UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
DEPARTAMENTO DE INFORMÁTICA APLICADA

INF01151 – SISTEMAS OPERACIONAIS II N
SEMESTRE 2022/1
ATIVIDADE DE PROGRAMAÇÃO GUIADA: WEB SOCKETS

OBJETIVO DA AULA PRÁTICA:

Esta atividade de Programação Guiada tem por objetivo demonstrar o funcionamento de WebSockets. Para isso, uma aplicação simples que realiza a troca de mensagens através do protocolo deverá ser desenvolvida. Nesta aula será utilizada a API Websockets do Python 3.

Tutoriais de apoio:
- Websockets 10.2 documentation

INSTALAÇÃO E CONFIGURAÇÃO DE DEPENDÊNCIAS:

Para essa atividade de programação guiada, vamos precisar dos softwares abaixo.

- Python 3. Para instruções de instalação, acesse https://docs.python.org/3/using/index.html. No Ubuntu Linux, você pode instalar o Python 3 com o seguinte comando:

```
apt-get install python3 python3-pip
```

Com o comando acima, o Python 3 e as principais dependências serão instaladas e configuradas automaticamente.

- Biblioteca gRPC e ferramental relacionado

```
python3 -m pip install websockets
```

UMA APLICAÇÃO DE CHAT COM WEB SOCKETS:

Você deverá implementar *endpoints* para comunicação através do protocolo WebSocket. O objetivo é observar o envio de mensagens de um cliente para o servidor implementado em Python. O servidor fornecerá os métodos necessários para a comunicação bem como um caminho para o recebimento das mensagens (`/chat`). Os passos 1 à 5 compreendem a implementação da aplicação que desempenhará o papel de Servidor.

Observe que, neste caso, a aplicação cliente pode ser um programa de linha de comando ou o próprio navegador. Para o caso de um navegador, a aplicação cliente está definida no arquivo `chat.html`. Para o caso do programa de linha de comando, o cliente pode ser iniciado com o comando abaixo:

```
python3 -m websockets ws://localhost:8080/chat
```

Passos:

1. **Faça o download da implementação-base:** O arquivo **PG1-WS.zip** está disponível no Moodle da disciplina, descompacte-o. Verifique no diretório descompactado quais arquivos estão presentes e examine o conteúdo deles.

2. **Analise o arquivo `websocket.py`:** Um passo importante para a comunicação através de WebSockets é a definição de uma URL (ou *template URI*) que o *endpoint* irá disponibilizar, além de outras definições, como por exemplo *encoders* usados para enviar mensagens.

   Na implementação disponível, o método `web_socket_router()` suporta a URL `/echo`, que invoca o método `echo()` passando como parâmetro o objeto `websocket` origem da requisição (que foi recebido na invocação do método `web_socket_router()`).

   Observe a implementação do método `echo()`. O mesmo simplesmente devolve para o cliente (usando o método `send()`) a mensagem recebida, a qual se encontra no objeto `websocket`.

   Para disparar a aplicação, digite o comando

   ```
   python3 websocket.py
   ```

   Você pode acessar a funcionalidade do método `echo()` via linha de comando com

   ```
   python3 -m websockets ws://localhost:8080/echo
   ```

   Ou no seu navegador, acessando o endereço

   ```
   http://localhost:8080/ui-chat
   ```

   Experimente digitar uma mensagem no programa de linha de comando ou na inputbox da página web exibida no seu navegador. A mensagem deverá ser ecoada na própria interface do programa de linha de comando / página web em que ela foi digitada.

   Observe que o método `http_handler` permite que o servidor possa atuar não apenas com websockets, mas também para servir páginas web. Essa funcionalidade será necessária para permitir que navegadores possam acessar o cliente de chat da nossa aplicação. O estudo desse método é deixado como sugestão de estudos adicionais.

3. Estenda a implementação contida no arquivo `websocket.py` para incluir um novo handler para o serviço de chat (novo método `chat()`) que deve ser implementado no escopo deste trabalho. Você pode usar o código do método `echo()` como base.

   Você sabe para que serve a palavra-chave `async` que precede a declaração do método?

4. **Persistência de informações das Sessões:** A comunicação entre clientes e servidor é realizada através de sessões. Assim, quando uma mensagem for recebida de um cliente, o servidor saberá quais são os clientes ativos para os quais a mensagem deverá ser enviada.

   Para suportar sessões, estenda a assinatura do método chat() para receber um arranjo de sessões (parâmetro `sessions`). A assinatura deverá suportar o valor padrão arranjo vazio para o parâmetro `sessions` (`sessions={}`).

   Para guardar o websocket de um cliente no arranjo de sessões, você pode usar a tupla Endereço IP / Porta TCP do cliente remoto como chave, recuperando-o do atributo `websocket.remote_address` do objeto `websocket` recebido como parâmetro no método `chat()`. Use o código de exemplo abaixo:

   ```
   remote = websocket.remote_address
   sessions[remote] = websocket
   ```

5. Após a linha de iteração para recuperar as mensagens do websocket

   ```
   async for message in websocket:
   ```

   inclua uma nova linha para enviar cada mensagem recuperada para cada um dos clientes ativos. Para isso, você deverá recuperar os sockets dos clientes ativos invocando o método `values()` do parâmetro `sessions`.

   ```
   for socket in sessions.values():
   ```

   A variável socket da iteração deverá ser usada para enviar a mensagem, da mesma forma como é feito no método `echo()`.

6. Para lidar com clientes que desconectam e removê-los do arranjo de sessões ativas, você poderá usar o bloco `try / finally` . O bloco de instruções parte do `try` deverá incluir a partir da instrução de iteração pelas mensagens do websocket. O bloco `finally`, por sua vez, deverá compreender uma instrução para remover a sessão. A chave dessa sessão que deverá ser removida é a própria tupla endereço IP / Porta TCP do cliente remoto) do arranjo de sessões

```
del sessions[remote]
```

7. Para finalizar a implementação do servidor de chat, modifique a implementação do método `web_socket_router` para definir uma nova URL `/chat`. Você poderá copiar a mesma especificação da URL `/echo`.

A implementação dos métodos acima é a base para o funcionamento de uma simples aplicação de chat em texto sobre WebSockets em Python. A parte do cliente está definida em juntamente com a definição da interface de *chat* no arquivo `chat.html`, com a implementação dos métodos de WebSockets em Javascript. A implementação do serviço de *echo* é similar e pode ser encontrada no arquivo `echo.html`.

8. **Observe as definições dos métodos-cliente:** No segmento `<script> ... </script>` do arquivo `chat.html` temos a criação de um canal websocket em `ws://localhost:8080/chat` e as implementações dos métodos de abertura, envio e fechamento. Basicamente, cada vez que uma guia do navegador da aplicação for aberta, uma conexão WebSocket é estabelecida. A aplicação permite que o cliente crie e envie mensagem para outros clientes conectados.
9. **Disparando a aplicação:** A partir do diretório pai da aplicação, execute o seguinte comando:

```
python3 websocket.py
```

Para acessar o serviço a partir da linha de comando, digite:

```
python3 -m websockets ws://localhost:8080/chat
```

Para acessar o serviço a partir do navegador, acesse o endereço:

```
http://localhost:8080/ui-chat
```

10. **Testando a aplicação:** Após a instanciação da aplicação servidor, inicie algumas instâncias de clientes (linha de comando ou navegador) e realize o envio de mensagens. A aplicação deve realizar a replicação de mensagens para todos os clientes conectados.
11. Inclua mensagens de log para indicar quando um cliente se conecta, desconecta e quando ele envia uma mensagem. Você deverá exibir a tupla Endereço IP / Porta TCP do cliente, que está disponível no atributo `websocket.remote_address`
12. **Problemas?** Revise os passos anteriores e corrija!

Yay! Se você chegou até aqui, parabéns! Faça o envio do relatório do programa no Moodle, e seu objetivo na atividade de programação guiada estará cumprido. O relatório deverá incluir o código fonte desenvolvido e screenshots do funcionamento do serviço e dos clientes instanciados para interagir com o serviço.