Faculdade de Engenharia da Universidade do Porto

Rua Roberto Frias, sn, 4200-465 Porto, Portugal

Server chat

Sistemas distribuídos

Relatório - TP2

Grupo: t4g06
Ana Cláudia Fonseca Santos - 200700742
Filipe Joaquim de Oliveira Reis Coelho - 201500072
Rui Emanuel Cabral de Almeida Quaresma - 201503005
Tiago Duarte Carvalho - 201506203

27 de Maio de 2018

Introdução

No âmbito da unidade curricular de Sistemas distribuídos foi-nos pedido o desenvolvimento de uma aplicação que permitisse a exploração de API's de serviços da Web. Deste modo, decidimos utilizar a Facebook API por implementar um dos padrões de autorização da web, OAuth 2.0, que maximiza a segurança e a simplicidade de implementação, e por considerarmos ser muito importante nos dias de hoje aprendermos a trabalhar com esta tecnologia e permitir assim a sua integração em futuros projetos que possamos desenvolver.

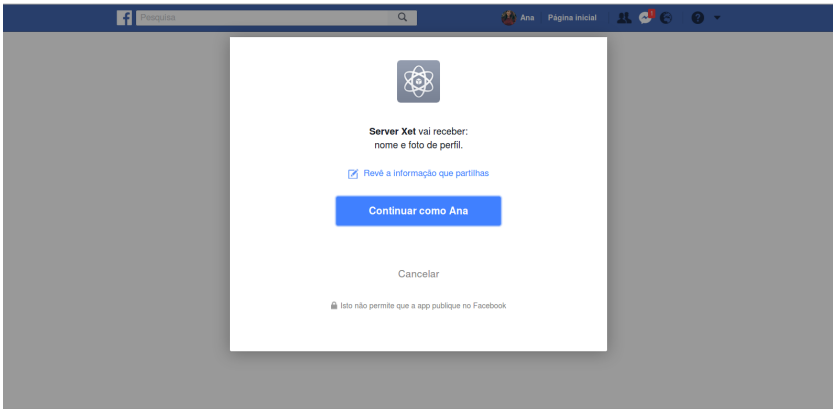
A aplicação desenvolvida é um chat que permite que trocar mensagens entre utilizadores. Existem salas já disponíveis, mas também permite que os utilizadores criem as suas próprias salas e que convidem outros para se juntarem a eles. Para se autenticar o utilizador pode fazer login através do facebook, ou então, entrar como guest. Esta aplicação é muito intuitiva permitindo ao utilizador perceber a cada passo o que está a fazer. As imagens seguintes ilustram o seu aspeto assim como a sua acessibilidade. A figura 4 e a 2 mostram os passos que o utilizador passa ao realizar a autenticação. No caso de utilizar a API externa do facebook é pedido que reintroduza a password e que dê acesso da aplicação aos dados pessoais, visto que é a partir dessa autorização que temos acesso a dados como nome de utilizador, por exemplo. Os dados partilhados com o servidor estão explícitos na janela de autenticação.

(a) Escolha de Login **(b)** Escolha de Username

Figura 1: Exemplo de login como Guest

Como neste projeto quisemos dar várias possibilidades ao utilizador no passo seguinte é mostrado o menu presente na figura 3, deste modo, ele pode escolher uma das salas já pré-definidas, criar a sua própria sala (privada ou pública) ou então entrar numa já criada por outro utilizador para a qual foi convidado. É de notar que apenas os utilizadores autenticados (isto é,

(a) Reintroduzir a password do facebook.



(b) Dar autorização à aplicação.

Figura 2: Exemplo de login por facebook.

não guests) é que podem criar salas, pois é a identificação deles que fica associada como dono da sala. Isto é importante para a funcionalidade de apagar as salas, função que só pode ser ativada caso seja requisitada pelo dono da sala.

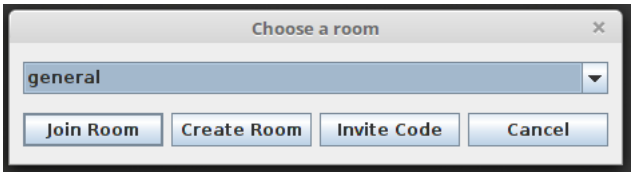
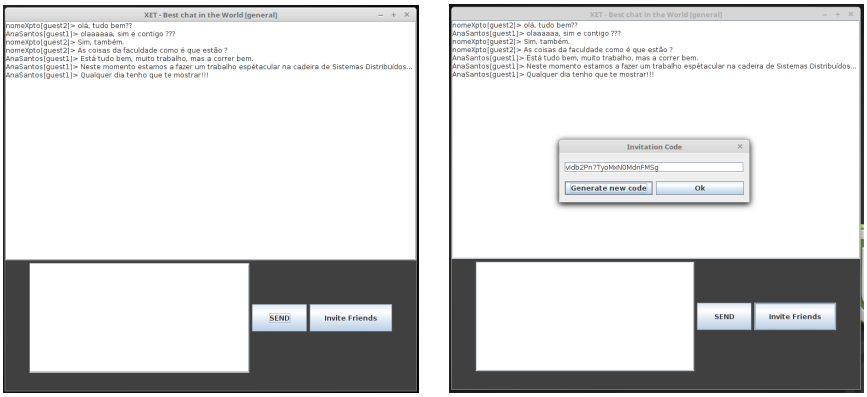


Figura 3: Menu para escolher a sala.



(a) Chat **(b)** Convidar utilizadores

Figura 4: Exemplo de interação entre dois utilizadores e forma de convidar outros.

Nos seguintes pontos abordaremos melhor a arquitetura utilizada para interação entre cliente-servidor, detalhes da implementação realizada, responder a eventuais questões e por fim a conclusão retirada sobre a realização deste trabalho.

1 Arquitetura

Nesta secção descreveremos cada componente da nossa aplicação e a forma como eles interagem. Esta aplicação distribuída foi realizada utilizando uma arquitetura baseada em *REST* (Representational State Transfer), ou seja, é um protocolo cliente-servidor sem estado, em que cada mensagem *HTTP* contem toda a informação necessária para compreender o pedido e desta forma nem o cliente nem o servidor necessitam gravar o estado das comunicações entre mensagens. Para isto são usadas já operações definidas como *POST, GET, PUT e DELETE,* assim como uma sintaxe universal para identificar cada recurso: no sistema *REST* cada recurso é identificado através do seu *URL*. Deste modo, quando o servidor é iniciado são criados os url's para cada tipo de pedido efetuado e o contexto para cada um deles definido. Estes endereços são do seguinte formato: *<endereço do servidor>:<porta>/<url-correspondente>*. Por exemplo, quando o utilizador realiza o login é feito um pedido *POST* ao servidor, que responde com as salas disponíveis. Posteriormente, depois de escolher a sala é realizado um novo pedido *HTTP* por método *POST* com o nome da sala escolhida. Quando o servidor recebe este pedido escolhe uma porta livre (a começar na porta 6000), que envia para o cliente para estabelecer uma ligação *SSLSocket* com este. O *SSLSocket* permite a criação de um socket com uma camada de segurança ao protocolo de transporte de rede subjacente. A camada de segurança passa por proteção de integridade (*SSL* protege contra modificação de mensagens), os servidores geralmente são autenticados e os clientes podem ser autenticados conforme solicitado pelos servidores e confidencialidade: SSL criptógrafa os dados enviados entre o cliente e o servidor. Assim, nesta aplicação, a ligação *SSLSocket* entre cliente-servidor para atualizar o estado de cada sala corresponde a uma ligação segura.

A figura 5 exemplifica a interação entre cliente-servidor, na medida que *1* significa os pedidos *HTTP* e *2* uma ligação *SSLSocket*, tal como exemplificado no seguinte esquema:

- Ligação
  - HTTP (1)
    - autenticação por facebook - "../auth/fb"
    - autenticação por guest - "../auth/guest"
    - pedido das salas disponiveis - ".../handshake"
    - escolha de sala - ".../room"
    - nova mensagem - ".../message"
    - convite para uma sala - ".../room/invitation"
  - SSLSocket (2)
    - Atualização do estado da sala

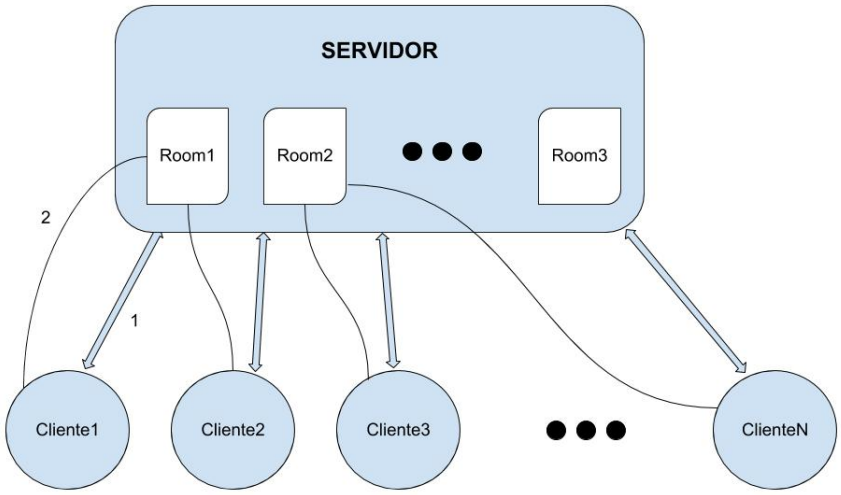


Figura 5

2 Implementação

A aplicação foi totalmente implementada na linguagem de programação *JAVA*. As bibliotecas usadas foram apenas as fornecidas pela *Java Standard Library* e não foram usadas quaisquer *Frameworks* externas. Portanto, tanto o *HTTPServer* como a implementação do procedimento de *OAuth* para autenticação via *Facebook* foram completamente implementados por nós.

Dado que a arquitetura da aplicação foi desenvolvida em *REST*, problemas relacionados com a concorrência não são verificados, uma vez que não há estados guardados no lado do servidor. Para além disso cada pedido corre na sua própria *thread* deste modo, mesmo que cheguem vários pedidos em simultâneo esta arquitetura está preparada para os satisfazer.

O protocolo *OAuth 2.0* contém várias implementações de autenticação e autorização para acomodar os mais variados tipos de aplicações, como web, nativas, servidor a servidor, etc. Neste caso, a implementação utilizada foi a padrão, também conhecido como *Authorization Code Flow*, que é também a mais segura. Ela funciona em 3 passos distintos:

1. Do lado do cliente é aberto um browser no domínio do fornecedor do protocolo (neste caso, o *facebook*) onde o *URL* contém contém informação como qual a aplicação que está a fazer o pedido, o endereço de redirecionamento após o login efetuado e um código de validação. O cliente deve consultar as permissões que a aplicação lhe pede e aceitar para proceder com o processo de autenticação.

2. De seguida, a página é redirecionada para o endereço de redirecionamento especificado no *URL* anterior (e este endereço tem de estar registado no fornecedor do protocolo), onde é enviado um código de utilização única que juntamente com uma chave secreta de identificação da aplicação pode ser usado para obter um *token* que fornece acesso aos dados do cliente (conhecido por *Access Token*). O servidor envia então um pedido ao fornecedor do protocolo com esses dados e recebe o *Access Token*.

3. Usando o *token* recebido, o servidor envia novo pedido ao servidor que contém os recursos do utilizador (no nosso caso, continua a ser o *Facebook* e os recursos são apenas o nome e um identificador). Tendo acesso a essa informação, o cliente agora comunica sempre com o servidor utilizando o código

de verificação enviado no passo 1 como identificador, para garantir assim que não houve modificação de mensagens por parte do fornecedor do protocolo.

É preciso referir que, como este protocolo envolve redirecionamento para o servidor e como o servidor não está hospedado num local público, o unico endereço de redirecionamento disponível para testes é o *localhost*, ou o *127.0.0.1*. Portanto, devido a esta limitação, apenas o computador que está a correr o servidor pode fazer autenticação pelo facebook, sendo que os outros terão de se autenticar como *guests*. Também é possível iniciar vários clientes no mesmo computador e iniciá-los com contas diferentes, permitindo assim testar as várias funcionalidades da aplicação.

Um outro caso particular é que, novamente devido ao servidor não ser público e, por consequência não ser utilizar este do protocolo sem ser em modo de desenvolvimento, o acesso ao uso deste é apenas permitido a contas autorizadas. Para faciliar os testes foram criados 4 contas de teste, cujas credenciais poderão ser encontradas no *README*.

3 Questões relevantes

O protocolo SSL permite garantir a integridade das mensagens e garantir a confidencialidade das mensagens, deste modo para as notificações em cada utilizador de mensagens novas foram usados SSLSocket's. Estas propriedades de canais seguros são obtidas recorrendo a algoritmos e protocolos criptográficos. De forma a concordar os serviços e os parâmetros a usar em cada canal, SSL especifica um protocolo de handshake que deverá ser executado pelas duas entidades comunicantes antes de iniciar a troca de mensagens de aplicação propriamente dita. Este protocolo implementa canais seguros a nível da camada de transporte. O seu uso envolve três operações:

- Estabelecimento da conexão TCP;
- Execução do protocolo de handshake.
- Transferência de dados.

Por questões de arquitetura da aplicação, preferimos fazer esta conexão após o utilizador escolher a *Room*, desta forma cada sala funciona como um pequeno servidor capaz de atualizar os utilizadores atuais aquando a chegada de novas mensagens.

As várias funções limitadas a um certo tipo de utilizadores (como a criação de salas que está limitada a utilizadores autenticados e a remoção da mesma, que está limitada ao criador) são todas validadas do lado do servidor. Enquanto que do lado da aplicação do cliente existe sempre opções que são desativadas caso o utilizador não tenha permissões para as executar, haveria sempre a possibilidade de enviarem pedidos de outras formas na tentativa de lubridiar o sistema, há sempre uma validação efetuada no lado do servidor. A forma que isto acontece é, sempre que um cliente se autentica no servidor, ele envia um código secreto, sempre diferente mesmo que entre com a mesma conta. O servidor regista esse código e no caso da autenticação ter sido efetuada pelo *Facebook*, o servidor recebe também um identificador da conta do facebook que fica associada a ações privilegiadas daquele cliente. Quando o cliente efetua alguma comunicação com o servidor, envia sempre o código aleatório que gerou quando fez a autenticação e nunca outro tipo de identificação. Para ser mais fácil a compreensão, vamos especificar de forma muito simplificada os passos do processo de criação de salas:

1. O cliente gera um código aleatório;

2. O cliente autentica-se pelo *Facebook*;

3. O servidor guarda o código do cliente (sempre diferente entre autenticações) e o identificador da conta do *Facebook* (sempre igual entre autenticações);

4. O cliente faz um pedido para criar uma nova sala, enviando o seu código gerado;

5. O servidor verifica se àquele código está associado um utilizador autenticado e não um guest;

6. Se for um utilizador autenticado, cria uma nova sala e regista como proprietário o identificador da conta do *Facebook*;

7. O servidor devolve para o cliente o nome da sala criada;

8. O cliente tenta juntar-se à sala utilizando o processo normal;

Foi também implementado um sistema que guarda as salas criadas bem como o seu proprietário e os utilizadores autorizados a entrar. Desta forma, mesmo que o servidor falhe por algum motivo, quando for reiniciado as salas voltarão ao ativo da forma que se encontravam.

4 Conclusão

O uso da arquitetura *REST* nesta aplicação permitiu de uma forma muito simples fazer uma aplicação em que a implementação entre cliente-servidor é completamente independente, segura e independente da plataforma usada. Apesar de ser uma arquitetura a evitar quando é necessário guardar informação de forma persistente, revelou-se muito útil para este tipo de desafio.

Em suma, realização projeto permitiu trabalharmos com tecnologias e protocolos muito atuais e necessárias para o futuro e pensamos que de uma forma geral cumprimos o essencial para este projeto, mas uma vez que foram implementados de raíz deu para ter uma ideia bastante clara de qual é o seu funcionamento e como poderão ser integrados de forma a tirarmos o máximo proveito das suas virtudes.