Computação Distribuída
2019 / 2020

Licenciatura em Engenharia Informática

Trabalho Prático #2 – Serviço de Chat

Introdução

Neste trabalho prático pretende-se construir um serviço de chat utilizando *sockets* TCP/IP. Este serviço é constituído por um servidor, que mantém uma lista de salas e utilizadores activos, e pelos clientes que acedem ao serviço.

Toda a interacção com o servidor será baseada em comandos. O seguinte excerto exemplifica uma interacção entre dois clientes (com uma linha temporal mais ou menos coerente) e o servidor.

```
> /username bob
/username ok
> /room
/room #welcome
> /create #teste
/create ok
> /rooms
/rooms #welcome, #teste
> /join #teste
/join ok
> /room
/room #teste
> /users
/users bob
```

```
> /username ted
/username ok
> /room
/room #welcome
```

```
> /join #teste
/join ok
> /users
/users bob, ted
> /msg Hi everyone!
/msg sent
```

```
> /msgs
/msgs (ted @#teste) Hi everyone!
> /exit
/exit ok
```

```
> /exit
/exit ok
```

Este trabalho tem por base o código fornecido no *Github* e é constituído pelos seguintes ficheiros:

- *server.py* → Contém o servidor na classe *Server* e exemplo de execução no *main*.
- *client.py* → Contém o cliente na classe *Client* e exemplo de execução no *main*.
- *tests.py* → Alguns unittests que validam o funcionamento correcto do servidor.

Funcionamento geral

Após iniciar a aplicação, o servidor deverá aguardar pela ligação dos clientes. Cada cliente deverá providenciar um nome de utilizador único, após o qual será automaticamente redirecionado para a sala *#welcome*. Os utilizadores não poderão prosseguir enquanto não inserirem um nome.

Inicialmente será disponibilizada apenas a sala *#welcome*, mas os clientes poderão criar e entrar noutras salas. As salas deverão ser identificadas pela hashtag "#" seguida de um nome.

As mensagens enviadas para uma sala deverão ser automaticamente reenviadas para todos os utilizadores que lá se encontrem. Também deverá ser possível o envio de mensagens privadas. Estas terão como destino apenas um utilizador em específico.

Para manter o serviço simples assuma-se que os clientes só podem estar numa sala de cada vez.

Lista de comandos

Os seguintes comando e respectivas respostas do servidor deverão ser correctamente implementados e testados.

Autenticação

- **Autenticação no servidor:** `/username <name>`
  - Retorna `/username ok` se o nome estiver disponível no servidor.
  - Retorna `/username taken` se o utilizador já existir.
  - Este comando deve ser executado logo após o início de ligação com o servidor. Caso contrário, o servidor deverá retornar `/username required`.

- **Autenticação no servidor:** `/exit`
  - Retorna `/exit ok`.

Salas

- **Sala actual:** `/room`
  - Retorna o nome da sala onde o utilizador se encontra, ex: `/room #welcome`.

- **Sala disponíveis:** `/rooms`
  - Retorna a lista de salas disponíveis no servidor, ex: `/rooms #welcome, #teste`.

- **Criar salas:** `/create <room>`
  - Retorna `/create ok` se sala não existir.
  - Retorna `/create room_exists` se a sala já existir.

- **Entrar numa sala:** `/join <room>`
  - Retorna `/join ok` se entrar na sala.
  - Retorna `/join no_room` se a sala não existir.

Utilizadores

- **Utilizadores na sala:** `/users`
  - Retorna o nome dos utilizadores na sala, incluíndo o próprio, ex: `/users bob, ted`

- **Utilizadores na sala:** `/allusers`
  - Retorna os nomes de todos os utilizadores e respectiva sala onde se encontram, ex: `/allusers bob@#teste, ted@#welcome, pascal@#teste`

Mensagens

- **Enviar mensagem para sala actual:** `/msg <msg>`
  - O servidor deverá retornar `/msg sent`.
  - Todos os utilizadores na sala (excepto o emissor) deverão receber uma mensagem com o formato `(username @room) <msg>`. Ex: `(bob @#welcome) Hello Everyone`.
  - Estas mensagens deverão ir para uma *queue* enquanto não forem obtidas pelos respectivos utilizadores.

- **Receber mensagens na *queue*:** `/msgs`
  - Retorna todas as mensagens separadas por \n, ex `/msgs <msg1>\n<msg2>`.
  - Retorna `/msgs none` se não existirem mais mensagens na lista.

- **Mensagens privadas:** `/pmsg <username> <msg>`
  - Retorna `/pmsg sent` se enviada correctamente.
  - Retorna `/pmsg no_user` se o destinatário não existir.
  - O destinatário deverá receber uma mensagem com o formato `(username @private) <msg>`. Ex: `(bob @#private) Hello there`.

- **Receber mensagens privadas na *queue*:** `/pmsgs`
  - Retorna todas as mensagens privadas separadas por \n, ex `/pmsgs <msg1>\n<msg2>`.
  - Retorna `/pmsgs none` se não existirem mais mensagens privadas na lista.

A lista de comandos pode ser estendida com outros comandos ou respostas no caso da implementação de extras que os requeiram.

Implementação

O código fornecido contém uma implementação inicial que deverá ser expandida pelos alunos.

Servidor: Após receber uma ligação de um cliente (método *Server.start()*), é executado o método ***handle_client(conn)***. Este método deverá ser alterado de forma a que o servidor lide correctamente com os pedidos e respostas.

Cliente: O cliente liga-se ao *socket* do servidor, e depois fica num *loop* a aguardar comandos do utilizador. O cliente deverá ser alterado de forma a que obtenha mensagens públicas e privadas com alguma regularidade, de forma a simular um *chat* interactivo. Considere a classe *threading.Timer* (https://docs.python.org/3/library/threading.html#timer-objects).

Unittests

O ficheiro *tests.py* contém um conjunto de *unittests* que servem para testar o funcionamento correcto do servidor em relação ao protocolo definido pelos comandos e suas respostas.

Á semelhança do trabalho anterior, alguns destes testes são relativamente simples de se entender: por exemplo o teste *TestChatProtocolStart.testUniqueUsername* verifica se o servidor retorna a resposta */username taken* quando o username já estiver em uso.

Os alunos não deverão modificar nada na classe *Server* que impeça a execução dos testes. Recomenda-se executar os testes regularmente de forma a verificar se estão a funcionar e se o servidor cumpre os requisitos.

Entrega e avaliação

Este trabalho deverá ser realizado em **grupos de 2 alunos** usando o *Github* como repositório de código e tem como data limite o **dia 17/Maio/2020 às 23h55**. Deverá também ser colocado no repositório um ficheiro de texto (ou modificar o *readme*) com a identificação dos alunos e extras implementados.

Como a utilização do *Github Classrooms* é ainda experimental, todos os ficheiros deverão também ser colocados num **ficheiro zip** (com o número dos elementos do grupo) e submetido via *moodle*.

Irá considerar-se a seguinte grelha de avaliação:

Correcção da solução (testes)	14 val.
Qualidade e modularização do código (pylint, etc.)	03 val.
Extras (~1 valor por extra)	03 val.

Alguns extras a considerar: (1) utilização de interfaces gráficas no cliente (ex: tkinter, pyQt), (2) utilizadores em múltiplas salas ao mesmo tempo, (3) funcionalidades de moderador, como remover salas e banir utilizadores, etc., (4) manter a lista de salas e dados de login de utilizadores actualizada em ficheiros (ex: usando o módulo *pickle* para a serialização) ou numa base de dados (ex: sqlite3), (5) envio e renderização de “memes” (imagens, vídeos) se se usar interfaces gráficas, (6) mecanismos de *broadcast* tipo aviso em todas as salas por parte de um super-moderador, (7) etc.., sejam criativos!

Nota: para não quebrarem os testes, alguns dos vossos extras poderão necessitar que implementem os vossos próprios comandos. Mantenham uma coerência com os comandos actuais, sff.

Bom trabalho!