

**Desenvolvendo serviços de**

**mensageria com RabbitMQ**

**O que é RabbitMQ?**  
  
RabbitMQ é um servidor de mensageria de código aberto (open source), que faz uso do protocolo AMQP (Advanced Message Queuing Protocol). O rabbit é compatível com muitas linguagens de programação e permite lidar com o tráfego de mensagens de forma simples e confiável. Vale falar que também possui uma interface de administração nativa e é multiplataforma.  
  
Um message broker é um sistema que permite que diferentes componentes, como aplicações e aplicativos, se comuniquem entre si, trocando informações. Para isso, geralmente utilizam uma estrutura de fila de mensagens, que será a responsável por armazenar e ordenar mensagens enquanto os consumidores (como aplicativos ou aplicações) não as processam.

/pom.xml

Instalando RabbitMQ

<dependency>

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

<artifactId>**spring-boot-starter-amqp**</artifactId>

</dependency>

<dependency>

<groupId>**org.springframework.amqp**</groupId>

<artifactId>**spring-rabbit-test**</artifactId>

<scope>**test**</scope>

</dependency>

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

<https://www.cloudamqp.com/>

Servidor em nuvem para criação de serviços de mensageria através do RabbitMQ (Message Brokers)

Interface gráfica do usuário, Site

Descrição gerada automaticamente com confiança média

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Tela de celular com publicação numa rede social

Descrição gerada automaticamente

**Criando um servidor de mensageria:**

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**Acessando o servidor:**

Tela de celular com publicação numa rede social

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Precisamos mapear as informações de conexão no servidor do RabbitMQ, para isso vamos usar o arquivo **/application.properties**

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

server.port=8083

spring.datasource.url=jdbc:postgresql://database-coti.cybi4ptzkfaq.us-east-2.rds.amazonaws.com:5432/bd\_apiusuarios

spring.datasource.driver-class-name=org.postgresql.Driver

spring.datasource.username=postgres

spring.datasource.password=coti2023

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format\_sql=true

spring.mail.host=smtp-mail.outlook.com

spring.mail.port=587

spring.mail.username=csharpcoti@outlook.com

spring.mail.password=@Admin12345

spring.mail.properties.mail.smtp.auth=true

spring.mail.properties.mail.smtp.starttls.enable=true

spring.rabbitmq.host=**amqps://uqpqjrqo:PlvqCP2e8FI\_X2j7MjmkRN8UZAGzhdhB@shark.rmq.cloudamqp.com/uqpqjrqo**

queue.name=**mensagens\_usuarios**

**Agora, precisamos criar a fila no servidor para onde serão enviadas as mensagens:**

*queue.name=mensagens\_usuarios*

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Tela de celular com publicação numa rede social

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Email

Descrição gerada automaticamente

**Acessando a fila:**

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Configurando o projeto Spring Boot para usar o RabbitMQ

Criando uma simples configuração:

**package** br.com.cotiinformatica;

**import** org.springframework.amqp.rabbit.annotation.EnableRabbit;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**@EnableRabbit**

@SpringBootApplication

**public** **class** ApiUsuariosApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*

(ApiUsuariosApplication.**class**, args);

}

}

Criando uma classe para configurar o RabbitMQ:

/api/config/RabbitMQConfig.java

**package** br.com.cotiinformatica.api.config;

**import** org.springframework.amqp.core.Queue;

**import** org.springframework.beans.factory.annotation.Value;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

@Configuration

**public** **class** RabbitMQConfig {

/\*

\* Ler o nome da fila que será acessada mapeado no arquivo

\* application.properties

\*/

@Value("${queue.name}")

**private** String queueName;

/\*

\* Método para conexão na fila

\*/

@Bean

**public** Queue queue() {

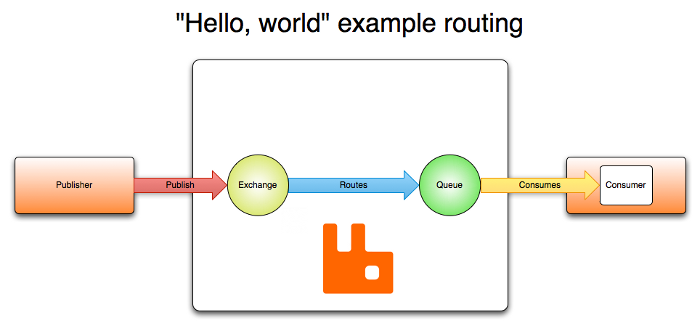
**return** **new** Queue(queueName, **true**);

}

}

**PRODUCER / PUBLISHER**

Consiste em um programa responsável por gravar um conteúdo na fila da mensageria. Ou seja, este programa cria novos itens na fila.



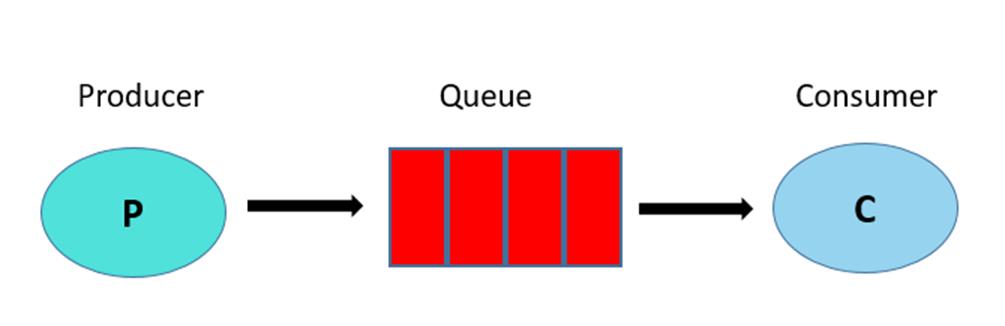
/infra/producers/**MessageProducer.java**

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente



**package** br.com.cotiinformatica.infra.producers;

**import** org.springframework.amqp.core.Queue;

**import** org.springframework.amqp.rabbit.core.RabbitTemplate;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

@Service

**public** **class** MessageProducer {

@Autowired

**private** RabbitTemplate rabbitTemplate;

@Autowired

**private** Queue queue;

/\*

\* Método acessado para gravar um item na FILA

\* PRODUCER / PUBLISHER -> Adicionar itens na fila

\*/

**public** **void** send(String message) {

rabbitTemplate.convertAndSend

(**this**.queue.getName(), message);

}

}

**Criando uma classe DTO para modelar os dados necessários para fazer um envio de email: (DATA TRANSFER OBJECT)**

**package** br.com.cotiinformatica.application.dtos;

**import** lombok.AllArgsConstructor;

**import** lombok.Getter;

**import** lombok.NoArgsConstructor;

**import** lombok.Setter;

**import** lombok.ToString;

@Setter

@Getter

@NoArgsConstructor

@AllArgsConstructor

@ToString

**public** **class** EmailMessageDTO {

**private** String emailTo;

**private** String subject;

**private** String body;

}

Voltando na camada de serviço de domínio:

Escrevendo mensagens na **FILA** para 2 ações do projeto.

* Usuário cadastrado com sucesso
  + EmailMessageDTO contendo uma mensagem de boas-vindas para o usuário.
* Recuperação de senha do usuário
  + EmailMessageDTO contendo uma mensagem de recuperação de senha.

/UsuarioDomainService.java

package br.com.cotiinformatica.domain.services;

import java.util.Date;

import org.modelmapper.ModelMapper;

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

import org.springframework.stereotype.Service;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.github.javafaker.Faker;

import br.com.cotiinformatica.application.dtos.EmailMessageDTO;

import br.com.cotiinformatica.application.dtos.GetUsuarioDTO;

import br.com.cotiinformatica.application.dtos.PostAutenticarDTO;

import br.com.cotiinformatica.application.dtos.PostCriarContaDTO;

import br.com.cotiinformatica.application.dtos.PostRecuperarSenhaDTO;

import br.com.cotiinformatica.application.dtos.ResponseAutenticarDTO;

import br.com.cotiinformatica.application.dtos.ResponseCriarContaDTO;

import br.com.cotiinformatica.application.dtos.ResponseRecuperarSenhaDTO;

import br.com.cotiinformatica.domain.interfaces.IUsuarioDomainService;

import br.com.cotiinformatica.domain.models.Usuario;

import br.com.cotiinformatica.infra.components.MD5Component;

import br.com.cotiinformatica.infra.components.TokenComponent;

import br.com.cotiinformatica.infra.producers.MessageProducer;

import br.com.cotiinformatica.infra.repositories.IUsuarioRepository;

@Service

public class UsuarioDomainService implements IUsuarioDomainService {

@Autowired // injeção de dependência

private IUsuarioRepository usuarioRepository;

@Autowired // injeção de dependência

private MD5Component md5Component;

@Autowired // injeção de dependência

private TokenComponent tokenComponent;

@Autowired // injeção de dependência

private MessageProducer messageProducer;

@Autowired // injeção de dependência

private ObjectMapper objectMapper;

@Override

public ResponseAutenticarDTO autenticar(PostAutenticarDTO dto) {

// procurar o usuário no banco de dados através do email e da senha

Usuario usuario = usuarioRepository.findByEmailAndSenha

(dto.getEmail(), md5Component.encrypt(dto.getSenha()));

// verificar se o usuário não foi encontrado

if (usuario == null)

throw new IllegalArgumentException

("Acesso negado. Usuário inválido.");

// transferir os dados do usuário para o DTO

ModelMapper modelMapper = new ModelMapper();

GetUsuarioDTO usuarioDTO = modelMapper.map

(usuario, GetUsuarioDTO.class);

// gerando o token JWT para o usuário

String token = null;

try {

token = tokenComponent.generateToken(usuario.getEmail());

} catch (Exception e) {

e.printStackTrace();

}

ResponseAutenticarDTO response = new ResponseAutenticarDTO();

response.setStatus(200);

response.setMensagem("Usuário autenticado com sucesso.");

response.setToken(token);

response.setUsuario(usuarioDTO);

return response;

}

@Override

public ResponseCriarContaDTO criarConta(PostCriarContaDTO dto) {

// verificar se já existe um usuário cadastrado com o email informado

if (usuarioRepository.findByEmail(dto.getEmail()) != null)

throw new IllegalArgumentException

("O email informado já está cadastrado. Tente outro.");

// transferir os dados do DTO para a classe de modelo de entidade

ModelMapper modelMapper = new ModelMapper();

Usuario usuario = modelMapper.map(dto, Usuario.class);

usuario.setSenha(md5Component.encrypt(usuario.getSenha()));

usuario.setDataHoraCriacao(new Date());

// gravando no banco de dados

usuarioRepository.save(usuario);

**//enviando mensagem de boas vindas**

**EmailMessageDTO emailMessageDTO = new EmailMessageDTO();**

**emailMessageDTO.setEmailTo(usuario.getEmail());**

**emailMessageDTO.setSubject("Seja bem vindo a API**

**de Usuários - COTI Informática");**

**emailMessageDTO.setBody(**

**"<div>" +**

**"<p>Parabéns, " + usuario.getNome()**

**+ ". Sua conta de**

**usuário foi criada com sucesso.</p>" +**

**"<p>Utilize o email e senha cadastrados para**

**acessar sua conta." +**

**"<p>Att,<br/>Equipe COTI Informática</p>"**

**);**

**//enviando a mensagem para a fila..**

**try {**

**messageProducer.send(objectMapper.writeValueAsString**

**(emailMessageDTO));**

**}**

**catch(Exception e) {**

**e.printStackTrace();**

**}**

ResponseCriarContaDTO response = new ResponseCriarContaDTO();

response.setStatus(201);

response.setMensagem("Usuário cadastrado com sucesso");

response.setDataHoraCadastro(new Date());

response.setUsuario(modelMapper.map(usuario, GetUsuarioDTO.class));

return response;

}

@Override

public ResponseRecuperarSenhaDTO recuperarSenha(PostRecuperarSenhaDTO dto) {

// procurar o usuário no banco de dados através do email

Usuario usuario = usuarioRepository.findByEmail(dto.getEmail());

// verificar se o usuário foi encontrado

if (usuario == null)

throw new IllegalArgumentException

("Email não encontrado. Usuário inválido.");

// gerar uma nova senha para o usuário

String novaSenha = new Faker().internet().password(8, 10, true);

**//enviando mensagem de recuperação de senha**

**EmailMessageDTO emailMessageDTO = new EmailMessageDTO();**

**emailMessageDTO.setEmailTo(usuario.getEmail());**

**emailMessageDTO.setSubject("Recuperação de Senha**

**(API de Usuários) - COTI Informática");**

**emailMessageDTO.setBody(**

**"<div>" +**

**"<p>Parabéns, " + usuario.getNome()**

**+ ". Uma nova senha foi gerada com sucesso.</p>" +**

**"<p>Utilize a senha <strong>" + novaSenha**

**+ "</strong> para acessar sua conta." +**

**"<p>Att,<br/>Equipe COTI Informática</p>"**

**);**

**//enviando a mensagem para a fila..**

**try {**

**messageProducer.send**

**(objectMapper.writeValueAsString(emailMessageDTO));**

**}**

**catch(Exception e) {**

**e.printStackTrace();**

**}**

// atualizando a senha do usuário no banco de dados

usuario.setSenha(md5Component.encrypt(novaSenha));

usuarioRepository.save(usuario);

ResponseRecuperarSenhaDTO response

= new ResponseRecuperarSenhaDTO();

response.setStatus(200);

response.setMensagem("Recuperação de senha realizado com sucesso.");

return response;

}

}

**Executando os testes:**

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**Na fila do servidor do RabbitMQ:**

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

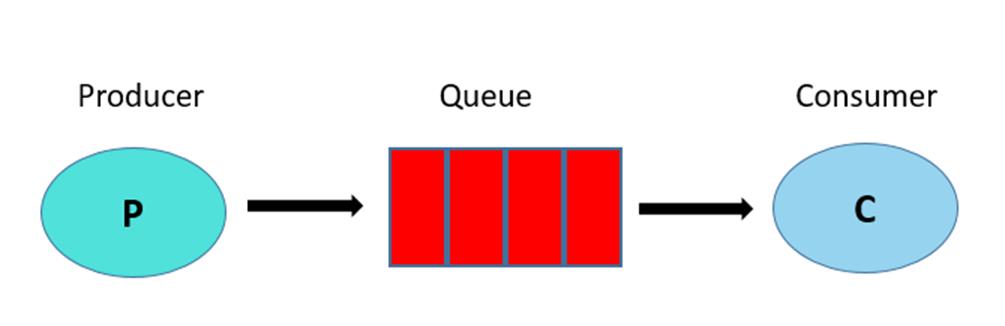
Descrição gerada automaticamente

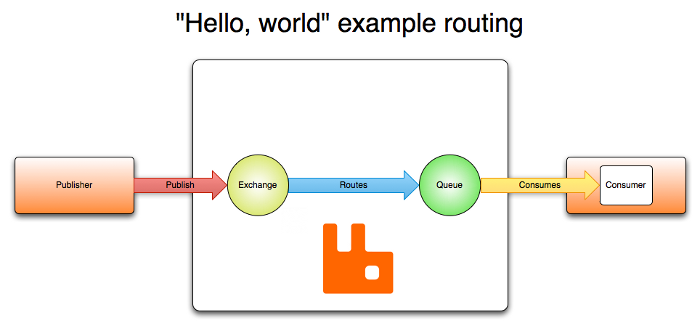
Interface gráfica do usuário, Texto, Email

Descrição gerada automaticamente

**CONSUMER**

Programa que irá ler e processar os itens da FILA. Neste caso será uma classe para varrer e processar o envio de cada mensagem contida na fila.





Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**package** br.com.cotiinformatica.infra.consumers;

**import** org.springframework.amqp.rabbit.annotation.RabbitListener;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.messaging.handler.annotation.Payload;

**import** org.springframework.stereotype.Service;

**import** com.fasterxml.jackson.databind.ObjectMapper;

**import** br.com.cotiinformatica.application.dtos.EmailMessageDTO;

**import** br.com.cotiinformatica.infra.components.EmailComponent;

@Service

**public** **class** MessageConsumer {

@Autowired // Injeção de dependência

EmailComponent emailComponent;

@Autowired // Injeção de dependência

ObjectMapper objectMapper;

/\*

\* Método para ler e processar a fila

\*/

**@RabbitListener(queues = { "${queue.name}" })**

**public void receive(@Payload String payload) throws Exception {**

**// lendo o conteudo da mensagem**

**// gravada na fila do RabbitMQ**

**EmailMessageDTO emailMessageDTO = objectMapper.readValue**

**(payload, EmailMessageDTO.class);**

**// enviar o email**

**emailComponent.sendMessage(emailMessageDTO.getEmailTo(),**

**emailMessageDTO.getSubject(),**

**emailMessageDTO.getBody());**

**}**

}

**Executando o projeto:**

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Tabela

Descrição gerada automaticamente

**Fila do RabbitMQ:**

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**Publicando o projeto no AWS com essa atualização:**

Gerando o DEPLOY do projeto:

C:\java\_dev\api\_usuarios>**mvnw clean package**

Texto

Descrição gerada automaticamente

**Deploy gerado:**

Interface gráfica do usuário, Texto, Aplicativo, Tabela, Excel

Descrição gerada automaticamente

**Enviando para o AWS:**

<https://aws.amazon.com/pt/>

Elastic BeanStalk

Serviço para publicação de aplicações web no AWS.

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

<http://apiusuarios-env.eba-fp62bpyj.us-east-2.elasticbeanstalk.com/swagger-ui/index.html>

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

**Abrindo o projeto Angular:**

Interface gráfica do usuário, Aplicativo, Excel

Descrição gerada automaticamente

**Abrindo o Visual Studio Code:**

Samsung@DESKTOP-P9F6D9F MINGW64 /c/dev\_angular\_sqs/projeto\_angular (master)

**$ code .**

Texto

Descrição gerada automaticamente

**Instalando o tema “Dracula”:**

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

**Executando o projeto:**

Samsung@DESKTOP-P9F6D9F MINGW64 /c/dev\_angular\_sqs/projeto\_angular (master)

**$ ng s**

Texto

Descrição gerada automaticamente

<http://localhost:4200/>

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Criando um arquivo de configuração no projeto para definirmos os endereços dos servidores das APIs:

/src/environments/environment.ts

export const environment = {

    apiUsuarios : "http://apiusuarios-env.eba-fp62bpyj

.us-east-2.elasticbeanstalk.com/",

    apiClientes : "http://apiclientes-env.eba-v3p5tzs5

.us-east-2.elasticbeanstalk.com/"

}

Tela de computador com jogo

Descrição gerada automaticamente

Acessando os endereços no componente:

/app.component.ts

import { Component } from '@angular/core';

import { FormGroup, FormControl } from '@angular/forms';

import { HttpClient } from '@angular/common/http';

import { environment } from 'src/environments/environment';

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  mensagem\_criarconta: *string* = '';

  constructor(

    private *httpClient*: HttpClient //injeção de dependência

  ) {

  }

  //criando um objeto para capturar os

  //campos do formulário de criação de usuário

  formCriarConta = **new** FormGroup({

    nome: **new** FormControl('', []), //campo 'nome'

    email: **new** FormControl('', []), //campo 'email'

    senha: **new** FormControl('', []) //campo 'senha'

  });

  //criando um objeto para capturar os

  //campos do formulário de autenticação

  formAutenticar = **new** FormGroup({

    email: **new** FormControl('', []), //campo 'email'

    senha: **new** FormControl('', []) //campo 'senha'

  });

  //criando um objeto para capturar os

  //campos do formulário de recuperação de senha

  formRecuperarSenha = **new** FormGroup({

    email: **new** FormControl('', []), //campo 'email'

  });

  //função para capturar o SUBMIT do formulário

  criarConta(): *void* {

    //capturando os valores preenchidos no formulário

    var dados = *this*.formCriarConta.value;

    //executando a chamada para a API

*this*.httpClient.post(environment.apiUsuarios

+ 'api/criar-conta', dados)

      .subscribe({ //capturando a resposta da API

        next: (*data*: *any*) => { //sucesso!

*this*.mensagem\_criarconta = *data*.mensagem;

*this*.formCriarConta.reset(); //limpar o formulário

        },

        error: (*e*) => { //erro!

          if(*e*.error.mensagem){

*this*.mensagem\_criarconta = *e*.error.mensagem;

          }

          else if(*e*.error.errors){

*this*.mensagem\_criarconta = *e*.error.errors;

          }

        }

      });

  }

  //função para capturar o SUBMIT do formulário

  autenticar(): *void* {

    //capturando os valores preenchidos no formulário

    var dados = *this*.formAutenticar.value;

    //executando a chamada para a API

*this*.httpClient.post(environment.apiUsuarios

+ 'api/autenticar', dados)

      .subscribe({ //capturando a resposta da API

        next: (*data*) => { //sucesso!

          alert(JSON.stringify(*data*));

        },

        error: (*e*) => { //erro!

          alert(JSON.stringify(*e*.error));

        }

      });

  }

  //função para capturar o SUBMIT do formulário

  recuperarSenha(): *void* {

    //capturando os valores preenchidos no formulário

    var dados = *this*.formRecuperarSenha.value;

    //executando a chamada para a API

*this*.httpClient.post(environment.apiUsuarios

+ 'api/recuperar-senha', dados)

      .subscribe({ //capturando a resposta da API

        next: (*data*) => { //sucesso!

          alert(JSON.stringify(*data*));

        },

        error: (*e*) => { //erro!

          alert(JSON.stringify(*e*.error));

        }

      });

  }

}

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

**Criando um componente para desenhar a página de gerenciamento de clientes:**

Samsung@DESKTOP-P9F6D9F MINGW64 /c/dev\_angular\_sqs/projeto\_angular (master)

**$ ng generate component clientes**

Texto

Descrição gerada automaticamente

Tela de computador com jogo

Descrição gerada automaticamente

Continua...