 'Location' field shows a file path 'C:\Users\vilt-\Documents\workspace-sts-3.9.10.RELEASE\CRUDSprin' with a 'Browse' button to its right. Below this, there are two rows of dropdown menus: 'Type:' with 'Maven' selected, 'Packaging:' with 'Jar' selected, 'Java Version:' with '8' selected, and 'Language:' with 'Java' selected. Further down are text fields for 'Group' (br.com.edu), 'Artifact' (crudspringmongodb), 'Version' (0.0.1-SNAPSHOT), 'Description' (CRUD com Spring e MongoDB), and 'Package' (br.com.edu). A 'Working sets' section at the bottom includes an unchecked checkbox 'Add project to working sets', a 'New...' button, a 'Working sets:' dropdown menu, and a 'Select...' button. At the very bottom, there is a row of buttons: a help icon (?), '< Back', 'Next >' (which is highlighted with a blue dashed border), 'Finish', and 'Cancel'.

**New Spring Starter Project**

Service URL:

Name:

☒ Use default location

Location:

Type:  Packaging:

Java Version:  Language:

Group:

Artifact:

Version:

Description:

Package:

Working sets

☐ Add project to working sets

Working sets:

Figura 3

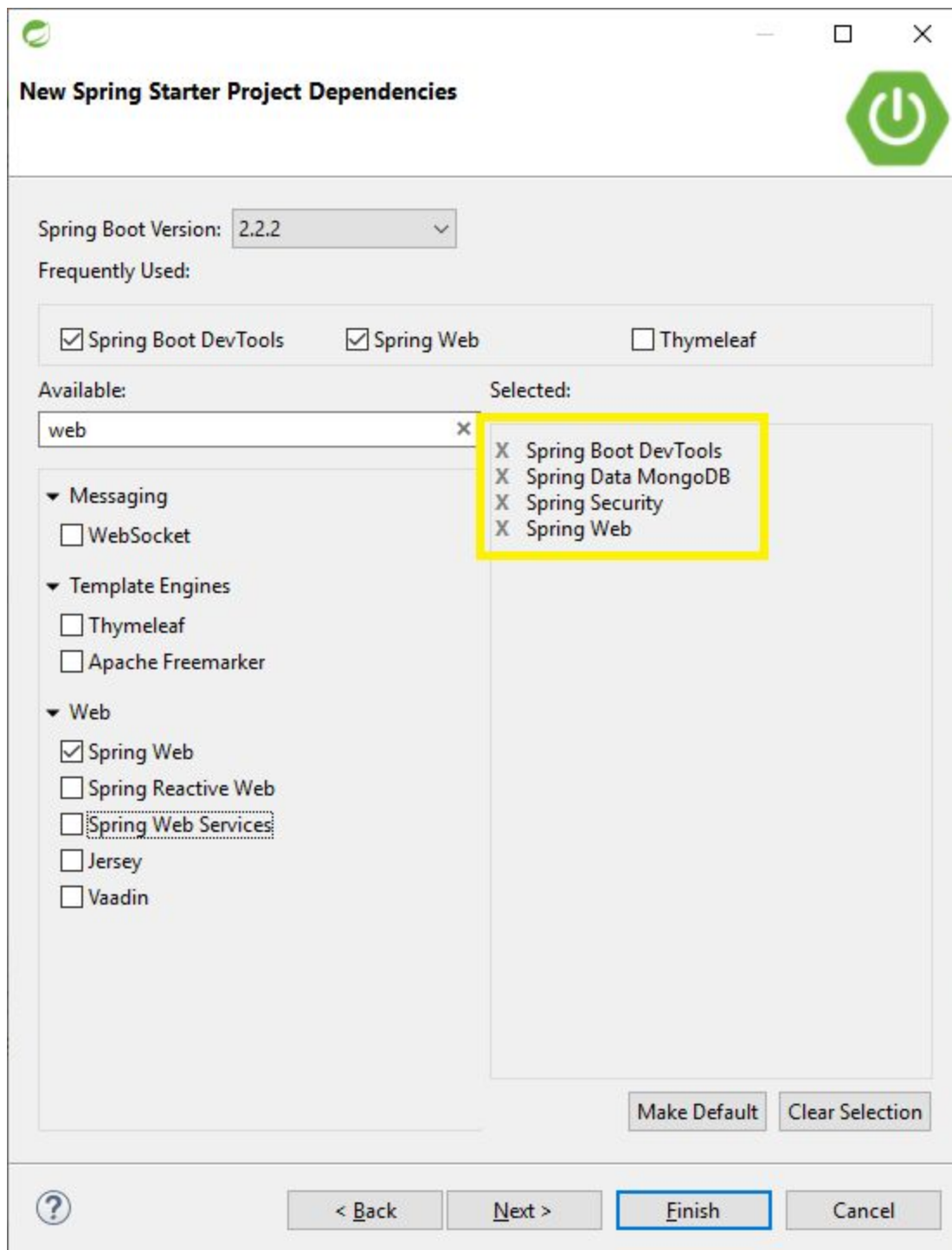


Figura 4

### 3 - Definir o nome do banco de dados no MongoDB no Eclipse:

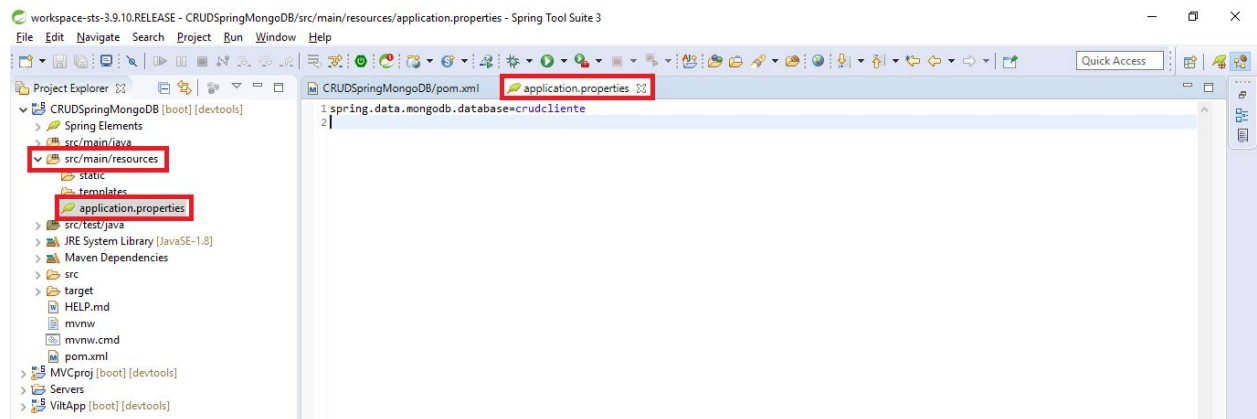


Figura 5

### 4 - Criar a classe Entidade de domínio. A classe que o spring gera por padrão para inicializar a aplicação, não é necessário fazer nenhuma alteração nele. Crie a classe no pacote que deseja:

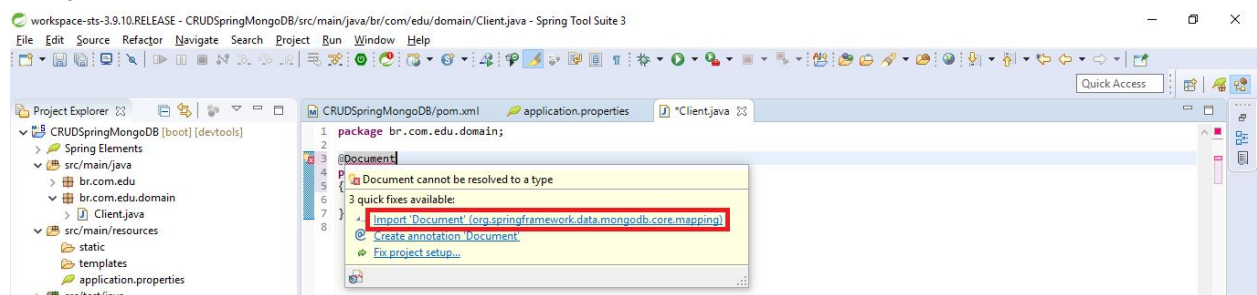


Figura 6

### 5 - Criar os atributos da classe de domínio e seus métodos padrões:

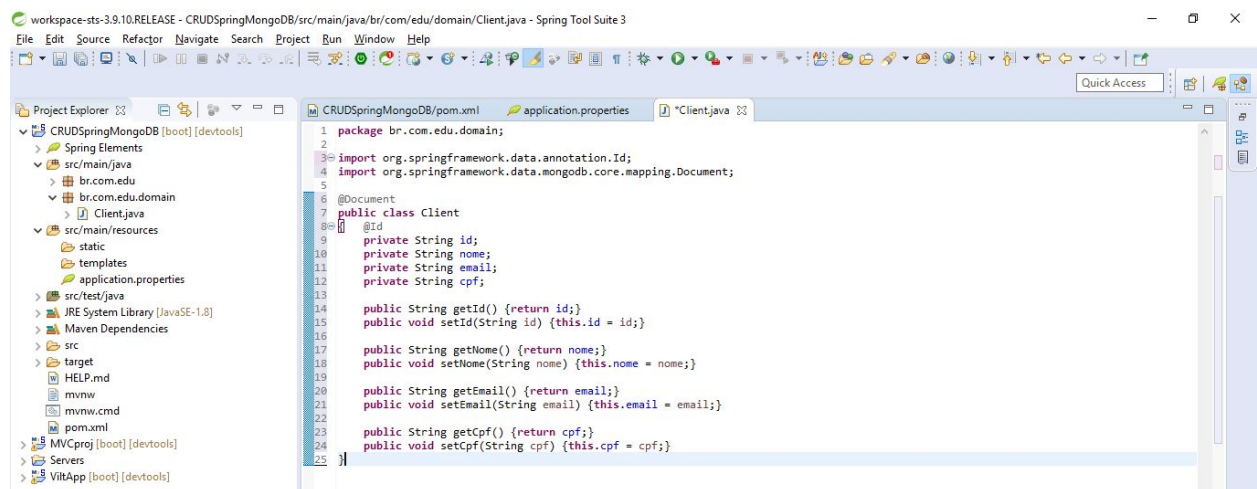


Figura 7



6 - Criar um repositório (interface) para que tenhamos acesso a esse *document* do MongoDB.

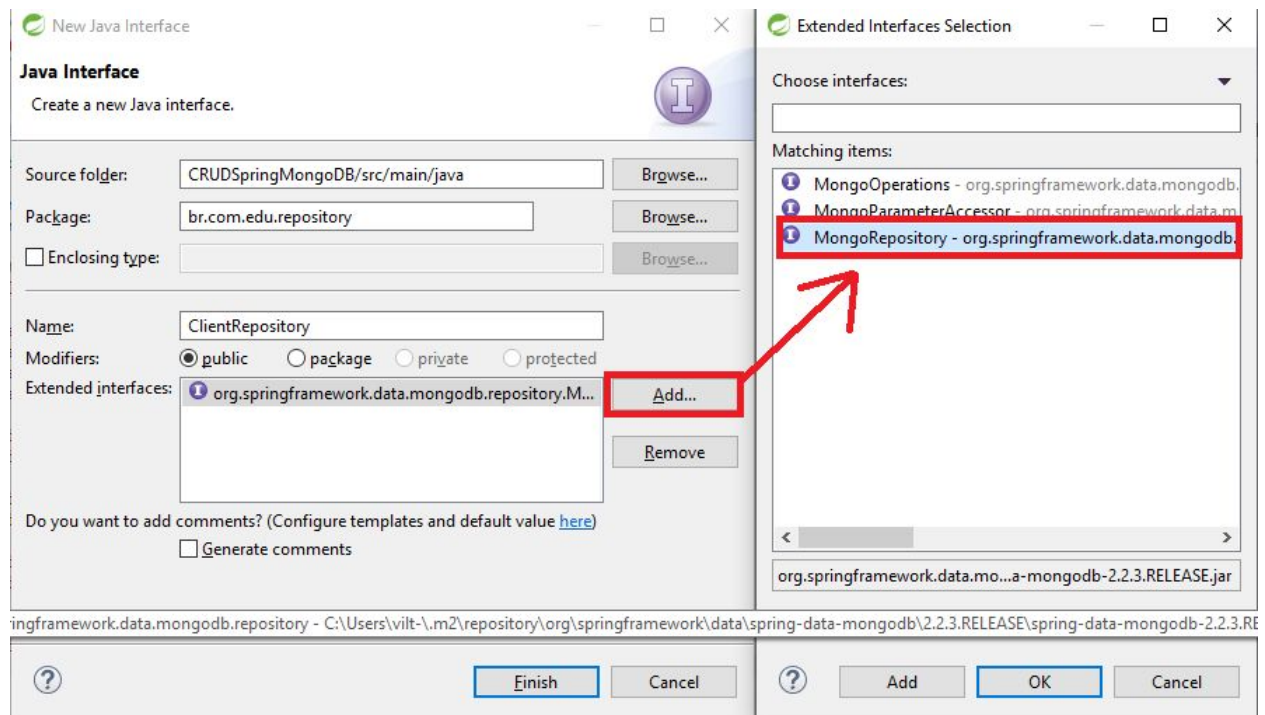


Figura 8

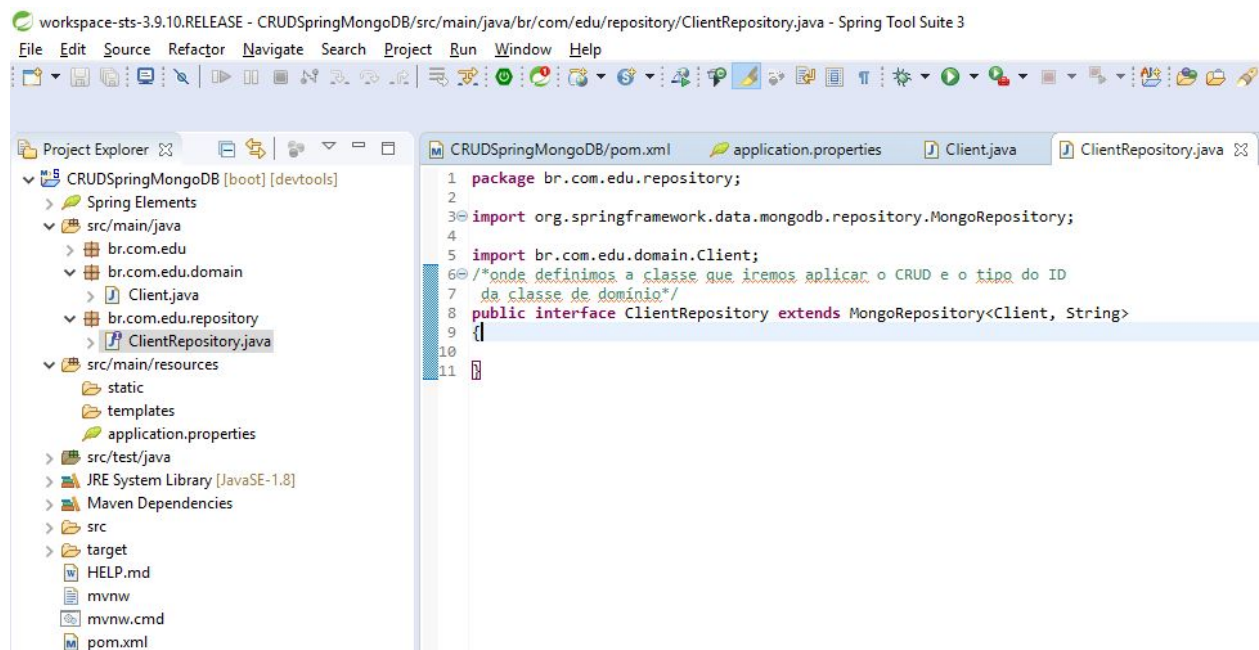


Figura 9

7 - Criar uma camada de serviço que será responsável por criar as ações que irão fazer a persistência e os acessos ao banco de dados:

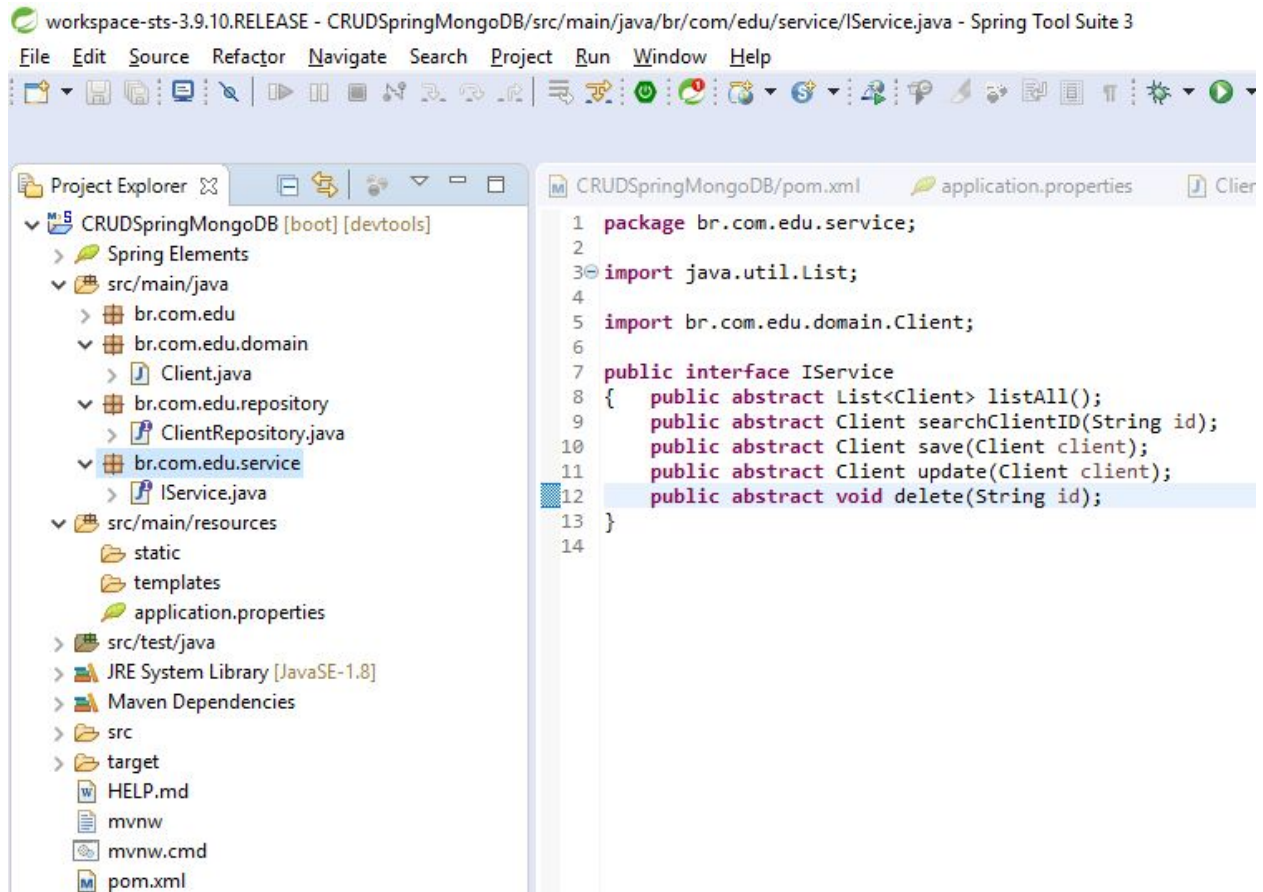


Figura 10

8 - Criar a classe de serviço, no mesmo pacote da interface serviço:

**New Java Class**

Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

---

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass:

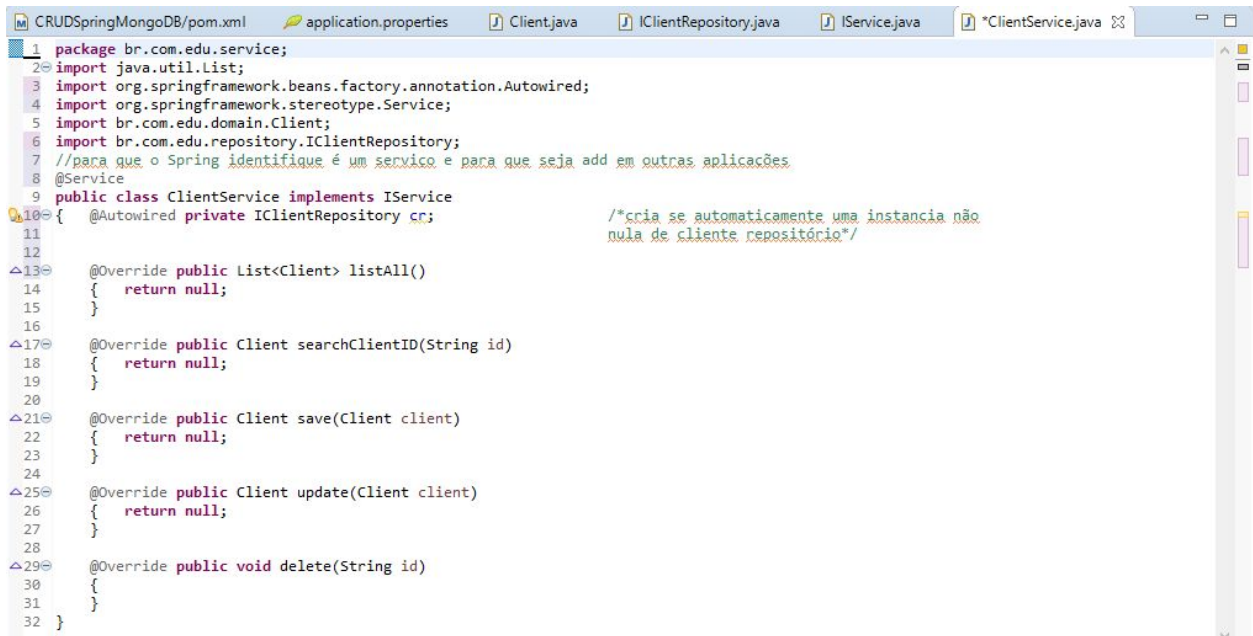
Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)  
☐ Constructors from superclass  
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))  
☐ Generate comments

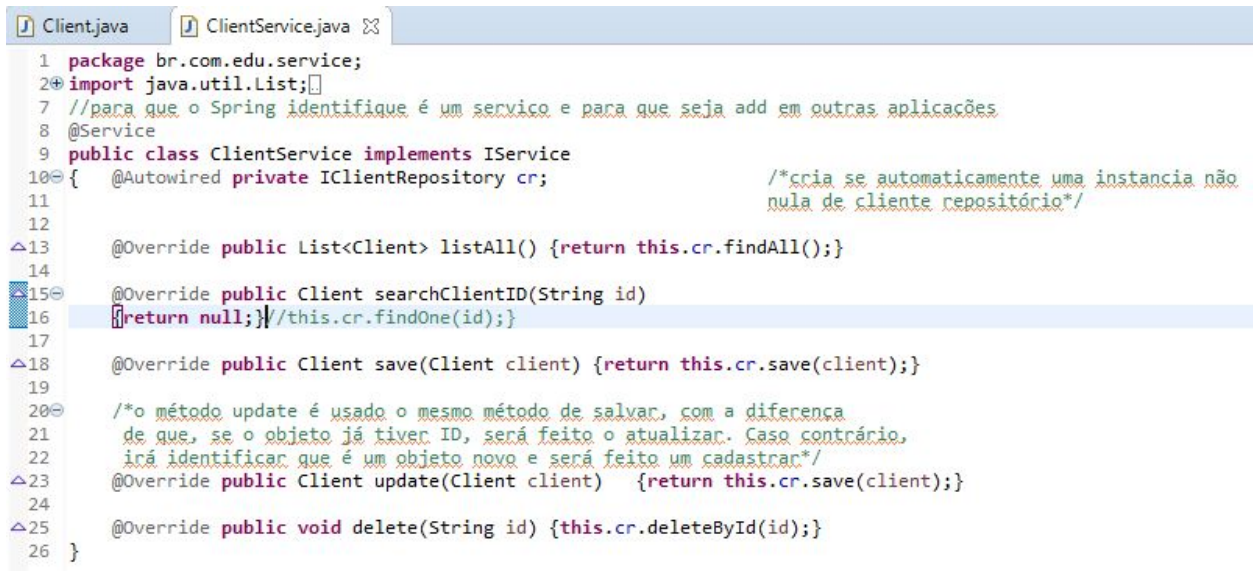
Figura 11



```
1 package br.com.edu.service;
2 import java.util.List;
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Service;
5 import br.com.edu.domain.Client;
6 import br.com.edu.repository.IClientRepository;
7 //para que o Spring identifique é um serviço e para que seja add em outras aplicações
8 @Service
9 public class ClientService implements IService
10 { @Autowired private IClientRepository cr; /*cria se automaticamente uma instancia não
11                                          nula de cliente repositório*/
12
13 @Override public List<Client> listAll()
14 { return null;
15 }
16
17 @Override public Client searchClientID(String id)
18 { return null;
19 }
20
21 @Override public Client save(Client client)
22 { return null;
23 }
24
25 @Override public Client update(Client client)
26 { return null;
27 }
28
29 @Override public void delete(String id)
30 {
31 }
32 }
```

Figura 12

## 9 - Implementar os métodos da classe de serviço do CRUD:



```
1 package br.com.edu.service;
2 import java.util.List;
3
4 //para que o Spring identifique é um serviço e para que seja add em outras aplicações
5 @Service
6 public class ClientService implements IService
7 { @Autowired private IClientRepository cr; /*cria se automaticamente uma instancia não
8                                          nula de cliente repositório*/
9
10 @Override public List<Client> listAll() {return this.cr.findAll();}
11
12 @Override public Client searchClientID(String id)
13 {return null;}/this.cr.findOne(id);}
14
15 @Override public Client save(Client client) {return this.cr.save(client);}
16
17 /*o método update é usado o mesmo método de salvar, com a diferença
18 de que, se o objeto já tiver ID, será feito o atualizar. Caso contrário,
19 irá identificar que é um objeto novo e será feito um cadastrar*/
20 @Override public Client update(Client client) {return this.cr.save(client);}
21
22 @Override public void delete(String id) {this.cr.deleteById(id);}
23 }
```

Figura 13



## 10 - Criar a classe de controle:

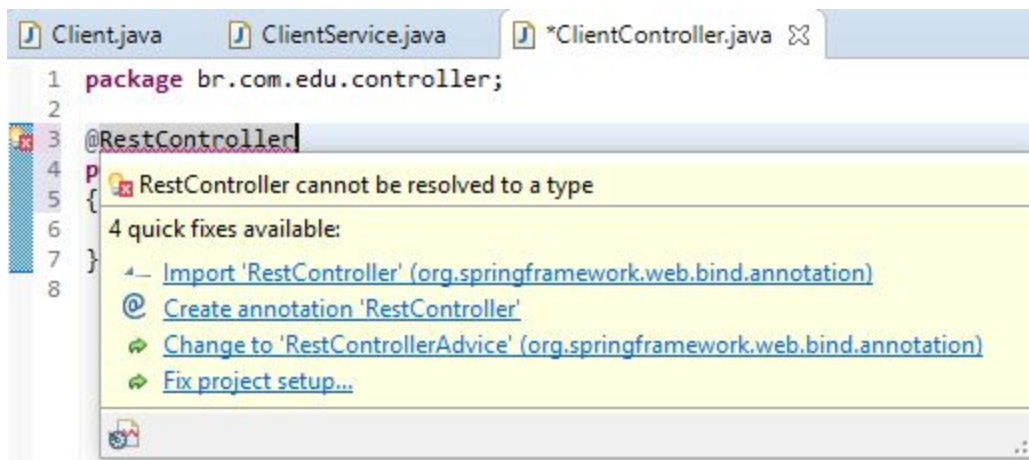


Figura 14

```
package br.com.edu.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import br.com.edu.domain.Client;
import br.com.edu.service.ClientService;
//This notation allows any class be a Rest type
@RestController//this path will be our app base path, it means that it will be
@RequestMapping(path="/api/clientes")//our URL
public class ClientController
{
    @Autowired private ClientService service;

    //método que usará e será feito dentro de um objeto próprio do Spring
    //da classe chamado ResponseEntity, que ajuda a retornar os dados da
    //da comunicação. Está definido como um tipo GET e não será definido um
    //path, pois estamos assumindo que o path do RequestMapping será usado
    //como padrão para retornar essa informação.
    @GetMapping public ResponseEntity<List<Client>> listAllClient()
    {return ResponseEntity.ok(this.service.listAll());}

    //como será feita listagem pelo ID, então será criado um path. As chaves indicam
    //que o id é um valor dinâmico.e estará na URL. Toda vez que fizer uma requisição
    //com "/api/clientes/id", será usado este {id}
    @GetMapping(path="/{id}") public ResponseEntity<Client> listClientID(@PathVariable(name="id") String id)
    {return ResponseEntity.ok(this.service.searchClientID(id));}

    //o @RequestBody faz com que o spring extraia automaticamente dados vindo
    //do PostRequest e criará um objeto do tipo Client automaticamente para nós
    @PostMapping public ResponseEntity<Client> save(@RequestBody Client client)
    {return ResponseEntity.ok(this.service.save(client));}
```

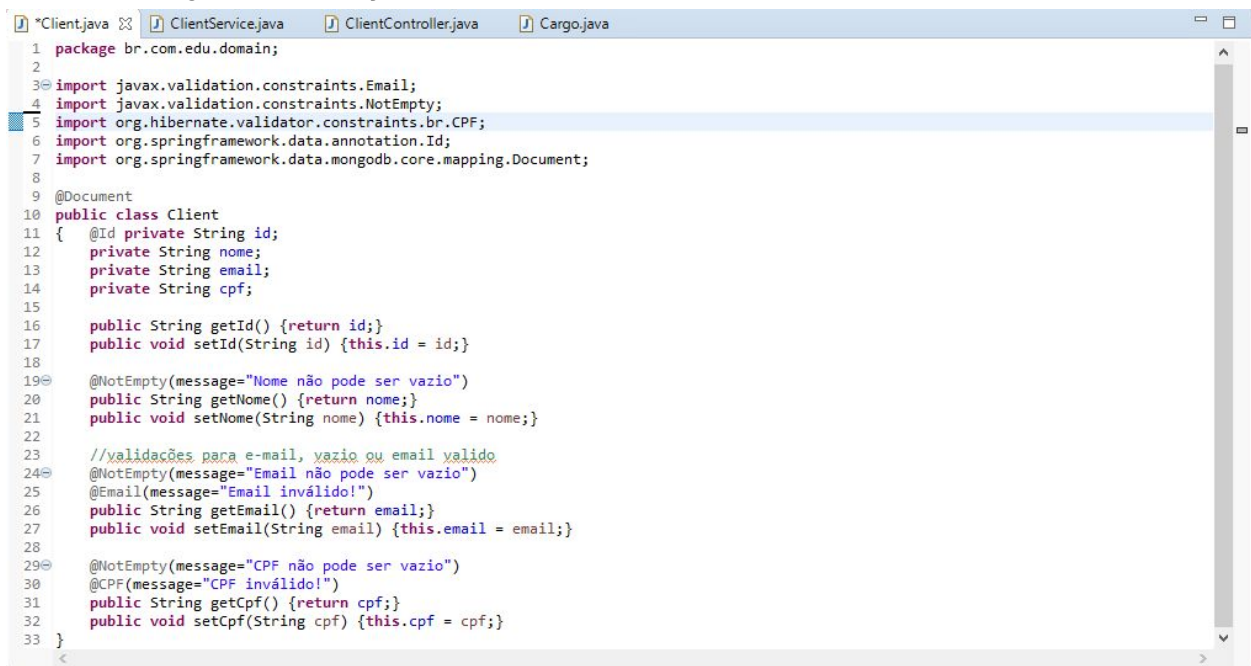
```

//lembrando que nesses casos o objeto cliente possui um id
@PutMapping(path="/{id}") public ResponseEntity<Client> update(@PathVariable(name="id") String id, @RequestBody Client client)
{
    client.setId(id);
    return ResponseEntity.ok(this.service.update(client));
}

@DeleteMapping(path="/{id}")
public ResponseEntity<Integer> delete(@PathVariable(name="id") String id)
{
    this.service.delete(id);
    //retorna um inteiro apenas para indicar que está tudo ok
    return ResponseEntity.ok(1);
}
}

```

## 11 - Aplicar regras de validação na classe de domínio:



```

1 package br.com.edu.domain;
2
3 import javax.validation.constraints.Email;
4 import javax.validation.constraints.NotEmpty;
5 import org.hibernate.validator.constraints.br.CPF;
6 import org.springframework.data.annotation.Id;
7 import org.springframework.data.mongodb.core.mapping.Document;
8
9 @Document
10 public class Client
11 {
12     @Id private String id;
13     private String nome;
14     private String email;
15     private String cpf;
16
17     public String getId() {return id;}
18     public void setId(String id) {this.id = id;}
19
20     @NotEmpty(message="Nome não pode ser vazio")
21     public String getNome() {return nome;}
22     public void setNome(String nome) {this.nome = nome;}
23
24     //validações para e-mail, vazio ou email valido
25     @NotEmpty(message="Email não pode ser vazio")
26     @Email(message="Email inválido!")
27     public String getEmail() {return email;}
28     public void setEmail(String email) {this.email = email;}
29
30     @NotEmpty(message="CPF não pode ser vazio")
31     @CPF(message="CPF inválido!")
32     public String getCpf() {return cpf;}
33     public void setCpf(String cpf) {this.cpf = cpf;}
34 }

```

Figura 15

## 12 - Configurar as validações na classe de controle

```

package br.com.edu.controller;

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

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;

```

```

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

import br.com.edu.domain.Client;
import br.com.edu.service.ClientService;
//This notation allows any class be a Rest type
@RestController//this path will be our app base path, it means that it will be
@RequestMapping(path="/api/clientes")//our URL
public class ClientController
{
    @Autowired private ClientService service;

    //método que usará e será feito dentro de um objeto próprio do Spring
    //da classe chamado ResponseEntity, que ajuda a retornar os dados da
    //da comunicação. Está definido como um tipo GET e não será definido um
    //path, pois estamos assumindo que o path do RequestMapping será usado
    //como padrão para retornar essa informação.
    @GetMapping public ResponseEntity<List<Client>> listAllClient()
    {return ResponseEntity.ok(this.service.listAll());}

    //como será feita listagem pelo ID, então será criado um path. As chaves indicam
    //que o id é um valor dinâmico.e estará na URL. Toda vez que fizer uma requisição
    //com "/api/clientes/id", será usado este {id}
    @GetMapping(path="/{id}") public ResponseEntity<Client> listClientID(@PathVariable(name="id") String id)
    {return ResponseEntity.ok(this.service.searchClientID(id));}

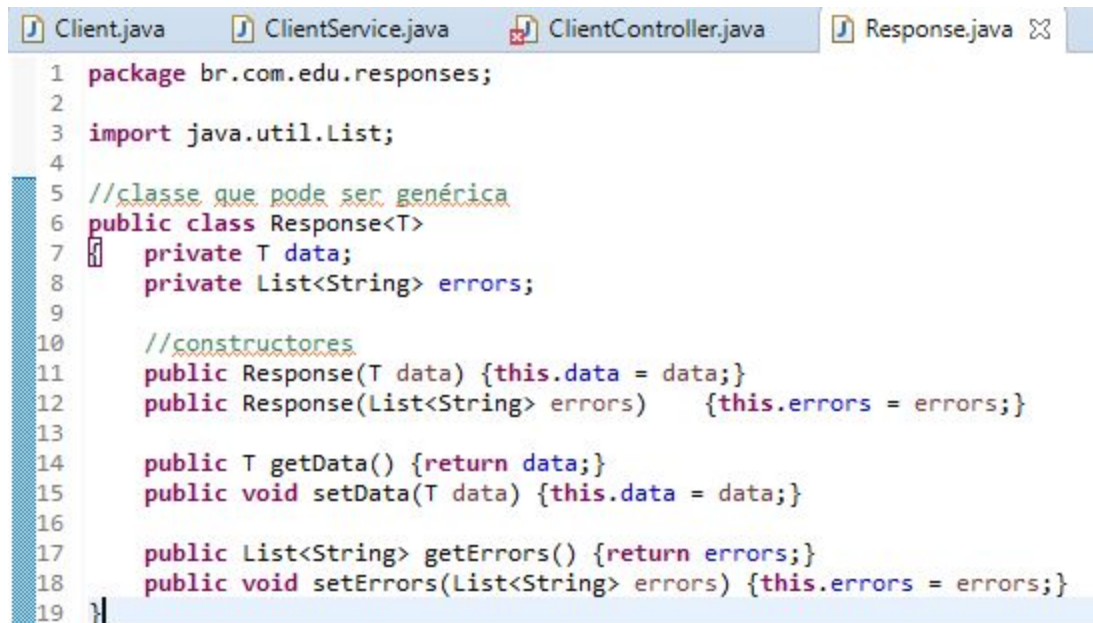
    //o @RequestBody faz com que o spring extraia automaticamente dados vindo
    //do PostRequest e criará um objeto do tipo Client automaticamente para nós
    //a notação "@Valid" é a notação responsável pelas validações.
    //o BindResult é a classe que me retorna o resultado da aplicação.
    @PostMapping public ResponseEntity<Client> save(@Valid @RequestBody Client client, BindingResult result)
    {
        if(result.hasErrors())//se houver erros
        {
            List<String> errors = new ArrayList<>();
            //insere numa lista de erros as mensagens de erros que podem aparecer
            result.getAllErrors().forEach(error -> errors.add(error.getDefaultMessage()));
            //um erro comum de tipo 400. Como deve ser retornado um tipo Client, devemos
            //criar uma classe que permita retornar uma mensagem
            return ResponseEntity.badRequest().body(errors);
        }
        return ResponseEntity.ok(this.service.save(client));
    }

    //lembrando que nesses casos o objeto cliente possui um id
    @PutMapping(path="/{id}") public ResponseEntity<Client> update(@PathVariable(name="id") String id, @RequestBody Client client)
    {
        client.setId(id);
        return ResponseEntity.ok(this.service.update(client));
    }

    @DeleteMapping(path="/{id}")
    public ResponseEntity<Integer> delete(@PathVariable(name="id") String id)
    {
        this.service.delete(id);
        //retorna um inteiro apenas para indicar que está tudo ok
        return ResponseEntity.ok(1);
    }
}

```

13 - Criar uma nova classe para que o método salvar anterior permite retornar dois tipos:



```
1 package br.com.edu.responses;
2
3 import java.util.List;
4
5 //classe que pode ser genérica
6 public class Response<T>
7 {
8     private T data;
9     private List<String> errors;
10
11     //constructores
12     public Response(T data) {this.data = data;}
13     public Response(List<String> errors) {this.errors = errors;}
14
15     public T getData() {return data;}
16     public void setData(T data) {this.data = data;}
17
18     public List<String> getErrors() {return errors;}
19     public void setErrors(List<String> errors) {this.errors = errors;}
20 }
```

Figura 16

14 - Configurar todos os métodos para a forma padrão de Response:

```
package br.com.edu.controller;

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

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import br.com.edu.domain.Client;
import br.com.edu.responses.Response;
import br.com.edu.service.ClientService;

//This notation allows any class be a Rest type
@RestController//this path will be our app base path, it means that it will be
@RequestMapping(path="/api/clientes")//our URL
public class ClientController
{
    @Autowired private ClientService service;

    //método que usará e será feito dentro de um objeto próprio do Spring
    //da classe chamado ResponseEntity, que ajuda a retornar os dados da
    //da comunicação. Está definido como um tipo GET e não será definido um
```



```

//path, pois estamos assumindo que o path do RequestMapping será usado
//como padrão para retornar essa informação.
@GetMapping public ResponseEntity<Response<List<Client>>> listAllClient()
{return ResponseEntity.ok(new Response<List<Client>>(this.service.listAll()));}

//como será feita listagem pelo ID, então será criado um path. As chaves indicam
//que o id é um valor dinâmico.e estará na URL. Toda vez que fizer uma requisição
//com "/api/clientes/id", será usado este {id}
@GetMapping(path="{id}") public ResponseEntity<Response<Client>> listClientID(@PathVariable(name="id") String id)
{return ResponseEntity.ok(new Response<Client>(this.service.searchClientID(id)));}

//o @RequestBody faz com que o spring extraia automaticamente dados vindo
//do PostRequest e criará um objeto do tipo Client automaticamente para nós
//a notação "@Valid" é a notação responsável pelas validações.
//o BindResult é a classe que me retorna o resultado da aplicação.
@PostMapping public ResponseEntity<Response<Client>> save(@Valid @RequestBody Client client, BindingResult result)
{
    if(result.hasErrors())//se houver erros
    {
        List<String> errors = new ArrayList<>();
        //insere numa lista de erros as mensagens de erros que podem aparecer
        result.getAllErrors().forEach(error -> errors.add(error.getDefaultMessage()));
        //um erro comum de tipo 400. Como deve ser retornado um tipo Client, devemos
        //criar uma classe que permita retornar uma mensagem de resposta papdrão
        return ResponseEntity.badRequest().body(new Response<Client>(errors));
    }
    return ResponseEntity.ok(new Response<Client>(this.service.save(client)));
}

//lembrando que nesses casos o objeto cliente possui um id
@PutMapping(path="{id}") public ResponseEntity<Response<Client>> update(@Valid @PathVariable(name="id") String id, @RequestBody Client
client, BindingResult result)
{
    if(result.hasErrors())//se houver erros
    {
        List<String> errors = new ArrayList<>();
        //insere numa lista de erros as mensagens de erros que podem aparecer
        result.getAllErrors().forEach(error -> errors.add(error.getDefaultMessage()));
        //um erro comum de tipo 400. Como deve ser retornado um tipo Client, devemos
        //criar uma classe que permita retornar uma mensagem de resposta papdrão
        return ResponseEntity.badRequest().body(new Response<Client>(errors));
    }
    client.setId(id);
    return ResponseEntity.ok(new Response<Client>(this.service.update(client)));
}

@DeleteMapping(path="{id}")
public ResponseEntity<Response<Integer>> delete(@PathVariable(name="id") String id)
{
    this.service.delete(id);
    //retorna um inteiro apenas para indicar que está tudo ok
    return ResponseEntity.ok(new Response<Integer>(1));
}
}

```

## 15 - Executar o projeto com Spring Boot:

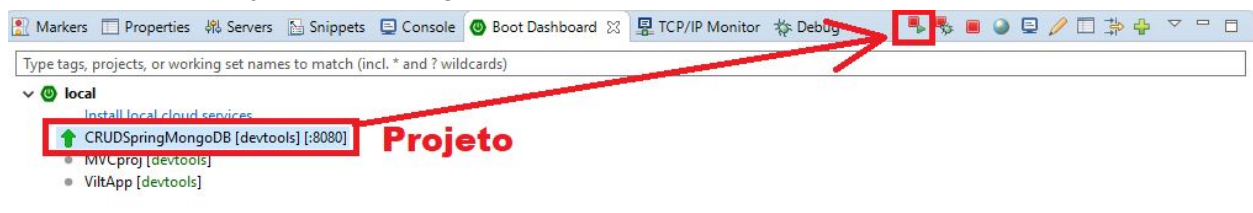


Figura 17

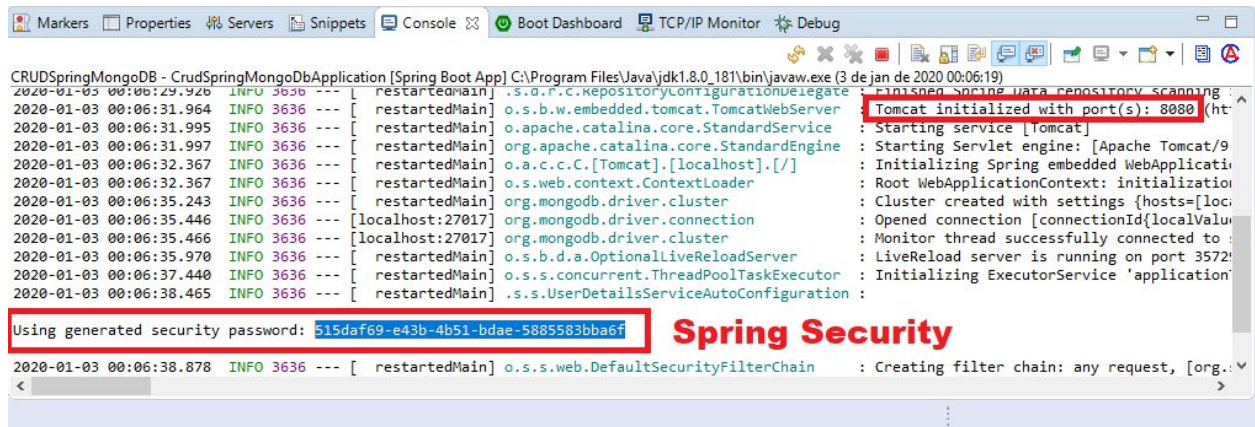


Figura 18

16 - Aplicar o GET pelo *postman* no URL + path:

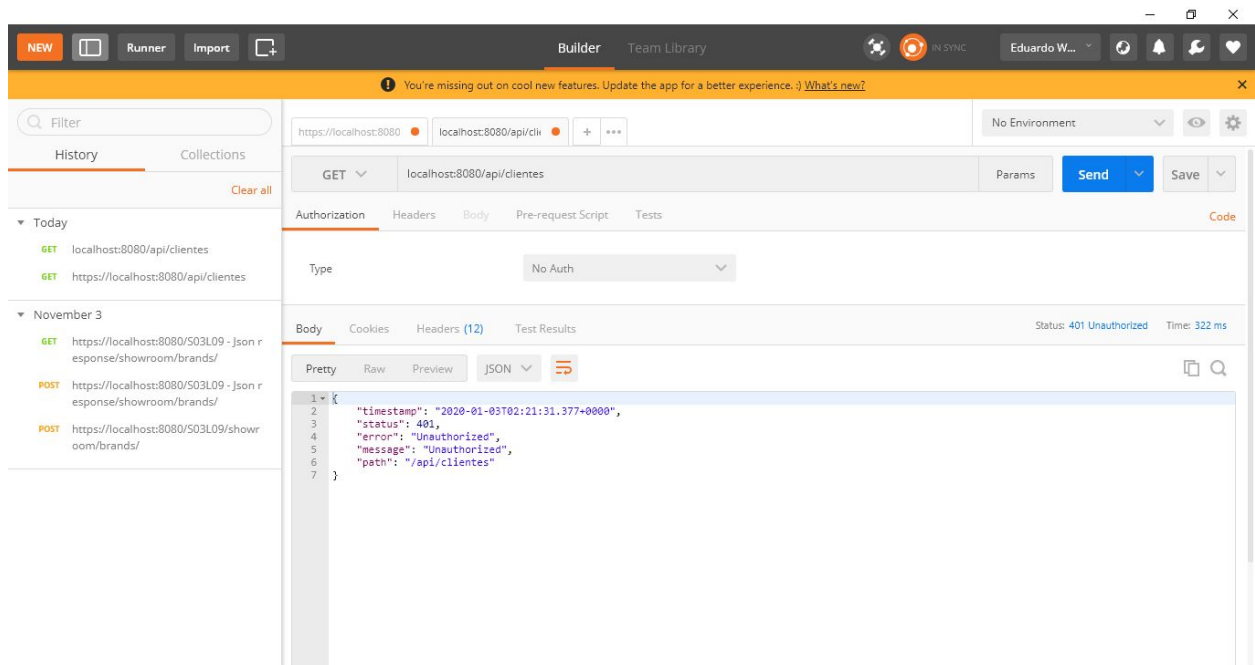


Figura 19

O fato de ter retornado 401 foi porque não definimos o usuário e senha na requisição, e o Spring Security está protegendo nossos recursos. No *postman*, é possível fazer essa autorização pelo *type*. Altere para *Basic Auth* e preencha os campos com *Username* como “user” e *password* com a senha definida pelo spring security, exibida na imagem 18 e em seguida, clique em *update request*.

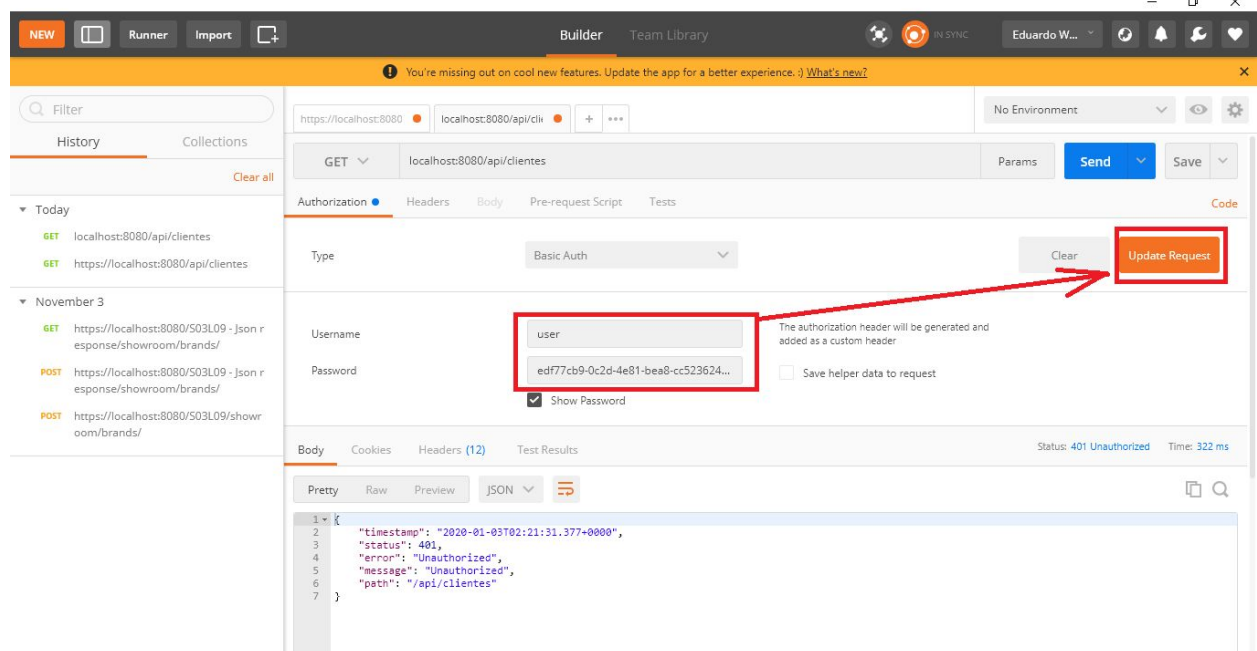


Figura 20

Perceba que no *Headers* foi atualizado e que se enviarmos novamente o GET na URL + path, não me retorna nenhum erro mas também não retorna nada de informação, pois não temos nada cadastrado no BD.

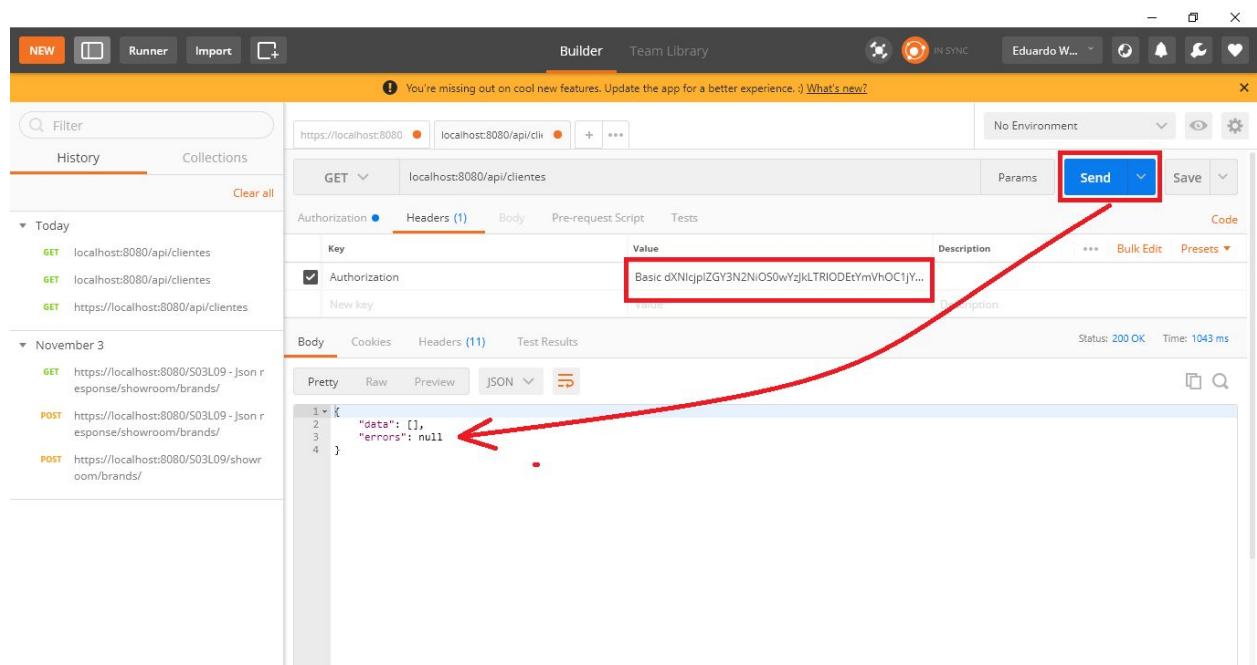


Figura 21

Agora vamos alterar o método para *POST* e vamos para a aba *Body* do *postman*. Lembre-se que este método realizará o cadastro de um cliente. Se enviarmos dados vazios em JSON, perceba que nos retornará mensagens de erro de validação de cliente.

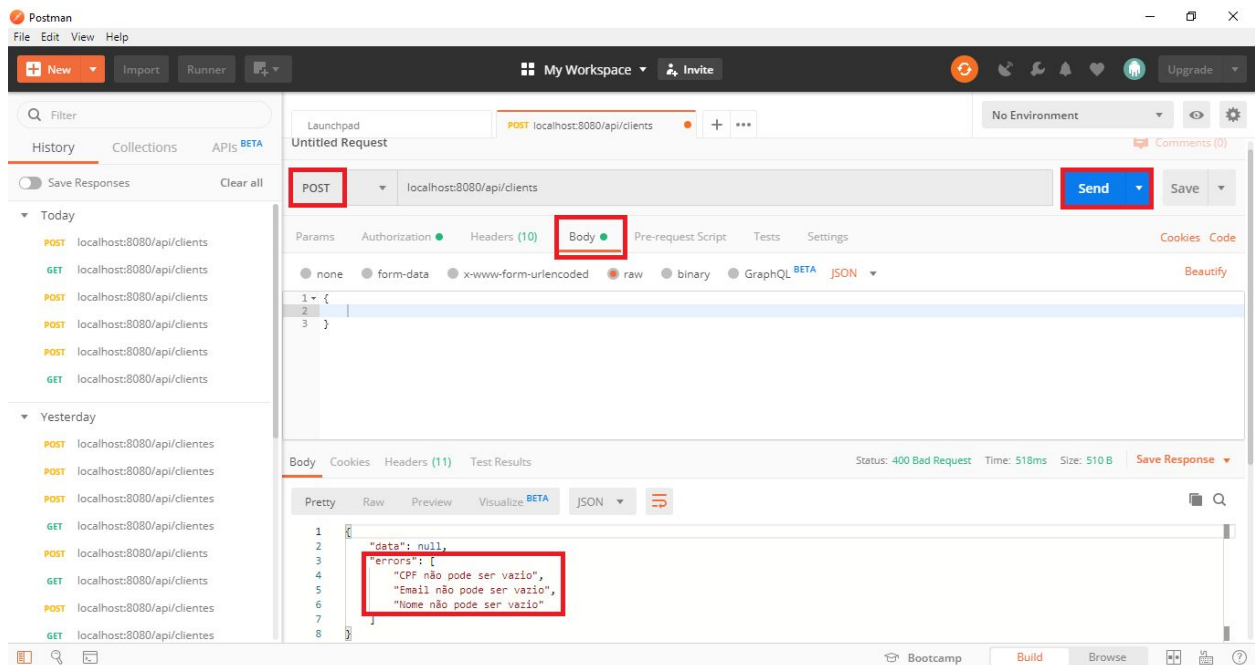


Figura 22

Se inserirmos dados válidos, nos retorna o cliente cadastrado e o id também que é gerado automaticamente pelo MongoDB.

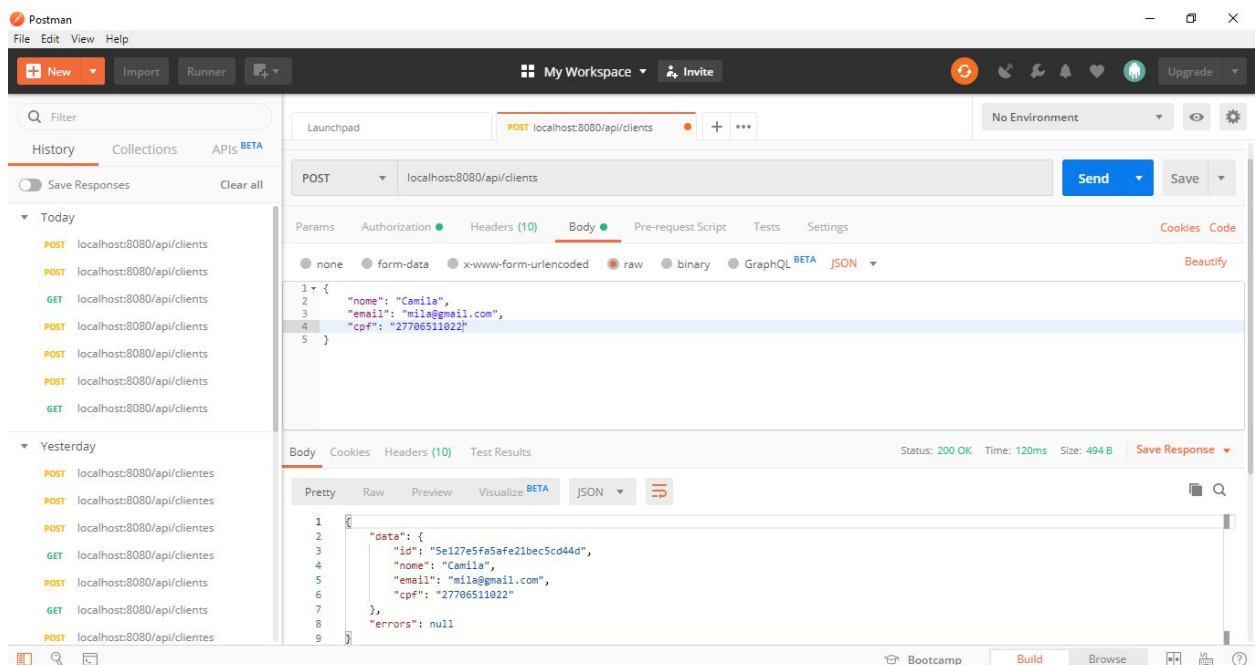


Figura 23



Retornemos para GET e veremos todos os clientes cadastrados.

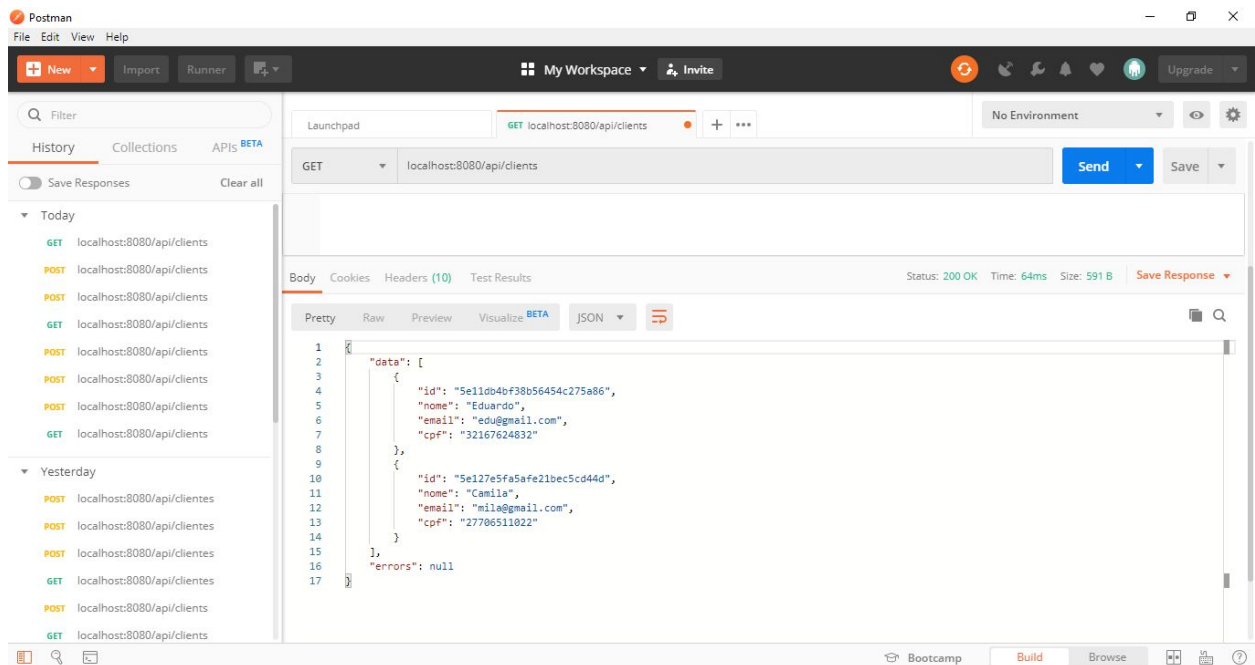


Figura 24

Agora vamos para o prompt de comando para ver o status do MongoDB:

1 - "mongod": Verifica o status da SGBD MongoDB.

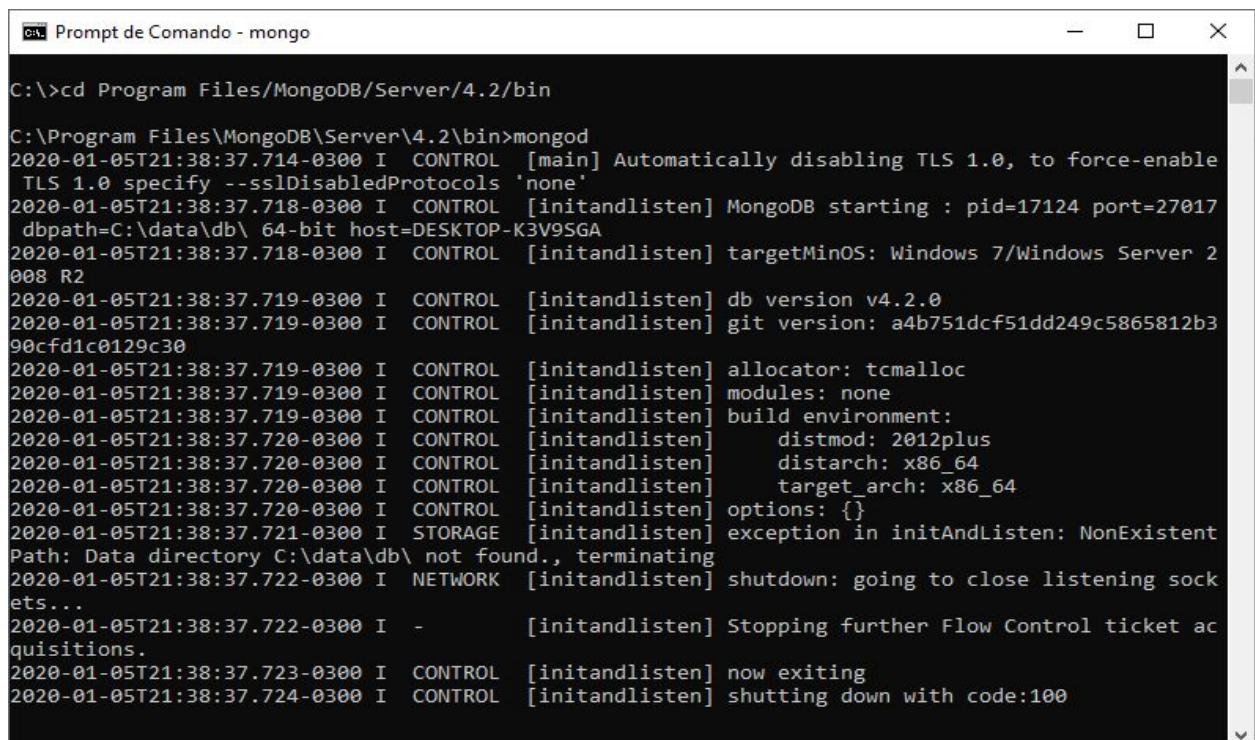
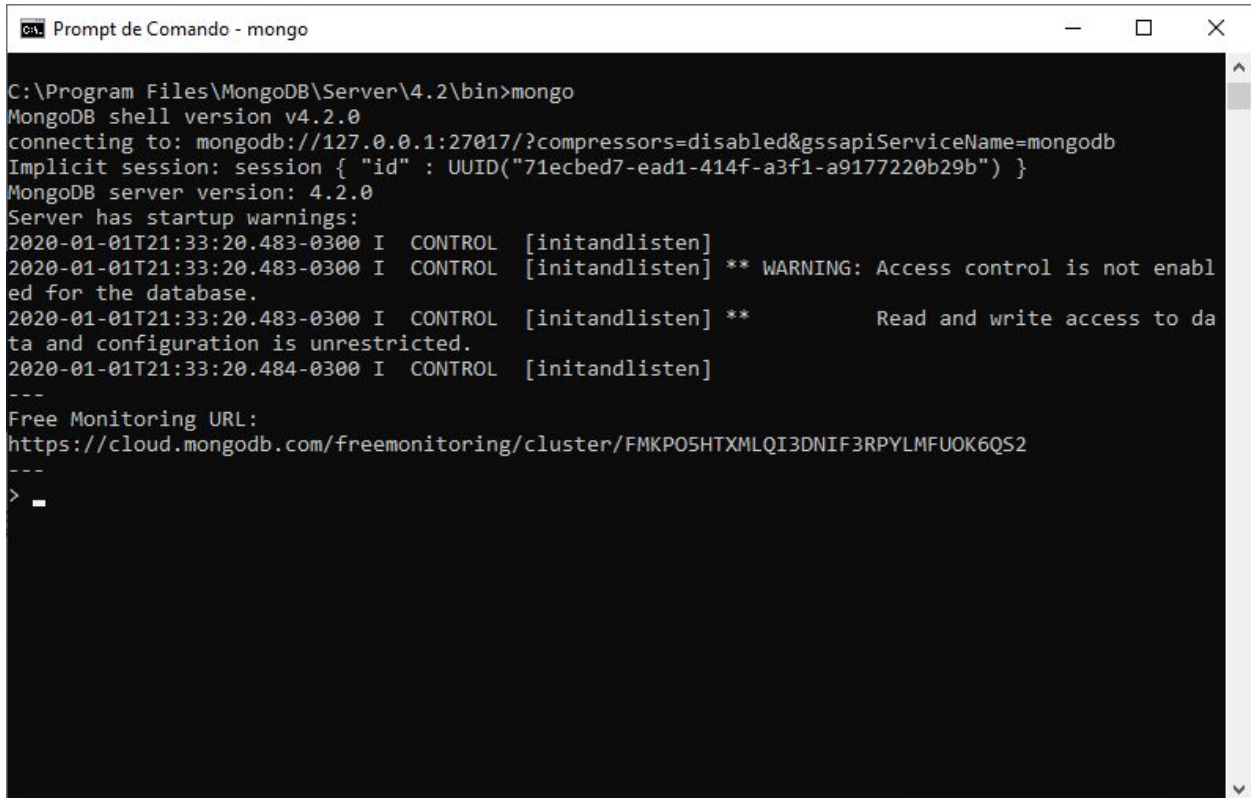


Figura 25

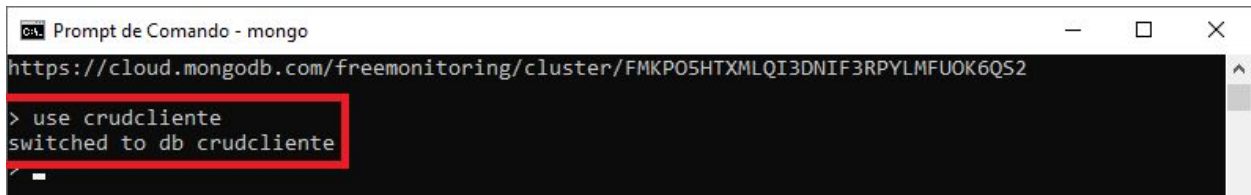
2 - “mongo”: acessa o console da linha de comando do MongoDB.



```
C:\Program Files\MongoDB\Server\4.2\bin>mongo
MongoDB shell version v4.2.0
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("71ecbed7-ead1-414f-a3f1-a9177220b29b") }
MongoDB server version: 4.2.0
Server has startup warnings:
2020-01-01T21:33:20.483-0300 I  CONTROL  [initandlisten]
2020-01-01T21:33:20.483-0300 I  CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2020-01-01T21:33:20.483-0300 I  CONTROL  [initandlisten] **           Read and write access to data and configuration is unrestricted.
2020-01-01T21:33:20.484-0300 I  CONTROL  [initandlisten]
---
Free Monitoring URL:
https://cloud.mongodb.com/freemonitoring/cluster/FMKP05HTXMLQI3DNIF3RPYLMFU0K6QS2
---
>
```

Figura 26

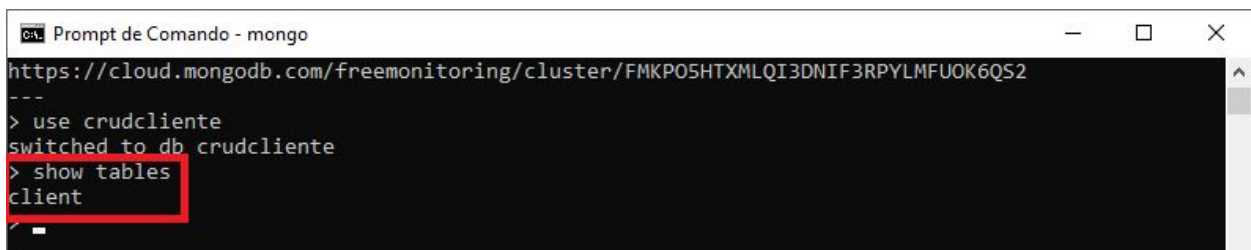
3 - “use NOME\_DATABASE”: acessa o banco de dados desejado.



```
https://cloud.mongodb.com/freemonitoring/cluster/FMKP05HTXMLQI3DNIF3RPYLMFU0K6QS2
---
> use crudcliente
switched to db crudcliente
✓
```

Figura 27

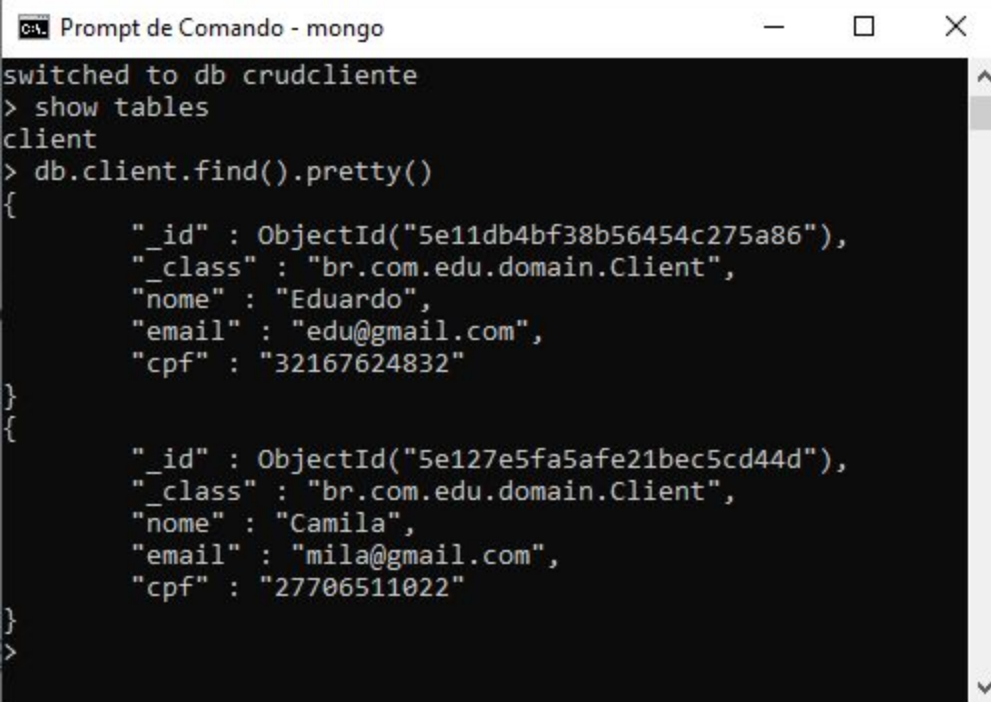
4 - “show tables”: Visualiza as tabelas do banco de dados da aplicação.



```
https://cloud.mongodb.com/freemonitoring/cluster/FMKP05HTXMLQI3DNIF3RPYLMFU0K6QS2
---
> use crudcliente
switched to db crudcliente
> show tables
client
✓
```

Figura 28

5 - `db.client.find().pretty()`: temos o acesso a tabela *client* e fazemos uma consulta de forma mais elegante.

A screenshot of a Windows command prompt window titled "Prompt de Comando - mongo". The window has a black background with white text. The text shows the following sequence of commands and output:

```
switched to db crudcliente
> show tables
client
> db.client.find().pretty()
{
  "_id" : ObjectId("5e11db4bf38b56454c275a86"),
  "_class" : "br.com.edu.domain.Client",
  "nome" : "Eduardo",
  "email" : "edu@gmail.com",
  "cpf" : "32167624832"
}
{
  "_id" : ObjectId("5e127e5fa5afe21bec5cd44d"),
  "_class" : "br.com.edu.domain.Client",
  "nome" : "Camila",
  "email" : "mila@gmail.com",
  "cpf" : "27706511022"
}
>
```