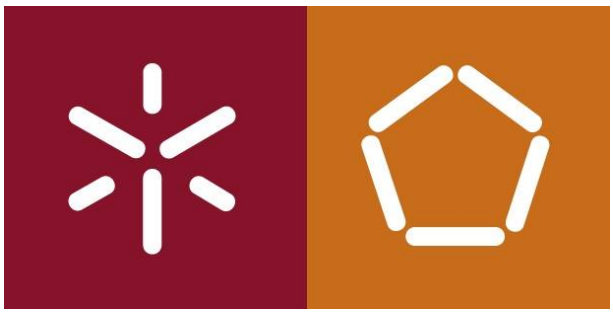

Universidade do Minho

Mestrado em Engenharia Informática

Paradigmas de Sistemas Distribuídos

Alarme Covid

Grupo 6

Flávio Martins (a65277)
Pedro Costa (a85700)
Joel Ferreira (a89982)
Ana Ferreira (pg44412)
Bruno Santos (pg44414)

19 de janeiro de 2021

Conteúdo

1 Introdução **2**

2 Arquitetura **3**

3 Componentes **4**

3.1 Diretório 4

3.2 Cliente 5

3.3 Servidor *Front-End* 6

3.4 Servidores Distritais 7

3.5 *Broker* 7

4 Conclusão **9**

1 Introdução

O presente trabalho prático foi realizado no âmbito da unidade curricular de Paradigmas de Sistemas Distribuídos e consistiu no desenvolvimento de um protótipo de uma plataforma para suporte de rastreio de contactos e deteção de concentração de pessoas.

O sistema desenvolvido deve envolver vários componentes de *software*, tais como a aplicação cliente, o servidor *front-end*, os servidores distritais e o diretório. Este tem como finalidade permitir que múltiplos utilizadores se liguem via aplicação cliente ao servidor *front-end*. Este é responsável pela autenticação, recebe informações ou pedidos do cliente e encaminha-os para o servidor distrital adequado. Os servidores distritais tratam do processamento dos pedidos do respetivo distrito e armazenam informação sobre o mesmo. Os utilizadores poderão, ainda, subscrever a informação pública em tempo real, assim como obter informação estatística pública disponibilizada pela interface REST do diretório.

Este relatório tem como finalidade a apresentação da arquitetura implementada e explicação do funcionamento de cada um dos componentes do sistema, assim como da forma como estes comunicam entre si.

2 Arquitetura

O sistema desenvolvido envolve vários componentes de *software*, nomeadamente a aplicação cliente, o servidor *front-end*, os servidores distritais, os *brokers* e o diretório. A arquitetura desenvolvida para o sistema encontra-se representada na Figura 1.

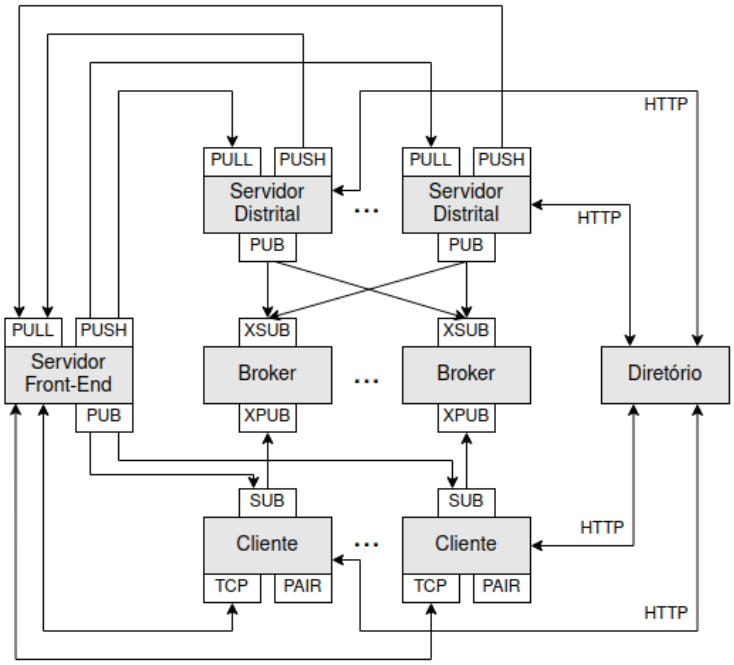

Figura 1: Arquitetura do sistema desenvolvido.

O cliente comunica via *sockets* TCP com o servidor *front-end*, que faz a autenticação, recebe informação ou pedidos do cliente e encaminha-os para o servidor distrital apropriado, com o qual está conectado através de *sockets* PUSH e PULL disponibilizadas pela biblioteca **ZeroMQ**, que servem para o envio e receção de mensagens, respetivamente. Os servidores distritais são responsáveis por processar os pedidos recebidos, devolver respostas e manter informação sobre o respetivo distrito, tal como a localização dos utilizadores do distrito. Quando um utilizador declara estar doente, o serviço deve notificar todos os utilizadores potencialmente contagiados por este, via *front-end*, pelo que cada servidor distrital possui, ainda, um *socket* do tipo PUSH para o envio das notificações de possível contágio, recebidas pelo servidor *front-end* através do *socket* PULL. Para fazer chegar esta notificação ao cliente, o servidor *front-end* possui, também, um *socket* do tipo PUB, onde os clientes se irão ligar para receber as notificações privadas, garantindo assim a autenticação do receptor.

Os clientes têm, também, a possibilidade de subscreverem a informação pública em tempo real relativa a, no máximo, três distritos. Deste modo, optou-se por implementar *brokers* que servem de intermediários entre os servidores distritais e os clientes. Assim, para a transmissão da notificação de informação pública sobre um dado distrito aos clientes subscritos, os servidores distritais têm um

socket PUB, através do qual comunicam com o *socket* XSUB do *broker*. Por sua vez, o *broker* envia a informação aos clientes subscritos através do *socket* XPUB, que possuem um *socket* do tipo SUB para a receção de notificações das subscrições. Na aplicação do cliente foi, ainda, implementado um *socket* PAIR, que tem como finalidade facilitar a subscrição e o cancelamento de subscrições do utilizador, bem como apagar todas as subscrições de notificações ativas no momento do *logout*.

O diretório, via uma API REST, recebe informação dos servidores distritais e permite a consulta pública de informação estatística através da aplicação cliente.

Para permitir uma maior heterogeneidade entre os componentes e uma vez que alguns são escritos em diferentes linguagens de programação, optou-se por utilizar protocolos de serialização de dados recorrendo a **Protocol Buffers**, uma vez que o grupo achou que seria bastante útil e interessante, tendo encontrado estes protocolos em trabalhos de anos passados.

3 Componentes

3.1 Diretório

O diretório tem como finalidade receber informação dos servidores distritais, de modo a permitir a consulta pública de informação estatística, via API REST, por parte da aplicação cliente. Este foi codificado na linguagem Java e utilizou-se a *framework* **Dropwizard**. O diretório encontra-se no módulo **API**.

As operações manipulam recursos através de representações. Desta forma, foram concebidas três representações, que se encontram no *package* **api**. `Coordinate` é a classe que representa uma coordenada, com o seu par latitude-longitude, `Location` contém uma coordenada, um distrito e o número de utilizadores presentes na respetiva localização e `District` contém um mapa com todas as localizações presentes nesse distrito.

Uma vez que REST é orientado a recursos, no *package* **resource** definiu-se o recurso **DistrictResource**. A presente classe contém um *ConcurrentHashMap* com todos os distritos, sendo que cada distrito possui várias localizações e cada localização apresenta a respetiva coordenada e o número de utilizadores. Esta classe oferece várias operações que permitem receber informações dos servidores distritais, que ficam armazenadas no *ConcurrentHashMap*, e responder a consultas de vários clientes, via API REST.

Os seguintes métodos HTTP permitem receber informação dos servidores distritais:

- POST - método não seguro e não idempotente.
  - /**District** - permite adicionar um distrito no *ConcurrentHashMap*.
  - /**User** - permite adicionar um utilizador. O distrito e a sua localização encontram-se no *body*.
  - /**Infected** - permite adicionar um infetado e atualizar os contactos infetados num dado distrito presente no *body*.

- PUT - método não seguro mas idempotente.
  - /**moveUser** - permite atualizar a localização de um dado utilizador.

Os seguintes métodos HTTP permitem a consulta pública da informação estatística pelos clientes:

- GET - método seguro e idempotente.
  - /**{district}**/**Users** - número de utilizadores de um dado distrito.
  - /**{district}**/**Infecteds** - número de infectados de um dado distrito.
  - /**TopRatio** - top 5 dos distritos com maior rácio de infectados / utilizadores.
  - /**Location**/**MostUsers** - top 5 das localizações que tiveram o maior número de pessoas em simultâneo.
  - /**AvgInfectedContacts** - número médio de utilizadores que se cruzaram com utilizadores declarados doentes.

3.2 Cliente

A aplicação cliente foi concebida utilizando Java e recorrendo à biblioteca ZeroMQ, encontrando-se no módulo **Client**. Esta disponibiliza várias opções, nomeadamente registar e efetuar login de um cliente; manter o serviço informado sobre a sua localização atual; comunicar ao serviço no caso de doença; receber várias notificações subscritas e consultar informação estatística variada.

A aplicação cliente, ao iniciar, lê um ficheiro de configuração, no qual se encontra o endereço e a porta do servidor *front-end*, a porta do *socket* XPUB do *broker* e o *url* do diretório para o qual os pedidos HTTP serão efetuados. Dispondo das configurações, são concebidos um *socket* TCP para se comunicar com o servidor *front-end* e um *socket* ZeroMQ do tipo SUB para receber as informações subscritas.

Todas as mensagens enviadas de/para o servidor *front-end* são deseralizadas/serializadas em dados estruturados através do mecanismo de Protocol Buffers. Os métodos usados na serialização encontram-se no package **protocol**.

O menu inicial da presente aplicação (fase AUTH) apresenta quatro opções: autenticação, registo, ver estatísticas e sair. No login é elaborada uma mensagem com o nome e a password do cliente e, depois de transformá-la em dados binários através do Protocol Buffers, envia para o servidor *front-end*. Se o login foi realizado com sucesso, o cliente actualiza a sua localização e recebe as subcrições que tem ativas. O registo é similar ao login. É pedido o nome de utilizador, a password e o distrito, que serão enviados ao servidor *front-end* de forma serializada para verificação. O cliente recebe a resposta, deserializa através do Protocol buffers implementado. Em caso de o sucesso, volta para o menu inicial.

Uma vez que as estatísticas são públicas, optou-se por colocá-las no menu inicial. Deste modo, um cliente não precisa de fazer login para as visualizar. Selecionada a opção de estatísticas, irá aparecer um novo menu (fase STATISTICS_MENU). Neste menu estão apresentadas todas as estatísticas solicitadas no enunciado. As diversas estatísticas são acedidas via API REST implementada no diretório.

Feito o login, aparecerá um novo menu (fase MAIN_MENU). Neste menu o cliente pode atualizar a sua localização, visualizar o número de pessoas numa dada localização, comunicar que se encontra infetado ou ativar e desativar notificações. Se o cliente ativar a opcção de infetado, é enviada essa informação para o fontendserver e aparecerá um novo menu (fase BLOCKED), onde o cliente pode, caso recupere, inverter o seu estado. Se o cliente selecionar a opção de ativar ou desativar notificações, aparecerá um novo sub-menu (fase NOTIFICATIONS) com os quatro tipos diferentes de notificações possíveis.

Na subscrição ou no cancelamento de subscrições, o objeto da classe `AuthenticatedUser`, instanciado quando o cliente faz login, contém o *socket* SUB usado para receber as informações subscritas, *socket* PUB usado para comunicar com o *broker* quando existe alterações nas configurações das notificações e um *HashMap* que explicita se cada notificação está ou não subscrita pelo cliente. No momento da autenticação com sucesso do cliente, é ainda inicializada uma *thread* `Notifications` que receberá e analisará as notificações. Assim, quando um cliente seleciona uma notificação, se esta já se encontra ativa, é informado o *socket* PAIR com intenção de desativar a subscrição e é enviada a respetiva informação para o servidor *front-end*. Se a notificação ainda não se encontra ativa, então é enviada uma mensagem ao servidor *front-end* com informação da adesão da subscrição e é informado o *socket* PAIR com intenção de subscrever a respetiva notificação.

3.3 Servidor *Front-End*

O servidor *front-end*, como seria de esperar, responde a todas as ações do cliente. Serve de intermediário entre clientes e servidores distritais, permitindo ao mesmo fazer as operações necessárias, ou seja:

- **Registo**: Operação trivial sem nada a explicar, apenas recolhe dados do utilizador.
- **Login**: Recolhe dados do utilizador e, caso o login seja válido, são enviadas ao utilizador as suas notificações ativas.
- **Comunicar que está doente**: Envia ao *district server* do utilizador em causa que esse utilizador está doente.
- **Subscrever/não subscrever uma notificação**: Esta operação serve apenas o propósito de persistir as subscrições de um cliente, permitindo que quando o mesmo faz logout, não perca os seus tópicos de interesse.
- ***Update* da localização**: Informa o *district server* correspondente acerca desta mudança de localização.
- **Obter o número de pessoas numa localização**: Envia para o *district server* correspondente o pedido desejado, sendo esse que vai acabar a operação.

É de notar que este servidor é capaz de comunicar com o *back-end* através dos tipos de *sockets* mencionados na Arquitetura. As mensagens, como mencionado anteriormente, são tratadas com Protocol Buffers, facilitando imenso a forma como se reage a cada evento, uma vez que a única coisa

que vai variar entre as várias operações é o tipo de mensagem quer externamente, nomeadamente na autenticação ou operação génerica (comunicar doença, etc...), como internamente (autenticação varia entre login e registo, por exemplo).

3.4 Servidores Distritais

Cada servidor distrital mantém os dados relevantes para os utilizadores daquele distrito, nomeadamente, localizações passadas e atual e, também, número e lista de pessoas que estão/estiveram em cada localização.

Todas as movimentações de utilizadores, assim como as suas infeções, chegam a estes servidores através do servidor *front-end*, como mencionado anteriormente, alterando então os valores acima mencionados. Aquando destas operações, trata-se também de atualizar a REST API, garantindo assim que os dados públicos se encontram válidos em tempo real.

Além do mencionado, numa localização de um dado distrito, deixam de haver pessoas, ocorre o aumento da concentração de pessoas ou há a diminuição da concentração de pessoas, o servidor distrital envia uma notificação a todos os clientes subscritos que subscrevem esse tópico através dos *sockets* do tipo PUB que comunicam essas mesmas publicações para o *Broker*.

Para além das notificações públicas o servidor distrital na ocorrência de um novo infetado irá devolver uma lista de contactos do utilizador, através do socket PUSH para o servidor de Frontend garantindo assim que depois o servidor de frontend trata de enviar essas mesmas notificações a user's registados e online.

3.5 *Broker*

A implementação de *broker* no nosso sistema tem como objetivo intermediar a interação entre *publishers*, isto é, os servidores distritais, e *subscribers*, os clientes.

Os clientes podem subscrever a várias ocorrências numa dada localização de um distrito ou num distrito. No momento em que ocorre um destes acontecimentos, o servidor distrital envia uma notificação a todos os clientes subscritos a esse distrito. Caso os servidores distritais possuam um número elevado de subscritores, o envio das notificações diretamente aos clientes iria exigir um grande esforço por parte de cada servidor. Assim, de forma a diminuir a carga de cada servidor distrital aquando do envio das notificações aos clientes subscritos, implementaram-se *brokers*, que recebem a informação e encontram-se ligados a conjuntos de clientes, ficando responsáveis por enviar as notificações aos que se encontrarem subscritos àquele tópico.

Cada *broker* possui *sockets* do tipo XSUB e XPUB. A escolha deste tipo de *sockets* deve-se ao facto de existirem múltiplos *publishers*. Além disso, a biblioteca ZeroMQ encaminha as subscrições dos subscritores aos *publishers*. Estes tipos de *sockets* comportam-se como os do tipo SUB e PUB, no entanto, os utilizados no *broker* expõem as subscrições como mensagens especiais. O *proxy* transmite estas mensagens de subscrição do lado do subscritor para o lado do *publisher*, ao lê-las do XSUB e escrevendo-as para o XPUB.

O envio da notificação é, então, feito pela conexão entre o *socket* PUB do servidor distrital e o XSUB do *broker*. De seguida, o *proxy* transmite a mensagem recebida, lendo-a do XSUB e escrevendo-a para o XPUB. Por último, a notificação é enviada através da conexão entre o XPUB presente no *broker* e o *socket* do tipo SUB constituinte do cliente.

4 Conclusão

O presente trabalho prático consistiu no desenvolvimento de um protótipo de uma plataforma para suporte de rastreio de contactos e deteção de concentração de pessoas. Com a sua realização foi possível consolidar a aprendizagem na unidade curricular de Paradigmas de Sistemas Distribuídos, nomeadamente no planeamento e implementação de sistemas distribuídos através da combinação e composição de componentes de *middleware* e de computação em nuvem.

Na realização do presente projeto foi possível cumprir todos os requisitos e funcionalidades propostos para o projeto.

De modo a generalizar a aplicação, foram concebidos ficheiros JSON para configurar todos os endereços dos diferentes componentes aplicação. Foram, também, utilizados Protocol Buffers que permitiram garantir que os dados entre as diferentes linguagens eram lidos de igual modo.

Quando o cliente efetua logout as subscrições antes requisitadas não são perdidas, uma vez que para cada cliente estas foram armazenadas no servidor *front-end* e, assim, quando este se autentica, as subscrições são carregadas na aplicação cliente.

Ainda, de modo a melhorar o desempenho do sistema, implementaram-se *brokers* com *sockets* do tipo XSUB e XPUB, que distribuem as notificações recebidas dos servidores distritais pelos clientes subscritos, o que permite diminuir a carga dos servidores.

Considera-se que a única limitação do projeto é o facto de existir apenas um servidor *front-end*, representando um *bottleneck* no sistema, por isso, como trabalhos futuros, propõe-se fazer a replicação deste servidor para escalar o sistema e fazer balanceamento da carga, garantindo uma melhor performance.

Em suma, considera-se que foi possível aplicar todo o conhecimento adquirido ao longo do semestre e desenvolver um sistema distribuído através da combinação e composição de componentes de *middleware* e de computação em nuvem.