UFC

UNIVERSIDADE FEDERAL DO CEARÁ CAMPUS QUIXADÁ
Sistemas Distribuídos

Francisco Wesley Pedro Lima 398261

---

1 O Sistema

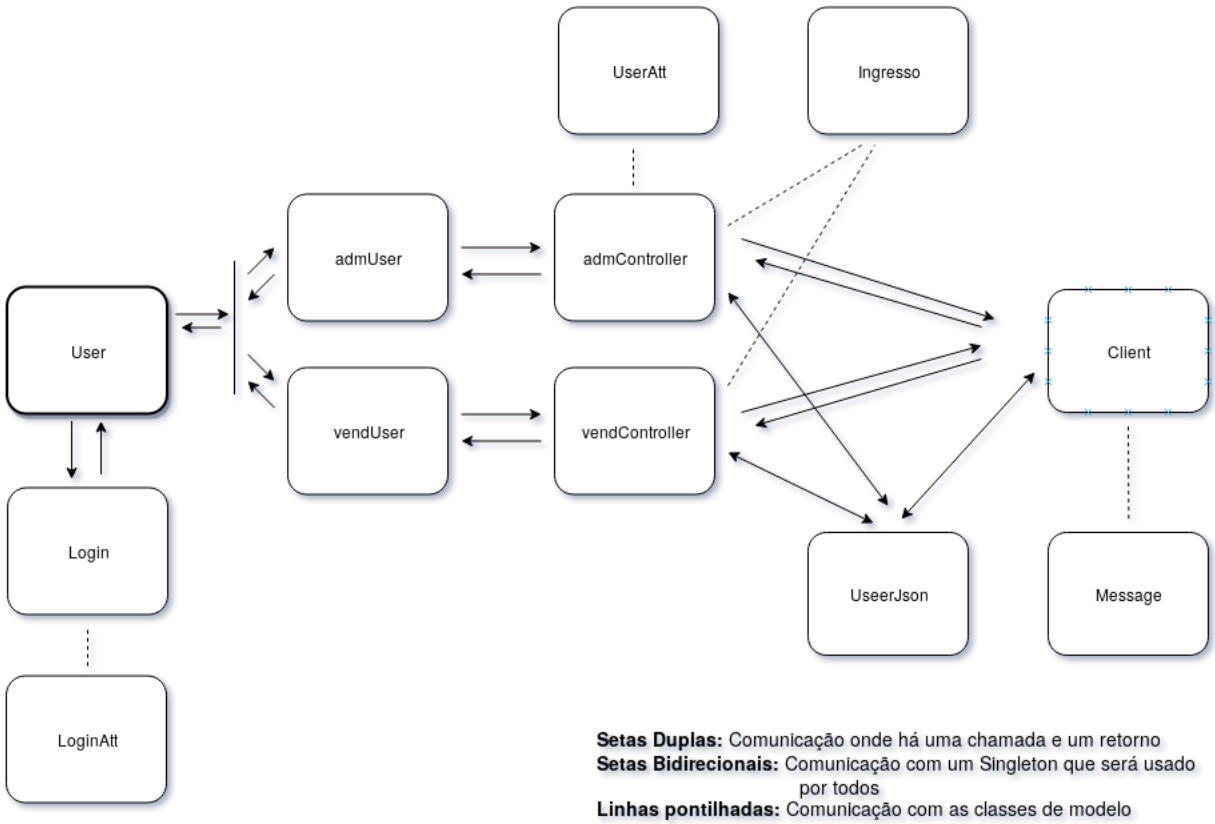
O referido trabalho traz a descrição de um Sistema Distribuído para vendas de Ingressos. O mesmo conta com as seguintes características:

- Login remoto;
- Dois níveis de usuário, administrados e vendedor;
- Permissões diferentes para cada nível de usuário;
- Aplicação cliente desenvolvida em Java;
- Aplicação servidora desenvolvida em Python;
- Integração com banco de dados MySql, e
- Uma grande usabilidade do sistema.

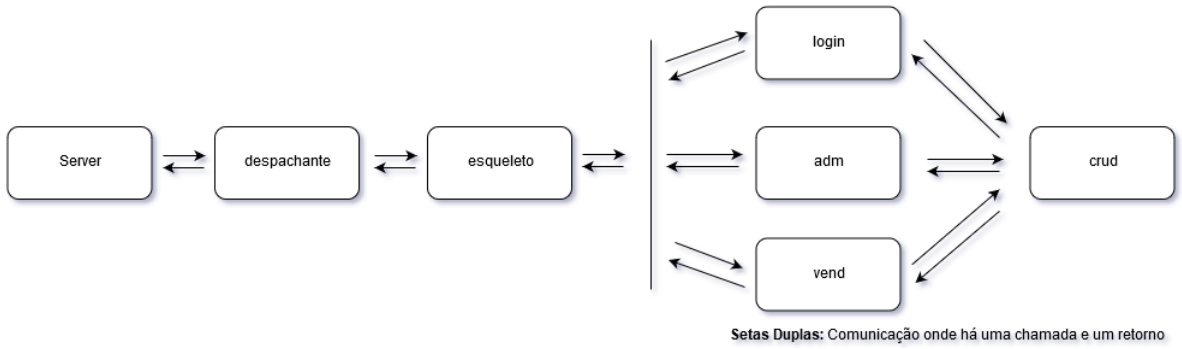
2 Estrutura do Projeto

A seguir, podemos ter um melhor entendimento de como o sistema se encontra, bem como a forma como a comunicação acontece entre as classes do mesmo. Mas a frente, teremos uma descrição dos métodos que cada classe possui, tendo como base apenas a sua assinatura, desprezando, neste documento, o código que compõe o mesmo, mas podendo ser consultado no projeto que segue acompanhando esse arquivo.
Vale ressaltar que nas implementações que seguem, a barra lateral que é representada no gráfico deve ser interpretada como um ponto de decisão, em que a depender o processamento da classe anterior, a mesma seguirá um dos caminhos que se encontra na bifurcação então descrita.
As classes principais estão com um destaque a mais que é uma sombra. Ambas encontram-se no lado esquerdo dos diagramas.

2.1 Cliente



2.2 Servidor



3 Descrição dos métodos

Abaixo, descreveremos os métodos que estão presentes em cada classe e que são partes que realmente compõem a aplicação. Construtores ou coisas métodos de encapsulamento serão desconsiderados.
O comentário correspondente a cada método está referenciado pelo seu número de linha.

3.1 Cliente

Desenvolvido em Java e é composto pelas seguintes classes e seus respectivos métodos:

3.1.1 User

```
1 public static void run();
```

1. Método que é chamado de dentro do método **main** e tem como função instanciar um objeto para a classe *Client* bem como chamar a classe *Login* requisitando o login do usuário e após receber o seu o seu retorno, verificar qual parte do programa instanciar, se uma chamada para o vendedor ou para o administrador do sistema.

3.1.2 Login

```
1 public LoginAtt Logar();
```

1. Recebe os dados do usuário, cria um objeto do tipo *Login*, instancia um **json** para encapsular os dados, envia para o a classe **Client** para consultar com o banco e ao final pega o retorno, desencapsula com o json e retorna para a classe **User**.

3.1.3 LoginAtt

```
1 // Contem apenas construtores e metodos encapsuladores
```

3.1.4 admUser

```
1 public void adm();
```

1. Método que é chamado pela classe **User** quando recebe os dados do servidor e identifica que quem deve operar o sistema é um administrador. Esse método serve apenas para rodar o menu do Administrador e dada uma determinada entrada, invocar na classe **admController** o método correspondente.

3.1.5 vendUser

```
1 public void vend();
```

1. Idem anterior, exceto por chamar a classe **vendController** e o método correspondente.

3.1.6 admController

```
1 public void cadastrarIngresso(Client client);
2 public void venderIngresso(Client client);
3 public void listarIngressos(Client client);
4 public void pesquisarIngresso(Client client);
5 public void atualizarIngresso(Client client);
6 public void removerIngresso(Client client);
7 public void cadastrarUsuario(Client client);
8 public void removerUsuario(Client client);
9 public void alterarSenha(Client client);
```

1. Recebe todos os dados que compõem um ingresso e salva no banco de dados.
2. Usado para vender ingressos.
3. Lista todos os ingressos cadastrados no banco de dados.
4. Pesquisa por ingressos dado uma determinada entrada. Essa entrada pode ser o código ou uma palavra que esteja no nome do evento.
5. Atualiza os dados de um ingresso que esteja relacionado ao código passado.
6. Remove um ingresso do banco de dados dado um determinado código que será validado para saber se existe para ser removido ou não.

7. Recebe CPF, Nome de usuário (username), senha (password), e categoria (category) que pode ser **[1] adm** ou **[2] vend**, representando Administrador e Vendedor respectivamente. Caso os dados estejam coesos, o novo usuário será cadastrado no banco de dados.
8. Método que irá remover um determinado usuário do sistema com base no seu CPF.
9. Método responsável por alterar a senha do administrador. O mesmo recebe o CPF e as senhas, faz a verificação no banco, e se for o caso, atualiza para a nova senha.

3.1.7 vendController

```
public void venderIngresso(Client client);
public void listarIngressos(Client client);
public void pesquisarIngresso(Client client)
public void alterarSenha(Client client)
```

1. Idem venderIngresso da seção anterior
2. Idem listarIngressos da seção anterior
3. Idem pesquisarIngresso da seção anterior
4. Idem alterarSenha da seção anterior

3.1.8 UserAtt

```
public String toString()
```

1. Exceto os métodos incapsuladores e construtor, contém o método acima que tem como função formatar todos os atributos de um determinado objeto e retornar os mesmos para que sejam impressos.

3.1.9 Ingresso

```
public String toString();
```

1. Idem anterior.

3.1.10 UserJson

```
public String LoginToGson(LoginAtt att);
public LoginAtt LoginFromGson(String json);
public String IngressoToGson(Ingresso ingresso);
public Ingresso IngressoFromGson(String json);
public List IngressoListFromGson(String jsonArray);
public String UserAttToGson(UserAtt userAtt);
public UserAtt UserAttFromGson(String json);
public String MessageToGson(Message message);
public Message MessageFromGson(String json)
```

1. Converte um objeto recebido do tipo **LoginAtt** e converte em json.
2. Recebe um json do servidor com os atributos de retorno do login de um usuário e converte para um objeto do tipo **LoginAtt**.
3. Recebe um objeto do tipo **Ingresso** e converte para um json a ser enviado para o servidor.
4. Recebe um json contendo as informações de um único ingresso do servidor e retorna um objeto do tipo **Ingresso**.

5. Recebe um json do servidor contendo um array de ingressos e transforma em um List de objetos do tipo **Ingresso**.
6. Converte um objeto do tipo **UserAtt** para um json a ser enviado para processamento no servidor.
7. Converte uma String de retorno que contém os atributos de um usuário após retornar de um banco de dados para um objeto do tipo **UserAtt**.
8. Converte um objeto **Message** em json para enviar para o servidor.
9. Retorna um objeto do tipo **Message** a partir de um json que contém os atributos referentes a conexão e os dados processados e retornados.

3.1.11 Message

```
1 public String toString();
```

1. Além dos métodos construtores incapsuladores, a classe contém o método *toString()* que deve ser usada para pegar as informações do objeto **Message** em uma String contendo todos os atributos formatados para a saída.

3.1.12 Client

```
1 public Client();
2 public String doOperation(String objeto, int methodId, String arguments);
3 public void sair();
```

1. Método responsável por criar a conexão com o Socket para que depois haja a troca de informações.
2. Responsável por receber os argumentos referentes ao objeto remoto que quero instanciar, o id do método e os argumentos que devem ser repassados para o mesmo. Ele ainda é responsável por montar a mensagem que contém todas essas informações em um json, enviar para o servidor, receber os dados de retorno, desencapsular do json e verificar se a resposta corresponde a requisição e ai sim, retornar para o método que o chamou, devolvendo uma String correspondente aos argumentos.
3. Usado apenas para fechar a conexão do Socket e depois fechar o programa.

3.2 Servidor

Aqui descreveremos brevemente quais são os métodos presentes e a relação que o mesmo tem com as chamadas por parte do cliente.

3.2.1 Server

```
1 def getRequest():
2 def sendReply(requestId, argument):
```

1. É executado quando o servidor começa a rodar e tem como propósito receber os dados de uma requisição do usuário e retornar esses dados para o local onde ele foi chamado.
2. Envia de volta para o cliente o resultado do processamento dos dados enviados ao servidor. A mesma encapsula os dados em um json que será interpretada e transformada em objeto do tipo **Message** do lado cliente.

3.2.2 despachante

```
def invoke(objectReference, methodId, arguments):
```

1. Contém apenas um método que é responsável por receber os atributos da mensagem que correspondem a referencia ao objeto remoto e o seu método. Assim sendo, o mesmo irá encaminhar as informações para o método correspondente que atenderá o chamado e irá encaminhar para o método desejado. Será visto com mais detalhes a seguir.

3.2.3 esqueleto

```
def login(methodId, jArguments):
def admController(methodId, jArguments):
def vendController(methodId, jArguments):
```

1. Recebe o id do método a ser chamado, os argumentos e finalmente encaminha para o método que irá executar a operação final. No caso, enviará para **login** que irá processar as informações e retornar o usuário correspondente a solicitação ou um erro.
2. É invocado quando as operações que serão chamadas correspondem a ações de um usuário de categoria administrativa.
3. Idem anterior, porém associado a operações que serão realizadas por um vendedor.

3.2.4 login

```
def logar(cpf, password):
```

1. Nesse caso, aqui mesmo é tratada as informações, verificado a questão do login, fazendo uma chamada a métodos no *crud* e depois retornada as informações para o usuário, respeitando todo o caminho de volta.

3.2.5 adm

```
# MethodId = 1
def cadastrarIngresso(arguments):
# MethodId = 2
def venderIngresso(arguments):
# MethodId = 3
def listarIngressos(arguments):
# MethodId = 4
def pesquisarIngresso(arguments):
# MethodId = 5
def atualizarIngresso(arguments):
# MethodId = 6
def removerIngresso(arguments):
# MethodId = 7
def cadastrarUsuario(arguments):
# MethodId = 8
def removerUsuario(arguments):
```

```
# MethodId = 9
def alterarSenha(arguments):
# Aux MethodId _8
# MethodId 0
def pesquisarUsuario(arguments):
# Aux MethodId _9
# MethodId 11
def pesquisarUsuarioSenha(arguments):
```

1. Recebe os dados vindos pelos argumentos, no caso referente a um novo ingresso e enviará para que seja feita a inclusão dos mesmos no banco de dados.
2. Recebe os dados do ingresso a ser vendido e depois atualiza as informações no banco de dados por meio do objeto *crud*.
3. Consulta no banco de dados e retorna um json de array com todos as informações de cada ingresso cadastrado no banco.
4. Recebe um determinado parâmetro de condição e por meio do *crud*, pesquisa e depois retorna as informações do ingresso solicitado.
5. Recebe os novos dados de um ingresso e envia para um determinado método do *crud* para ser atualizado no banco.
6. Dada as informações de um certo ingresso, invoca o método correspondente no *crud* e remove esse ingresso do banco de dados.
7. Recebe os dados de um novo usuário e envia para ser inserido no banco de dados.
8. Assim como no ingresso, recebe o CPF de um determinado usuário e envia para ser removido do banco de dados.
9. Recebe os dados do usuário e sua nova senha. Envia por meio do *crud* para o método correspondente e atualiza no banco de dados.
0. Funciona como auxiliar para pesquisar por um determinado usuário antes de remover ele do banco de dados.
11. Idem anterior, porém para alterar a senha.

3.2.6 vend

```
# MethodId = 1
def venderIngresso(arguments):
# MethodId = 2
def listarIngressos(arguments):
# MethodId = 3
def pesquisarIngresso(arguments):
# MethodId = 4
def alterarSenha(arguments):
# Aux MethodId _4
# MethodId 5
def pesquisarUsuarioSenha(arguments):
```

1. Idem método 2 da classe **adm**
2. Idem método 3 da classe **adm**
3. Idem método 4 da classe **adm**

4. Idem método 9 da classe **adm**
5. Idem método 11 da classe **adm**

3.2.7 crud

```
def open():
def close():
def select(fields, tables, where=None):
def insert(values, table, fields=None):
def update(sets, table, where=None):
def delete(table, where):
def updateSenha(newPass, where):
```

1. Responsável por abrir a conexão com o banco de dados.
2. Responsável por fechar a conexão com o banco de dados.
3. Dado os parâmetros repassados, o mesmo usará esses para selecionar determinadas informações que se deseja no banco de dados.
4. Assim como no anterior, recebe as informações e as inserem no banco de dados.
5. Atualiza determinadas informações que são recebidas no banco de dados.
6. Remove um dado do banco de dados obedecendo as restrições recebidas como parâmetro.
7. Usada especificamente para alterar a senha do usuário passado como condição no parâmetro *where*.