# Desafio WireCard

By: Matheus Correia Politano

# Sobre Projeto

Neste projeto apresentarei o passo a passo de uma Web API. Essa API irá validar o pagamento e vai gerar uma resposta em Json. Neste projeto não fiz a utilização de banco de dados, a fim de facilitar a execução da API. Também optei por não abordar o Docker na API, pela simplicidade do projeto.

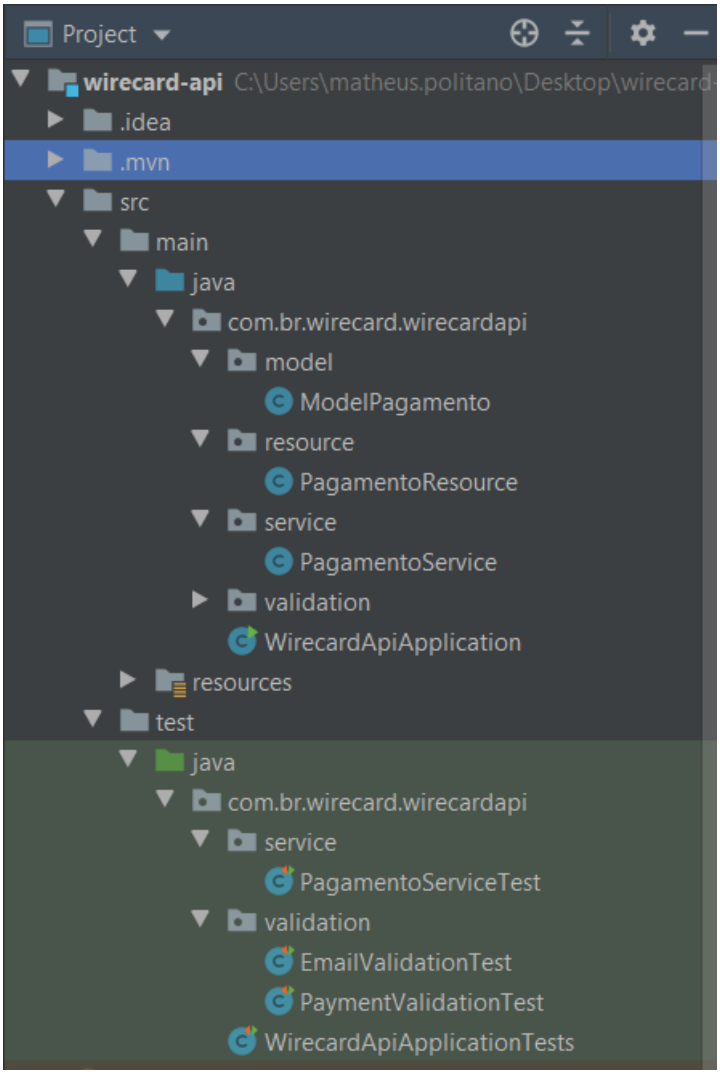
# Requisitos mínimos

- Apache Maven 3.6
- Java 11
- Intellij IDEA (Opcional)

# Começando o projeto

O projeto foi criado com o Spring Boot que fornece a maioria dos componentes baseados no Spring necessários em aplicações em geral de maneira pré-configurada, tornando possível termos uma aplicação rodando em produção rapidamente com o esforço mínimo de configuração e implantação.

## Estrutura do desafio




### MODEL

A primeira etapa para a API é criar um model. No meu caso o nome escolhido foi ModelPagamento

```
package com.br.wirecard.wirecardapi.model;
public class ModelPagamento {

    private Long id;

    private String name;

    private String cpf;

    private String email;

    private Integer amount;

    private Boolean type;

    private String card;

    private String cardHolderName;

    private String cardNumber;

    private String cardExpirationDate;

    private String cardCVV;
```

## Explicando o código

Aqui são todas as variáveis que vão guardar os valores da request. Vou explicar todas variáveis.

- id :Valor que vai identificar a validação
- name :Recebe o nome do cliente
- cpf: Recebe o cpf  do cliente
- email:Recebe o email do cliente
- amount: Recebe o valor
- type: Essa variável é responsável por determina para api qual a forma do pagamento caso true o pagamento será cartão e caso for false será boleto

Essas variáveis não são obrigatórias caso a forma de pagamento seja boleto7

- card:Recebe o nome da bandeira do cartão
- cardHolderName:Recebe o nome do titular do cartão
- cardExpirationDate:Recebe a data de validade do cartão
- cardCVV:Recebe o CVV do cartão

# VALIDATION

O VALIDATION vai aplicar as regras de negócio para a API. Essas regras vão determinar se irá ocorrer a transação.

## InitValidation

O InitValidation vai contém as variáveis que serão usadas para a manipulação dos dados recebidos do model

```
public class initValidation extends ModelPagamento {
    List<String> responseList = new ArrayList<String>();
    private boolean havaError;
    private List<String> erros = new ArrayList<String>();
    private List<String> success = new ArrayList<String>();
```

Explicando o código

- responseList :Vai armazenar os valores que serão enviados no Json
- haveError: Determina se houve erro ou não
- erros: Armazena todos os erros caso existem
- success: Armazena as variáveis caso o pagamento seja bem sucedida

## EmailValidation

Essa etapa vai fazer a validação do email

```
public class emailValidation extends initValidation {
             public void isEmail() {

            String regex = "^[\\w-_\\.+]*[\\w-_\\.]\\@([\\w]+\\.)+[\\w]+[\\w]$";
            this.setHavaError(!getEmail().matches(regex));
            if (isHavaError()==true){
                setErros("Email invalido");
            }
```

Explicando o código

A variável String regex que vai ser utilizado posteriormente para verificar se o email é valido ou não. Caso não seja envia o valor True para a variável haveError e enviar qual foi o erro para a List erros. O método matches informa se essa sequência corresponde ou não à expressão regular especificada. Uma invocação desse método da forma str.matches (regex) produz exatamente o mesmo resultado que a expressão Pattern.matches (regex, str).

## PaymentValidation

Essa etapa vai fazer a validação do pagamento via cartão

```
public class paymentValidation extends emailValidation{
    /*Esse método aplica as regras
    de negocio para o pagamento
    * */
    public void payment(){
        /*O tipo type caso seja true determina que o
        * pagamento será feito através
        * do cartão de crédito, e caso seja
        * false a transação ocorrerá via
        * boleto
        * */
        if(this.getType() == true){
            Calendar c = new GregorianCalendar();
```

```
            Calendar f = Calendar.getInstance();
            /*A primeira regra de negocio é verificar se as informações
            * do cartão estão preechida caso nenhum campo seja null a regra de
negócio
            * irá para a segunda etapa. Nessa etapa ocorre também a verificação
no preenchimento adequado
            * de alguns campos como CVV, data de validade do cartão e o numero
do cartão
            * */
            if(this.getCardCVV() != null && this.getCard() != null &&
this.getCardExpirationDate() != null &&this.getCardHolderName() != null
&&this.getCardNumber() != null && this.getCardCVV().length() == 3 &&
this.getCard() != null && this.getCardExpirationDate().length() == 7
&&this.getCardNumber().length() == 12 ){
                /*A segunda faze da validação é verificar se o nome do comprador
                * e o nome do titular do cartão são correspondente, caso o
contrario o cartão
                * negará a transação
                * */
                if(this.getCardHolderName().equals(this.getName())){
                try {

c.set(Calendar.YEAR,Integer.parseInt(getCardExpirationDate().split("/")[1]));

c.set(Calendar.MONTH,Integer.parseInt(getCardExpirationDate().split("/")[0]));
                    System.out.println(c.after(f));
                }catch(Exception ex){
                    this.setErros("Formato invalido da data de validade do
cartão");
                }
                /*A terceira etapa da regra de negocio é verificar se a data de
validae é superior a data atual
                * caso a data seja anterior ao da campra a regra de negocio
negara a negociação*/
                 if(c.after(f)){
                     if(isHavaError()==false)
                     {
                         this.setSuccess("Cartão");
                         this.setSuccess("Pagamento aprovado");
                     }
                 }else {
                     this.setHavaError(true);
                     this.setErros("Transação negada : Cartão invalido, pois já
está vencido");

                 }

             }else {

                 this.setHavaError(true);
                 this.setErros("Transação negada : O nome de cadastro é
diferente do titular do cartão");

             }
```

```
        }else{
            this.setHavaError(true);
            this.setErros("dados do cartão imcompleto ou invalidos");
        }
```

Explicando o código

Esse código é responsável pela validação do pagamento via cartão. O primeiro passo é verificar se o type recebe o valor True. Após essa verificação o método vai levar até a primeira regra de negócio, essa regra é simples apenas analisa todas as variáveis do cartão, essas variáveis não podem ser nulas ou receber valores inválidos. A segunda regra de negocio verifica se o nome do cliente é o mesmo do titular do cartão caso não seja enviaremos uma mensagem de pagamento negado. A última regra de negocio analisa da data de vencimento do cartão e caso essa data seja anterior ao data atual a API rejeitara a compra.

## Pagamento via Boleto

```
/*======================BOLETO=====================================================
    *Para chegar até aqui é necessário que o type seja false
    *
    *
    * Essa if  é responsável pela validação dos dados
    * para geração do bolero
    * */
    if (getCpf().length()==11 && getName()!= null && getCpf() != "0")
    {
     /*Aqui é gerado o numero do boleto
     * */
    Integer rand_int1 = ThreadLocalRandom.current().nextInt();
    while (rand_int1 < 0){
        rand_int1 = ThreadLocalRandom.current().nextInt();
    }
    this.setSuccess("Boleto");
    this.setSuccess(rand_int1.toString());
    this.getErros();
```

```
    /*Todos os campos que não fazem parte
    * do scopo do boleto serão atribuidos o valor
    * null*/
    setCard(null);
    setCardCVV(null);
    setCardExpirationDate(null);
    setCardHolderName(null);
    setCardNumber(null);
}
    else   {
        this.setHavaError(true);
        this.setErros("CPf invalido ou nome null");
    }
}
```

Explicando o código

Esse código é responsável pela validação do pagamento via boleto. O primeiro passo é verificar se o type recebe o valor False. Precisamos enviar um código para o nosso cliente esse código vai receber uma valor aleatório que será armazenado em nossa List success. Perceve que todas variáveis que não são necessárias para o pagamento via boleto receberam o valor null.

## PagamentoService

No interior dessa classe será feito a união das validações para enviar ao Resource

```
public class PagamentoService extends paymentValidation {


    public void validaçao(){
        try {
            isEmail();
            payment();
        }catch (NullPointerException ex){
            this.setHavaError(true);
            this.setErros("Campos Incopletos ou preenchimento invalido");
        }
```

```
        if(isHavaError()==false) {
            setResponseList(getSuccess());

        }else{
            setResponseList(getErros());
        }
```

Explicando o código

O método validacao vai chamar todos os outros métodos de validacao. Caso seja lançada em expection do tipo NullPointerException é porque algum campo não foi preenchido de forma correta . O if  desse código tem como função controlar qual List será enviado como Json. E para saber qual mensagem ele vai enviar verifica-se a variável haveError case seja true ele enviará a mensagem de erro e caso o contrário ira mandar as informações do pagamento.

## PagamentoResource

```
public class PagamentoResource {
    @PostMapping(value = "/")

    public ResponseEntity<List<String>> persist(@RequestBody final
PagamentoService pagamentoService, Pagamento pagamento){
        //Executa a validação
        pagamentoService.validaçao();

        List<String> lista = new ArrayList <String>();

        lista = pagamentoService.getResponseList();



        return  new ResponseEntity<>(lista,HttpStatus.OK);
    }
```

Explicando o código

Aqui recebermos as informações enviadas pra a API e para receber essas informações através do Post  atribuímos @RequestBody e a classe dento do persist. O primeiro código do método é responsável por chamar o método

validação do PagamentoService. Depois criamos uma List que vai receber o responseList, essa variável vai ser enviada como resposta.

## Teste unitário

# EmailValidationTest

```
public class EmailValidationTest extends EmailValidation {

    @Test
    /*Esse metodo tem como objetivo testar
    * a função isEmail que valida os es email na aplicação
    * atribuimos um email invalido e a função tem que retornar True
    * */
    public void testaEmailInvalido(){
        setEmail("fsdfsdfsdfsdfsdf");
        isEmail();
        Boolean teste =this.isHavaError();


        assertThat(teste).isEqualTo(true);

    }
    @Test
    /*Esse metodo tem como objetivo testar
     * a função isEmail que valida os es email na aplicação
     * atribuimos um email valido e a função tem que retornar false
     * */
    public void testaEmaiValido(){
        setEmail("matheusteste@gmail.com");
        isEmail();
        Boolean teste =this.isHavaError();


        assertThat(teste).isEqualTo(false);

    }
```

Explicando o código

O primeiro método de teste é executado esperando que o erro aconteça, perceba que o email atribuído é invalido, portando irá atribuir True para a variável. O segundo método teste recebe um valor valido portanto  o valor false para na variável

# PaymentValidationTest

```
public class PaymentValidationTest extends PaymentValidation {

    /*Esse teste tem como função
    * verificar se o type for false
    * sera gerado um numero para o boleto*/
    @Test
    public void gerarBoleto()
    {
        this.setAmount(50);
        this.setName("Matheus");
        this.setCpf("49073517800");
        this.setType(false);
        this.payment();
        assertThat(getSuccess().size()).isEqualTo(2);

    }
    @Test
    public void erroParaGerarBoleto()
    {
        this.setAmount(50);
        this.setName("Matheus");

        //CPF invalido
        this.setCpf("49073517");

        this.setType(false);
        this.payment();
        assertThat(getErros().size()).isEqualTo(1);

    }
```

Explicando o código

O primeiro método de teste é executado esperando a geração do boleto corretamente, perceba que o campo type é false e que os campos de cartão não precisão serem preenchidos e a variável CPF recebe um CPF válido, portanto List success será copulada. O segundo método recebe um CPF invalido, portanto a List Erros será copulada

```
@Test
public void erroNaValidacaoDoCartaoPorTerNumberNull()
{
    this.setType(true);
    this.setCard("MasterCard");
    /*
    CardNumber recebe null
    this.setCardNumber("123456789123");
     */this.setCardHolderName("Matheus Politano");
     this.setCardExpirationDate("05/2020");
     this.setCardCVV("123");
     this.payment();
     assertThat(getErros().size()).isEqualTo(1);
}
@Test
public void divergenciaNoNomeCard()
{
    this.setName("Matheus Gabriel");
    this.setType(true);
    this.setCard("MasterCard");

    this.setCardNumber("123456789123");
    this.setCardHolderName("Matheus Politano");
    this.setCardExpirationDate("05/2020");
    this.setCardCVV("123");
    this.payment();
    assertThat(getErros().size()).isEqualTo(1);
}
```

Explicando o código

O primeiro método de teste é executado sem a atribuição para o cardNumber, portanto List Erros será copulada. O segundo método recebe todos os dados

corretamente porém os nome do cliente é diferente do nome do titular do cartão o que não irá permitir a transação e a List Erros será copulada.

```
@Test
    public void dataDeVencimentoInvalida()
    {
        this.setName("Matheus Politano");
        this.setType(true);
        this.setCard("MasterCard");

        this.setCardNumber("123456789123");
        this.setCardHolderName("Matheus Politano");
        this.setCardExpirationDate("05/2015");
        this.setCardCVV("123");
        this.payment();
        assertThat(getErros().size()).isEqualTo(1);
    }
    @Test
    public void pagamentoFeitoComSucesso()
    {
        this.setName("Matheus Politano");
        this.setType(true);
        this.setCard("MasterCard");

        this.setCardNumber("123456789123");
        this.setCardHolderName("Matheus Politano");
        this.setCardExpirationDate("05/2020");
        this.setCardCVV("123");
        this.payment();
        assertThat(getSuccess().size()).isEqualTo(2);
    }
```

Explicando o código

O primeiro método de teste é executado com dados invalida de vencimento, portanto a transação não será permitida, e a List Erros será copulada. O segundo método teste é executado com todos os dados atribuído de maneira correta portanto o transação via cartão será feita com sucesso

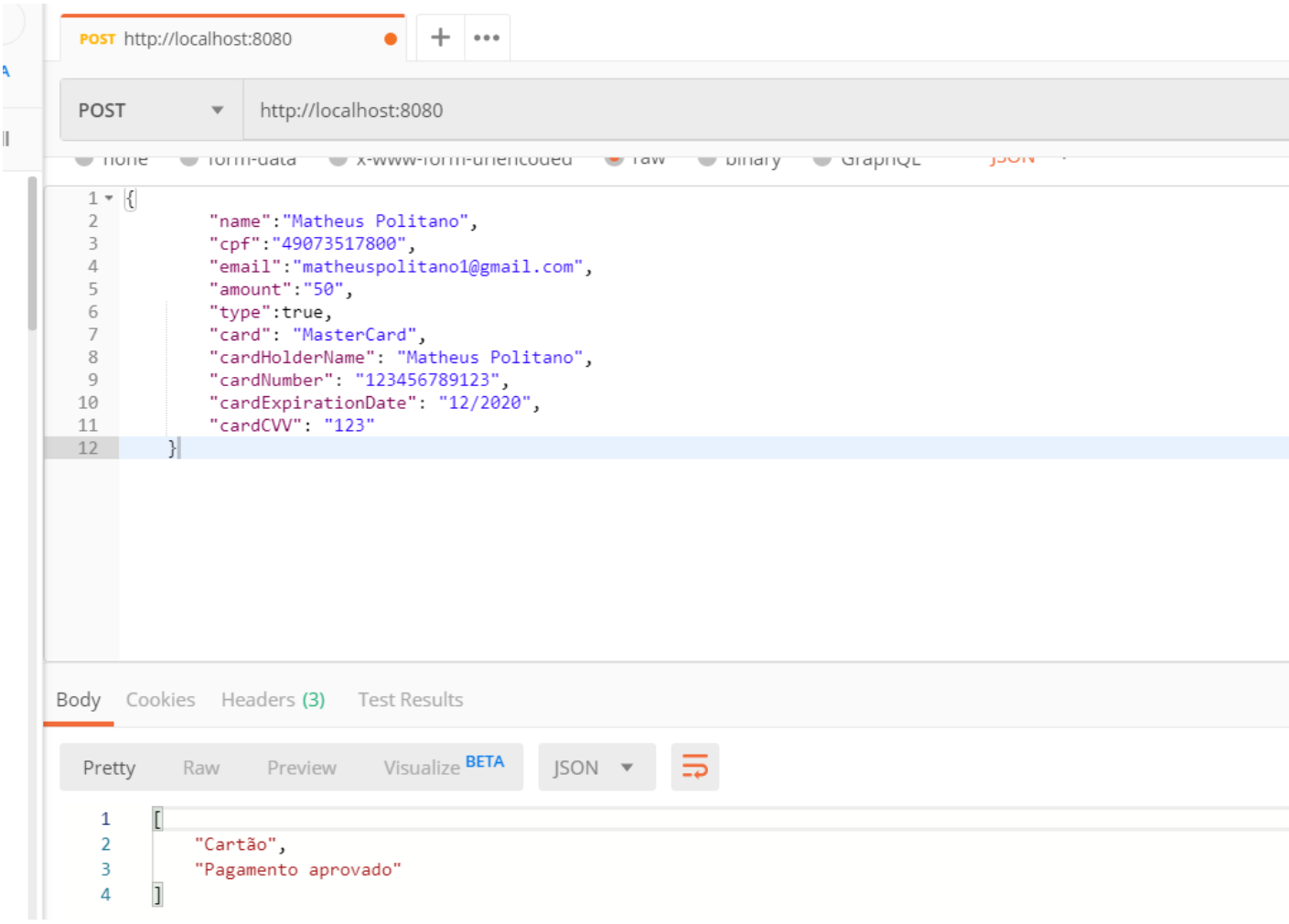
# PagamentoServiceTest

```
@Test
public void tratamentoDeErroComDadosNull()
{

    this.validaçao();
    assertThat(getErros().size()).isEqualTo(1);



}
@Test
public void validacaoPassandoTodosOsParametrosCorretamenteComPagementoNoCartao()
{
    this.setCpf("49073517800");
    this.setAmount(50);
    this.setEmail("matheusteste@gmail.com");
    this.setName("Matheus Politano");
    this.setType(true);
    this.setCard("MasterCard");

    this.setCardNumber("123456789123");
    this.setCardHolderName("Matheus Politano");
    this.setCardExpirationDate("05/2020");
    this.setCardCVV("123");
    this.validaçao();
    System.out.println(this.getSuccess());
    assertThat(this.getSuccess().size()).isEqualTo(2);
}
@Test
public void validacaoPassandoTodosOsParametrosCorretamenteComPagementoNoBoleto()
{
    this.setCpf("49073517800");
    this.setAmount(50);
    this.setEmail("matheusteste@gmail.com");
    this.setName("Matheus Politano");
    this.setType(false);

    this.validaçao();
    System.out.println(this.getSuccess());
    assertThat(this.getSuccess().size()).isEqualTo(2);
}
```

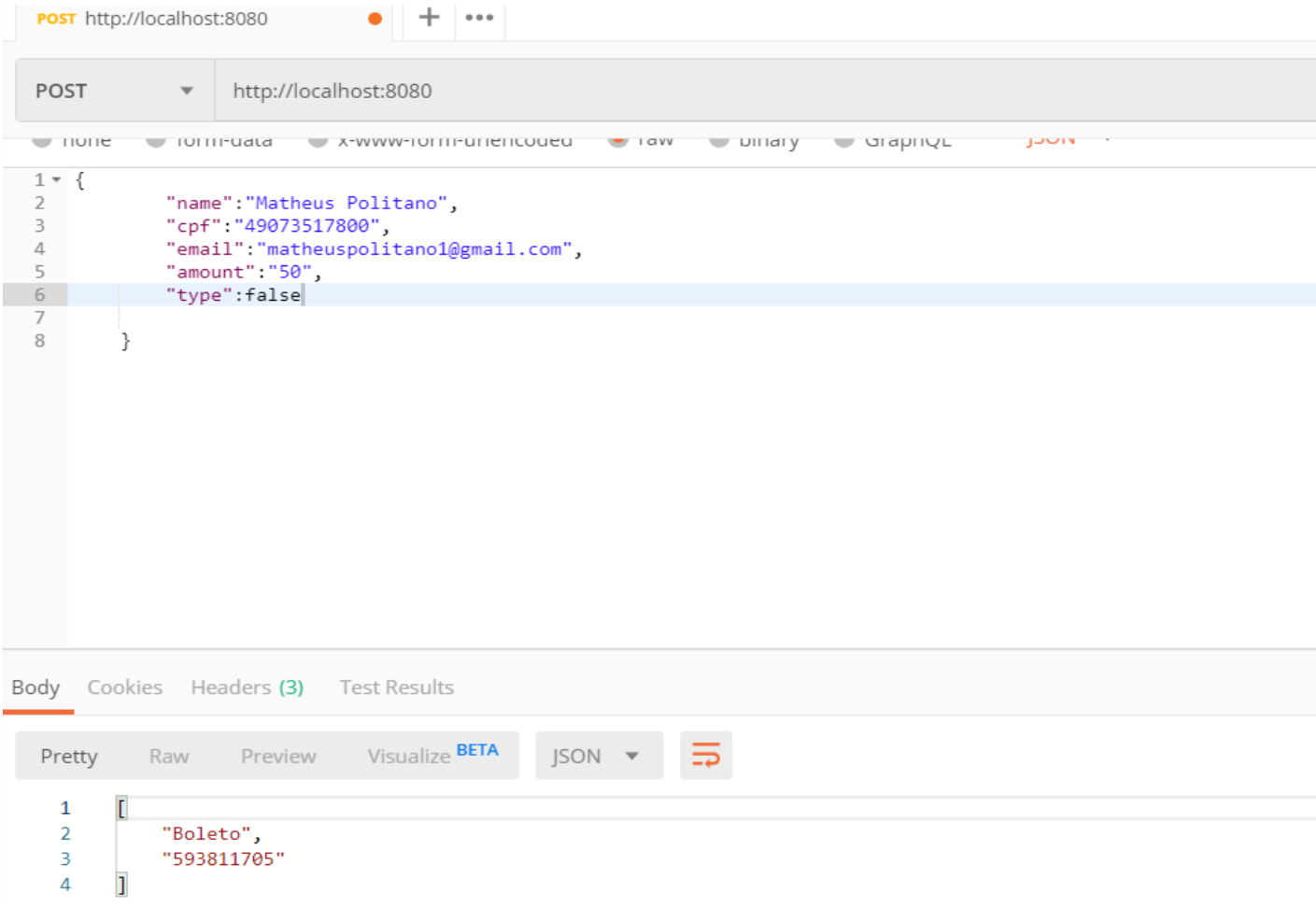
Explicando o código

O primeiro método de teste é executado para cair no catch e copular a List Erros. O segundo método é passado todos os dados válido com o type True para que o pagamento seja por cartão e não haverá nenhum erro. O terceiro método de teste atribui todos os dados corretamente com type false portanto o pagamento será feito via boleto.

# Consumindo a API

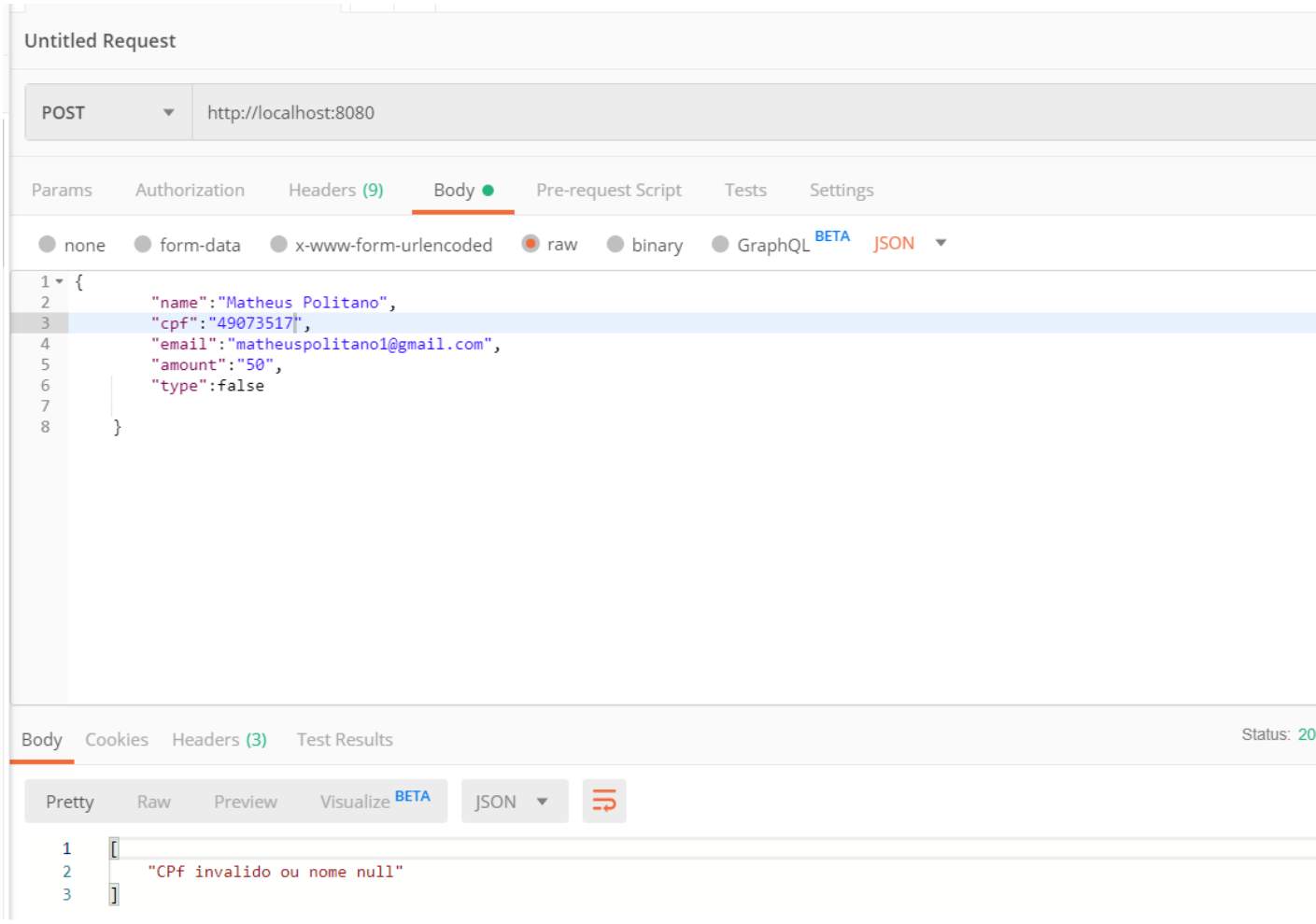



A maneira mais prática para consumir os dados é através do Postman. No exemplo acima eu com o método Http post atribui todos os campos necessários. A API var rodar e gerar uma resposta com o Status

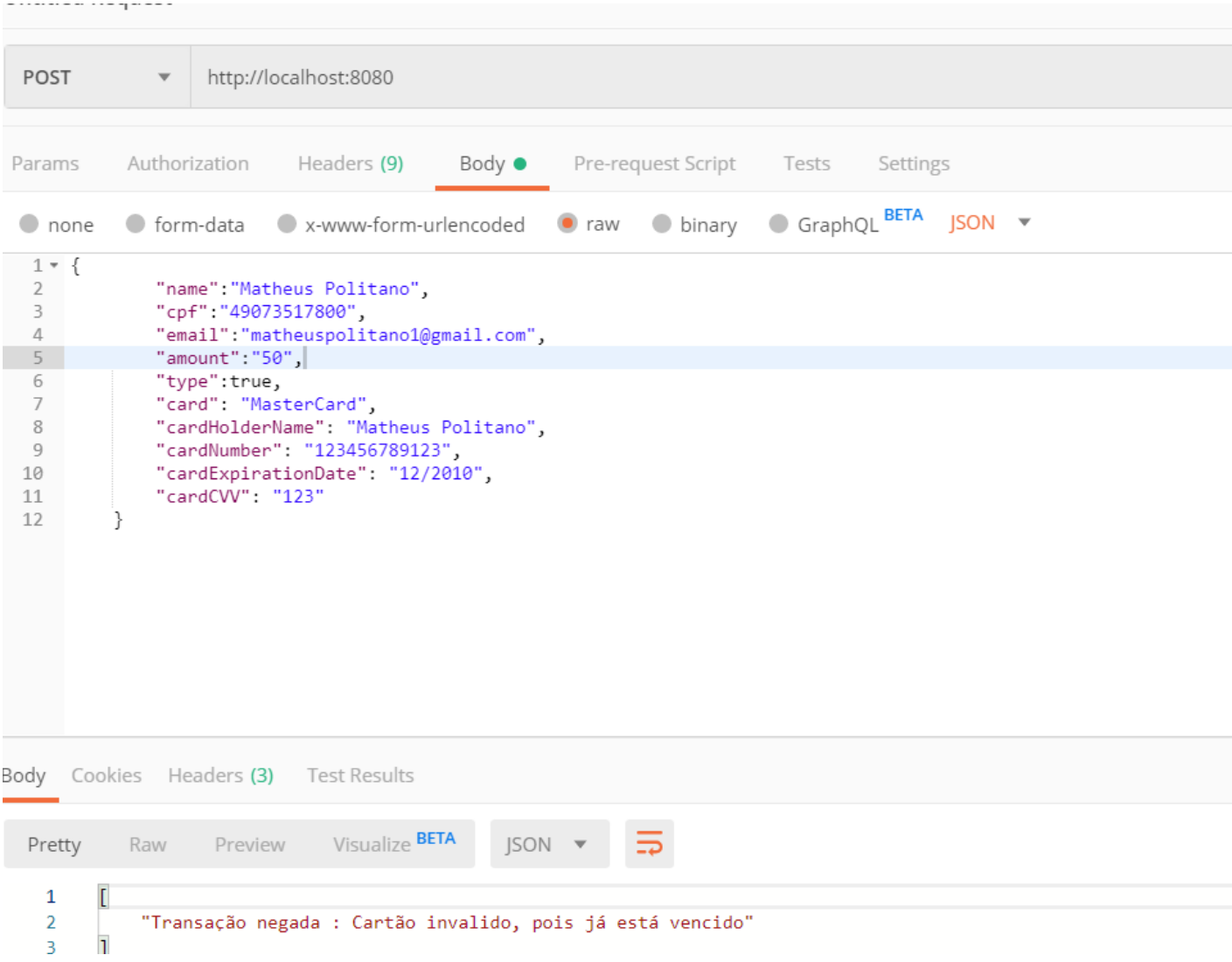



No exemplo acima eu atribui os elementos necessário para gerar o boleto observe que não é necessário preencher os campos que são atrelados ao cartão e o type precisa ser false. A resposta da API  é a forma de pagãmente e o código

do boleto




Acima observamos como a API responde quando atribuímos os dados da maneira incorreta nesse caso o CPF não é válido portanto não gera boleto




No exermplo acima vemos um erro ao tentar fazer uma compra com um cartão que já não é valido, pois a data de valida é inferior a data atual