Universidade do Minho

Mestrado Integrado em Engenharia Informática

Departamento de Informática

Paradigmas de Sistemas Distribuídos

Peer Lending

Diogo Figueiredo Pimenta A75369
Isabel Sofia da Costa Pereira A76550
Maria de La Salete Dias Teixeira A75281

13 de Janeiro de 2019

Conteúdo

1 Arquitetura do Sistema

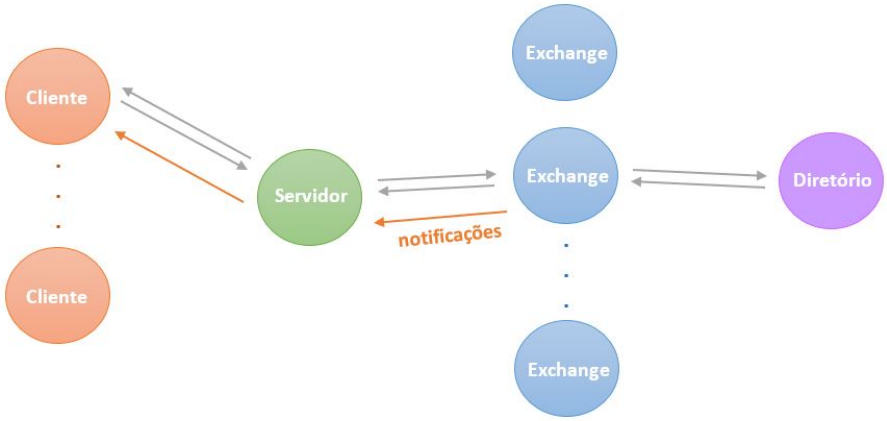

Figura 1: Arquitetura do sistema

Para o desenvolvimento deste projeto, o grupo guiou-se pela arquitetura descrita no enunciado do mesmo, estando esta representada na figura 1. Assim, tem-se quatro componentes principais, sendo estes o Cliente, o Servidor *front-end*, o *Exchange* e o Diretório.

O Cliente envia pedidos para o Servidor e recebe as respetivas respostas através de *Protocol Buffers*. Por sua vez, o Servidor, quando recebe pedidos dos Clientes, envia pedidos para os *Exchange* necessários e recebe as respetivas respostas, também através de *Protocol Buffers*, que são encaminhadas para os Clientes. Cada *Exchange* é responsável por certas empresas, definindo-se a que *Exchange* pertence cada empresa pelo resto da divisão do nome da empresa pelo número de *Exchanges*. Desta forma, aplica-se uma distribuição dos pedidos efetuados. Um *Exchange*, quando recebe um pedido do *front-end*, requisita os dados necessários ao Diretório. O Diretório disponibiliza uma interface *RESTful*, tendo esta sido implementada utilizando o *Dropwizard*. Assim, é o Diretório que armazena todos os dados do programa, tal como as informações sobre as empresas e os leilões e empréstimos em curso. Por último, o Cliente pode também requisitar ao *front-end* a subscrição sobre leilões ou emissões de dívida, obtendo assim informações em tempo real. Quando é feita esta subscrição, o Cliente recebe as informações desejadas graças à implementação do *pub_sub* do *ZeroMQ*. Para além deste caso, pretende-se utilizar também o *push_pull* do *ZeroMQ* para informar os investidores e empresas do resultado final de um leilão ou emissão à taxa fixa.

Os clientes podem realizar pedidos ao Servidor de autenticação como empresa ou como investidor. Uma empresa autenticada pode criar um leilão ou efetuar uma emissão de dívida. Um investidor autenticado pode consultar todos

os leilões e empréstimos ativos, a lista de empresas, a informação relativa a uma empresa que este desejar e, consultar ainda, o leilão e empréstimo ativo de uma determinada empresa. Além disso, pode licitar num leilão e subscrever numa emissão de dívida.

2 Cliente

O ponto inicial da interação do utilizador com o sistema é um Cliente com interface gráfica desenvolvida em Java/Swing. Esta, quando interpelada pelo cliente através dos diversos botões, comunica com o Servidor de *front-end* através de um *stub*. Este *stub* contém todas as ações necessárias para o funcionamento do Cliente com o sistema e permite programar a interface como se as ações ocorressem localmente. Dentro do *stub* encontram-se, encapsulados, os meios de comunicação do cliente com o Servidor de *front-end* através de mensagens serializadas em *protocol-buffers*. Estas mensagens são as que foram especificadas no tópico anterior, sendo a exemplificação de algumas dessas apresentada de seguida.

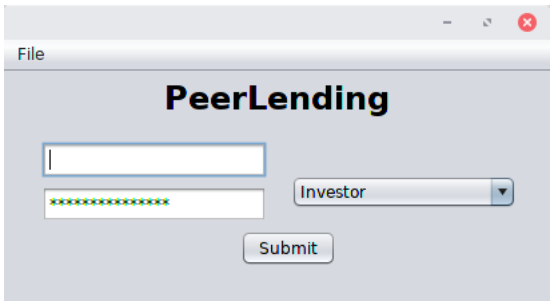

Figura 2: Página de login.

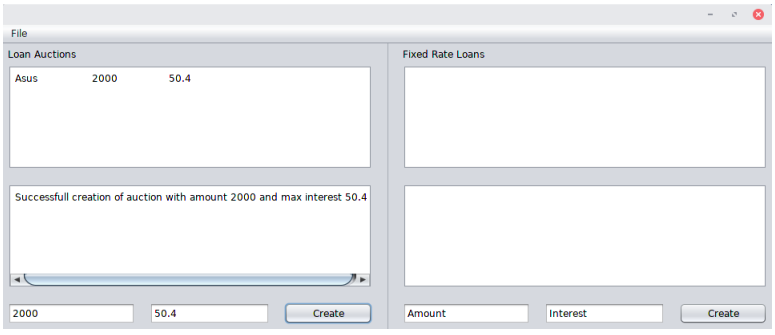

Figura 3: Criação de leilão.

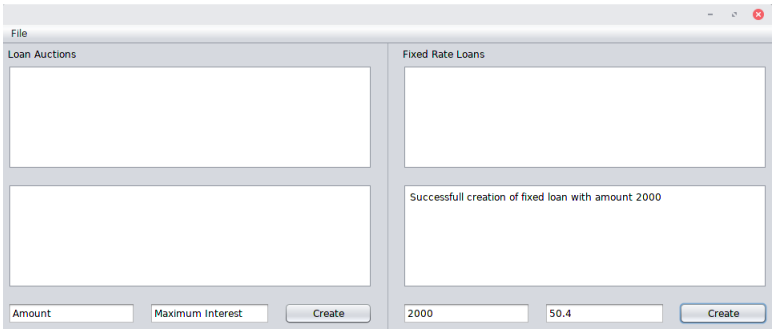

Figura 4: Criação de empréstimo a taxa fixa.

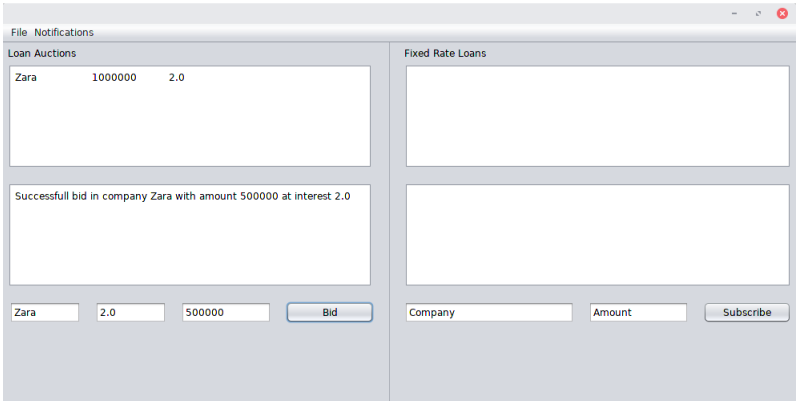

Figura 5: Licitar num leilão.

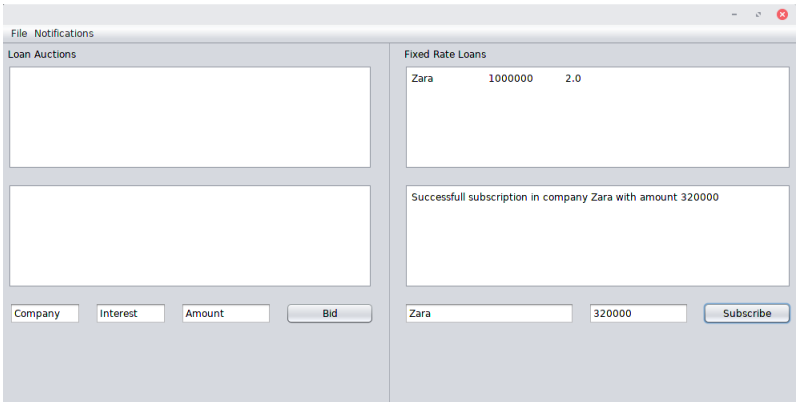

Figura 6: Subscrever empréstimo a taxa fixa.

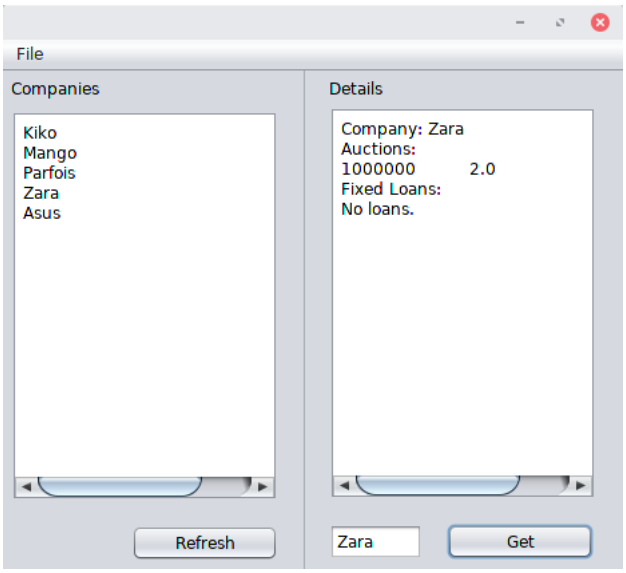

Figura 7: Consulta ao diretório de empresas.

3 Servidor *Front-End*

O Servidor de *front-end* funciona como uma camada inicial, isolada, de autenticação. Antes do Cliente conseguir alcançar uma *Exchange* com um pedido, tem de passar o processo de autenticação neste Servidor. Com o Cliente autenticado, este Servidor tem uma função semelhante a um *proxy* reverso. Por outras palavras, o Cliente efetua um pedido e este é direcionado ao Servidor de *front-end*. Este Servidor, com uma função determinística aplicada ao nome da empresa especificado no pedido, determina para qual *Exchange* deve direcionar o pedido. Para lidar com estas ligações, este Servidor cria um ator por cada Cliente ligado. Caso o Cliente seja uma empresa, cria também uma ligação com a *Exchange* que lida com as ações possíveis da empresa autenticada. Caso o Cliente seja um investidor, abre uma ligação com cada *Exchange* para poder redirecionar os pedidos.

4 Exchange

O *Exchange* é responsável por comunicar com duas entidades, o Servidor e o Diretório, sendo assim o intermediário entre estas duas entidades. Desta forma, o *Exchange* recebe pedidos do Servidor, que se encontram definidos como *protocol-buffers*, identifica o tipo de pedido recebido e executa a ação correspondente, aplicando antes várias verificações de se é plausível a execução deste ou não. Como toda a informação se encontra no Diretório, o *Exchange* contacta este,

através de uma interface *RESTful*, para obter tais informações e aplicar mudanças, caso seja necessário com o sucesso das verificações aplicadas ao pedido. A resposta a enviar ao Servidor, quer seja esta de sucesso, insucesso ou mesmo com informação, é codificada em *protocol-buffers*. Além disso, se os pedidos envolverem a criação de um leilão ou empréstimo, e for de facto possível a criação destes, o *Exchange* dá inicio à contagem do tempo que os mesmos poderão estar ativos. Após esse tempo, o leilão e o empréstimo dão-se como encerrados sendo necessário avisar os participantes dos mesmos de tal, através de *ZeroMQ*, com o devido resultado.

Para o *Exchange* poder atender vários pedidos simultaneamente, este cria uma *thread* sempre que recebe um pedido de conexão de um ator do Servidor. Desta forma, tem-se uma *thread* por cada Cliente para atender os pedidos deste.

5 Diretório

O Diretório é responsável por armazenar todas as informações relativas às empresas, empréstimos e leilões. Assim, na classe *Diretorio* criou-se 3 *maps*, sendo que um armazena as informações das empresas, outro os empréstimos atuais e o último os leilões atuais do sistema. Em todos estes *maps* as chaves são o nome da empresa, que se considerou ser único para cada empresa e, como só pode haver um empréstimo e um leilão ativo em cada empresa, também neste caso o nome da empresa será único. Para dar mais modularidade ao sistema e facilitar a organização e armazenamento das informações, utilizou-se classes auxiliares *Empresa*, *Emprestimo*, *Leilao* e *Oferta*. A classe *Oferta* armazena, para cada investidor que faz uma licitação a um leilão, o montante e a taxa licitados.

Para cada empresa é guardado o nome, o histórico de empréstimos e o histórico de leilões bem sucedidos. Para cada empréstimo (emissão à taxa fixa) é guardado o nome da empresa, o montante desejado, a taxa, o montante total oferecido pelos investidores e um *map* de investidores em que a chave é o nome do investidor e o valor o montante oferecido pelo mesmo. Para cada leilão é guardado o nome da empresa, o montante desejado, a taxa máxima disposta a ser paga e um *map* de investidores alocados em que a chave é o nome do investidor e o valor a *Oferta* realizada pelo mesmo.

Para guardar, alterar e partilhar estas informações, o Diretório disponibiliza uma interface *RESTful*, onde os dados são recebidos e enviados por pedidos *HTTP* em formato *JSON*. Tendo em conta as necessidades do sistema, foram desenvolvidas as seguintes funcionalidades:

Método	Pedido	URL	Descrição
getEmpresas	GET	http://localhost:8080 /diretorio/get_empresas	devolve uma lista com representações de todas as empresas
getEmprestimos	GET	http://localhost:8080 /diretorio/get_emprestimos	devolve uma lista com representações dos empréstimos atuais
getLeiloes	GET	http://localhost:8080 /diretorio/get_leiloes	devolve uma lista com representações dos leiloes atuais
getEmpresa	GET	http://localhost:8080 /diretorio/get_empresa/{nome}	devolve uma representação da empresa desejada
getEmprestimo	GET	http://localhost:8080 /diretorio/get_emprestimo/ {empresa}	devolve uma representação do empréstimo atual para a empresa indicada
getLeilao	GET	http://localhost:8080 /diretorio/get_leilao/{empresa}	devolve uma representação do leilão atual para a empresa indicada
lastEmprestimo	GET	http://localhost:8080 /diretorio/last_emprestimo/ {empresa}	devolve uma representação do último empréstimo finalizado na empresa desejada
lastLeilao	GET	http://localhost:8080 /diretorio/last_leilao/{empresa}	devolve uma representação do último leilão bem sucedido na empresa desejada
putEmprestimo	PUT	http://localhost:8080 /diretorio/add_emprestimo/ {empresa}/ {montante}_{taxa}	guarda o empréstimo atual da empresa indicada
putLeilao	PUT	http://localhost:8080 /diretorio/add_leilao/ {empresa}/ {montante}_{taxa}	guarda o leilão atual da empresa indicada
addInvestidorEmprestimo	POST	http://localhost:8080 /diretorio/ add_investidor_ emprestimo/{empresa}/ {investidor}/ {montante}	adiciona um investidor ao empréstimo atual da empresa
addInvestidorLeilao	POST	http://localhost:8080 /diretorio/add_investidor_leilao/ {empresa}/{investidor}/ {montante}_{taxa}	adiciona um investidor ao leilão atual da empresa
endEmprestimo	POST	http://localhost:8080 /diretorio/end_emprestimo/ {empresa}	remove o empréstimo dos empréstimos atuais e adiciona-o ao histórico da empresa
endLeilao	POST	http://localhost:8080 /diretorio/end_leilao/ {empresa}/investidores?	remove o leilão dos leilões atuais e adiciona-o ao histórico da empresa com os investidores indicados

Tabela 1: Funcionalidades do Diretório

O método *endLeilao* recebe o nome, montante e taxa oferecidos dos investidores como *query param*, pelo que para enviar um pedido *HTTP* para o mesmo deve-se fazê-lo como no exemplo que se segue.

```
http://localhost:8080/diretorio//end_leilao/Mango/investidores?
?inv=Joao&m=500&t=0.3&inv=Diogo&m=400&t=0.25
```

6 Apreciação Crítica

Na realização deste trabalho foi possível aplicar vários dos conhecimentos adquiridos na cadeira, tendo-se usado então o paradigma Cliente-Servidor com *Multithreaded Client*, serialização de dados com o uso de *Protocol Buffers*, atores com a aplicação de *Erlang* e interfaces *RESTful*. O único conhecimento mais relevante que não foi aplicado foi a utilização de *Message Oriented Middleware* que seria utilizado, através do uso da biblioteca *ZeroMQ*, para implementar o envio, aos participantes de um leilão ou empréstimo, do resultado destes quando estes terminam. Além disso, também serviria para aplicar subscrições, por parte dos Clientes, a leilões ou empréstimos, sendo os subscritores avisados quando houvesse alguma mudança no contexto desse objeto. Desta forma, esta implementação é colocada como trabalho futuro.

Em relação às dificuldades encontradas no decorrer do trabalho, pode-se apontar a complexidade da interligação entre diferentes componentes, nomeadamente a interligação de *Erlang* com *ZeroMQ*, sendo esta uma das razões pela qual a sua implementação não foi bem sucedida. Para além disso, pode-se apontar também a utilização de *Protocol Buffers* entre *Erlang* e *Java*, devido ao facto de em *Erlang* se utilizar *big-endian*, pelo que a descoberta deste erro acarretou uma perda de tempo significativa.

Quanto a questões de arquitetura, teve-se o cuidado de tornar o programa o mais concorrente possível para assim se evitar esperas desnecessárias e *bottlenecks*. A nível de escalabilidade, consideramos que esta se torna facilmente escalável pois é simples adicionar novos Clientes e *Exchanges* ao problema.

Dado como concluído o trabalho, consideramos que o que foi implementado foi bem sucedido e se encontra de acordo com o que requerido para essas funcionalidades.