TÓPICO 07 - WEB SERVICES

Disciplina de Backend - Professor Ramon Venson - SATC 2026.1

Web Services

Web Services são serviços que podem ser acessados pela **Web**.

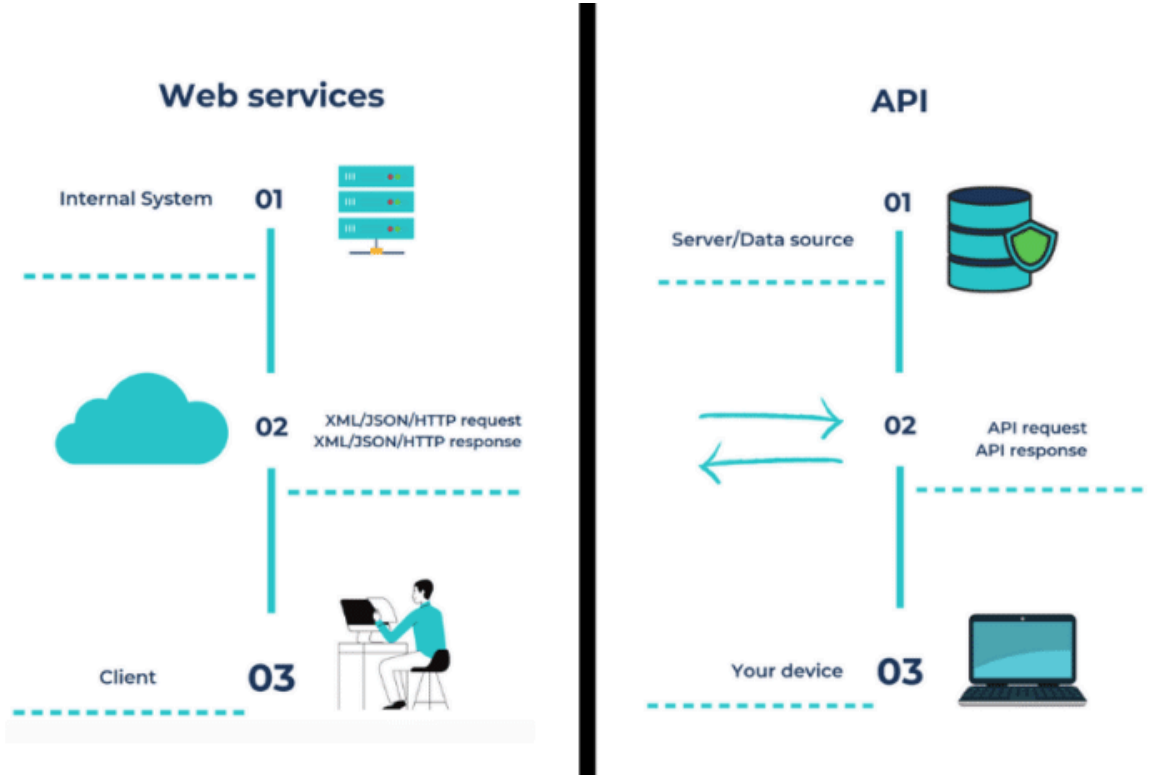
Um web service permite que **diferentes tipos de clientes acessem o serviço**, mesmo em diferentes plataformas.

Características

Um web service possui as seguintes características:

- **Interoperabilidade**, trabalhando com diferentes tipos de clientes
- **Portabilidade**, podendo ser implantado em diferentes ambientes com mínimo impacto para clientes e integrações.
- **Escalabilidade**, podendo atender múltiplos usuários.

Web Services e Web APIs



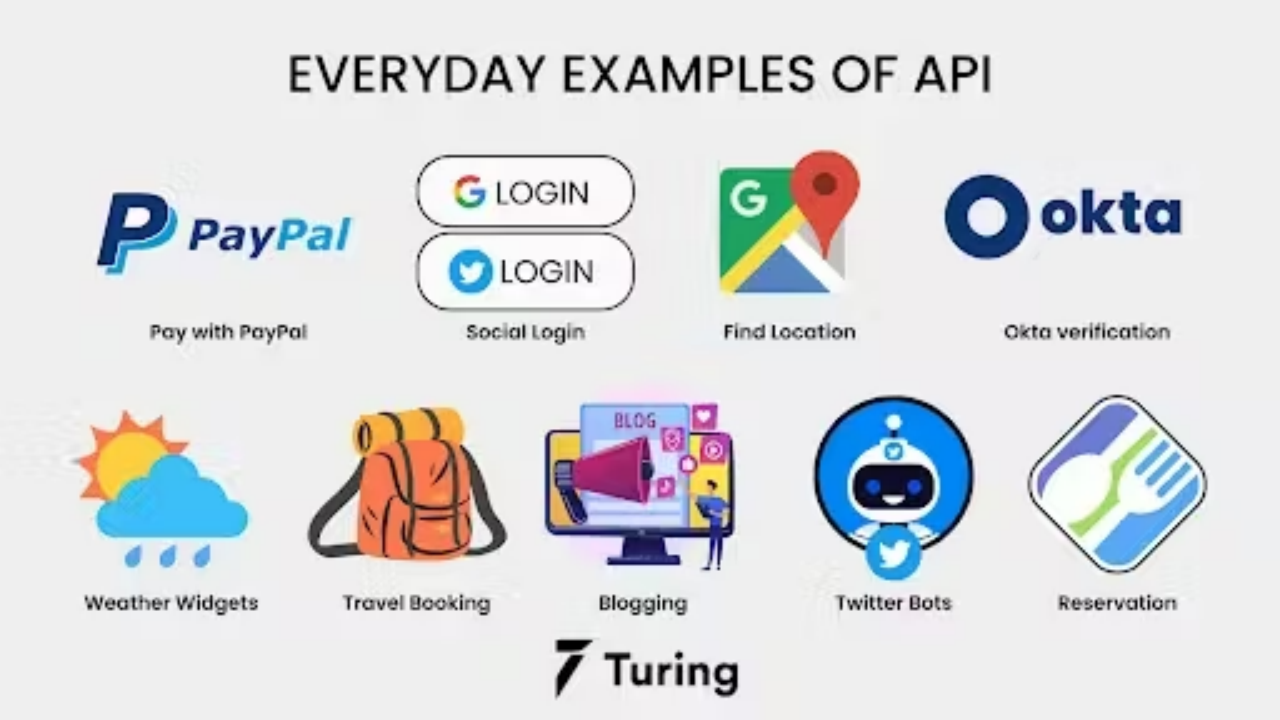
Web API é um termo mais comum para descrever os serviços que fornecem dados para outros sistemas.

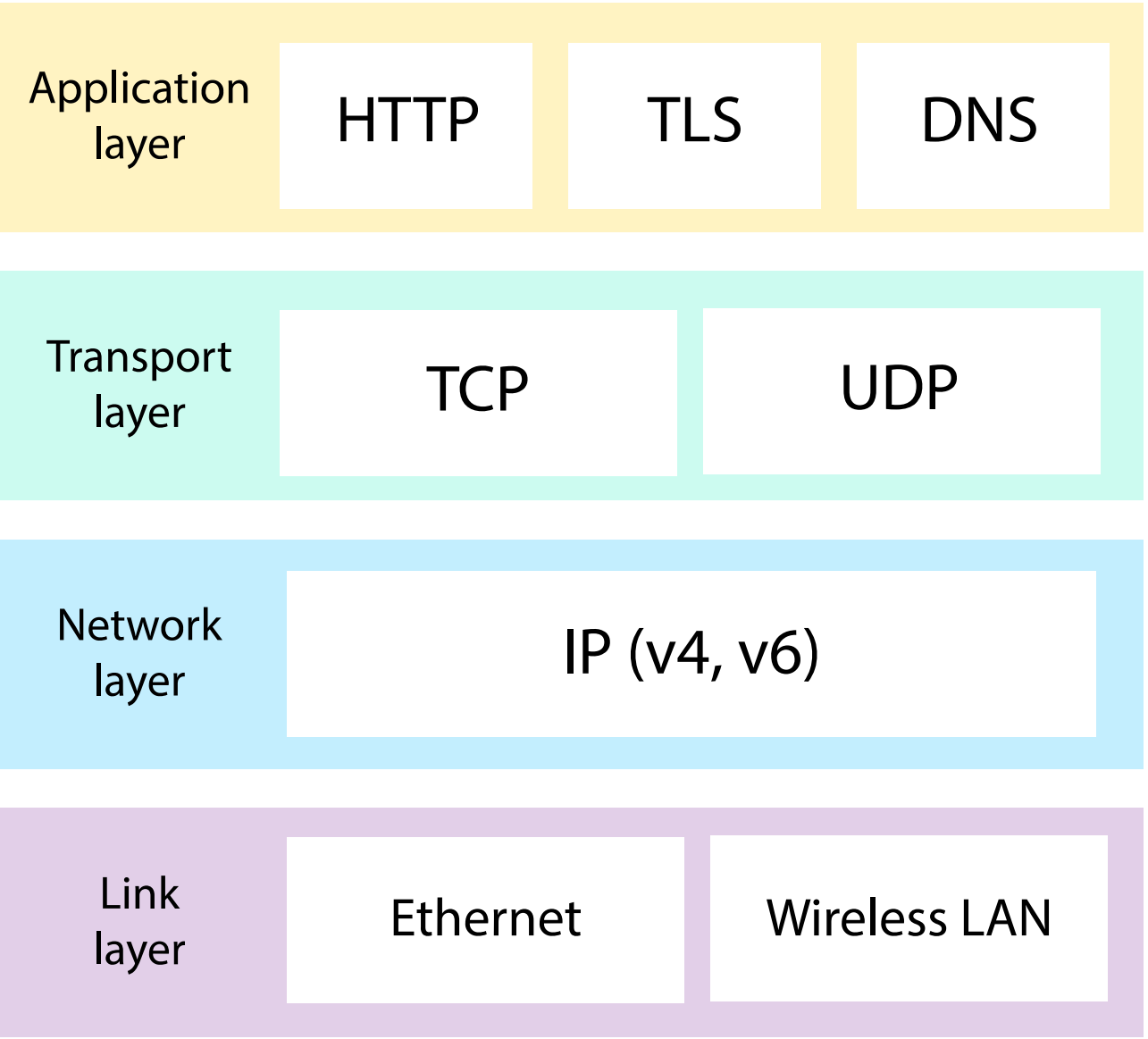
Estabelecendo um padrão de comunicação público, essas APIs são consumidas por diferentes clientes.

Web API populares

São muitos os web services famosos que podemos acessar, como:

- Google Maps, para mapas
- OpenWeather, para previsão do tempo
- OpenAI, para LLMs
- GitHub, para repositórios
- Stripe, para pagamentos





Protocolos de Comunicação

Web services podem ser acessados por meio de diferentes protocolos de comunicação, como:

- `HTTP` e `HTTPS`

Formatos de Dados

Dado que web services são acessados por meio de diferentes plataformas, é necessário que os dados sejam enviados e recebidos em um formato padronizado. Esses formatos incluem:

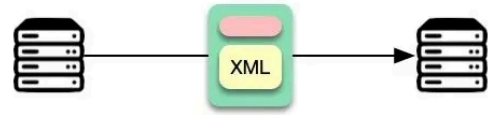
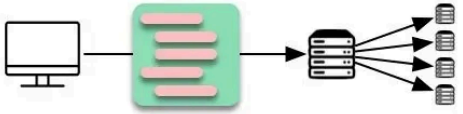
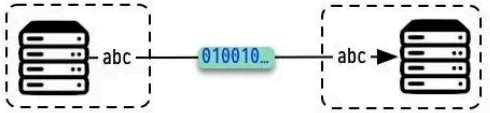
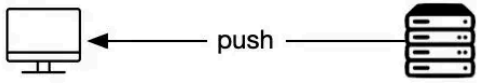
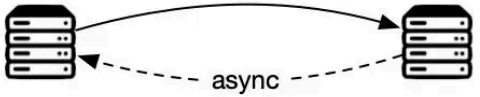
- `XML`
- `JSON`
- `YAML`
- `CSV`
- `Plain Text`

Arquiteturas

Web services podem ser implementados usando diferentes arquiteturas, como:

- `SOAP`
- `REST`
- `gRPC`
- `GraphQL`
- Entre outros

Top 6 Most Popular API Architecture Styles — ByteByteGo.com

Style	Illustration	Use Cases
SOAP		XML-based for enterprise applications
RESTful		Resource-based for web servers
GraphQL		Query language reduce network load
gRPC		High performance for microservices
WebSocket		Bi-directional for low-latency data exchange
Webhook		Asynchronous for event-driven application

Comunicação em tempo real

`WebSocket` não é uma arquitetura no mesmo nível de `REST` ou `SOAP`.

Ele é um protocolo para comunicação bidirecional e persistente, útil em cenários como:

- Chats
- Jogos online
- Notificações em tempo real
- Dashboards ao vivo

Comparando abordagens

Tecnologia	Formato comum	Melhor uso
`REST`	JSON	CRUD e APIs web tradicionais
`SOAP`	XML	Integrações corporativas e contratos formais
`gRPC`	Protobuf	Comunicação entre serviços com alta performance
`GraphQL`	JSON	Consultas flexíveis e múltiplos clientes
`WebSocket`	Frames	Atualizações em tempo real

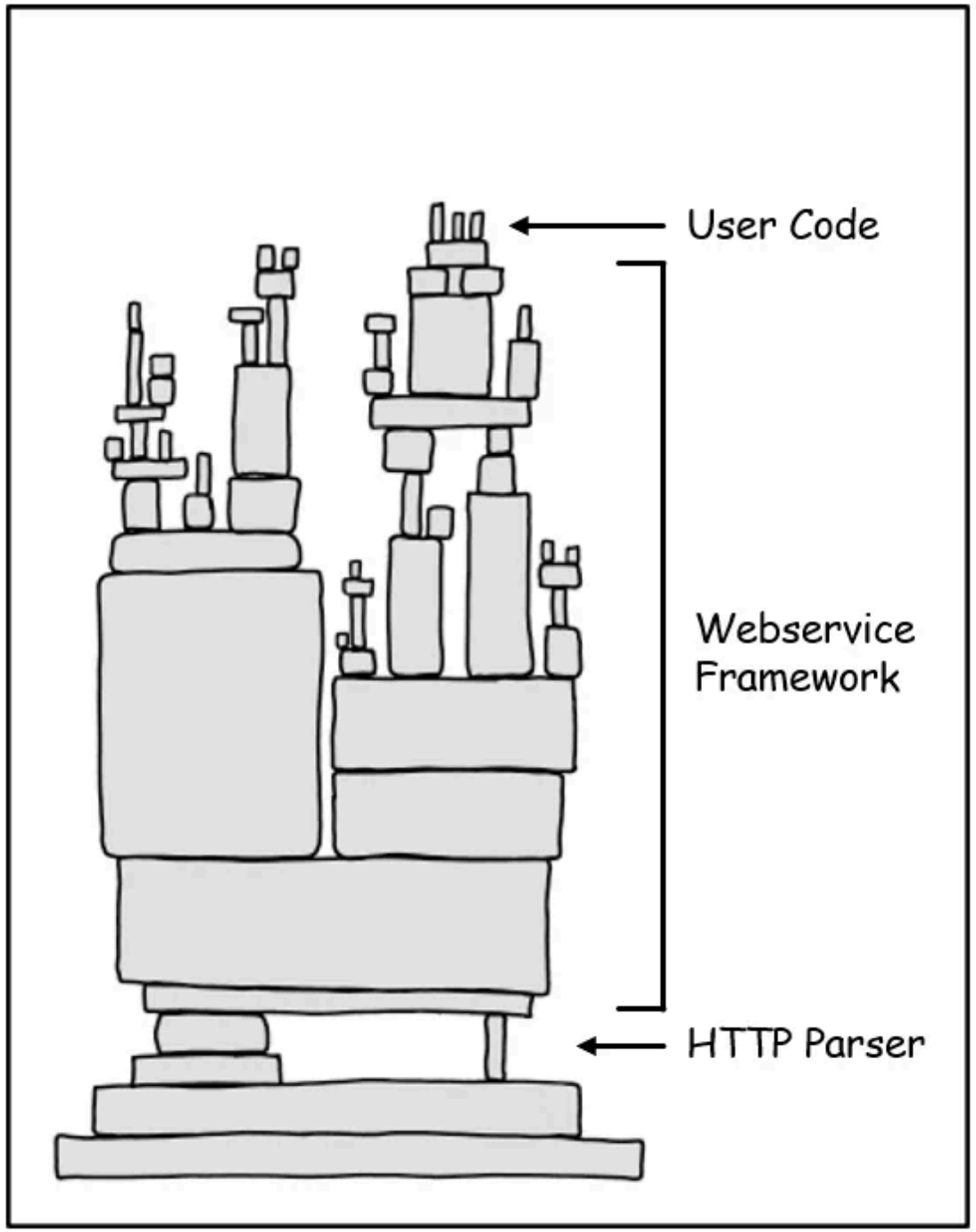
Exemplo de contrato de API

```
POST /produtos HTTP/1.1
Host: api.exemplo.com
Content-Type: application/json

{
  "nome": "Mouse Gamer",
  "preco": 149.90
}
```

Resposta esperada

```
HTTP/1.1 201 Created
Content-Type: application/json

{
  "id": 101,
  "nome": "Mouse Gamer",
  "preco": 149.90
}
```

Esse contrato define rota, método, formato e status esperado.



Frameworks para Web Services

Existem diversos frameworks que facilitam o desenvolvimento de web services. Exemplos são:

- `Spring Web`
- `NestJS`
- `Express`
- `Flask`
- `Laravel`

Spring Web

Exemplo:

```
@RestController
public class HelloWorldController {
    @GetMapping("/")
    public String olaMundo() {
        return "Olá mundo";
    }
}
```

NestJS

Exemplo:

```
import { Controller, Get } from '@nestjs/common';
@Controller()
export class AppController {
  @Get()
  getHello(): string {
  return 'Hello World!';
  }
}
```

Express

Exemplo:

```
const express = require('express')
const app = express()
const port = 3000
app.get('/', (req, res) => {
  res.send('Hello World!')
})
app.listen(port, () => {
    console.log(`Example app listening on port ${port}`)
}
```

Flask

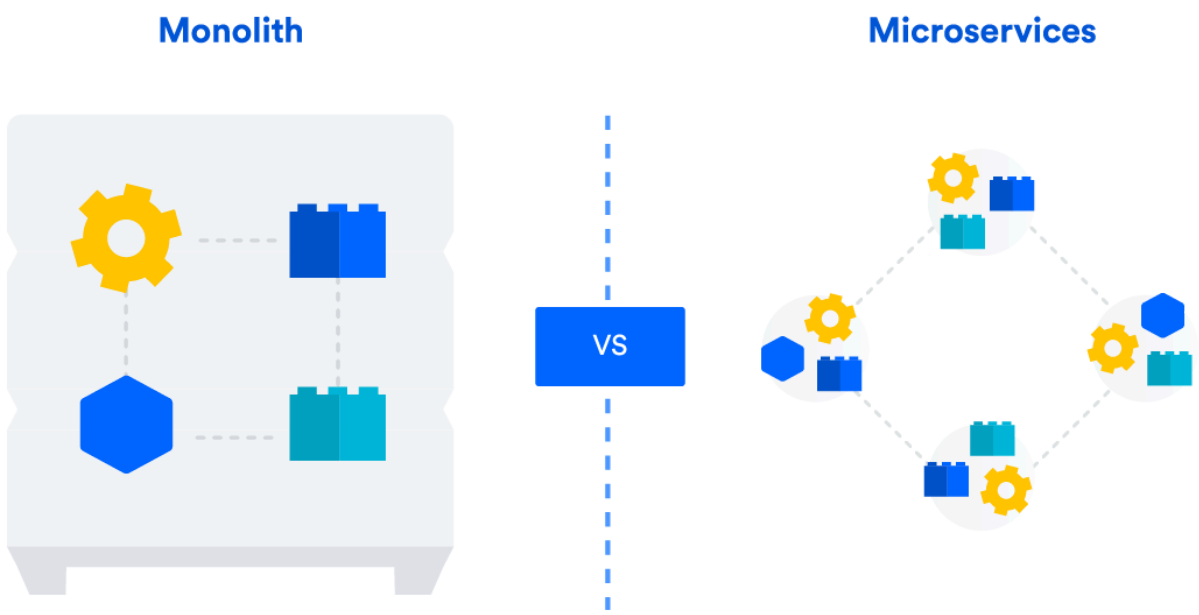
Exemplo:

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello_world():
    return 'Hello, World!'
```

Laravel

Exemplo:

```php
<?php
class PostController extends Controller
{
    public function index(): JsonResponse
    {
        return response()->json("Olá mundo!", 200);
    }
}
?>
```



Microsserviços e Monolitos

Existem dois tipos de arquitetura de software frequentemente utilizados na construção de web services.

Enquanto **monolitos** são sistemas completos, com múltiplos domínios, **microsserviços** são sistemas menores que trabalham em conjunto.

Deploy de Web Services

Para disponibilizar um web service para acesso público, é necessário hospedar em um servidor. Isso pode ser feito em diferentes tipos de infraestrutura:

- `Servidores Físicos` - Configurando e gerenciando um servidor físico;
- `Servidores Virtuais Privados (VPS)` - Utilizando um servidor virtual contratado;
- `Serviços de Nuvem` - Com diferentes provedores de nuvem (AWS, Azure, GCloud...)

Material de Apoio

- What's API?