Relatório Final
NotGoogleDocs

Sistemas Distribuídos

3º ano do Mestrado Integrado em Engenharia Informática e Computação

Ana Moura - 201201786

David Caminha - 201203963

João Soares - 201206052

Mafalda Falcão - 201204016

5 de junho de 2015

Índice

- Índice
- Introdução
- Arquitetura
- Implementação
  - Sistema Client-Server
  - Chat
- Problemas Relevantes
  - Segurança
  - Tolerância a Falhas e Conflitos
  - Escalabilidade
- Conclusão

Introdução

Este projeto consiste num editor de ficheiros de texto partilhados, em tempo real. Com esta aplicação, será possível a edição do um ficheiro por diversos utilizadores, acompanhando e obtendo as atualizações dos outros utilizadores quando estas são feitas.

Na concepção desta ideia, baseamo-nos no Google Docs, uma aplicação Web de edição e partilha de ficheiros,conhecida e utilizada por todos nós. Para além da edição de texto, adicionamos a ideia de um chat (troca de mensagens instantâneas) entre os utilizadores que estão a editar o mesmo ficheiro, para facilitar o trabalho em grupo.

Esta aplicação consiste num sistema distribuído ***Client-Server***, baseado no paradigma ***REST***, em que o controlo de versões e acesso dos utilizadores é controlado pelo servidor. Após a aprovação do servidor, é apresentada uma interface gráfica ao utilizador (Client) para leitura e edição de texto em ficheiros partilhados, com uma janela de chat associada. Aí, o utilizador poderá efetuar as alterações desejadas ao ficheiro e submete-las para o servidor. O Servidor irá guardar as alterações recebidas num registo **LOG** e atribuir-lhe um **TimeStamp** relativo a quando estas foram recebidas. Os outros utilizadores que estejam a editar o mesmo documento devem periodicamente interrogar o servidor por novas alterações, enviando o TimeStamp da ultima alteração que receberam, e aplicar as alterações recebidas ao ficheiro que lhes é apresentado na GUI.

Arquitetura

Tal como já foi referido anteriormente, o Real Time Text Editor é uma aplicação que integra um sistema distribuído Client-Server, baseado no paradigma *REST*.

O projeto está dividido em dois módulos:

- Client - Implementação da estrutura lógica e gráfica para a criação de um Client.

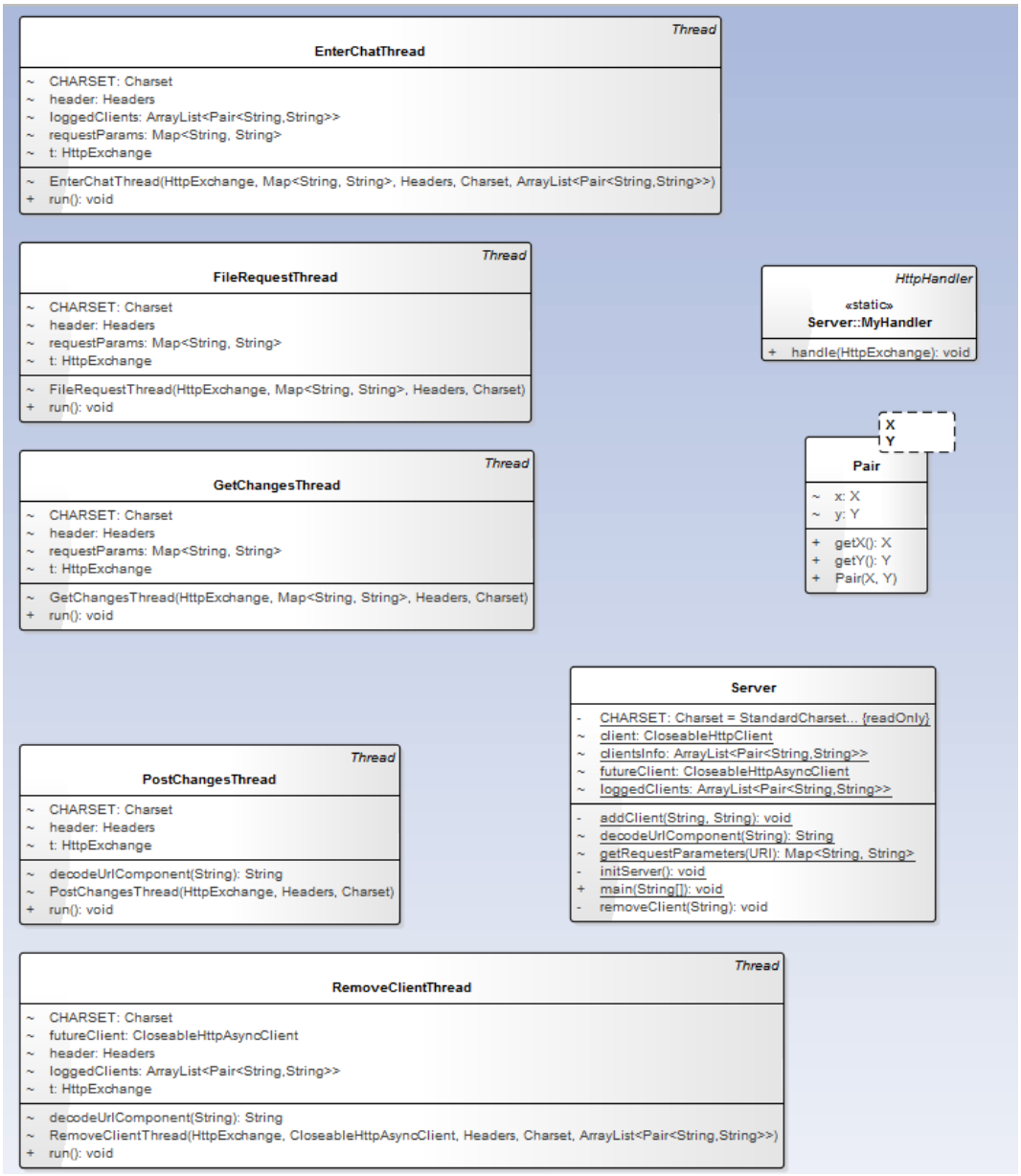
- Server - Implementação da estrutura lógica do servidor.

Cada um é composto pelos seguintes ficheiros:

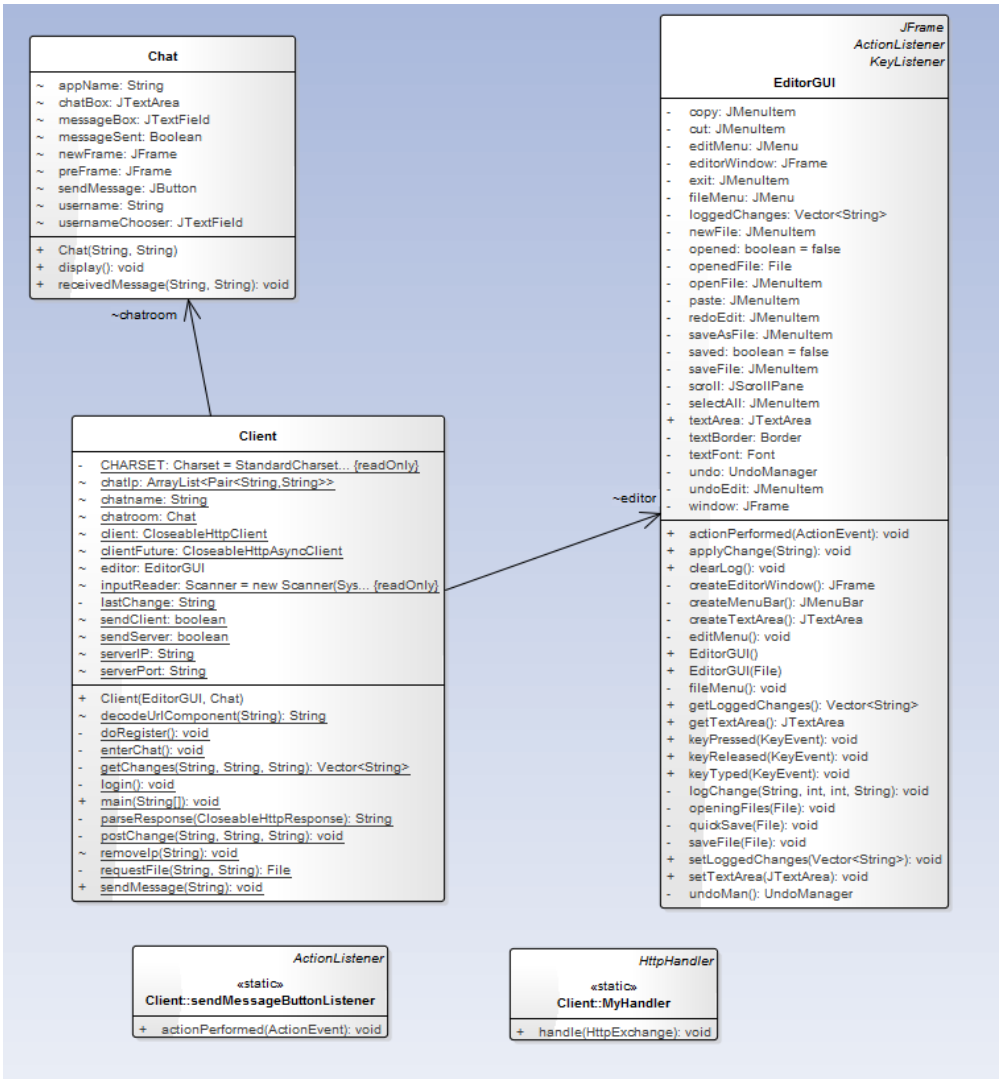
- Client:
  - Chat.java - implementação da parte gráfica, relativa à estrutura e funcionamento da janela de chat;
  - Client.java - "main" do cliente, onde se invoca a aplicação, por parte do(s) cliente(s) (junta a parte lógica (de pedidos http, conexão com o servidor e ao chat) com a parte gráfica presente no ficheiro EditorGUI.java;
  - EditorGUI.java - implementação da parte gráfica, relativa ao corpo do editor de texto.

- Server:
  - Pair.java - criação de um par de dados, que vai ser importante para as utilidades de login e register implementadas na aplicação;
  - Server.java - "main" do servidor, onde se invoca a aplicação, por parte do servidor.

A estrutura acima descrita poderá ser vista, nos diagramas UML seguinte:



(Package server)

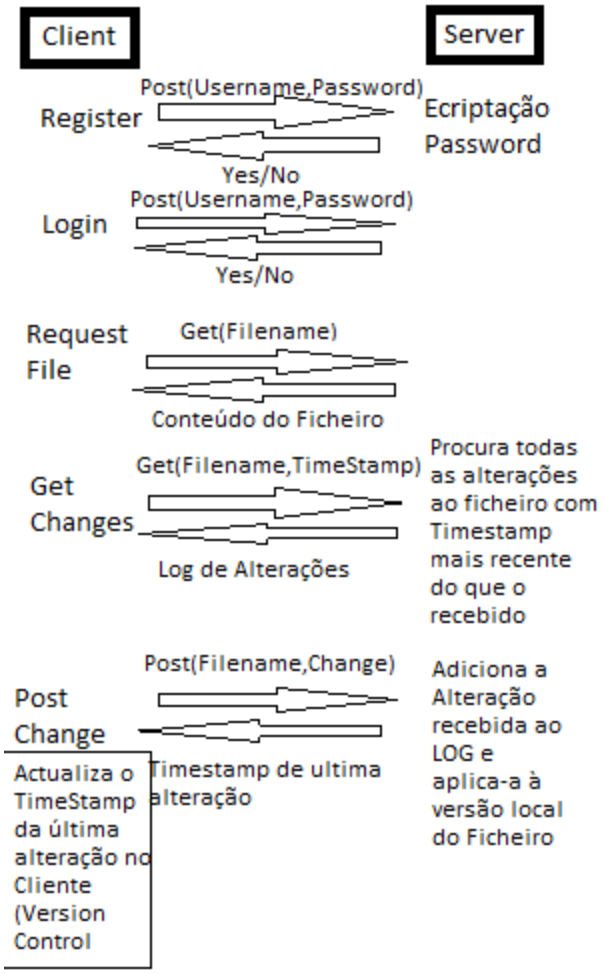


(Package client)

Implementação

Posteriormente, iremos falar dos procedimentos implementados na nossa aplicação, em linguagem Java.

- **Sistema Client-Server**



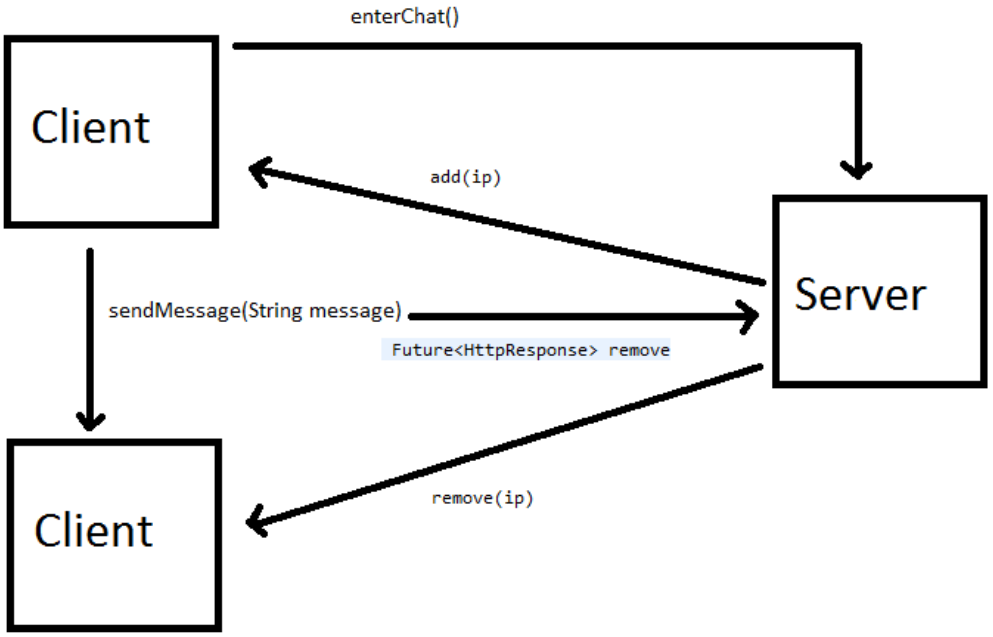
O Servidor foi implementado utilizando a biblioteca HTTPServer disponibilizada pela linguagem Java. O seu funcionamento baseia-se numa função *Handler* que recebe e trata todos os pedidos efetuados ao contexto "/sdisdocs", efetua o *parsing* dos pedidos recebidos e trata-os de acordo. A resposta aos pedidos é feita por lançamento de uma *Thread,* para evitar os

bloqueios no lado do servidor (exceto no caso do *Register* e *Login*). A encriptação das passwords é tratada com a biblioteca de *Java Security* e a encriptação aplicada é do tipo *SHA-256*.

O Client foi implementado utilizando a biblioteca *HTTPClient* da *Apache*. O seu funcionamento inicia-se com uma UI, pela consola, onde o utilizador escolhe registar-se ou fazer login. Seguidamente, este envia um pedido ao servidor para editar um ficheiro partilhado, recebe o contéudo deste e abre uma janela gráfica de edição de texto. O editor de texto foi implementado em *SWING* e contém um *DocumentListener*, que deteta alterações ao conteúdo do mesmo e que despoleta o pedidos de atualização ao servidor.

Os pedidos para os métodos *GET* e *POST* são similares, no entanto, no método GET os parametros são enviados directamente no URL e no método *POST* são enviados atravéz do *request body*. Os formatos são idênticos (conjunto de pares separados pelo caractere "&"). Para poder enviar caracteres especiais é necessário utilizar a funcionalidade *encoding* da biblioteca *java.net* As respostas são enviadas em formato JSON e são analisadas usando a biblioteca *org.json*.

- **Chat**



A funcionalidade do chat funciona como um subsistema dentro da nossa aplicação, pois tem as suas próprias comunicações bem como próprios handlers, este subsistema ao contrário do resto do projeto não vai de acordo com o paradigma REST.

Não existe necessidade de resolver problemas de concorrência pois um pedido não influência outro. Todos os pedidos dentro do chat são feitos assincronamente, podendo funcionar em paralelo e o primeiro pedido a ser resolvido será o primeiro a ser demonstrado na GUI, não influênciando em nada a continuação do funcionamento do mesmo.

Para esta implementação, foram usadas as bibliotecas *httpasyncclient-4.1*, *httpasyncclient-cache-4.1*, *httpcore-nio-4.4.1*, necessárias para a implementação assíncrona do cliente.

Um cliente, a partir do momento que faz login, envia um pedido get ao servidor para receber os ips de quem está presente no chat, espera resposta com array de strings contendo os ips.

O envio de mensagens dentro do chat é feito por pedidos *httpPost*, não esperando qualquer tipo de resposta; apenas foi implementado um *FutureCallback<HttpResponse>(),* para que se pudesse receber se a mensagem foi enviada com sucesso, com insucesso ou se a mesma foi cancelada.

Se uma mensagem for enviada sem sucesso, é feita uma ligação síncrona ao servidor, que apenas espera uma resposta de acknowledge, para que o servidor seja informado que um cliente não responde. Após essa informação, o servidor manda um pedido *httpPost* assíncrono, para que os clientes saibam que é para remover um dado utilizador, esta informação aos clientes não espera também qualquer tipo de resposta.

Pontos Fortes:

- ➔ A implementação descentralizada do chat, faz com que o mesmo tenha uma capacidade de se manter online, mesmo numa possível falha do servidor, fazendo com que os clientes já conectados possam usufruir do chat sem problemas.
- ➔ O chat apresenta mensagens de sistema caso o próprio utilizador tenha se desconectado de todos os outros clientes e servidor.
- ➔ O chat apresenta mensagens de sistema a todos os clientes, sempre que um cliente pare de responder.

- ➔ A implementação assíncrona do chat permite que os clientes comuniquem sem necessitarem de esperar pelas respostas uns dos outros eliminando assim possiveis conflitos/concorrencias.

Problemas Relevantes

Para garantir o desenvolvimento e eficácia da nossa aplicação, há que ter especial atenção a determinados aspetos, de maior relevância, tais como:

- **Segurança**

O acesso é feito a partir de um login (username + password) cujas informacoes estão guardadas no servidor. A password é encriptada de forma a garantir a procteção da conta do utilizador.

- **Tolerância a Falhas e Conflitos**

Caso o servidor vá abaixo este guarda sempre o ultimo estado em que estava e como os clientes guardam as suas alteracoes num vector podem continuar a escrever e se o servidor recuperar as alteracoes serão enviadas. Caso um cliente falhe visto que o vector de alterações é verificado constantemente, este só perde as alterações que estava a enviar no momento minimizando assim as perdas.

- **Escalabilidade**

Como o servidor cria threads para responder aos pedidos que vão chegando é possivel ter varios clientes a enviar pedidos ao servidor sem que este tenha de esperar para responder ao seguinte garantindo assim uma velocidade aceitavel de edição.

Conclusão

Em modo de conclusão, tendo em conta toda a complexidade envolvida na implementação, pensamos que alcançamos todos os objetivos inicialmente propostos.

Não foi um trabalho fácil, mas foi-se compondo incrementalmente, até à data de entrega, o que deixa, em todos nós, um satisfação e gratificação imensas.

Nenhum de nós tinha feito nenhuma aplicação, neste âmbito. Por isso, é inevitável realçar a nossa aprendizagem acerca dos protocolos, paradigmas, bibliotecas utilizadas e novos conceitos e temos a certeza que todos esses conhecimento irão ser bastante úteis, num futuro próximo.

Relativamente ao trabalho desenvolvido, dividimos em quatro grandes módulos em que cada elemento era responsável pelo seu planeamento. O desenvolvimento de cada um dos módulos foi iniciado individualmente e, numa segunda parte, o grupo reuniu-se e dividiu as tarefas restantes de todas as partes para que todos estivessem a par do desenvolvimento de cada módulo.

- *Ana Moura* - Desenvolvimento do chat assíncrono Client-Client e Client-Server.
  - **Participação: 25%**
  - **Contribuição: 25%**
- *David Caminha* - Desenvolvimento da parte do Server, no sistema Client-Server.
  - **Participação: 25%**
  - **Contribuição: 25%**
- *João Soares* - Desenvolvimento da parte do Client, no sistema Client-Server.
  - **Participação: 25%**
  - **Contribuição: 25%**
- *Mafalda Falcão* - Design da Aplicação, tanto do editor de texto como da janela de chat.
  - **Participação: 25%**
  - **Contribuição: 25%**

Possíveis Melhorias

- Multiplos ficheiros suportados pelo servidor reservados a clientes específicos;
- Janelas de Chat individuais para cada ficheiro suportado;

- Multiplos Servidores e adaptação a funcionamento do tipo Cloud;
- Melhoramentos na acessibilidade da GUI e melhoramentos visuais;
- Funcionalidades Extra na edição de texto;