UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
DEPARTAMENTO DE INFORMÁTICA APLICADA

LEONARDO LEITES
MATHEUS CANDIOTTO ROCHA
RAFAEL BORGES STEPHANOU
RICARDO SETTON ALENCAR DE CARVALHO
VICTOR HUGO SILVEIRA

ETAPA 2 DO TRABALHO FINAL DA DISCIPLINA DE TÉCNICAS DE CONSTRUÇÃO DE PROGRAMAS (INF01120)

Porto Alegre
2025

LEONARDO LEITES
MATHEUS CANDIOTTO ROCHA
RAFAEL BORGES STEPHANOU
RICARDO SETTON ALENCAR DE CARVALHO
VICTOR HUGO SILVEIRA

ETAPA 2 DO TRABALHO FINAL DA DISCIPLINA DE TÉCNICAS DE CONSTRUÇÃO DE PROGRAMAS (INF01120)

Nesta etapa, vamos definir as funcionalidades requeridas relativas à implementação e teste em relação ao modelo de programa definido nas etapas anteriores a ser desenvolvido na disciplina.

Orientador(a): Prof. Dra. Karina Kohl Silveira

Porto Alegre
2025

Mudanças em relação à etapa anterior

- Não conseguimos implementar o projeto em sua totalidade;
- Todos os nomes dos argumentos, métodos e classes foram traduzidos para a língua inglesa;
- Alguns atributos de identificação (IDs), atributos/métodos estáticos, métodos construtores, *getters* e *setters* que estavam faltando foram criados para várias classes;
- A classe "Creators" foi excluída;
- Alguns métodos que estavam duplicados foram removidos;
- Foram adicionadas novas classes para garantir o encapsulamento.

Implementação

A seguir, constam os detalhes da implementação do nosso sistema:

- **Classe Book:**
    - **Atributos:**
        - **id:** identificador do livro usado em consultas pelo sistema
        - **title**
        - **author**
        - **isbn**
        - **releaseYear**
        - **numPages**
        - **genre**
        - **numBooksCreated (static)** O total de livros armazenados no sistema
    - **Métodos:**
        - **Construtor(sem id em parâmetro):** Cria uma instância de livro pela primeira vez
        - **Construtor(com em parâmetro):** Recria uma instância de livro informando a id
        - **GetI**
        - **SetId**
        - **getTitle**
        - **setTitle**
        - **getAuthor**
        - **setAuthor**

        - **getIsbn**
        - **setIsbn**
        - **getReleaseYear**
        - **setReleaseYear**
        - **getNumPages**
        - **setNumPages**
        - **getGenre**
        - **setGenre**
        - **static getNumBooksCreated**
        - **toString:** Transforma o objeto livro em String na impressão
        - **toCsvLine:** Define o comportamento do objeto livro quando impresso como em CSV

- **<u>Classe BookClub:</u>**
    - **Atributos:**
        - **id:**
        - **name**
        - **participants:** Users que entraram no clube
        - **polls:** Polls a serem realizadas pelo clube
        - **meetings:** Meetings a serem realizadas pelo clube
        - **creator:** User que criou o clube
        - **numBookClubsCreated (static)**

    - **Métodos:**
        - **Construtor (com id em parâmetro)**
        - **Construtor (com id em parâmetro)**
        - **getId**
        - **setId**
        - **getTitle**
        - **setTitle**
        - **getAuthor**
        - **setAuthor**
        - **getIsbn**
        - **setIsbn**

        - **getReleaseYear**
        - **setReleaseYear**
        - **getNumPages**
        - **setNumPages**
        - **getGenre**
        - **setGenre**
        - **getNumBooksCreated (static)**
        - **toString**
        - **toCsvLine**

- **<u>Classe Poll (Abstract):</u>**

    - **Atributos:**
        - **id**
        - **bookClub: clube referente à poll**
        - **question: pergunta exibida aos usuários na poll**
        - **options: opções a serem votadas pelos usuários**
    - **Métodos:**
        - **Construtor (com id em parâmetro)**
        - **Construtor (sem id em parâmetro)**
        - **getId**
        - **getBookClub**
        - **setBookClub**
        - **getQuestion**
        - **getOptions**
        - **getVotes**
        - **getWinner:** Retorna a opção com o maior número de votos
        - **vote:** Insere um voto no índice (parâmetro) referente a uma das opções
        - **getVoteAsCsv:** Informa como escrever os votos em Csv
        - **getType:** Informa o tipo da poll (date ou book)

- **<u>Classe DatePoll (extendsPoll):</u>**
    - **Métodos:**

        - **Construtor(com id como parâmetro)**
        - **Construtor(sem id como parâmetro)**
        - **getType**

- **<u>Classe BookPoll (extendsPoll):</u>**
    - **Métodos:**
        - **Construtor(com id como parâmetro)**
        - **Construtor(sem id como parâmetro)**
        - **getType**

- **<u>Classe Meeting:</u>**
    - **Atributos:**
        - **id**
        - **creator**
        - **participants**
        - **type**
        - **date**
        - **location**
        - **bookClub**
        - **numMeetingsCreated (static)**
    - **Métodos:**
        - **Construtor**
        - **getId**
        - **getCreator**
        - **getBookClub**
        - **getParticipants**
        - **getType**
        - **getLocation**
        - **getNumberMeetingsCreated**

- **Classe User:**
    - **Atributos:**
        - **id**
        - **name**
        - **surname**
        - **email**
        - **cpf**
        - **password**
        - **joinedBookClubs**
        - **createdBookClubs**
        - **numUsersCreated**
    - **Métodos:**
        - **Construtor (com id em parametro)**
        - **Construtor (sem id em parametro)**
        - **getId**
        - **setId**
        - **getName**
        - **setName**
        - **getEmail**
        - **setEmail**
        - **getCpf**
        - **setCpf**
        - **getPassword**
        - **getJoinedBookClubs**
        - **getNumUsersCreated**
        - **toString**
        - **getCreatedBookClubs**

- **Classe BookClubRepository:**
    - **Atributos:**
        - **Path:** Localizacao do arquivo de armazenamento
    - **Métodos:**

        - **loadAll:** Lê todos os clubes do Csv. Recebe UserService para salvar os criadores e participantes do clube.
        - **saveAll:** Salva todos os clubes de uma lista de livros no Csv.

- **<u>Classe UserRepository:</u>**
    - **Atributos:**
        - **Path:** Localizacao do arquivo de armazenamento
    - **Métodos:**
        - **loadAll:** Lê todos os users do Csv.
        - **saveAll**

- **<u>Classe BookClubService:</u>**
    - **Atributos:**
        - **repo (bookClubRepository)**
    - **Métodos:**
        - **Construtor:** Recebe userService
        - **RebuildCreators:** Constroi array de criadores de clubes através dos índices armazenados na lista dos clubes
        - **clubNameExists:** Checa se o nome do clube é único no siste,a
        - **createClub:** Insere e salva um clube especificado no sistema
        - **leaveClub:** Remove um usuário especificado de um clube especificado
- **<u>Classe BookService:</u>**
    - **Atributos:**
        - **ArrayList<Book>**
        - **File**
    - **Métodos:**
        - **loadBooks**
        - **findBookByIsbn**
        - **registerBook**
        - **printBooks**
        - **getBooks**
        - **deleteBook**
        - **saveAllBooks**

- **Classe MeetinService:**
    - **Atributos:**
        - **meetings:** Lista das meetings do sistema
        - **FILE_NAME**
        - **sdf:** Formato da data para uma meeting
    - **Métodos:**
        - **getMeetings**
        - **addMeeting**
        - **loadMeetings**
        - **saveMeetings**

- **Classe PollService:**
    - **Atributos:**
        - **polls**
        - **FILE_NAME**
    - **Métodos:**
        - **getPolls**
        - **CreatePollForClub:** cria uma poll, estabelecendo uma relação de armazenamento bilateral com o clube especificado
        - **loadPolls**
        - **savePolls**
        -
- **Classe UserService:**
    - **Atributos:**
        - **repo(UserRepository)**
        - **ArrayList<Users>**
    - **Métodos:**
        - **Construtor**

        - **registerUser**
        - **reloadUsers**
        - **findUserByEmail**
        - **getUser**
        - **fieldsAreEmpty**
        - **syncIdCounter**
        - **printUsers**
        - **alterPassword**
        - **findById**

- **<u>Classe AppSystem:</u>**
    - **Atributos:**
        - **regScreen**
        - **userService**
        - **bookService**
        - **pollService**
        - **meetingService**
    - **Métodos:**
        - **Construtor:** Recebe RegistrationScreen. Inicializa os serviços do sistema
        - **InitRegistrationController:** Configura os eventos na tela de registro (RegScreen)

- **<u>Classe Main:</u>**
    - **Métodos:**
        - **Construtor:** Inicia o programa, aplica fonte personalizada e ativa o controlador principal
        - **ApplyCustomFont:** Carrega fonte externa, registra no sistema e aplica na interface

        - **SetUIFont:** Substitui as fontes padrão de Swing pelas personalizadas

- **Classe ResetPasswordController:**
    - **Atributos:**
        - **resetPasswordScreen**
        - **resetPasswordScreen1**
    - **Metodos:**
        - **initController**
        - **handleVerify**
        - **handleCadastrar**

- **Classe AppController:**
    - **Métodos:**
        - **Start:** Inicializa o aplicativo exibindo a tela de loading
        - **showLoadingScreen:** Mostra a tela de loading por 2 segundos e, ao terminar, abre a tela de login
        - **showLoginScreen:** Cria a tela de login, conecta o controlador e mostra pro usuário

- **Classe HomeController**
    - **Atributos:**
        - **homeScreen**
        - **feedScreen**
    - **Métodos:**
        - **Construtor**
        - **openMyGroups:** Cria a tela dos grupos dos quais o usuário é dono e a torna visível
        - **showMyGroups:** Cria a tela dos grupos dos quais o usuário participa e a torna visível

        - **enterGroup:** Cria a tela dos grupos dos quais o usuário pode participar e a torna visível

- **Classe LoginController**
    - **Atributos:**
        - **LoginScreen**
        - **userService**
    - **Métodos:**
        - **Construtor**
        - **initController**
        - **handleLogin**
        - **handleRegister**
        - **handleReset**
        - **authenticateUser**

- **GroupCreationController**
    - **Atributos:**
        - **view**
        - **loggedUser**
        - **clubService**
    - **Métodos:**
        - **Construtor**
        - **initController**
        - **criarGrupo**
        - **voltar**

Classes Puramente de Interface: Como fogem à proposta do trabalho, serão apenas mencionadas.

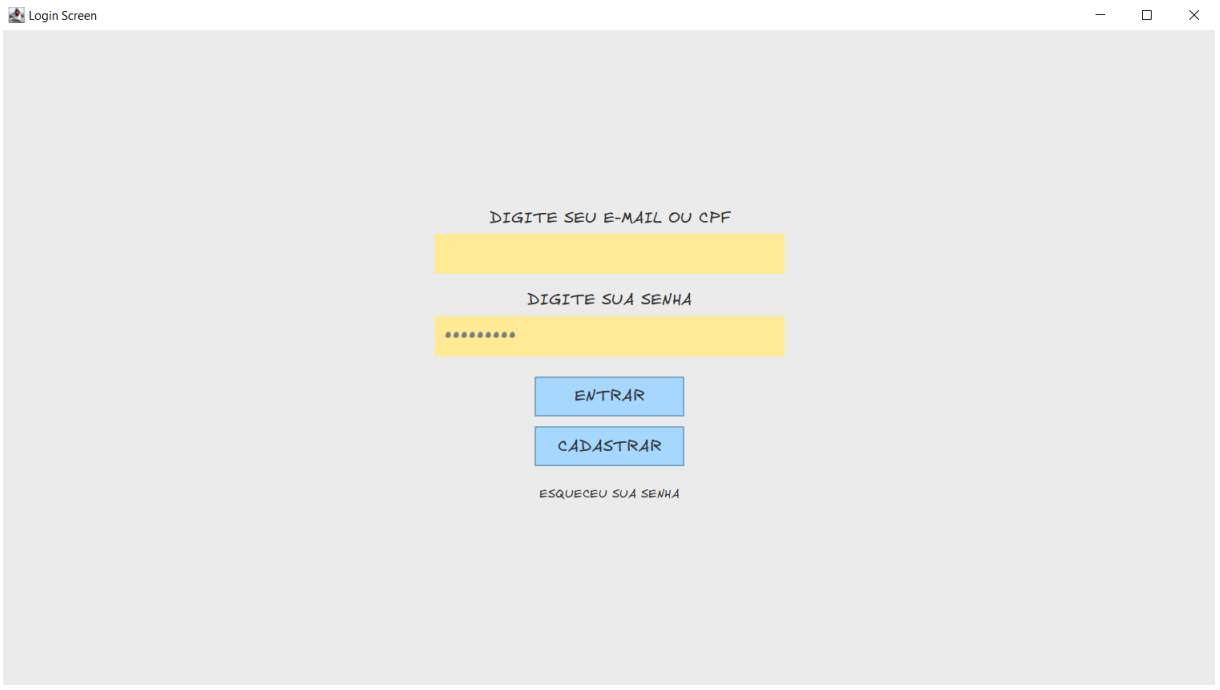
AddBookScreen, DeleteGroupScreen, ExitScreen, FeedScreen, GroupCreationScreen, GroupManagementScreen, GroupUserScreen.java, GroupVoteScreen,

GroupVotingCloseScreen, HomeScreen, LoadingScreen. LoginScreen, NewVoteScreen, ParticipateScreen, RegistrationScreen, ResetPasswordScreen, ResetPasswordScreen1 BackButtonComponent, ButtonComponent, CardComponent, DatePickercomponent, GroupVotingInfoComponent, InfoBarComponent, PasswordField, TextField, VoteComponent.

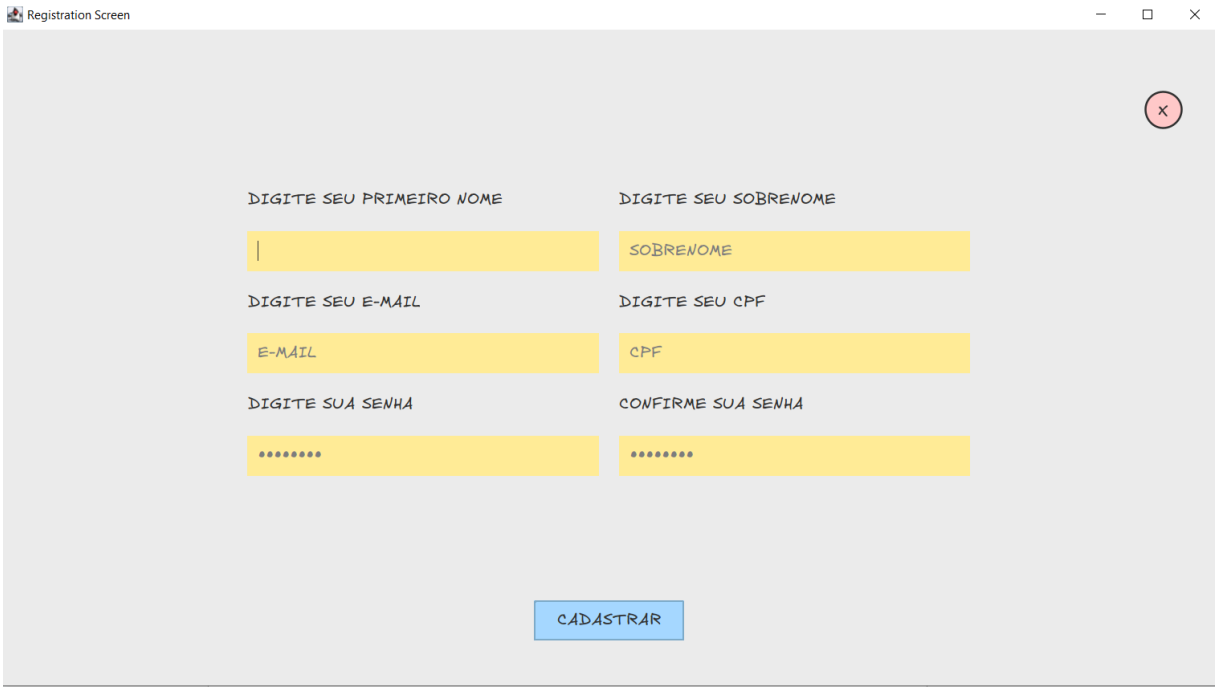
Executável

A interface proposta na etapa anterior foi totalmente criada dentro do projeto. Contudo, existem partes que atualmente não se encontram funcionais.

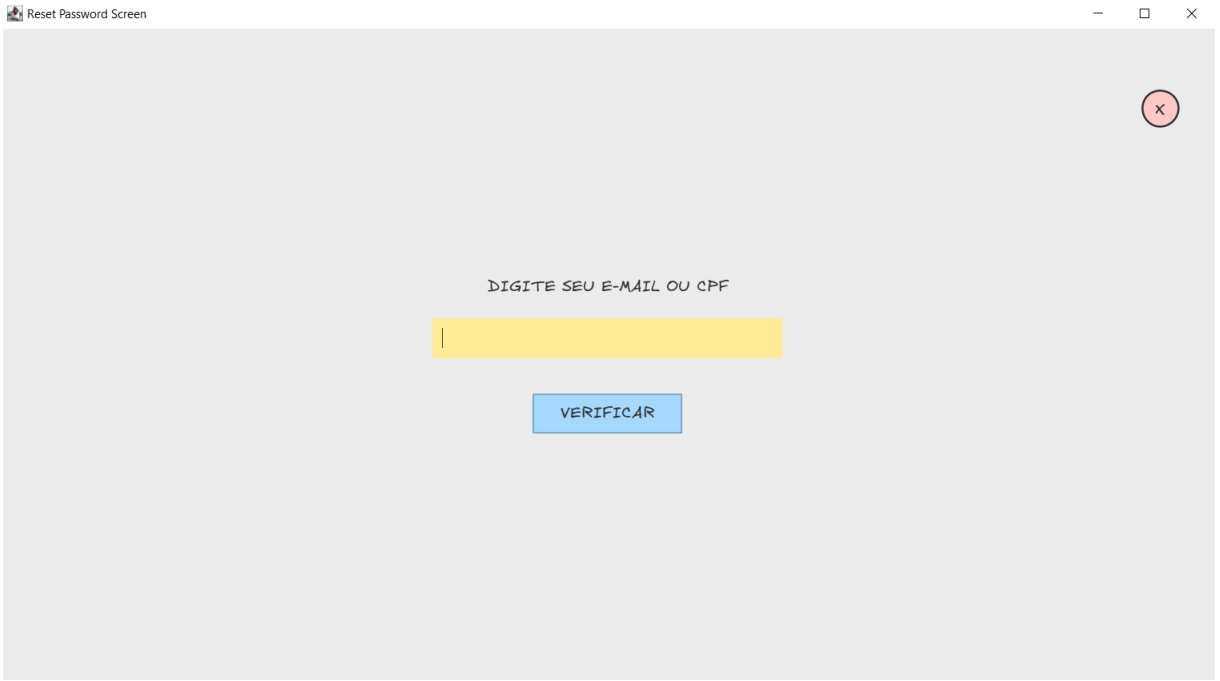
Login Screen: A tela de login apresenta os campos de email/cpf e senha, além de uma opção para cadastro de novos usuários e reset de senha.



Registration Screen: Tela com os campos necessários para criação de novos usuários.

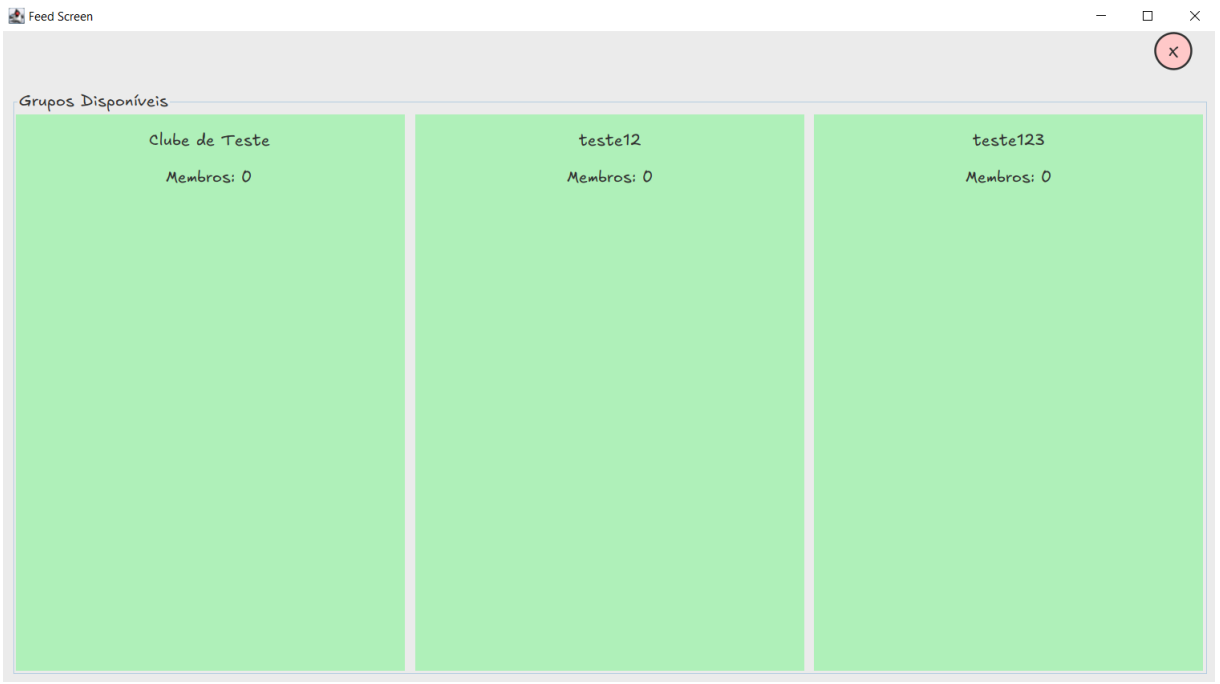


Reset Password Screen: Tela para resetar senha de usuários.



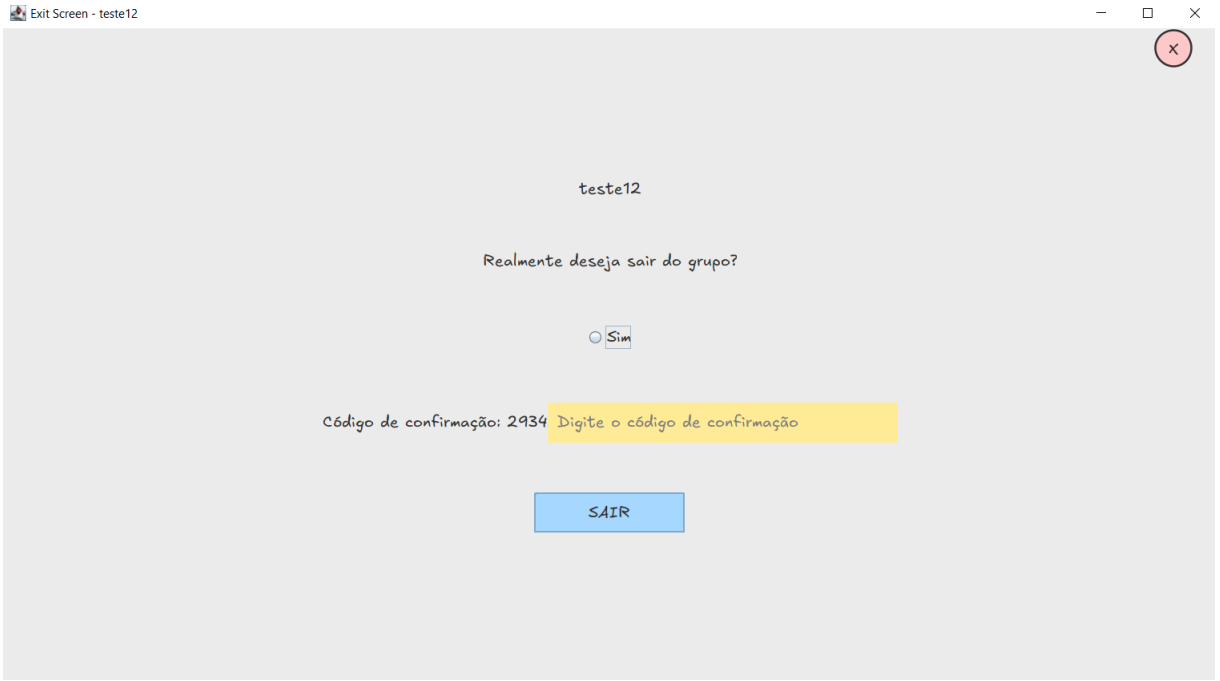
Home Screen: Tela que contém todos grupos cadastrados, além de mostrar os grupos que o usuário já faz parte, conta com três opções de botões, meus grupos, ver grupos e entrar. Duas delas são para exibição de grupos que o usuário faz parte ou administra e a outra é para exibir o feed de grupos disponíveis para ingressar.

Feed Screen: Ao clicar em "Entrar" o usuário recebe todos os grupos disponíveis para ingresso.



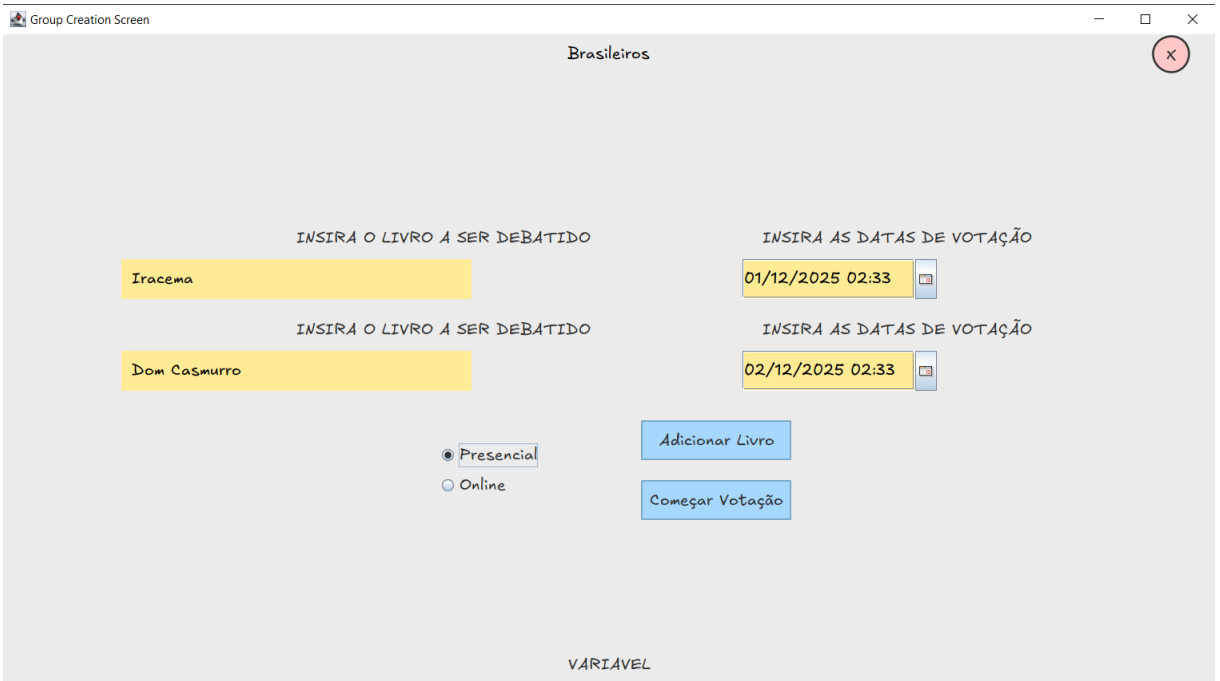
Participate Screen: Nesta tela são exibidos os grupos que o usuário faz parte, com a opção de sair de algum grupo indesejado.

Exit Screen: Tela exibida ao clicar para sair de algum grupo, a tela apresenta uma mensagem de confirmação e um código para verificação. Essa implementação não foi concluída a tempo e não está integrada ao trabalho.

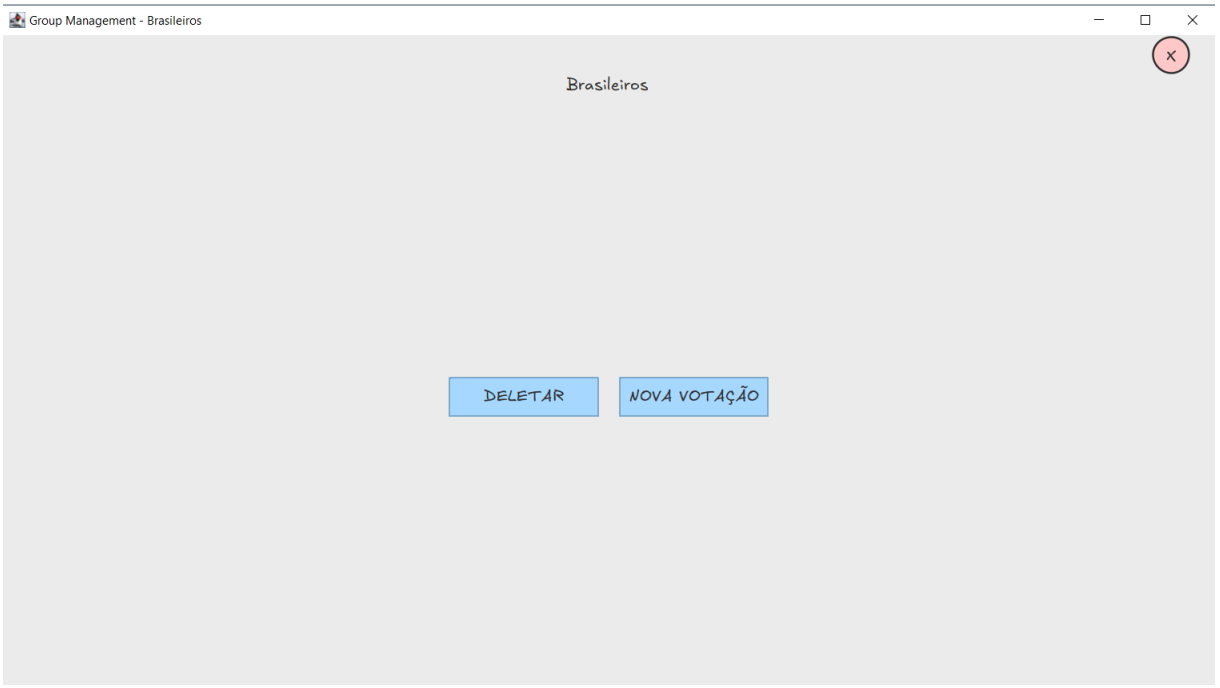


Group User Screen: Tela que permite o usuário gerenciar os grupos que ele administra e também criar novos grupos, também mostra os outros grupos que o usuário participa mas não é dono.

Group Creation Screen: Nessa tela o usuário preenche as informações para a criação de um novo grupo, com duas opções de livros para leitura, período de votação e se o encontro é presencial ou online. Nessa tela também tem a opção de adicionar um livro, mas não implementamos essa funcionalidade a tempo.



Group Management Screen: Tela que permite ao dono de um grupo, deletar o grupo ou criar uma nova votação. Ambas funcionalidades não estão implementadas.



Teste

O grupo trabalhou em conjunto para implementar os testes automatizados do projeto. Enquanto alguns integrantes realizavam o desenvolvimento do código, outros buscavam projetar a estrutura dos testes da melhor maneira possível, visando cobrir dos casos mais gerais até os mais específicos. Abaixo, listamos, separados por classes de teste, os testes que foram construídos para essa etapa do trabalho:

> **BookClubRepositoryTest e UserRepositoryTest**
> São classes de testes referentes ao armazenamento de dados em arquivos. Por mais que tenham sido úteis durante o desenvolvimento do software, não foram solicitadas para o relatório. Consequentemente, não serão detalhadas.

BookClubServiceTest

Testa a classe **BookClubService**, responsável pela criação e gerenciamento de clubes dentro do sistema:

testMultipleClubsCreation:
Verifica se:

- Um mesmo usuário consegue criar múltiplos clubes
- A lista de clubes criados pelo usuário é atualizada corretamente
- O serviço mantém todos os clubes criados

Entradas:

- Usuário u com ID definido
- Dois nomes distintos de clubes

testGetClubsForUser
Verifica se:

- O criador de vários clubes aparece associado a todos eles
- Participantes aparecem corretamente apenas nos clubes em que participam
- Usuários que não participam nem criam clubes têm lista vazia

Entradas:

- Criador
- Participante
- Usuário externo stranger
- Três nomes distintos de clubes

testDuplicateClubNameIsNotAllowed
Verifica se:

- Nomes de clubes não são duplicados para o mesmo usuário
- O serviço mantém apenas um clube registrado com aquele nome

Entradas:

- Usuário u com ID definido
- Tentativa de criar dois clubes com o nome "Clube"

testLeaveClub
Verificar se:

- O criador não pode sair do próprio clube
- Membros comuns podem sair do clube normalmente
- A saída remove corretamente o usuário da lista de participantes

- A saída remove corretamente o clube da lista de clubes do usuário

Entradas:

- Criador
- Membro
- Nome de um clube

BookServiceTest

Testa a classe **BookService**, responsável pelo cadastro, persistência, busca e remoção de livros dentro do sistema.

testConstructor

Verifica se:

- O construtor carrega corretamente os livros salvos no arquivo
- Os livros registrados antes da chamada de saveAllBooks() são recuperados por uma nova instância
- A ordem e os dados dos livros carregados continuam como esperados

Entradas:

- Dois livros distintos
- Arquivo de livros vazio inicial

testSaveAndLoad

Verifica se:

- O método saveAllBooks() salva corretamente todos os livros registrados
- O método loadBooks() recupera os livros corretamente

Entradas:

- Dois livros distintos
- Arquivo de persistência vazio antes do teste

testDeleteBooks

Verifica se:

- O método deleteBook() remove corretamente um livro da lista de livros do serviço
- A ordem e o conteúdo dos livros restantes permanecem corretos
- O serviço não mantém referências ao livro deletado

Entradas:

- Dois livros distintos

findBookByIsbn

Verifica se:

- O método findBookByIsbn() retorna exatamente o livro correto para cada ISBN cadastrado
- A busca funciona para múltiplos livros

Entradas:

- Dois livros, com ISBNs distintos

ServicePollTest

Testa a classe **PollService**, responsável pela criação, persistência, carregamento e votação de enquetes associadas a clubes.

testCriarBookPollParaClube

Verifica se:

- Uma BookPoll é criada corretamente
- O relacionamento bidirecional entre clube e poll é mantido

Entradas:

- Clube clubA
- Parâmetros relativos a uma Book Poll

testCriarDatePollParaClube

Verifica se:

- Uma DatePoll é criada corretamente
- A poll recebe um ID inicial válido
- As opções cadastradas são mantidas corretamente

Entradas:

- Clube
- Parâmetros relativos a uma DatePoll

testIdsIncrementamCorretamente

Verifica se:

- As enquetes criadas recebem IDs incrementados automaticamente
- A ordem de criação é preservada e refletida nos IDs

Entradas:

- Três polls inseridas no clubeA de tipos Book e Date

testSaveAndLoadPolls

Verifica se:

- As enquetes criadas são salvas corretamente no arquivo polls.xls
- O método loadPolls() reconstrói corretamente todas as enquetes salvas
- As enquetes carregadas são associadas novamente aos seus clubes

Entradas:

- Clube
- Duas polls: uma do tipo Book e outra Date

testVotingSystem

Verifica se:

- A votação incrementa corretamente os votos nas opções
- A persistência mantém o total de votos após recarregar o PollService
- O relacionamento entre clube e enquetes é restaurado após o carregamento

Entradas:

- Um Clube

- Uma Poll com três votos distribuídos entre as opções

UserServiceTest

Testa a classe **UserService**, responsável pelo gerenciamento, busca e registro de usuários no sistema, incluindo auto-incremento de IDs e persistência básica.

testFindByEmail

Verifica se:

- Um usuário previamente registrado pode ser encontrado pelo e-mail
- A busca retorna null para e-mails inexistentes

Entradas:

- Usuário com e-mail
- Consulta pelos e-mails válido e inválido

testFindById

Verifica se:

- A busca encontra corretamente o usuário dado um ID específico
- A busca retorna null quando o ID não existe

Entradas:

- Usuário com ID definido manualmente (99)
- Consultas pelos IDs 99 e 100

autoIncrementalIdTest

Verifica se:

- O sistema atribui IDs automaticamente em ordem crescente
- O usuário encontrado por esse ID corresponde aos dados registrados

Entradas:

- Três usuários registrados

Experiência do grupo com os testes unitários

A implementação dos testes inicialmente demandou um tempo extra para entender a estrutura do JUnit e a organização dos testes. Apesar disso, a adoção deles mostrou-se extremamente vantajosa. Os testes nos ajudaram a identificar erros cedo, especialmente em métodos de leitura de arquivos e manipulação de IDs, uma vez que facilitam a visualização do real comportamento do programa. Isso aumentou a confiança do grupo ao refatorar o código, pois qualquer alteração podia ser rapidamente verificada. No geral, a experiência demonstrou que o uso de testes unitários realmente eleva a qualidade do software e reduz o tempo gasto em depuração.

Reuso

Escolhemos a biblioteca de logging *tinylog* para implementarmos no projeto devido a sua facilidade de usabilidade e de configuração (não precisamos instanciar um objeto do tipo *Logger* para realizar os *logs*, por exemplo). Identificamos – como principal dificuldade – a importação das bibliotecas necessárias para o projeto de forma correta. Como utilizamos arquivos .JAR externos, precisamos identificar uma solução para esse impasse. Apesar disso, o resto da implementação do sistema de *logging* ocorreu sem maiores percalços. A configuração foi realizada através da criação do arquivo *tinylog.properties*, que especifica o tipo, local e formatação do arquivo de saída *application.log*, que contém as informações de execução do sistema.

Como de costume, reusamos as bibliotecas padrões da linguagem Java para o desenvolvimento do programa, além de termos reutilizado os métodos da biblioteca *tinylog* para realizar o *logging* e na herança de algumas superclasses. Ao seguir esses princípios, podemos perceber uma melhora significativa na clareza do código, já que não temos código duplicado de forma desnecessária.

Procuramos classificar as informações de registro de uma forma simples, porém, abrangente. A maior parte das ações do sistema serão registradas com suas devidas importâncias (como informação, alerta e erro). Por outro lado, preferimos não registrar algumas ações específicas voltadas ao usuário (como informações relacionadas ao e-mail e senha) justamente por se tratarem de dados sensíveis. Por fim, entendemos que o sistema de *logging* facilita – e muito –

localizar eventuais problemas no código-fonte devido ao processo de organização das informações de execução do programa em um arquivo de saída unificado. Dessa forma, temos uma clareza maior em relação ao o que temos que corrigir quando algo está errado.