



PMR3412 - Redes Industriais - 2021

Aula 07 - Aplicações TCP/IP: HTTP (cont.) e WebSockets

Prof. Dr. Newton Maruyama

28 de Setembro de 2023

PMR-EPUSP

Os slides que serão utilizados nesse ano são baseados no curso desenvolvido para os anos 2020, 2021 e 2022. Participaram da concepção do curso e desenvolvimento do material os seguintes professores:

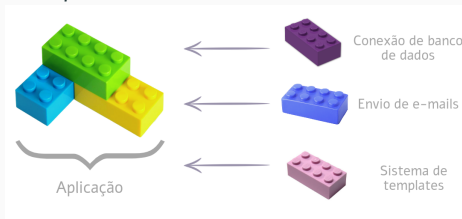
- ▶ Prof. Dr. André Kubagawa Sato
- ▶ Prof. Dr. Marcos de Sales Guerra Tsuzuki
- ▶ Prof. Dr. Edson Kenji Ueda
- ▶ Prof. Dr. Agesinaldo Matos Silva Junior
- ▶ Prof. Dr. André César Martins Cavalheiro

1. Hypertext Transfer Protocol (HTTP) (cont.)
2. A Web
3. HTTP/2
4. WebSockets
5. Referências

Hypertext Transfer Protocol (HTTP) **(cont.)**



- ▶ Flask é um micro-framework multiplataforma para desenvolvimento web.
- ▶ Foi lançado em 2010 por Armin Ronacher.
- ▶ Um micro-framework possui característica modular:



- ▶ Características:
 - ▶ Simplicidade,
 - ▶ Rapidez no desenvolvimento,
 - ▶ Projetos menores e mais leves,
 - ▶ Arquitetura robusta.
- ▶ Exemplos de frameworks não minimalistas que utilizam a linguagem Python:
Django

- ▶ Flask é construído através da utilização de duas bibliotecas:
 1. Jinja: mecanismo de template para a linguagem de programação Python (<https://jinja.palletsprojects.com/en/3.1.x/>).



2. Werkzeug WSGI (Web Server Gateway Interface): especificação para uma interface simples e universal entre servidores web e aplicações web ou frameworks para a linguagem de programação Python (<https://werkzeug.palletsprojects.com/en/2.2.x/>).



- ▶ Sites que utilizam Flask: Airbnb, Netflix, Samsung, Uber, Trivago, ...

► Versão longa:

```
def decorator(funcao):  
    def wrapper():  
        print ("Estou antes da execucao da funcao passada como argumento")  
        funcao()  
        print ("Estou depois da execucao da funcao passada como argumento")  
    return wrapper  
def outra_funcao():  
    print ("Sou um belo argumento!")  
  
funcao_decorada = decorator(outra_funcao)  
funcao_decorada()
```

► Versão sintética:

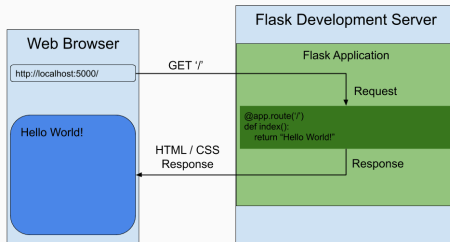
```
def decorator(funcao):  
    def wrapper():  
        print ("Estou antes da execucao da funcao passada como argumento")  
        funcao()  
        print ("Estou depois da execucao da funcao passada como argumento")  
    return wrapper  
@decorator  
def outra_funcao():  
    print ("Sou um belo argumento!")  
  
outra_funcao()
```

- ▶ O código abaixo está contido no arquivo hello.py

```
from flask import Flask

app = Flask(__name__)
@app.route("/")
def hello_world():
    return 'Hello World!'
```

- ▶ O diagrama a seguir ilustra como o flask processa a requisição do browser para URL "/":



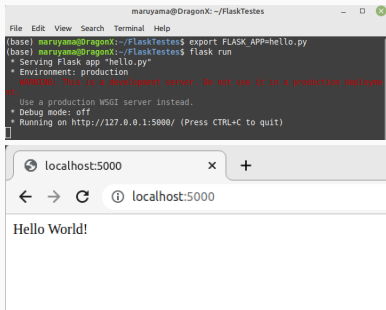
- ▶ Para executar no Windows PowerShell:

```
env:FLASK\_APP = "hello.py"  
python -m flask run
```

- ▶ Para executar no Linux:

```
export FLASK_APP=hello.py  
flask run
```

- ▶ Teste:



- ▶ Obviamente poderíamos fazer a função `hello_world` devolver o texto formatado em HTML como indicado abaixo (arquivo `hello1.py`):

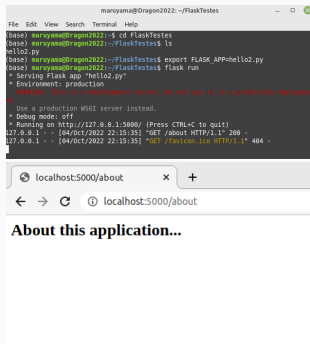
```
from flask import Flask

app = Flask(__name__)
@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

▶ Exemplo de routing (arquivo hello2.py):

```
from flask import Flask

app = Flask(__name__)
@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
@app.route('/about')
def about():
    return '<h2>About this application...</h2>'
```

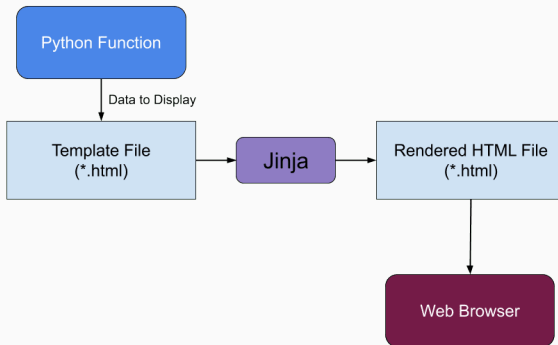


The screenshot shows a terminal window titled 'maruyama@Ragon2022: ~/FlaskTestes'. The terminal output is as follows:

```
File Edit View Search Terminal Help
(base) maruyama@Ragon2022:~$ cd FlaskTestes
(base) maruyama@Ragon2022:~/FlaskTestes$ ls
hello2.py
(base) maruyama@Ragon2022:~/FlaskTestes$ export FLASK_APP=hello2.py
(base) maruyama@Ragon2022:~/FlaskTestes$ flask run
 * Serving Flask app "hello2.py"
 * Environment: production
   WARNING: This is a development server. Do not use it in a production system.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [04/Oct/2022 22:15:35] "GET /about HTTP/1.1" 200 -
127.0.0.1 - - [04/Oct/2022 22:15:35] "GET /favicon.ico HTTP/1.1" 404 -
```

Below the terminal, a web browser window is open to 'localhost:5000/about'. The browser address bar shows 'localhost:5000/about' and the page content displays 'About this application...' in a large, bold, black font.

- ▶ Em Flask os templates são processados através da biblioteca Jinja:



▶ Código Python (arquivo appform.py):

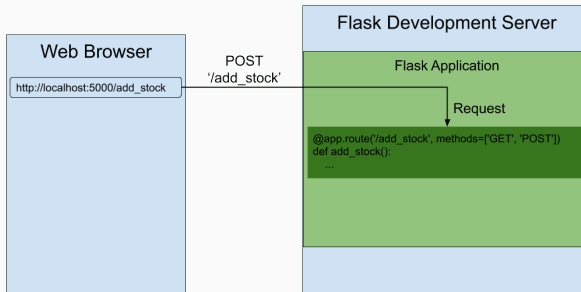
```
from flask import Flask, escape, render_template, request

app = Flask(__name__)
@app.route('/add_stock', methods=['GET', 'POST'])
def add_stock():
    if request.method == 'POST':
        # Print the form data to the console
        for key, value in request.form.items():
            print(f'{key}: {value}')
        return render_template('add_stock.html')
```

▶ Template html (arquivo add_stock.html)

```
<!doctype html>
<h2>Add a Stock:</h2>
<form method="post">
  <label for="stockSymbol">Stock Symbol <em>(required)</em></label>
  <input type="text" id="stockSymbol" name="stock_symbol" required pattern="[A-Z]{1,5}" /> <!-- Updated! -->
  <br>
  <label for="numberOfShares">Number of Shares <em>(required)</em></label>
  <input type="text" id="numberOfShares" name="number_of_shares" required /> <!-- Updated! -->
  <br>
  <label for="purchasePrice">Purchase Price ($) <em>(required)</em></label>
  <input type="text" id="purchasePrice" name="purchase_price" placeholder="$300.00" required /> <!-- Updated! -->
  <br>
  <input type="submit">
</form>
```

- ▶ Diagrama esquemático ilustrando a submissão do forms através do método POST:



- ▶ Exemplo de submissão de forms:

The image shows a terminal window and a web browser. The terminal window, titled "maruyama@DragonX: ~/FlaskTestes/appforms", displays the following output:

```
(base) maruyama@DragonX:~/FlaskTestes/appforms$ export FLASK_APP=appforms.py
(base) maruyama@DragonX:~/FlaskTestes/appforms$ flask run
* Serving Flask app "appforms.py"
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [05/Oct/2022 16:42:59] "GET /add_stock HTTP/1.1" 200 -
stock_symbol: ADS
number_of_shares: 1000
purchase_price: 30.00
127.0.0.1 - - [05/Oct/2022 16:45:15] "POST /add_stock HTTP/1.1" 200 -
```

The web browser window shows the URL "localhost:5000/add_stock". The page content is:

Add a Stock:

Stock Symbol (required)

Number of Shares (required)

Purchase Price (\$) (required)

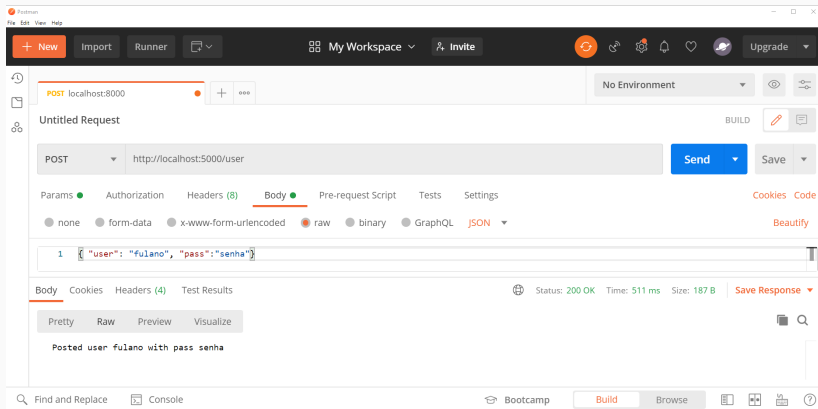
- ▶ Envio de username e password através de objetos json:

```
from flask import Flask, request
app = Flask(__name__)

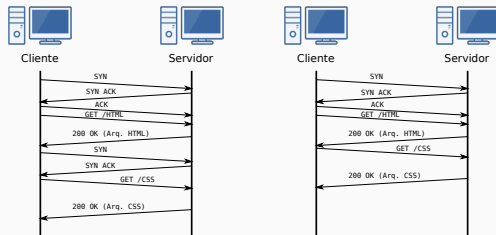
@app.route('/user', methods=['GET', 'POST'])
def user():
    if request.method == 'POST':
        return 'Posted user ' + request.json["user"] + ' with pass ' + request.json["pass"]
    else:
        return 'Got user with name ' + request.args.get('name')
```


HTTP Flask - Exemplo 5 - Teste com Postman

- ▶ Testando requisições POST com o Postman:
`https://www.postman.com/`.
- ▶ Deve sempre ser utilizado a versão Desktop.



- ▶ O HTTP/1.1 introduziu a possibilidade de conexões persistentes para melhorar o desempenho do protocolo.
- ▶ Na versão anterior, era necessária uma requisição para cada recurso. Isto é, se uma página continha uma figura e um arquivo CSS, eram necessárias três requisições (HTML + CSS + imagem).
- ▶ Devemos lembrar que o protocolo TCP realiza um handshake de três vias para cada conexão.
- ▶ Outra possibilidade introduzida no HTTP/1.1 é o *pipelining*, que permite múltiplas requisições sobre uma única conexão TCP. Porém não é muito empregado e foi sucedido pelos novos mecanismos da versão HTTP/2.



- ▶ Como o protocolo HTTP é *stateless*, a princípio não é possível reter informações entre uma requisição e outra.
- ▶ Por este motivo foram propostos os cookies, que permitem anexar alguns poucos dados na resposta do servidor para o browser do usuário.
- ▶ É tipicamente utilizado para determinar se requisições têm origem no mesmo *browser*, como no caso de logins, carrinhos de compra, preferências do usuário, temas, etc.
- ▶ O cabeçalho de resposta `Set-Cookie` pode ser utilizado pelo servidor para enviar um cookie para o cliente.

```
Set-Cookie: <cookie-name>=<cookie-value>
```

- ▶ Em requisições subsequentes do cliente, ele deve incluir o cookie no cabeçalho da requisição:

```
GET /sample_page.html HTTP/2.0  
Host: www.example.org  
Cookie: <cookie-name>=<cookie-value>
```

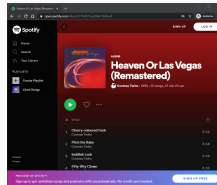
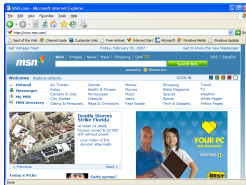
- ▶ O protocolo HTTP foi desenvolvido considerando o desempenho, buscando diminuir a carga para transferências. Um dos mecanismos mais importantes nesta área é o cache.
- ▶ Caching consiste em armazenar uma cópia de um recurso e retorná-lo quando for requisitado. Isto evita a necessidade de baixar novamente um recurso do servidor, diminuindo o número de requisições.
- ▶ O cache pode ser operado de duas formas:
 - ▶ Cache privado do browser: o browser faz o cache de todos os documentos obtidos via HTTP. Dedicado apenas a um usuário.
 - ▶ Cache compartilhado de proxy: armazena documentos para serem servidos para múltiplos usuários.
- ▶ É possível estabelecer um prazo de expiração para o cache de um recurso com o cabeçalho:

```
Cache-Control: max-age=31536000
```

A Web

A Web - Hipertexto, Web Pages, and Web Apps

- ▶ Para quem acompanhou a evolução da Web, observou-se pelo menos três fases:
 - ▶ Documento Hipertexto: consiste na maior parte de texto com formatação básica, além do suporte para hiperlinks.
 - ▶ Página Web: adicionada a capacidade de processar recursos multimídia, como imagens e áudio. Também implementou novos primitivos para leiautes mais elaborados. Era, no entanto, limitado no quesito interação, que geralmente ocorria a partir de formulários.
 - ▶ Aplicações Web: página web com bastante interação, transformando em uma aplicação propriamente dita, que pode responder a interação de usuários diretamente no browser. O *Outlook Web Access* e o *Gmail* são exemplos de precursores desta tecnologia.



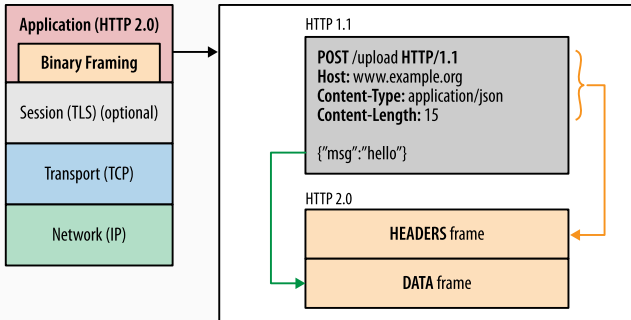
- ▶ No protocolo HTTP, o conteúdo é gerado pelo servidor e transmitido para o cliente, que geralmente utiliza o browser para processá-lo.
- ▶ O conteúdo pode ser estático ou dinâmico, i.e., gerado por um programa (Backend) no momento da requisição.
- ▶ No medida em que a Web evoluía, o modo como o conteúdo é processado e renderizado também passou por alterações. Inicialmente podemos identificar três estratégias:
 - ▶ conteúdo estático: páginas estáticas são conteúdos (HTML, CSS, mídias) que não mudam com frequência e são transferidos para todos os clientes sem modificação. Podem ser utilizados geradores de sites estáticos para simplificar a produção deste tipo de conteúdo.
 - ▶ renderização no servidor: a partir da requisição, o servidor processa os dados e gera o conteúdo (HTML, CSS, mídias) para ser transferido ao cliente.
 - ▶ renderização no cliente: o servidor responde a requisição com dados pertinentes e os elementos de exibição (HTML, CSS) são gerenciados pelo browser (através de scripts na linguagem Javascript). Devido a estratégias como o AJAX (Asynchronous JavaScript and XML), não é necessário recarregar a página a cada ciclo requisição/resposta.

HTTP/2

- ▶ Introduzido em 2015, visa melhorar a performance do protocolo HTTP (RFC 7540).
- ▶ Não modifica a semântica do HTTP/1.1; sendo assim, métodos, campos de cabeçalho, URI e códigos de resposta continuam iguais.
- ▶ Introduz uma nova camada, que possibilita a multiplexação completa das requisições/resposta, assim como a compressão do cabeçalho.
- ▶ Além disso, permite priorizar certas requisições e introduz o *push* do servidor.
- ▶ Exceto por desenvolvedores de servidores Web ou aplicações com sockets puro, não necessita de adaptações em relação ao HTTP/1.1.

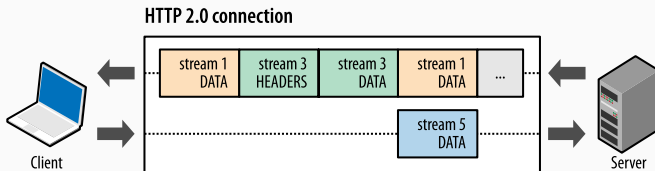
HTTP/2 - Camada de frame binário

- ▶ Codificação do texto puro em frames binários durante o transporte. O frame é a menor unidade de comunicação do HTTP/2, possuindo cabeçalho.
- ▶ Cada conexão pode transportar múltiplos streams bidirecionais, que por sua vez pode conter mais de uma mensagem.
- ▶ Sendo assim, o HTTP/2 converte a comunicação para uma troca de frames de codificação binária que são mapeados em mensagens que pertencem a um determinado stream.



HTTP/2 - Multiplexação de requisição e resposta

- ▶ No HTTP/1.1, requisições paralelas só são possíveis se o cliente inicia múltiplas conexões, o que bastante ineficaz no protocolo TCP.
- ▶ O mecanismo de streams de frames binários permite múltiplas requisições/respostas em uma única conexão através da multiplexação completa.
- ▶ A multiplexação consiste em dividir a mensagem em frames independentes, intercalá-los, enviá-los e depois reagrupá-los na outra ponta.
- ▶ É possível intercalar frames de streams diferentes para depois reagrupá-los usando informações de seu cabeçalho.
- ▶ Isto acarreta em tempos mais curtos de carregamento de páginas. Também permite priorizar determinadas requisições em detrimento de outras



- ▶ Seguindo na mesma linha de raciocínio do HTTP/2, o HTTP/3 busca avançar mais ainda na performance do protocolo HTTP.
- ▶ Ainda não possui especificação, apenas em versão *draft*, atualizado em 21 de julho de 2021. Por isso, ainda não apresenta ampla adoção na Internet (apesar do Chrome e Firefox suportarem).
- ▶ Principal avanço é a adoção do protocolo QUIC, que é um protocolo de camada 3, em substituição ao TCP.
- ▶ O QUIC é baseado no UDP e possui mecanismos similares ao de multiplexação do HTTP/2, só que implementado diretamente na camada de transporte.
- ▶ Isto elimina conflitos e ambiguidades entre os mecanismos de tratamento de perda de pacote e controle de fluxo dos protocolos TCP e HTTP/2.

WebSockets

- ▶ A principal limitação do protocolo HTTP é a impossibilidade do servidor enviar dados não requisitados.
- ▶ Uma das soluções mais populares para este problema é o protocolo WebSockets (RFC 6455), que permite uma comunicação bidirecional de dados binários ou de texto.
- ▶ Apesar do nome, o Websockets é diferente da API sockets, possuindo muito mais funcionalidades. O nome deriva da versatilidade do protocolo Websockets, sendo o mecanismo disponível em um browser que mais se aproxima do sockets puro.
- ▶ O Websockets pode ser dividido em duas partes: o protocolo e a API.
- ▶ Outras soluções para envio de dados pelo servidor incluem: Server-Sent Events (SSE) e HTTP/2 server push.

- ▶ O WebSocket é um protocolo independente do HTTP, porém é possível utilizá-lo no browser a partir do processo de Handshake.
- ▶ O cliente deve iniciar a requisição da seguinte forma:

```
GET /chat HTTP/1.1
Host: example.com:8000
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhllHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
```

- ▶ Também pode ser especificados extensões e sub-protocolos.
- ▶ O servidor deve então responder com:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzshZrBk+xOo=
```

- ▶ onde `Sec-WebSocket-Accept` é obtido concatenando `Sec-WebSocket-Key` com "258EAF5-E914-47DA-95CA-C5AB0DC85B11", gerando o hash SHA-1 e convertendo com Base64.

Websockets - O formato do data frame

- ▶ Uma vez estabelecida a conexão WebSocket, tanto o cliente como o servidor podem enviar mensagens a qualquer momento.
- ▶ Estas mensagens são enviadas em frames, que podem conter no dados de texto (em UTF-8) ou binários no seu payload.
- ▶ Pode ser aplicada uma máscara no payload a partir de uma chave. Além disso, a mensagem pode ser fragmentada em múltiplos frames.
- ▶ As aplicações não “enxergam” as estrutura do frame, isto é gerenciado pelo protocolo.
- ▶ O formato dos frames é o seguinte:

Bit	+0..7		+8..15		+16..23	+24..31
0	FIN		Opcode	Mask	Length	<i>Extended length (0–8 bytes) ...</i>
32	...					
64	...				<i>Masking key (0–4 bytes) ...</i>	
96	...				<i>Payload ...</i>	
...	...					

Websockets - WebSocket API no cliente

- ▶ A API do WebSocket é utilizada em conjunção com o protocolo WebSocket no servidor. A especificação pode ser consultada em <https://www.w3.org/TR/websockets/>
- ▶ Como a API é disponibilizado pelo browser, ela é baseada em Javascript, que é a linguagem de programação compreendida pelo browser.
- ▶ A API é bastante simples, basta criar o objeto e atribuir funções de callback:

```
const ws = new WebSocket('wss://example.com/socket');

ws.onerror = function (error) { ... }
ws.onclose = function () { ... }

ws.onopen = function () {
  ws.send("Connection established. Hello server!");
}

ws.onmessage = function(msg) {
  if(msg.data instanceof Blob) {
    processBlob(msg.data);
  } else {
    processText(msg.data);
  }
}
```

Referências

- ▶ MDN Web Docs sobre HTTP:
<https://developer.mozilla.org/en-US/docs/Web/HTTP>
- ▶ Capítulos 12 e 17 do livro “High Performance Browser Networking” de Ilya Grigorik (disponível em <https://hpbn.co/>)

The End!