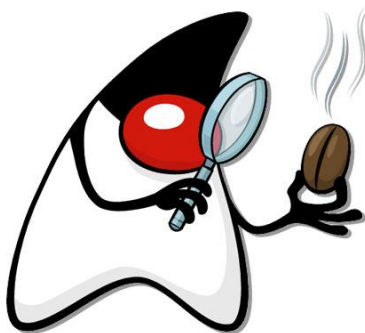


Bean Validations

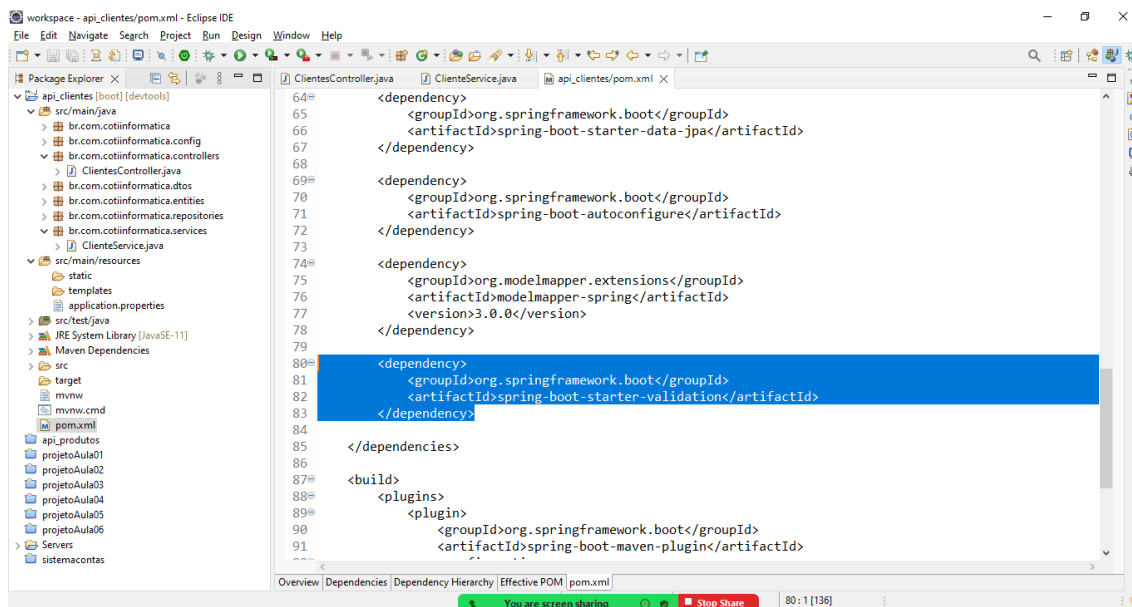


Java Bean Validation

Biblioteca Java voltada para validação de dados em classes JavaBean. Podemos utilizar esta biblioteca para validar os DTOs (Objetos de transferência de dados da API).

Para cada requisição da API que recebe um DTO, podemos utilizar a biblioteca Bean Validations de forma verificar os dados recebidos através destes DTOs.

Instalando:
/pom.xml



<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

/dtos/**PostClientesDTO.java**

Incluindo a validação dos campos para cadastro do cliente.

```
package br.com.cotiinformatica.dtos;

import javax.validation.constraints.Email;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Pattern;
import javax.validation.constraints.Size;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@Setter
@Getter
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class PostClientesDTO {

    @Size(min = 6, max = 150,
        message = "Informe de 6 a 150 caracteres.")
    @NotBlank(message = "Por favor, informe o nome do cliente.")
    private String nome;

    @Pattern(regexp = "^[0-9]{11}$",
        message = "Informe 11 dígitos numéricos.")
    @NotBlank(message = "Por favor, informe o cpf do cliente.")
    private String cpf;

    @Email(message = "Informe um endereço de email válido.")
    @NotBlank(message = "Por favor, informe o email do cliente.")
    private String email;

    @Pattern(regexp = "^[0-9]{11}$",
        message = "Informe 11 dígitos numéricos (DDD + Telefone)")
    @NotBlank(message = "Por favor, informe o telefone do cliente.")
    private String telefone;
}
```

/dtos/**PurClientesDTO.java**

Incluindo a validação dos campos para edição do cliente.

```
package br.com.cotiinformatica.dtos;

import javax.validation.constraints.Email;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Pattern;
import javax.validation.constraints.Size;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@Setter
@Getter
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class PutClientesDTO {

    @Min(value = 1,
        message = "Por favor, informe um id maior do que zero.")
    private Integer idCliente;

    @Size(min = 6, max = 150,
        message = "Informe de 6 a 150 caracteres.")
    @NotBlank(message = "Por favor, informe o nome do cliente.")
    private String nome;

    @Pattern(regex = "^[0-9]{11}$",
        message = "Informe 11 dígitos numéricos.")
    @NotBlank(message = "Por favor, informe o cpf do cliente.")
    private String cpf;

    @Email(message = "Informe um endereço de email válido.")
    @NotBlank(message = "Por favor, informe o email do cliente.")
    private String email;

    @Pattern(regex = "^[0-9]{11}$",
        message = "Informe 11 dígitos numéricos (DDD + Telefone)")
    @NotBlank(message = "Por favor, informe o telefone do cliente.")
    private String telefone;
}
```

/services/**ClienteService.java**

Voltando na camada de serviço:

```
package br.com.cotiinformatica.services;

import java.util.Date;

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

import br.com.cotiinformatica.entities.Cliente;
import br.com.cotiinformatica.repositories.IClienteRepository;

@Service
public class ClienteService {

    // injeção de dependência
    @Autowired
    private IClienteRepository clienteRepository;

    // Método para realizar o cadastro de um cliente
    public void cadastrar(Cliente cliente) {

        // gerando a data de cadastro do cliente
        cliente.setDataCadastro(new Date());

        // gravando no banco de dados
        clienteRepository.save(cliente);
    }
}
```

Para que as validações mapeadas nos DTOS funcionem, precisamos adicionar a seguinte anotação nos métodos do controlador: **@Valid**

```
package br.com.cotiinformatica.controllers;

import java.util.List;

import javax.validation.Valid;

import org.modelmapper.ModelMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
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.RestController;
```

```
import br.com.cotiinformatica.dtos.GetClientesDTO;
import br.com.cotiinformatica.dtos.PostClientesDTO;
import br.com.cotiinformatica.dtos.PutClientesDTO;
import br.com.cotiinformatica.dtos.ResponseClientesDTO;
import br.com.cotiinformatica.entities.Cliente;
import br.com.cotiinformatica.services.ClienteService;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;

@Api(tags = "Clientes")
@RestController
public class ClientesController {

    // injeção de dependência
    @Autowired
    private ClienteService clienteService;

    @ApiOperation("Serviço para cadastro de clientes.")
    @PostMapping("/api/clientes")
    public ResponseEntity<ResponseClientesDTO> post
        (@Valid @RequestBody PostClientesDTO dto) {

        ResponseClientesDTO response = new ResponseClientesDTO();

        try {

            ModelMapper modelMapper = new ModelMapper();
            Cliente cliente = modelMapper.map
                (dto, Cliente.class);

            clienteService.cadastrar(cliente);

            response.setStatus(201);
            response.setMensagem
                ("Cliente cadastrado com sucesso.");
            response.setCliente(modelMapper.map
                (cliente, GetClientesDTO.class));

            return ResponseEntity.status
                (HttpStatus.CREATED).body(response);
        } catch (IllegalArgumentException e) {

            response.setStatus(400);
            response.setMensagem(e.getMessage());

            return ResponseEntity.status
                (HttpStatus.BAD_REQUEST).body(response);
        } catch (Exception e) {

            response.setStatus(500);
            response.setMensagem(e.getMessage());
```

```
        return ResponseEntity.status  
            (HttpStatus.INTERNAL_SERVER_ERROR).body(response);  
    }  
}  
  
@ApiOperation("Serviço para edição de clientes.")  
@PutMapping("/api/clientes")  
public ResponseEntity<ResponseClientesDTO> put  
    (@Valid @RequestBody PutClientesDTO dto) {  
  
    ResponseClientesDTO response = new ResponseClientesDTO();  
  
    try {  
  
        //TODO  
  
        return ResponseEntity.status  
            (HttpStatus.OK).body(response);  
  
    } catch (IllegalArgumentException e) {  
  
        response.setStatus(400);  
        response.setMensagem(e.getMessage());  
  
        return ResponseEntity.status  
            (HttpStatus.BAD_REQUEST).body(response);  
  
    } catch (Exception e) {  
  
        response.setStatus(500);  
        response.setMensagem(e.getMessage());  
  
        return ResponseEntity.status  
            (HttpStatus.INTERNAL_SERVER_ERROR).body(response);  
    }  
}  
  
@ApiOperation("Serviço para exclusão de clientes.")  
@DeleteMapping("/api/clientes/{id}")  
public ResponseEntity<ResponseClientesDTO> delete  
    (@PathVariable("id") Integer idCliente) {  
  
    ResponseClientesDTO response = new ResponseClientesDTO();  
  
    try {  
  
        //TODO  
  
        return ResponseEntity.status  
            (HttpStatus.OK).body(response);  
  
    } catch (IllegalArgumentException e) {
```

```

        response.setStatus(400);
        response.setMensagem(e.getMessage());

        return ResponseEntity.status
            (HttpStatus.BAD_REQUEST).body(response);

    } catch (Exception e) {

        response.setStatus(500);
        response.setMensagem(e.getMessage());

        return ResponseEntity.status
            (HttpStatus.INTERNAL_SERVER_ERROR).body(response);

    }

}

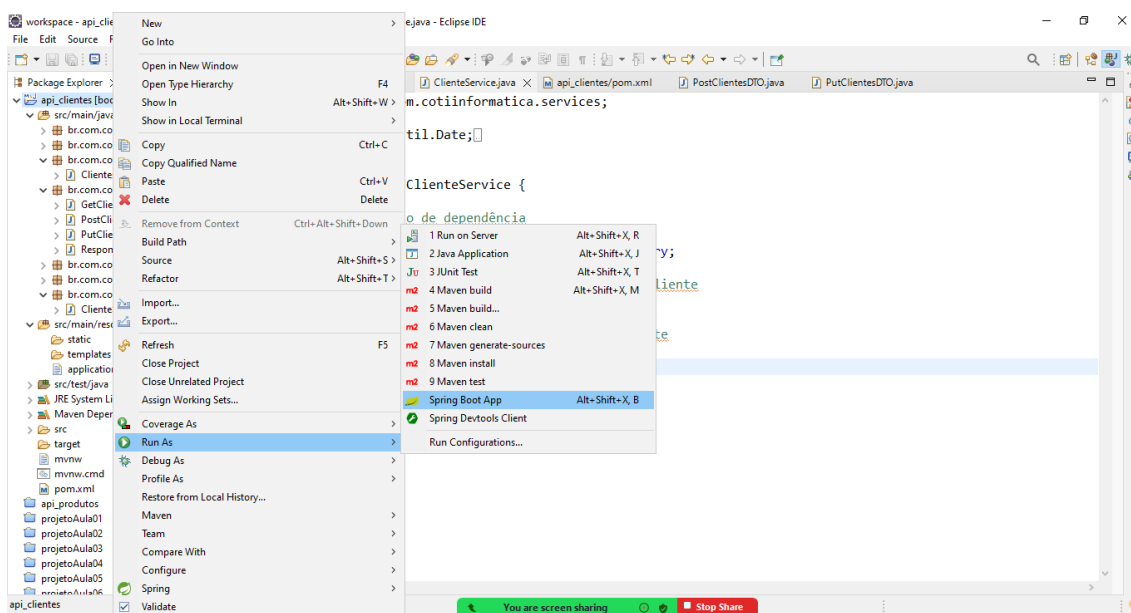
@ApiOperation("Serviço para consulta de clientes.")
@GetMapping("/api/clientes")
public ResponseEntity<List<GetClientesDTO>> getAll() {
    return null;
}

@ApiOperation("Serviço para consulta de cliente por id.")
@GetMapping("/api/clientes/{id}")
public ResponseEntity<GetClientesDTO> getById
    (@PathVariable("id") Integer idCliente) {
    return null;
}

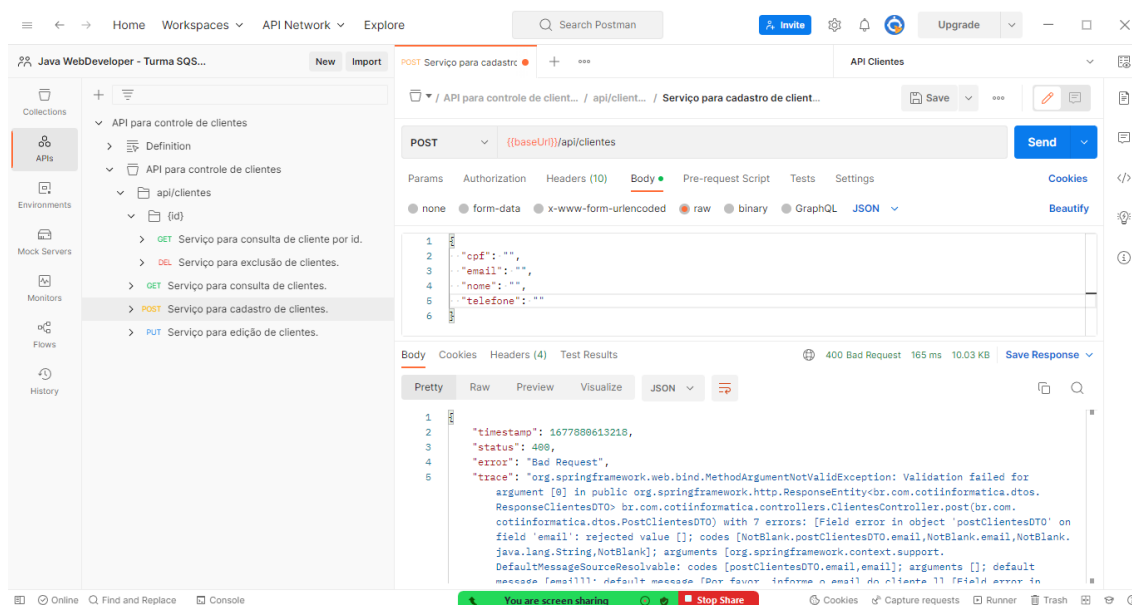
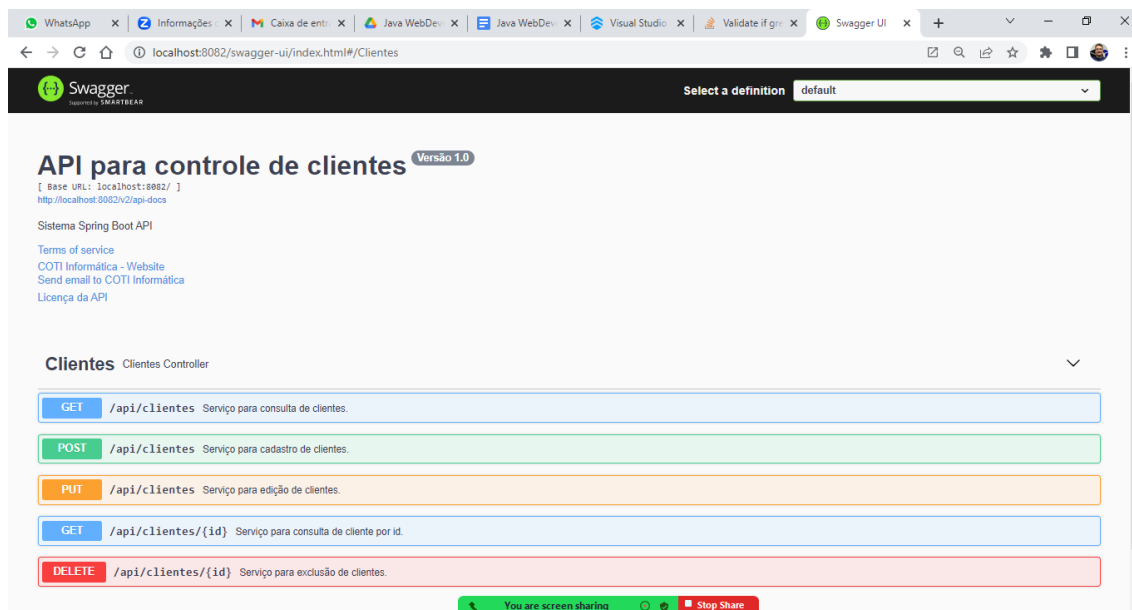
}

```

Executando o projeto:



<http://localhost:8082/swagger-ui/index.html>




BAD REQUEST

Erro 400

Código de erro do protocolo HTTP que indica que o cliente da API enviou dados inválidos na requisição. Geralmente está relacionado a algum erro de validação de dados ou mesmo regra de negócio.

Para que possamos retornar corretamente os erros de validação gerados pelo BeanValidations na API, precisamos criar uma classe que faça o tratamento deste conteúdo e formate o seu retorno.

 **New Java Package**


Java Package
Create a new Java package.

Creates folders corresponding to packages.

Source folder: [Browse...](#)

Name:

[?](#) [Finish](#) [Cancel](#)

 **New Java Class**

Java Class
Create a new Java class.

Source folder: [Browse...](#)

Package: [Browse...](#)

☐ Enclosing type: [Browse...](#)

Name:

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

Superclass: [Browse...](#)

Interfaces: [Add...](#) [Remove](#)

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

[?](#) [Finish](#) [Cancel](#)

```
package br.com.cotiinformatica.handlers;

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

import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.context.request.WebRequest;
import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;

@ControllerAdvice
public class ErrorHandler extends ResponseEntityExceptionHandler {

    @Override
    protected ResponseEntity<Object> handleMethodArgumentNotValid
        (MethodArgumentNotValidException ex,
         HttpHeaders headers, HttpStatus status,
         WebRequest request) {

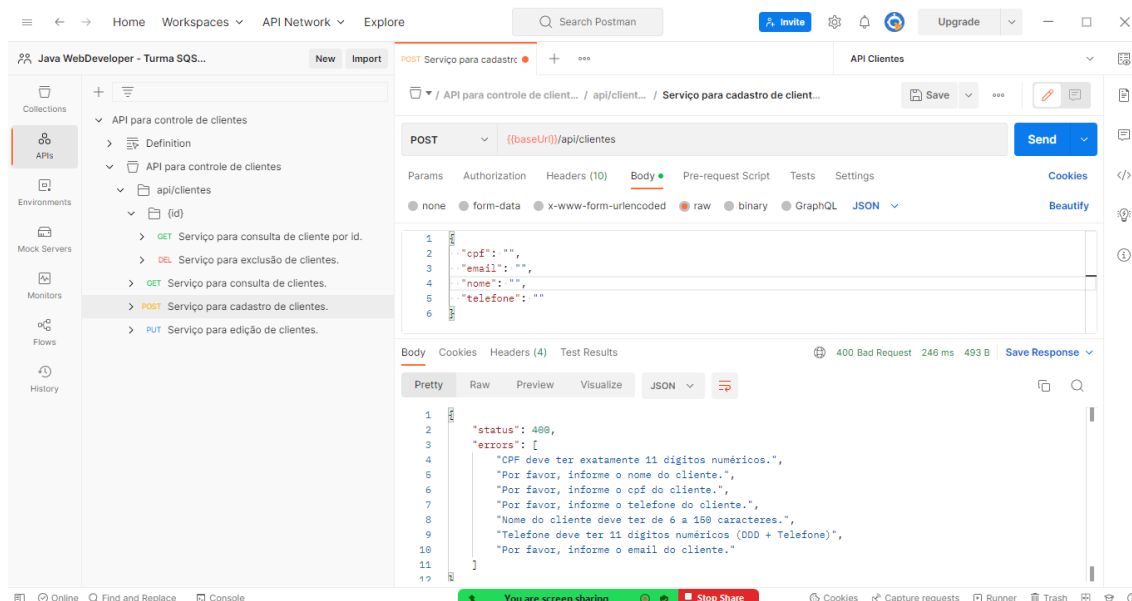
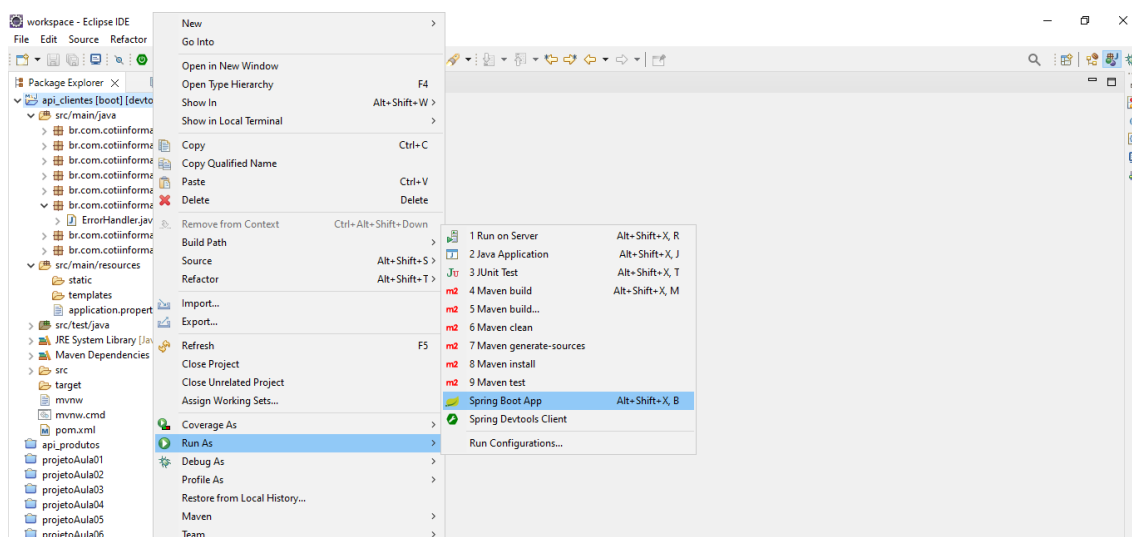
        Map<String, Object> body = new LinkedHashMap<>();
        body.put("status", status.value());

        List<String> errors = ex.getBindingResult()
            .getFieldErrors().stream()
            .map(x -> x.getDefaultMessage())
            .collect(Collectors.toList());

        body.put("errors", errors);

        return new ResponseEntity<>(body, headers, status);
    }
}
```

Executando e testando o projeto:

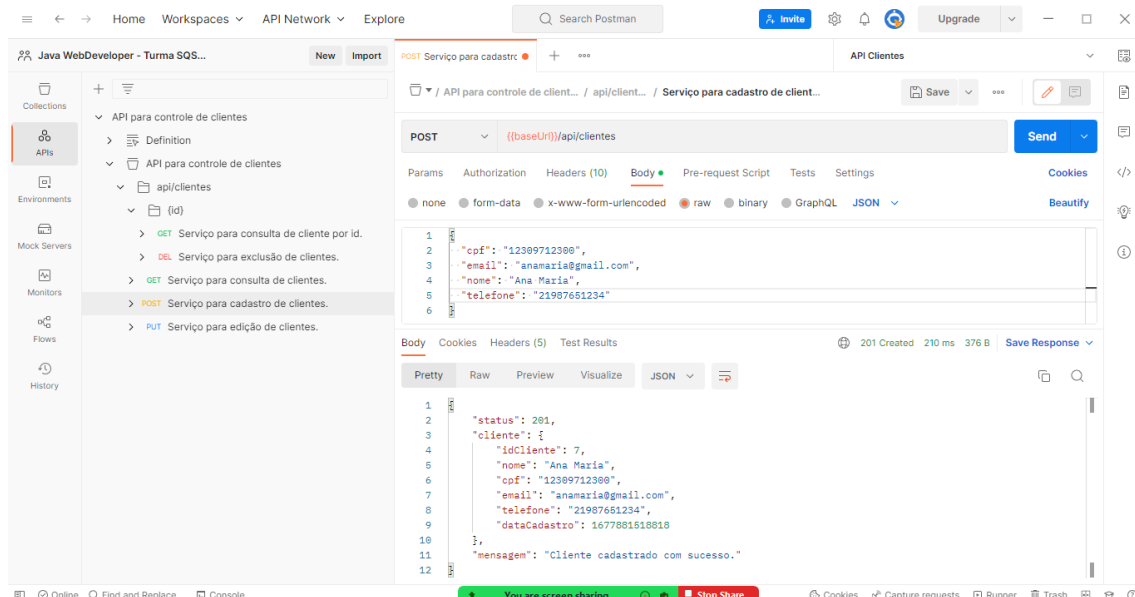


RESPONSE:

```

{
  "status": 400,
  "errors": [
    "CPF deve ter exatamente 11 dígitos numéricos.",
    "Por favor, informe o nome do cliente.",
    "Por favor, informe o cpf do cliente.",
    "Por favor, informe o telefone do cliente.",
    "Nome do cliente deve ter de 6 a 150 caracteres.",
    "Telefone deve ter 11 dígitos numéricos (DDD + Telefone)",
    "Por favor, informe o email do cliente."
  ]
}

```



JPQL - Java Persistence Query Language

Sintaxe de consultas para que possamos escrever queries nos projetos que utilizam Hibernate & JPA sem a necessidade de escrita de código SQL.

A sintaxe para escrita destas consultas tem como características:

- Não utilizam código SQL.
- As consultas são feitas baseadas nos nomes das classes de entidade mapeadas pela JPA e não das tabelas do banco de dados.

/repositories/**IClienteRepository.java**

```
package br.com.cotiinformatica.repositories;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
```

```
import br.com.cotiinformatica.entities.Cliente;
```

```
@Repository
```

```
public interface IClienteRepository extends JpaRepository<Cliente, Integer> {
```

```
    /*
     * Consulta JPQL - Java Persistence Query Language
     * para retornar 1 cliente do banco de dados através do email
     * se nenhum cliente for encontrado, o método retorna null
     */
```

```
@Query("select c from Cliente c where c.email = :pEmail")
Cliente findByEmail(@Param("pEmail") String email);
```

```
/*
 * Consulta JPQL - Java Persistence Query Language
 * para retornar 1 cliente do banco de dados através do cpf
 * se nenhum cliente for encontrado, o método retorna null
 */
@Query("select c from Cliente c where c.cpf = :pCpf")
Cliente findByCpf(@Param("pCpf") String cpf);
}
```

Voltando na camada de serviço:

/services/**ClienteService.java**

```
package br.com.cotiinformatica.services;

import java.util.Date;

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

import br.com.cotiinformatica.entities.Cliente;
import br.com.cotiinformatica.repositories.IClienteRepository;

@Service
public class ClienteService {

    // injeção de dependência
    @Autowired
    private IClienteRepository clienteRepository;

    // Método para realizar o cadastro de um cliente
    public void cadastrar(Cliente cliente) {

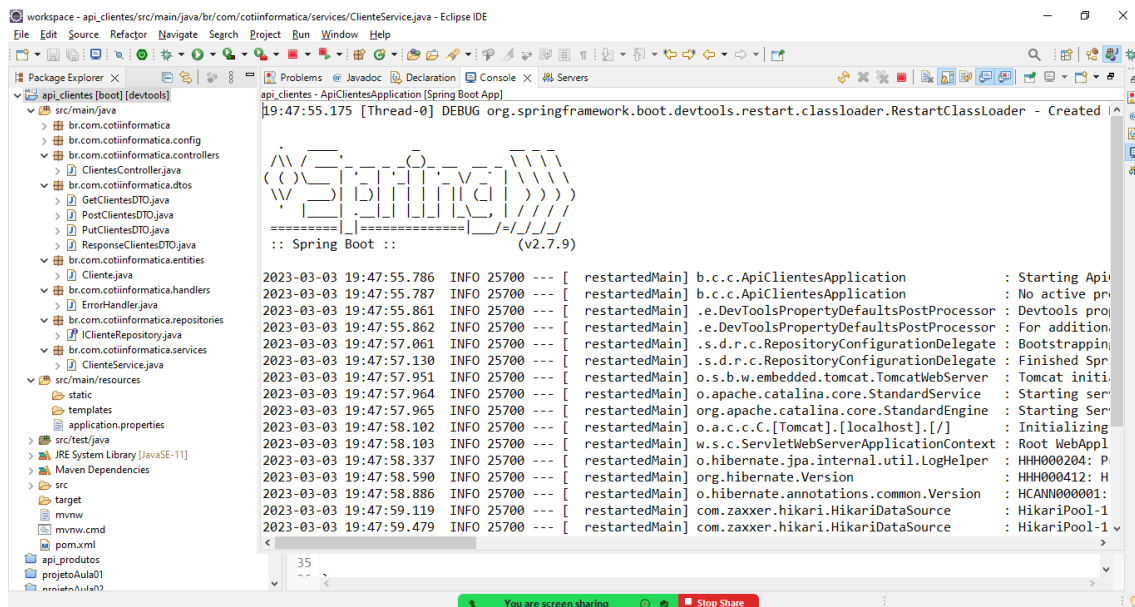
        // verificar se o email do cliente já está
        // cadastrado no banco de dados
        if (clienteRepository.findByEmail(cliente.getEmail()) != null)
            throw new IllegalArgumentException
                ("O email informado já está cadastrado. Tente outro.");

        // verificar se o cpf do cliente já está cadastrado no banco de dados
        if (clienteRepository.findByCpf(cliente.getCpf()) != null)
            throw new IllegalArgumentException
                ("O cpf informado já está cadastrado. Tente outro.");

        // gerando a data de cadastro do cliente
        cliente.setDataCadastro(new Date());

        // gravando no banco de dados
        clienteRepository.save(cliente);
    }
}
```

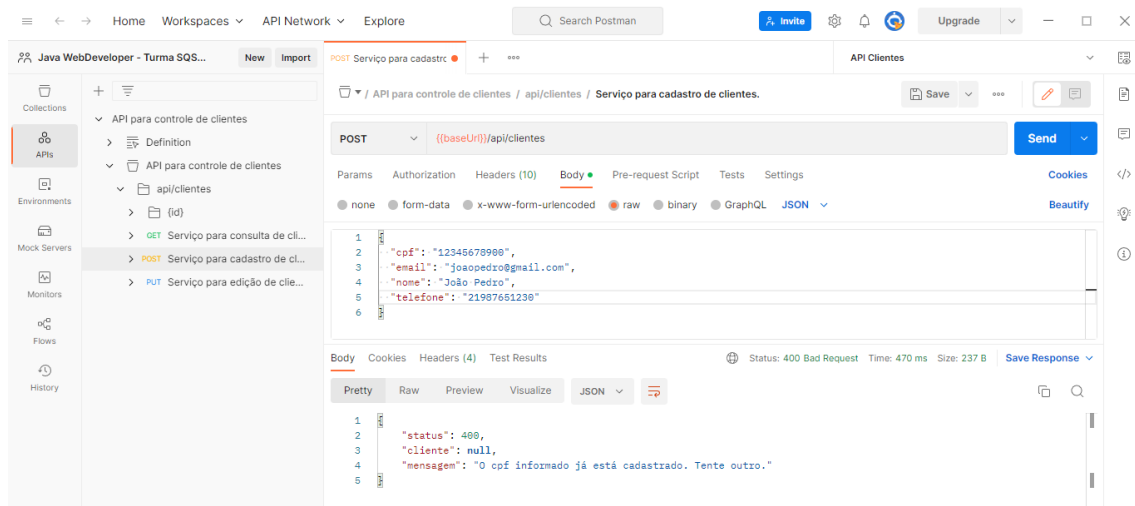
Executando e testando:



```

[19:47:55.175 [Thread-0] DEBUG org.springframework.boot.devtools.restart.classloader.RestartClassLoader - Created
=====
:: Spring Boot ::
(v2.7.9)

2023-03-03 19:47:55.786 INFO 25700 --- [ restartedMain] b.c.c.ApiClientesApplication : Starting Api
2023-03-03 19:47:55.787 INFO 25700 --- [ restartedMain] b.c.c.ApiClientesApplication : No active pr
2023-03-03 19:47:55.861 INFO 25700 --- [ restartedMain] e.DevToolsPropertyDefaultsPostProcessor : Devtools prop
2023-03-03 19:47:55.862 INFO 25700 --- [ restartedMain] e.DevToolsPropertyDefaultsPostProcessor : For addition
2023-03-03 19:47:57.061 INFO 25700 --- [ restartedMain] s.d.r.c.RepositoryConfigurationDelegate : Bootstrappin
2023-03-03 19:47:57.130 INFO 25700 --- [ restartedMain] s.d.r.c.RepositoryConfigurationDelegate : Finished Spr
2023-03-03 19:47:57.951 INFO 25700 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initi
2023-03-03 19:47:57.964 INFO 25700 --- [ restartedMain] o.apache.catalina.core.StandardEngine : Starting ser
2023-03-03 19:47:57.965 INFO 25700 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Ser
2023-03-03 19:47:58.102 INFO 25700 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing
2023-03-03 19:47:58.103 INFO 25700 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebAppl
2023-03-03 19:47:58.337 INFO 25700 --- [ restartedMain] o.hibernate.jpa.internal.util.LogHelper : HHH000204: P
2023-03-03 19:47:58.590 INFO 25700 --- [ restartedMain] org.hibernate.Version : HHH000412: H
2023-03-03 19:47:58.886 INFO 25700 --- [ restartedMain] o.hibernate.annotations.common.Version : HCANN000001:
2023-03-03 19:47:59.119 INFO 25700 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1
2023-03-03 19:47:59.479 INFO 25700 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1
  
```

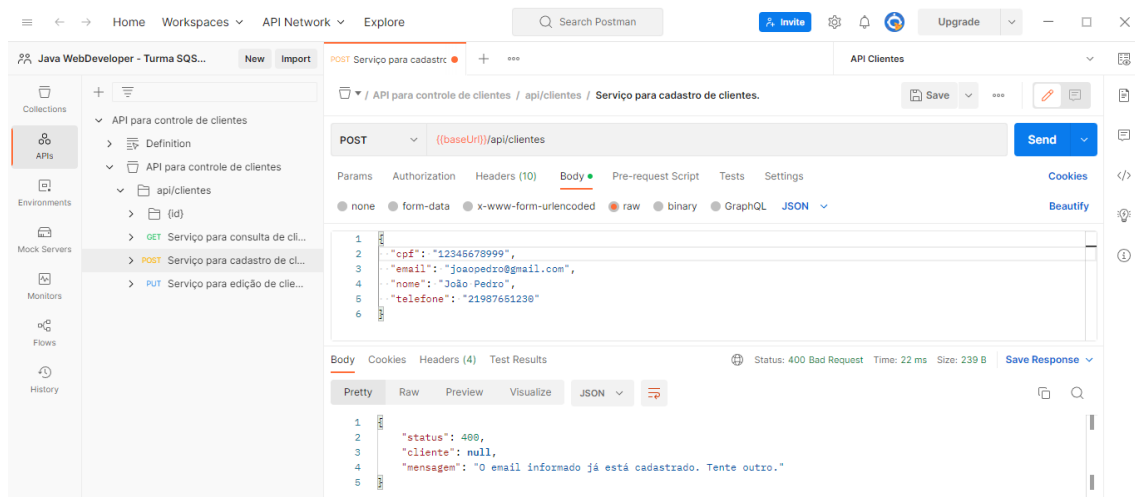


```

POST /api/clientes

{
  "cpf": "1234567890",
  "email": "joao.pedro@gmail.com",
  "nome": "João Pedro",
  "telefone": "2198765123"
}

{
  "status": 400,
  "cliente": null,
  "mensagem": "O cpf informado já está cadastrado. Tente outro."
}
  
```



```

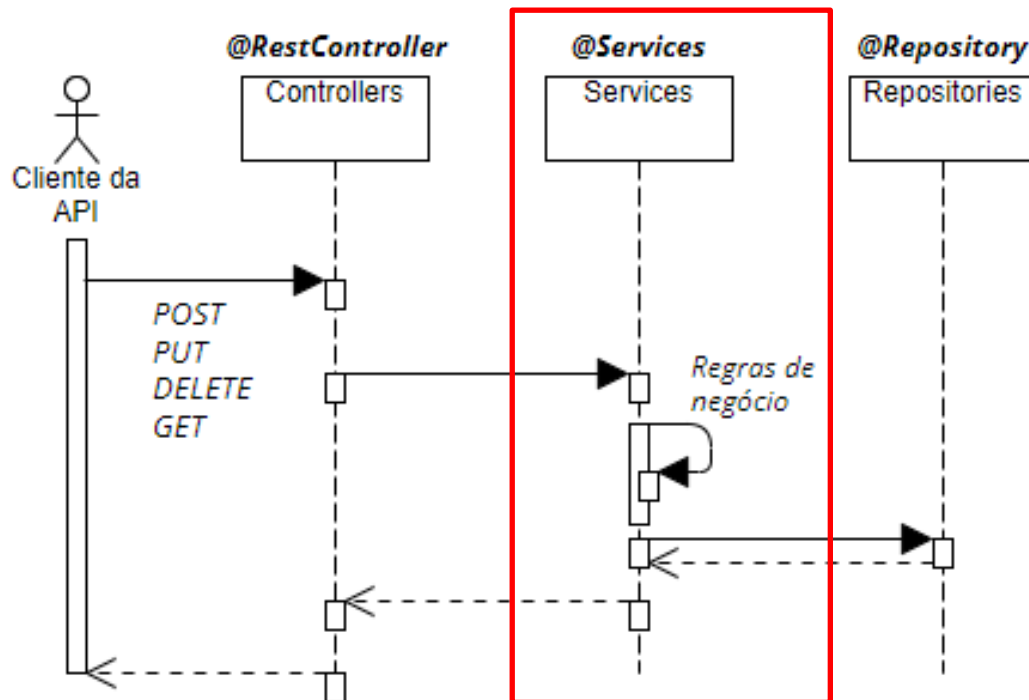
POST /api/clientes

{
  "cpf": "1234567890",
  "email": "joao.pedro@gmail.com",
  "nome": "João Pedro",
  "telefone": "2198765123"
}

{
  "status": 400,
  "cliente": null,
  "mensagem": "O email informado já está cadastrado. Tente outro."
}
  
```

Voltando na camada de serviço, vamos implementar as demais regras de negócio do projeto:

/services/**ClienteService.java**



```

package br.com.cotiinformatica.services;

import java.util.Date;
import java.util.List;
import java.util.Optional;

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

import br.com.cotiinformatica.entities.Cliente;
import br.com.cotiinformatica.repositories.IClienteRepository;

@Service
public class ClienteService {

    // injeção de dependência
    @Autowired
    private IClienteRepository clienteRepository;

    // Método para realizar o cadastro de um cliente
    public void cadastrar(Cliente cliente) {

        // verificar se o email do cliente já
        // está cadastrado no banco de dados
        if (clienteRepository.findByEmail
            (cliente.getEmail()) != null)
    
```

```
        throw new IllegalArgumentException("O email  
            informado já está cadastrado. Tente outro.");  
  
        // verificar se o cpf do cliente já  
        // está cadastrado no banco de dados  
        if (clienteRepository.findByCpf(cliente.getCpf()) != null)  
            throw new IllegalArgumentException("O cpf informado  
                já está cadastrado. Tente outro.");  
  
        // gerando a data de cadastro do cliente  
        cliente.setDataCadastro(new Date());  
  
        // gravando no banco de dados  
        clienteRepository.save(cliente);  
    }  
  
    // Método para realizar a edição de um cliente  
    public void atualizar(Cliente cliente) {  
  
        // buscando o cliente no banco de dados através do id  
        Optional<Cliente> optional = clienteRepository.findById  
            (cliente.getIdCliente());  
  
        // verificar se o cliente existe no banco de dados  
        if(optional.isEmpty())  
            throw new IllegalArgumentException("Cliente não  
                encontrado. Verifique o ID informado.");  
  
        // buscar o cliente no banco de dados através do Cpf  
        Cliente clientePorCpf = clienteRepository  
            .findByCpf(cliente.getCpf());  
  
        if(clientePorCpf != null  
            && clientePorCpf.getIdCliente() != cliente.getIdCliente())  
            throw new IllegalArgumentException("O CPF informado  
                já está cadastrado para outro cliente.  
                Tente outro.");  
  
        // buscar o cliente no banco de dados através do Email  
        Cliente clientePorEmail = clienteRepository.findByEmail  
            (cliente.getEmail());  
  
        if(clientePorEmail != null  
            && clientePorEmail.getIdCliente()  
            != cliente.getIdCliente())  
            throw new IllegalArgumentException  
                ("O Email informado já está cadastrado  
                para outro cliente. Tente outro.");  
  
        cliente.setDataCadastro(optional.get()  
            .getDataCadastro());  
    }
```



```
//atualizando o cliente no banco de dados
clienteRepository.save(cliente);
}

// Método para excluir o cliente
public Cliente excluir(Integer idCliente) {

    // buscar o cliente no banco de dados através do id
    Optional<Cliente> optional = clienteRepository
        .findById(idCliente);

    // verificando se o cliente foi encontrado
    if(optional.isEmpty())
        throw new IllegalArgumentException("Cliente não
            encontrado. Verifique o ID informado.");

    //capturando o cliente
    Cliente cliente = optional.get();

    //excluindo o cliente no banco de dados
    clienteRepository.delete(cliente);

    //retornando os dados do cliente
    return cliente;
}

// Método para consultar todos os clientes
public List<Cliente> consultarTodos() {

    // buscar os clientes cadastrados no banco de dados
    return clienteRepository.findAll();
}

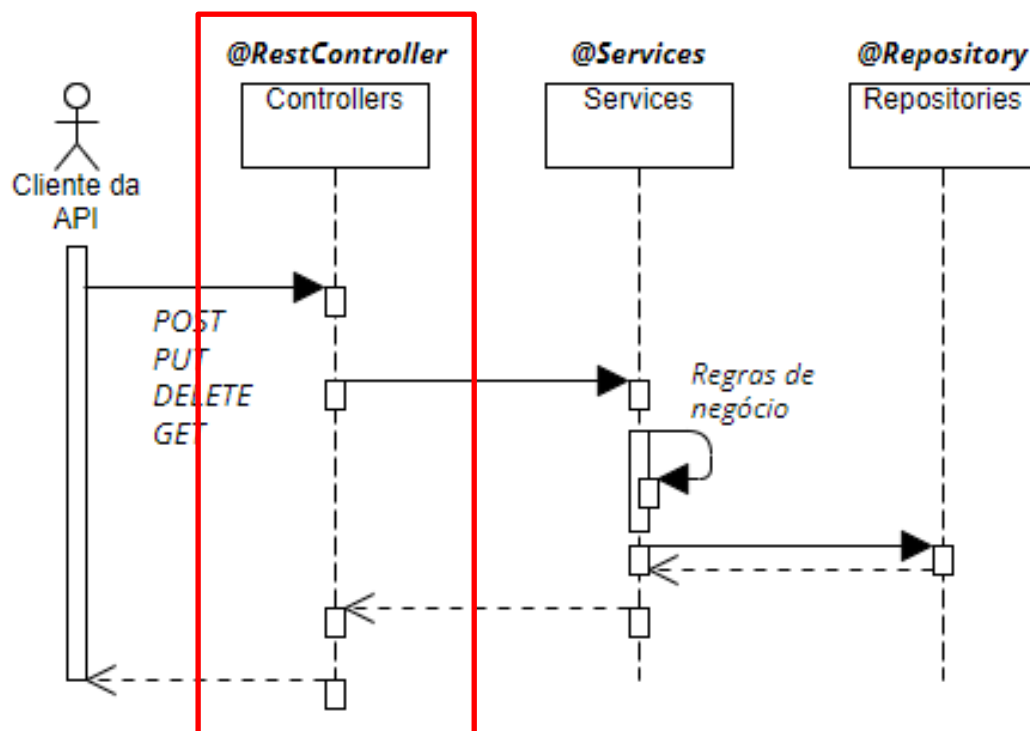
// Método para consultar 1 cliente através do id
public Cliente obterPorId(Integer idCliente) {

    // buscar o cliente através do ID
    Optional<Cliente> optional = clienteRepository
        .findById(idCliente);

    //verificar se o cliente foi encontrado
    if(optional.isPresent())
        //retornando os dados do cliente
        return optional.get();

    return null;
}
}
```

Voltando para o controlador:



```
package br.com.cotiinformatica.controllers;
```

```
import java.util.List;
```

```
import javax.validation.Valid;
```

```
import org.modelmapper.ModelMapper;
import org.modelmapper.TypeToken;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
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.RestController;
```

```
import br.com.cotiinformatica.dtos.GetClientesDTO;
import br.com.cotiinformatica.dtos.PostClientesDTO;
import br.com.cotiinformatica.dtos.PutClientesDTO;
import br.com.cotiinformatica.dtos.ResponseClientesDTO;
import br.com.cotiinformatica.entities.Cliente;
import br.com.cotiinformatica.services.ClienteService;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
```

```
@Api(tags = "Clientes")
@RestController
public class ClientesController {
```

```
// injeção de dependência
@Autowired
private ClienteService clienteService;

@ApiOperation("Serviço para cadastro de clientes.")
@PostMapping("/api/clientes")
public ResponseEntity<ResponseClientesDTO>
    post(@Valid @RequestBody PostClientesDTO dto) {

    ResponseClientesDTO response = new ResponseClientesDTO();

    try {

        ModelMapper modelMapper = new ModelMapper();
        Cliente cliente = modelMapper.map(dto, Cliente.class);

        clienteService.cadastrar(cliente);

        response.setStatus(201);
        response.setMensagem("Cliente cadastrado com sucesso.");
        response.setCliente(modelMapper.map
            (cliente, GetClientesDTO.class));

        return ResponseEntity.status(HttpStatus.CREATED).body(response);
    } catch (IllegalArgumentException e) {

        response.setStatus(400);
        response.setMensagem(e.getMessage());

        return ResponseEntity.status
            (HttpStatus.BAD_REQUEST).body(response);
    } catch (Exception e) {

        response.setStatus(500);
        response.setMensagem(e.getMessage());

        return ResponseEntity.status
            (HttpStatus.INTERNAL_SERVER_ERROR).body(response);
    }
}

@ApiOperation("Serviço para edição de clientes.")
@PutMapping("/api/clientes")
public ResponseEntity<ResponseClientesDTO> put
    (@Valid @RequestBody PutClientesDTO dto) {

    ResponseClientesDTO response = new ResponseClientesDTO();

    try {

        ModelMapper modelMapper = new ModelMapper();
        Cliente cliente = modelMapper.map(dto, Cliente.class);

        clienteService.atualizar(cliente);

        response.setStatus(200);
        response.setMensagem("Cliente atualizado com sucesso.");
        response.setCliente
            (modelMapper.map(cliente, GetClientesDTO.class));

        return ResponseEntity.status(HttpStatus.OK).body(response);
    }
}
```

```
} catch (IllegalArgumentException e) {

    response.setStatus(400);
    response.setMensagem(e.getMessage());

    return ResponseEntity.status
        (HttpStatus.BAD_REQUEST).body(response);

} catch (Exception e) {

    response.setStatus(500);
    response.setMensagem(e.getMessage());

    return ResponseEntity.status
        (HttpStatus.INTERNAL_SERVER_ERROR).body(response);

}

@ApiOperation("Serviço para exclusão de clientes.")
@DeleteMapping("/api/clientes/{id}")
public ResponseEntity<ResponseClientesDTO> delete
    (@PathVariable("id") Integer idCliente) {

    ResponseClientesDTO response = new ResponseClientesDTO();

    try {

        Cliente cliente = clienteService.excluir(idCliente);

        ModelMapper modelMapper = new ModelMapper();

        response.setStatus(200);
        response.setMensagem("Cliente excluído com sucesso.");
        response.setCliente(modelMapper.map
            (cliente, GetClientesDTO.class));

        return ResponseEntity.status(HttpStatus.OK).body(response);

    } catch (IllegalArgumentException e) {

        response.setStatus(400);
        response.setMensagem(e.getMessage());

        return ResponseEntity.status
            (HttpStatus.BAD_REQUEST).body(response);

    } catch (Exception e) {

        response.setStatus(500);
        response.setMensagem(e.getMessage());

        return ResponseEntity.status
            (HttpStatus.INTERNAL_SERVER_ERROR).body(response);

    }

}

@ApiOperation("Serviço para consulta de clientes.")
@GetMapping("/api/clientes")
public ResponseEntity<List<GetClientesDTO>> getAll() {

    ModelMapper modelMapper = new ModelMapper();
```

```
List<Cliente> clientes = clienteService.consultarTodos();
List<GetClientesDTO> clientesDTO = modelMapper.map
    (clientes, new TypeToken<List<GetClientesDTO>>() {}.getType());

return ResponseEntity.status(HttpStatus.OK).body(clientesDTO);
}

@ApiOperation("Serviço para consulta de cliente por id.")
@GetMapping("/api/clientes/{id}")
public ResponseEntity<GetClientesDTO> getById
    (@PathVariable("id") Integer idCliente) {

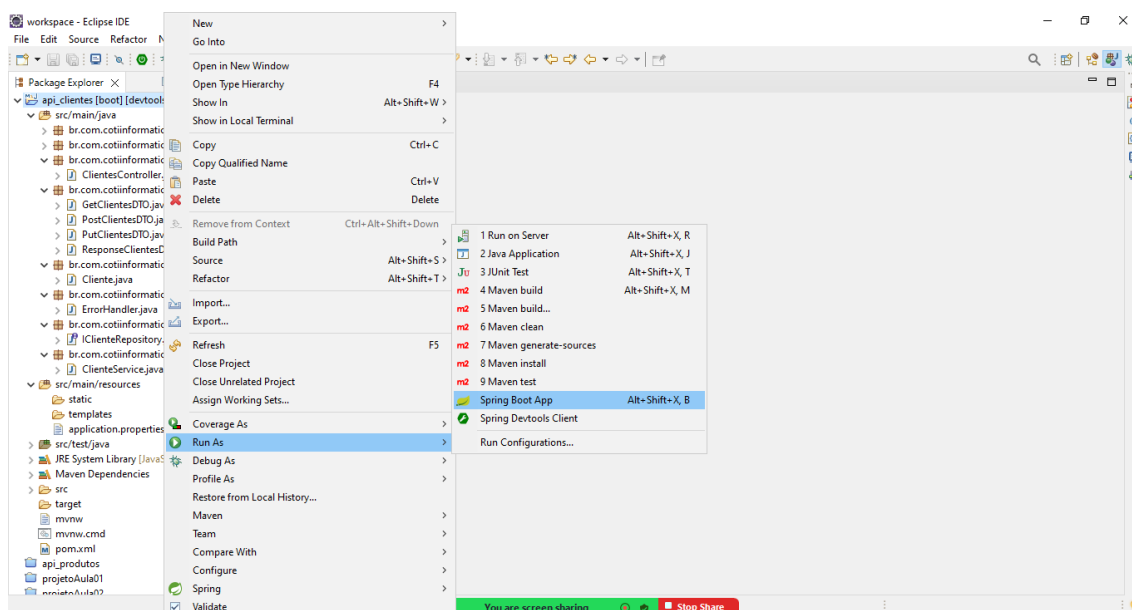
    Cliente cliente = clienteService.obterPorId(idCliente);

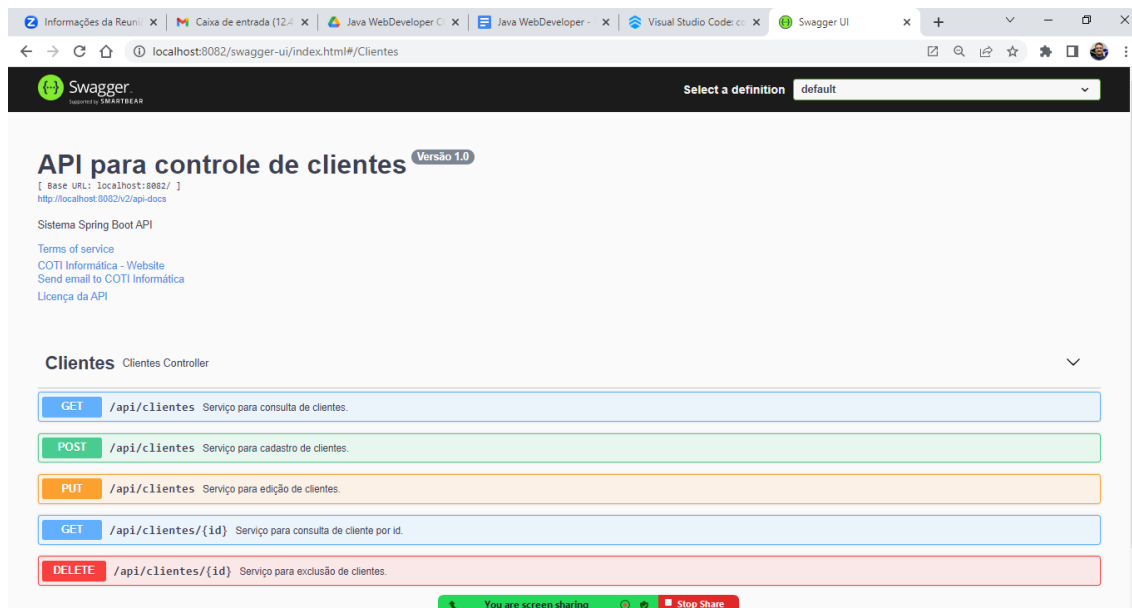
    if(cliente != null) {

        ModelMapper modelMapper = new ModelMapper();
        GetClientesDTO clienteDTO
            = modelMapper.map(cliente, GetClientesDTO.class);

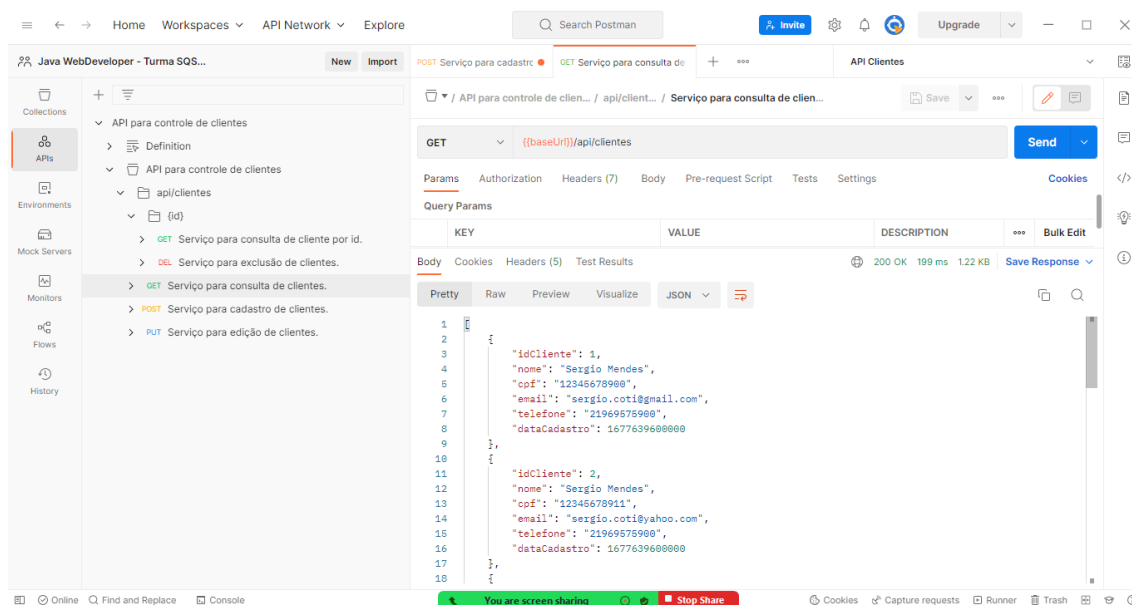
        return ResponseEntity.status(HttpStatus.OK).body(clienteDTO);
    }
    else {
        return ResponseEntity.status(HttpStatus.NO_CONTENT).body(null);
    }
}
}
```

Executando e testando:





No POSTMAN:



Retorno da consulta:

```

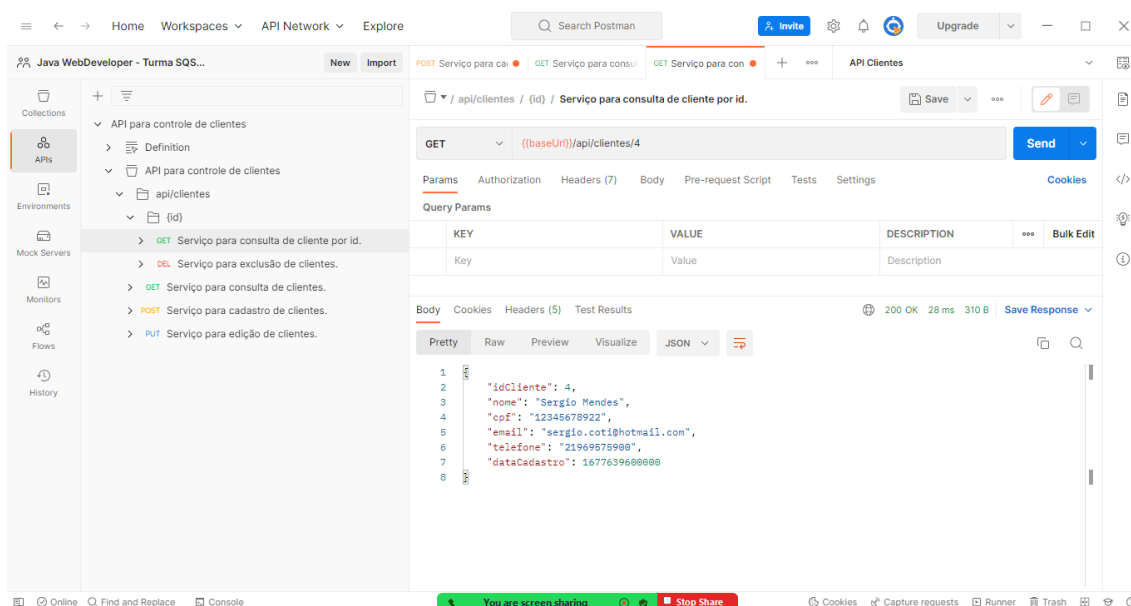
[
  {
    "idCliente": 1,
    "nome": "Sergio Mendes",
    "cpf": "12345678900",
    "email": "sergio.coti@gmail.com",
    "telefone": "21969575900",
    "dataCadastro": 1677639600000
  },
  {
  
```

```

    "idCliente": 2,
    "nome": "Sergio Mendes",
    "cpf": "12345678911",
    "email": "sergio.coti@yahoo.com",
    "telefone": "21969575900",
    "dataCadastro": 1677639600000
  },
  {
    "idCliente": 3,
    "nome": "Sergio Mendes",
    "cpf": "12345678999",
    "email": "sergio.coti@bol.com",
    "telefone": "21969575900",
    "dataCadastro": 1677639600000
  }
]

```

Consultando 1 cliente baseado no ID:



The screenshot shows the Postman interface with a GET request to `{{baseUri}}/api/clientes/4`. The response is a JSON object representing a client.

KEY	VALUE	DESCRIPTION
Key	Value	Description

```

{
  "idCliente": 4,
  "nome": "Sergio Mendes",
  "cpf": "12345678922",
  "email": "sergio.coti@hotmail.com",
  "telefone": "21969575900",
  "dataCadastro": 1677639600000
}

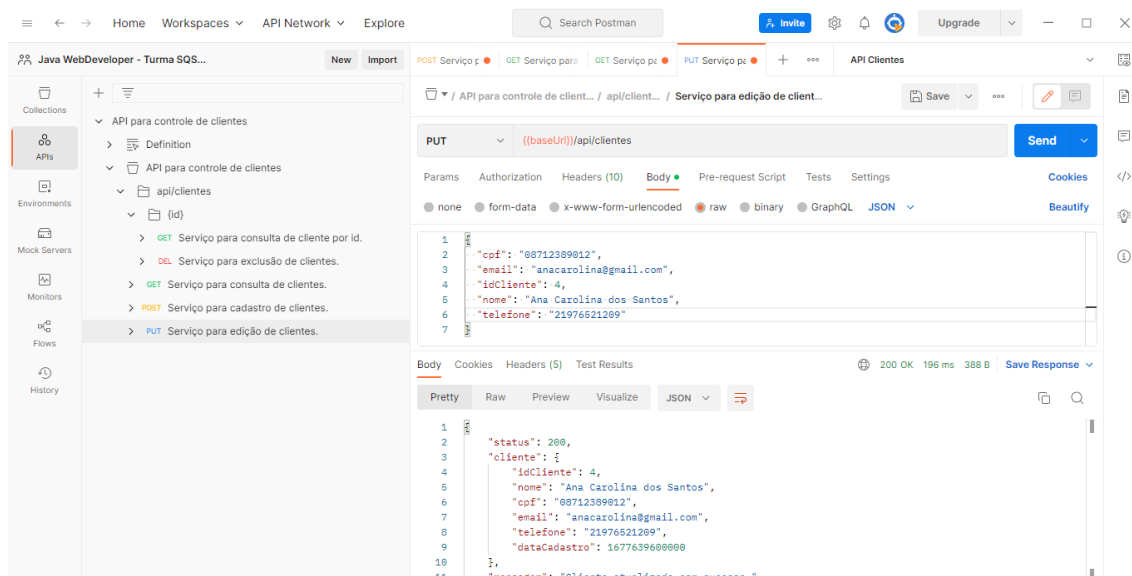
```

```

{
  "idCliente": 4,
  "nome": "Sergio Mendes",
  "cpf": "12345678922",
  "email": "sergio.coti@hotmail.com",
  "telefone": "21969575900",
  "dataCadastro": 1677639600000
}

```

Atualizando:

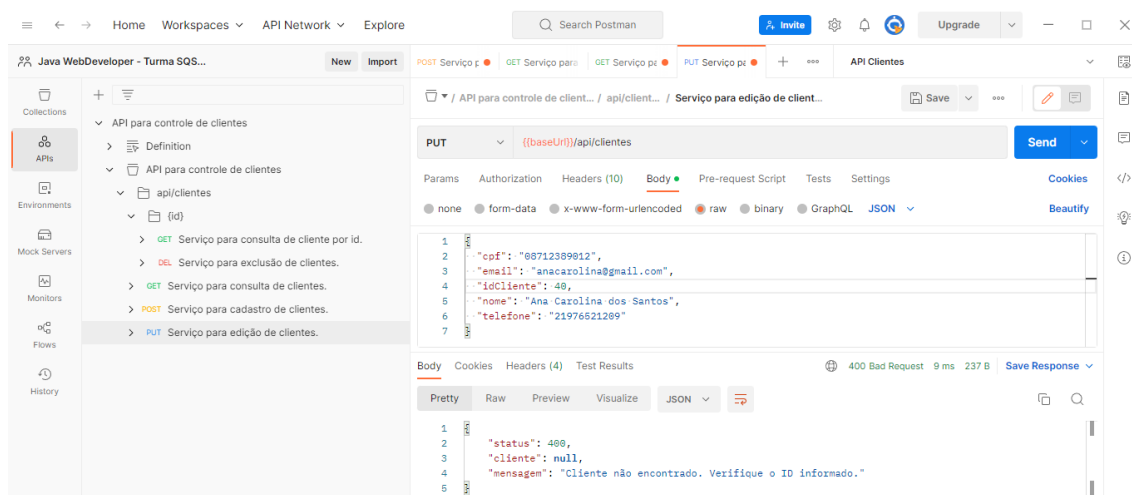


The screenshot shows the Postman interface with a PUT request to `API para controle de clientes / api/clientes`. The request body is a JSON object:

```
1 {
2   "cpf": "087123899012",
3   "email": "anacarolina@gmail.com",
4   "idCliente": 4,
5   "nome": "Ana Carolina dos Santos",
6   "telefone": "21976521289"
7 }
```

The response status is 200 OK. The response body is a JSON object:

```
1 {
2   "status": 200,
3   "cliente": {
4     "idCliente": 4,
5     "nome": "Ana Carolina dos Santos",
6     "cpf": "087123899012",
7     "email": "anacarolina@gmail.com",
8     "telefone": "21976521289",
9     "dataCadastro": "1677639600000"
10  },
11  "mensagem": "Cliente atualizado com sucesso."
12 }
```



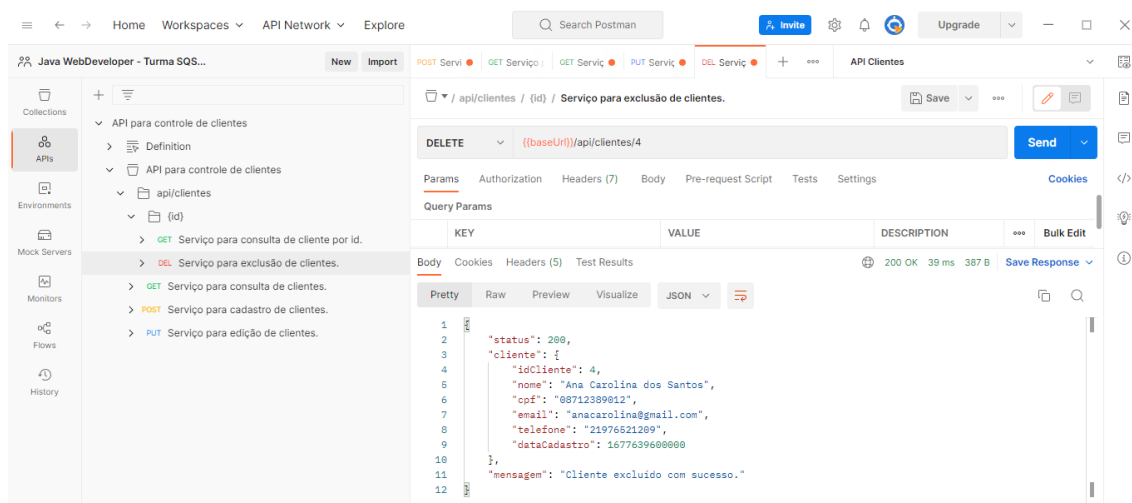
The screenshot shows the Postman interface with a PUT request to `API para controle de clientes / api/clientes`. The request body is a JSON object:

```
1 {
2   "cpf": "087123899012",
3   "email": "anacarolina@gmail.com",
4   "idCliente": 40,
5   "nome": "Ana Carolina dos Santos",
6   "telefone": "21976521289"
7 }
```

The response status is 400 Bad Request. The response body is a JSON object:

```
1 {
2   "status": 400,
3   "cliente": null,
4   "mensagem": "Cliente não encontrado. Verifique o ID informado."
5 }
```

Excluindo:

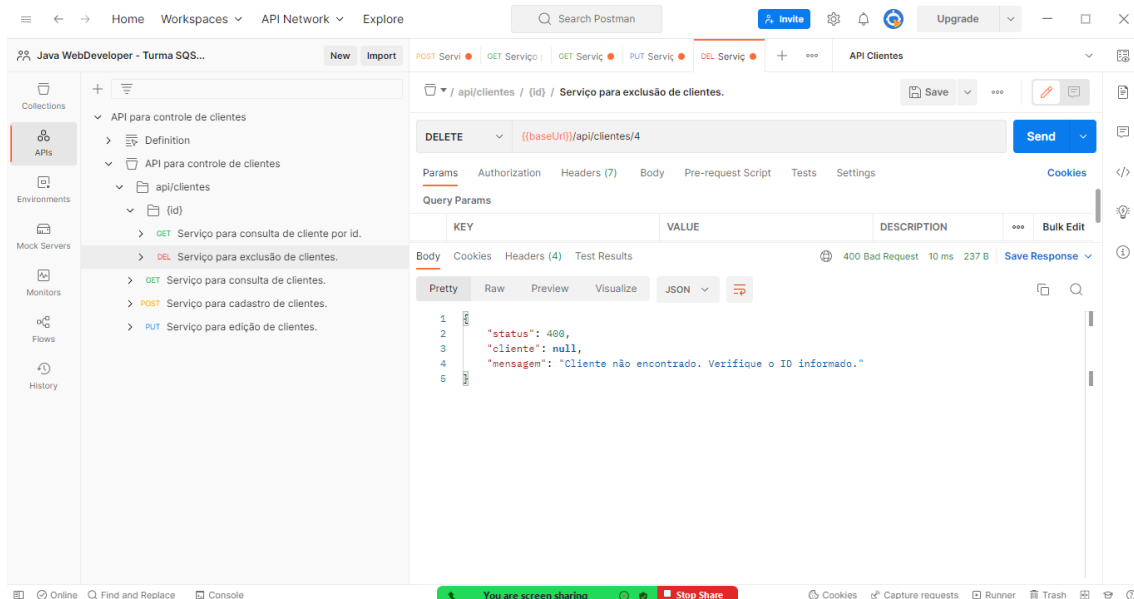


The screenshot shows the Postman interface with a DELETE request to `API para controle de clientes / api/clientes / (id)`. The request body is a JSON object:

```
1 {
2   "cpf": "087123899012",
3   "email": "anacarolina@gmail.com",
4   "idCliente": 40,
5   "nome": "Ana Carolina dos Santos",
6   "telefone": "21976521289"
7 }
```

The response status is 200 OK. The response body is a JSON object:

```
1 {
2   "status": 200,
3   "cliente": {
4     "idCliente": 4,
5     "nome": "Ana Carolina dos Santos",
6     "cpf": "087123899012",
7     "email": "anacarolina@gmail.com",
8     "telefone": "21976521289",
9     "dataCadastro": "1677639600000"
10  },
11  "mensagem": "Cliente excluído com sucesso."
12 }
```

Publicando o trabalho no GITHUB:

```
Samsung@DESKTOP-P9F6D9F MINGW64 ~/Desktop/COTI - Aulas/2023 -
Java WebDeveloper SQS 18h as 22h (Início em
09.01)/workspace/api_clientes (main)
```

```
$ git add .
```

```
warning: in the working copy of 'pom.xml', LF will be replaced
by CRLF the next time Git touches it
```

```
Samsung@DESKTOP-P9F6D9F MINGW64 ~/Desktop/COTI - Aulas/2023 -
Java WebDeveloper SQS 18h as 22h (Início em
09.01)/workspace/api_clientes (main)
```

```
$ git commit -m 'Serviços, BeanValidations e conclusão da API'
```

```
[main ee62502] serviços, BeanValidations e conclusão da API
7 files changed, 224 insertions(+), 21 deletions(-)
```

```
create mode 100644
```

```
src/main/java/br/com/cotiinformatica/handlers/ErrorHandler.java
```

```
Samsung@DESKTOP-P9F6D9F MINGW64 ~/Desktop/COTI - Aulas/2023 -
Java WebDeveloper SQS 18h as 22h (Início em
09.01)/workspace/api_clientes (main)
```

```
$ git push -u origin main
```

```
Enumerating objects: 36, done.
```

```
Counting objects: 100% (36/36), done.
```

```
Delta compression using up to 8 threads
```

```
Compressing objects: 100% (15/15), done.
```

```
Writing objects: 100% (20/20), 3.96 KiB | 1014.00 KiB/s, done.
```

```
Total 20 (delta 5), reused 0 (delta 0), pack-reused 0
```

```
remote: Resolving deltas: 100% (5/5), completed with 4 local
objects.
```

```
To https://github.com/smendescoti/api_clientes.git
```

```
d8c1eb1..ee62502 main -> main
```

```
branch 'main' set up to track 'origin/main'.
```