# MINICURSO

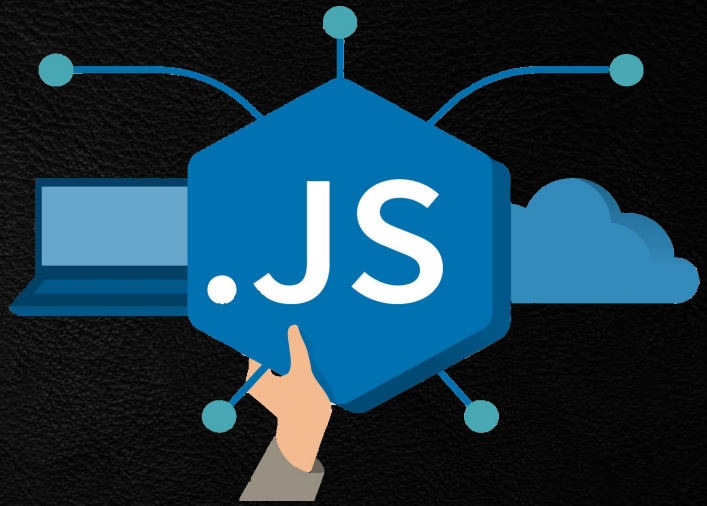
# Criando REST Api com Node.js e React

# Ementa

- JavaScript
  - O que é JavaScript
  - Revisão
  - ES6
  - Node
  - ReactJS
- O que é API ?
- O que é REST ?
- O que é uma RESTful ?
- Desenvolvimento de um backend em Node.js
- Desenvolvimento de uma aplicação frontend em ReactJs

# JavaScript

JavaScript é uma linguagem de programação que permite implementar dinamicidade em páginas web.



# JavaScript

- Variáveis
- Operações matemáticas
- Funções
- Condicionais
- Operadores lógicas
- Condição ternária
- Estrutura de repetição
- Requisições Ajax
- Promisse
- Axios

# ECMAScript - ES6

- O que é ES6?
  - Simplesmente a mais nova versão do javascript.
- Objetivos?
  - Ser uma linguagem melhor para construir aplicações complexas;
  - Resolver problemas antigos do JavaScript;
  - Facilidade no desenvolvimento de *libraries;*
- Como usar?
  - A grande maioria dos *browsers* ainda não dão suporte ao ES6;
  - Então usamos um *transpiler* como o BABEL

# ECMAScript - ES6

- Const e Let
- Operações em array
- Arrow Function
- Classes
- Operações Rest/Spread
- Desestruturação

# O que é Node.js ?

- É uma plataforma construída sobre o motor **JavaScript** do Google Chrome para facilmente construir aplicações de rede rápidas e escaláveis;
- Node.js usa um modelo de I/O direcionada a evento não bloqueante que o torna leve e eficiente;
- Roda em cima do motor V8;
- Utilizar JavaScript no backend;
- Comparável a PHP/ RUBY/ PYTHON/ GO;
- Plataforma.

# O que é Node.js ?

- Frameworks
  - ExpressJS:
    - Sem opinião;
    - Ótimo para iniciar;
    - Micro-serviços.
  - Frameworks opinados:
    - AdonisJS
    - NestJS

# O que é NPM ?

- Node Package Manager
  - É um repositório online para publicação de projetos de código aberto para o Node.js;
  - É  um utilitário de linha de comando que interage com este repositório online;
  - Instalação de pacotes;
  - Gerenciamento de versão;
  - Gerenciamento de dependências;
- Comparáveis:
  - Composer no PHP;
  - Gems no Ruby;
  - Pip no Python

# O que é REACT ?

- Biblioteca para construção de interface;
- Utilizado para construção de Single-Page-Applications;
- Podemos considerar um framework?
- Tudo fica dentro do JavaScript;
- React/ ReactJS/ React Native.

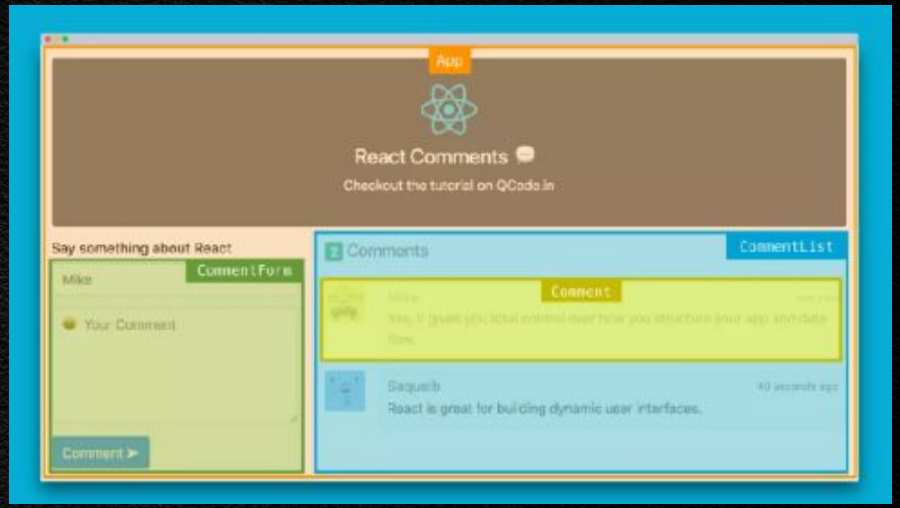
# O que é REACT ?

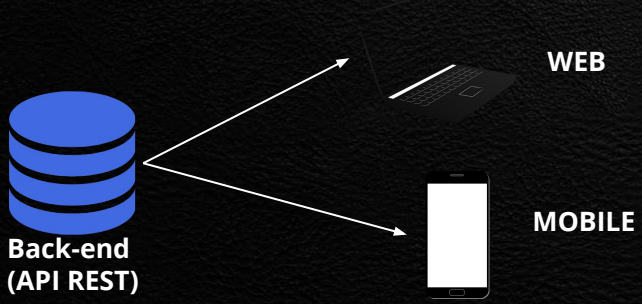
```
import React from "react";
import "./button.css";
import icon from "./button.png";

function Button() {
  return (
    <button>
      <img src={icon} />
    </button>
  );
}
```

# O que é REACT ?

- Vantagens:
  - Organização do código;
    - Componentização.
  - Divisão de responsabilidade;
    - Back-end: Regra de negócio;
    - Front-end: Interface.
  - Uma API e múltiplos clientes;
  - Programação declarativa;



# O que é REACT ?

- JSX

```
// Antes
function Button() {
  return React.createElement(
    "button",
    { type: button },
    React.createElement(
      "span",
      { class: "icon" })
  );
}
<button type="button">
  <span class="icon">!</span>
</button>;
```

```
// Com JSX
function Button() {
  return (
    <button type="button">
      <span class="icon"></span>
    </button>
  );
}
```

```
// Nossos próprios elementos
// (componentes)
function Header() {
  return <Button />;
}
```

# O que é REACT ?

- Imperativo vs Declarativo 


```
const notificacoes = 0;
function montaBadge(num) {
  if (notificacoes === 0 && num > 0) {
    // Adiciona badge
    // container.appendChild(badge)!!.
  }
  if (notificacoes !== 0 && num > 0) {
    // Apenas muda o número
    // badge.innerHTML = num!!.
  }
  if (notificacoes !== 0 && num === 0) {
    // Remove badge
    // container.removeChild(badge)
  }
}
```

```
// Não comparamos com o estado anterior
function Badge({ num }) {
  return (
    <div id="container">
      {num > 0 && <div id="badge">{num}</div>}
      <span class="icon"></span>
    </div>
  );
}
```

# O que é uma API?

- Cliente (Client) Garçom (pedidos, levar seus pedidos, para a cozinha) (API) Cozinha (Server).
- Acrônimo de Application Programming Interface (Interface de Programação de Aplicações) é basicamente um conjunto de rotinas e padrões estabelecidos por uma aplicação, para que outras aplicações possam utilizar as funcionalidades desta aplicação.
  - Responsável por estabelecer comunicação entre diferentes serviços.
  - Meio de campo entre as tecnologias.
  - Intermediador para troca de informações.

# O que é REST?

REpresentational State Transfer

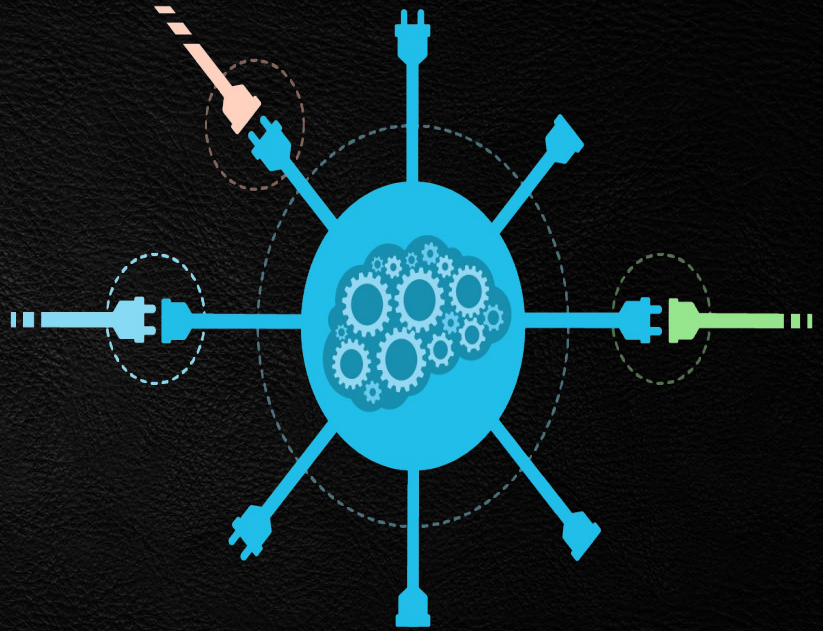
- Modelo arquitetural;
- Surgiu em 2000;

# Por que usar REST?

- Separação entre cliente e servidor;
- Escalabilidade;
- Independente de linguagem;
- Mercado

# Constraints

- Cliente-servidor;
- Stateless;
- Cache;
- Interface Uniforme;
- Sistema em Camadas;
- Código sob demanda



# REST / RESTful

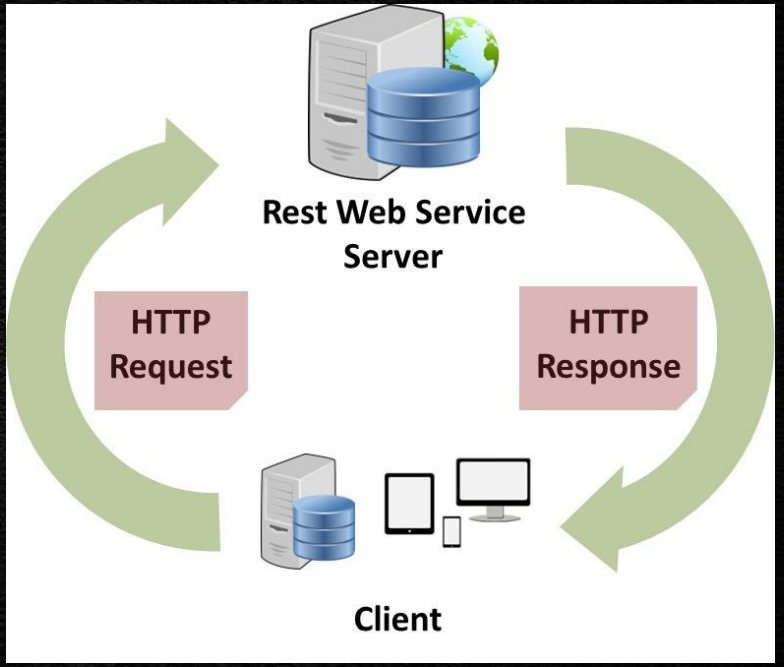
RESTful: Se trata de uma API que foi implementada usando o modelo REST.

REST: Um modelo arquitetural com características próprias.

OBS: O modelo REST não restringe ao uso de um protocolo específico, porém que possa ser implementado é necessário o uso de algum.

# Protocolo HTTP

Protocolo requisição-resposta



# Composição da requisição HTTP

```
[MÉTODO] [URI] HTTP/[Versão]
[Cabeçalhos]

[CORPO/PAYLOAD]
```

```
POST /produtos HTTP/1.1
Content-Type: application/json
Accept: application/json

{
   "nome": "Notebook i7",
   "preco": 2100.0
}
```

# Composição da resposta HTTP

```
HTTP/[Versão] [STATUS]
[Cabeçalhos]

[CORPO]
```

```
HTTP/1.1 201 Created
Location: /produtos/331
Content-Type: application/json

{
   "codigo": 331,
   "nome": "Notebook i7",
   "preco": 2100.0
}
```

# Códigos de status do HTTP

Os códigos de status do protocolo HTTP são divididos por níveis:

- Nível 200 (sucesso)
  - 200: Ok
  - 201: Criado
  - 204: Sem conteúdo


- Nível 300 (redirecionamento)
  - 301: Movido permanentemente
  - 302: Encontrado


- Nível 400 (Erro no cliente)
  - 400: Requisição mal feita
  - 403: Proibido
  - 404: Não encontrado


- Nível 500 (Erro no servidor)
  - 500: Erro interno no servidor
  - 501: Não implementado
  - 404: Não encontrado

# Códigos de status do HTTP

Maneira incorreta de usar os status HTTP:

**REQUISIÇÃO**

```
POST /produtos HTTP/1.1

{
  "preco": 2100
}
```

**RESPOSTA**

```
HTTP/1.1 200 OK

{
  "erro": "Nome não informado"
}
```

# Códigos de status do HTTP

**REQUISIÇÃO**

```
POST /produtos HTTP/1.1

{
  "preco": 2100
}
```

**RESPOSTA**

```
HTTP/1.1 400 Bad Request

{
  "erro": "Nome não informado"
}
```

# Recursos no REST

Algo que tenha relevância e possa ser referenciado no software:

- Documento;
- Linha de uma tabela no BD;
- Resultado de uma consulta;
- Pode ser abstrato ou concreto;
- Pode ser uma coleção ou um objeto único.

# Recursos no REST


# URI

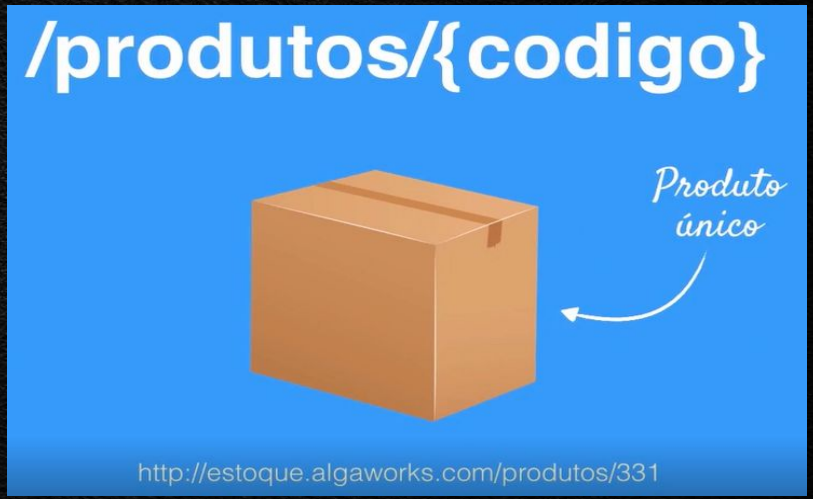
Uniform Resource Identifier

- Um recurso no REST precisa ser identificado de alguma maneira. O REST usa URI para identificar recursos.
- É um conjunto de caracteres que busca identificar recursos de uma forma não ambígua.
- Uma URL é um tipo de URI;
- Uma URI deve ser um substantivo que identifique de forma clara  o recurso

# URI

http://estoque.com/produtos


http://estoque.com/produtos/331



# URI

- Um recurso não deve ser modelado como um verbo

**REQUISIÇÃO**

```
POST /salvarProduto HTTP/1.1

{
  "nome": "Notebook i7",
  "preco": 2100
}
```

**RESPOSTA**

```
HTTP/1.1 201 Created

...
```

# URI

**REQUISIÇÃO**

```
POST /produtos HTTP/1.1

{
  "nome": "Notebook i7",
  "preco": 2100
}
```

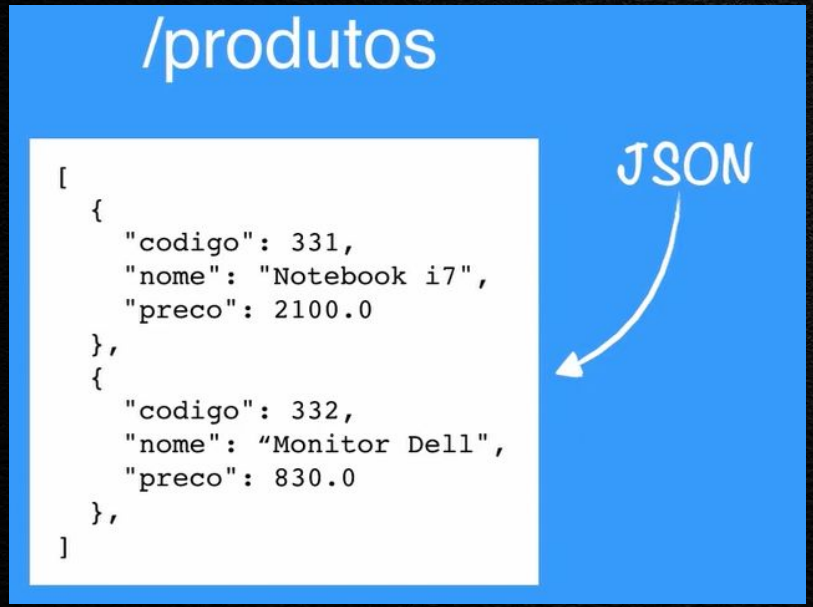
**RESPOSTA**

```
HTTP/1.1 201 Created

...
```

# Representação

Código que descreve o estado atual do recurso.




Tipos de representação:

- XML
- JPG
- JSON

# Métodos HTTP

Semântica de operações possíveis de serem executadas por um determinado recurso.

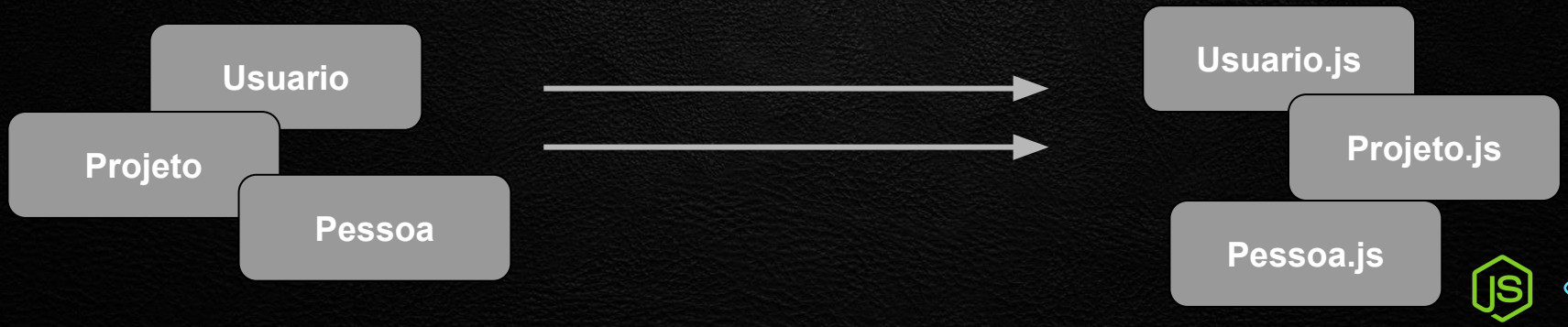
- GET: Busca recursos;
- POST: Cria um novo recurso;
- PUT: Atualiza um recurso existente;
- DELETE: Remove um recurso.

# FINAL PRIMEIRO DIA

# SEQUELIZE

# Sequelize

- ORM
  - ORM (Object Relational Mapper) é uma técnica de mapeamento objeto relacional que permite fazer uma relação dos objetos com os dados que os mesmos representam.
  - Abstração do banco de dados;
  - Tabelas viram models;



# Sequelize

- Manipulação de dados
  - Sem SQL (geralmente);
  - Apenas código JavaScript;

```sql
INSERT INTO users (name, email)
 VALUES ( "Moisés Costa",
 "moises@flamengo.com.br"
 )
```

```
User.create({
      name:”Moisés Costa”,
      email:”moises@flamengo.com.br”
})
```

```sql
SELECT * FROM users WHERE email =
“moises@flamengo.com.br” LIMIT 1
```

```
 User.findOne({
     where:{
       email:”moises@flamengo.com.br”
     }
 })
```

# Sequelize

- Migrations
  - Controle de versão para base de dados;
  - Cada arquivo contém instruções para criação, alteração ou remoção de tabelas ou colunas;
  - Mantém a base atualizada entre todos desenvolvedores do time e também no ambiente de produção;
  - Cada arquivo é uma migration e sua ordenação ocorre por data.

```
module.exports = {
  up: (queryInterface, Sequelize) => {
    return queryInterface.createTable('users', {
      id: {
        type: Sequelize.INTEGER,
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
      },
      name: {
        type: Sequelize.STRING,
        allowNull: false,
      },
      email: {
        type: Sequelize.STRING,
        allowNull: false,
        unique: true,
      },
    });
  },

  down: queryInterface => {
    return queryInterface.dropTable('users');
  },
};
```

Instrução para criar uma nova tabela

Criação de 3 campos com suas propriedades.

Instrução para deletar a tabela.

# Arquitetura MVC

**Model**

O model armazena a abstração do banco, utilizado para manipular os dados contidos nas tabelas do banco. Não possuem responsabilidade sobre a regra de negócio da nossa aplicação.

**Controller**

O controller é o ponto de entrada das requisições da nossa aplicação, uma rota geralmente está associada diretamente com um método do controller. Podemos incluir a grande parte das regras de negócio da aplicação nos controllers (conforme a aplicação cresce podemos isolar as regras).

**Visão**

A view é o retorno ao cliente, em aplicações que não utilizando o modelo de API REST isso pode ser um HTML, mas no nosso caso a view é apenas nosso JSON que será retornado ao front-end e depois manipulado pelo ReactJS ou React Native

# A face de um controller

- Classes;
- Sempre retorna um JSON;
- Não chama outro controller/método;
- Quando criar um novo controller:
  - Apenas  5 métodos;
  - Estou falando da mesma entidade ?

```
class UserController {
    index() {}
    show() {}
    store() {}
    update() {}
    delete() {}
}
```

# JWT

- Autenticação JWT
  - POST http://api.com/sessions

```
{
  "email": "diego@rocketseat.com.br",
  "password": "123456"
}
```

TOKEN JWT

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

# Referências


- http://nodebr.com/o-que-e-node-js/
- http://nodebr.com/o-que-e-a-npm-do-nodejs/
- https://github.com/Rocketseat/youtube-api-rest-restful