



# **Programação Orientada a Objeto e Linguagem de Programação**

## **Mini Simulador de Rede Social**

Arthur Henrique Fernandes Fonseca - 12112619  
Guilherme Neves Rodrigues - 12118711  
Misael Alexandrino - 12118461  
Vinícius de Oliveira da Silva - 12118123

## Introdução

Inicialmente, o grupo reuniu-se para estruturar como iriam desenvolver o software de rede social. Foi decidido que seria utilizado o Sistema Gerenciador de Banco de Dados MySQL para armazenar e gerenciar as informações essenciais para o funcionamento da plataforma. No momento de criação do diagrama de classes, definiu-se que haveria 4 classes principais: Usuário, Mensagens, Acom migos e Solicitações, as quatro se relacionando entre si, formando uma base sólida para a implementação.

### Estrutura de Dados

A estrutura de dados adotada foi projetada para ser compatível com um número indefinido de entradas, garantindo a escalabilidade e flexibilidade necessárias para acomodar um crescimento orgânico da plataforma. Cada classe no sistema desempenha um papel fundamental na organização e manipulação dos dados:

**Usuário:** Representa os usuários da rede social e contém informações como nome, e-mail, senha e outros atributos relevantes.

```
package com.example.demo1.Database;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

1 usage
public class Usuario {

    public int id;
    6 usages
    public String nome;
    4 usages
    public String email;
    4 usages
    public String senha;
    4 usages
    public String dtaNascimento;
    4 usages
    public String telefone;
    no usages
    private int userId = 0;

    1 usage
    public boolean incluirUsuario() {
        String query = "INSERT INTO usuario (nome, email, senha, dtaNascimento, telefone) VALUES (?, ?, ?, ?, ?)";
```

Foi criado os métodos incluir usuário, atualizar usuário e remover usuário:

```
1 usage
public boolean incluirUsuario() {
    String query = "INSERT INTO usuario (nome, email, senha, dataNascimento, telefone) VALUES (?, ?, ?, ?, ?)";

    Database obDatabase = new Database( table: "usuario");
    this.id = obDatabase.insert(query, this.nome, this.email, this.senha, this.dataNascimento, this.telefone);
    return true;
}

no usages
public boolean atualizarUsuario() {
    String query = "UPDATE usuario SET nome = ?, email = ?, senha = ?, dataNascimento = ?, telefone = ? WHERE id = ?";

    return (new Database( table: "usuario")).update(query, this.nome, this.email, this.senha, this.dataNascimento,
        this.telefone, this.id);
}

no usages
public boolean excluirUsuario() {
    String condition = "id = " + this.id;

    return (new Database( table: "usuario")).delete(condition);
}
```

Mensagem: Responsável por armazenar as mensagens trocadas entre os usuários, registrando detalhes como remetente, destinatário, data e conteúdo.

```
public class Mensagem {
    1 usage
    private int id;
    2 usages
    private int idUsuario;
    2 usages
    private int idUsuarioAmigo;
    2 usages
    private String conteudo;

    1 usage
    public Mensagem(int id, int idUsuario, int idUsuarioAmigo, String conteudo) {
        this.id = id;
        this.idUsuario = idUsuario;
        this.idUsuarioAmigo = idUsuarioAmigo;
        this.conteudo = conteudo;
    }

    no usages
    public String getConteudo() { return conteudo; }

    no usages
    public int getIdUsuarioAmigo() { return idUsuarioAmigo; }

    no usages
    public int getIdUsuario() { return idUsuario; }
}
```



Amigos: Gerencia as conexões entre os usuários, registrando quem está conectado a quem na rede social.

```
module-info.java (com.example.demo1)  HelloApplication.java  SolicitacaoController.java  FriendsController.java x
1 package com.example.demo1.Controller;
2
3 > import ...
17
18
19 public class FriendsController extends Conexao {
20
21     7 usages
22     public static int qrId = 0;
23
24     @FXML
25     private VBox vboxDynamicLabels; // Referência ao VBox dinâmico do FXML
26     2 usages
27     private BorderPane mainBorderPane; // Referência ao BorderPane principal
28
29     // Método para definir o BorderPane principal
30     1 usage
31     public void setMainBorderPane(BorderPane mainBorderPane) { this.mainBorderPane = mainBorderPane; }
32
33     @FXML
34     public void initialize() {
35         // Obter a lista de amigos usando o método da classe Conexao
36         List<Map<String, Object>> consultarListaAmigos = Conexao.consultarListaAmigos(qrId);
37
38         // Verificar se a lista não está vazia antes de iterar sobre ela
39         if (consultarListaAmigos != null && !consultarListaAmigos.isEmpty()) {
40             // Adiciona um Label, um botão de chat e um botão de remover para cada amigo na lista
41             for (Map<String, Object> amigo : consultarListaAmigos) {
42                 HBox friendBox = new HBox(10);
43
44                 // Extrai o nome e o idUsuarioAmigo do Map
45                 String nomeAmigo = (String) amigo.get("nome_amigo");
46                 int idUsuarioAmigo = (int) amigo.get("idUsuarioAmigo");
47
48                 Label label = new Label(nomeAmigo);
49
50                 Region region = new Region();
51                 HBox.setHgrow(region, Priority.ALWAYS);
52
53                 Button chatButton = new Button("@ Chat");
54                 chatButton.setStyle("-fx-background-color: #4caf50; -fx-text-fill: white; -fx-font-size: 12px;");
55                 chatButton.setOnAction(event -> {
56                     try {
57                         System.out.println(idUsuarioAmigo);
58                         entrarChat(idUsuarioAmigo);
59                     } catch (IOException e) {
60                         throw new RuntimeException(e);
61                     }
62                 });
63
64                 Button removerButton = new Button("Remover");
65                 removerButton.setStyle("-fx-background-color: #880000; -fx-text-fill: white; -fx-font-size: 12px;");
66                 removerButton.setOnAction(event -> removerListaAmigos(qrId, idUsuarioAmigo));
67
68                 friendBox.getChildren().addAll(label, region, chatButton, removerButton);
69                 vboxDynamicLabels.getChildren().add(friendBox);
70             }
71         }
72
73         // Métodos de ação para os botões
74
75     1 usage
76     @FXML
77     private void entrarChat(int idUsuarioAmigo) throws IOException {
78         // Consulta o banco de dados para obter informações do amigo
79         List<Map<String, Object>> amigos = Conexao.consultarListaAmigos(qrId);
80
81         // Procura o amigo com o ID correspondente
82         Map<String, Object> amigoEncontrado = amigos.stream().flatMap(amigo -> Stream.of(amigo))
83             .filter(amigo -> (int) amigo.get("idUsuarioAmigo") == idUsuarioAmigo)
84             .findFirst().orElse(null);
85
86         if (amigoEncontrado != null) {
87             String nomeAmigo = (String) amigoEncontrado.get("nome_amigo");
88
89             FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource("/com/example/demo1/ChatView.fxml"));
90             BorderPane chatScreen = fxmlLoader.load();
91
92             // Obtém o controlador do ChatController e configura o ID e o nome do usuário amigo
93             ChatController chatController = fxmlLoader.getController();
94             chatController.setIdUsuarioAmigo(idUsuarioAmigo);
95             chatController.setQrId(qrId); // Adicione esta linha
96         }
97     }
```

```
module-info.java (com.example.demo1)  HelloApplication.java  SolicitacaoController.java  FriendsController.java x
97
98 // Obtém o texto da mensagem do TextField
99 String mensagemEnviada = chatController.getMessageTextFieldText();
100
101 if (mensagemEnviada != null && !mensagemEnviada.isEmpty()) {
102     Conexao.enviarMensagem(qrId, idUsuarioAmigo, mensagemEnviada);
103 } else {
104     System.out.println("A mensagem está vazia ou nula. Não será enviada.");
105 }
106
107 // Obtém e exibe as mensagens existentes
108 List<Map<String, Object>> mensagensSimplificadas = Conexao.obterMensagensSimplificado(qrId, idUsuari
109 for (Map<String, Object> mensagem : mensagensSimplificadas) {
110     int idUsuarioRemetente = (int) mensagem.get("idUsuario");
111     String conteudo = (String) mensagem.get("conteudo");
112
113     chatController.exibirMensagemSimplificado(mensagem);
114     // Faça o que precisar com idUsuarioRemetente e conteudo
115     System.out.println("ID do Remetente: " + idUsuarioRemetente);
116     System.out.println("Conteúdo da Mensagem: " + conteudo);
117 }
118
119 // Obtém a cena da raíz do BorderPane
120 mainBorderPane.setCenter(chatScreen);
121
122 System.out.println("Entrar no chat");
123 } else {
124
125
126
127
128
129
130
131 > 1 usage
132 public static void getUsuarioId(int userId) { qrId = userId; }
133
134 ;
135
136
137
138 }
139
```

Solicitações: Tem a função de controlar as solicitações de amizade no sistema, conectando o usuário com o outro, oferecendo a opção de se tornar ou não amigos na rede social.

```

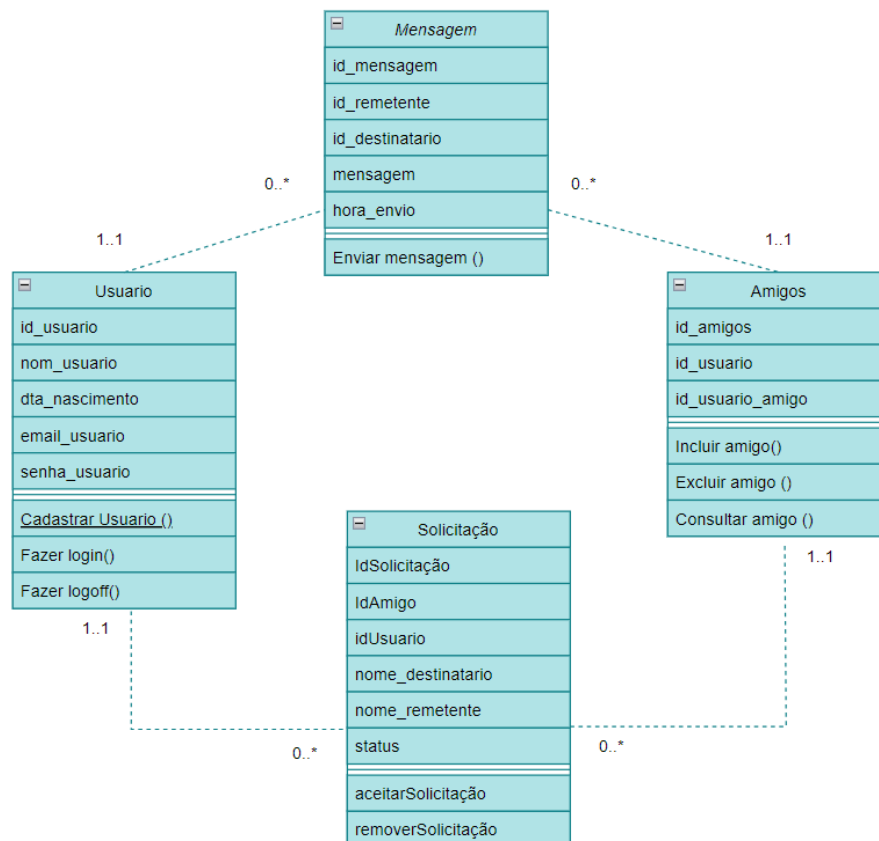
module-info.java (com.example.demo1)  HelloApplication.java  SolicitacaoController.java x
1  package com.example.demo1.Controller;
2  > import ...
19
20  public class SolicitacaoController extends Conexao {
21      5 usages
22      public static int qId = 0;
23      2 usages
24      public static String userName = "";
25      no usages
26      @FXML
27      private ListView<String> listaSolicitacoes;
28
29      @FXML
30      private VBox vboxDynamicLabels; // Corrigido para VBox
31
32      public void initialize() {
33          // Obter a lista de amigos usando o método da classe Conexao
34          Map<String, List<Object>> consultarListaSolicitacoes = Conexao.consultarListaSolicitacoes(qId);
35
36          if (consultarListaSolicitacoes != null && !consultarListaSolicitacoes.isEmpty()) {
37              List<Object> listaIdsSolicitacaoObj = consultarListaSolicitacoes.get("listaIdsSolicitacao");
38              List<Object> listaStringNomesObj = consultarListaSolicitacoes.get("listaStringNomes");
39              List<Object> listaIdAmigoObj = consultarListaSolicitacoes.get("listaIdAmigo");
40
41              for (int i = 0; i < listaIdsSolicitacaoObj.size(); i++) {
42                  if (listaIdsSolicitacaoObj.get(i) instanceof Integer &&
43                      listaStringNomesObj.get(i) instanceof String &&
44                      listaIdAmigoObj.get(i) instanceof Integer) {
45
46                      int idSolicitacao = (int) listaIdsSolicitacaoObj.get(i);
47
48                      int idSolicitacao = (int) listaIdsSolicitacaoObj.get(i);
49                      String nome = (String) listaStringNomesObj.get(i);
50                      int idAmigo = (int) listaIdAmigoObj.get(i);
51
52                      Map<String, Object> dadosSolicitacao = new HashMap<>();
53                      dadosSolicitacao.put("idSolicitacao", idSolicitacao);
54                      dadosSolicitacao.put("nome", nome);
55                      dadosSolicitacao.put("idAmigo", idAmigo);
56
57                      HBox friendBox = new HBox(10);
58                      Label label = new Label(nome);
59
60                      Region region = new Region();
61                      HBox.setHgrow(region, Priority.ALWAYS);
62
63                      Button aceitarButton = new Button("Aceitar");
64                      aceitarButton.setUserData(dadosSolicitacao);
65
66                      aceitarButton.setStyle("-fx-background-color: #4caf50; -fx-text-fill: white; -fx-font-size: 12px; -fx-padding: 5px 15px; -fx-border-radius: 5px; -fx-border: 1px solid #4caf50;");
67                      aceitarButton.setOnAction(event -> {
68                          Map<String, Object> valorUserData = (Map<String, Object>) aceitarButton.getUserData();
69                          System.out.println((String) valorUserData.get("nome"));
70
71                          aceitarListaSolicitacao(qId, (int) valorUserData.get("idSolicitacao"), (String) valorUserData.get("nome"), (int) valorUserData.get("idAmigo"), valorUserData);
72                          removerListaSolicitacao(qId, idSolicitacao);
73                      });
74
75                      Button recusarButton = new Button("Recusar");
76
77                      Map<String, Object> valorUserData = (Map<String, Object>) aceitarButton.getUserData();
78                      System.out.println((String) valorUserData.get("nome"));
79
80                      aceitarListaSolicitacao(qId, (int) valorUserData.get("idSolicitacao"), (String) valorUserData.get("nome"), (int) valorUserData.get("idAmigo"), valorUserData);
81                      removerListaSolicitacao(qId, idSolicitacao);
82
83                      });
84
85                      Button recusarButton = new Button("Recusar");
86                      recusarButton.setStyle("-fx-background-color: #880000; -fx-text-fill: white; -fx-font-size: 12px; -fx-padding: 5px 15px; -fx-border-radius: 5px; -fx-border: 1px solid #880000;");
87                      recusarButton.setOnAction(event -> {
88                          removerListaSolicitacao(qId, idSolicitacao);
89                      });
90
91                      friendBox.getChildren().addAll(label, region, aceitarButton, recusarButton);
92                      vboxDynamicLabels.getChildren().add(friendBox);
93
94                  }
95              }
96
97          1 usage
98          @FXML
99          public static void getIdUsuario(int userId) { qId = userId; }
100          1 usage
101          public static void getNomeUsuario(String userName) { userName = userName; }
102
103      }

```

A interconexão entre essas classes permite uma representação abrangente das relações sociais, mensagens trocadas e detalhes do usuário, proporcionando uma estrutura de dados robusta e eficiente.

## UML - Diagrama de Classe

O diagrama de classe foi elaborado seguindo os princípios da UML (Unified Modeling Language), proporcionando uma visualização clara das relações entre as entidades do sistema. A Figura 1 apresenta o diagrama de classe destacando as classes Usuário, Mensagem e Amigos, bem como os relacionamentos entre elas.



A Figura 1 ilustra de maneira visual a estrutura hierárquica das classes, seus atributos e métodos, oferecendo uma referência valiosa para o desenvolvimento e compreensão do sistema de rede social.

## Banco de Dados

Inicialmente, o grupo reuniu-se para estruturar como iriam desenvolver o software de rede social. Definiu-se que utilizariam o Sistema Gerenciador de Banco de Dados MySQL. No momento de criação do diagrama de classes, definiu-se que haveria 3 classes: **Usuario**, **Mensagem** e **Amigos**, com as três se relacionando entre si.

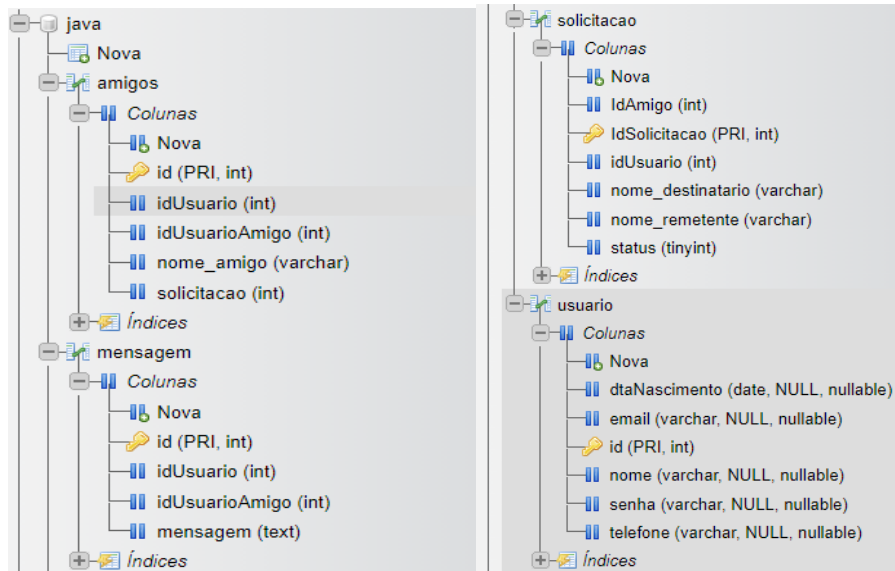
A escolha do MySQL como Sistema Gerenciador de Banco de Dados foi motivada pela familiaridade do grupo com essa ferramenta. A estrutura relacional do MySQL oferece uma organização eficiente dos dados, possibilitando consultas complexas e relacionamentos entre



diferentes entidades. Isso atende às demandas de um sistema que lida com um número variável e dinâmico de entradas

A estrutura do banco de dados MySQL foi projetada considerando as entidades Usuario, Mensagem, Amigos e Solicitações:

Figura 2 - Estrutura do Banco de Dados



Essas tabelas e seus relacionamentos foram projetados para suportar eficientemente as operações necessárias em uma rede social.

O código a seguir é responsável por estabelecer uma conexão com o banco de dados MySQL usando as informações fornecidas e manipular quaisquer exceções que possam ocorrer durante o processo.

```
package com.example.demo1.Database;

import ...

17 usages 3 inheritors
public class Conexao {

    no usages
    public static int qrId = 0;
    1 usage
    private static final String URL = "jdbc:mysql://localhost:3306/java";
    1 usage
    private static final String USUARIO = "root";
    1 usage
    private static final String SENHA = "123456";

    11 usages
    public static Connection obterConexao() throws SQLException {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            return DriverManager.getConnection(URL, USUARIO, SENHA);
        } catch (ClassNotFoundException ex) {
            throw new SQLException("Driver do banco de dados não localizado", ex);
        }
    }
}
```



O código Java na imagem está estabelecendo uma conexão com um banco de dados MySQL. As variáveis URL, USUARIO e SENHA são declaradas nas linhas 5, 6 e 7, respectivamente, e contêm a URL do banco de dados, o nome de usuário e a senha.

Um método público estático chamado Conexao é declarado na linha 9, que retorna um objeto Connection e lança uma SQLException. Este método tenta estabelecer a conexão usando a classe DriverManager, conforme indicado na linha 11.

Se a conexão for bem-sucedida, o objeto de conexão é retornado, como mostrado na linha 12. Caso contrário, uma ClassNotFoundException é lançada, conforme indicado na linha 13.

## Interface Gráfica

No desenvolvimento do projeto, uma parte fundamental é a interface gráfica, que proporciona a interação do usuário com o sistema. O design da interface foi inicialmente prototipado no Figma, uma ferramenta de design colaborativo. As principais telas consideradas foram:

Tela de Login:

Essa tela é o ponto de entrada do usuário no sistema. Ela geralmente contém campos para inserção de e-mail/username e senha.

A utilização de campos de entrada (TextField), botões de login (Button), e possíveis mensagens de erro ou feedbacks são componentes típicos dessa tela.

Figura 3 - Tela de Login (Protótipo)

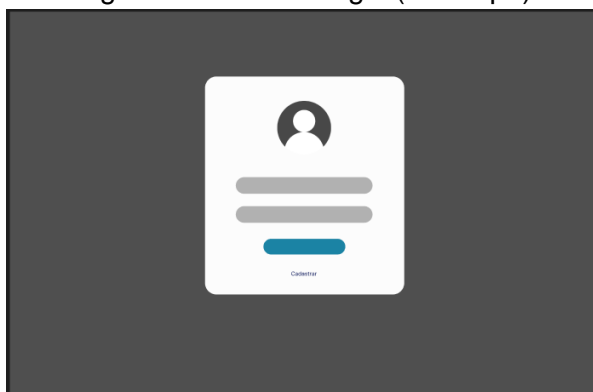
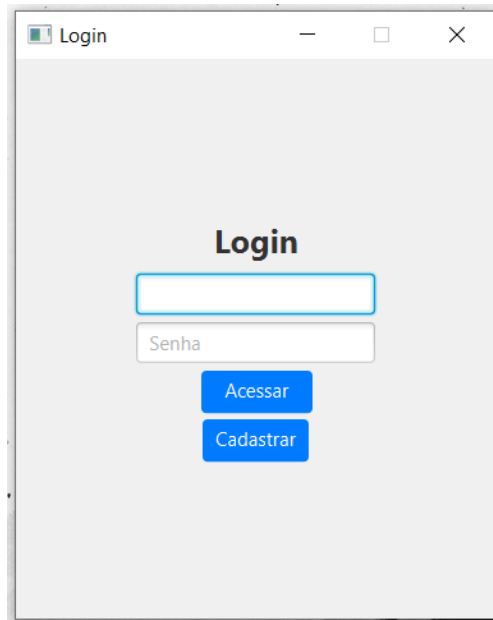


Figura 4 - Tela Login

A mockup of a login window titled "Login". It features a light gray background. In the center, the word "Login" is displayed in bold. Below it are two input fields: the first is empty, and the second is labeled "Senha". Under the input fields are two blue buttons: "Acessar" and "Cadastrar". The window has standard OS window controls (minimize, maximize, close) in the top right corner.

Tela de Cadastro:

Nesta tela, os usuários podem criar novas contas no sistema. Campos como nome, e-mail, senha, data de nascimento e telefone geralmente estão presentes.

A inclusão de validações nos campos, botão de confirmação de cadastro e links para retornar à tela de login são elementos importantes.

Figura 5 - Tela de Cadastro(Protótipo)

A mockup of a registration form titled "CADASTRO". The form is a white rounded rectangle centered on a dark gray background. It contains several input fields: "Nome", "Email", "Senha", "Confirmar senha", "data nascimento", and "telefone". At the bottom of the form is a blue button labeled "Salvar".

Figura 6 - Tela de Cadastro



A mockup of a registration screen titled "Cadastro". It features a light gray background with a white form area. The form contains the following fields: a text input field, an "E-mail" field, a "Senha" field, a "Confirmar Senha" field, a "Data de Nascimento" field with a calendar icon, and a "Telefone" field. A green "Cadastrar" button is positioned below the form. The window has a title bar with the text "Cadastro" and standard minimize, maximize, and close buttons.

Tela de Conversa:

Essa tela é central para a interação social na plataforma. Geralmente, ela exibe a lista de amigos, permitindo a seleção de um amigo para iniciar uma conversa. Componentes como listas (ListView), caixa de mensagem (TextField), e botões de envio são comuns nesta tela.

Figura 7 - Tela de Conversa (Protótipo)

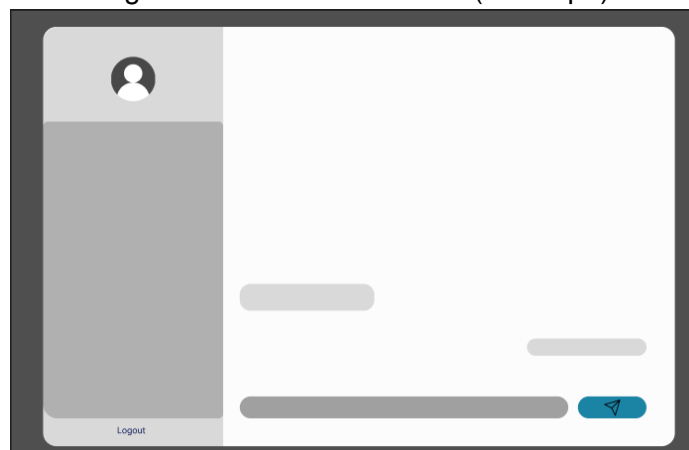
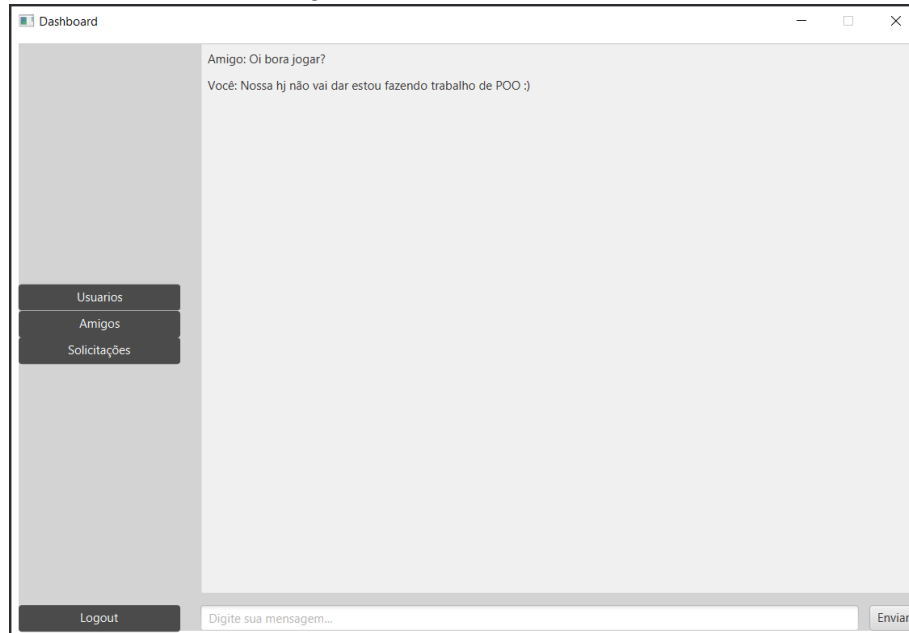


Figura 8 - Tela de Conversa



#### Tela de Solicitação de Amizade:

Esta tela é dedicada ao gerenciamento de solicitações de amizade. Ela exibe as solicitações pendentes e fornece opções para aceitar ou recusar. Pode incluir botões de aceitar/recusar, mensagens de solicitação e uma lista das solicitações pendentes.

Figura 9 - Tela de Solicitação (Protótipo)

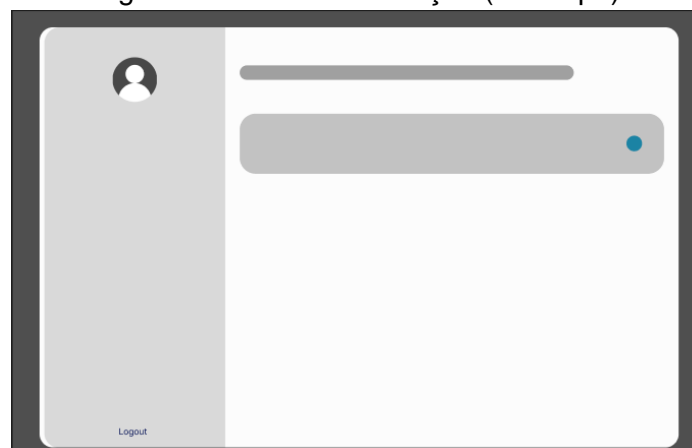
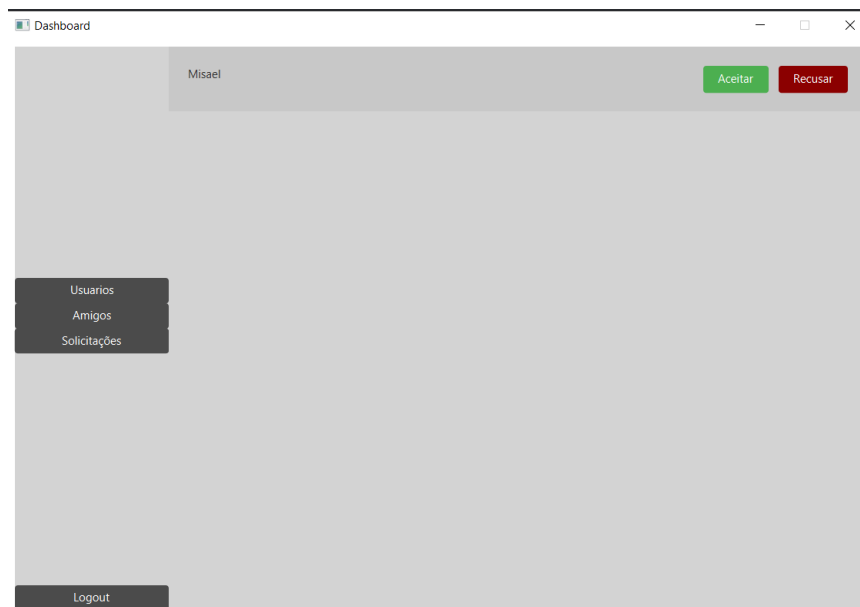


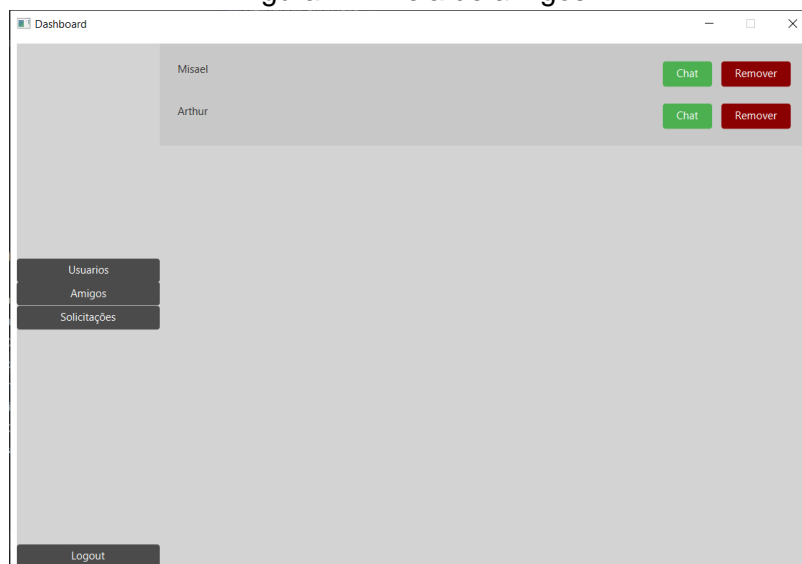
Figura 10 - Tela de Solicitação



### Tela de Amigos:

Inicialmente essa tela não foi planejada no protótipo, porém, durante o desenvolvimento foi adicionada com a intenção de diferenciar os Amigos já adicionados, daqueles que estavam esperando resposta da solicitação. Ela tem a função de remover amigos ou iniciar uma conversa.

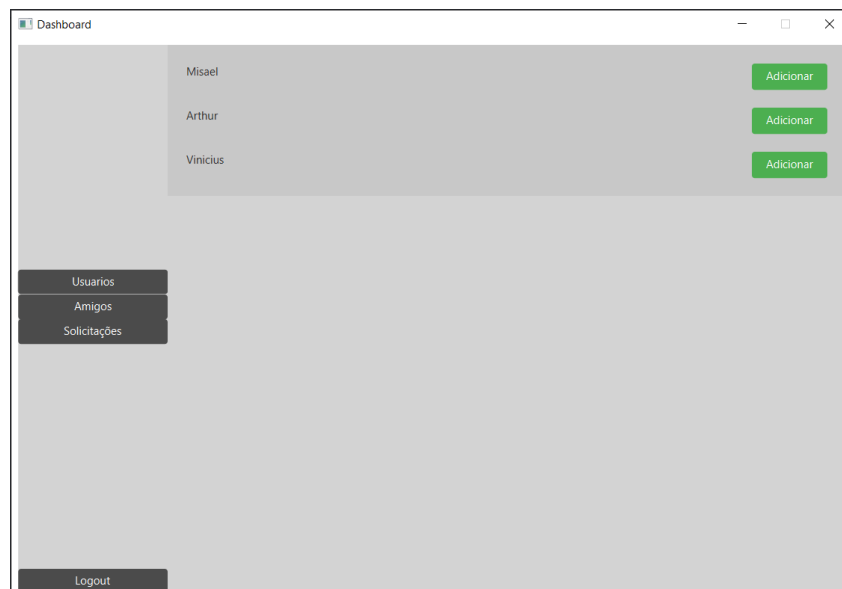
Figura 11 - Tela de amigos



### Tela de Usuários da Rede Social:

Nesta tela é feito a consulta de usuários da rede social, além da consulta é mostrado uma lista de usuários existentes como recomendação, com a opção de adicionar o usuário como amigo.

Figura 12 - Tela de Consulta de usuário



### Implementação JavaFX:

Para traduzir esses protótipos para a implementação em JavaFX, foi necessário criar controladores (Controller) para cada tela, definindo a lógica por trás da interação do usuário. A comunicação eficaz entre as diferentes telas e seus controladores é essencial para a experiência fluida do usuário.

A utilização de FXML para a definição da interface e o controle da aplicação através de eventos (como cliques em botões) foram práticas essenciais no projeto com o JavaFX. Além disso, a organização da interface em componentes reutilizáveis, como CSS para estilos e layouts específicos, contribuiu para um código limpo e modular.

Ao considerar a implementação das funcionalidades específicas de uma rede social, é necessário integrar as operações do banco de dados, como a manipulação de mensagens, solicitações de amizade e informações do usuário.

Foram utilizadas as seguintes bibliotecas:

```
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.layout.StackPane;
import javafx.stage.Modality;
import javafx.stage.Stage;
import javafx.stage.StageStyle;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
```

```
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
```

```
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.VBox?>
```

Concluir o Mini Simulador de Rede Social foi desafiador para nossa equipe. Enfrentamos duas dificuldades principais: fazer as mensagens aparecerem corretamente para ambos os usuários no chat e usar o JavaFX para a interface, algo novo para todos nós.

A lógica das mensagens exigiu muitos testes e ajustes para garantir que tudo funcionasse bem. Foi um processo de tentativa e erro, mas ao final, conseguimos uma solução eficiente.

A utilização do JavaFX também foi um desafio, já que ninguém na equipe tinha experiência com essa ferramenta. Foi essencial pesquisar bastante e realizar diversos testes para aprender e aplicar os conceitos necessários.

Em contrapartida, a familiaridade prévia com o banco de dados permitiu uma abordagem mais eficiente na gestão e recuperação de informações dos usuários, amigos e solicitações. Essa parte do projeto demonstrou como a experiência prévia em determinadas ferramentas pode facilitar a implementação de funcionalidades específicas.

Superar essas dificuldades não apenas nos ensinou muito sobre programação e interfaces gráficas, mas também ressaltou a importância de trabalhar juntos e persistir diante de obstáculos técnicos. A satisfação de ver a interface funcionando como esperado tornou todo o esforço valioso. Essa experiência certamente deixou uma marca positiva em nosso desenvolvimento técnico e na forma como enfrentamos desafios complexos.