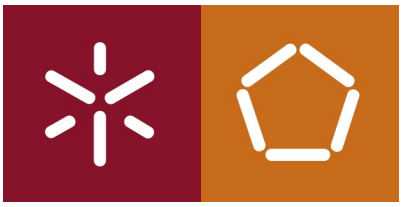

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Paradigmas de Sistemas Distribuídos

PeerLending: serviço de empréstimos entre pares

Grupo 12

Ana Rodrigues	(A78763)
Armando Santos	(A77628)
Cláudia Correia	(A77431)

20 de Janeiro de 2019

Conteúdo

Resumo

Este documento diz respeito ao trabalho prático proposto na unidade curricular de Paradigmas de Sistemas Distribuídos da Universidade do Minho.

O objetivo deste projeto consiste no desenvolvimento de um protótipo de um serviço de intermediação de empréstimos a empresas por parte de investidores.

1 Introdução

O presente documento tem como finalidade justificar as estratégias adotadas na criação de um serviço de empréstimos a empresas, que envolve vários componentes de *software*: o cliente, o servidor de *front-end*, a *exchange* e o diretório.

Este serviço possui dois tipos de utilizadores diferentes, nomeadamente as empresas, que requisitam os empréstimos, e os investidores, que fazem o financiamento. As empresas podem realizar leilões/emissões à taxa fixa, enquanto os investidores podem licitar/subscrever essas ações. A existência de notificações em tempo real é útil para informar os clientes e os investidores dos acontecimentos. Para além disto, existe um diretório a partir do qual se podem consultar informações acerca das empresas existentes e dos seus históricos de ações, e também dos leilões/emissões que estão a decorrer em tempo real.

Relativamente ao funcionamento do sistema, os clientes ligam-se ao servidor de *front-end* que, por sua vez, os autentica, recebe pedidos e encaminha-os para uma, entre várias, *exchange*. As *exchanges* são responsáveis por processar os pedidos dos clientes. Por sua vez, para além de requisitar serviços, o cliente pode também decidir ser notificado relativamente às ações de determinadas empresas. O diretório disponibiliza uma interface *RESTful*, que pode ser utilizada para consultar informações do sistema.

Todos estes componentes do *software* deverão ser implementados utilizando Java (cliente, *exchange*, diretório), Erlang (servidor *front-end*), Protocol Buffers, ZeroMQ e Dropwizard.

2 Estrutura do sistema

Como ponto de partida para a resolução do problema, foram tomadas diversas decisões quanto à estrutura da solução, de modo a facilitar a implementação da mesma e a cumprir com os requisitos mínimos obrigatórios. Na figura abaixo encontra-se especificado o funcionamento da arquitetura geral do sistema.

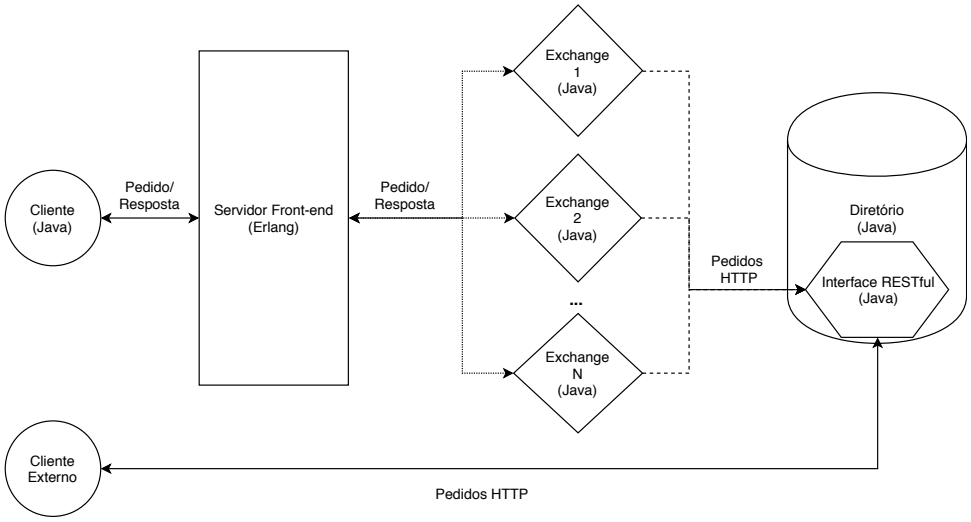

Figura 1: Esquema da arquitetura geral do sistema.

Relativamente à propagação de pedidos entre os diferentes componentes da arquitetura, a estratégia utilizada foi a apresentada em baixo. A interação entre o cliente e o servidor de *front-end* deverá ser feita com *sockets* TCP para distinguir sessões de diferentes clientes. Relativamente ao servidor de *front-end* e à *exchange*, estes devem comunicar através de *sockets* ZeroMQ REQ-REP, para efetuar pedidos e tratar da respetiva resposta ao pedido realizado. Por fim, as *exchanges* deverão atualizar a informação existente no diretório efetuando pedidos HTTP, através da interface RESTful deste componente.

A comunicação entre os diferentes componentes deverá ser efetuada recorrendo a Protocol Buffers.

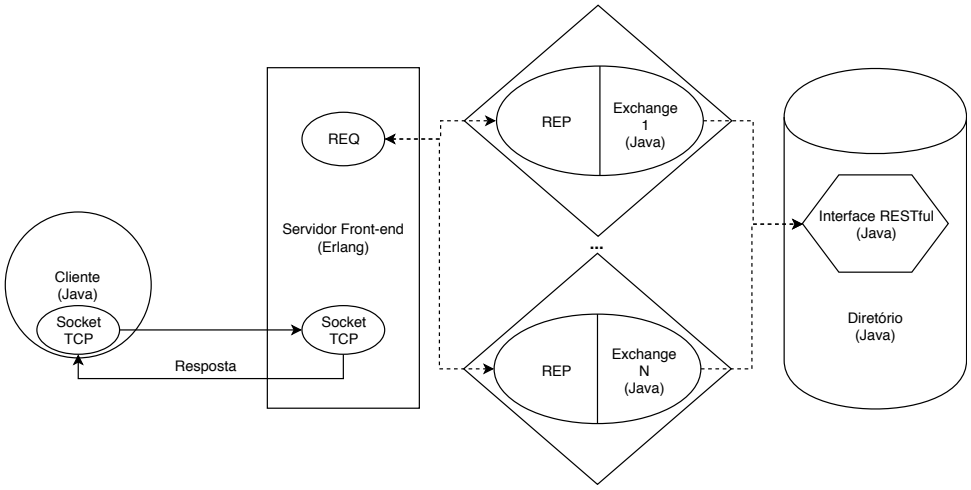

Figura 2: Esquema da interação entre os componentes do sistema.

No que toca a notificações, optou-se por recorrer a *sockets* ZeroMQ PUB-SUB de forma a expressar o interesse dos clientes em receber notificações de uma ou mais empresas, isto é, subscrever determinados eventos segundo um critério. O conjunto dos *sockets* XPUB e XSUB com o *proxy*, tem como objetivo adicionar um intermediário ao sistema, isto é, um ponto estático da rede onde todos os clientes e todas as *exchanges* estão conectados. Desta forma, torna-se mais simples de adicionar mais *publishers* e *subscribers* à rede, e as *exchanges* limitam-se a propagar as notificações para o intermediário, em vez de para todos os clientes conectados. Apesar de só ser utilizado um intermediário, perante um cenário onde exista um grande número de utilizadores, podem ser inseridos mais intermediários, tornando todo o processo mais eficiente.

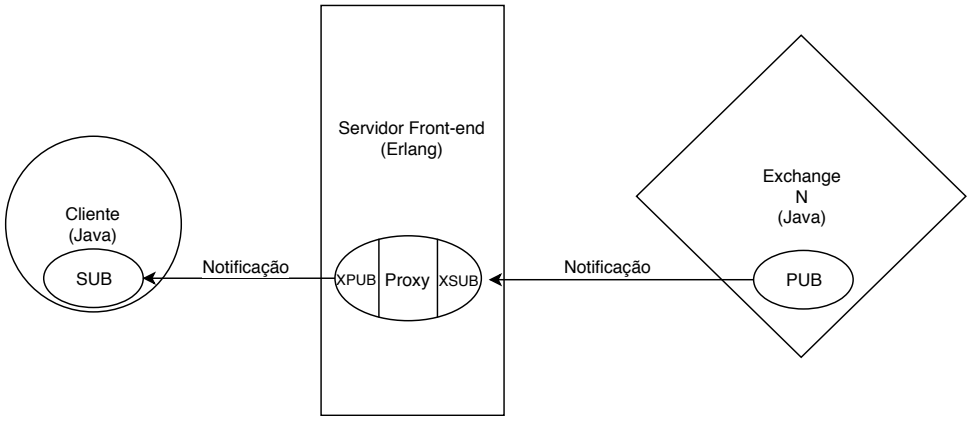

Figura 3: Esquema da propagação de notificações no sistema.

3 Implementação

3.1 Cliente

O cliente comunica com o servidor de *front-end* através de *sockets* TCP, enviando e recebendo mensagens deste com recurso a Protocol Buffers.

```
message Message {
    optional string type = 1; // Bid, Subscription, Auction, Emission
    optional int32 amount = 2;
    optional float interest = 3;
    optional string company = 4;
    optional string investor = 5;
    optional Authentication auth = 6;
    optional Result res = 7;
}
```

3.1.1 Interface com o utilizador

Um cliente pode ser um investidor (efetua licitações/subscrições) ou uma empresa (cria leilões/emissões), sendo, portanto, necessário diferenciar os dois. Esta distinção é realizada no servidor de *front-end*, que efetua a autenticação do cliente e retorna o resultado da mesma, juntamente com o tipo de utilizador em questão. Desta forma, o programa cliente saberá que opções apresentar ao utilizador.

Consoante o papel do utilizador, são apresentados os respetivos menus:

Investidor:

1. Licitação em leilão;
2. Subscrição de emissão à taxa fixa;
3. Registar ou cancelar interesse em ser notificado relativamente a certos tópicos;
4. Terminar sessão.

Empresa:

1. Criação de leilão;
2. Emissão à taxa fixa;
3. Registar ou cancelar interesse em ser notificado relativamente a certos tópicos;
4. Terminar sessão.

3.1.2 Notificações

Para ser possível receber as notificações do seu interesse, foi criado no cliente um *socket* ZeroMQ SUB e lançada uma *thread*, denominada `Subscriber`. Esta *thread* fica "atenta" às publicações que vão sendo feitas, notificando o cliente quando estas chegam.

Para além disso, sempre que um cliente efetua uma ação (licitar, subscrever emissão, criar leilão, criar emissão), é realizada uma subscrição ao tópico correspondente, de modo a este conseguir acompanhar o decorrer do leilão, recebendo assim as notificações relativas ao mesmo.

Para além da subscrição automática que é feita pelo programa cliente sempre que o utilizador participa numa ação, este também pode optar por ativar as notificações relativas a uma ou mais empresas. De modo a diferenciar as notificações de algo em que o cliente está a participar das que este apenas "ativou", definiram-se os seguintes tópicos de subscrição:

Tabela 1: Tópicos de subscrição de notificações.

	Participa na ação		**Ativa notificações**	
	Leilão	**Emissão**	**Leilão**	**Emissão**
Criação	Auction:Company	Emission:Company	CreateAuction:Company	CreateEmission:Company
Licitação	Auction:Company	Emission:Company	BidAuction:Company	BidEmission:Company
Fim	Auction:Company	Emission:Company	EndAuction:Company	EndEmission:Company

Tal como foi referido em 3.1.2, foi feita a distinção dos tópicos de subscrição dos clientes que participam numa dada ação (licitação, subscrição, criação de leilão, criação de emissão) dos clientes que apenas "ativam" as notificações do seu interesse. Isto deve-se ao facto de, sempre que um leilão/emissão termina, ser efetuado *unsubscribe* apenas dos tópicos dos clientes participantes, garantindo, assim, que os restantes clientes continuam a receber as respetivas notificações que subscreveram.

3.2 Servidor *front-end*

O servidor de *front-end* faz a ponte entre os clientes e as unidades de processamento (*exchanges*), tendo sido implementado em Erlang. De forma a tornar possível a utilização dos diversos tipos de *sockets* que suportam a arquitetura geral (figura 1), recorreu-se à biblioteca de ZeroMQ ERLMZQ2.

3.2.1 Autenticação

Tanto os investidores como as empresas já se encontram pré-populados no servidor de *front-end*. Sendo assim, a partir do momento em que um cliente efetua uma conexão, este é autenticado e a sua sessão mantida.

3.2.2 Encaminhamento de pedidos para as *exchanges*

Uma das funções deste servidor é fazer chegar às *exchanges* os pedidos de cada cliente, no entanto, por uma questão de escalabilidade, cada *exchange* é responsável por um certo conjunto de empresas e o servidor de *front-end* deve ser capaz de encaminhar corretamente cada pedido. É importante mencionar que dada a pré-população do sistema, cada componente possui conhecimento acerca das *exchanges* existentes e do conjunto de empresas pelo qual cada uma é responsável.

3.2.3 Intermediação de notificações

Com o objetivo de distribuir o custo de propagação de notificações provenientes das *exchanges*, o servidor de *front-end* serve também como um *broker*, posto isto, é lançado um processo que irá agir como *proxy*. Este processo possui *sockets* PUB-SUB, em que o *socket* SUB subscreve todo o tipo de tópicos, recebendo todas as notificações das *exchanges*, e o *socket* PUB encaminha-as para os clientes conectados.

É importante salientar que o plano inicial de implementação deste *broker* consistia na utilização de *sockets* do tipo XSUB-XPUB, tal como foi apresentado na figura 3, uma vez que estes possuem mecanismos de propagação de subscrições/notificações mais eficientes. No entanto, a biblioteca ERLMZQ2 não fornece este tipo de *sockets*.

3.3 *Exchange*

As *exchanges* efetuam o processamento dos leilões e das emissões à taxa fixa, sendo que cada uma é responsável por um conjunto de empresas. Cada *exchange* recebe do servidor de *front-end* os pedidos efetuados pelos clientes, através de um *socket* ZeroMQ REP, tratando cada um da forma mais adequada.

Para tratar as ações em curso, as *exchanges* mantêm registo dos leilões e das emissões que estão a decorrer no momento. Para além disto, armazenam também o histórico de cada empresa, para conseguir processar devidamente as ações que lhe são solicitadas.

3.3.1 Licitações e subscrições

No caso de uma nova licitação (`Bid`) para um dado leilão, esta é apenas inserida nas licitações do leilão se este de facto estiver em curso. O mesmo raciocínio aplica-se ao caso de uma subscrição (`Subscription`), mas com um passo extra: antes da subscrição ser inserida, verifica-se se é ultrapassado o total estipulado pela empresa, se sim, a subscrição é rejeitada, caso contrário, é inserida. Esta abordagem foi adotada pois, caso o montante pretendido pela empresa já tenha sido atingido, não faz sentido continuar a aceitar subscrições, visto que se isto acontecer a taxa de juro pretendida pela empresa será ultrapassada.

3.3.2 Leilões e emissões à taxa fixa

No caso da criação de um leilão, este é efetuado apenas caso essa empresa não possua mais nenhum leilão em curso. Já no caso da criação de uma emissão, esta só pode ser efetuada caso a empresa tenha efetuado pelo menos um leilão com sucesso. Caso isto se verifique, uma vez que existe uma taxa de juro fixa associada, é necessário calcular a taxa que será atribuída à emissão. Este cálculo é efetuado com base nas emissões anteriormente efetuadas pela empresa:

- **a empresa nunca efetuou uma emissão:** é selecionada a taxa mais alta alocada no leilão mais recente;
- **a emissão mais recente foi mal sucedida:** é selecionada a taxa de 1.1 vezes a da emissão mais recente;
- **a emissão mais recente foi bem sucedida:** é selecionada a taxa mais baixa, tendo em conta todas as taxas das emissões efetuadas que foram bem sucedidas e a taxa mais alta do leilão mais recente.

3.3.3 Conclusão de leilões e emissões à taxa fixa

Ao criar um leilão ou uma emissão, a *exchange* agenda uma TimerTask, denominada `AuctioneerTask`, que é lançada passado um minuto do início do leilão/emissão. Quando a TimerTask é acionada, esta toma a decisão final acerca do leilão/emissão ao qual está associada, e envia a respetiva notificação.

3.3.4 Notificações

Visto que são as *exchanges* que processam a criação e o decorrer dos leilões e emissões, é nestas que vão ser enviadas as publicações relativas ao decorrer das ações. Para tal, foi criado um `Publisher`, partilhado por todas as *exchanges* e pelas `AuctioneerTask` lançadas. Este `Publisher` é responsável, então, por enviar as mensagens correspondentes aos tópicos referidos na Tabela 1, sempre que uma ação é processada.

3.3.5 Atualização do diretório

Sempre que uma *exchange* cria um leilão/emissão, esta efetua um pedido HTTP POST para que o leilão/emissão em curso seja inserido na estrutura do diretório. Para além disto, sempre que um leilão/emissão é terminado, é feito um pedido HTTP POST para que essa ação seja adicionada ao histórico da empresa. Este processo foi concretizado recorrendo ao Apache HTTP Client e à biblioteca Jackson.

3.4 Diretório

O diretório funciona como um repositório de dados relativos ao serviço de empréstimos. Embora tenha sido implementado em Java, o diretório disponibiliza uma interface

RESTful, recorrendo à *framework* Dropwizard, que pode ser acedida por clientes externos através de pedidos HTTP GET. Para realizar o *parsing* do JSON, foi utilizada a biblioteca Jackson.

A informação que é possível consultar no diretório é a seguinte:

- empresas registadas no sistema (`/directory/companies`);
- histórico de ações de uma empresa (`/directory/company/`*`name`*);
- leilões em curso (`/directory/available/auctions`);
- emissões à taxa fixa em curso (`/directory/available/emissions`).

Para além de poderem ser feitas consultas, a interface do diretório disponibiliza também formas de atualizar a informação nele contida. Os componentes responsáveis por estas atualizações são as *exchanges*, tal como foi explicitado em 3.3.5.

4 Conclusões e Trabalho Futuro

De uma forma geral, conclui-se que o resultado obtido cumpre com os requisitos estipulados em 1. Para além das funcionalidades básicas, foram adotadas estratégias para obter um melhor desempenho do sistema, como, por exemplo, a criação de um *broker* para distribuir os custos de propagação das notificações.

Como trabalho futuro, poderia ser melhorada a interface com o utilizador, apresentando detalhes como, por exemplo, o valor ainda em falta de uma emissão a decorrer, isto é, a quantidade monetária que a empresa ainda necessita para atingir o montante total que estipulou. Para além disso, poderiam ser inseridos mais intermediários (*brokers*), de modo a tornar todo o processo das notificações mais eficiente. Com o objetivo de tornar o povoamento do sistema reajustável, poderia também ser utilizado um ficheiro de configuração que especificasse o número de *exchanges* a utilizar, e as empresas correspondentes a cada uma.