

# Programação Web 1

*Application Programming Interface*

# Objetivo de Aprendizagem

- Conhecer o conceito de API
- Construir as primeiras APIs em Nodejs

# Agenda

- Conceito
- Arquiteturas Típicas
- REST
- Criando a primeira API

Conceito

"Conjunto de rotinas que realiza comunicação entre aplicações para compartilhamento de rotinas, mensagens e protocolo, transparente ao usuário. Permite a integração de rotinas em uma interface única."

Fonte: [schoolofnet.com](http://schoolofnet.com)

# API

- *Application Programming Interface*
- A ideia central das APIs é compartilhar informação sem compartilhar código
- Cada API pode ser desenvolvida de forma independente
- As APIs precisam apenas de uma interface comum para trocar informações

# Exemplo de API

O sistema de *software* do instituto meteorológico contém dados diários. A aplicação no celular “conversa” com esse sistema por meio de APIs e mostra atualizações meteorológicas no app do telefone.

# Vantagens

- Integração
- Expansão
- Facilidade de manutenção
- Inovação
- Código e dados ocultos ao usuário



# Integração

- APIs integram novas aplicações com sistemas existentes
- Desenvolvimento mais rápido
- Aproveitam código existente

# Expansão

- APIs podem ser usadas em várias plataformas
- Ex: integração de mapas em sites, Android, iOS, etc.

# Facilidade de Manutenção

- A API atua como um *gateway* entre sistemas
- Alterações em uma parte não afetam a outra

# Inovação

- Setores inteiros mudam com novas aplicações
- Alterações rápidas via API, sem reescrever o código de toda a aplicação

# Arquiteturas Típicas

# Arquiteturas Típicas

- Arquitetura Cliente/Servidor
- Cliente envia requisição, servidor responde
- Ex: app de clima (cliente) e banco de dados (servidor)

# Exemplos de Arquiteturas

- SOAP
- *Remote Procedure Call* (RPC)
- *WebSocket*
- REST

# SOAP

- *Simple Object Access Protocol*
- Usa XML
- Complexo, pouco flexível
- Popular no início das APIs



# RPC

- Remote Procedure Call
- Funções no servidor acessadas via chamadas remotas

# WebSocket

- Arquitetura moderna
- Usa JSON
- Mais eficiente que REST
- Menos flexível

REST

# *Representational State Transfer*

- Mais popular e flexível
- Usa métodos HTTP
- **Stateless**
  - Não há registro entre as requisições
- **Layered System**
  - Múltiplas camadas interagem

# Métodos HTTP no REST

- GET, POST, PUT, DELETE
- Ferramentas para uso/teste:
  - Postman

# APIs Públicas

Repositório

REST APIs

---

# Desafios

- Definição de *endpoint*
- Versionamento
- Autenticação
  - HTTP
  - API keys
  - OAuth

# Segurança

- Ausência de métodos de autenticação e controle
- Dados não encriptados
- Comprometimento de chaves



# Primeira API

# Funcionamento de APIs RESTful

## Cliente/Servidor Requisição/Resposta

1. O cliente envia uma solicitação ao servidor. O cliente segue a documentação da API para formatar a solicitação de modo que o servidor entenda.
2. Servidor autentica o cliente e confirma que o cliente tem o direito de fazer essa solicitação.
3. Servidor recebe a solicitação e a processa internamente.
4. Servidor retorna uma resposta ao cliente. A resposta contém informações que indicam ao cliente se a solicitação foi bem-sucedida. A resposta também inclui informações solicitadas pelo cliente.

# Requisição

- *Endpoint* (URL)
- Método HTTP
- Parâmetros
  - Através da URL
  - *Cookies*

# Resposta

- Cabeçalho HTTP
- Status HTTP
  - 200: resposta genérica de êxito
  - 201: resposta de êxito do método POST
  - 400: solicitação incorreta que o servidor não pode processar
  - 404: recurso não encontrado
- Corpo da mensagem (JSON)

```
{  
  "name": "John",  
  "age": 30  
}
```

# Primeira API

v1 da API Hello.

1. `npm init -y`
2. `npm install express`
3. Criar o arquivo `app.js`

# app.js

```
1  const express = require('express');
2  const app = express();
3  const PORT = 3000;
4  app.listen(PORT, ()⇒ {
5      console.log(`Hello API on port ${PORT}`);
6  })
```

# Primeira API

/v1/hi

4. Criar o primeiro *endpoint*: /v1/hi/ A resposta esperada é uma mensagem em formato JSON a ser exibida no navegador do usuário em resposta a uma requisição GET

```
1  {  
2    "msg": "Hello, World!"  
3  }
```

```
1  app.get('/v1/hi', function(req, res) {  
2    const out = {  
3      msg: "Hello, world!"  
4    }  
5    res.json(out)  
6  })
```

# Referências






Prof. José Roberto Bezerra

[jbroberto@ifce.edu.br](mailto:jbroberto@ifce.edu.br)

---

Powered by  Slidify