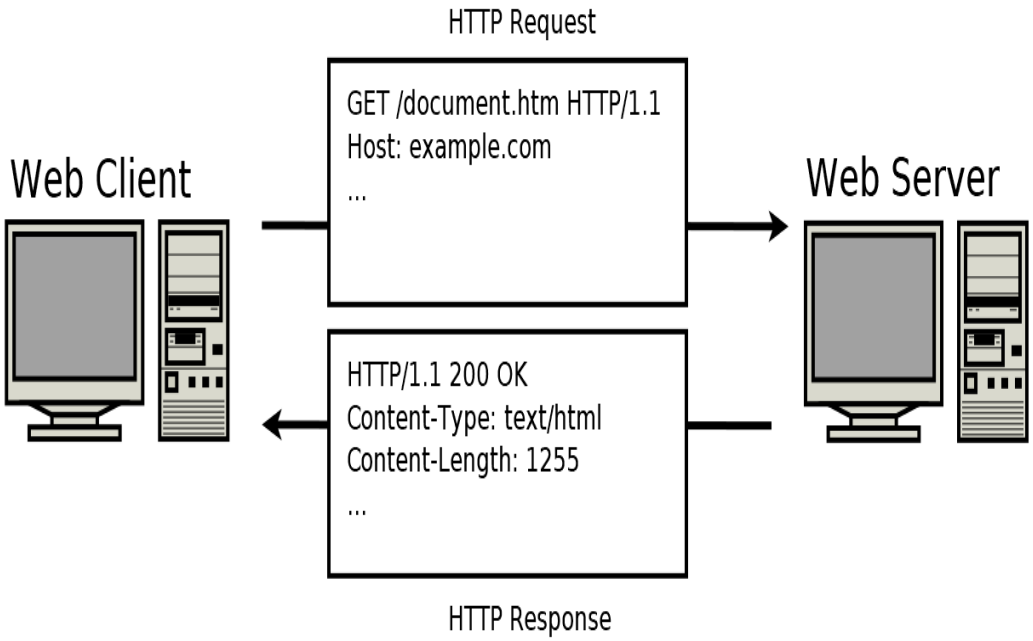
Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e Multimédia

Redes de Computadores - 2022/2023 SV

1ª Fase - Servidor Web



Docente Nelson Costa

Realizado por grupo 5:
Diogo Lobo 48168
Pedro Silva 48965
Diogo Santos 48626

16 de abril de 2023

Conteúdo

Lista de Figuras

1 Introdução

Este projeto tem como objetivo o desenvolvimento de uma Rede de Computadores. Começamos por criar um servidor web, depois uma LAN para dois PCs usando um switch e finalmente evoluímos de forma a conectar a nossa LAN a outros e criar uma típica rede empresarial. Nesta primeira fase, vamos desenvolver um cliente Web utilizando a linguagem Java onde estabelecemos uma ligação TCP com o servidor e trocamos mensagens HTTP com o mesmo.

2 Configuração do Servidor Web

Para podermos testar o nosso cliente, precisamos de criar um servidor local utilizando o programa XAMPP. Este programa simula um servidor Web no computador onde se encontra instalado, recebendo pedidos e enviando respostas HTTP.

Inicializamos o modulo Apache no painel de controlo do XAMPP.

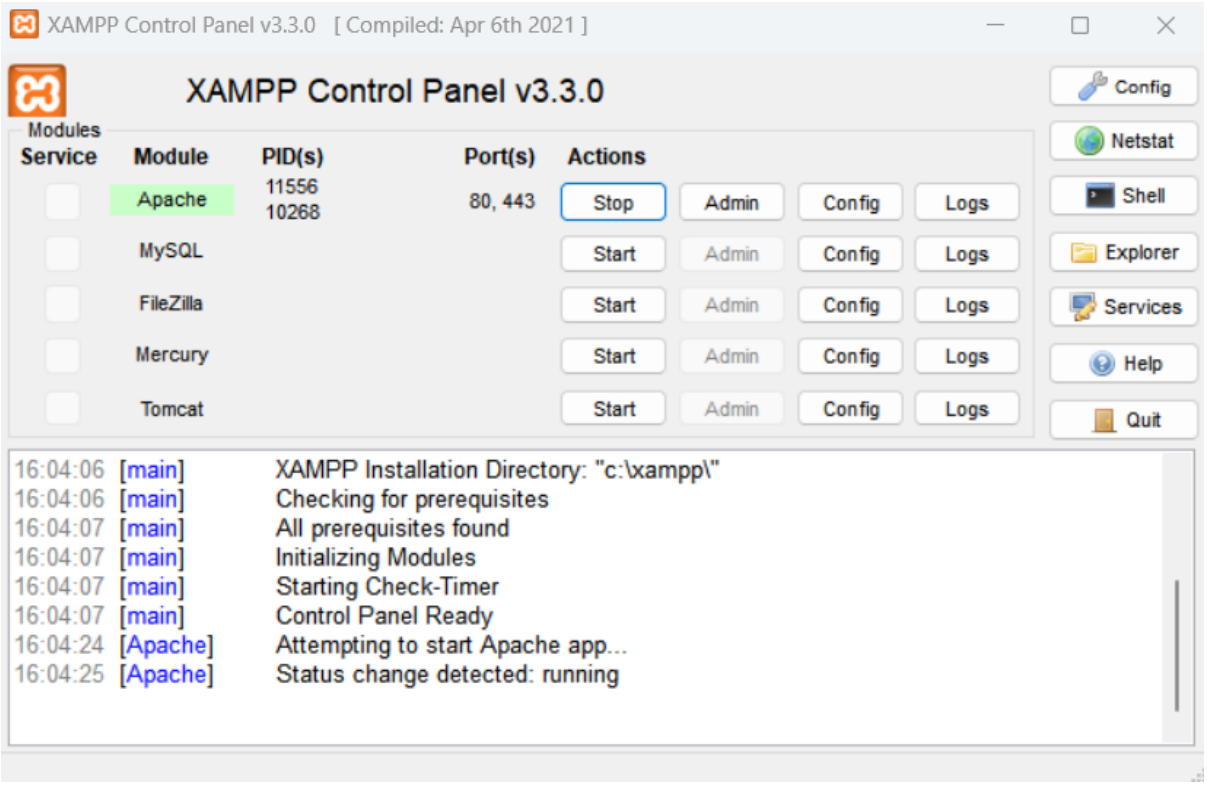


Figura 1: Painel de controlo XAMPP

Depois de inicializado podemos aceder ao servidor através do nosso browser pelo nosso endereço IP ou pelo link http://127.0.0.1/. Isto também resulta para qualquer dispositivo na mesma rede basta introduzir o endereço IP da máquina onde o servidor está a correr no motor de busca.

Precisamos então de descobrir qual o endereço associado ao servidor LAN, logo vamos abrir a linha de comandos e executar o comando *ipconfig*.

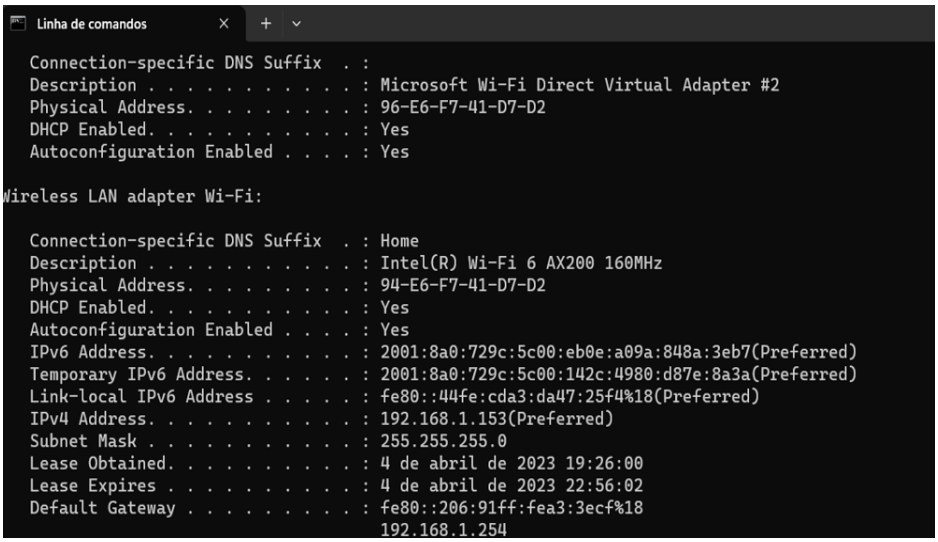


Figura 2: Linha de comandos após *ipconfig*

Como o computador está ligado por Wi-Fi, encontramos a informação desejada na secção **Wireless LAN adapter WiFi** no campo **IPv4 Address**, ou seja, *192.168.1.153*.

Correndo o Web Server na nossa máquina, ficamos com o seguinte resultado quando pesquisamos pelo link http://127.0.0.1/ no motor de busca (Imagem da esquerda) e ao utilizarmos outro dispositivo como por exemplo o telemóvel obtemos o mesmo resultado ao pesquisar um endereço IP da máquina principal (Imagem da direita):

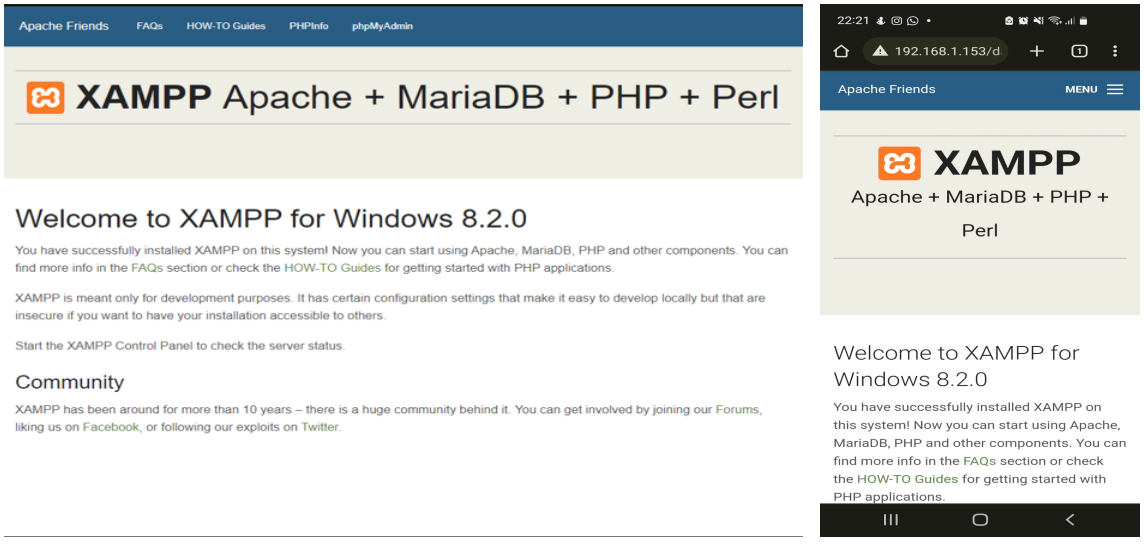
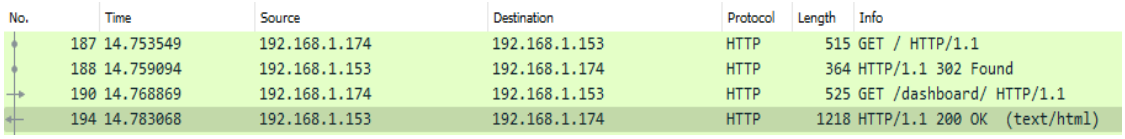


Figura 3: Testes ao Servidor Web

Após verificar que funciona igualmente num computador e noutro dispositivo verificou-se a interseção do pedido na plataforma Wireshark podendo confirmar que o IP source corresponde ao do dispositivo, como se pode ver na seguinte imagem



No.	Time	Source	Destination	Protocol	Length	Info
187	14.753549	192.168.1.174	192.168.1.153	HTTP	515	GET / HTTP/1.1
188	14.759094	192.168.1.153	192.168.1.174	HTTP	364	HTTP/1.1 302 Found
190	14.768869	192.168.1.174	192.168.1.153	HTTP	525	GET /dashboard/ HTTP/1.1
194	14.783068	192.168.1.153	192.168.1.174	HTTP	1218	HTTP/1.1 200 OK (text/html)

Figura 4: Verificação do pedido na aplicação Wireshark

Ao analisar esta imagem pode-se verificar o pedido pelo dispositivo e as respostas. Primeiro a de 302 FOUND onde a resposta foi temporariamente movida para o URL dado pelo header da localização e de seguida a 200 OK que indica a conclusão do pedido e que foi bem sucedido

3 Desenvolvimento do Web Client

Nesta parte do desenvolvimento do trabalho foi necessário criar um web client que comunicasse com o servidor. Para a criação deste cliente foi utilizada a linguagem Java, devido a ser esta a linguagem onde aprendemos a realizar este tipo de processos noutras cadeiras como IERCD, e desenvolvidos vários métodos com o propósito de interagir com o servidor.

3.1 Método *socketInit()*

O primeiro método desenvolvido foi o *socketInit()* que vai criar um socket, o seu input e output e envia o pedido HTTP com o método GET

```java
/**
 * Criámos a inputSocket utilizando um BufferedReader para fazer print pelo servidor
 * e a outputSocket para enviar para o servidor,
 * acabamos por enviar o pedido HTTP com este output utilizando o método GET.
 * @param host String com o endereço IP
 * @param Port Int com o num da porta
 * @param URI "/"
 * @return
 */
2 usages
public static BufferedReader socketInit(String host,int Port, String URI) throws IOException {

    Socket socket = new Socket(host,Port);
    InputStreamReader input = new InputStreamReader(socket.getInputStream());
    BufferedReader inputSocket = new BufferedReader(input);
    PrintStream outputSocket = new PrintStream(socket.getOutputStream());
    outputSocket.println("GET " + URI + " HTTP/1.0\r\n");
    // socket.close(); Não se pode fechar o socket para funcionar
    return inputSocket;
}
```

Figura 5: Método *socketInit()*

Neste método é estabelecido uma conexão TCP entre o cliente e o servidor, com recurso às bibliotecas **java.io** e **java.net**. De seguida é criado um input utilizando um bufferedReader para fazer print pelo servidor e um output para enviar o output desejado para o servidor. No final envia-se o pedido HTTP com o output utilizando o método GET.

3.2 Método *httpRead()*

O próximo método a ser desenvolvido foi o *httpRead()* que vai ler a mensagem de resposta enviada pelo servidor e informa-nos na consola qual o status code.

```java
/**
 * lê a mensagem de resposta enviada pelo servidor e faz print na consola com base
 * na resposta dada. No fim redirecionamos para a função output() com a primeira linha da
 * resposta, a socket e a informação sobre a resposta do servidor.
 * @param code primeira linha da resposta
 * @param socket BufferedReader com os pacotes de entrada
 */
2 usages
public static void httpRead(String code,BufferedReader socket) throws IOException {
    System.out.println();
    //O código esta após o primeiro espaço exemplo: HTTP/1.1 302 Found
    String info = code.split( regex: " ")[1];

    switch (info) {
        case "200" -> {
            System.out.println("200 OK");
        }
        case "301" -> {
            System.out.println("301 MOVED");
        }
        case "302" -> {
            System.out.println("302 FOUND");
        }
        case "400" -> {
            System.out.println("400 BAD REQUEST");
        }
        case "401" -> {
            System.out.println("401 UNAUTHORIZED");
        }
        case "403" -> {
            System.out.println("403 FORBIDDEN");
        }
        case "404" -> {
            System.out.println("404 ERROR");
        }
        default -> System.out.println("UNRECOGNIZED CODE");
    }
    output(code,socket,info);
}
```

Figura 6: Método *httpRead()*

Para descobrirmos qual o status code basta ler o número após o primeiro espaço na primeira linha da resposta como por exemplo: ”HTTP/1.1 302 Found”. No fim redirecionamos para a função output que vai-nos exibir a informação que desejamos.

3.3 Método *output()*

Nesta função vamos analisar informação enviada pelo método anterior, ou seja, qual o status code.

```java
/**
 *se a informação enviada for "301" ou "302", redirecionamos para a
 * função getURL() para podermos criar uma nova socket com a nova informação e chamamos outra vez httpRead.
 * Se não fazemos print da mensagem de resposta
 * inteira, o que vai também mostrar o código HTML da página
 * @param code primeira linha da resposta
 * @param socket BufferedReader com os pacotes de entrada
 * @param info contém a informação do pedido http
 */
1 usage
public static void output(String code,BufferedReader socket,String info) throws IOException {
    String msg = code;
    System.out.println();
    //Se a resposta for 301 ou 302
    if(info.equals("301") || info.equals("302")){
        System.out.println("\nRedirecting...\n");
        //Encontramos o Location Header
        String URL = getURL(code,socket);
        //Criamos a nova Socket
        BufferedReader inSock=socketInit(IP,PORT,URL);
        String line = inSock.readLine();
        //Ler a resposta
        httpRead(line,inSock);
    //Se não imprimimos a mensagem da resposta inteira
    }else {
        System.out.println();
        System.out.println("Reply message: ");
        System.out.println();
        while (msg!=null){
            msg=socket.readLine();
            System.out.println(msg);
        }
    }
}
```

Figura 7: Método *output()*

Se este for "301"ou "302"precisamos de redirecionar o programa para o método *getURL()* que nos vai criar uma nova socket com a nova informação (aprofundaremos a seguir). A seguir chamamos outra vez o método *httpRead()* mas desta vez com a informação desta nova socket. Por fim se o código não for nenhum dos indicados em cima exibimos a mensagem de resposta na sua totalidade, também contendo o HTML da página.

3.4 Método *getURL()*

A função *getURL()* é utilizada para encontrar o Location Header, que indica o URL utilizado para redirecionar a pagina.

```java
/**
 * Encontra a Location Header, ou seja, o endereço de reencaminhamento.
 * Escreve a mensagem de resposta na consola e retorna o novo URL que
 * vai ser usado para criar a nova socket
 * @param code primeira linha da resposta
 * @param socket BufferedReader com os pacotes de entrada
 * @return String com o novo URL
 */
1 usage
private static String getURL(String code, BufferedReader socket) throws IOException {

    String msg = code;
    // System.out.println("Code-->"+code);
    String URL = "";

    //Percorremos os pacotes de entrada
    while (msg!=null){
        System.out.println(msg);
        //Se encontrarmos o campo "Location:" guardamos esse endereço
        if(msg.contains("Location:")){
            URL = msg.substring( beginIndex: 10);
            // System.out.println("URL--> "+URL);
            break;
        }
        msg = socket.readLine();
    }
    return URL;
}
```

Figura 8: Método *getURL()*

Este método apenas é utilizado quando o código resposta do servidor é ”301”ou ”302”, escreve-se no final a mensagem da resposta na consola antes de retornar o URL que vai ser utilizado na criação do socket na função *output()*.

3.5 Método *Main()*

O método *Main()* é o utilizado para o envio do pedido HTTP. É utilizado duas constantes predefinidas, IP=”127.0.0.1”e Port=80. Inicialmente é utilizado o método *socketInit()* para a sua inicialização e de seguida utiliza-se a função *httpRead()* que recebe como parâmetros o socket inicializado e uma string sendo essa string o *readLine()* do socket.

```
2 usages
public static String IP = "127.0.0.1";
2 usages
public static int PORT = 80;


no usages
public static void main(String[] args) throws IOException {
    BufferedReader inSock = socketInit(IP,PORT, URI: "/");
    String line = inSock.readLine();
    // System.out.println("line-->"+line);
    httpRead(line,inSock);
}
```

Figura 9: Método *main()*

3.6 Resultados finais

Ao correr o código desenvolvido podemos verificar que os códigos recebidos 302 FOUND que redireciona o servidor para o endereço na Location Header e mostra a mensagem de resposta e repete o processo onde o código recebido se torna no 200 OK. Com este código final é também retornado o código HTML do website do servidor.

```
302 FOUND


Redirecting...

HTTP/1.1 302 Found
Date: Sun, 16 Apr 2023 10:49:03 GMT
Server: Apache/2.4.54 (Win64) OpenSSL/1.1.1p PHP/8.2.0
X-Powered-By: PHP/8.2.0
Location: http:///dashboard/
Content-Length: 115
Connection: close
Content-Type: text/html; charset=UTF-8

<br />
<b>Warning</b>:  Undefined array key "HTTP_HOST" in <b>C:\xampp\htdocs\index.php</b> on line <b>7</b><br />

200 OK


Reply message:

Date: Sun, 16 Apr 2023 10:49:03 GMT
Server: Apache/2.4.54 (Win64) OpenSSL/1.1.1p PHP/8.2.0
Last-Modified: Thu, 29 Dec 2022 18:57:59 GMT
ETag: "1442-5f0fc1045fbc0"
Accept-Ranges: bytes
Content-Length: 5186
Connection: close
Content-Type: text/html
```

Figura 10: Códigos de servidor e resposta

4 Interação Cliente-Servidor

Com recurso à aplicação Wireshark foi capturada a interação do cliente com o servidor. Ao observar as imagens seguintes é possível ver os dois pedidos GET enviados e as respetivas respostas a cada um desses pedidos.

No.	Time	Source	Destination	Protocol	Length	Info
25	5.906793	127.0.0.1	127.0.0.1	HTTP	46	GET / HTTP/1.0
27	5.908649	127.0.0.1	127.0.0.1	HTTP	409	HTTP/1.1 302 Found (text/html)
36	5.914114	127.0.0.1	127.0.0.1	HTTP	46	GET http:///dashboard/ HTTP/1.0
38	5.917747	127.0.0.1	127.0.0.1	HTTP	5504	HTTP/1.1 200 OK (text/html)

Figura 11: Pedidos e respostas entre o cliente e servidor

```
▾ Hypertext Transfer Protocol
  ▾ GET / HTTP/1.0\r\n
    ▾ [Expert Info (Chat/Sequence): GET / HTTP/1.0\r\n]
        [GET / HTTP/1.0\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Request Method: GET
      Request URI: /
      Request Version: HTTP/1.0
    \r\n
    [HTTP request 1/1]
```

Figura 12: Pedido de conexão GET

```
▾ Hypertext Transfer Protocol
  ▾ HTTP/1.1 302 Found\r\n
    ▾ [Expert Info (Chat/Sequence): HTTP/1.1 302 Found\r\n]
        [HTTP/1.1 302 Found\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Response Version: HTTP/1.1
      Status Code: 302
      [Status Code Description: Found]
      Response Phrase: Found
    Date: Thu, 13 Apr 2023 13:13:03 GMT\r\n
    Server: Apache/2.4.54 (Win64) OpenSSL/1.1.1p PHP/8.2.0\r\n
    X-Powered-By: PHP/8.2.0\r\n
    Location: http:///dashboard/\r\n
  ▸ Content-Length: 115\r\n
    Connection: close\r\n
    Content-Type: text/html; charset=UTF-8\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.001856000 seconds]
    [Request in frame: 25]
    [Request URI: /]
    File Data: 115 bytes
```

Figura 13: Resposta ao pedido de conexão GET

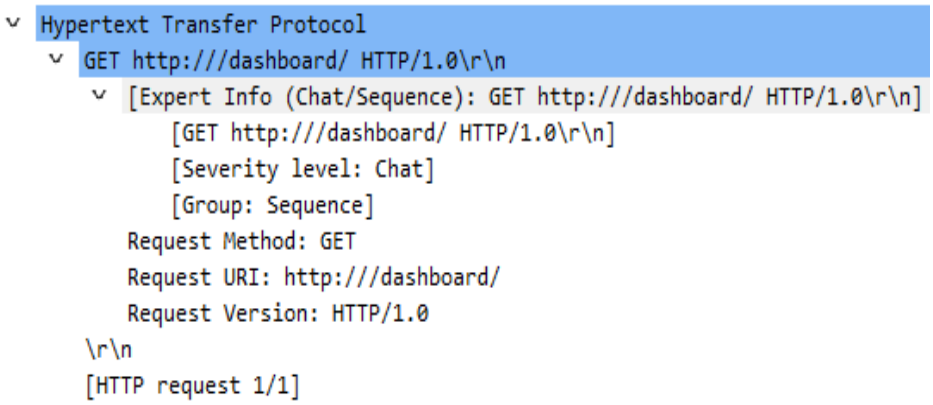


Figura 14: Pedido de GET de URI

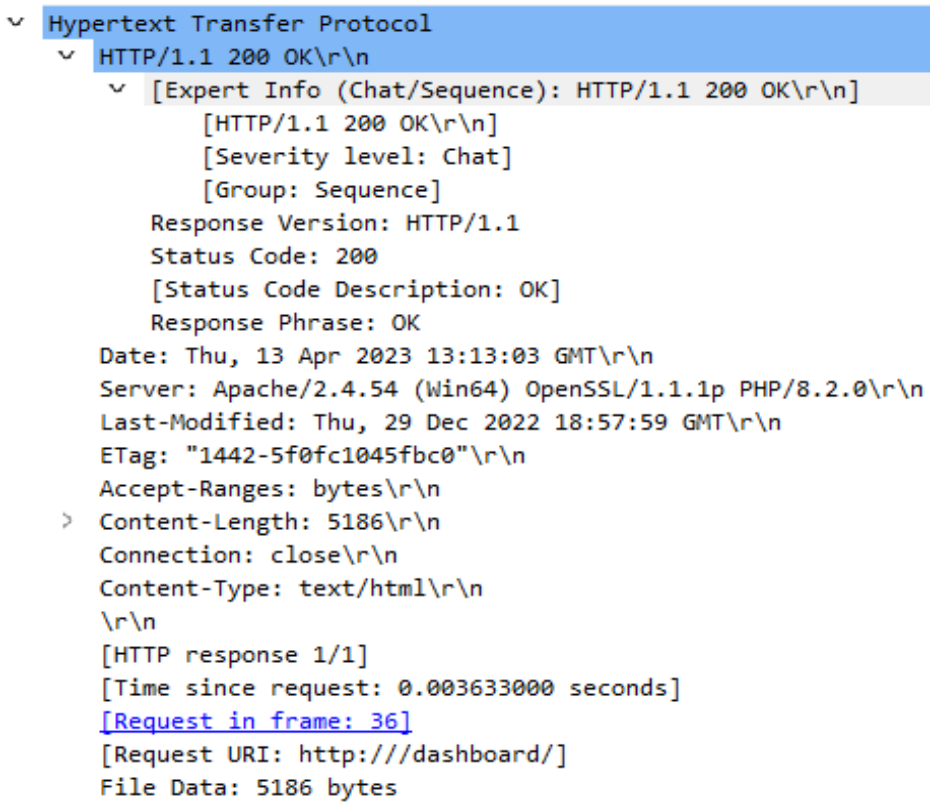


Figura 15: Resposta ao pedido de URI

5 Conclusões

O objetivo desta fase do projeto era o de consolidar e aprofundar os conhecimentos adquiridos ao longo de Redes de Computadores mais especificamente a estrutura cliente-servidor e o protocolo HTTP. Ao conceber-mos o cliente Web conseguimos estabelecer uma ligação TCP, enviar um pedido HTTP e receber uma resposta HTTP apresentando-a ao utilizador. Também tivemos em consideração as diferentes resposta HTTP possíveis e desenvolvemos o cliente com isso em mente. Conseguimos atingir todos os requisitos desta fase do projeto demonstrando o domínio que temos sobre a matéria lecionada.