NEWTON PAIVA

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

FLAVIO AUGUSTO CARVALHO DE OLIVEIRA GOMES DE ABREU
RAFAELA JUNQUEIRA LAVORATO PIMENTA

MINI SIMULADOR DE REDE SOCIAL

BELO HORIZONTE – MG

2023

12119104 FLAVIO AUGUSTO CARVALHO DE OLIVEIRA GOMES DE ABREU
12121158 RAFAELA JUNQUEIRA LAVORATO PIMENTA

MINI SIMULADOR DE REDE SOCIAL

Trabalho final da disciplina de Programação Orientada a Objetos e Linguagem de Programação, apresentado a Universidade Newton Paiva.

Professor(a): Michelle Hanne e João Paulo Aramuni

BELO HORIZONTE – MG

2023

Sumário

1. INTRODUÇÃO:

O problema abordado neste projeto é a implementação de um Mini Simulador de Rede Social. O objetivo é criar um sistema capaz de cadastrar usuários, gerenciar amizades entre eles e permitir o envio de mensagens. O sistema deve ser capaz de lidar com um número indefinido de usuários.

O ambiente de desenvolvimento utilizado foi o IntelliJ IDEA e a linguagem de programação do projeto foi JAVA. As tecnologias utilizadas para esse trabalho foram: o Java Swing para a construção da interface gráfica, o PostgreSQL foi o banco de dados utilizado para armazenar os dados e o JDBC (Java Database Connectivity) para interação com o banco de dados.

1.2. Estrutura de dados:

A estrutura de dados utilizada nesse projeto é baseada em classes, com foco na orientação a objetos. A classe principal é o “Usuario”, que representa um usuário da rede social. Cada usuário possui um nome, e-mail, senha e uma lista de amigos. A lista de amigos é uma coleção de objetos do tipo “Usuario”, permitindo a associação de um número indefinido de amigos a cada usuário.

Também temos a classe “adicionarAmigo”, essa classe possui classes internas estáticas que são: cadastrar, enviarMensagem, excluirAmigo, consultarAmigos, listarMensagens, logarUsuario. Cada classe interna executa operações específicas relacionadas a amigos, mensagens e autenticação de usuários no banco de dados.

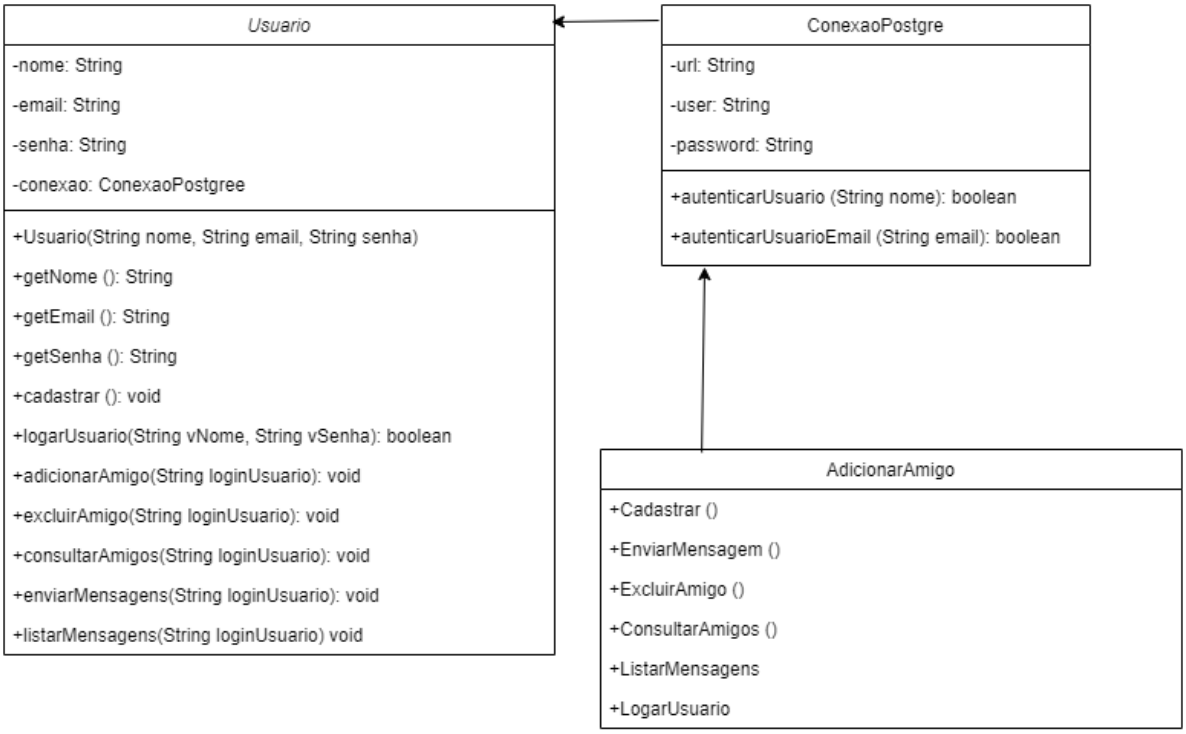
E por fim, temos a classe “conexaoPostgre” que trata da conexão com o banco de dados PostgreSQL e contém métodos com queries SQL para interagir com os dados de amigos, mensagens e usuários.

A estrutura de banco de dados é composta por tabelas:

- “usuários”: armazena informações de nome, e-mail e senha dos usuários;
- “amigos”: relaciona os usuários que são amigos;
- “enviarmensagem”: registra as mensagens enviadas entre usuários.

1.3. UML – Diagrama de classes:

A seguir, o diagrama de classes do Mini Simulador de Rede social, utilizamos o site https://app.diagrams.net/ para criação do diagrama.



2. DESENVOLVIMENTO:

No desenvolvimento do Mini simulador de Rede Social, a utilização dos conceitos importantes na orientação a objetos é fundamental para criar uma estrutura flexível, modular e de fácil manutenção. Segue a explicação desses conceitos:

2.1 Classe abstrata:

É utilizada para definir uma estrutura comum e compartilhar comportamentos entre classes relacionadas. No caso do simulador de rede social, optamos por não utilizar a classe abstrata por limitar a flexibilidade na criação de novas implementações, o que dificultaria na evolução do projeto, já que temos interesse em aprimorar o projeto futuramente.

2.2. **Herança**:

É utilizada para estabelecer uma relação de “é um” entre classes, onde uma classe mais específica herda características e comportamentos de uma classe mais genérica. No simulador de rede social, a classe “AdicionarAmigo” herda funcionalidades da classe “ConexaoPostgre”, aproveitando seus métodos para realizar operações específicas, como adicionar amigos, enviar mensagens, etc. A herança permite reutilizar código e estabelecer hierarquias de classes, facilitando a organização e a manutenção do código.

2.3. **Interface:**

A interface é utilizada para definir um conjunto de métodos que uma classe deve implementar. No simulador de rede social, optamos por não utilizar pois nosso código já estava muito extenso e a criação de interfaces adicionais poderia introduzir um overhead de implementação desnecessário, tornando o código mais extenso do que o necessário.

2.4. **Polimorfismo:**

É a capacidade de um objeto ser tratado de diferentes maneiras, dependendo do contexto em que é utilizado. No contexto do mini simulador de rede social, o polimorfismo está presente no código por meio da sobrescrita de métodos e do uso de métodos comuns com diferentes implementações em classes derivadas. Embora não haja interfaces formais declaradas no código fornecido, algumas classes, como “AdicionarAmigo”, “EnviarMensagem”, “ExcluirAmigo”, “ConsultarAmigos” e “ListarMensagens”, fornecem diferentes funcionalidades que podem ser tratadas de maneira semelhante por meio de métodos comuns. Essas mesmas classes e outras apresentadas podem ser tratados de maneira polimórfica, já que cada método oferece funcionalidades específicas, mas pode ser chamado de maneira uniforme através da hierarquia de classes.

O polimorfismo simplifica a lógica do programa, tornando-o mais flexível e facilitando futuras modificações. Em síntese, o uso desses conceitos no desenvolvimento do projeto foi fundamental para criar uma estrutura modular, flexível e de fácil manutenção.

2.5. Padrões de projetos utilizados

O código desse projeto não há explicitamente a aplicação de padrões de design conhecidos. No entanto, é possível identificar alguns conceitos que se relacionam com padrões de design, há um conceito de métodos que criam instâncias de objetos (“inserirUsuario”, “adicionarAmigoBanco”, etc.), o que se assemelha à ideia do padrão Factory Method, no entanto, não é aplicado formalmente como um padrão. Em algumas das classes internas estáticas, como “EnviarMensagem”, “ExcluirAmigo”, etc., pode-se observar um padrão de template, onde essas classes seguem uma estrutura semelhante de métodos (prepare, execute, etc.), embora também não haja uma abstração explícita para tal. Seria interessante se tivéssemos considerado a utilização do padrão Singleton para garantir apenas uma instância da conexão com o banco de dados em toda a aplicação, porém os integrantes do grupo não sabem como aplicar isso.

A seguir, será explicado os principais métodos de cada classe:

2.6. Explicações das classes:

2.6.1. Classe Usuario:

No início da classe Usuario são criados os atributos, construtores e os métodos get e setters.

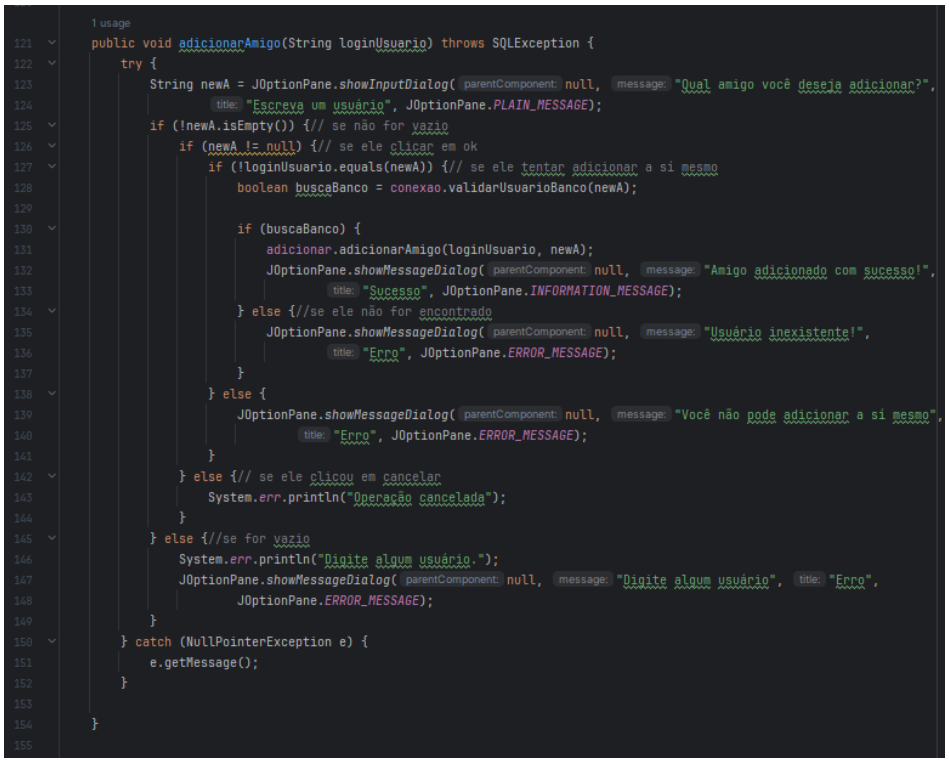
O método cadastrar () é responsável por coletar informações do usuário (nome, e-mail, senha) por meio de caixas de diálogo “JOptionPane”. Realiza validações sobre os dados inseridos para garantir que não sejam vazios ou nulos. O uso de throws SQLException indica que esse método pode lançar uma exceção do tipo SQLException, que precisa ser tratada quando usado. Em seguida, verifica se o usuário já existe no banco de dados por meio de chamadas aos métodos da classe “ConexaoPostgre” (“autenticarUsuario”). Se não existir, utiliza o método “inserirUsuario” para adicionar o novo usuário ao banco de dados.

```java
public void cadastrar() throws SQLException {
    try {
                nome = JOptionPane.showInputDialog( parentComponent: null, message: "Digite seu nome:", title: "",
                        JOptionPane.PLAIN_MESSAGE);
        while (nome.length() < 1 || nome.isEmpty() || nome.isBlank()) {
            nome = JOptionPane.showInputDialog( parentComponent: null, message: "Inválido! \nDigite novamente:",
                    title: "Erro", JOptionPane.ERROR_MESSAGE);
        }
        email = JOptionPane.showInputDialog( parentComponent: null, message: "Digite seu e-mail:", title: "",
                JOptionPane.PLAIN_MESSAGE);
        while (email.length() < 1 || email.isEmpty() || email.isBlank()) {
            email = JOptionPane.showInputDialog( parentComponent: null, message: "Inválido! \nDigite novamente:",
                    title: "Erro", JOptionPane.ERROR_MESSAGE);
        }
        senha = JOptionPane.showInputDialog( parentComponent: null, message: "Crie uma senha:", title: "",
                JOptionPane.PLAIN_MESSAGE);
        while (senha.length() < 1 || senha.isEmpty() || senha.isBlank()) {
            senha = JOptionPane.showInputDialog( parentComponent: null, message: "Inválida! \nDigite novamente:",
                    title: "Erro", JOptionPane.ERROR_MESSAGE);
        }

        if (conexao.autenticarUsuario(nome) == true) {
            JOptionPane.showMessageDialog( parentComponent: null, message: "Nome já está em uso! Por favor, " +
                    "escreva um novo nome.", title: "", JOptionPane.ERROR_MESSAGE);
        } else if (conexao.autenticarUsuario(email) == true) {
            JOptionPane.showMessageDialog( parentComponent: null, message: "O email já está em uso. Por favor, " +
                    "escreva outro email.", title: "", JOptionPane.ERROR_MESSAGE);
        } else {
            novoUsuario.inserirUsuario(nome, email, senha);
            Usuario user = new Usuario(nome, email, senha);
            usuarios.add(user);
        }
    } catch (NullPointerException e) {
        System.err.println("Operação cancelada");
    }

}
```

Em seguida, o método logarUsuario(String vNome, String vSenha) realiza a validação do login do usuário. Usa o método “verificarUsuario” da classe “AdicionarAmigo.LogarUsuario” para autenticar as credenciais do usuário no banco de dados. O throws SQLException, Exception indica que pode lançar exceções desses tipos caso ocorra um problema ao executar operações no banco de dados. Se a autenticação for bem-sucedida, exibe uma mensagem de boas-vindas.

```java
    1 usage
105 public boolean logarUsuario(String vNome, String vSenha) throws SQLException, Exception {
106
107     boolean buscaBanco = login.verificarUsuario(vNome, vSenha);
108
109     if (buscaBanco) {
110         JOptionPane.showMessageDialog( parentComponent: null, message: "Bem vindo a mini Rede Social!",
111                 title: "", JOptionPane.INFORMATION_MESSAGE);
112         return true;
113     } else {
114         JOptionPane.showMessageDialog( parentComponent: null, message: "Acesso negado! Usuário ou senha inválidos.",
115                 title: "Erro", JOptionPane.ERROR_MESSAGE);
116         return false;
117     }
118
119 }
120
```

O método adicionarAmigo(String loginUsuario) solicita ao usuário que insira o nome do amigo que deseja adicionar. Verifica se o nome não está vazio e se corresponde a um usuário existente no banco de dados. Se o usuário existir, utiliza o método “autenticarUsuario” da classe “AdicionarAmigo” para adicionar a conexão entre os dois usuários como amigos no banco de dados. Pode lançar uma exceção SQLException.



```
1 usage
public void adicionarAmigo(String loginUsuario) throws SQLException {
    try {
        String newA = JOptionPane.showInputDialog( parentComponent: null, message: "Qual amigo você deseja adicionar?",
                title: "Escreva um usuário", JOptionPane.PLAIN_MESSAGE);
        if (!newA.isEmpty()) {// se não for vazio
            if (newA != null) {// se ele clicar em ok
                if (!loginUsuario.equals(newA)) {// se ele tentar adicionar a si mesmo
                    boolean buscaBanco = conexao.validarUsuarioBanco(newA);

                    if (buscaBanco) {
                        adicionar.adicionarAmigo(loginUsuario, newA);
                        JOptionPane.showMessageDialog( parentComponent: null, message: "Amigo adicionado com sucesso!",
                                title: "Sucesso", JOptionPane.INFORMATION_MESSAGE);
                    } else {//se ele não for encontrado
                        JOptionPane.showMessageDialog( parentComponent: null, message: "Usuário inexistente!",
                                title: "Erro", JOptionPane.ERROR_MESSAGE);
                    }
                } else {
                    JOptionPane.showMessageDialog( parentComponent: null, message: "Você não pode adicionar a si mesmo",
                            title: "Erro", JOptionPane.ERROR_MESSAGE);
                }
            } else {// se ele clicou em cancelar
                System.err.println("Operação cancelada");
            }
        } else {//se for vazio
            System.err.println("Digite algum usuário.");
            JOptionPane.showMessageDialog( parentComponent: null, message: "Digite algum usuário", title: "Erro",
                    JOptionPane.ERROR_MESSAGE);
        }
    } catch (NullPointerException e) {
        e.getMessage();
    }

}
```

O método excluirAmigo(String loginUsuario) solicita ao usuário o nome do amigo que deseja excluir, realiza validações semelhantes ao método”`adicionarAmigo”. Se o amigo existir, utiliza o método “excluirAmigo” da classe “AdicionarAmigo.ExcluirAmigo” para remover a conexão de amizade entre os usuários, interagindo com o banco de dados. Também pode lançar uma exceção SQLException.

```java
public void excluirAmigo(String loginUsuario) throws SQLException {
    try {
        String amigoExcluir = JOptionPane.showInputDialog( parentComponent: null, message: "Qual amigo você quer excluir?",
                title: "Escreva o email o Amigo que deseja Excluir", JOptionPane.PLAIN_MESSAGE);
        if(!amigoExcluir.isEmpty()) {
            boolean buscaBanco = conexao.autenticarUsuario(amigoExcluir);
            if (buscaBanco) {
                excluir.excluirAmigo(loginUsuario, amigoExcluir);
                JOptionPane.showMessageDialog( parentComponent: null, message: "Amigo excluído com sucesso!",
                        title: "Deu certo", JOptionPane.INFORMATION_MESSAGE);
            } else {
                JOptionPane.showMessageDialog( parentComponent: null, message: "Amigo não foi encontrado.",
                        title: "Erro", JOptionPane.ERROR_MESSAGE);
            }
        }else{
            System.err.println("Digite o email de algum usuário.");
            JOptionPane.showMessageDialog( parentComponent: null, message: "Digite o email de algum usuário.",
                    title: "Erro", JOptionPane.ERROR_MESSAGE);
        }
    } catch(NullPointerException e){
        e.getMessage();
    }

}
```

O método consultarAmigos(String loginUsuario) invoca o método “consultarAmigos” da classe “AdicionarAmigo.consultarAmigos”, que busca no banco de dados todos os amigos relacionados ao usuário e exibe uma lista de amigos em uma caixa de diálogo. Também pode lançar uma exceção SQLException.

```java
    1 usage
181     public void consultarAmigos(String loginUsuario) throws SQLException {
182         consultar.consultarAmigos(loginUsuario);
183
184     }
185
```

O método enviarMensagens(String loginUsuario) e listarMensagens(String loginUsuario) são responsáveis por enviar mensagens para um amigo específico e listar mensagens trocadas entre o usuário e um amigo, respectivamente. Ambos interagem com o banco de dados por meio dos métodos `enviarMensagem` e `listarMensagens` da classe `AdicionarAmigo.EnviarMensagem` e `AdicionarAmigo.ListarMensagens`. Ambos podem lançar uma exceção SQLException.

```java
public void enviarMensagens(String loginUsuario) throws SQLException {
    String mensagemDoAmigo = JOptionPane.showInputDialog( parentComponent: null,
            message: "Para qual amigo deseja enviar mensagem?", title: "Digite o email do Amigo", JOptionPane.QUESTION_MESSAGE);
    boolean amigoEncontrado = conexao.autenticarUsuario(mensagemDoAmigo);
    try {
        if(mensagemDoAmigo != null) {
            if (amigoEncontrado) {
                String conteudo = JOptionPane.showInputDialog( parentComponent: null, message: "Digite a mensagem:",
                        title: "", JOptionPane.PLAIN_MESSAGE);
                if(!conteudo.isEmpty()) {
                    enviar.enviarMensagem(loginUsuario, mensagemDoAmigo, conteudo);
                    JOptionPane.showMessageDialog( parentComponent: null, message: "Mensagem enviada!",
                            title: "Sucesso", JOptionPane.INFORMATION_MESSAGE);
                }else{
                    System.err.println("Conteúdo da mensagem não pode estar vazio.");
                    JOptionPane.showMessageDialog( parentComponent: null, message: "Digite alguma mensagem!",
                            title: "Erro", JOptionPane.ERROR_MESSAGE);
                }
            } else {
                JOptionPane.showMessageDialog( parentComponent: null, message: "Amigo não encontrado.",
                        title: "Erro!", JOptionPane.ERROR_MESSAGE);
            }
        }else{
            System.err.println("Operação cancelada.");
        }
    } catch(NullPointerException e){
        e.getMessage();
    }
}
```

```java
public void listarMensagens(String loginUsuario) throws SQLException {
    try {
        String conversa = JOptionPane.showInputDialog( parentComponent: null,
                message: "Deseja ver a conversa com qual amigo?", title: "Digite o email do Amigo",
                JOptionPane.PLAIN_MESSAGE);
        if(!conversa.isEmpty()) {
            boolean buscaBanco = conexao.autenticarUsuario(conversa);

            if (buscaBanco) {
                listar.listarMensagens(loginUsuario, conversa);
            } else {
                JOptionPane.showMessageDialog( parentComponent: null,
                        message: "Não há registro de mensagens com esse usuário.",
                        title: "Conversa vazia", JOptionPane.INFORMATION_MESSAGE);
            }
        }else{
            System.err.println("Insira algum amigo");
            JOptionPane.showMessageDialog( parentComponent: null,
                    message: "Insira algum amigo", title: "Erro",
                    JOptionPane.ERROR_MESSAGE);
        }
    } catch(NullPointerException e){
        e.getMessage();
    }

}
```

Além disso, são utilizados blocos try-catch para capturar exceções específicas, como NullPointerException quando alguma operação for realizada com valores nulos. O uso desses blocos ajuda a tratar exceções que podem ocorrer durante a execução dos métodos. O método JOptionPane.showInputDialog é utilizado para exibir caixas de diálogo interativas para o usuário inserir informações. Esses métodos estão intimamente conectados à lógica de interação com o usuário, validação de dados e manipulação do banco de dados por meio das classes que cuidam da conexão e operações no banco (“ConexaoPostgre” e classes internas como em “AdicionarAmigo”).

2.6.2. Classe Adicionarmigo:

Essa classe define uma série de classes dentro do pacote “bdconexao” que interage com o banco de dados “redesocial0”.

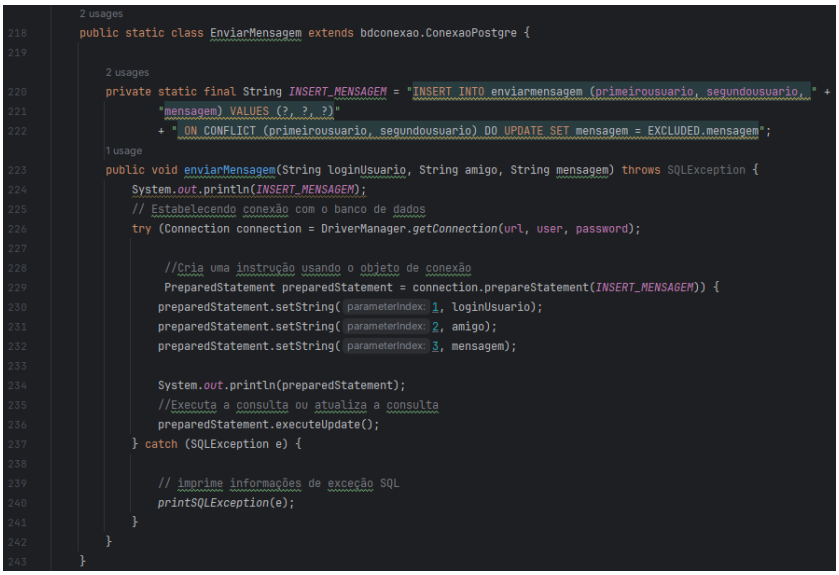
O método “adicionarAmigo(String loginUsuario, String novoAmigo)” insere um novo registro na tabela amigos do banco de dados. Possui os parâmetros: “loginUsuario” que representa o usuário que está conectando um novo amigo e “novoAmigo” que Indica o usuário que está sendo adicionado como amigo.

```
    2 usages
    private static final String INSERT_AMIGO = "INSERT INTO amigos" +
            " (usuario1, usuario2) VALUES " +
            " (?, ?);";

    1 usage
    public void adicionarAmigo(String loginUsuario, String novoAmigo) throws SQLException {
        System.out.println(INSERT_AMIGO);
        // Estabelecendo conexão com o banco de dados
        try (Connection connection = DriverManager.getConnection(url, user, password);

             //Cria uma instrução usando o objeto de conexão
             PreparedStatement preparedStatement = connection.prepareStatement(INSERT_AMIGO)) {
            preparedStatement.setString( parameterIndex: 1, loginUsuario);
            preparedStatement.setString( parameterIndex: 2, novoAmigo);

            System.out.println(preparedStatement);
            //Executa a consulta ou atualiza a consulta
            preparedStatement.executeUpdate();
        } catch (SQLException e) {

            // imprime informações de exceção SQL
            printSQLException(e);
        }
    }
```

Em seguida temos a classe interna estática cadastrar () responsável por adicionar um novo usuário à tabela usuários do banco de dados. Possui os parâmetros “nome” que é o nome do novo usuário, “e-mail” sendo o e-mail do novo usuário e “senha” a senha associada ao novo usuário.

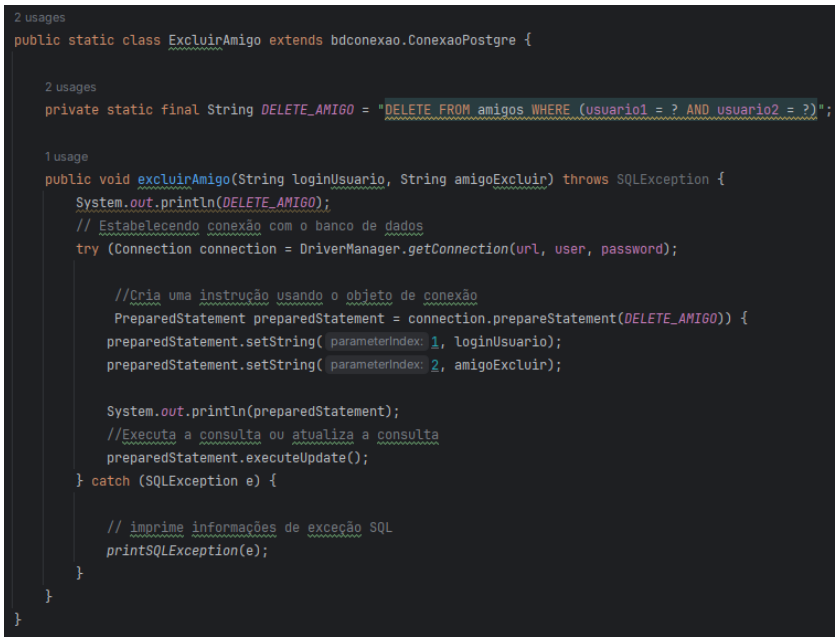
```
public static class Cadastrar extends ConexaoPostgre {

    2 usages
    private static final String INSERT_USER = "INSERT INTO usuarios" +
            " (nome, email, senha) VALUES " +
            " (?, ?, ?);";

    1 usage
    public void inserirUsuario(String nome, String email, String senha) throws SQLException {
        System.out.println(INSERT_USER);
        // Estabelecendo conexão com o banco de dados
        try (Connection connection = DriverManager.getConnection(url, user, password);

             //Cria uma instrução usando o objeto de conexão
             PreparedStatement preparedStatement = connection.prepareStatement(INSERT_USER)) {
            preparedStatement.setString( parameterIndex: 1, nome);
            preparedStatement.setString( parameterIndex: 2, email);
            preparedStatement.setString( parameterIndex: 3, senha);
            System.out.println(preparedStatement);
            //Executa a consulta ou atualiza a consulta
            preparedStatement.executeUpdate();
        } catch (SQLException e) {

            // imprime informações de exceção SQL
            printSQLException(e);
        }
    }

}
```

A classe interna estática EnviarMensagem() insere uma nova mensagem na tabela enviarmensagem. Possui os parâmetros: “loginUsuario” usuário que está enviando a mensagem, “amigo” usuário destinatário da mensagem e “mensagem” conteúdo da mensagem enviada.



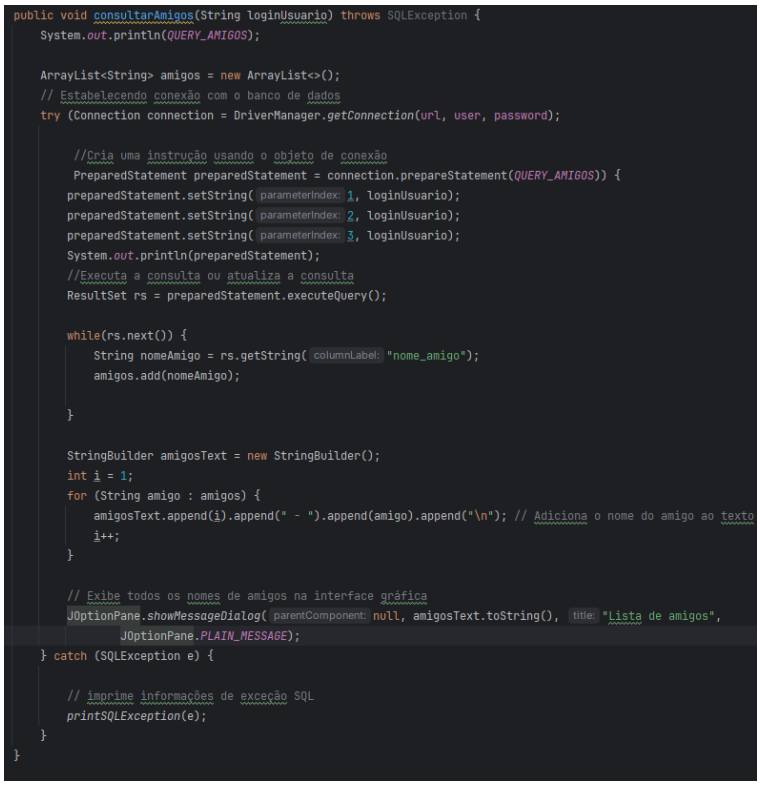
```
2 usages
public static class EnviarMensagem extends bdconexao.ConexaoPostgre {

    2 usages
    private static final String INSERT_MENSAGEM = "INSERT INTO enviarmensagem (primeirousuario, segundousuario, " +
            "mensagem) VALUES (?, ?, ?)"
            + " ON CONFLICT (primeirousuario, segundousuario) DO UPDATE SET mensagem = EXCLUDED.mensagem";
    1 usage
    public void enviarMensagem(String loginUsuario, String amigo, String mensagem) throws SQLException {
        System.out.println(INSERT_MENSAGEM);
        // Estabelecendo conexão com o banco de dados
        try (Connection connection = DriverManager.getConnection(url, user, password);

             //Cria uma instrução usando o objeto de conexão
             PreparedStatement preparedStatement = connection.prepareStatement(INSERT_MENSAGEM)) {
            preparedStatement.setString( parameterIndex: 1, loginUsuario);
            preparedStatement.setString( parameterIndex: 2, amigo);
            preparedStatement.setString( parameterIndex: 3, mensagem);

            System.out.println(preparedStatement);
            //Executa a consulta ou atualiza a consulta
            preparedStatement.executeUpdate();
        } catch (SQLException e) {

            // imprime informações de exceção SQL
            printSQLException(e);
        }
    }
}
```

A Classe interna estática ExcluirAmigo() remove a conexão de amizade entre dois usuários da tabela amigos. Possui os parâmetros: “loginUsuario” usuário que deseja remover o amigo e “amigoExcluir” usuário que será excluído da lista de amigos.



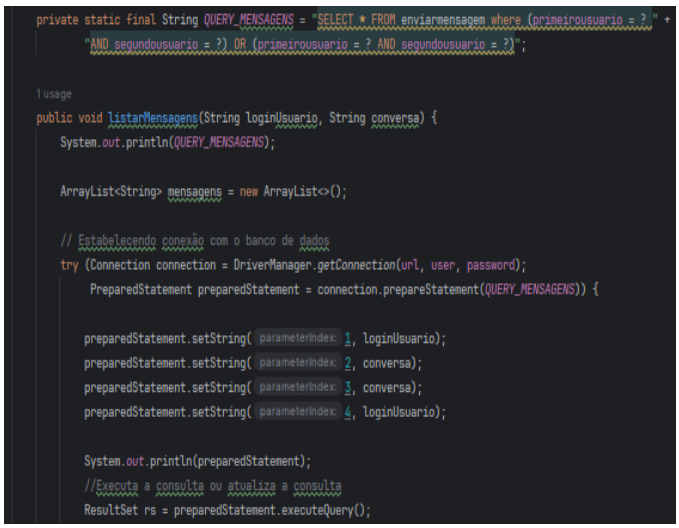
```
2 usages
public static class ExcluirAmigo extends bdconexao.ConexaoPostgre {

    2 usages
    private static final String DELETE_AMIGO = "DELETE FROM amigos WHERE (usuario1 = ? AND usuario2 = ?)";

    1 usage
    public void excluirAmigo(String loginUsuario, String amigoExcluir) throws SQLException {
        System.out.println(DELETE_AMIGO);
        // Estabelecendo conexão com o banco de dados
        try (Connection connection = DriverManager.getConnection(url, user, password);

             //Cria uma instrução usando o objeto de conexão
             PreparedStatement preparedStatement = connection.prepareStatement(DELETE_AMIGO)) {
            preparedStatement.setString( parameterIndex: 1, loginUsuario);
            preparedStatement.setString( parameterIndex: 2, amigoExcluir);

            System.out.println(preparedStatement);
            //Executa a consulta ou atualiza a consulta
            preparedStatement.executeUpdate();
        } catch (SQLException e) {

            // imprime informações de exceção SQL
            printSQLException(e);
        }
    }
}
```

A classe interna estática consultarAmigos() busca e exibe todos os amigos de um usuário específico. Possui o parâmetro: “loginUsuario” usuário para o qual se deseja verificar os amigos.



```
public void consultarAmigos(String loginUsuario) throws SQLException {
    System.out.println(QUERY_AMIGOS);

    ArrayList<String> amigos = new ArrayList<>();
    // Estabelecendo conexão com o banco de dados
    try (Connection connection = DriverManager.getConnection(url, user, password);

         //Cria uma instrução usando o objeto de conexão
         PreparedStatement preparedStatement = connection.prepareStatement(QUERY_AMIGOS)) {
        preparedStatement.setString( parameterIndex: 1, loginUsuario);
        preparedStatement.setString( parameterIndex: 2, loginUsuario);
        preparedStatement.setString( parameterIndex: 3, loginUsuario);
        System.out.println(preparedStatement);
        //Executa a consulta ou atualiza a consulta
        ResultSet rs = preparedStatement.executeQuery();

        while(rs.next()) {
            String nomeAmigo = rs.getString( columnLabel: "nome_amigo");
            amigos.add(nomeAmigo);

        }

        StringBuilder amigosText = new StringBuilder();
        int i = 1;
        for (String amigo : amigos) {
            amigosText.append(i).append(" - ").append(amigo).append("\n"); // Adiciona o nome do amigo ao texto
            i++;
        }

        // Exibe todos os nomes de amigos na interface gráfica
        JOptionPane.showMessageDialog( parentComponent: null, amigosText.toString(), title: "Lista de amigos",
                JOptionPane.PLAIN_MESSAGE);
    } catch (SQLException e) {

        // imprime informações de exceção SQL
        printSQLException(e);
    }
}
```

A classe interna estática ListarMensagens() recupera e exibe as mensagens trocadas entre dois usuários. Possui os parâmetros: “loginUsuario” usuário que está consultando as mensagens e “conversa” usuário com quem estão sendo trocadas as mensagens.



```
private static final String QUERY_MENSAGENS = "SELECT * FROM enviarmensagem where (primeirousuario = ?) " +
        "AND segundousuario = ?) OR (primeirousuario = ? AND segundousuario = ?)";

1 usage
public void listarMensagens(String loginUsuario, String conversa) {
    System.out.println(QUERY_MENSAGENS);

    ArrayList<String> mensagens = new ArrayList<>();

    // Estabelecendo conexão com o banco de dados
    try (Connection connection = DriverManager.getConnection(url, user, password);
         PreparedStatement preparedStatement = connection.prepareStatement(QUERY_MENSAGENS)) {

        preparedStatement.setString( parameterIndex: 1, loginUsuario);
        preparedStatement.setString( parameterIndex: 2, conversa);
        preparedStatement.setString( parameterIndex: 3, conversa);
        preparedStatement.setString( parameterIndex: 4, loginUsuario);

        System.out.println(preparedStatement);
        //Executa a consulta ou atualiza a consulta
        ResultSet rs = preparedStatement.executeQuery();
```

```
        while (rs.next()) {
            String usuarioEnvio = rs.getString( columnLabel: "primeirousuario");
            String mensagem = rs.getString( columnLabel: "mensagem");

            if (loginUsuario.equals(usuarioEnvio)) {
                // Se for o remetente, adiciona a mensagem como enviada por ele
                mensagens.add("Você: " + mensagem);
            } else {
                // Se for o destinatário, adiciona a mensagem como recebida dele
                mensagens.add(conversa + ": " + mensagem);
            }
        }

        StringBuilder mensagensText = new StringBuilder();
        for (String mensagem : mensagens) {
            mensagensText.append(mensagem).append("\n");
        }

        JOptionPane.showMessageDialog( parentComponent: null, mensagensText.toString(),
                title: "Conversa com "+ conversa, JOptionPane.PLAIN_MESSAGE);
    } catch (SQLException e) {
        // imprime informações de exceção SQL
        printSQLException(e);
    }
}
```

Os métodos seguem estruturas parecidas, primeiro, “INSERT_X” é a constante que contém o comando SQL para inserir dados nas tabelas do bando de dados “redesocial0”, depois imprime no console o comando SQL que será executado para inserir o novo amigo, para cadastrar um amigo, enviar mensagens, dentre outros. Isso é útil para depuração e verificação. Em seguida, o try estabelece uma conexão com o banco de dados usando o “DriverManager, passando as credenciais” (URL, usuário e senha). O “PreparedStatement” cria um objeto “preparedStatement” a partir do comando SQL definido anteriormente, em seguida é definido os valores a serem inseridos nos lugares dos placeholders “?”, isso evita problemas como SQL injection. Os valores substituídos do comando SQL final são impressos para serem executados no banco de dados. O “preparedStatement.executeUpdate()” executa a atualização no banco de dados, por exemplo, insere um novo registro de amizade ou um novo registro de usuário, dentre outros. Em caso de exceção SQL durante a execução da inserção, o método “printSQLException()” é chamado para lidar com a exceção e fornecer informações detalhadas sobre o erro. O uso do “try-with-resources” garante que a conexão com o banco de dados seja fechada automaticamente após a execução do bloco try, independentemente do resultado da operação.

Temos alguns casos que se diferem como por exemplo o uso ON CONFLICT em algumas classes estáticas internas que especifica a ação a ser tomada em caso de conflito de chave primária. Outro caso é a constante “QUERY_AMIGOS” que armazena uma consulta SQL, essa consulta é usada para recuperar os amigos de um usuário a partir da tabela amigos. A consulta usa uma cláusula CASE para determinar os amigos, independentemente se eles estão na coluna “usuario1” ou “usuario2” e “QUERY_MENSAGENS” que armazena para recuperar as mensagens trocadas entre dois usuários específicos, “primeirousuario” e “segundousuario”.O ArrayList utilizados nas classes “ConsultarAmigos” e “ListarMensagens” inicializa uma lista vazia para armazenar os nomes dos amigos e as mensagens obtidas do banco de dados. O “executeQuery()” Executa a consulta SQL no banco de dados e armazena o resultado em um ResultSet e o “while” itera sobre o resultado (ResultSet) para obter os amigos e as mensagens retornados pela consulta. O StringBuilder cria a mensagem de texto e constrói uma única mensagem com todas as mensagens da lista para ser exibida na interface gráfica e o último caso o

JOptionPane.showMessageDialog(...): que uma caixa de diálogo com os nomes dos amigos recuperados e o histórico da conversa entre os usuários.

Por fim, o método “verificarUsuario” dentro da classe “LogarUsuario” é responsável por validar as credenciais de um usuário no banco de dados. O “VALIDAR_USER” é uma constante que armazena uma consulta SQL para verificar se um usuário com um determinado nome e senha existe na tabela “usuarios”. O “boolean verificarUsuario...”: recebe o nome de usuário e senha como parâmetros para verificar a existência do usuário no banco de dados. O “boolean amigoEncontrado = false” inicializa uma variável booleana que indicará se o usuário foi encontrado no banco de dados. Depois ocorre o estabelecimento da conexão e criação do PreparedStatement como nas classes anteriores. O “resultSet.next()” avança para o primeiro resultado e o “resultSet.getInt("count")” recupera o valor da coluna "count" do resultado, se “resultSet.next()” retornar verdadeiro e o valor de "count" for maior que zero, significa que o usuário foi encontrado. Portanto, “amigoEncontrado` é definido como verdadeiro”. O “return amigoEncontrado” retorna o valor booleano indicando se o usuário foi encontrado no banco de dados com base nas credenciais fornecidas, esse método retorna `true` se o usuário com o nome e senha fornecidos existir no banco de dados, caso contrário, retorna `false`.

```java
public static class LogarUsuario extends bdconexao.ConexaoPostgre {

    1 usage
    private static final String VALIDAR_USER ="SELECT COUNT(*) AS count FROM usuarios WHERE email = ? " +
            "AND senha = ?";

    2 usages
    public boolean verificarUsuario(String email, String senha) throws SQLException {
        boolean amigoEncontrado = false;

        try (Connection connection = DriverManager.getConnection(url, user, password);

             PreparedStatement preparedStatement = connection.prepareStatement(VALIDAR_USER)) {

            preparedStatement.setString( parameterIndex: 1, email);
            preparedStatement.setString( parameterIndex: 2, senha);

            ResultSet resultSet = preparedStatement.executeQuery();

            System.out.println(preparedStatement);

            // Se houver algum resultado na consulta, significa que o usuário foi encontrado
            if (resultSet.next() && resultSet.getInt( columnLabel: "count") > 0) {
                amigoEncontrado = true;
            }
        } catch (SQLException e) {
            printSQLException(e);
        }

        return amigoEncontrado;
    }

}
```

2.6.3. Classe ConexaoPostgre:

A classe “ConexaoPostgre” lida com a conexão com um banco de dados PostgreSQL para operações relacionadas à verificação de usuários por nome e e-mail. Alguns comandos utilizados nessa classe foram explicados anteriormente. No início são definidos alguns atributos como a URL de conexão com o banco de dados PostgreSQL, o usuário do banco de dados e a senha do usuário do banco de dados. Depois é feita uma consulta SQL que verifica a existência de um usuário com um determinado nome e e-mail na tabela “usuarios” com o método “VALIDAR_QUERY_NOME ="SELECT COUNT(*)...”.

```
9 usages  7 inheritors
public class ConexaoPostgre {
    9 usages
    public final String url = "jdbc:postgresql://localhost/redesocial0";
    9 usages
    public final String user = "postgres";
    9 usages
    public final String password = "123456";

    1 usage
    private static final String VALIDAR_QUERY_NOME ="SELECT COUNT(*) AS count FROM usuarios WHERE nome =?";
    1 usage
    private static final String VALIDAR_QUERY_EMAIL ="SELECT COUNT(*) AS count FROM usuarios WHERE email =?";

```

Os métodos “autenticarUsuario” e “autenticarUsuarioEmail” recebe um nome de usuário como parâmetro e verifica se esse nome e e-mail existe na tabela `usuarios` no banco de dados PostgreSQL. Retorna um booleano indicando se o usuário foi encontrado.

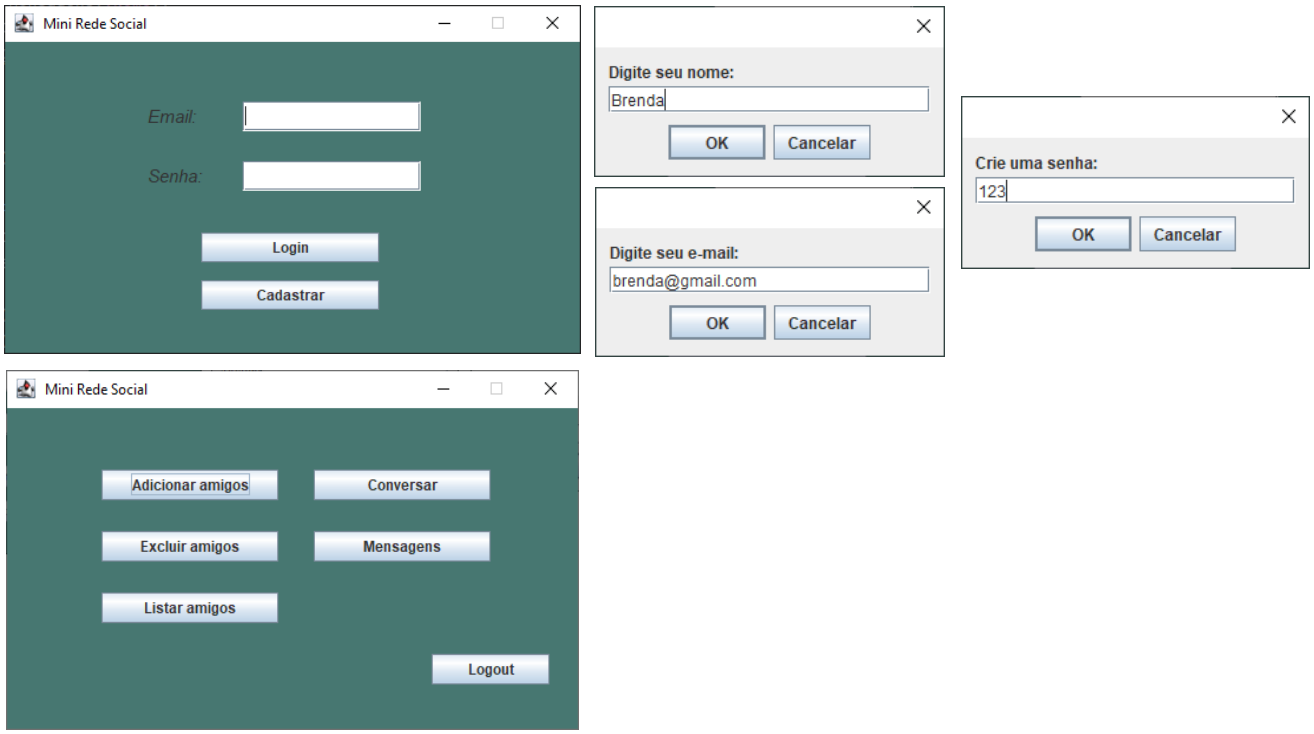
```
6 usages
public boolean autenticarUsuario(String nome) throws SQLException {
    boolean encontrado = false;

    try (Connection connection = DriverManager.getConnection(url, user, password);

         PreparedStatement preparedStatement = connection.prepareStatement(VALIDAR_QUERY_NOME)) {

        preparedStatement.setString( parameterIndex: 1, nome);

        ResultSet resultSet = preparedStatement.executeQuery();

        System.out.println(preparedStatement);

        // Se der resultado, é porque o usuario foi encontrado
        if (resultSet.next() && resultSet.getInt( columnLabel: "count") > 0) {
            encontrado = true;
        }
    } catch (SQLException e) {
        printSQLException(e);
    }

    return encontrado;
}
```

```
public boolean autenticarEmailUsuario(String email) throws SQLException {
    boolean encontrado = false;

    try (Connection connection = DriverManager.getConnection(url, user, password);

         PreparedStatement preparedStatement = connection.prepareStatement(VALIDAR_QUERY_EMAIL)) {

        preparedStatement.setString( parameterIndex: 1, email);

        ResultSet resultSet = preparedStatement.executeQuery();

        System.out.println(preparedStatement);


        if (resultSet.next() && resultSet.getInt( columnLabel: "count") > 0) {
            encontrado = true;
        }
    } catch (SQLException e) {
        printSQLException(e);
    }

    return encontrado;
}
```

O método “public static void printSQLException(SQLException ex)” lida com a impressão de exceções do tipo “SQLException”, exibindo informações detalhadas sobre a exceção, incluindo a “SQLState”, “ErrorCode”, e mensagens associadas. Isso ajuda na depuração e tratamento de erros relacionados ao banco de dados.

```
    9 usages
65 @    public static void printSQLException(SQLException ex) {
66          for (Throwable e: ex) {
67              if (e instanceof SQLException) {
68                  e.printStackTrace(System.err);
69                  System.err.println("SQLState: " + ((SQLException) e).getSQLState());
70                  System.err.println("Error Code: " + ((SQLException) e).getErrorCode());
71                  System.err.println("Message: " + e.getMessage());
72                  Throwable t = ex.getCause();
73                  while (t != null) {
74                      System.out.println("Cause: " + t);
75                      t = t.getCause();
76                  }
77              }
78          }
79      }
80
81  }
```

2.7. Interface gráfica



2.8. Banco de dados

O banco de dados "redesocial0" foi projetado para suportar uma rede social básica. Ele é composto por três tabelas principais: "usuarios", "amigos" e "enviarmensagem". Cada tabela desempenha um papel fundamental na funcionalidade e na estrutura da rede social.

A tabela “usuários” armazena informações sobre os usuários da rede social, contém colunas como “nome”, “senha” e “email” e Restringe os campos “nome”, “senha” e “email” como obrigatórios (NOT NULL). O “nome” é um campo do tipo VARCHAR(255), capaz de armazenar até 255 caracteres, a “senha” é do tipo VARCHAR(20), permitindo senhas com até 20 caracteres e “email” é um campo do tipo VARCHAR(255) para armazenar endereços de e-mail.

A tabela “amigos” é responsável por manter o relacionamento entre os usuários que são amigos na rede social. Tem duas colunas: “usuario1” e “usuario2”, ambas do tipo VARCHAR(70). Essa tabela não permite que os campos de usuário sejam nulos (NOT NULL). Registra as conexões de amizade entre dois usuários por meio das colunas “usuario1” e “usuario2”.

A tabela “enviarmensagem” é projetada para armazenar mensagens enviadas entre usuários. Contém as colunas “primeirousuario”, “segundousuario” e “mensagem”. “primeirousuario” e “segundousuario” são do tipo VARCHAR(200), representando os usuários envolvidos na troca de mensagens, “mensagem” é do tipo VARCHAR(255) e armazena o conteúdo da mensagem. A combinação de 'primeirousuario' e 'segundousuario' forma a chave primária desta tabela (PRIMARY KEY), garantindo que cada par de usuários possa ter apenas uma única troca de mensagem.

As possíveis melhorias poderiam ser por exemplo a adição de campos adicionais na tabela “usuários”, como data de registro, informações de perfil, entre outros. Também poderia ser feito a implementação de restrições de chave estrangeira para manter a integridade referencial, garantindo que apenas usuários existentes possam ser adicionados como amigos ou enviar mensagens. Esse banco de dados fornece uma estrutura básica para uma rede social, mas é importante considerar aspectos de segurança, escalabilidade e usabilidade ao desenvolver uma aplicação real baseada nele.

3. CONCLUSÃO

Durante o desenvolvimento do nosso projeto de Mini Simulador de Rede Social, enfrentamos desafios consideráveis ao implementar o banco de dados, especialmente por não termos experiência prévia nessa área. Apesar dos obstáculos, conseguimos concluir a implementação do banco de dados e finalizar o projeto.

Além disso, vale destacar que a dificuldade para implementar banco de dados foi enriquecedora. A compreensão das estruturas de armazenamento de dados e sua integração com a lógica da aplicação nos proporcionou insights valiosos sobre a importância da organização e gerenciamento eficiente das informações em um sistema. Essa experiência reforçou nossa determinação em aprimorar continuamente nossas habilidades técnicas e a abordagem para solucionar problemas complexos no desenvolvimento de software.

Reconhecemos que a interface gráfica desempenha um papel fundamental na melhoria da usabilidade e da experiência do usuário. Por isso, continuaremos estudando para que com o tempo possamos aprimorar o projeto e fazer algo mais bonito visualmente.

É também relevante mencionar que decidimos não utilizar uma classe abstrata e nem a interface neste projeto. Embora reconheçamos o poder desses conceitos para compartilhar comportamentos comuns entre classes relacionadas e definir um conjunto de métodos que uma classe deve implementar, nossa abordagem foi focada no uso do conceito de polimorfismos nas classes apresentadas.

Em resumo, apesar das dificuldades encontradas, este projeto nos proporcionou uma base sólida para a construção de uma rede social e uma oportunidade valiosa de aprendizado e aprimoramento de habilidades. Ao enfrentar desafios, adquirimos conhecimento sobre a estruturação de projetos de software e a importância da colaboração em equipe.