Web Services

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Ementa

- O que é um WebService
- O que é uma API
- WS x API
- SOAP x REST
- Criando um WebService SOAP
- Criando um WebService REST
- Criando um cliente para consumir os WebServices

WebService

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

WebService (O que é)

- É um conjunto de métodos que são invocados por outros programas utilizando tecnologias Web.
- Um WebService (WS) é utilizado para transferir dados através de protocolos utilizadas nessas plataformas.
- Não confundir WS com uma página Web, por exemplo, um WS transmite apenas informações.
- Um WS é muito utilizado em integrações entre sistemas.

WebService (Como Funciona)

- Em todo WS existe um protocolo de comunicação para que sistemas externos possam se “conversar”.

- Os dois mais utilizados são: **SOAP** e o **REST**

WebService (Benefícios)

- **Integração de informações entre Sistemas:** É possível realizar a troca de informações entre dois sistemas. Permitem ligar qualquer tipo de sistema, independente de plataforma (Windows, Linux e etc...) e independente de linguagem (Java, Perl, Python, Delphi e etc...).

- **Reutilização de Código:** Um WS pode ser utilizado por várias plataformas com diferentes objetivos de negócio. O código é feito uma única vez e utilizado várias vezes e em lugares diferentes.

- **Maior Segurança:** Um WS evita que se comuniquem diretamente com a base de dados.

API

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

API (O que é)

- É um acrônimo para ***Interface de Programação de Aplicativos*** que permite a comunicação entre dois softwares.

- Fornecem serviços de softwares que se comunicam com serviços de outros softwares sem precisar saber como ambos são implementados.

API x WS (Qual é a Diferença?)

- A principal diferença da API para o WS está no tipo de protocolo de comunicação. Enquanto o WS é usado para REST, SOAP e XML, a API é utilizada para qualquer padrão de comunicação.

API x WS (Qual é a Diferença?)

WebService

- É uma aplicação
- Todo WebService é uma API
- Protocolo de comunicação robusto e complexo, menos seguro
- Métodos customizados
- Não armazena os dados em Cache
- Linguagem complexa e difícil de integrar
- Necessário ter uma rede para o seu funcionamento

API

- É uma interface
- Todas as API não são WebService
- Arquitetura de Software com mais recursos, abordagem simplificada, com diretrizes mais simples e mais segura
- Métodos padronizados
- Armazena dados em Cache

- Não necessita de um rede para o seu funcionamento (Serverless)

SOAP

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

SOAP (O que é)

- SOAP (Simple Object Access Protocol – Procolo de acesso simples a objetos).
- Utiliza como mensagem documentos do tipo XML para a troca de informações e, geralmente, serve-se do protocolo HTTP/HTTPS para transportar tais dados.
- Juntamente com o protocolo SOAP temos o documento WSDL (Web Service Definition Language), que descreve a localização do WS, as operações disponíveis e informações para realizar a comunicação.

SOAP (Como Testar?)

- Software mais popular para testar WS SOAP's:
- https://www.soapui.org/downloads/soapui/

REST

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

REST (O que é)

- REST (Representational State Transfer – Transferência de Estado Representacional)
- É um protocolo de comunicação mais recente que veio para simplificar a comunicação com os WS's.
- Também se baseia no protocolo HTTP/HTTPS e permite utilizar vários formatos de documentos na troca mensagem, como: JSON, XML, RSS e outros.
- É um dos protocolos mais rápidos e com maior capacidade de transmissão de dados.

REST (Como Testar?)

- Software mais popular para testar WS REST:
- https://www.postman.com/downloads/

Exercícios

01 – Instalando os Clientes para SOAP e REST

02 – Testando um SOAP com o SoapUI

03 – Testando um REST com o Insomnia

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Verbos HTTP

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Verbos HTTP

- Sua API deverá prover uma URL base e os clientes irão operar de acordo com os verbos HTTP. (www.dominio.com/rest/notas/)

- **Verbo GET:** Sem passagem de ID, retorna todos os dados daquele serviço/recurso. Com a passagem de ID, retorna um registro específico.
- **Verbo POST:** Cria um novo dado daquele serviço/recurso. Passando o ID como parâmetro.
- **Verbo DELETE:** Remove o dado daquele serviço/recurso. Passando o ID como parâmetro.
- **Verbo PUT:** Atualiza o dado daquele serviço/recurso. Passa o ID como parâmetro e passa os dados no body da requisição (normalmente no formato JSON).
- **Verbo PATH:** Atualiza o dado daquele serviço/recurso. Passa o ID como parâmetro e passa somente os dados atualizados no body da requisição (normalmente no formato JSON).

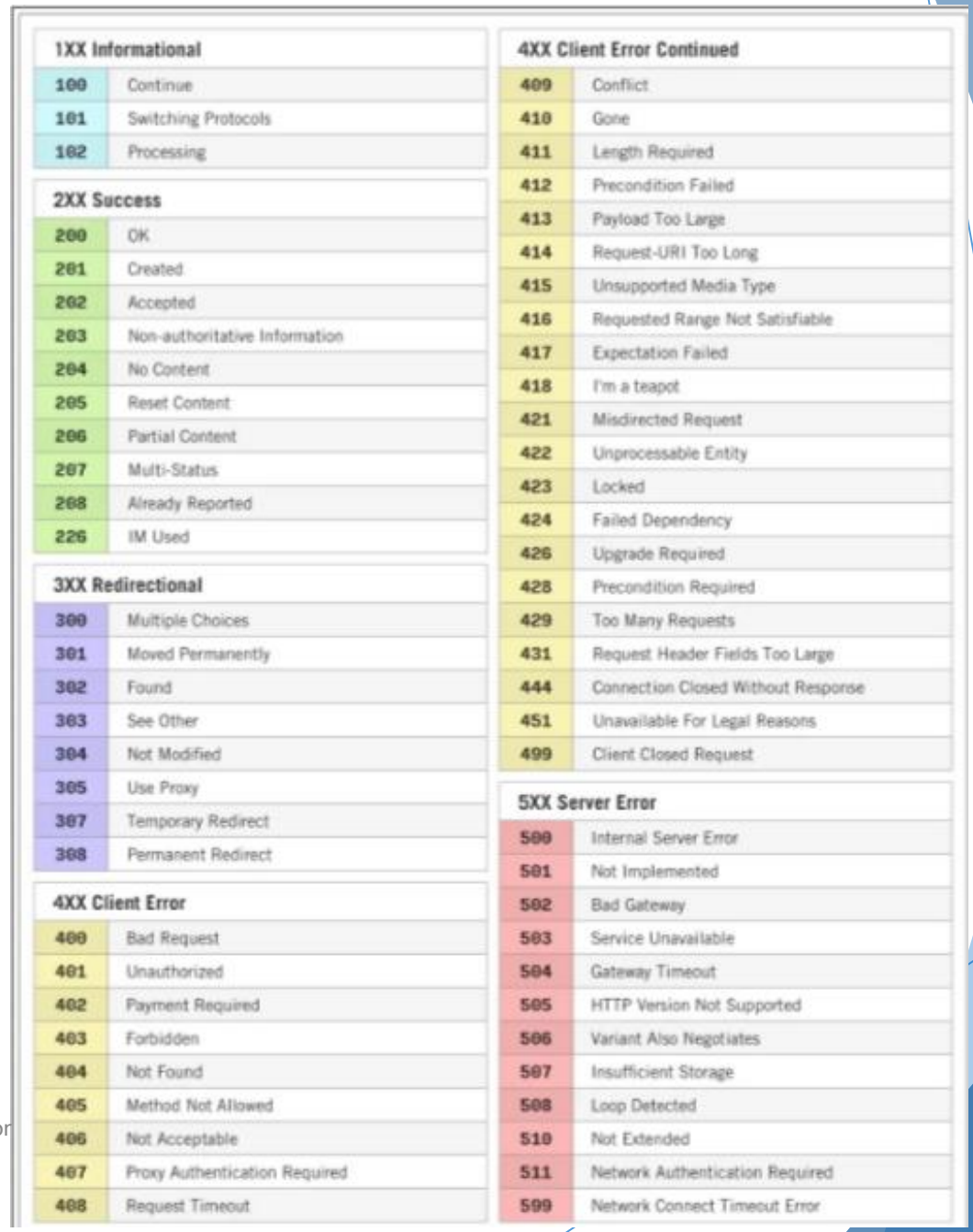
Status Code HTTP

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Status Code HTTP

Os mais comuns que vamos ver:

- 200, 400, 404 e 500.



1XX Informational	
100	Continue
101	Switching Protocols
102	Processing

2XX Success	
200	OK
201	Created
202	Accepted
203	Non-authoritative Information
204	No Content
205	Reset Content
206	Partial Content
207	Multi-Status
208	Already Reported
226	IM Used

3XX Redirectional	
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect
308	Permanent Redirect

4XX Client Error	
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout

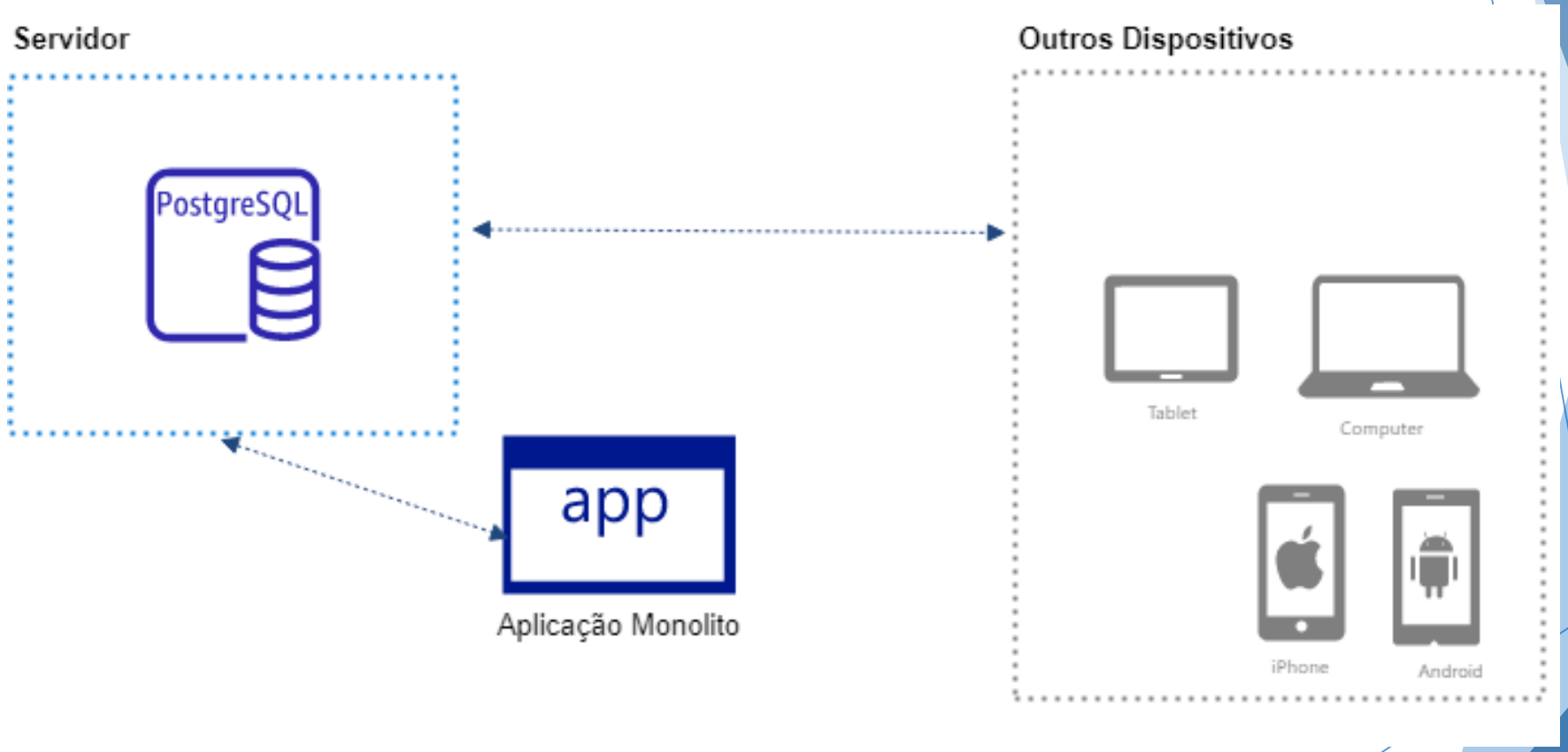
4XX Client Error Continued	
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Payload Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed
418	I'm a teapot
421	Misdirected Request
422	Unprocessable Entity
423	Locked
424	Failed Dependency
426	Upgrade Required
428	Precondition Required
429	Too Many Requests
431	Request Header Fields Too Large
444	Connection Closed Without Response
451	Unavailable For Legal Reasons
499	Client Closed Request

5XX Server Error	
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported
506	Variant Also Negotiates
507	Insufficient Storage
508	Loop Detected
510	Not Extended
511	Network Authentication Required
599	Network Connect Timeout Error

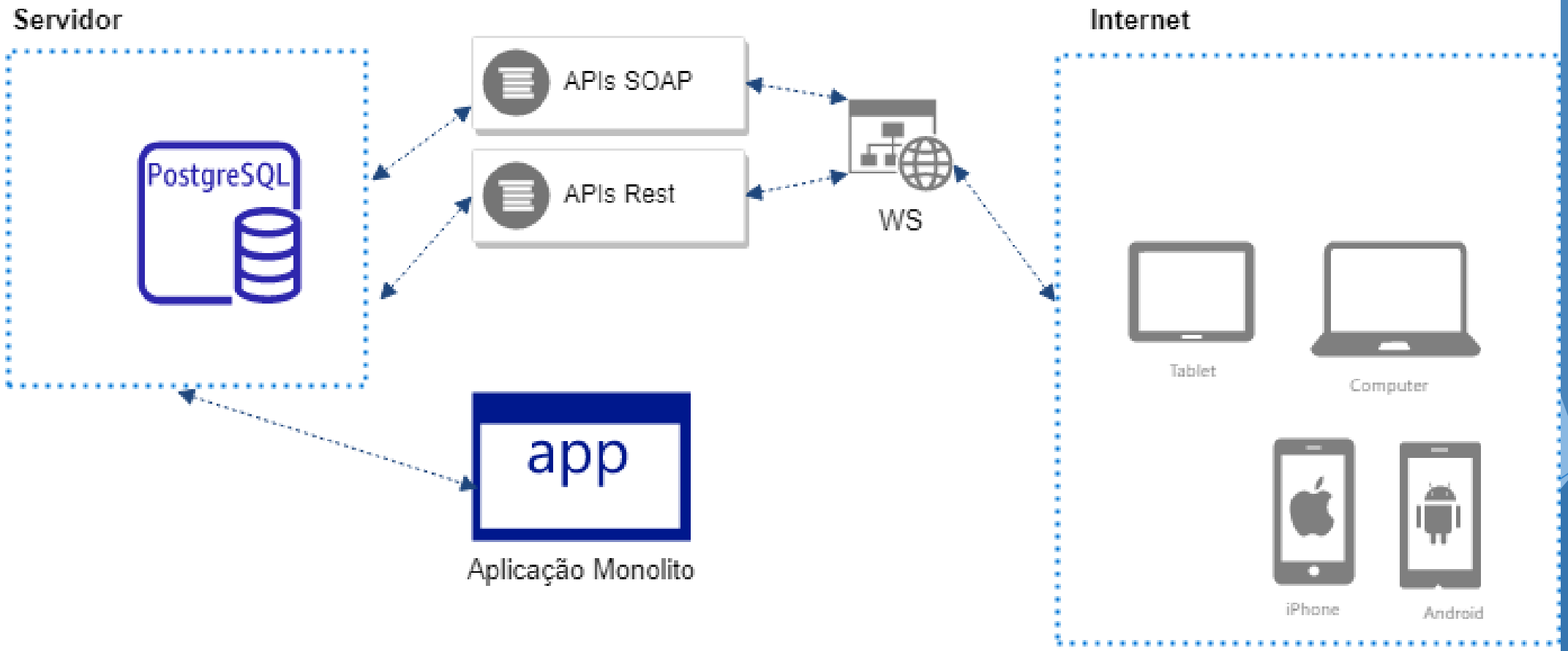
Arquiteturas

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Arquitetura Client/Server – Sem WS



Arquitetura Cloud – Com WS

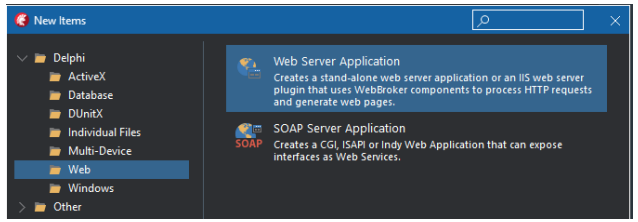


Delphi e WebServices

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Delphi e WebServices

- Alguns Frameworks para criação de WebServices no Delphi:



<- SOAP - Nativo

RAD Server

DataSnap Servers

<- REST - Nativos

Horse

RDW
RESTDATAWARE
COMPONENTS

<- REST – Open Source

Criando um WS em SOAP

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Exercícios

04 – Criando o primeiro WS SOAP

05 – WS de uma Calculadora

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Criando um WS em REST

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Exercícios

06 – Criando o primeiro WS REST

07 – WS de uma Calculadora

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Criando um Client

Para consumir WS em SOAP

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Exercícios

08 – Consumir o Hello World

09 – Consumir a Calculadora

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Criando um Client

Para consumir WS em REST

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Exercícios

10 – Consumir o Hello World

11 – Consumir a Calculadora

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Trafegando JSON

API REST

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

O que é JSON

- É um formato de representação de dados **baseado** no Javascript.
- Composto por **chave/valor,** onde a chave é o nome do atributo e o valor é o valor atribuído a essa chave.
- Formato leve para realizarmos trocar de informações entre APIs.

Tipos de Dados

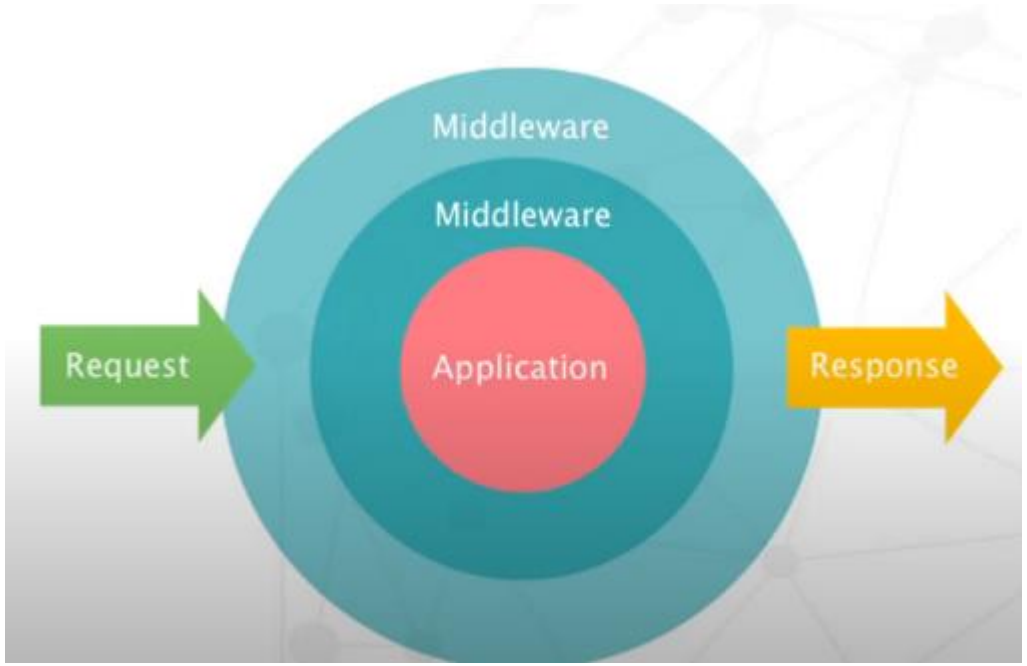
- **String:** Separados por aspas (duplas ou simples). Ex.: **“Devs2Blu”** ou **‘Devs2Blu’**.
- **Número:** Sem aspas e pode ser inteiro ou real. Ex.: **1 (inteiro)** ou **10.75 (real)**.
- **Booleano:** Tipo lógico. Ex.: **true** ou **false**.
- **Nulo:** Valor nulo. Ex.: **{ “nome”: null }**.

Benefícios - JSON

- Leitura mais simples (em comparação com XML, HTML e outros).
- Analisador mais fácil.
- Velocidade na transmissão dos dados.
- Arquivo com tamanho reduzido.
- Grandes empresas utilizam JSON nas APIs.

Middleware - Jhonson

- Recurso com capacidade para realizar o parsing de JSON para o HORSE.



Exercícios

12 – Exemplo GET, POST e DELETE (Pessoas)

13 – GET, POST e DELETE (Cargo)

***Requisitos biblioteca Jhonson**

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Basic Authentication

API REST

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Autenticação x Autorização

- **Autenticação:** é como se você fosse para uma festa e o segurança exigisse suas credenciais como nome na lista e CPF para liberar sua entrada.
- **Autorização:** é como se você já estivesse dentro da festa e quisesse subir ao palco do show, mas para isso você precisaria ser autorizado pelo segurança.

Basic Authentication

- Tipo de autenticação bem **simples** especificado no protocolo HTTP.

- O cliente envia uma requisição com o header (**Authorization**) que contém a palavra **Basic** e o nome de usuário e senha, separados por dois pontos no formato **base64**.

- Por exemplo, para autorizar o usuário **armando** e com a senha **123456**, o client enviaria no header:

Authorization: Basic YXJtYW5kbzoxMjM0NTY=

Práticas de Segurança

- **Não exponha dados sensíveis na URL:** https://api.test.com/orders/?apiKey=123456

- **Valide os parâmetros da requisição:** Sempre valide os parâmetros da requisição antes de processar sua lógica de negócio. Caso exista algum parâmetro fora da especificação da sua API rejeite a chamada de quem solicitou.

- **Sempre crie API usando o protocolo HTTPS**

- **Utilize o Rate Limit:** Isso é uma prática para evitar a sobrecarga de requisições à sua API. (O Horse tem um Middleware para utilizarmos)

Exercícios

14 – Usando o Basic Authentication

***Requisitos biblioteca Basic-Auth**

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Compression

API REST

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Importância do Compression

- Forma de reduzir o tamanho dos dados trafegados. Sem alterar os dados originais.
- Velocidade no tempo de processamento.
- Diminuição de custo em Clouds.

Accept-Encoding

- O header **Accept-Encoding** indica qual codificação de conteúdo, usualmente um algoritmo de compressão, o client está apto a entender.

Header	Valor
Accept-Encoding	gzip
Accept-Encoding	gzip, compress, br
Accept-Encoding	br;q=1.0, gzip;q=0.8, *;q=0.1
Accept-Encoding	Compress
Accept-Encoding	*

Accept-Encoding

- O servidor seleciona uma das propostas, e informa o client da escolha feita com a utilização do header **Content-Encoding** (Qual codificação o servidor utilizou).

- O servidor pode escolher **não comprimir** o corpo da resposta:

Caso a compressão não for capaz de reduzir o seu tamanho.

Caso o servidor hospedado já esteja sobrecarregado no seu poder computacional.

Accept-Encoding (Diretivas)

- **gzip**: Formato de compressão usando o Lempel-Ziv coding (LZ77), com CRC de 32-bits.
- **compress**: Formato de compressão usando o algoritmo Lempel-Ziv-Welch (LZW).
- **deflate**: Formato de compressão usando a estrutura zlib, com o algoritmo de compressão deflate.
- **br**: Formato de compressão usando o algoritmo de *Brotli*.
- **identity**: Nenhuma transformação é usada.
- ***: Aceita qualquer codificação.

Content-Encoding

- Indica quais codificações foram aplicadas no corpo da resposta da requisição:

Header	Valor
Content-Encoding	gzip
Content-Encoding	compress
Content-Encoding	deflate
Content-Encoding	gzip, identity
Content-Encoding	deflate, gzip

Exemplo

- No lado do client, você pode anunciar uma lista de esquemas de compactação que serão enviados no HTTP. O Accept-Encoding é usado para negociar a codificação do conteúdo:

Accept-Encoding: gzip, deflate

- O Servidor responde com o esquema usado, indicado pelo Content-Encoding:

Content-Encoding: gzip

Benefícios

- Informações extraídas da página do projeto do Horse:

🚀Estatisticas

Usando middleware, a resposta foi aproximadamente 67 vezes menor. Os dados foram coletados usando o projeto disponível na pasta samples (delphi) . Por padrão, as respostas menores ou iguais a 1024 bytes não serão compactadas.

Propriedade	Sem	Com
Tempo (ms)	108	126
Tamanho(bytes)	15.770	236

Exercícios

15 – Usando o Compression

***Requisitos biblioteca Compression**

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Trafegando Stream

API REST

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

PDFs, Imagens, Arquivos e etc...

- Converter em base64? (depende muito pois vai aumentar o tamanho do arquivo).
- Trafegar em Stream.
- Utilizar o form-data da requisição.
- Gerar um link público do meu servidor (porém menos seguro).

Exercícios

16 – Usando o Stream

***Requisitos biblioteca Stream**

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

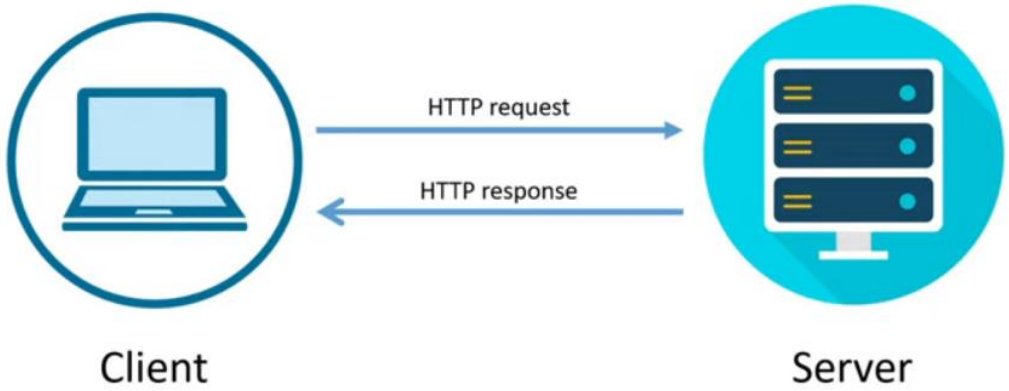
Swagger

Documentação de API REST

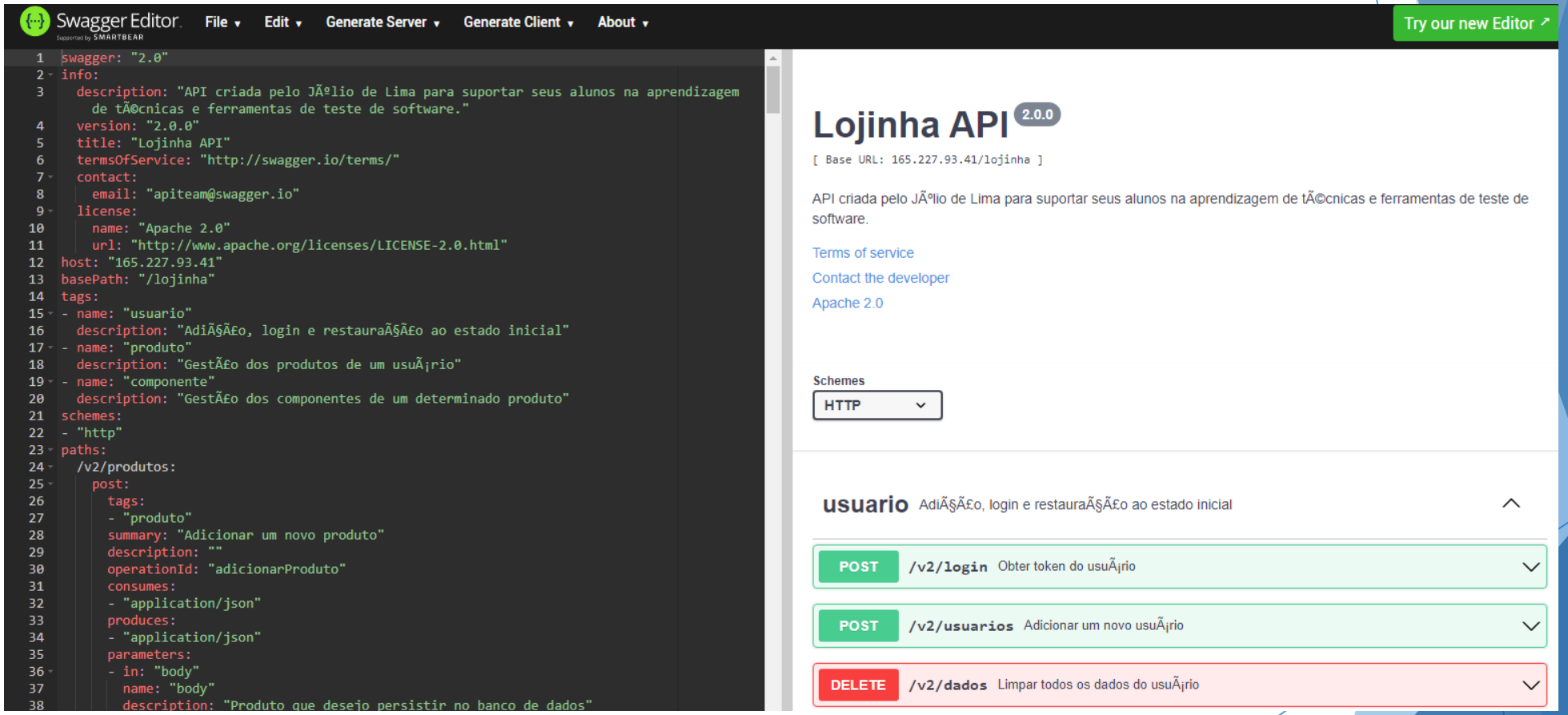
Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Requisição

- Verbo HTTP
- URL
- Parâmetros
- Headers
- Body
- Response
- Status Code



https://editor.swagger.io/



Exercícios

17 – Criando API e Documentando com o Swagger

18 – Documentando o recurso Cargo

***Requisitos biblioteca Swagger**

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

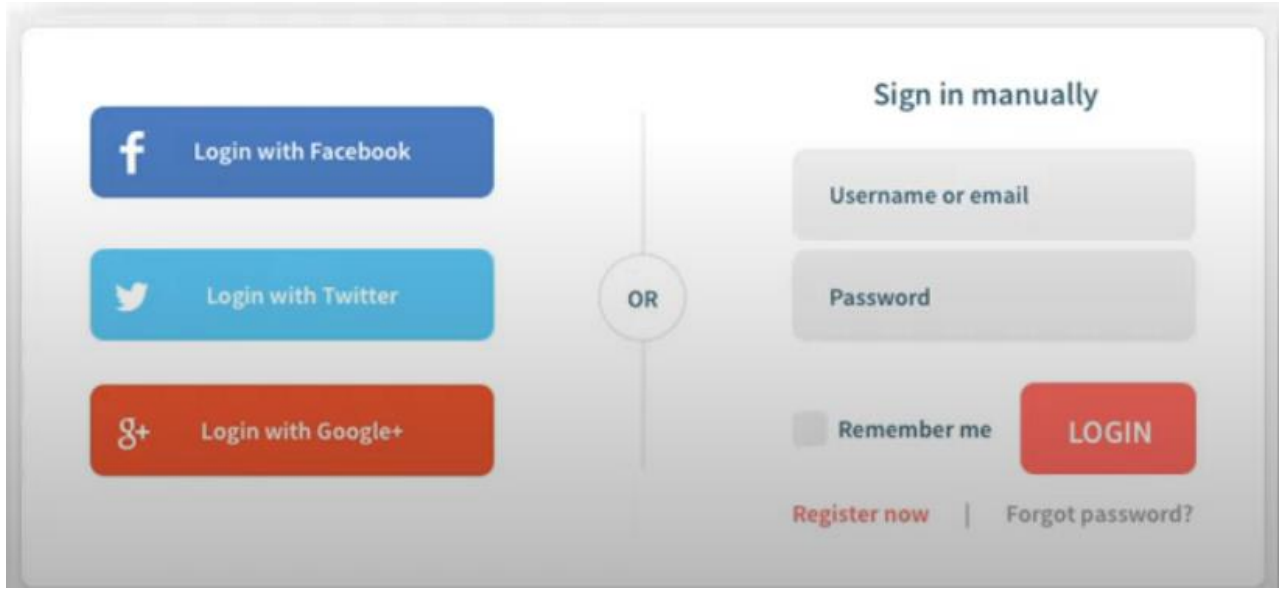
Token JWT

API REST

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Principais Métodos de Autenticação

- **OAuth:** É um tipo de método de **autorização** de acesso, apesar de ter também um fluxo de **autenticação**. Ele se assemelha ao JWT, mas com um tipo de camada de aplicação própria que permite o uso desse tipo de autenticação através de sites de terceiros.



O que é o JWT?

- **JSON Web Token:** É uma convenção aberta (**RFC 7519**) que possibilita de uma maneira compacta e autocontida, **transmitir com segurança informação entre duas partes no formato de um objeto JSON.**
- **As informações podem ser verificadas e confiadas porque elas são assinadas** digitalmente por criptografia utilizando uma chave secreta (HMAC) ou par chaves pública/privada utilizando RSA ou ECDSA.
- O cliente envia uma requisição com o header (Authorization) que contém a palavra Bearer e o JWT

```
Authorization: Bearer dGhpYWdvOnRofh35sDfhgs4fGsnGgdadSaWFnb0AxMjM=
```

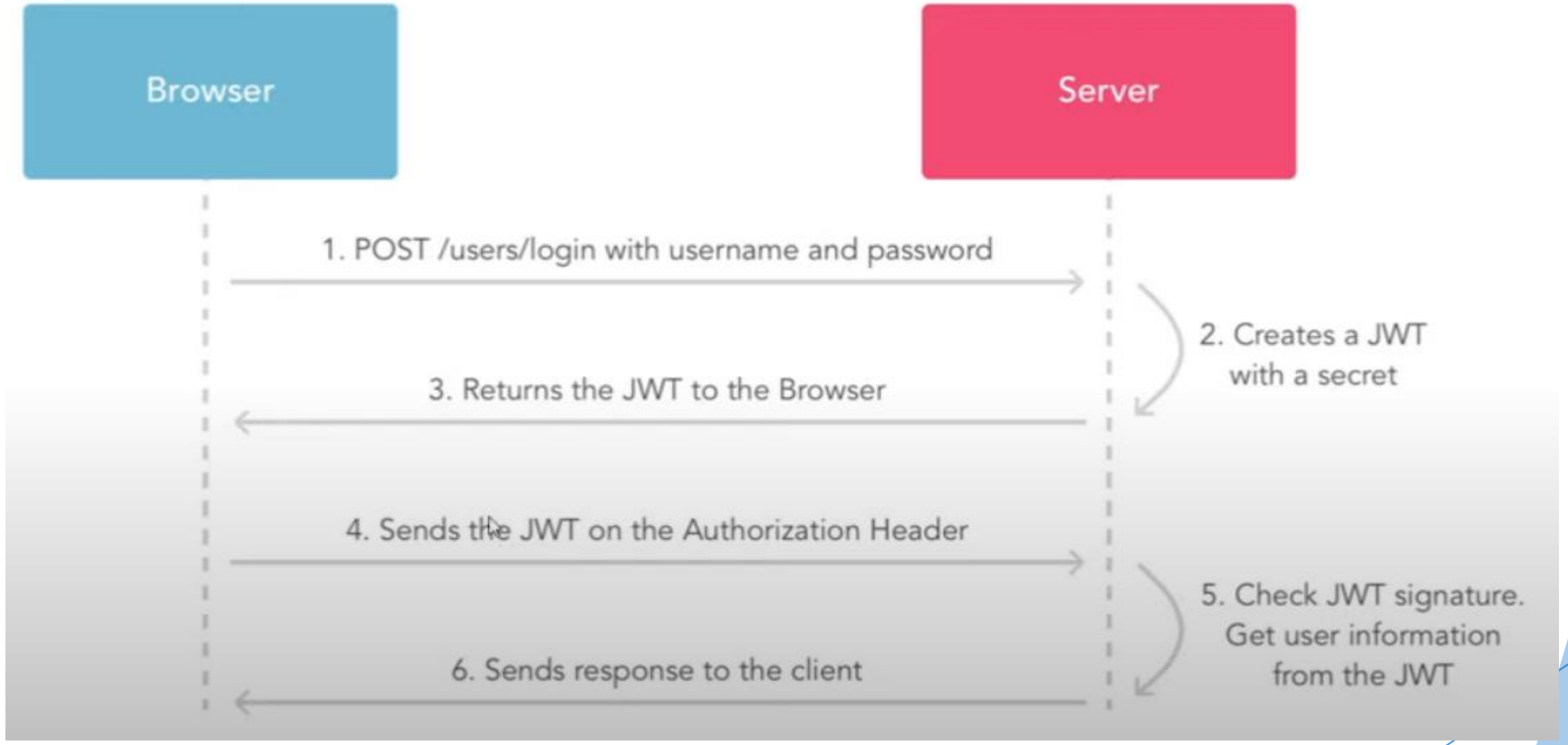
Por que usar?

- O JWT é útil em diversos cenários, porém os dois comuns são:

- **Autenticação:** O token é utilizado para verificar a identidade de um usuário e suas permissões.

- **Troca de Informação:** Por ser um meio seguro para duas aplicações conversarem, eles garantem a identidade das partes envolvidas e se a informação não foi alterada no meio do caminho.

Fluxo para Utilizar o Token



Mais um pouco sobre token...

- O token possui tempo de uso.
- Normalmente criamos um RefreshToken.
- Rules (cadastro de produtos, clientes e etc...)

Exercícios

19 – Usando o JWT

20 – Client para utilizar o JWT

***Requisitos bibliotecas Cors, JOSE, HashLib4Pascal e JWT**

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Exercício Geral

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br

Horse e Banco de Dados

22 – Retornando dados do banco

23 – CRUD – Produtos

24 – CRUD – Unidade de Medida

***Requisitos biblioteca horse-query e dataset-serialize**

Prof.: Armando Neto - kikobatery@hotmail.com - www.armandoneto.dev.br