Trabalho final Rede Social – NewtWork

Ana Raquel Mendes Maia – 12118899

Gabriel Seleguini – 11320007

Luan Maxwell de Oliveira Santos – 12118287

Marcos Paulo Dos Guimarães Ribeiro – 12118405

Sara de Souza Viana - 12118225

Introdução: para nosso projeto, foram utilizadas as IDEs “Eclipse” e “IntelliJ” pelos membros do trabalho para realizar a programação do sistema. Além disso, realizamos a conexão ao banco de dados utilizando o PostgreSQL. Em nosso Banco de Dados, realizamos a criação de algumas tabelas para desenvolvermos alguns métodos e relações futuras, porém, fundamentalmente nosso sistema é composto de três tabelas principais: tbusuario, tbamizade e tbmensagem.

O sistema consta com seis funções principais:

- Cadastro
- Login
- Adicionar amigos
- Excluir amigos
- Enviar mensagens
- Listar conversas

Desenvolvimento: nosso projeto conta com 11 classes, sendo 1 classe para exceções personalizadas, 1 Main com o front end da interface gráfica, 8 classes que são responsáveis por encaminhar os dados ao banco e retornar informações de lá e 1 classe “Usuario” que é basicamente a alma do nosso projeto.

A classe RedeSocialGUI é responsável por interagir com o usuário, puxando as lógicas existentes nas demais classes do sistema e apresentando um menu amigável para melhorar a experiência do usuário. Todos os ícones existentes nela, remetem aos métodos criados nas outras classes.

A classe DomainException seria utilizada para lançar nossas exceções personalizadas para melhorar a experiência e imersão do usuário final.

Dentro do nosso pacote conexaobd há 8 classes para encaminhar os dados ao banco.

A classe **AdicionarAmigo** é responsável por receber o método adicionarAmigo() da classe Usuario e adicionar a relação de amizade no banco na tabela **tbamizade**.

A classe **CadastrarUsuario** é responsável por receber os dados de um usuário através do método cadastrarAmigo() da classe Usuário e adicioná – lo na tabela **tbusuario**.

A classe **EnviarMensagem** é responsável por receber o método enviarMensagem() da classe Usuario e inserir a mensagem na tabela **tbmensagem**.

A classe **ExcluirAmigo** é responsável por receber o método e excluirAmigo() da classe Usuario e remover a linha da tabela **tbamizade**.

A classe **ListarAmigos** é responsável por receber o método listarAmigos() da classe Usuario e retornar uma query do banco de dados na tabela **tbamizade** com todos os amigos do usuário logado.

A classe **ListarMensagens** é responsável por receber o método listarMensagens() da classe Usuario e retornar uma query do banco de dados na tabela **tbmensagem** contendo todas as mensagens trocadas com um determinado amigo do usuário logado.

A classe **LogarUsuario** é responsável por receber o método logarUsuario() da classe Usuario e verificar se o mesmo existe na tabela **tbusuario**. Caso exista, retorna **true** e permite o login.

A classe **ConexaoPostgre** é responsável estabelecer a conexão com o banco de dados RedeSocial do PostgreSQL. As demais classes do pacote conexaobd recebem as informações dessa classe através da herança. Além disso, nessa classe e em suas filhas está presente também algumas lógicas que validam se um usuário existe no banco, se o e-mail já foi preenchido, nome de usuário em uso, se um amigo já foi adicionado e etc.

A classe **Usuario** é responsável por boa parte da lógica do projeto. Podemos dizer que ela é a “mãe de tudo”, pois sem ela, nenhuma outra classe conseguiria existir. Por conta disso, nessa classe vemos uma relação de **composição** com as demais classes de conexão ao banco. Nela são criados nossos principais atributos: nome, e-mail, senha, naturalidade, nascimento e gênero. Há os Construtores, Getters e Setters de cada atributo. Para facilitar a compreensão, os métodos dessa classe receberam os mesmos nomes que as classes do pacote conexaobd.

Cada um dos métodos é responsável por receber os dados fornecidos na Main RedeSocialGUI e processar as informações. São feitos todos os testes para verificar que os dados foram digitados corretamente e exigir que o usuário não digite dados inválidos, cometendo assim erros. Cada um dos métodos aponta para as classes com seus respectivos nomes, onde elas são responsáveis por encaminhar ou retornar informações do banco, através das funções **executeQuery()** e **executeUpdate()**.

Banco de dados: feito através do PostgreSQL. O Banco **RedeSocial** foi adicionado ao GitHub juntamente com o projeto.

Interface Gráfica: foram utilizados componentes de Java Swing e principalmente na classe RedeSocialGUI.

Conclusão: foi um trabalho bem divertido de realizar. Pudemos misturar boa parte de nossos aprendizados e coloca – los em prática. A programação totalmente orientada a objetos no decorrer do nosso desenvolvimento e utilização de metodologias aprendidas em sala de aula, como por exemplo, o uso de **Herança** na classe ConexaoPostgre para suas filhas do pacote conexaobd, o uso de **Composição / Agregação** com a instância de objetos das classes de conexão dentro da classe usuário, pois, uma classe precisa da outra

para existir. Optamos por dividir o trabalho em 3 partes entre os membros do grupo: banco de dados, back-end e front-end. Contudo, todos os membros tiveram participações e contribuíram com ideias e testes nas partes dos demais membros.

Sobre as dificuldades, no front-end tivemos um pouco de dificuldade para estabelecer a lógica para o sistema mudar a tela depois de logar. Fizemos algumas tentativas e não estava muito certo. Tentamos fazer com dois painéis: um painel de login e outro pós login para gerar uma dinâmica melhor. Tivemos um pouco de dificuldade para fazer com que as informações presentes no painel inicial de login sumissem após a efetivação do login, e consequentemente, passasse a exibir as informações do sistema após o login (botão de enviar mensagens, adicionar amigos e conversas, por exemplo). Porém essa ideia deu certo depois de algumas tentativas.

No back – end, tivemos um erro na lógica de listar os amigos. Quando adicionamos um amigo no banco, o usuário logado é o usuário de id1, porém se ele for adicionado por outra pessoa, será o usuário de id2. Quando listávamos os amigos, apenas víamos os amigos que nós mesmos adicionamos, e não quando também fomos adicionados. Foram diversos testes e pesquisas na Web para identificar o comando SQL que nos auxiliaria na lógica para listar os amigos, independentemente de quem adicionou quem.

Sobre erros, realizamos a tratativa de diversos pontos. Antes nosso sistema permitia adicionar a si mesmo como amigo, digitar um e-mail já em uso, aceitar um gênero diferente de M ou F, por exemplo. Além disso, não havíamos notado, mas a lógica para listar as conversas entre dois amigos estava invertida, exibindo como se o usuário 1 tivesse enviado a mensagem do usuário 2. Todos os problemas citados foram identificados ao longo de testes entre os membros do grupo e corrigimos os mesmos.

A última dificuldade, foi a maior: corrida contra o tempo. Tivemos um prazo de um mês para realização do projeto, porém, boa parte dos conhecimentos (como por exemplo a conexão com o banco de dados) que são primordiais para o funcionamento do sistema foram repassados aos poucos, com pouco menos que 15 dias para a entrega. O nosso front-end dependia do back-end para ficar pronto, e sem esses conhecimentos, a equipe de front-end não conseguia realizar o design do sistema. Por isso, foi necessário agilizar a equipe no tempo e fazer somente o essencial. Tivemos diversas ideias extras que queríamos adicionar, como por exemplo criar grupos de conversas, posts, curtidas e etc.

Infelizmente não pudemos acrescentar tudo que queríamos, porém, foi um trabalho bem divertido e gostamos bastante da proposta. Reunir todos os conteúdos aprendidos foi ótimo, e por isso, agradecemos a oportunidade dada pelos instrutores Michelle Hanne e João Aramuni.

Att;

Ana Raquel, Gabriel Seleguini, Luan Maxwell, Marcos Paulo e Sara Viana.