Instituto Politécnico do Cávado e do Ave

João Azevedo – 18845

Carlos Santos – 19432

João Rodrigues –19431

Comunicação de Dados – Serviços Web

Barcelos, junho 2021

Resumo

O objetivo deste Trabalho Prático é estudar o funcionamento de uma API REST,

criar uma aplicação servidor capaz de correr serviços web, criar uma aplicação que faça uso da API de serviços definida e apresente uma interface ao utilizador. Um serviço Web que consiste na troca de mensagens(chat), serviço de transferência de ficheiros e serviço de gestão de utilizadores.

Iram ser feitas descrições sobre como foi interpretado o enunciado proposto, quais as decisões tomadas durante a implementação assim como breves explicações do código desenvolvido.

No desenvolvimento deste trabalho foi utilizada a linguagem de programação Python, html e javascript. Meter aqui mais coisas relacionadas com o brower

Por fim, pretendia-se que este trabalho, fosse concluído com sucesso co todos os objetivos propostos.

Introdução

Este relatório foi concretizado no âmbito do trabalho prático realizado para a Unidade Curricular de Comunicações de Dados da Licenciatura em Engenharia de Sistemas Informáticos, lecionada na Escola Superior de Tecnologia do Instituto Politécnico do Cávado e do Ave.

As diversas tarefas realizadas neste trabalho proporcionaram a oportunidade de, não só aplicar as competências lecionadas/adquiridas nas aulas, mas também o desenvolvimento de novas.

O presente relatório encontra-se dividido em três partes e aborda a forma como o enunciado foi interpretado, quais as decisões tomadas durante a sua implementação e demonstrações de código com explicação.

Repositório GitHub: meter?

Interpretação do Enunciado

A partir do enunciado e devido á sua clareza, facilmente percebemos quais funcionalidades teríamos de implementar e quais estruturas de dados seriam necessárias para o fazer.

Dito isto, foi feito um servidor onde tivesse todas as funcionalidades descritas no enunciado.

Adicionalmente criamos um web brower

Decisões Tomadas

A partir da interpretação anterior, foi decidido que a primeira tarefa a realizar seria planear as estruturas que iriamos usar assim como idealizar o funcionamento geral do projeto.

Após isso, foram criadas as estruturas de dados necessárias para que fosse possível não só armazenar dados, mas também comunicar entre cliente e servidor.

Depois, foi iniciado o desenvolvimento da interface visual básica necessária para o envio e receção de mensagens, juntamente com o código básico relativo à troca de pacotes entre cliente e servidor.

Estrutura de Dados



As Estruturas de Dados escolhidas para utilizar foram listas de dicionários, estas listas vão conter vários dicionários de utilizadores e mensagens. Como exemplos temos o caso dos ‘users’ esta lista possui um dicionário com as chaves ‘username’ e ‘password’ os seus valores são ‘ricardo’ e ‘18845’ respetivamente. Por outro lado, nas ‘messages’ as chaves são ‘id\_sms’ representa um identificador da mensagem, ‘sms’ representa a mensagem enviada pelo utilizador, ‘sender’ representa quem enviou a mensagem ‘receiver’ representa o/s utilizador/es que devem receber a mensagem. Por fim o ‘status’ identifica se a mensagem foi ou não lida pelo/s recetores.

1. Serviço de troca de Mensagens de Texto Persistentes

Uma API de serviço para troca de mensagens de texto persistentes entre vários utilizadores, estas mensagens devem ser guardadas numa caixa de mensagens persistentes por utilizador. Cada utilizador depois de autenticado deve ser capaz de:

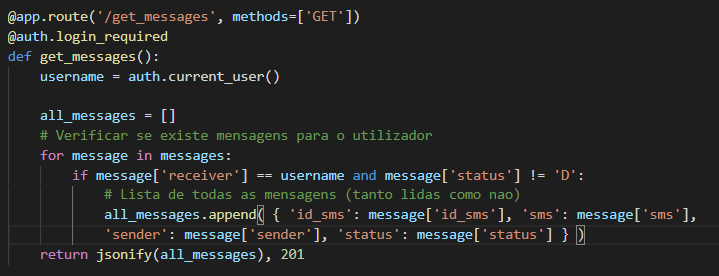
* Enviar uma mensagem para um utilizador (ou grupo de utilizadores).
* Consultar as mensagens entretanto recebidas na caixa de mensagens. Distinguir entre as mensagens já lidas e as mensagens novas.
* Uma mensagem enviada para um grupo é equivalente a ser enviada para cada utilizador membro desse grupo.
* Listar o número de mensagens na caixa de mensagens.
* Remover uma mensagem da caixa de mensagens.

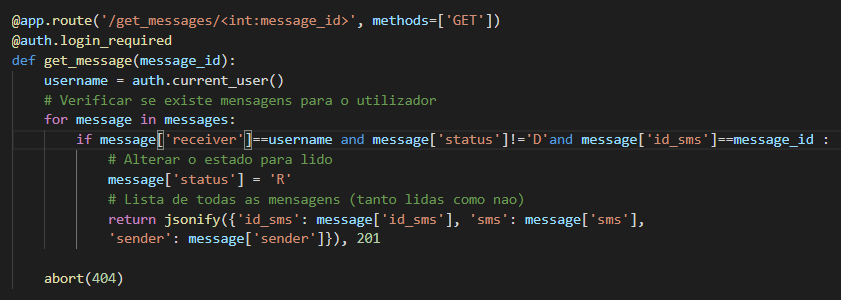


Para enviar uma mensagem para um utilizador ou um grupo de utilizadores uma mensagem, criou-se a rota <http://localhost:5000/send_message> com o método POST.

Primeiramente era necessário fazer o login no servidor através do auth.login\_required, dentro da função verificou com uma condição a existência de um json com os campos ‘message’(possuía a mensagem a enviar) e ‘receiver’(utilizador ou grupo de utilizadores para onde se pretende enviar a mensagem) o campo ‘receiver’ é enviado pelo utilizador com o seguinte formato ‘{”receiver”: “ricardo,Carlos,Alberto” }’ com o split separamos os diferentes utilizadores por vírgulas e assi ter os recetores da mensagem.

Para além disso verificar a existência do utilizador que envia a mensagem e com isto tudo verificado podemos adicionar na estrutura criada de mensagens e guardar esta num ficheiro.



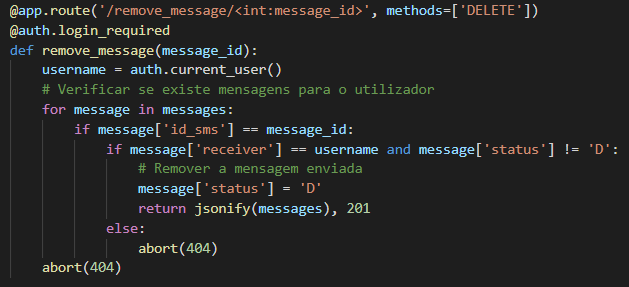


Para consultar todas as mensagens entretanto recebidas na caixa de mensagens criou-se a rota <http://localhost:5000/get_messages> com o método GET.

Primeiramente era necessário fazer o login no servidor através do auth.login\_required, de seguida com auxílio de um ciclo for percorreu-se todas as mensagens dentro da estrutura de dados onde estas se encontravam armazenadas e adicionou-se apenas aquelas que eram direcionadas ao utilizador com ajuda de uma lista auxiliar (all\_message).

Para ler e consultar uma mensagem em específica criou-se a rota <http://localhost:5000/get_message/message_id> com o método GET.

A grande diferença entre esta rota e a anterior encontra-se no estado da mensagem este passa de “N(Not Read)” a “R(Read)” permitindo assim, identificar aquelas mensagens que foram ou não lidas.



Para remover uma mensagem da caixa de mensagem, criou-se a rota <http://localhost:5000/remove_message>/message\_id com o método DELETE.

Primeiramente era necessário fazer o login no servidor através do auth.login\_required, de seguida com um ciclo for percorreu-se a estrutura de Dados ‘messages’ onde continha todas menagens e verificou-se se o utilizador possuía alguma com o id em específico(message\_id) em caso afirmativo o ‘status’ mensagem passava a ‘D’ de deleted ou seja ficava como apaga do sistema para o utilizador, em caso negativo era dado um abort(404) assim como não possuía qualquer mensagem com aquele id.

1. Serviço de troca de Mensagens de Texto Instantâneas

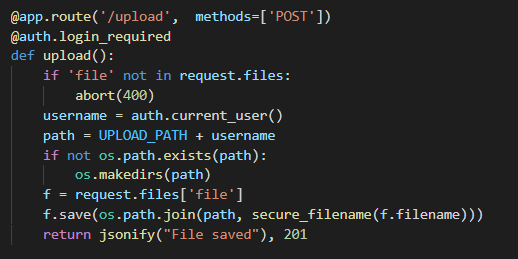
Uma API de serviço para troca de mensagens de texto instantâneas usando canais de comunicação. Uma mensagem enviada para um canal é entregue, imediatamente, a todos os utilizadores registados nesse canal que estão à escuta. Cada utilizador depois de autenticado deve ser capaz de:

* Enviar uma mensagem para um canal específico.
* Registar o próprio utilizador num canal. Este registo é persistente, um utilizador só precisa de registar uma vez no canal.
* Cancelar o registo num canal, i.e., deixar de receber mensagens desse canal.

1. Serviços de Transferência de Ficheiros

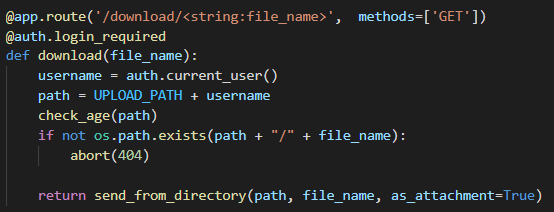
Uma API de serviço para transferência de ficheiros. Os ficheiros são temporariamente guardados numa área de trabalho individual por utilizador. Cada utilizador depois de autenticado deve ser capaz de:

* Carregar um ficheiro do computador local para a área de trabalho.
* Descarregar um ficheiro da área de trabalho para o computador local.
* Listar os ficheiros na área de trabalho.
* Apagar um ficheiro individual da área de trabalho.
* Cada ficheiro na área de trabalho deve ter associado um tempo de vida. Expirado o tempo de vida, o ficheiro deve ser apagado automaticamente do servidor.



Para carregar um ficheiro do computador local para a área de trabalho, criou-se a rota <http://localhost:5000/upload> com o método POST.

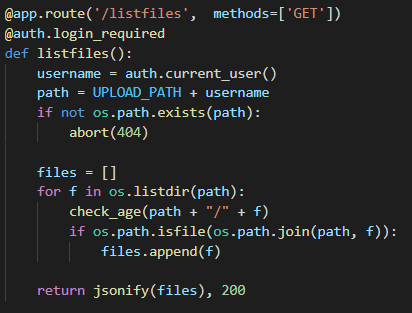
Primeiramente era necessário fazer o login no servidor através do auth.login\_required, de seguida verificar se o campo ‘file’ estava no json enviado pelo cliente, em caso negativo dava abort(400), em caso afirmativo o caminho(path) iria ser ‘UPLOAD\_PATH’ + username do cliente, o ‘UPLOAD\_PATH’ representa uma pasta raiz onde dentro desta eram ser guardadas todas as pastas de cada cliente. Após a criação do caminho verificou-se a existência ou não do mesmo, em caso negativo a sua criação e por fim guardar o ficheiro dentro do servidor.



Para descarregar um ficheiro da área de trabalho para o computador local, criou-se a rota <http://localhost:5000/download/file_name> com o método GET.

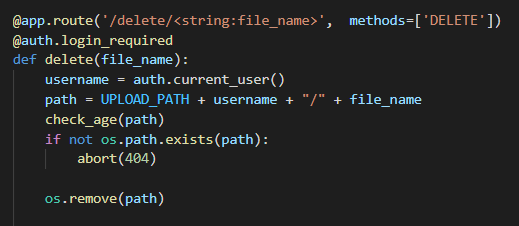
Primeiramente era necessário fazer o login no servidor através do auth.login\_required, de seguida é usada a função ‘check\_age’ para verificar o tempo de vida do ficheiro, ou seja, se o tempo de vida já expirou e caso tenha é removido do servidor.

Caso a função retorne um false, assim como não existe o caminho o servidor retorna um erro ao cliente em caso da função retornar true assim como o caminho existe a função download retorna ao cliente o ficheiro que foi pedido.



Para listar os ficheiros na área de trabalho, criou-se a rota <http://localhost:5000/listfiles> com o método GET.

Primeiramente era necessário fazer o login no servidor através do auth.login\_required, de seguida é verificado a existência do caminho caso exista é criada uma lista(files) auxiliar para guardar todos os ficheiros que estejam na pasta e que não estejam fora do seu tempo de vida (verificado com a função check\_age) por fim é retornado pela função todos os ficheiros que o cliente possui.



Para apagar um ficheiro individual da área de trabalho, criou-se a rota <http://localhost:5000/delete/file_name> com o método DELETE.

Primeiramente era necessário fazer o login no servidor através do auth.login\_required, seguida é usada a função ‘check\_age’ para verificar o tempo de vida do ficheiro, ou seja, se o tempo de vida já expirou e caso tenha é removido do servidor. Depois de verificado esta condição o ficheiro é removido do servidor a pedido do cliente.

1. Serviço de Gestão de Utilizadores

Uma API de serviço para a gestão de utilizadores deste sistema. Esta gestão será feita pelo utilizador administrador, que será o único com acesso a estas configurações:

* Criar utilizadores, atribuindo uma palavra-chave individual.
* Permitir alterar a palavra-chave de um utilizador.
* Remover um utilizador, isto é, remover todas as informações guardadas no servidor para esse utilizador.
* Definir grupos de utilizadores (para mensagens persistentes).
* Definir canais (para mensagens instantâneas).

Conclusão

Durante este trabalho prático, como já falado anteriormente, foi-nos proposto desenvolver uma plataforma de comunicação. Neste desenvolvimento tivemos a oportunidade de aplicar os conteúdos e matérias lecionadas durante as aulas. Com base nas mesmas, na pesquisa em grupo e no conhecimento já adquirido foi possível resolver o enunciado proposto.

Praticamente todo o código desenvolvido foi um desafio para nós, visto que nunca tínhamos desenvolvido algo semelhante a um servidor. No que toca à troca de mensagens entre cliente e servidor, aproveitamos a lógica lecionada nas aulas assim como algum do código. Esse código sofreu algumas alterações necessárias para atingirmos as nossas metas de desenvolvimento.

Estamos perfeitamente mentalizados que algumas das técnicas praticadas nesta solução não serão as mais eficientes. Para terminar, este trabalho prático foi uma peça essencial não só para o nosso desenvolvimento técnico e profissional, mas também para o nosso sucesso na Unidade Curricular.