



CURSO DE ANALISE E DESENVOLVIMENTO DE SISTEMAS

# **RELATÓRIO FINAL CONECTA NEWTON**

## **MINI SIMULADOR DE REDE SOCIAL**

Camila

Deymerson Patrick da Costa Moreira: 12113087

Giovane Luiz dos Santos: 12114095

Guilherme Rosestolato dos Santos: 12113898

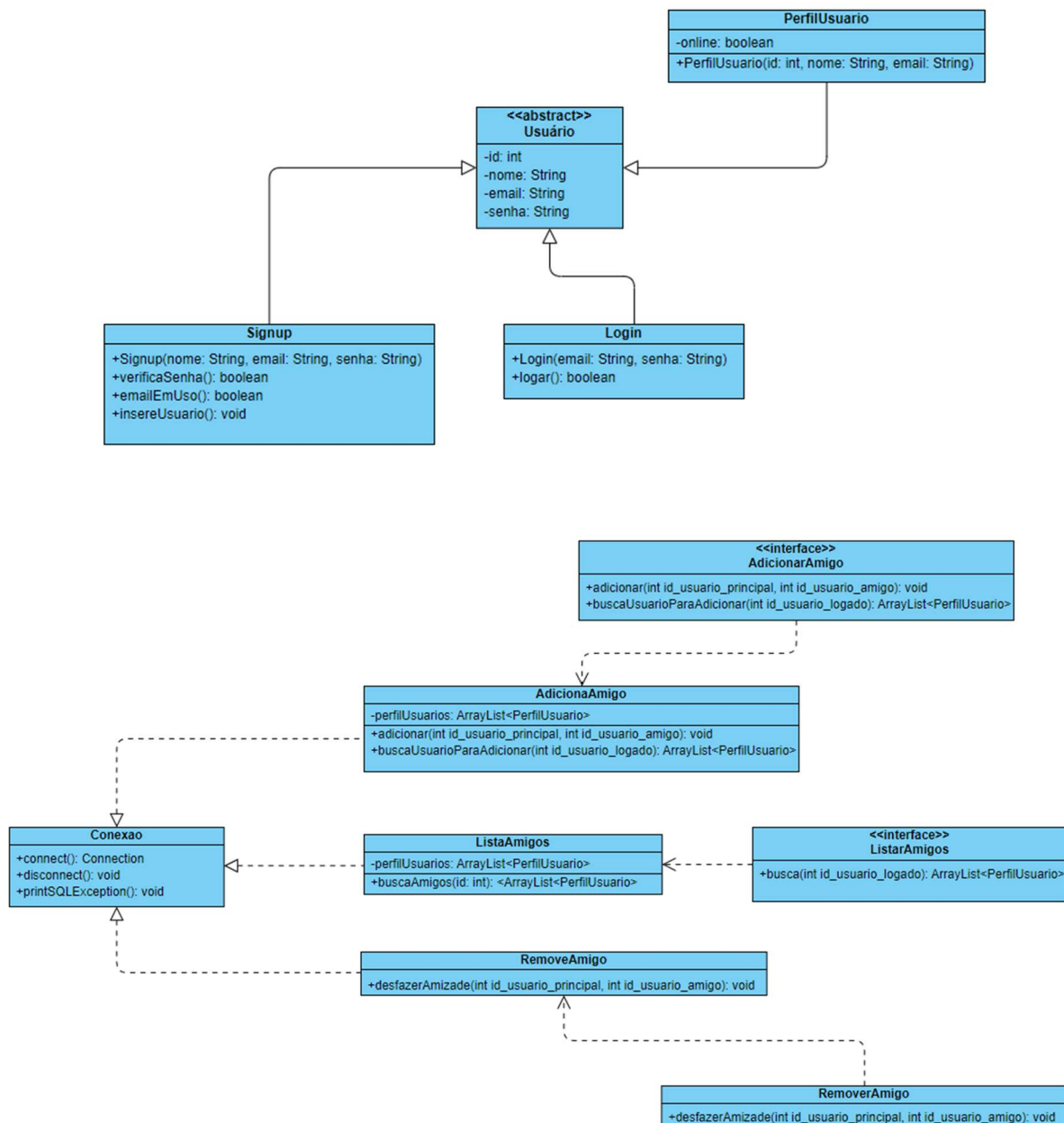
Lucas Antônio Souza Jacomete: 12117209

Belo Horizonte  
2022

## Introdução

Para implementar o mini simulador de rede social conforme o problema, analisamos o funcionamento básico de uma rede social, inspirando-se em aplicativos já existentes como Twitter e Instagram. No levantamento que a equipe fez, acreditou-se que uma solução para a rede social apresentada no problema necessita de uma opção para cadastro dos usuários (Signup), uma opção para entrar no sistema (Login), e um menu de navegação para navegar entre as opções de adicionar amigo, listar as amizades do usuário logado e desfazer uma amizade do mesmo. Demos o nome para esse mini simulador de **Conecta Newton**.

Após análise e levantamento de requisitos, apontou-se os seguintes diagramas de classes.



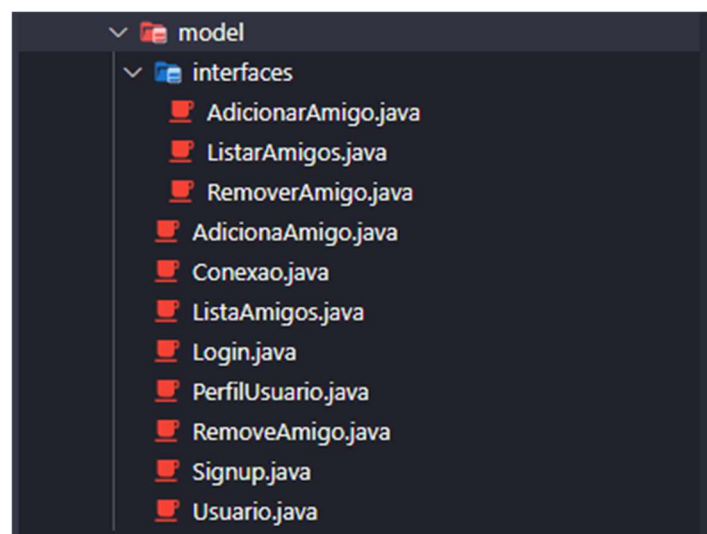
## Desenvolvimento

Para o desenvolvimento, aplicou-se na prática o aprendizado adquirido em sala de aula, utilizando classes com seus atributos e métodos, e relacionamento entre tabelas com heranças e interfaces. Para salvar e manipular estes dados utilizou-se o banco de dados PostgreSQL, com uma instância na nuvem na plataforma Elephant (*saiba mais em [ElephantSQL - PostgreSQL as a Service](#)*). Com o banco de dados temos a possibilidade de salvar inúmeros usuários e praticidade para manipular os dados (incluir, editar e excluir). A hospedagem em nuvem traz vantagens de conseguir rodar o aplicativo em computadores diferentes, sem a necessidade de criar um banco de dados local.

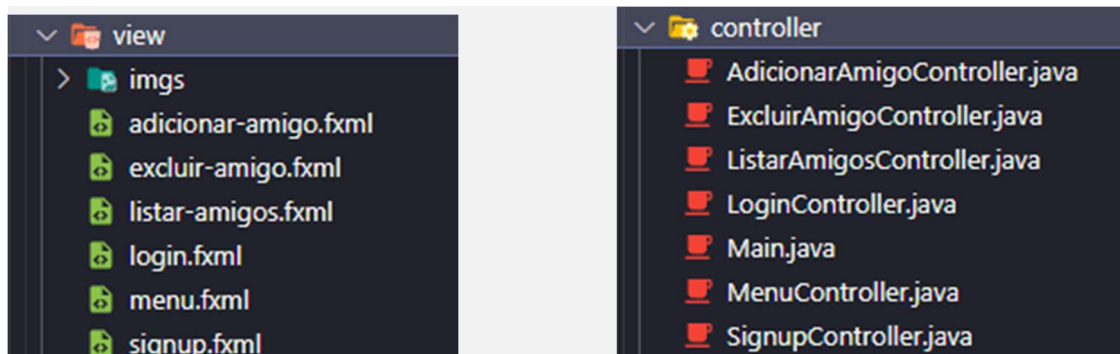
Para a interface gráfica, utilizou-se a ferramenta SceneBuilder, porém seus arquivos gerados são em formato .fxml. Para que o Java trabalhe com esse formato, é necessário a biblioteca JavaFX (*saiba mais em [JavaFX \(openjfx.io\)](#)*).

Utilizou-se também, um modelo de arquitetura de software conhecida como MVC (Model - View - Controller). Essa arquitetura possibilita ter uma boa organização e separação de componentes dentro do projeto Java.

Na camada Model, se encontram as classes do sistema e as classes de banco de dados.



Na camada View, se encontram as telas e imagens do sistema e na camada Controller, se encontram as classes que manipulam os objetos da tela e criam os objetos das classes contidas na camada Model.



Na camada Model, observa-se a existência de uma classe chamada Usuário.

```
public abstract class Usuario {  
    private int id;  
    private String nome;  
    private String email;  
    private String senha;  
  
    public Usuario(String nome, String email, String senha) {  
        this.nome = nome;  
        this.email = email;  
        this.senha = senha;  
    }  
  
    public Usuario(String email, String senha) {  
        this.email = email;  
        this.senha = senha;  
    }  
  
    public Usuario(int id, String nome, String email) {  
        this.id = id;  
        this.nome = nome;  
        this.email = email;  
    }  
}
```

Essa classe ficou definida como abstrata, pois ela por si só contém atributos semelhantes de outras três classes do sistema: **Signup**, **Login** e **PerfilUsuario**. A classe possui um construtor para cada classe citada anteriormente (Login, Signup e PerfilUsuario).

A classe **Signup** é responsável por criar um novo usuário no sistema. Para isso, ela recebe em seu construtor os atributos nome, e-mail e senha. A classe Signup também possui o método estático **emailEmUso**, que verifica se já existe algum cadastro com o e-mail informado. O mesmo foi criado como estático para não ter a necessidade de instanciar uma nova classe, uma vez que se existir uma conta com o e-mail informado, a conta não é

criada. Outro método existente é o **insereUsuario**, responsável por inserir o usuário no banco de dados.

```
public class Signup extends Usuario {  
  
    public Signup(String nome, String email, String senha) {  
        super(nome, email, senha);  
    }  
  
    public static boolean emailEmUso(String email) { ...  
  
    public void insereUsuario() throws SQLException { ...  
  
}
```

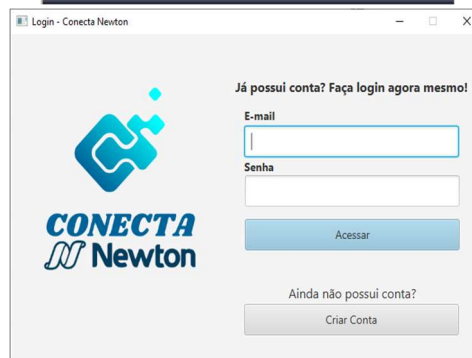


A interface de usuário para a criação de uma conta no Conecta Newton. No topo, há o título "Crie sua conta". À esquerda, o logo "CONECTA Newton" é exibido. À direita, há quatro campos de entrada: "Nome", "E-mail", "Senha" e "Confirmar senha". Abaixo dos campos, há dois botões: "Criar Conta" (em azul) e "Retonar para o Login" (em cinza).

*Tela de cadastro Conecta Newton (mini simulador)*

Outra classe importante para o funcionamento do mini simulador é a classe **Login**, responsável por validar se o usuário está cadastrado e se a senha informada pelo usuário ao logar, está de acordo com a senha que está salva no banco de dados. A mesma recebe como parâmetro em seu construtor o e-mail e senha. Além de possuir um método cujo nome é **logar**, responsável por fazer as validações acima citadas e retornar verdadeiro em caso de sucesso nas validações.

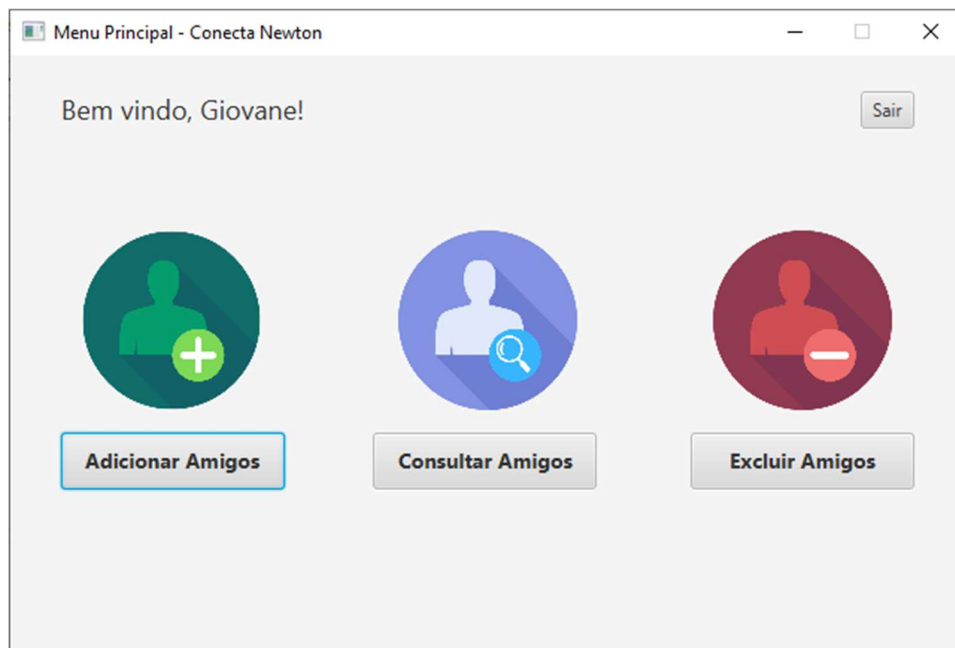
```
public class Login extends Usuario {  
  
    public Login(String email, String senha) {  
        super(email, senha);  
    }  
  
    public boolean login() { ...  
  
}
```



A interface de usuário para o login no Conecta Newton. No topo, há o título "Já possui conta? Faça login agora mesmo!". À esquerda, o logo "CONECTA Newton" é exibido. À direita, há dois campos de entrada: "E-mail" e "Senha". Abaixo dos campos, há dois botões: "Acessar" (em azul) e "Criar Conta" (em cinza).

*Tela de Login Conecta Newton (mini simulador)*

Ao clicar no botão de “Acessar”, o usuário é redirecionado ao Menu Principal, responsável por fazer o controle das opção de Adicionar Amigos, Consultar Amigos e Excluir Amigos



Para ter acesso aos dados dos usuários, criou-se a classe **PerfilUsuario**. Essa classe é responsável por manipular os perfis de usuários cadastrados no sistema, sem ter acesso a senha dos usuários.

```
public class PerfilUsuario extends Usuario {
    private boolean online;

    public PerfilUsuario(int id, String nome, String email) {
        super(id, nome, email);
    }

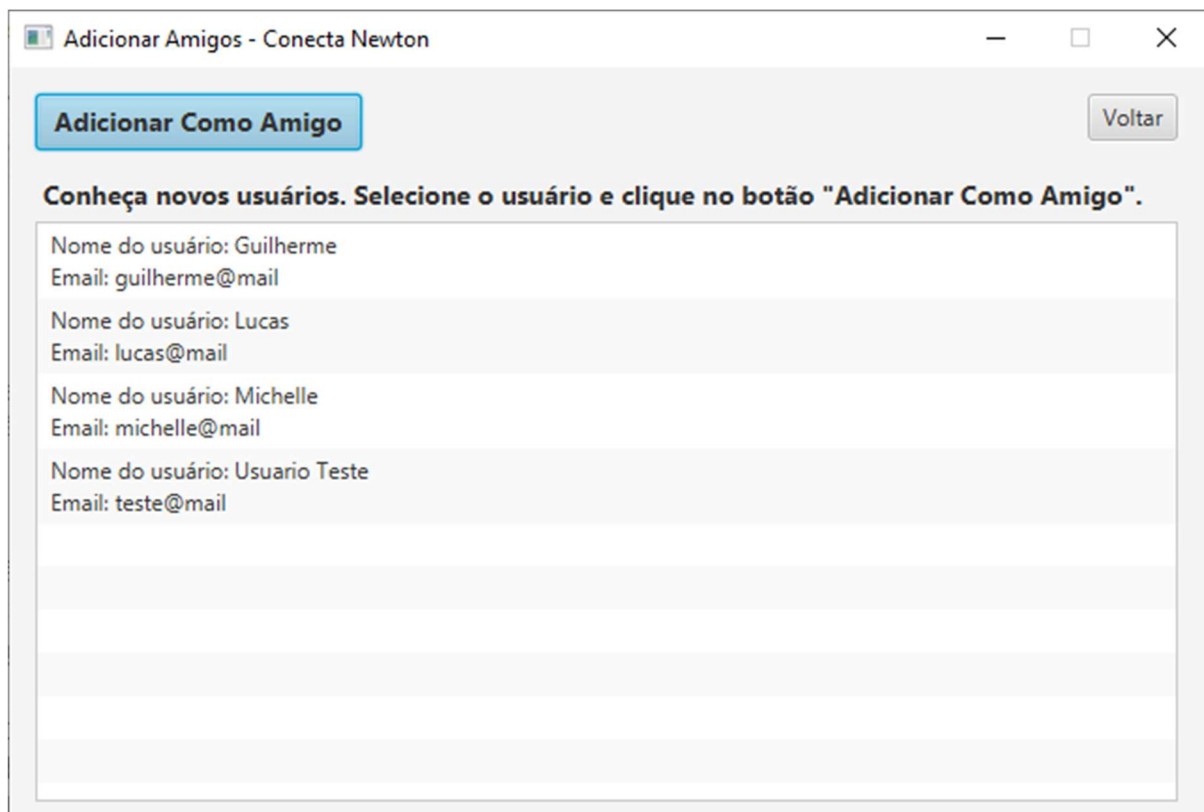
    public boolean isOnline() {
        return online;
    }

    public void setOnline(boolean online) {
        this.online = online;
    }
}
```

Ela possui um atributo online e recebe em seu construtor os atributos de id, nome e e-mail. Ao carregar uma lista de perfis, essa classe se faz extremamente útil para retorno das informações necessárias (id, nome e e-mail apenas).

Ao selecionar o menu de opções **Adicionar Amigos**, o sistema aciona a classe controladora **AdicionarAmigosController**, onde tem-se uma listagem com todos os usuários que podem ser amigos do atual. Essa listagem é carregada a partir do método

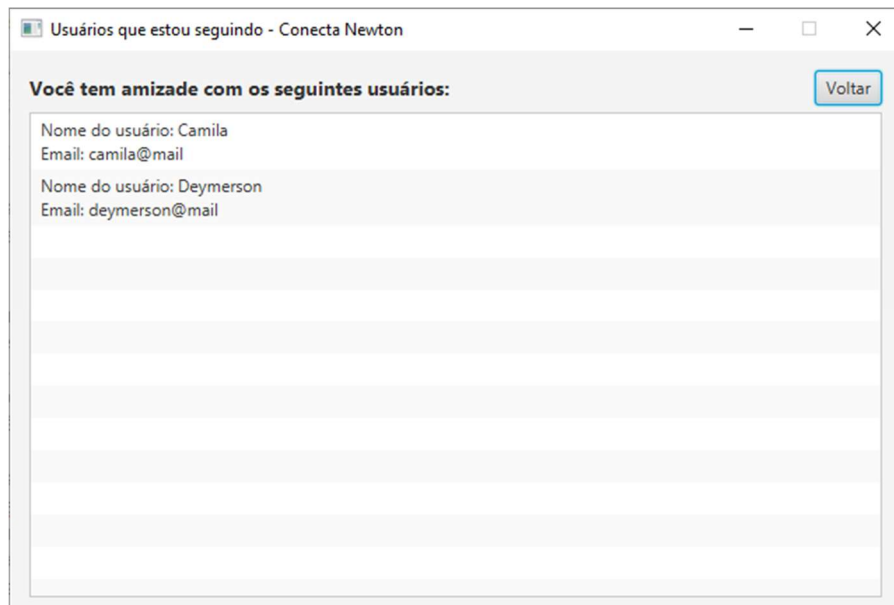
**buscaUsuarioParaAdicionar**, implementado na classe **AdicionaAmigo** através de uma Interface. Esse método retorna uma lista de perfis para adicionar como amigo (ArrayList<PerfilUsuario>).



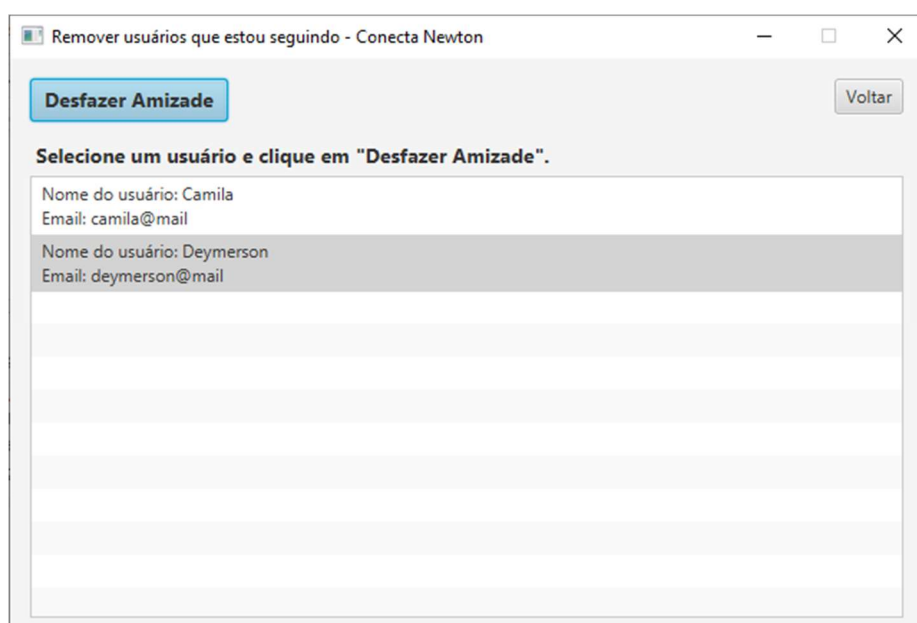
Ao selecionar um usuário e clicar no botão de “Adicionar Como Amigo”, o sistema registra a amizade no banco de dados e retorna a listagem atualizada com os usuários que ainda não tem amizade com o atual.

No início do projeto, utilizou-se a abordagem como nas redes sociais mencionadas anteriormente, onde temos o relacionamento entre usuários como “seguidor” e “seguindo”. Mas após análise da Professora Michelle e reunião com demais participantes do grupo, optou-se pela simplicidade e com isso, implementou-se no sistema a amizade sendo direta usuário com usuário.

Ao selecionar o menu de opções **Consultar Amigos**, o sistema aciona a classe controladora **ListarAmigosController**, onde tem-se uma listagem com todos os usuários que são amigos do usuário atual. Essa listagem é carregada a partir do método **busca**, implementado na classe **ListarAmigos** através de uma Interface. Esse método retorna uma lista de perfis de usuários amigos (`ArrayList<PerfilUsuario>`).



Ao selecionar o menu de opções **Excluir Amigos**, o sistema aciona a classe controladora **ExcluirAmigoController**, onde tem-se uma listagem com todos os usuários que são amigos do usuário atual. Essa listagem é carregada a partir do método **buscaAmigos**, que pertence a classe **ListaAmigos**. Ao selecionar um usuário clicar no botão de “Desfazer Amizade”, o sistema por meio do método **desfazerAmizade** que é implementado na classe **RemoveAmigo**, remove a amizade no banco de dados e retorna a listagem atualizada com os usuários que ainda são amigos do atual.





## Conclusão

Ao final do projeto, percebeu-se o quão importante é a programação orientada a objeto para um desenvolvimento de software, seja ela simples como o mini simulador de rede sociais, apresentada no trabalho ou sistemas mais robustos. No início teve-se a dificuldade de integrar o sistema com os layouts desenhados no ScenneBuilder. Mas após estudos de documentação e materiais em vídeo no YouTube, nosso projeto chegou ao fim cumprindo com o objetivo principal do trabalho. Infelizmente não foi possível implementar o envio de mensagens em nosso programa, mas ficará para futuras implementações se possível.