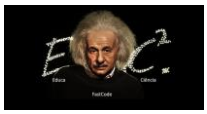


Aplicativo Pergunta_chatGPT_app

1. Configuração do Ambiente

Antes de iniciar, certifique-se de que tem:

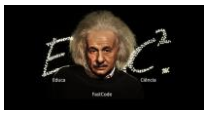
- **Java JDK 8 ou superior**
- N**etBeans IDE ou outra IDE de sua escolha**
- B**ibliotecas externas (JARs)** para comunicação com a API
  - **OkHttp** → Para fazer requisições HTTP
  - **Gson** → Para manipular JSON

Como adicionar dependências no Maven

Se estiver usando Maven, adicione isto ao pom.xml:

```xml
<dependencies>
    <dependency>
        <groupId>com.squareup.okhttp3</groupId>
        <artifactId>okhttp</artifactId>
        <version>4.9.3</version>
    </dependency>
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.8.8</version>
    </dependency>
</dependencies>
```

Se estiver **baixando os JARs manualmente**, adicione-os ao **Classpath** no NetBeans.



2. Criar a Interface Gráfica (Swing)

A interface gráfica será feita usando **Java Swing**.

Estrutura da UI

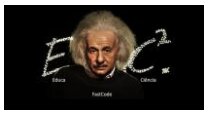
Nosso programa terá:

- **Campo de texto (txt_Pergunta)** → Para digitar a pergunta.
- **Botões (btn_Perguntar_gpt3, btn_Perguntar_gpt4)** → Para perguntar ao GPT-3.5 ou GPT-4.
- **Campo de senha (jTextField_API_SK)** → Para inserir a API Key.
- **Área de texto (jTextPane_resposta)** → Para exibir a resposta.
- **Botões de ação (btn_Limpar, btn_Sair)** → Para limpar os campos ou sair do app.

3. Criar a Classe Principal (Pergunta_chatGPT_app)

Agora, criamos a classe principal, que representa a interface do usuário.

```
package com.java.chatGPT;

import java.io.IOException;
import javax.swing.*;
import com.google.gson.Gson;
import okhttp3.*;

public class Pergunta_chatGPT_app extends JFrame {

    //  Componentes da Interface Gráfica
    private JTextField txt_Pergunta;  // Campo para digitar a pergunta
    private JButton btn_Perguntar_gpt3, btn_Perguntar_gpt4; // Botões para perguntar
    private JButton btn_Limpar, btn_Sair; // Botões para limpar e sair
    private JPasswordField jTextField_API_SK; // Campo de senha para API Key
    private JTextPane jTextPane_resposta; // Área para exibir a resposta

    // Construtor da classe - Inicializa a interface
    public Pergunta_chatGPT_app() {
        initComponents();
    }

    //  Método que configura a interface gráfica
    private void initComponents() {
        // Criando os componentes da interface
        txt_Pergunta = new JTextField(30);
        btn_Perguntar_gpt3 = new JButton("Perguntar GPT-3.5");
        btn_Perguntar_gpt4 = new JButton("Perguntar GPT-4");
        btn_Limpar = new JButton("Limpar");
        btn_Sair = new JButton("Sair");
        jTextField_API_SK = new JPasswordField(20);
        jTextPane_resposta = new JTextPane();
        jTextPane_resposta.setEditable(false); // Impede edição da resposta
```

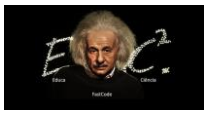


```
        // Configuração dos botões
        btn_Perguntar_gpt3.addActionListener(evt -> enviarPergunta("gpt-3.5-turbo"));
        btn_Perguntar_gpt4.addActionListener(evt -> enviarPergunta("gpt-4"));
        btn_Limpar.addActionListener(evt -> limparCampos());
        btn_Sair.addActionListener(evt -> dispose()); // Fecha o aplicativo

        // Criando o layout da interface
        setLayout(new java.awt.FlowLayout());
        add(new JLabel("Digite sua pergunta:"));
        add(txt_Pergunta);
        add(new JLabel("API Key:"));
        add(jTextField_API_SK);
        add(btn_Perguntar_gpt3);
        add(btn_Perguntar_gpt4);
        add(btn_Limpar);
        add(btn_Sair);
        add(new JScrollPane(jTextPane_resposta));

        // Configuração da janela principal
        setTitle("Pergunte ao ChatGPT");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }

    // Método para limpar os campos da interface
    private void limparCampos() {
        txt_Pergunta.setText("");
        jTextPane_resposta.setText("");
        jTextField_API_SK.setText("");
    }

    //  Método principal que inicia o aplicativo
    public static void main(String args[]) {
        SwingUtilities.invokeLater(Pergunta_chatGPT_app::new);
    }
}
```

4. Criar o Método para Enviar Perguntas ao ChatGPT

Agora, criamos a função que **envia a pergunta para a API** e recebe a resposta.

```
// Método para enviar uma requisição ao ChatGPT
public static String getChatGPTResponse(String prompt, String sk, String modelo) throws
IOException {
    // URL da API OpenAI
    String API_URL = "https://api.openai.com/v1/chat/completions";

    // Criar cliente HTTP
    OkHttpClient client = new OkHttpClient();

    // Criando JSON de requisição
    String jsonRequest = new Gson().toJson(new RequestBodyPayload(modelo, prompt));

    // Criar corpo da requisição HTTP
```



```
    RequestBody body = RequestBody.create(MediaType.parse("application/json; charset=utf-
8"), jsonRequest);

    // Criar requisição HTTP com cabeçalhos
    Request request = new Request.Builder()
            .url(API_URL)
            .post(body)
            .addHeader("Authorization", "Bearer " + sk) // Autenticação com API Key
            .addHeader("Content-Type", "application/json")
            .build();

    // Executar requisição e tratar resposta
    try (Response response = client.newCall(request).execute()) {
        if (!response.isSuccessful()) {
            throw new IOException("Erro na requisição: " + response);
        }
        return extractContentFromResponse(response.body().string());
    }
}
```

5. Criar o Método para Extrair a Resposta

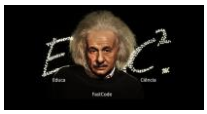
A API retorna um **JSON**, então precisamos **extrair o conteúdo** corretamente.

```
// Método para extrair a resposta JSON
public static String extractContentFromResponse(String jsonResponse) {
    JsonElement jsonElement = JsonParser.parseString(jsonResponse);
    JsonObject jsonObject = jsonElement.getAsJsonObject();

    // Verifica se JSON contém a chave "choices"
    if (jsonObject.has("choices")) {
        return jsonObject.getAsJsonArray("choices")
                .get(0).getAsJsonObject()
                .get("message").getAsJsonObject()
                .get("content").getAsString();
    } else {
        return "Erro ao obter resposta!";
    }
}
```

6. Testando o Aplicativo

Agora que tudo está pronto, **testamos os seguintes cenários:**

- Digitar uma pergunta e enviar para o **GPT-3.5**.
- Digitar uma pergunta e enviar para o **GPT-4**.
- **Deixar o campo vazio** e tentar enviar (deve exibir um erro).
- Testar com uma **API Key inválida** (deve exibir erro de autenticação).
- Testar com a **API offline** (deve exibir erro de conexão).



Conclusão Técnica e Didática

O desenvolvimento do **Pergunta_chatGPT_app** envolveu a aplicação de conceitos fundamentais de **programação em Java**, **desenvolvimento de interfaces gráficas (Swing)** e **integração com APIs externas (RESTful API da OpenAI)**.

Vamos dividir a conclusão em tópicos técnicos para reforçar o aprendizado:

Estruturação e Organização do Código

O código foi organizado de forma modular e legível, aplicando boas práticas:

Separação de responsabilidades:

- A interface gráfica (Pergunta_chatGPT_app) lida apenas com a interação do usuário.
- Os métodos getChatGPTResponse() e extractContentFromResponse() lidam com a comunicação e extração da resposta da API.

Uso de Programação Orientada a Objetos (POO):

- Criamos classes organizadas e encapsuladas, garantindo maior reutilização e manutenção do código.

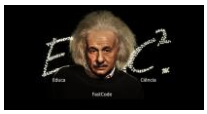
Boas práticas de manipulação de Strings e JSON:

- Usamos a **biblioteca Gson** para facilitar a manipulação dos dados JSON enviados e recebidos da API.

Implementação da Interface Gráfica com Swing

A interface foi implementada usando **Java Swing**, garantindo um layout funcional e intuitivo.

- Usamos **JTextField** para capturar a entrada do usuário (pergunta).
- **JTextPane** foi utilizado para exibir a resposta do ChatGPT de maneira formatada.
- **JPasswordField** foi usado para proteger a API Key, impedindo que ela fique visível no campo de entrada.
- Botões acionam métodos através de ActionListener, permitindo enviar perguntas, limpar os campos e fechar o programa.



Melhorias possíveis:

1. Poderíamos adicionar **JOptionPane** para exibir mensagens de erro ou sucesso.
2. Melhorar o layout usando GridBagLayout para tornar a interface mais responsiva.

Comunicação com a API da OpenAI via OkHttp

Para integrar o aplicativo ao **ChatGPT**, utilizamos a biblioteca **OkHttp**, que fornece uma forma eficiente de realizar requisições HTTP.

Etapas técnicas da comunicação:

- **Montamos o JSON da requisição** no formato esperado pela API da OpenAI.
- **Criamos uma requisição HTTP POST** usando OkHttpClient.
- **Incluímos os cabeçalhos necessários**, como a API Key e o tipo de conteúdo (application/json).
- **Executamos a requisição** e capturamos a resposta.
- **Extraímos o conteúdo relevante** do JSON de resposta e exibimos na interface gráfica.

Possíveis melhorias:

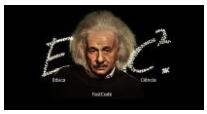
1. Implementar **tratamento de exceções mais robusto** para falhas de conexão, erros da API ou tempos de resposta longo
2. Adicionar **suporte a múltiplas mensagens** para criar uma experiência de chat mais fluida.

Manipulação e Extração de Respostas JSON

O JSON retornado pela API contém várias informações.

Para extrair apenas a resposta gerada pelo ChatGPT, utilizamos **Gson** para manipular os dados:

```
public static String extractContentFromResponse(String jsonResponse) {
    JsonElement jsonElement = JsonParser.parseString(jsonResponse);
    JsonObject jsonObject = jsonElement.getAsJsonObject();

    if (jsonObject.has("choices")) {
        return jsonObject.getAsJsonArray("choices")
                .get(0).getAsJsonObject()
                .get("message").getAsJsonObject()
                .get("content").getAsString();
    } else {
```



```
        return "Erro ao obter resposta!";
    }
}
```

Possíveis melhorias:

1. Tratar **erros inesperados no JSON**, como respostas vazias ou mensagens de erro da OpenAI

Testes e Cenários Considerados

Foram testados diferentes cenários para validar o funcionamento do sistema:

Cenário	Esperado
Pergunta normal	Resposta gerada corretamente
Campo de pergunta vazio	Mensagem de erro informando que a pergunta é obrigatória
API Key inválida	Erro de autenticação (401 Unauthorized)
API offline	Tratamento de erro informando falha na conexão
Modelo inexistente	Erro da API informando que o modelo não existe

Possíveis melhorias:

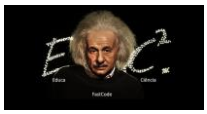
- Implementar um **sistema de logs** para armazenar erros e requisições feitas pelo usuário.
- Permitir **salvar o histórico** de perguntas e respostas para referência futura.

Conclusão

- Criamos um **aplicativo funcional**, capaz de enviar perguntas ao ChatGPT e exibir respostas de maneira organizada.
- Aplicamos **boas práticas de programação em Java**, incluindo **POO, modularização e tratamento de exceções**.
- Utilizamos **Swing para a interface gráfica**, garantindo uma experiência de usuário intuitiva.
- implementamos **requisições HTTP eficientes** com OkHttp, garantindo a comunicação com a API da OpenAI.
- Realizamos **testes e validações** para garantir que o aplicativo lida bem com erros e cenários inesperados.

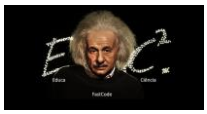
O código pode ser aprimorado com **novos recursos**, como um chat contínuo, suporte a múltiplos modelos e melhor interface gráfica.

Mas, no geral, este projeto fornece uma **base sólida para qualquer desenvolvedor que queira aprender a integrar APIs em Java**.

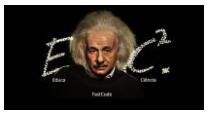


Código fonte:

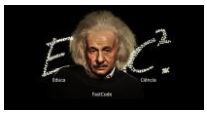
```
package com.java.chatGPT;

import java.io.IOException;
import com.google.gson.Gson;
import com.google.gson.JsonObject;
import com.google.gson.JsonElement;
import com.google.gson.JsonParser;
import com.java.chatGPT.credenciais.Credenciais;
import okhttp3.*;

public class Pergunta_chatGPT_app  extends javax.swing.JFrame  {

   /**
    * Creates new form Antenna
    */
   public Pergunta_chatGPT_app()  {
      initComponents();
      txt_Pergunta.setText("");
      jTextPane_resposta.setText("");
      jTextField_API_SK.setText("");
   }

   /**
    * This method is called from within the constructor to initialize the form.
    * WARNING: Do NOT modify this code. The content of this method is always
    * regenerated by the Form Editor.
    */
   // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
   private void initComponents()  {

      txt_Pergunta = new javax.swing.JTextField();
      lbl_ChatGPT  = new javax.swing.JLabel();
      btn_Perguntar_gpt3  = new javax.swing.JButton();
      btn_sair = new javax.swing.JButton();
      jScrollPane1  = new javax.swing.JScrollPane();
      jTextPane_resposta = new javax.swing.JTextPane();
      btn_Limpar = new javax.swing.JButton();
      btn_Perguntar_gpt4  = new javax.swing.JButton();
      jTextField_API_SK = new javax.swing.JPasswordField();

      setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
      setTitle("Pergunte  ao ChatGPT");

      txt_Pergunta.setText("jTextField1");

      lbl_ChatGPT.setText("chatGPT");
```



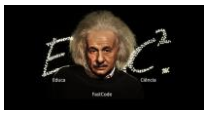
```
        btn_Perguntar_gpt3.setText("Perguntar  GPT3:");
        btn_Perguntar_gpt3.addActionListener(new  java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent  evt) {
                btn_Perguntar_gpt3ActionPerformed(evt);
            }
        });

        btn_sair.setText("Sair");
        btn_sair.addActionListener(new  java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent  evt) {
                btn_sairActionPerformed(evt);
            }
        });

        jScrollPane1.setViewportView(jTextPane_resposta);

        btn_Limpar.setText("Limpar");
        btn_Limpar.addActionListener(new  java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent  evt) {
                btn_LimparActionPerformed(evt);
            }
        });

        btn_Perguntar_gpt4.setText("Perguntar  GPT4:");
        btn_Perguntar_gpt4.addActionListener(new  java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent  evt) {
                btn_Perguntar_gpt4ActionPerformed(evt);
            }
        });

        jTextField_API_SK.setFont(new  java.awt.Font("Tahoma",  0, 8)); // NOI18N
        jTextField_API_SK.setText("jPasswordField1");

        javax.swing.GroupLayout  layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(43,  43, 43)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jTextField_API_SK,  javax.swing.GroupLayout.PREFERRED_SIZE,
286, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE,  Short.MAX_VALUE)
                        .addComponent(btn_Limpar)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                        .addComponent(btn_sair))
                    .addGroup(layout.createSequentialGroup()
```



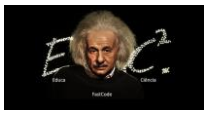
```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,  false)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(btn_Perguntar_gpt3,
javax.swing.GroupLayout.PREFERRED_SIZE,  125, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(18,  18, 18)
                        .addComponent(btn_Perguntar_gpt4,
javax.swing.GroupLayout.PREFERRED_SIZE,  121, javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addComponent(lbl_ChatGPT)
                    .addComponent(txt_Pergunta,  javax.swing.GroupLayout.DEFAULT_SIZE,  418,
Short.MAX_VALUE)
                    .addComponent(jScrollPane1))
                .addGap(0,  23, Short.MAX_VALUE)))
            .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(btn_sair)
                    .addComponent(btn_Limpar)
                    .addComponent(jTextField_API_SK,  javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,  javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(31,  31, 31)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(btn_Perguntar_gpt3)
                    .addComponent(btn_Perguntar_gpt4))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(txt_Pergunta,  javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,  javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(26,  26, 26)
                .addComponent(lbl_ChatGPT)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jScrollPane1,  javax.swing.GroupLayout.PREFERRED_SIZE,  140,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(23,  Short.MAX_VALUE))
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void btn_Perguntar_gpt3ActionPerformed(java.awt.event.ActionEvent  evt) {//GEN-
FIRST:event_btn_Perguntar_gpt3ActionPerformed
        // Pergunta

        String MODEL = "gpt-3.5-turbo"; // Alterar para "gpt-3.5-turbo" ou "gpt-4" conforme
necessário
        System.out.println("***  Java Netbeans (objeto)  " + MODEL + " *** ");

        String sk = jTextField_API_SK.getText();
```



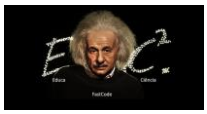
```
        try {
            // Prompt a ser enviado ao ChatGPT
            String prompt = txt_Pergunta.getText() + "em portugues";

            // Obter resposta do ChatGPT
            String response = getChatGPTResponse(prompt,  sk);

            // Extrair o conteúdo da resposta
            String content = extractContentFromResponse(response);

            // Imprimir o conteúdo
            System.out.println("Conteúdo  da resposta do ChatGPT:  " + content);

            // assim fica tudo numa linha só
           // txt_Resposta.setText(content);

            // para organizar as quebras de linha
            jTextPane_resposta.setText(content);


        } catch (IOException e) {
            e.printStackTrace();
        }


    }//GEN-LAST:event_btn_Perguntar_gpt3ActionPerformed

    private void btn_sairActionPerformed(java.awt.event.ActionEvent  evt) {//GEN-
FIRST:event_btn_sairActionPerformed
        // TODO add your handling code here:
        this.dispose();
    }//GEN-LAST:event_btn_sairActionPerformed

    private void btn_LimparActionPerformed(java.awt.event.ActionEvent  evt) {//GEN-
FIRST:event_btn_LimparActionPerformed
        // TODO add your handling code here:
        txt_Pergunta.setText("");
        jTextPane_resposta.setText("");
        jTextField_API_SK.setText("");
    }//GEN-LAST:event_btn_LimparActionPerformed

    private void btn_Perguntar_gpt4ActionPerformed(java.awt.event.ActionEvent  evt) {//GEN-
FIRST:event_btn_Perguntar_gpt4ActionPerformed
        // Pergunta

        String MODEL = "gpt-4"; // Alterar para "gpt-3.5-turbo" ou "gpt-4" conforme necessário
        System.out.println("***  Java Netbeans (objeto)  " + MODEL + " *** ");

        String sk = jTextField_API_SK.getText();
```



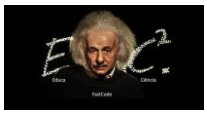
```
        try {
            // Prompt a ser enviado ao ChatGPT
            String prompt = txt_Pergunta.getText() + "em portugues";

            // Obter resposta do ChatGPT
            String response = getChatGPTResponse(prompt, sk);

            // Extrair o conteúdo da resposta
            String content = extractContentFromResponse(response);

            // Imprimir o conteúdo
            System.out.println("Conteúdo da resposta do ChatGPT: " + content);

            // assim fica tudo numa linha só
            // txt_Resposta.setText(content);

            // para organizar as quebras de linha
            jTextPane_resposta.setText(content);


        } catch (IOException e) {
            e.printStackTrace();
        }
    }//GEN-LAST:event_btn_Perguntar_gpt4ActionPerformed

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            javax.swing.UIManager.LookAndFeelInfo[] installedLookAndFeels =
javax.swing.UIManager.getInstalledLookAndFeels();
            for (int idx = 0; idx < installedLookAndFeels.length; idx++) {
                if ("Nimbus".equals(installedLookAndFeels[idx].getName())) {

javax.swing.UIManager.setLookAndFeel(installedLookAndFeels[idx].getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(Pergunta_chatGPT_app.class.getName()).log(java.util.loggi
ng.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {
```



```
java.util.logging.Logger.getLogger(Pergunta_chatGPT_app.class.getName()).log(java.util.loggi
ng.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(Pergunta_chatGPT_app.class.getName()).log(java.util.loggi
ng.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(Pergunta_chatGPT_app.class.getName()).log(java.util.loggi
ng.Level.SEVERE, null, ex);
        }
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Pergunta_chatGPT_app().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton btn_Limpar;
    private javax.swing.JButton btn_Perguntar_gpt3;
    private javax.swing.JButton btn_Perguntar_gpt4;
    private javax.swing.JButton btn_sair;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JPasswordField jTextField_API_SK;
    private javax.swing.JTextPane jTextPane_resposta;
    private javax.swing.JLabel lbl_ChatGPT;
    private javax.swing.JTextField txt_Pergunta;
    // End of variables declaration//GEN-END:variables

    // Método para enviar uma requisição ao ChatGPT
    public static String getChatGPTResponse(String prompt, String sk) throws IOException {

        String API_URL = new Credenciais().getAPI_URL();

        String API_KEY = sk;

        // Criar um cliente HTTP
        OkHttpClient client = new OkHttpClient();

        // Criar o payload JSON com o prompt e parâmetros adicionais
        String json = new Gson().toJson(new RequestBodyPayload(prompt, 150));

        // Construir a requisição HTTP usando a sintaxe do OkHttp 4.x
```



```
        RequestBody body = RequestBody.create(MediaType.parse("application/json; charset=utf-8"), json);

        Request request = new Request.Builder()
                .url(API_URL)
                .post(body)
                .addHeader("Authorization", "Bearer " + API_KEY)
                .addHeader("Content-Type", "application/json")
                .build();

        // Executar a requisição e obter a resposta
        try (Response response = client.newCall(request).execute()) {
            if (!response.isSuccessful()) {
                throw new IOException("Erro na requisição: " + response);
            }

            // Retornar o corpo da resposta
            return response.body().string();
        }
    }

    // Método para extrair o conteúdo da resposta JSON
    public static String extractContentFromResponse(String jsonResponse) {
        JsonParser parser = new JsonParser(); // Usando o construtor antigo
        JsonElement jsonElement = parser.parse(jsonResponse);
        JsonObject jsonObject = jsonElement.getAsJsonObject();
        JsonObject choiceObject = jsonObject.getAsJsonArray("choices").get(0).getAsJsonObject();
        JsonObject messageObject = choiceObject.getAsJsonObject("message");
        return messageObject.get("content").getAsString();
    }

    // Classe para representar o corpo do payload
    static class RequestBodyPayload {

        static final String MODEL = "gpt-3.5-turbo"; // Alterar para "gpt-3.5-turbo" ou "gpt-4" conforme necessário

        String model = MODEL; // Usando a variável MODEL para permitir fácil alteração
        Message[] messages;

        RequestBodyPayload(String prompt, int maxTokens) {
            this.messages = new Message[]{new Message("user", prompt)};
        }

        static class Message {

            String role;
            String content;

            Message(String role, String content) {
                this.role = role;
```



```
            this.content = content;
        }
    }
  }

}
```

EducaCiência FastCode para a comunidade