

Programação Web 1

Application Programming Interface

Objetivo de Aprendizagem

- Conhecer o conceito de API
- Construir a primeira API em Nodejs

Agenda

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

Conceito

API

“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

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 = 8000;
4  app.listen(PORT, ()⇒ {
5      console.log(`Hello API on port ${PORT}`);
6  })
```

API Hello

`/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  }
```

```
13  app.get('/v1/hi', function(req, res) {  
14    const out = {  
15      msg: "Hello, world!"  
16    }  
17    res.status(200).json(out)  
18  })
```

API Hello

`/v1/hi/user/:name`

5. Criar um *endpoint* para que a mensagem seja concatenada com o nome do usuário passado como parâmetro pela URL. 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, [name]"  
3  }
```

```
20  app.get('/v1/hi/user/:name', function(req, res) {  
21    const out = {  
22      msg: "Hello, " + req.params.name  
23    }  
24    res.status(200).json(out)  
25  })
```


API Hello

/v1/hi (POST)

6. Criar um *endpoint* que atenda requisições via método POST (formulários HTML). A resposta esperada deve ser a mesma do item 5.

- Para que a API possa obter dados a partir de um formulário é necessário adicionar processar caracteres codificados

```
28 app.use(express.urlencoded({ extended: true })))
29 app.post('/v1/hi', function(req, res) {
30     const { name } = req.body;
31
32     if (!name) {
33         return res.status(400).json({ error: "Name is required" });
34     }
35
36     const out = {
37         msg: `Hello, ${name.toUpperCase()} from POST!`,
38     };
39     res.status(200).json(out);
40 }
```

API Hello

`/*splat`

7. Criar um *endpoint* coringa (`/*splat`) para o caso de acesso a uma URL inexistente. A resposta esperada é uma mensagem de **erro** em formato JSON a ser exibida no navegador do usuário em resposta a uma requisição GET

```
1  {  
2    "error": "Invalid endpoint"  
3  }
```

```
42  // express v4 use '/*'  
43  // express v5 use '/*splat'  
44  app.get('*', function(req, res) {  
45    const err = {  
46      error: 'Invalid endpoint'  
47    }  
48    res.status(404).json(err)  
49  })
```

API Hello

Testando os *endpoints*

8. Inicie a aplicação (`node app`)
9. Utilize o navegador para acessar todos os *endpoints* em `http://localhost:8000`
 - `/v1/hi`
 - `/v1/hi/user/:name`
 - Qualquer outro (Erro)
10. Para testar o *endpoint* que atende o método POST é necessário usar um formulário **OU** o comando `curl`
 - Ver o arquivo `form.html` fornecido na Aula 11 - API do Google Classroom
 - Abra o arquivo no navegador, preencha o formulário e envie

API Hello

Testando os *endpoints*

10. Para testar o *endpoint* que atende o método POST é necessário usar um formulário **OU** o comando

```
curl
```

- **OU**

- No console digite `curl -d 'name=Prog%20Web%201'`

`http://localhost:8000/v1/hi`

- Note que os caracteres precisam ser codificados para URL (URL-encoded), por isso o espaço em branco está sendo substituído por `%20`. Para evitar isso, utilize:

- `curl --data-urlencode 'name=Prog Web 1'`

`http://localhost:8000/v1/hi`

Exercícios

1

Modificar a v1 API Hello de forma seja exibido um log de saída no console com o seguinte formato: data/hora, endpoint, saída (JSON).

2


Criar a versão 2 (v2) da API Hello de forma que as mensagens seja exibidas em 3 idiomas diferentes (pt-br, es, en). A opção do idioma pode ser passada através da URL e também através do formulário. As duas versões da API devem ficar disponíveis para o usuário.

Referências

- API Hello
- body-parser obsoleto

Prof. José Roberto Bezerra

jbroberto@ifce.edu.br

Powered by  Slidify