Alexandre W. B.
Francisco

TYPESCRIPT

O QUE É?

TYPESCRIPT

TypeScript é uma linguagem de programação de código aberto criada pela Microsoft, que expande o JavaScript ao adicionar um sistema de tipos estático. Esta tipagem permite detectar erros em tempo de compilação, ajudando na manutenção e na redução de bugs em projetos maiores.

INSTALAÇÃO

npm init -y

```
alexa@LAPTOP-ALE MINGW64 ~/Documents/2- Estudos/CC/4ª Fase/Desenvolvimento Web I
/Agosto/Configuração de ambiente/seminario_TypeScript (main)
$ npm init -y
Wrote to C:\Users\alexa\Documents\2- Estudos\CC\4ª Fase\Desenvolvimento Web I\Ag
osto\Configuração de ambiente\seminario_TypeScript\package.json:

{
  "name": "seminario_typescript",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

INSTALAÇÃO

npm i typescript -D

```
alexa@LAPTOP-ALE MINGW64 ~/Documents/2- Estudos/CC/4ª Fase/Desenvolvimento Web I
/Agosto/Configuração de ambiente/seminario_TypeScript (main)
$ npm i typescript

added 1 package, and audited 2 packages in 11s

found 0 vulnerabilities
```

INSTALAÇÃO

npm i typescript -D

```
alexa@LAPTOP-ALE MINGW64 ~/Documents/2- Estudos/CC/4ª Fase/Desenvolvimento Web I
/Agosto/Configuração de ambiente/seminario_TypeScript (main)
$ npx tsc --init

Created a new tsconfig.json with:

  target: es2016
  module: commonjs
  strict: true
  esModuleInterop: true
  skipLibCheck: true
  forceConsistentCasingInFileNames: true


You can learn more at https://aka.ms/tsconfig
```

TSCONFIG.JSON

O “tsconfig.json” é um arquivo de configuração usado para definir como o compilador TypeScript deve compilar o código. Ele especifica opções como a versão do ECMAScript de saída, diretórios de entrada e saída, e regras de tipagem, permitindo controle detalhado sobre o processo de compilação e facilitando a automação e integração contínua em projetos TypeScript.

RECURSOS TYPESCRPIT

TIPOS DE DADOS

GENERICS

ENUMS

INTERFACES

Esses elementos são fundamentais para construir aplicações mais seguras e escaláveis, pois oferecem tipagem estática e um controle mais rigoroso sobre a estrutura do código e os dados utilizados.

TIPOS DE DADOS

01 **Tipos primitivos**
Boolean, Number, String, Null...

02 **Objetos**
Objeto, Class, Interfaces, Enum...

03 **Principais**
Unknown, any

04 **Tipos de fundo**
Never

ENUM

permite que um desenvolvedor defina um conjunto de constantes nomeadas. Usar enums pode facilitar a documentação da intenção ou criar um conjunto de casos distintos. O TypeScript fornece enums numéricos e baseados em strings.

```
enum Teclas {
  Cima = 'ArrowUp',
  Baixo = 'ArrowDown',
  Esquerda = 'ArrowLeft',
  Direita = 'ArrowRight',
}

window.addEventListener('keydown', function (e) {
    if(e.key === Teclas.Cima){
      console.log('pressionou a tecla arrow up');
    }
});
```

GENERICS

Genéricos em TypeScript são uma maneira de escrever código que pode trabalhar com vários tipos de dados, em vez de ser limitado a um único tipo de dados.

Genéricos permitem que você escreva funções, classes e interfaces que recebem um ou mais parâmetros de tipo, que agem como marcadores de posição para os tipos de dados reais que serão usados quando a função, classe ou interface for usada.

```
function identity<T>(arg: T): T {
  return arg;
}

let output = identity<string>('Hello');
```

INTERFACES

Interfaces em TypeScript fornecem uma maneira de definir um contrato para um tipo, que inclui um conjunto de propriedades, métodos e eventos. É usado para impor uma estrutura para um objeto, classe ou argumento de função.

```typescript
interface User {
  name: string;
  age: number;
}

const user: User = {
  name: 'John Doe',
  age: 30,
};
```

DECORATOR

Decoradores são um recurso do TypeScript que permite modificar o comportamento de uma classe, propriedade, método ou parâmetro. Eles são uma maneira de adicionar funcionalidade adicional ao código existente e podem ser usados para uma ampla gama de tarefas, incluindo registro, otimização de desempenho e validação.

CLASSES

Classes em TypeScript são um modelo para criar objetos (instâncias de uma classe), fornecendo uma maneira de estruturar objetos e encapsular dados e comportamento.

Sintaxe similar ao Java e C# por exemplo

HERANÇA

Herança se refere a um mecanismo onde uma subclasse herda propriedades e métodos de sua classe pai. Isso permite que uma subclasse reutilize o código e o comportamento de sua classe pai enquanto também adiciona ou modifica seu próprio comportamento.

MODIFICADORES DE ACESSO

No TypeScript, modificadores de acesso são palavras-chave usadas para controlar a visibilidade e acessibilidade de propriedades e métodos de classe.

Public
Pivate
Protected

MODULOS

01 No TypeScript, módulos são usados para organizar e reutilizar código.

02 Módulos internos são usados para organizar código dentro de um arquivo

03 Módulos externos são usados para organizar código em vários arquivos.

COMPILAÇÃO

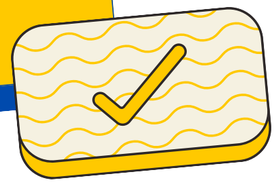
npm tsc

```
alexa@LAPTOP-ALE MINGW64 ~/Documents/2- Estudos/CC/4ª Fase/Desenvolvimento Web I
/Agosto/Configuração de ambiente/seminario_TypeScript (main)
$ npx tsc
```

.JSON

O .JSON é um formato de arquivo usado para representar e armazenar dados estruturados de maneira simples e legível, sendo basicamente um texto que segue a sintaxe dos objetos em JavaScript e é usado para transmitir dados em aplicações web.

.JSON



01 Segurança

02 Simplicidade

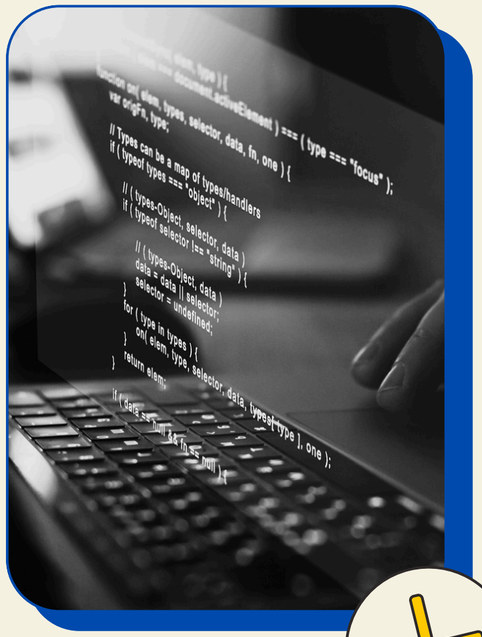
03 Consistencia

04 Suporte Nativo

05 Versatilidade

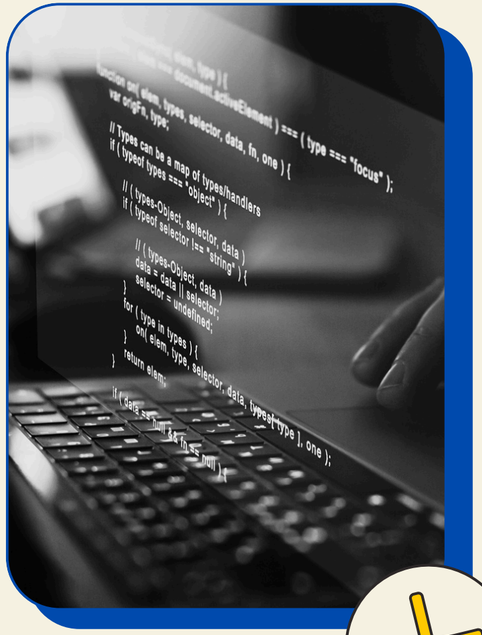
06 Facil Serialização /Deserialização

EXEMPLOS



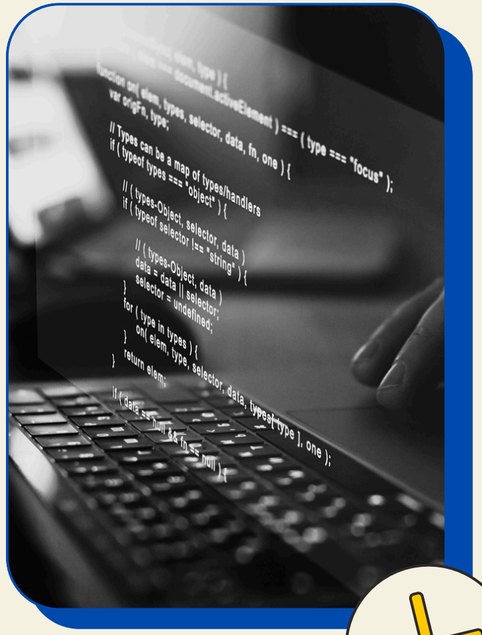
```
{
  "apiUrl": "https://api.example.com",
  "port": 8080
}
```

EXEMPLOS



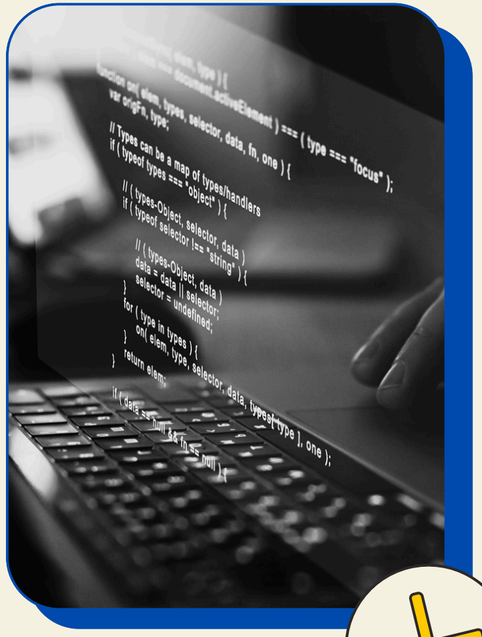
```
// Definindo a interface para o JSON
interface Config {
  apiUrl: string;
  port: number;
}
```

EXEMPLOS

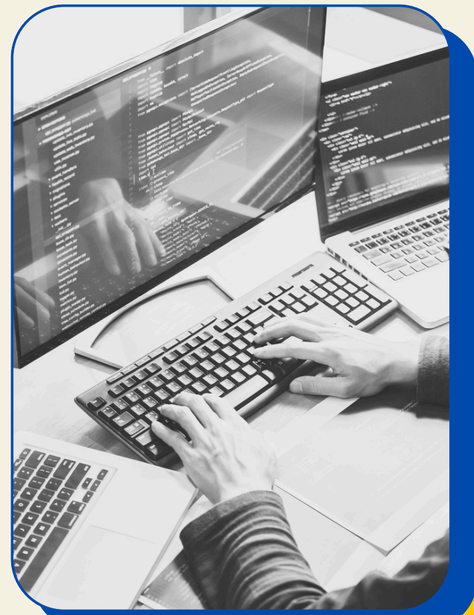


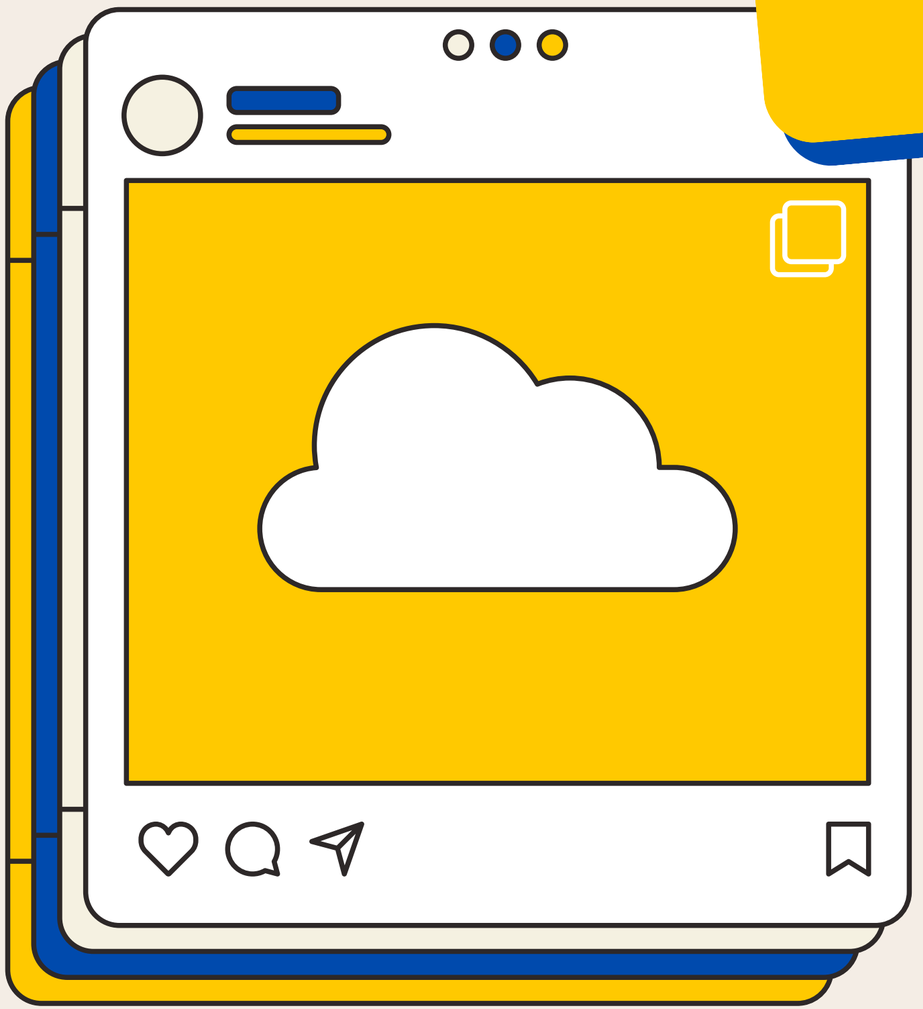
```
// Importando o arquivo JSON
import config from './config.json';
```

EXEMPLOS



```
// Usando os dados do JSON
const apiUrl: string = config.apiUrl; const port:
number = config.port; console.log(`API URL:
${apiUrl}, Port: ${port}`);
```





.JSON

01 **Ausência de Tipagem Intrínseca**

02 **Overhead de Parsing**

03 **Limitações de Estrutura**

.JSON

JSON é um formato simples e amplamente usado para troca de dados, especialmente em APIs, com a vantagem de ser fácil de ler e integrar com TypeScript através de interfaces que garantem tipagem e segurança.

.JSON

No entanto, JSON carece de tipagem intrínseca, pode ter impacto de performance ao ser manipulado em grande escala, e não suporta estruturas complexas ou métodos.

LOG IN