



12/8/2019

# Treinamento Angular 8

COTI INFORMÁTICA



[WWW.COTIINFORMATICA.COM.BR](http://WWW.COTIINFORMATICA.COM.BR)

AV RIO BRANCO, 185 – SALA 308 – CENTRO – RIO DE JANEIRO - RJ

## Sumário

Programas para instalar: .....	4
Para verificar a versão do Angular. ....	6
Documentação oficial do Angular para estudar. ....	7
O que é Angular? .....	8
Diferenças entre Angular e AngularJS .....	9
História do Angular.....	10
A versão 2.0.....	10
Versão 4.0.....	10
Versão 5.0.....	11
Versão 6.0.....	11
Versão 7.0.....	11
Versão 7.1.....	12
O que é Typescript?.....	13
<b>TypeScript é um superconjunto de JavaScript</b> .....	13
Orientado a Objeto .....	14
Erro de tempo de compilação .....	15
Fundamentos do TypeScript .....	15
Módulo .....	15
Interface .....	16
Classes .....	16
Funções .....	16
Variáveis .....	16
Tipos de TypeScript .....	16
Classes e Interface.....	17
Classes .....	17
Interfaces.....	19
Herança .....	20
TypeScript: operadores aritméticos .....	22
Adição (+) .....	22
Subtração (-).....	23
Multiplicação (*).....	23
Divisão (/) .....	23
Módulo (%).....	24

Exponenciação (**) .....	24
Incremento (++).....	24
Decremento (- -).....	24
TypeScript: Operadores lógicos .....	25
&& (and).....	25
(or) .....	26
! (not).....	26
O que é Programação Funcional? .....	27
O que é Node?.....	28
O que é Express? .....	29
Classe e Objeto.....	29
Encapsulamento.....	30
Associação de Classes .....	30
Herança .....	31
Interface .....	32
Desenvolvimento de Software .....	33
Arquitetura MVC .....	34
Arrays, Vetor e Matriz .....	36
Classificação dos arrays.....	36
<b>Declaração de vetores</b> .....	37
Declaração de Matrizes.....	38
<b>Acesso aos elementos do vetor</b> .....	38
Exemplos com Vetores.....	38
O que é o operador ternário? .....	38
CRIANDO UM NOVO PROJETO: .....	41
Para criar o diretório: .....	43
Para criar a classe.....	43
Para criar um componente.....	44
PROJETO ESTRUTURADO.....	45
PROJETO COM INTERFACE, CALCULO FUNCIONAL E HERANÇA .....	52
NOVO PROJETO MDB .....	61
Módulo Programação Funcional .....	73
Exercícios.....	77

Professores:

**Edson Belém**

profedsonbelem@gmail.com

Cel.: 98199-0108

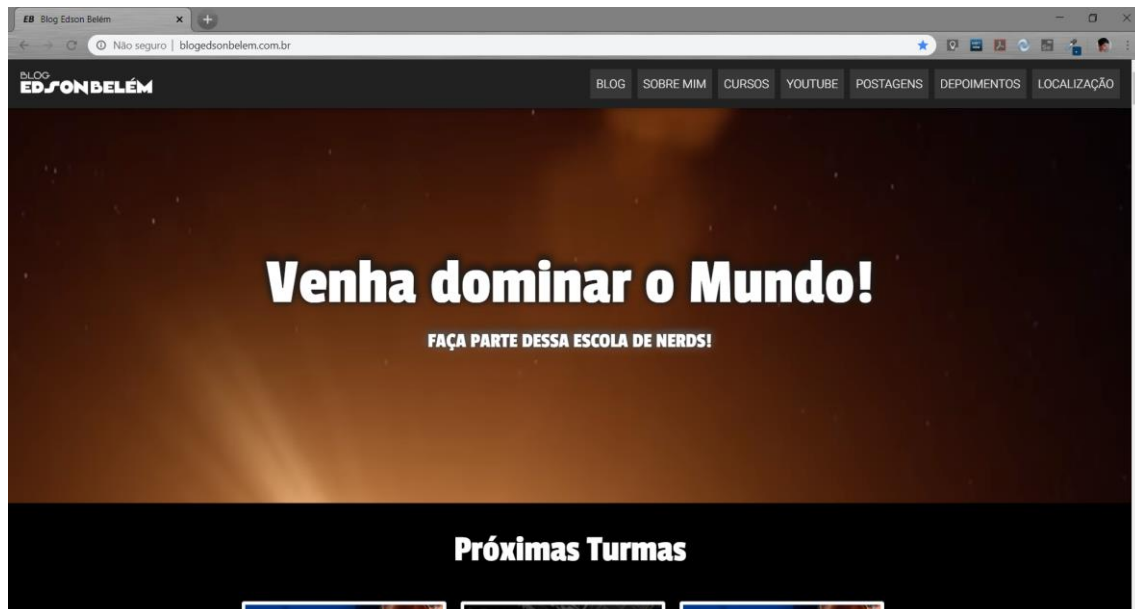
**Luciana Medeiros**

lucianamedeiros.coti@gmail.com

Cel.: 98201-2525

Visitem nosso blog.

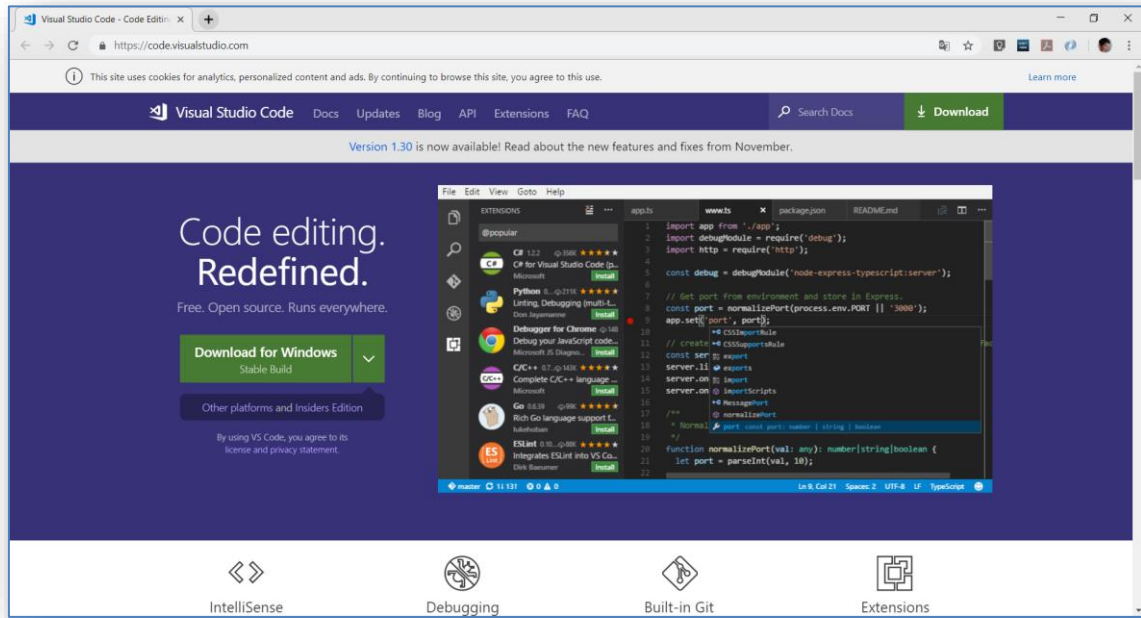
[www.blogedsonbelem.com.br](http://www.blogedsonbelem.com.br)



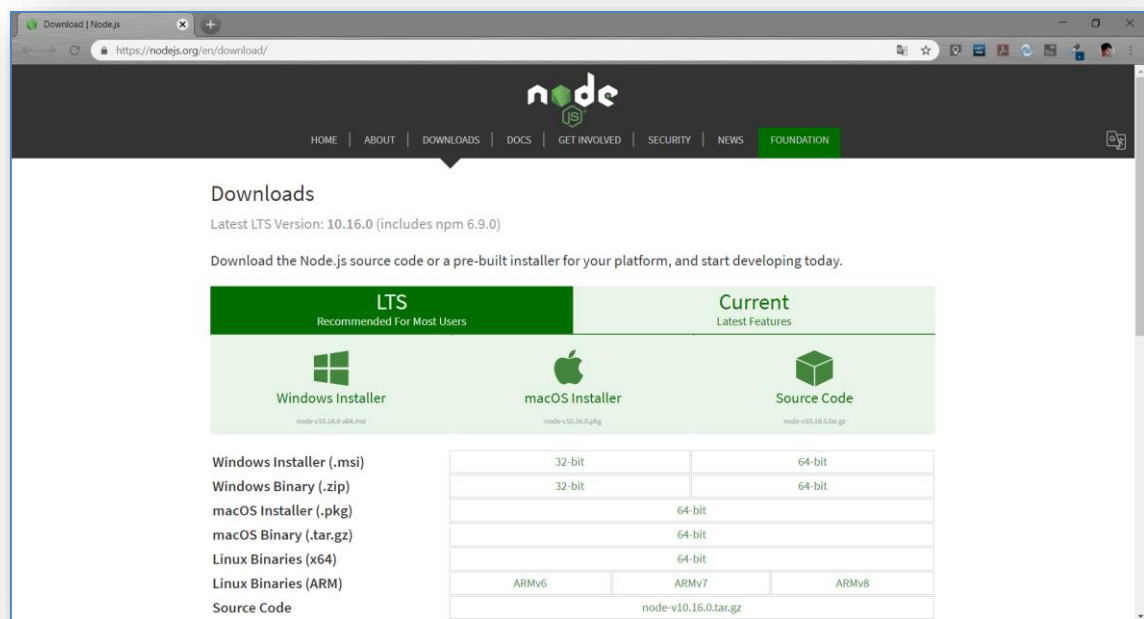
## Programas para instalar:

- Visual Studio Code ;
- Node.js;
- Angular cli;

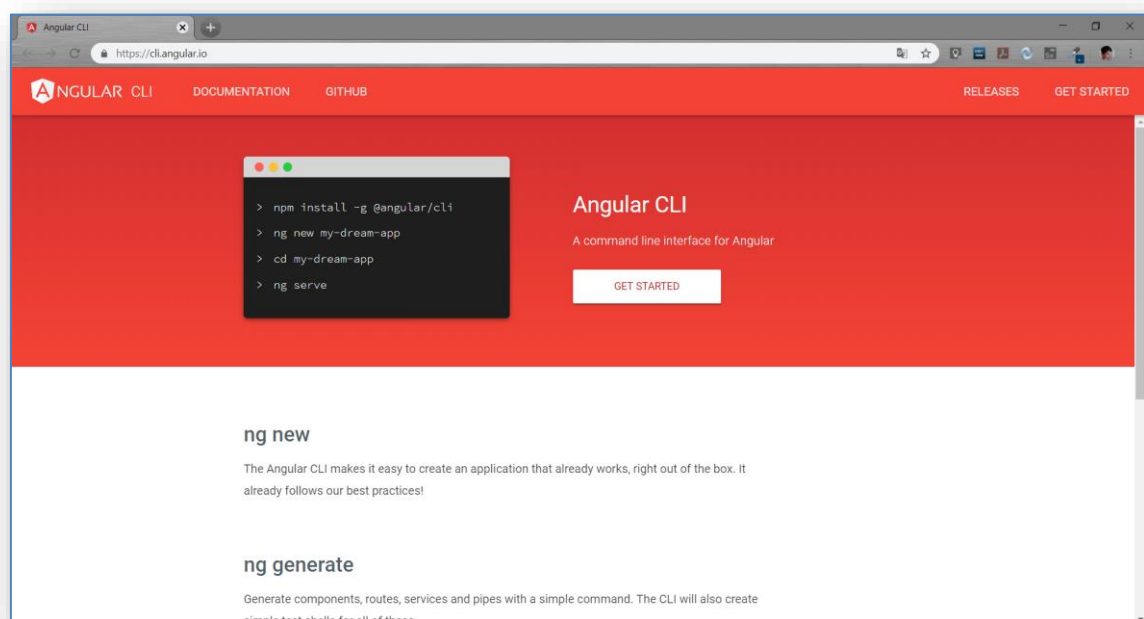
<https://code.visualstudio.com/>



<https://nodejs.org/en/download/>



<https://cli.angular.io/>



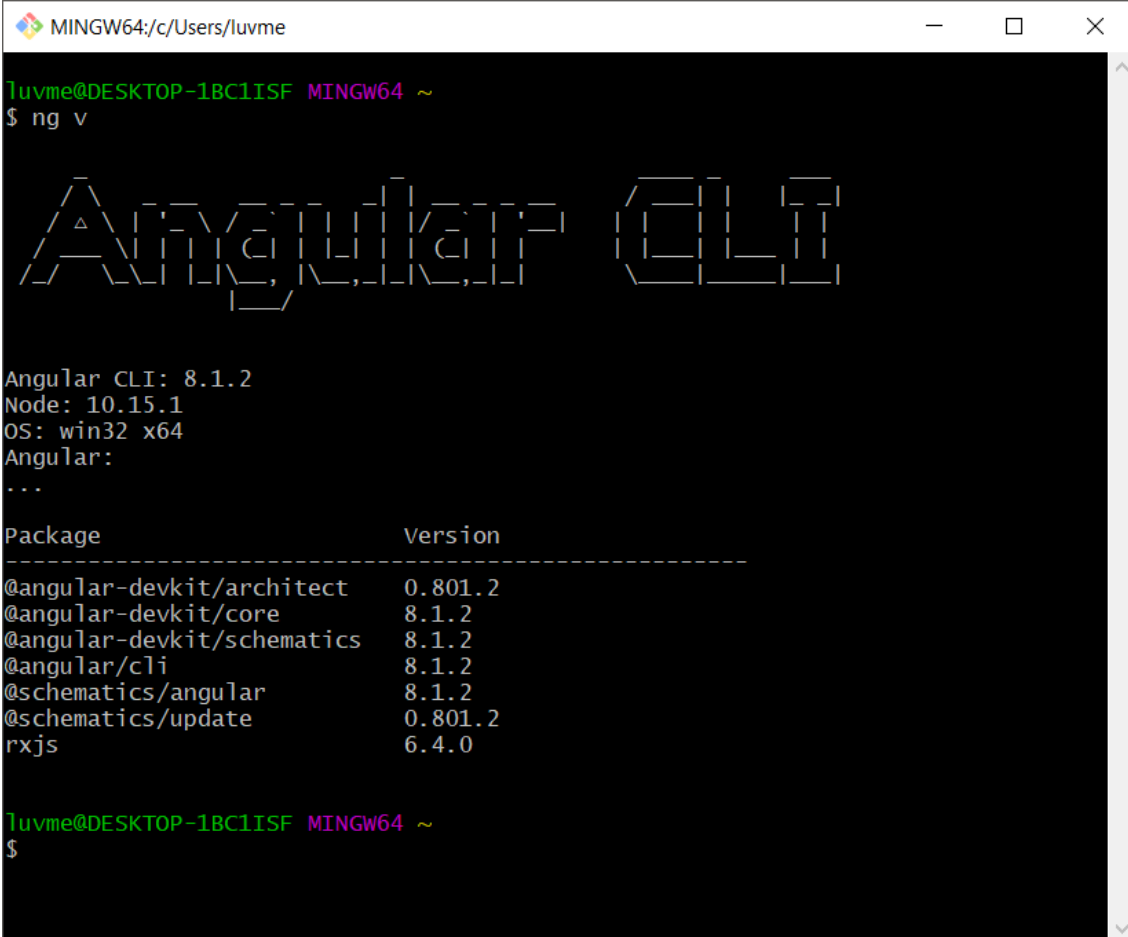
Para instalar o angular cli:

Abrir o terminal (cmd) e digitar:

**`"npm install -g @angular/cli"`** ou

**`"npm install -g @angular/cli@8.1.2"`** (para escolher a versão que queremos instalar colocamos o "@" e o numero da versão desejada.

As versões que estamos usando no curso:



```
MINGW64:/c/Users/luvme
luvme@DESKTOP-1BC1ISF MINGW64 ~
$ ng v

Angular CLI: 8.1.2
Node: 10.15.1
OS: win32 x64
Angular:
...

Package                                  Version
-----
@angular-devkit/architect                0.801.2
@angular-devkit/core                     8.1.2
@angular-devkit/schematics               8.1.2
@angular/cli                             8.1.2
@schematics/angular                     8.1.2
@schematics/update                       0.801.2
rxjs                                      6.4.0

luvme@DESKTOP-1BC1ISF MINGW64 ~
$
```

Para verificar a versão do Angular.

Digitar no terminal:

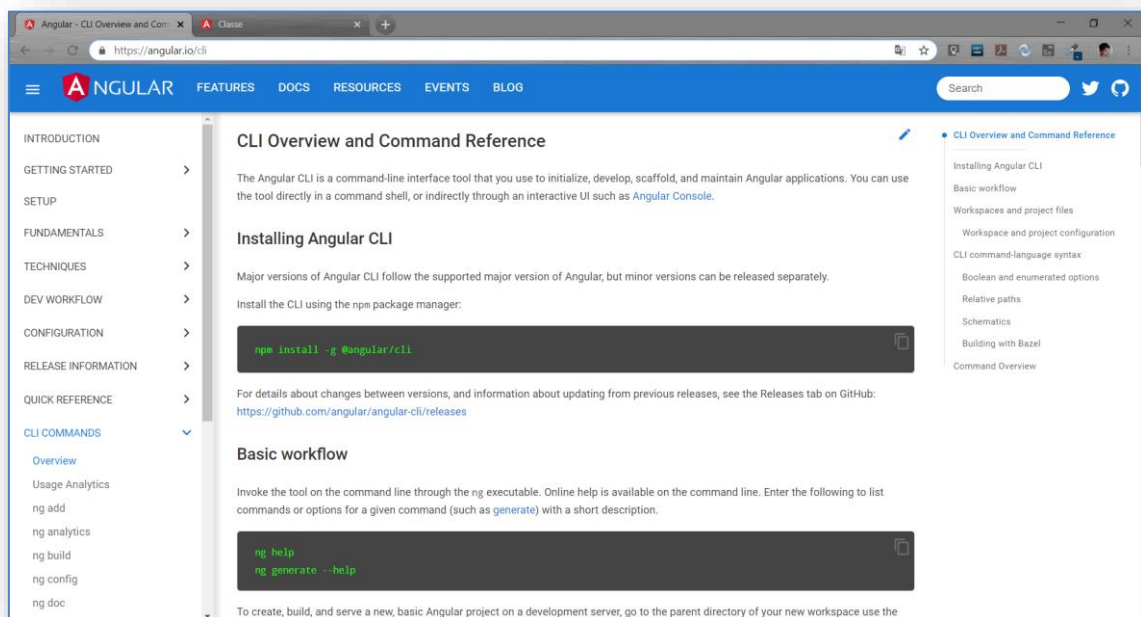
**`"ng v"`**

<https://www.typescriptlang.org/>



Documentação oficial do Angular para estudar.

<https://angular.io/cli>



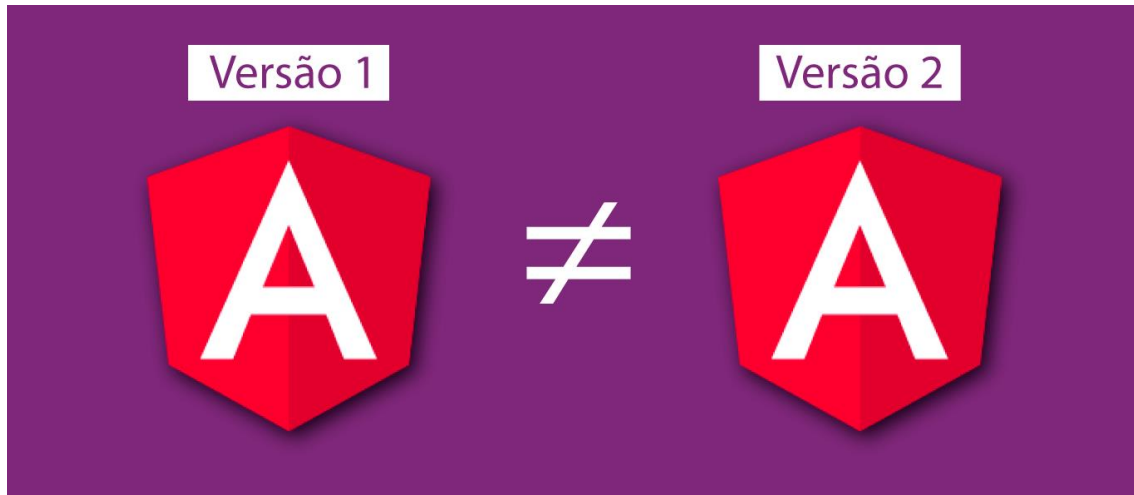


## O que é Angular?



Angular (comumente referido como "Angular 2+" ou "Angular 2") é uma plataforma de aplicações web de código-fonte aberto e front-end baseado em TypeScript liderado pela Equipe Angular do Google e por uma comunidade de indivíduos e corporações. Angular é uma reescrita completa do AngularJS, feito pela mesma equipe que o construiu.

## Diferenças entre Angular e AngularJS



Arquitetura de uma aplicação Angular. Os principais blocos são módulos, componentes, templates, metadados, enlace de dados, directivas, serviços e injeção de dependência. Angular foi uma base de reescrita do AngularJS.

- Angular não tem um conceito de "escopo" ou controladores, em vez disso, ele usa uma hierarquia de componentes como o seu principal conceito arquitetônico.
- Angular tem uma expressão diferente de sintaxe, concentrando-se no uso de "[ ]" para a propriedade de ligação, e no uso de "( )" para ligação do evento
- Modularidade – muito das funcionalidades principais foram movidas para os módulos
- Angular recomenda o uso da linguagem da Microsoft, o TypeScript, que apresenta as seguintes características:
  - É baseado em classes de Programação Orientada a Objeto
  - Tipagem Estática
  - Programação genérica
- O TypeScript é um superconjunto do ECMAScript 6 (ES6), e é compatível com ECMAScript 5 (i.e.: JavaScript). Angular também inclui ES6:
  - Lambdas
  - Iteradores
  - For/Of loops
- Carregamento dinâmico
- Modelo de compilação assíncrono
- A substituição de controladores e \$escopo com componentes e diretrizes – um componente é uma directiva com um modelo
- Programação reativa de suporte usando RxJS

## História do Angular



Originalmente, a reescrita do AngularJS foi chamado de "Angular 2" pela equipe, mas isto levou à confusão entre os desenvolvedores. Para esclarecer, a equipe anunciou que termos separados devem ser usados para cada Framework, com "AngularJS", referindo-se a 1.X versões e "Angular" sem o "JS", referindo-se às versões à partir da 2 até a última.

### A versão 2.0

O Angular 2.0 foi anunciado no ng-Europe conference 22-23 de setembro de 2014. As mudanças drásticas na versão 2.0 criou uma considerável controvérsia entre os desenvolvedores. Em 30 de abril de 2015, os desenvolvedores Angular anunciaram que o Angular 2 foi transformado de Alfa a Developer Preview. O Angular 2 mudou-se para o Beta em dezembro de 2015, e a primeira versão foi publicada em Maio de 2016. A versão final foi lançada em 14 de setembro de 2016.

### Versão 4.0

Em 13 de dezembro de 2016 Angular 4 foi anunciado, ignorando o 3 para evitar uma confusão devido ao desalinhamento da versão do pacote do roteador que já foi distribuído como v3.3.0. A versão final foi lançada em 23 de Março de 2017. O Angular 4 é compatível com o Angular 2.

O Angular versão 4.3 é uma versão menor, o que significa que ele contém alterações que não são de última hora e que é uma atualização pequena para 4.x.x.

### Recursos na versão 4.3

Introdução de HttpClient, uma biblioteca menor, mais poderosa, e mais fácil de usar, para fazer Solicitações HTTP.

Novo roteador ciclo de vida de eventos para Guardas e Resoluções. Quatro novos eventos: GuardsCheckStart, GuardsCheckEnd, ResolveStart, ResolveEnd juntam-se ao conjunto existente de ciclo de vida, tais como a NavigationStart. Condicionalmente desativa animações.

### Versão 5.0

Angular 5 foi lançado em 1 de novembro de 2017. Os principais aprimoramentos Angular 5 incluem suporte para web apps progressivos, uma compilação otimizadora e melhorias relacionadas ao Material Design.

### Versão 6.0

O lançamento da sexta versão do Angular ocorreu no dia quatro de Maio de 2018. Esta versão teve um foco menor na base do framework, e maior na cadeia de ferramentas e em como tornar mais fácil e rápida a migração com Angular nas atualizações futuras, como os comandos da CLI (Interface de Linha de Comando): ng update, ng add, Angular Elements, Componentes Angular Material + CDK, Componentes Iniciais Angular Material, CLI Workspaces, Suporte a biblioteca, Provedores de Árvore Shakable, Melhoramentos no desempenho de animações, e atualização do RxJS para a versão 6.

### Versão 7.0

A sétima versão do Angular foi lançada no dia dezoito de Outubro de 2018. Atualizações relacionadas ao desempenho de aplicativos, Angular Material & CDK, Rolagem Virtual, Melhor acessibilidade do elemento de formulário select (também conhecido por combobox ou dropdownlist), agora suporta Content Projection usando o padrão web para elementos personalizados, e atualizações de dependência em relação ao Typescript 3.1, RxJS 6.3, Node 10 (ainda suportando Node 8).

### Versão 7.1

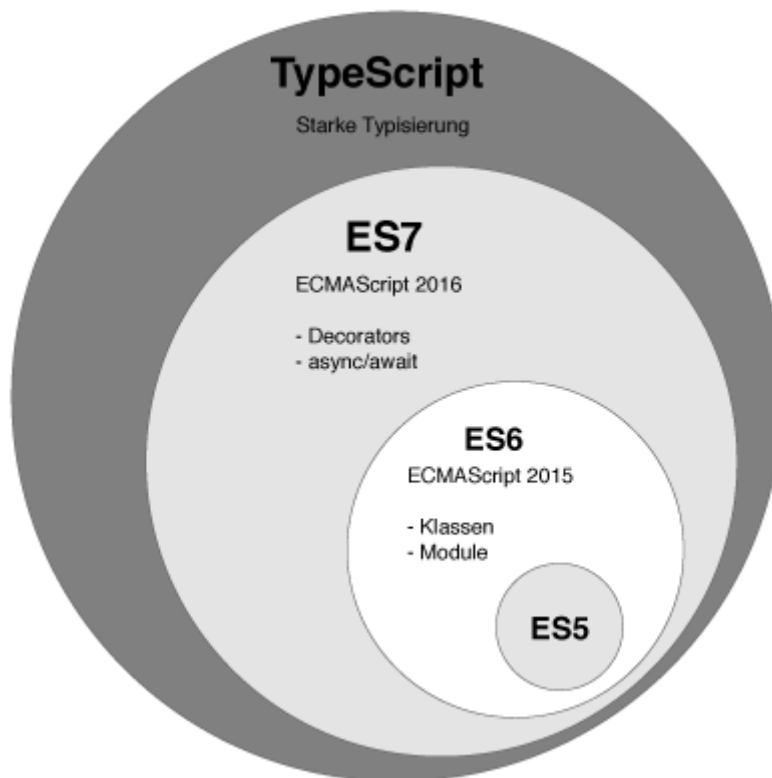
O lançamento do Angular 7 ocorreu em novembro de 2018 e está disponível

### Versão 8.0

O lançamento da mais nova versão do Angular ocorreu em maio de 2019 e conta com algumas melhorias no funcionamento dos formulários reativos, melhoria na sintaxe para declaração de rotas lazy-loading e a esperada disponibilização do compilador Ivy, que nesta versão ainda está em preview e precisa ser habilitado, mas não é recomendado que se utilize em produção.

Cada versão está prevista para ser compatível com a versão anterior. O Google se comprometeu a fazer atualizações duas vezes por ano.

## O que é Typescript?



TypeScript é um superconjunto de JavaScript desenvolvido pela Microsoft que adiciona tipagem e alguns outros recursos a linguagem. Anders Hejlsberg, arquiteto da linguagem C# e criador das linguagens Delphi e Turbo Pascal, trabalhou no desenvolvimento do TypeScript. A linguagem pode ser usada para desenvolver aplicações JavaScript no lado cliente e lado servidor (Node.js).

Foi considerada pelo público a 4ª linguagem "mais amada", de acordo com uma pesquisa conduzida pelo *site* Stack Overflow em 2018, e está entre as 15 linguagens mais populares, de acordo com uma pesquisa conduzida pela RedMonk

### TypeScript é um superconjunto de JavaScript

Isso significa que o TypeScript estende JavaScript com funcionalidade extra que não está presente na versão atual do JavaScript suportada pela maioria dos navegadores. Então, quais são esses recursos?

Se você estiver familiarizado com linguagens como Java ou C #, provavelmente sabe que cada variável requer um tipo e esse tipo deve ser declarado antecipadamente . JavaScript é uma linguagem fracamente tipada ou digitada dinamicamente. As variáveis em JavaScript não estão diretamente associadas a nenhum tipo de valor específico, e qualquer variável pode ser atribuída (e reatribuída) a todos os tipos de valores:

```
var foo = 42;    // foo is now a Number
var foo = 'bar'; // foo is now a String
var foo = true;  // foo is now a Boolean
```

Em uma linguagem fracamente tipada, o tipo de um valor depende de como ele é usado . Por exemplo, se eu puder passar uma string para o operador de adição e ela será automaticamente interpretada como um número ou causará um erro se o conteúdo da string não puder ser traduzido em um número . Da mesma forma, posso concatenar cadeias e números ou usar strings como booleanos, etc.

Em uma linguagem fortemente tipada, uma variável tem um tipo e esse tipo não pode mudar . O que você pode fazer com uma variável depende do seu tipo. Se tentarmos compilar o código acima no TypeScript, obteremos o erro

```
[ts] Subsequent variable declarations must have the same type.
Variable 'foo' must be of type 'number', but here has type
'boolean'.
```

A vantagem de uma linguagem fortemente tipada é que você é forçado a tornar explícito o comportamento do seu programa . Se você quiser adicionar um número e uma string ao seu código, você deve traduzir a string em um número a ser usado como um operando do operador de adição. Isso torna o código mais fácil de entender porque não há comportamento (ou menos) oculto . Isso também torna seu código mais detalhado.

### **Lembrar:**

A tipificação forte é opcional no TypeScript, mas o uso desse recurso torna seu aplicativo mais previsível e facilita a depuração, portanto, você deve definitivamente usá-lo.

### **Orientado a Objeto**

A programação orientada a objetos (OOP) é um conceito antigo usado por vários idiomas, como Java ou C #, para ajudar o desenvolvedor a associar dados e métodos em um "objeto". Em **OOP** objeto são criados e são capazes de interagir uns com os outros usando os métodos de enfrentamento público um do outro. JavaScript em si não é uma linguagem orientada a objetos na maneira que C ++, C # ou Java são. O TypeScript, por outro lado, pode ser tratado como uma

linguagem orientada a objetos por causa das construções de linguagem introduzidas sobre os fechamentos de JavaScript.

O TypeScript traz muitos recursos orientados a objetos que perdemos no JavaScript por um longo tempo. Temos conceitos explícitos de:

- Classes
- Interfaces
- Construtores
- Modificadores de acesso (públicos e privados)

### Erro de tempo de compilação

Deixe-me explicar primeiro o que **run-time error** é como este é um caso comum. Estamos falando sobre **run-time error** quando enfrentamos qualquer problema ao usar nosso aplicativo. Em outras palavras - se você cometer um erro ao criar seu site e escrever algum código, você o verá apenas enquanto estiver usando o aplicativo / site. Por exemplo, se você cometer um erro de digitação em um arquivo de página da Web, o console do seu navegador mostrará um erro quando você carregar essa página.

Devido à fraca tipagem em JavaScript, muitas vezes acontece que tentamos realizar operações em uma variável com um tipo diferente. Se a nossa lógica de aplicativo for complexa, talvez não percebamos que estamos tentando atribuir elementos de um tipo diferente uns aos outros. Em JavaScript, só saberemos sobre o erro quando alguém acionar um código e ele falhar. Por outro lado, o TypeScript pode nos fornecer erros de tempo de compilação, o que significa que ele pode detectar erros ao compilar código para JavaScript e corrigi-lo antes de implantar na produção. Claro que não vai pegá-los todos, mas ainda muito.

### Fundamentos do TypeScript

O TypeScript fornece aos desenvolvedores conceitos orientados a objeto e compila a verificação do tipo de tempo em cima do JavaScript, o que ajuda a escrever código mais estruturado, sustentável e robusto. O TypeScript introduz alguns dos termos orientados a objetos padrão, como Classes, Interfaces, Module e Variables, que no final são convertidos em várias formas diferentes de JavaScript. A estrutura de código para um arquivo TypeScript típico é mostrada abaixo.

### Módulo

O módulo é como um namespace no mundo .NET e pode conter classes e interfaces. Os módulos não possuem nenhum recurso próprio, eles apenas fornecem um contêiner que pode ser usado para estruturar o código de forma



lógica. Olhe para o módulo apenas como um contêiner de entidade lógica / comercial.

## Interface

Interfaces são exatamente como interfaces no .NET que fornecem um contrato para as classes implementarem. O TypeScript ajuda a fornecer verificação de erros de tempo de compilação para as classes que implementam essas interfaces. Se todos os métodos não foram implementados corretamente (incluindo a assinatura do método), o TypeScript sinaliza aqueles em tempo de design, bem como o tempo de compilação. Coisa interessante sobre Interfaces é que eles não existem em JavaScript e, portanto, quando compilamos um arquivo TypeScript em JavaScript, as Interfaces são omitidas.

## Classes

O conceito de Classes é novamente muito semelhante ao mundo .NET / Java. Classes contém variáveis, propriedades e métodos que formam uma entidade lógica. O TypeScript também permite definir o escopo da variável e funções com palavras-chave como “private” e “public”, embora esse escopo não tenha qualquer efeito sobre o JavaScript gerado.

## Funções

Funções são métodos em que a lógica é implementada. O TypeScript fornece suporte de tempo de compilação para garantir que qualquer pessoa que chame a função mencionada concorde com o argumento de entrada e com o tipo de valor de retorno.

## Variáveis

Variáveis são os campos definidos dentro de uma classe ou função. O TypeScript nos permite definir uma variável usando a palavra-chave “var” e atribuir um tipo de dados a ela. Uma vez que um tipo de dado é atribuído, qualquer uso adicional da variável deve ser com o mesmo tipo de dados, caso contrário, o TypeScript gerará erro no design e no tempo de compilação. O TypeScript também é inteligente o suficiente para inferir o tipo de uma variável e depois tratá-la como esse tipo quando uma variável é declarada e inicializada. Nos casos em que o TypeScript não é capaz de inferir o tipo, ele atribuirá esse tipo de variável “any”.

## Tipos de TypeScript

O TypeScript fornece alguns tipos primitivos (mostrados abaixo), bem como um tipo dinâmico “any”. “Any” É como “dynamic” palavra-chave em c # em que podemos atribuir qualquer tipo de valor para a variável. O TypeScript não sinaliza erros de tipo para variáveis do tipo “any”.

No TypeScript, definimos uma variável com um tipo apenas acrescentando o nome da variável com dois-pontos, seguido do nome do tipo, como mostrado no exemplo abaixo.

[Esconder o código da cópia](#)

```
var num: number = 30; //variable num is of type number
```

Abaixo está a lista de tipos primitivos disponíveis no TypeScript:

- Number: “ number” É um tipo de número primitivo em TypeScript. Não existe um tipo diferente para floatou doubleno TypeScript
- Boolean: O booleantipo " " representa trueou falsecondição
- String: O " string" representa uma seqüência de caracteres semelhantes ao C #
- Null: “ null” É um tipo especial que atribui nullvalor a uma variável
- Undefined: O “ undefined” também é um tipo especial e pode ser atribuído a qualquer variável

## Classes e Interface

### Classes

Classes TypeScript são uma unidade básica de abstração muito semelhante às classes C # / Java. No TypeScript, uma classe pode ser definida com a palavra-chave “ class” seguida do nome da classe. Classes TypeScript podem conter construtor, campos, propriedades e funções. O TypeScript permite que os desenvolvedores definam o escopo da variável dentro das classes como “ public” ou “ private”. É importante notar que as palavras-chave “ public/ private” só estão disponíveis no TypeScript, uma vez convertido para JavaScript, não há como distinguir entre os dois e ambos podem ser chamados. O TypeScript define um construtor usando a palavra-chave " constructor".

Exemplo em Typescript

```
class Student {  
    private firstName: string;  
    private lastName: string;  
    yearOfBirth: number; //Public scope by default  
    schoolName: string;  
    city: string;  
    //Constructor  
    constructor(firstName: string, lastName: string, schoolName: string,  
        city: string, yearOfBirth: number) {  
  
        this.firstName = firstName;  

```

```

        this.lastName = lastName;
        this.yearOfBirth = yearOfBirth;
        this.city = city;
        this.schoolName = schoolName;

    }
    //Function
    age() {
        return 2014 - this.yearOfBirth;
    }
    //Function
    printStudentFullName(): void {
        alert(this.lastName + ',' + this.firstName);
    }
}

```

### Exemplo em Javascript

```

var Student = (function () {
    //Constructor
    function Student(firstName, lastName, schoolName, city, yearOfBirth) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.yearOfBirth = yearOfBirth;
        this.city = city;
        this.schoolName = schoolName;
    }
    //Function
    Student.prototype.age = function () {
        return 2014 - this.yearOfBirth;
    };

    //Function
    Student.prototype.printStudentFullName = function () {
        alert(this.lastName + ',' + this.firstName);
    };
    return Student;
})();

```

No construtor acima definido em TypeScript, temos poucas variáveis de entrada que são então mapeadas para variáveis locais dentro de uma classe, podemos modificar esse construtor para implementar declaração e mapeamento de variáveis implícitas definindo o escopo juntamente com o nome da variável na definição do construtor, como mostrado abaixo:

```
constructor(private firstName: string, private lastName: string, public
schoolName: string,
            public yearOfBirth: number) {}
```

No caso acima, as variáveis agora são definidas e declaradas apenas dentro do argumento do construtor. O escopo mencionado junto com o argumento se torna o escopo da variável para essa classe.

Para consumir a classe, o comportamento é semelhante ao C #, onde usamos a newpalavra-chave “ ” para inicializar o objeto de classe, passar quaisquer parâmetros se o construtor exigir e, em seguida, chamar as funções ou acessar as publicvariáveis da classe.

```
var student = new Student('Tom', 'Hanks', 'World Acting School',1950);
var age = student.age();
var fullName = student.printStudentFullName();
var schoolName = student.schoolName;
```

## Interfaces

O TypeScript oferece suporte para Interfaces para usá-los como um contrato para classes semelhantes a C #. Para declarar uma interface, usamos a palavra-chave “ interface” seguida do nome da interface. O importante a saber sobre interfaces é que, quando compilado em JavaScript, o código da interface é ignorado e não há JavaScript correspondente gerado.

```
interface IStudent {
    yearOfBirth: number;
    age : () => number;
}
```

As classes implementam interfaces usando a palavra-chave “ implement” seguida do nome da interface. Como em C #, as classes podem implementar várias interfaces e o TypeScript faz uma verificação de tempo de design para garantir que a classe esteja implementando todos os métodos dessa interface.

```
class Student implements IStudent
```

Aqui, a Studentclasse agora terá que implementar o agemétodo “ ” e definir a propriedade ” yearOfBirth”. Caso contrário, o TypeScript mostrará o erro de

tempo de projeto com o erro mencionando qual propriedade / método não foi implementada na classe.

### Herança

Ter classes e interface significa que o TypeScript também suporta herança, que é um recurso muito poderoso e alinha a escrita do código do lado do cliente com a maneira como escrevemos código C#. Usando herança, podemos estender classes, implementar e estender interfaces e escrever código que seja reconhecido de perto com OOPs. Em TypeScript, quando estendemos uma classe base na classe filha, usamos a palavra-chave “super” para chamar o construtor da classe base ou até mesmo os publicmétodos da classe base.

Para estender uma classe no TypeScript, usamos “extend” palavra-chave após o nome da classe e, em seguida, seguimos pela classe pela qual precisamos estender. Também podemos herdar interfaces em outras interfaces.

```
//Interface
interface IStudent {
    yearOfBirth: number;
    age : () => number;
}

//Base Class
class College {
    constructor(public name: string, public city: string) {
    }
    public Address(streetName: string) {
        return ('College Name:' + this.name + ' City: ' + this.city + '
Street Name: ' + streetName);
    }
}

//Child Class implements IStudent and inherits from College
class Student extends College implements IStudent {
    firstName: string;
    lastName: string;
    yearOfBirth: number;
    //private _college: College;
    //Constructor
    constructor(firstName: string, lastName: string, name: string, city:
string, yearOfBirth: number) {
        super(name, city);
        this.firstName = firstName;
        this.lastName = lastName;
        this.yearOfBirth = yearOfBirth;
    }
    age () {
        return 2014 - this.yearOfBirth;
    }
    CollegeDetails() {
```

```

        var y = super.Address('Maple Street');
        alert(y);
    }
    printDetails(): void {
        alert(this.firstName + ' ' + this.lastName + ' College is: ' +
this.name);
    }
}

```

No TypeScript, também podemos definir qualquer parâmetro com um valor padrão e, ao chamar essa função, não precisamos passar o valor desses parâmetros. Se não passarmos o valor, então a função recebe o valor padrão atribuído, senão o valor passado na chamada de função é obtido como mostrado no exemplo abaixo:

```

constructor(private firstName: string, private lastName: string, public
schoolName: string,
            public yearOfBirth: number = 1990){}

```

No exemplo acima, atribuímos um valor opcional ao yearOfBirth campo “ ” e se no caso a função de chamada não passar nenhum valor para este campo, o construtor irá inicializá-lo com 1990.

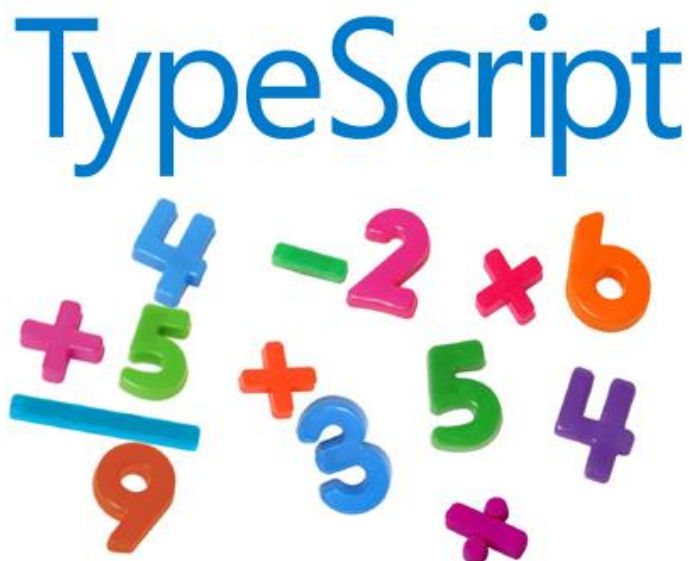
Similar é o caso dos parâmetros Opcionais, podemos definir um parâmetro adicionando um “ ?” após o nome do parâmetro. Nesse caso, quando a função é chamada, não precisamos passar o valor para esse parâmetro.

```

Subject(subjectList?: string[]) {
    if (subjectList == null) {
        alert('Oh, You have not subscribed to any course');
    }
}

```

Aqui, se chamarmos o método de assunto sem passar nenhum valor de parâmetro, o TypeScript não mostrará nenhum erro.



Os operadores aritméticos são aqueles que nós estudamos na escola, aquelas funções básicas de somar, subtrair, multiplicar e dividir. Utilizando esses operadores nós podemos criar operações matemática com valores numéricos.

Nesse artigo eu irei demonstrar como utilizar esses operadores com JavaScript, utilizando o TypeScript.

Abaixo você tem uma lista com os operadores que nós podemos utilizar com JavaScript:

- adição (+)
- subtração (-)
- multiplicação (\*)
- divisão (/)
- módulo — resto da divisão — (%)
- exponenciação (\*\*)
- incremento (++)
- decremento (- -)

### Adição (+)

Nós utilizamos esse operador para somar valores numéricos:

```
var num1 = 13;  
var num2 = 7;  
console.log(num1 + num2); // resultado 20
```

## Subtração (-)

Nós utilizamos esse operador para subtrair valores numéricos, retornando a diferença entre eles

```
var num1 = 20;  
var num2 = 10;  
console.log(num1 - num2); // resultado 10
```

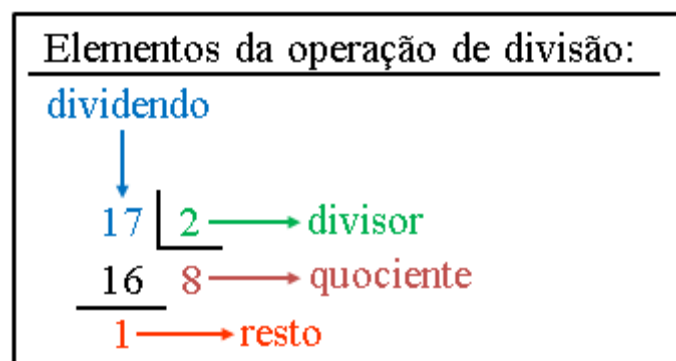
## Multiplicação (\*)

Nós utilizamos esse operador multiplicar valores numéricos

```
var num1 = 3;  
var num2 = 5;  
console.log(num1 * num2); // resultado 15
```

## Divisão (/)

Esse operador produz o quociente de seus **operandos** onde o operando da esquerda é o dividendo e o da direita é o divisor. Para que você possa relembrar, segue uma imagem abaixo retirada do Google.



```
var num1 = 20;  
var num2 = 5;  
console.log(num1 / num2); // resultado 4
```



## Módulo (%)

Esse operador retorna o resto da divisão

```
var num1 = 3;
var num2 = 2;
console.log(num1 % num2); // resultado 1
var num3 = 20;
var num4 = 5;
console.log(num3 % num4); // resultado 0
var num5 = -1;
var num6 = 2
console.log(num5 % num6); // resultado -1
```

## Exponenciação (\*\*)

Esse operador retorna o valor do primeiro operando elevado ao segundo operado

```
var num1 = 3;
var num2 = 2;
console.log(num1 ** num2); // resultado 9
```

## Incremento (++)

Esse operador incrementa um valor ao operando.

- Caso ele seja utilizado depois do operando Ex.: operando++, ele retorna o valor antes de incrementar.
- Caso ele seja utilizado antes do operando Ex.: ++ operando, ele retorna o valor já incrementado

```
var num1 = 3;
console.log(num1++); // resultado 3
var num2 = 1;
console.log(++num2); // resultado 2
```

## Decremento (- -)

Esse operador decrementa (subtrai um de) seu operando e retorna um valor.

- Caso ele seja utilizado depois do operando Ex.: operando - -ele retorna o valor antes de decrementar
- Caso ele seja utilizado antes do operando Ex.: — operando, ele retorna o valor já decrementado

Esses são os operadores básicos para que possamos criar as nossas expressões do dia a dia.

## TypeScript: Operadores lógicos

# and or not

Operadores lógicos são utilizados para comparar dois ou mais valores, retornando um valor Boolean. Abaixo você tem uma breve descrição deles:

- && (And): O operador (and) irá retornar true somente se todos os valores da expressão forem verdadeiros
- || (OR): O operador (or) irá retornar true se um dos valores comparados forem verdadeiros
- ! (NOT): O operador (not) irá retornar o inverso do esperado na expressão

Para ficar mais claro, veja abaixo alguns exemplos:

### && (and)

```
let num1: Number = 1;
let num2: Number = 3;

let valida = num1 == num2;
let valida2 = num1 > num2;
let valida3 = num1 < num2;

console.log(valida); //false
console.log(valida2); //false
console.log(valida3); //true
```

Analisado os exemplos acima nós temos:

- num1 == num2: retorna false porque num1 e num2 são diferentes
- num1 > num2: retorna false porque num1 não é maior que num2
- num1 < num2: retorna true porque num2 é maior que num1

## || (or)

```
let num1: Number = 1;
let num2: Number = 3;

let valida = num1 > 3 || num2 < 10;
let valida2 = num1 > 1 || num2 > 3;
let valida3 = num1 >= 1 || num2 >= 3;

console.log(valida); //true
console.log(valida2); //false
console.log(valida3); //true
```

Analisado o exemplo acima nós temos:

- `num1 > 3 || num2 < 10`: retorna true porque mesmo que `num1` seja menor que 3, `num2` é menor que 10 (temos uma expressão verdadeira)
- `num1 > 1 || num2 > 3`: retorna false porque `num1` não é maior que 1 e nem `num2` é maior que 3
- `num1 >= 1 || num2 >= 3`: retorna true porque estamos comparando se os valores são maior ou igual aos valores iniciados nas nossas variáveis

## ! (not)

```
let num1: Number = 1;
let num2: Number = 3;

let valida = num1 != num2;

console.log(valida); //true
```

- `num1 != num2`: retorna true porque `num1` e `num2` são diferentes.

## O que é Programação Funcional?



Em ciência da computação, programação funcional é um paradigma de programação que trata a computação como uma avaliação de funções matemáticas e que evita estados ou dados mutáveis. Ela enfatiza a aplicação de funções, em contraste da programação imperativa, que enfatiza mudanças no estado do programa<sup>[1]</sup>. Enfatizando as expressões invés de comandos, as expressões são utilizados para calculo de valores com dados imutáveis.

Uma função, neste sentido, pode ter ou não ter parâmetros e um simples valor de retorno. Os parâmetros são os valores de entrada da função, e o valor de retorno é o resultado da função. A definição de uma função descreve como a função será avaliada em termos de outras funções. Por exemplo, a função é definida em termos de funções de exponenciação e adição. Do mesmo modo, a linguagem deve oferecer funções básicas que não requerem definições adicionais.

A Programação funcional trata a funções de forma em que estas possam ser passadas como parâmetro e valores para outras e funções e podendo ter o resultado armazenado em uma constante.<sup>[3]</sup>

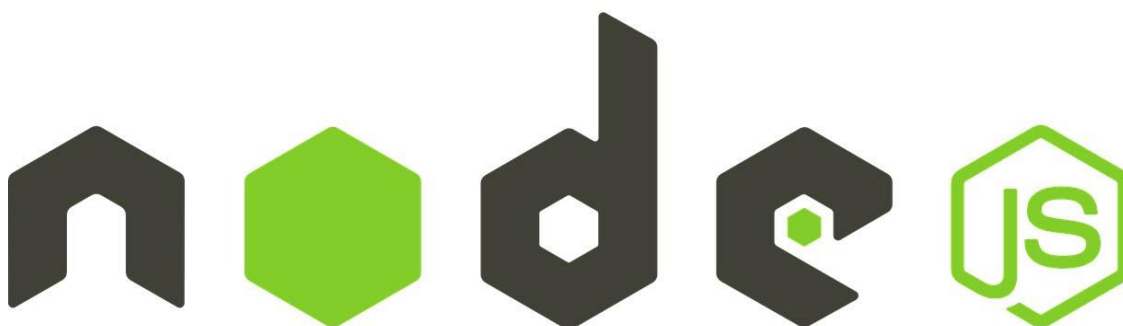
Linguagens de programação funcionais, especialmente as puramente funcionais, tem sido mais usadas academicamente que no desenvolvimento comercial de software. Entretanto, algumas linguagens notáveis usadas na indústria e no comércio incluem Erlang (aplicações concorrentes), R (estatística), Mathematica (matemática simbólica), J, K (análise financeira) e XSLT. Importantes influências na programação funcional foram o cálculo lambda, as linguagens de programação APL e Lisp, e mais recentemente ML, Haskell, OCaml, F# e Elixir.

A **programação funcional** é um paradigma que visa estruturar a construção de softwares seguindo o modelo de funções matemáticas, objetivando a imutabilidade dos dados e o acoplamento de funções, que resultam em

benefícios como consistência tanto do código quanto dos dados – veremos os porquês no decorrer do artigo.

Semelhante a uma função matemática, um código funcional tem a sua saída condicionada exclusivamente à sua entrada, de modo que uma entrada garanta sempre a mesma saída após a execução do mesmo método, eliminando os chamados *side-effects*, que são as mudanças de estado ocasionadas por outros fatores que não dependam dos parâmetros do método.

## O que é Node?



Node.js é um interpretador, com código aberto, de código JavaScript de modo assíncrono e orientado a eventos, focado em migrar a programação do Javascript do lado do cliente para os servidores, criando assim aplicações de alta escalabilidade (como um servidor web), capazes de manipular milhares de conexões/requisições simultâneas em tempo real, numa única máquina física. O Node.js é baseado no interpretador V8 JavaScript Engine (interpretador de JavaScript open source implementado pelo Google em C++ e utilizado pelo Chrome). Foi criado por Ryan Dahl em 2009, e seu desenvolvimento é mantido pela fundação Node.js em parceria com a Linux Foundation.

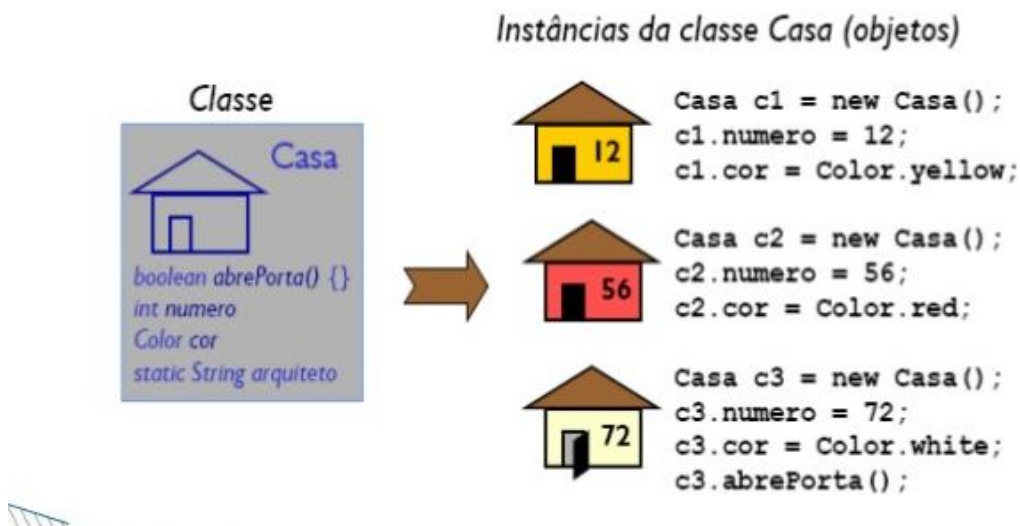
## O que é Express?



Express.js , ou simplesmente Express , é uma estrutura de aplicativo da web para o Node.js , lançada como software livre e de código aberto sob a licença MIT . Ele é projetado para construir aplicativos da Web e APIs . Ele foi chamado de framework de servidores padrão de fato para o Node.js.

O autor original, TJ Holowaychuk , descreveu-o como um servidor inspirado no Sinatra , o que significa que é relativamente mínimo com muitos recursos disponíveis como plugins. Express é o componente backend da pilha MEAN , junto com o software de banco de dados MongoDB e a estrutura de frontend AngularJS .

## Classe e Objeto



Uma classe é uma forma de definir um tipo de dado em uma linguagem orientada a objeto. Ela é formada por dados e comportamentos.

Para definir os dados são utilizados os atributos, e para definir o comportamento são utilizados métodos. Depois que uma classe é definida podem ser criados diferentes objetos que utilizam a classe. A Listagem 1 mostra a definição da classe Empresa, que tem os atributos nome, endereço, CNPJ, data de fundação,

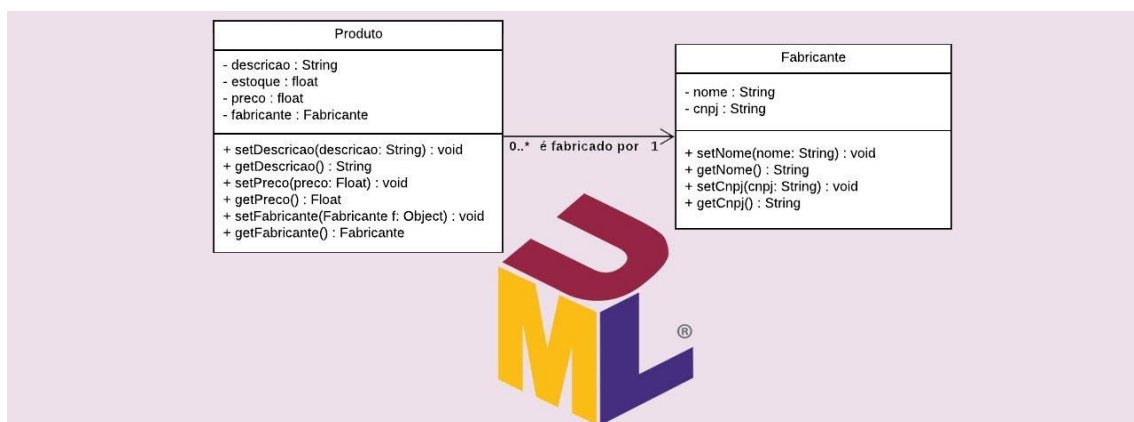
faturamento, e também o método imprimir, que apenas mostra os dados da empresa.

## Encapsulamento

O conceito do encapsulamento consiste em “esconder” os atributos da classe de quem for utilizá-la. Isso se deve por dois motivos principais.

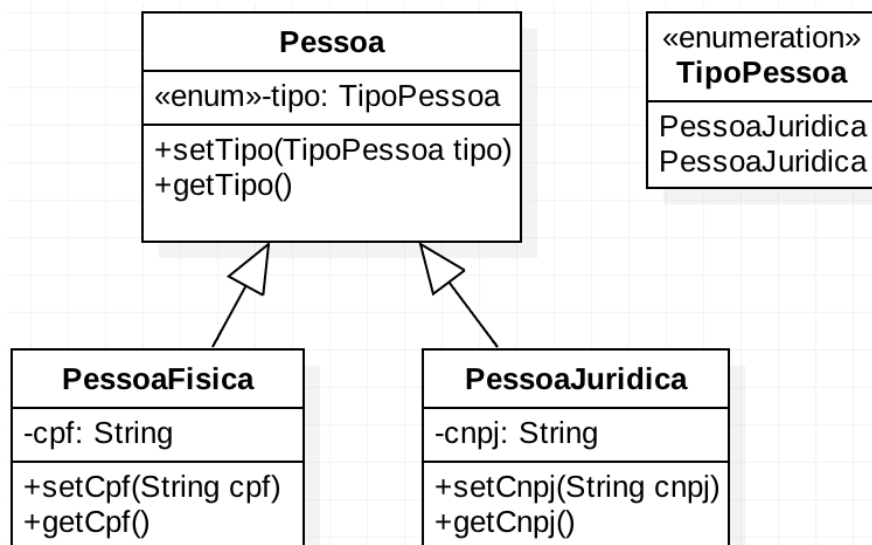
Um é para que alguém que for usar a classe não a use de forma errada como, por exemplo, em uma classe que tem um método de divisão entre dois atributos da classe - se o programador java não conhecer a implementação interna da classe, ele pode colocar o valor zero no atributo do dividendo, mas se a classe estiver corretamente encapsulada podemos impedir que o programador faça isso.

## Associação de Classes



A associação de classes indica quando uma classe tem um tipo de relacionamento "tem um" com outra classe como, por exemplo, uma pessoa tem um carro e isso indica que a classe Pessoa tem uma associação com a classe Carro. Esse tipo de relacionamento entre classes é importante, porque define como as classes interagem entre elas nas aplicações.

## Herança

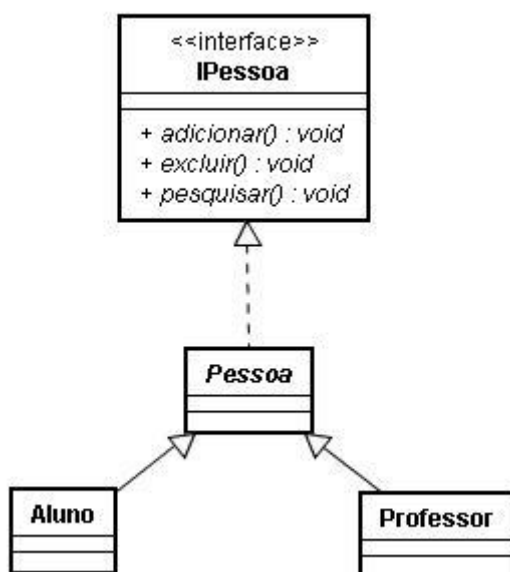


A herança é um tipo de relacionamento que define que uma classe "é um" de outra classe como, por exemplo, a classe `Funcionario` que é uma `Pessoa`, assim um `Funcionario` tem um relacionamento de herança com a classe `Pessoa`. Em algumas linguagens, como C, é possível fazer herança múltipla, isto é, uma classe pode herdar de diversas outras classes, mas em Java isso não é permitido, pois cada classe pode herdar de apenas outra classe.

Herança é um princípio de orientação a objetos, que permite que classes compartilhem atributos e métodos, através de "heranças". Ela é usada na intenção de reaproveitar código ou comportamento generalizado ou especializar operações ou atributos. O conceito de herança de várias classes é conhecido como herança múltipla. Como exemplo pode-se observar as classes `'aluno'` e `'professor'`, onde ambas possuem atributos como nome, endereço e telefone. Nesse caso pode-se criar uma nova classe chamada por exemplo, `'pessoa'`, que contenha as semelhanças entre as duas classes, fazendo com que `aluno` e `professor` herdem as características de `pessoa`, desta maneira pode-se dizer que `aluno` e `professor` são subclasses de `pessoa`. Também podemos dizer que uma classe pode ser abstrata(`abstract`) ou seja ela não pode ter uma instância, ela apenas "empresta" seus atributos e métodos como molde para novas classes.



## Interface



Em algumas linguagens de programação, o termo interface (ou protocolo) é uma referência à característica que permite a construção de interfaces que isolam do mundo exterior os detalhes de implementação de um componente de software.

O princípio da interface é um alicerce da programação modular que, por sua vez, é precursora e parte da programação orientada a objeto. Na programação orientada a objeto, a interface de um objeto consiste de um conjunto de métodos que um objeto deve suportar. É importante notar que as variáveis de instância não fazem parte da interface de um objeto pois devem ser acessadas somente pelos "métodos de acesso". Historicamente, as interfaces são derivadas dos arquivos de cabeçalho da Linguagem C (normalmente arquivos com extensão ".h") que separam o contexto sintático de um módulo (ou protótipos de funções) da sua implementação.

Algumas linguagens de programação orientadas a objeto exigem que a interface do objeto seja especificada de forma separada da implementação do objeto, enquanto outras não fazem esta exigência. Por exemplo, em linguagens de programação como Objective-C, a classe do objeto define a sua interface e é declarada em um arquivo de cabeçalho (header em inglês) e, por outro lado, a implementação da classe é mantida em um arquivo chamado de "arquivo fonte". Devido à tipagem dinâmica existente na Objective-C, que permite o envio de mensagens para qualquer objeto, a interface de uma classe é importante para determinar para quais métodos um objeto de uma classe responde.

A linguagem de programação Java, que recebeu influência da Objective-C, utiliza outra abordagem para o conceito de interface, assim como outras

linguagens orientadas a objeto, onde a interface especifica um conjunto de métodos ou funcionalidades comuns a um conjunto de classes. Ver interface na linguagem Java.

## Desenvolvimento de Software



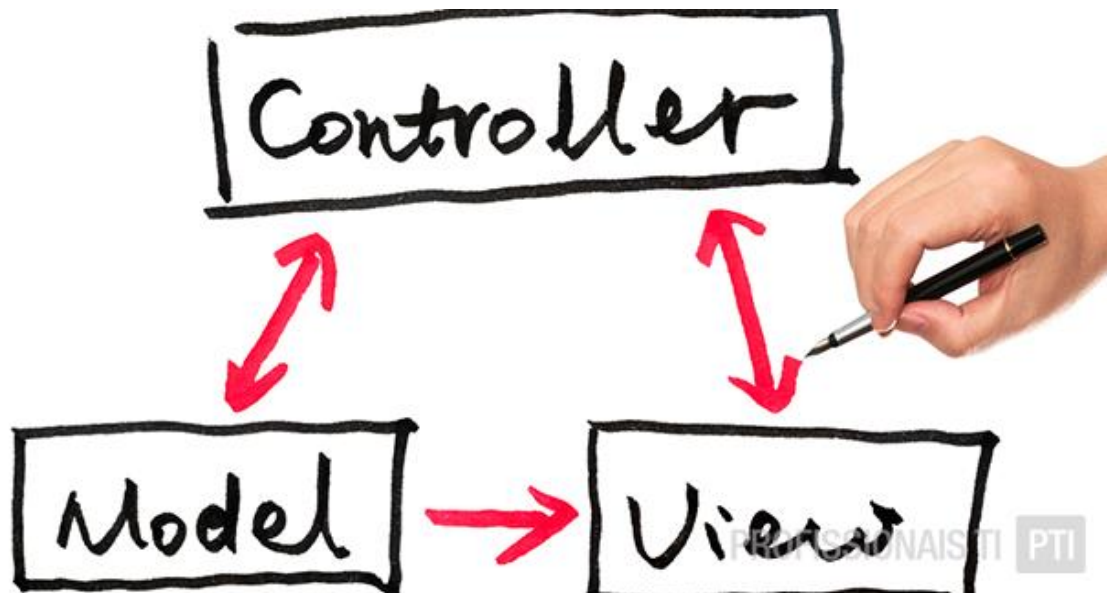
Em arquitetura de software há muitas camadas entre o hardware e o usuário final. Cada uma pode ser dita como tendo um front-end e um back-end. O front-end é uma abstração, simplificando o componente subjacente pelo fornecimento de uma interface amigável, como por um exemplo um navegador de Internet, ou um formulário para um determinado usuário.

Em projetos de software, por exemplo, a arquitetura modelo-visão-controlador fornece o -end e o back-end para o banco de dados, o usuário e para os componentes de processamento. A separação de sistemas de software em front-end e back-end simplifica o desenvolvimento e separa a manutenção. Uma regra de ouro é que o lado frontal (ou "cliente") é qualquer componente manipulado pelo usuário. O código do lado do servidor (ou back-end) reside no servidor. A confusão surge quando alguém tem que fazer edições na parte frontal para arquivos do lado servidor. Muitos projetistas HTML, por exemplo, não precisam estar no servidor quando eles estão desenvolvendo o HTML. Reciprocamente, engenheiros do lado servidor, por definição, nunca estão em qualquer coisa, mas num servidor. Considera-se os dois para, finalmente, fazer um site web funcional e interativo.

Para subsistemas de computação maiores, um gerenciador de arquivos gráfico é um front-end para o sistema de arquivos do computador e um shell faz a

interface com o sistema operacional. O front-end veste o usuário e o back-end executa os programas do sistema operacional em resposta.

## Arquitetura MVC



**Arquitetura Modelo-Visão-Controlador** (do inglês: Model-View-Controller - MVC) é um padrão de arquitetura de software (não confundir com design pattern) que separa a representação da informação da interação do usuário com ela. Normalmente usado para o desenvolvimento de interfaces de usuário que divide uma aplicação em três partes interconectadas. Isto é feito para separar representações de informação internas dos modos como a informação é apresentada para e aceita pelo usuário. O padrão de projeto MVC separa estes componentes maiores possibilitando a reutilização de código e desenvolvimento paralelo de maneira eficiente.

O modelo (**model**) consiste nos dados da aplicação, regras de negócios, lógica e funções. Uma visão (**view**) pode ser qualquer saída de representação dos dados, como uma tabela ou um diagrama. É possível ter várias visões do mesmo dado, como um gráfico de barras para gerenciamento e uma visão tabular para contadores. O controlador (**controller**) faz a mediação da entrada, convertendo-a em comandos para o modelo ou visão. As ideias centrais por trás do MVC são a reusabilidade de código e separação de conceitos.

Tradicionalmente usado para interfaces gráficas de usuário (GUIs), esta arquitetura tornou-se popular para projetar aplicações web e até mesmo para

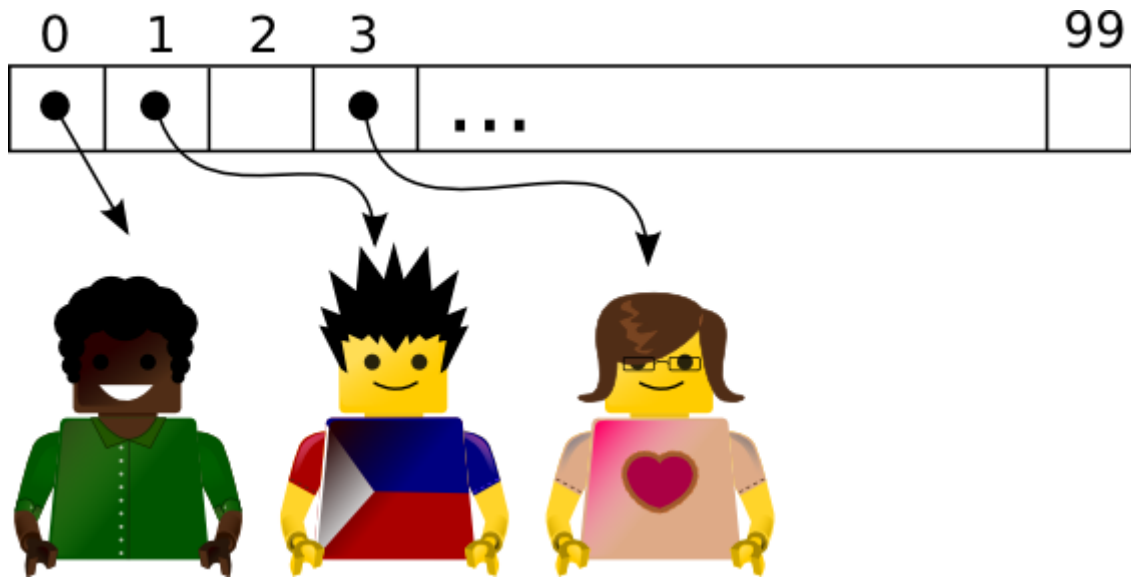
aplicações móveis, para desktop e para outros clientes.[3] Linguagens de programação populares como Java, C#, Ruby, PHP e outras possuem frameworks MVC populares que são atualmente usados no desenvolvimento de aplicações web.

**Camada de apresentação ou visualização (View)** - Não se dedica em saber como o conhecimento foi retirado ou de onde ela foi obtida, apenas mostra a referência. Segundo Gamma et al (2006), "A abordagem MVC separa a View e Model por meio de um protocolo inserção/notificação (subscribe/notify). Uma View deve garantir que sua expressão reflita o estado do Model. Sempre que os dados do Model mudam, o Model altera as Views que dependem dele. Em resposta, cada View tem a oportunidade de modificar-se". Adiciona os elementos de exibição ao usuário : HTML, ASP, XML, Applets. É a camada de interface com o usuário. É utilizada para receber a entrada de dados e apresentar visualmente o resultado.

**Camada de lógica da aplicação (Model)** - É o coração da execução, responsável por tudo que a aplicação vai fazer a partir dos comandos da camada de controle em um ou mais elementos de dados, respondendo a perguntas sobre o sua condição e a instruções para mudá-las. O modelo sabe o que o aplicativo quer fazer e é a principal estrutura computacional da arquitetura, pois é ele quem modela o problema que está se tentando resolver. Modela os dados e o comportamento por trás do processo de negócios. Se preocupa apenas com o armazenamento, manipulação e geração de dados. É um encapsulamento de dados e de comportamento independente da apresentação.

**Camada de controle (Control)** - É responsável por interpretar as ações de entrada através do mouse e teclado realizadas pelo usuário. O Controle (Controller) envia essas ações para o Modelo (Model) e para a janela de visualização (View) onde serão realizadas as operações necessárias.

## Arrays, Vetor e Matriz



Um **array** é uma estrutura de dados homogênea que mantém uma série de elementos de dados de mesmo tipo. Pode-se acessar os elementos individuais armazenados no array por meio de uma posição de índice associada, geralmente numérica.

No geral, os arrays possuem tamanho fixo, ou seja, número de posições definida; em algumas linguagens de programação, existem estruturas de arrays que possuem tamanho variável. Vamos estudar aqui os arrays tradicionais, de tamanho especificado.

### Classificação dos arrays

Os arrays são classificados de acordo com a sua dimensão de armazenamento de dados, como segue:

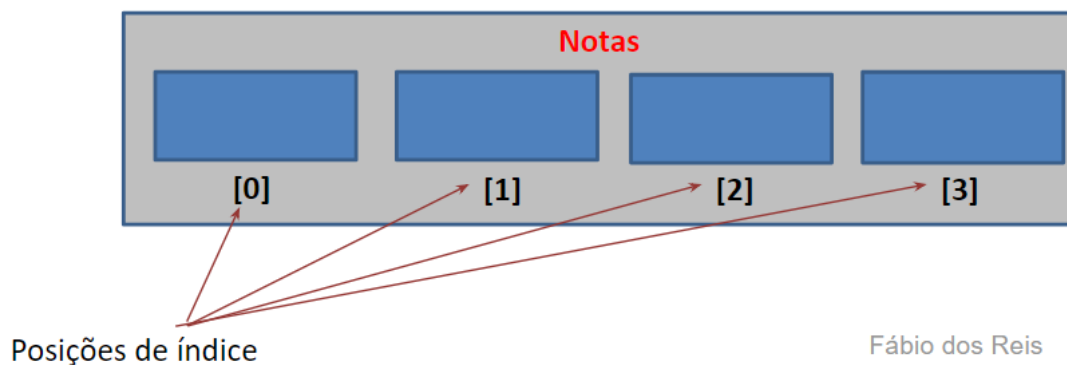
- Unidimensional: Vetor
- Bidimensional: Matriz
- Tridimensional: Cubo

Uma **matriz** é uma coleção de variáveis de mesmo tipo, acessíveis com um único nome e armazenados contiguamente na memória.

A individualização de cada variável de um vetor é feita através do uso de índices.

Os **Vetores** são matrizes de uma só dimensão.

Um **vetor** é um array unidimensional, ou seja, de uma única dimensão; é análogo a uma linha única de dados em uma planilha ou tabela. A figura a seguir ilustra a estrutura interna de um vetor de quatro posições, que permite portanto armazenar até quatro dados, de nome Notas



No geral a contagem das posições se inicia em zero (0), de modo que a primeira posição do vetor será a posição 0, a segunda posição será 1, e assim por diante; a última posição do vetor será a de número  $n - 1$ , onde  $n$  é o número total de posições disponíveis (tamanho do array). Assim, em um vetor de 4 posições a última posição será  $4 - 1 = 3$ . As posições em um vetor são sempre indicadas pelo número da posição entre colchetes [ ].

## Declaração de vetores

Podemos declarar um vetor em português estruturado usando a seguinte sintaxe:

```
nomeVetor: vetor [i..f] de Tipo_Dados
```

Onde:

- nomeVetor é o nome escolhido para o vetor, que deve seguir as regras de nomeação de variáveis.
- i = Valor da primeira posição do vetor (preferencialmente zero)
- F = Valor da última posição do vetor (tamanho do vetor - 1)
- Tipo\_Dados = tipo dos dados que serão armazenados nas posições do vetor

## Declaração de Matrizes

```
int Vetor[5]; // declara um vetor de 5 posições  
int Matriz[5][3]; // declara uma matriz de 5 linhas e 3 colunas
```

### Acesso aos elementos do vetor

Para acessar os elementos de um vetor usa-se índices. O índice define a posição da variável dentro do vetor.

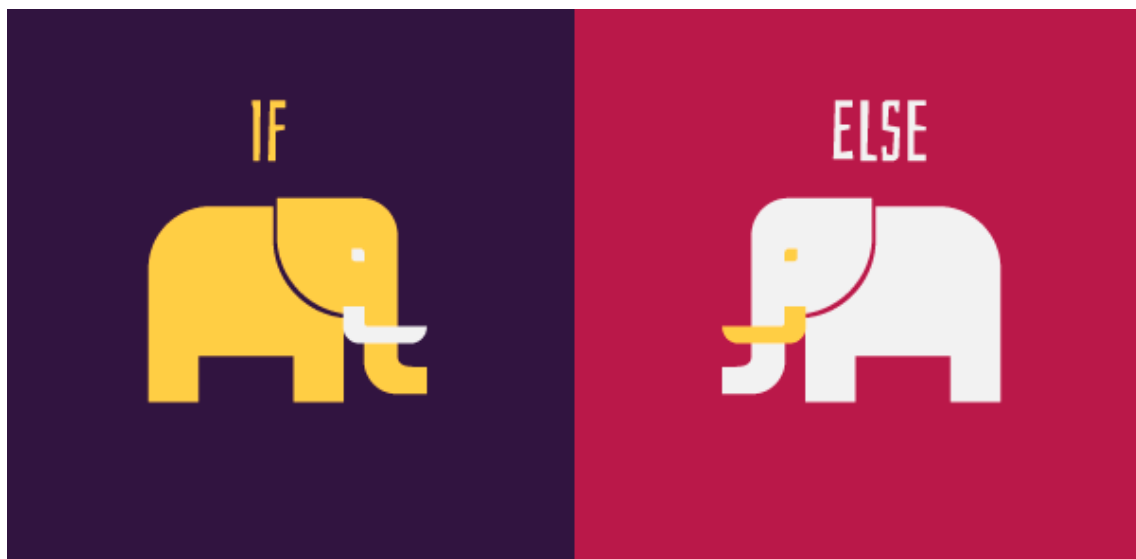
Em todos os vetores tem o primeiro elemento na posição 0(zero). Assim, se tomarmos "K" como sendo o tamanho do vetor a última posição é a de índice "K-1"

```
Vetor[0] = 4; // Coloca 4 na primeira posição de "Vetor"  
Vetor[4] = 8; // Coloca 8 na última posição de "Vetor"
```

### Exemplos com Vetores

```
int Vetor[5]; // declara um vetor de 5 posições  
int Matriz[5][3]; // declara uma matriz de 5 linhas e 3 colunas  
Vetor[0] = 9; // coloca 9 na primeira posição do vetor  
Vetor[4] = 30 // coloca 30 na última posição do vetor  
  
Matriz[0][1] = 15; // coloca 15 na célula que está na primeira linha  
// e na segunda coluna da matriz
```

## O que é o operador ternário?



No desenvolvimento de uma aplicação, é muito comum utilizarmos estruturas condicionais como, por exemplo, ifs e elses ou switch case. Porém, em algum momento da nossa vida, provavelmente, um desses testes que realizamos é tão simples que retorna um valor para apenas duas possibilidades.

Por exemplo, suponhamos que precisamos criar uma funcionalidade para gerar uma bonificação e a regra para essa funcionalidade é a seguinte:

- Se o salário for maior que R\$ 1000, o bônus é de 10%
- Se o salário for menor ou igual a R\$ 1000, o bônus é de 15%

Uma solução seria utilizar um if e else como já conhecemos, como por exemplo no Java:

```
double salario = 1000;  
double bonus = 0.0;  
  
if (salario > 1000) {  
    bonus = salario * 0.10;  
} else {  
    bonus = salario * 0.15;  
}  
  
System.out.println(bonus);
```

Nesse exemplo o resultado é 150.0 pois o salário é menor que R\$ 1000.0, ou seja, bônus de 15%.

Entretanto, perceba que o que estamos fazendo é apenas um teste bem básico que tem apenas uma única linha de código dentro do if ou do else. Será que não existe uma maneira mais simples de resolver o mesmo problema?

Em um cenário similar a esse, podemos também utilizar o operador ternário que funciona com o mesmo conceito do if e else, porém, a única diferença é que precisamos devolver um valor após o teste estritamente em uma única linha!

```
double salario = 1000;  
double bonus = salario * (salario > 1000 ? 0.10 : 0.15);  
System.out.println(bonus);
```

Com esse código acima temos o mesmo resultado de antes, ou seja, 150.0. Mas como funciona esse operador ternário? A estrutura de um operador ternário é compreendida da seguinte forma:



### ***condição? valor se for verdadeiro : valor se for falso***

Portanto, inicialmente temos um teste (podemos adicionar um teste qualquer), ou seja, qualquer teste devolve um valor booleano, então, definimos o primeiro parâmetro que é justamente o valor que será retornado caso o teste for verdadeiro e o segundo que será retornado caso for falso!

Justamente pelo fato de realizar essas 3 operações, o chamamos de operador ternário. Mas isso é só em Java? Não! Diversas linguagens implementam esse mesmo recurso! Vejamos alguns exemplos:

#### **JavaScript:**

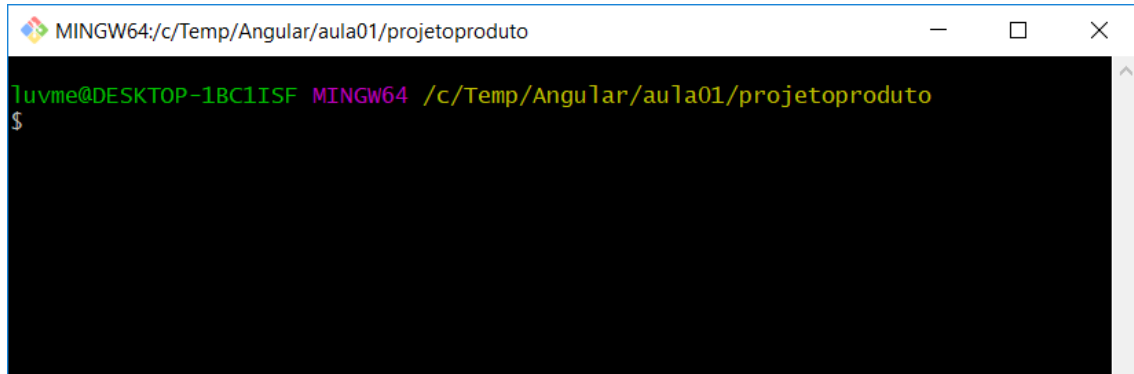
```
var salario = 1000;  
var bonus = salario * (salario > 1000 ? 0.10 : 0.15);
```

## CRIANDO UM NOVO PROJETO:

Para criar o projeto:

Criar uma pasta/diretorio para trabalhar com o projeto.

Pelo terminal(cmd), entrar nessa pasta.



```
MINGW64:/c/Temp/Angular/aula01/projetoproduto
luvme@DESKTOP-1BC1ISF MINGW64 /c/Temp/Angular/aula01/projetoproduto
$
```

Digitar o comando para criação do projeto:

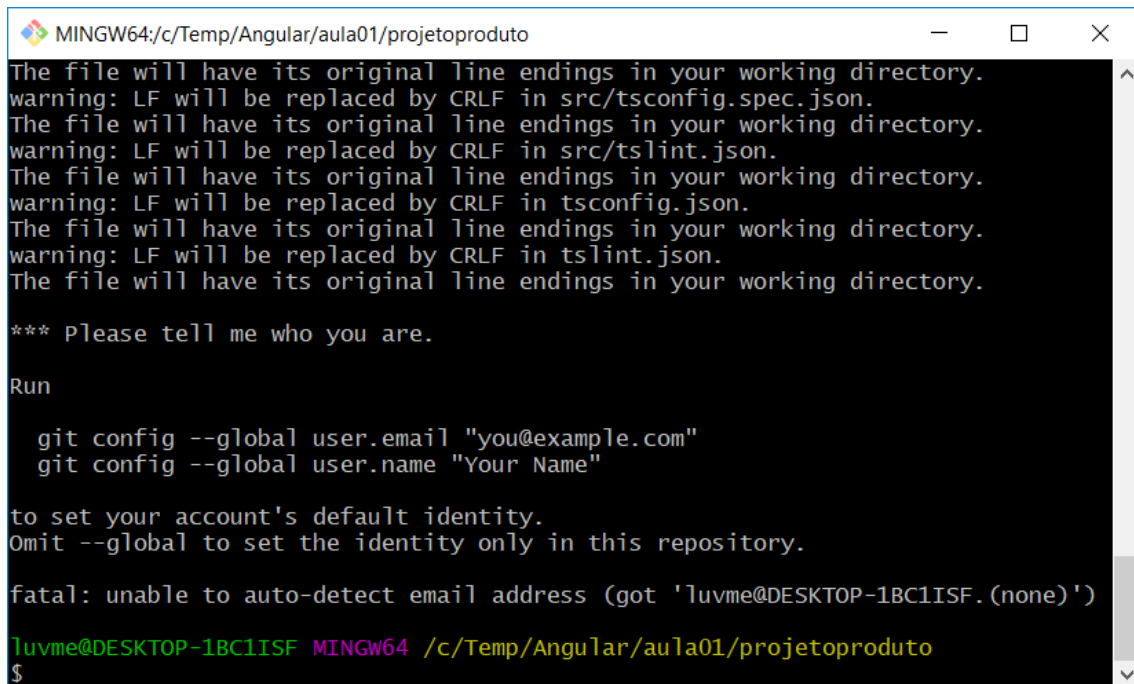
***“ng new nome\_do\_projeto”***

Digitar “y” para a criação das rotas.

Selecionar “CSS” para o estilo.

Aguardar criar o projeto..

Projeto criado...



```
MINGW64:/c/Temp/Angular/aula01/projetoproduto
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in src/tsconfig.spec.json.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in src/tslint.json.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in tsconfig.json.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in tslint.json.
The file will have its original line endings in your working directory.

*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'luvme@DESKTOP-1BC1ISF.(none)')
luvme@DESKTOP-1BC1ISF MINGW64 /c/Temp/Angular/aula01/projetoproduto
$
```

Para entrar no Visual Code:

Digitar no terminal:

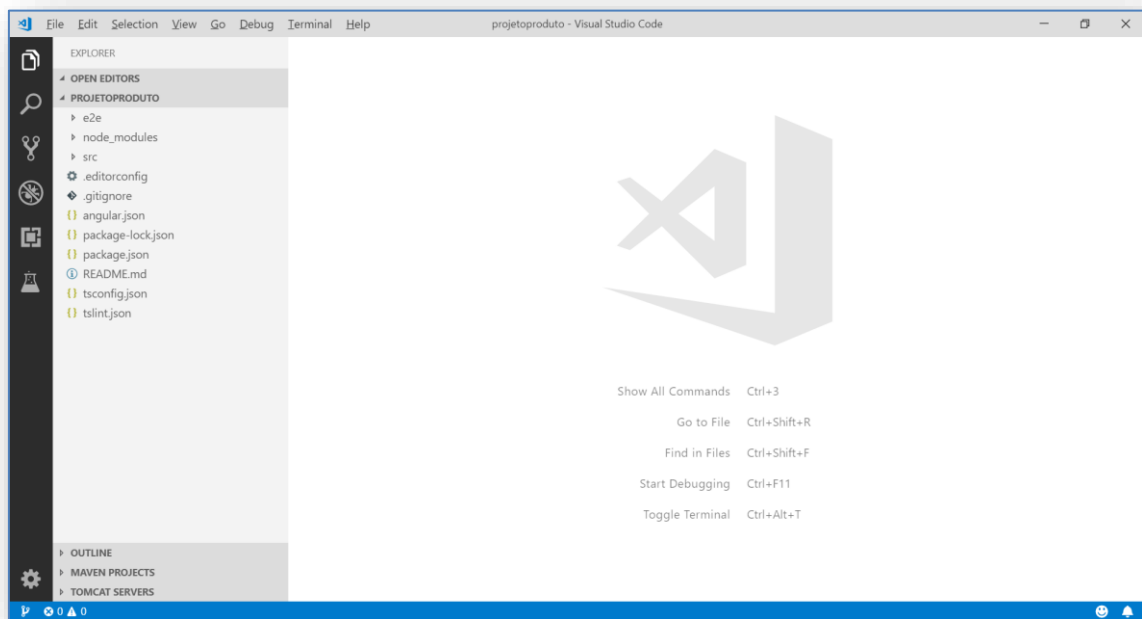
- ***“cd nome\_do\_seu\_projeto”*** (para entrar no diretório do projeto)
- ***“code .”***

```
MINGW64:/c/Temp/exemplo

The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in src/main.ts.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in src/polyfills.ts.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in src/styles.css.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in src/test.ts.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in tsconfig.app.json.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in tsconfig.json.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in tsconfig.spec.json.
The file will have its original line endings in your working directory.
warning: LF will be replaced by CRLF in tslint.json.
The file will have its original line endings in your working directory.
Successfully initialized git.

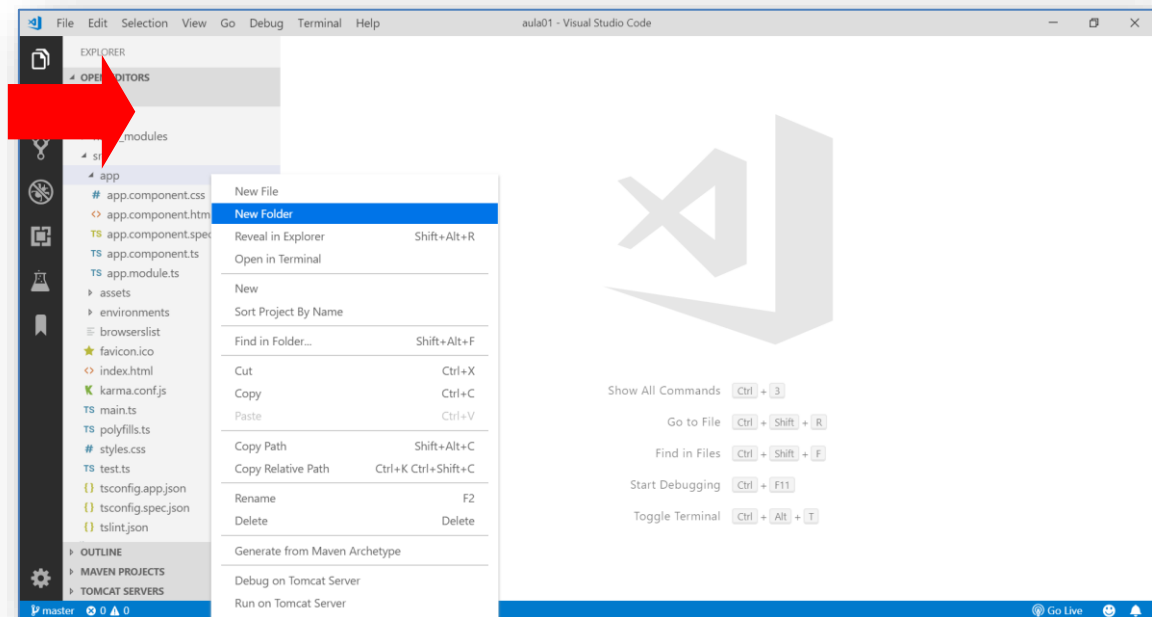
luvm@DESKTOP-1BC1ISF MINGW64 /c/Temp
$ cd exemplo

luvm@DESKTOP-1BC1ISF MINGW64 /c/Temp/exemplo (master)
$ code |.
```



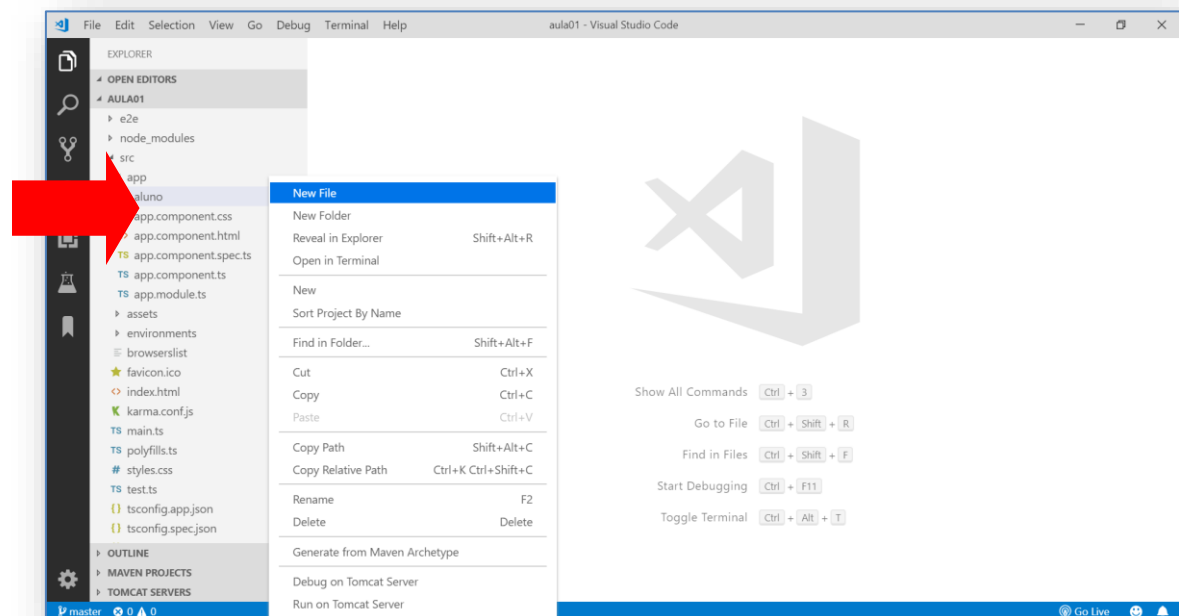
Para criar o diretório:

Clicar com o botão direito em “app” -> new folder -> digitar o nome do diretório



Para criar a classe...

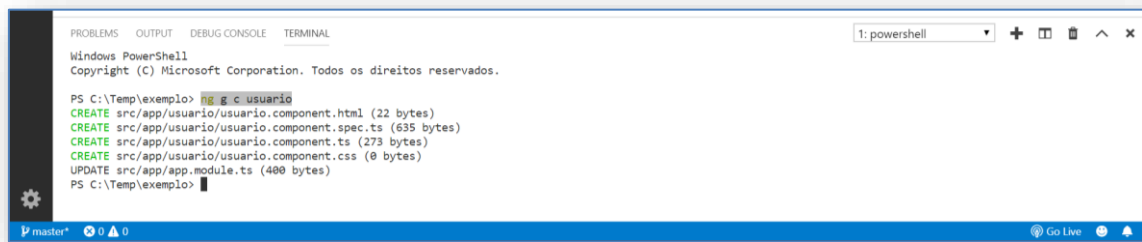
Clicar com o botão direito em app -> new file...



## Para criar um componente

Digitar no terminal do Visual Code.

**“ng g c nome\_do\_componente”**

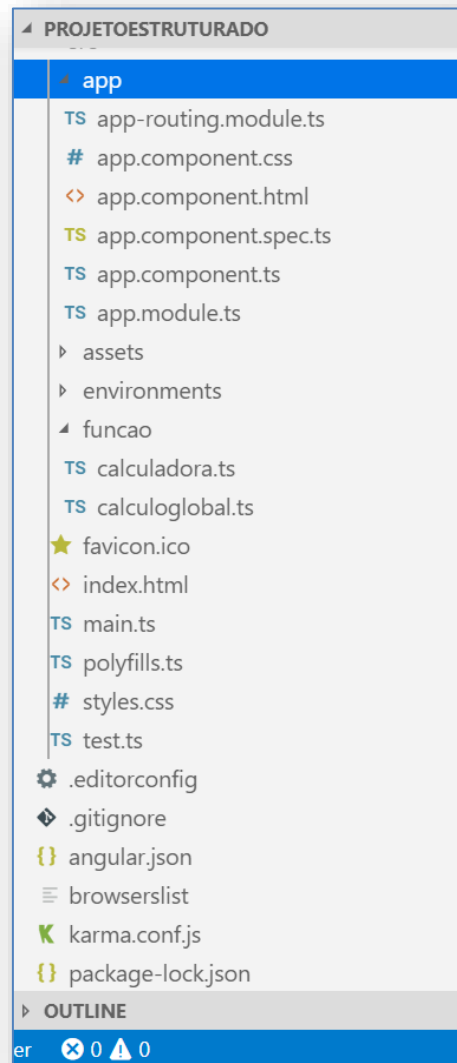


```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

PS C:\Temp\exemplo> ng g c usuario
CREATE src/app/usuario/usuario.component.html (22 bytes)
CREATE src/app/usuario/usuario.component.spec.ts (635 bytes)
CREATE src/app/usuario/usuario.component.ts (273 bytes)
CREATE src/app/usuario/usuario.component.css (0 bytes)
UPDATE src/app/app.module.ts (400 bytes)
PS C:\Temp\exemplo>
```

## PROJETO ESTRUTURADO

Estrutura do projeto depois de finalizado:



Criar o diretório “função”.

Criar a classe “calculadora.ts”

## calculadora.ts

```
// EXPORTANDO UMA FUNÇÃO DE SOMA
export const soma = function (n1: number, n2: number): number {
    return +n1 + +n2;
}

export const subtracao = function (n1: number, n2: number):
number {
    return +n1 - +n2;
}

export const divisao = function (n1: number, n2: number): number
{
    if (n2 == 0) {
        //LANÇA O ERRO CASO O NUM2 SEJA ZERO
        throw new Error('Nao pode dividir por zero');
    }
    return +n1 / +n2;
}

export const multiplicacao = function (n1: number, n2: number):
number {
    return +n1 * +n2;
}

export const dobro = function (n1: number): void {
    console.log('Dobro:', (n1 * 2));
}

export const metade = function (n1: number): void {
    console.log('metade:', (n1 / 2));
}

export const triplo = function (n1: number): void {
    console.log('triplo:', (n1 * 3));
}
```

---

## calculoglobal.ts

```
export * from './calculadora';
```

---

## app.module.ts

(Classe responsável pela importações do projeto)

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { FormsModule } from '@angular/forms'
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    FormsModule, //HABILITANDO O USO DO FORMULARIO
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

---



## app.component.ts

(Classe que programa os eventos da tela)

```
import { Component } from '@angular/core';
import { soma, subtracao, divisao, multiplicacao, dobro, metade,
triplo } from 'src/funcao/calculadora';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  //INICIANDO AS VARIÁVEIS
  numero1: number = 0;
  numero2: number = 0;
  total: number = 0;
  mensagem: string = "";

  constructor() {

  }

  //MÉTODO DO BOTÃO DE IMPRIMIR A SOMA
  printSoma() {
    //MOSTRA A MSG DE SOMA
    this.mensagem = "Soma:";
    //FAZ O CÁLCULO
    this.total = soma(this.numero1, this.numero2);
  }

  printSubtracao() {
    this.mensagem = "Subtracao:";
    this.total = subtracao(this.numero1, this.numero2);
  }

  printDivisao() {
    this.mensagem = "Divisao:";
    try {
      this.total = divisao(this.numero1, this.numero2);
    }
  }
}
```

```

        } catch (err) { //A DIVISAO PODE DAR ERRO
            console.log(err.error);
        }
    }

    printMultiplicacao() {
        this.mensagem = "Multiplicacao:";
        this.total = multiplicacao(this.numero1, this.numero2);
    }

    printDobro() {
        dobro(this.numero1);
    }

    printMetade() {
        metade(this.numero1);
    }

    printTriplo() {
        triplo(this.numero1);
    }
}

```

---

## app.component.html

```
<h1>Projeto Estrutura do Calculo</h1>
```

```
Numero1<br/>
```

```
<input type="number" [(ngModel)]="numero1" />
<br/>
```

```
Numero2
```

```
<br/>
<input type="number" [(ngModel)]="numero2" />
<br/>
```

```
<button (click)="printSoma();">Soma</button><br/>
```

```
<button (click)="printSubtracao();">Subtracao</button><br/>
<button (click)="printDivisao();">Divisao</button><br/>
<button
(click)="printMultiplicacao();">Multiplicacao</button><br/>
<button (click)="printDobro();">Dobro</button><br/>
<button (click)="printTriplo();">Triplo</button><br/>
<button (click)="printMetade();">Metade</button><br/>
<br/>
{{mensagem}}:{{total}}
```

---

## Para rodar o projeto

Digitar no terminal:

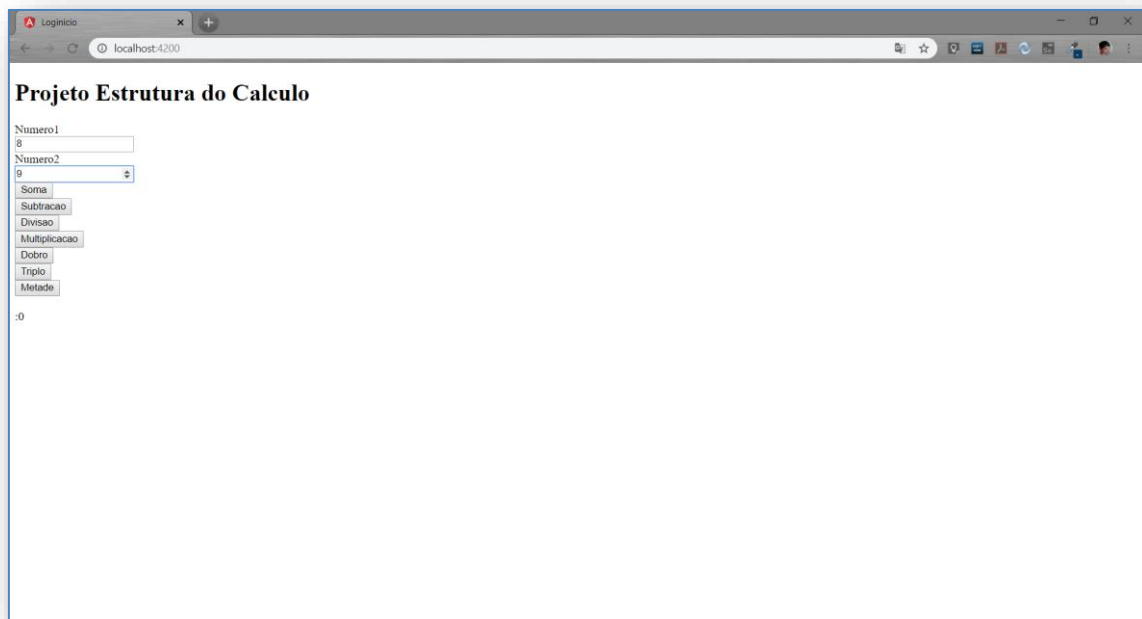
**"ng s -o"** ou

**"ng s -o --port 1234"** para caso já tenha a porta padrão (4200) em uso.

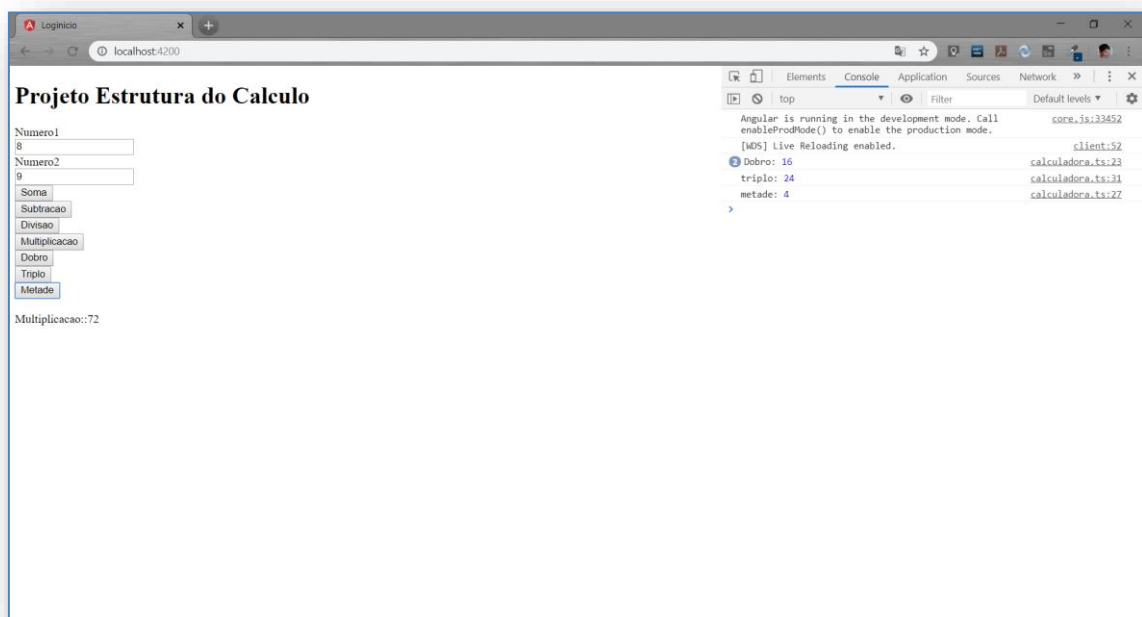
<http://localhost:4200/>



Digitando os dados

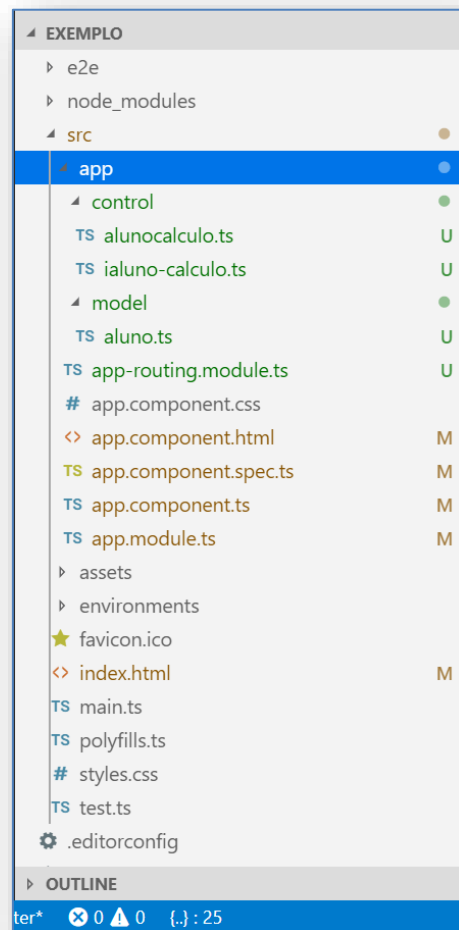


Clicando nos botões. Os cálculos aparecem abaixo do ultimo botão e os outros cálculos aparecem no console (na lateral)



## PROJETO COM INTERFACE, CALCULO FUNCIONAL E HERANÇA

Estrutura do projeto depois de finalizado:



Criar o pacote model.

Criar a classe Aluno.ts

## aluno.ts

```
export class Aluno {  
  
    id: number;  
    nome: string;  
    disciplina: string;  
    nota1: number;  
    nota2: number;  
    media: number;  
    situacao: string;  
  
}
```

---

Criar o pacote control.

Criar a interface ialuno-calculo.ts

## ialuno-calculo.ts

```
export interface IAlunoCalculo {  
  
    gerarMedia(): void;  
    gerarSituacao(): void;  
    funcionalSomaItem(): void;  
    funcionalSoma(): void;  
  
}
```

---

## alunocalculo.ts

```
import { IAlunoCalculo } from './ialuno-calculo';
import { Aluno } from '../model/aluno';

//CLASSE ALUNOCALCULO IMPLEMENTANDO A INTERFACE IALUNOCALCULO
export class AlunoCalculo implements IAlunoCalculo {

    //VETOR DE ALUNOS COM OS DADOS DA CLASSE
    listAluno = [{
        id: 10,
        nome: "lu",
        disciplina: "java",
        nota1: 9.,
        nota2: 8.,
        media: 0,
        situacao: "",
    }, {
        id: 11,
        nome: "belem",
        disciplina: "java",
        nota1: 7.,
        nota2: 8.,
        media: 0,
        situacao: "",
    }
    ] as Aluno[];

    //CRIANDO AS VARIÁVEIS PARA TRABALHAR E JÁ INICIALIZANDO
    aluno: Aluno;
    alunos: Aluno[] = [];
    soma: number[];
    somatudo: number;

    //CONTRUTOR DA CLASSE INICIANDO ALUNO COM ESPAÇO DE MEMORIA
    constructor() {
        this.aluno = new Aluno();
    }

    //CALCULO DE GERAR MEDIA DO TIPO VOID (SEM RETORNO)
    public gerarMedia(): void {
```

```

        this.aluno.media = (this.aluno.nota1 + this.aluno.nota2)
    / 2;
    }

    //CALCULO DE GERAR A SITUAÇÃO DO ALUNO UTILIZANDO O TERNÁRIO
    public gerarSituacao(): void {
        this.aluno.situacao = (this.aluno.media >= 7) ?
"aprovado" : "reprovado"
    }

    //METODO FUNCIONAL UTILIZANDO O FILTER
    //SOMAR TODAS AS NOTA1 E NOTA2 DOS ID'S MAIORES QUE ZERO
    public funcionalSomaItem(): void {
        this.soma = this.listAluno.filter(x => x.id > 0).map(x
=> {
            return (x.nota1 + x.nota2)
        }
    );
    }

    //METODO FUNCINAL UTILIZANDO FILTER E REDUCE
    public funcionalSoma(): void {
        this.somatudo = this.listAluno.filter(x => x.id > 0).
            reduce((sum, aluno) => {
                return sum + (aluno.nota1 + aluno.nota2);
            }, 0);
    }
}

```

---

## app.module.ts

(Classe responsável pela importações do projeto)

```

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModuleModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { FormsModule } from '@angular/forms';

@NgModule({

```



```

    declarations: [
        AppComponent
    ],
    imports: [
        FormsModule, //HABILITAR O USO DE FORMULARIO (NGMODEL)
        BrowserModule,
        AppRoutingModule
    ],
    providers: [],
    bootstrap: [AppComponent]
})
export class AppModule { }

```

---

## app.component.ts

(Classe que programa os eventos da tela)

```

import { Component } from '@angular/core';
import { AlunoCalculo } from '../control/alunocalculo';

@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})
//APPCOMPONENT HERDA ALUNOCALCULO
export class AppComponent extends AlunoCalculo {

    title = 'classe';

    //CRIAMOS O METODO PARA ACIONAR O BOTÃO DE CALCULO DE MEDIA
    calcMedia() {
        this.gerarMedia();
    }

    //CRIAR O METODO PARA ACIONAR O BOTÃO
    calcSituacao() {
        this.gerarSituacao();
    }
}

```

```

//MESMO DO ANTERIOR
calcFuncional() {
    this.funcionalSoma();
}
}

```

---

## app.component.html

```

<h1>Projeto Aula 01 Treinamento</h1>

<div class="projeto">

    <div class="card">
        <h3>Cálculo de Aluno</h3>
        <input type="number" name="id" [(ngModel)]="aluno.id"
placeholder="Digite o Id" class="form" />
        <br />
        <input type="text" name="nome" [(ngModel)]="aluno.nome"
placeholder="Digite o Nome" class="form" />
        <br />
        <input type="text" name="disciplina"
[(ngModel)]="aluno.disciplina" placeholder="Digite a Disciplina"
class="form" />
        <br />
        <input type="number" name="nota1"
[(ngModel)]="aluno.nota1" placeholder="Digite a Nota1"
class="form" />
        <br />
        <input type="number" name="nota2"
[(ngModel)]="aluno.nota2" placeholder="Digite a Nota2"
class="form" />
        <br />
        <button (click)="calcMedia()" class="btn">Media</button>
        <button (click)="calcSituacao()"
class="btn">Situacao</button>
        <button (click)="calcFuncional()"
class="btn">Funcional</button>
    </div>
</div>

```

```

        <button (click)="calcFuncionalItem()"
class="btn">Funcional Item</button>
        <br><br>
        <b>Aluno:</b> {{aluno | json}}
        <br>
        <!-- CHAMANDO DA CLASSE ALUNOCALCULO -->
        <b>Soma Todas as notas:</b> {{somatudo | json}}
        <br>
        <!-- CHAMANDO DA CLASSE ALUNOCALCULO -->
        <b>Mostra por aluno:</b> {{soma | json}}
        <br>

    </div>

</div>

```

---

## app.component.css

```

h1{
    margin-top: 0;
    margin-bottom: 10px;
    padding: 10px;
    background-color: #ccc;
    text-align: center;
}
.projeto{
    margin-top: 20px;
    display: flex;
    flex-direction: row;
    justify-content: center;
    align-items: center;
}
.card{
    border: 1px solid #ccc;
    border-radius: 10px;
    box-shadow: 5px 5px 10px #aaa;
    padding: 10px;
    width: 350px;
    background: #fff;
}

```

```

}
h3{
    text-align: center;
}
.form{
    width: 100%;
    margin-bottom: 10px;
}
.btn{
    margin-right: 10px;
}

```

---

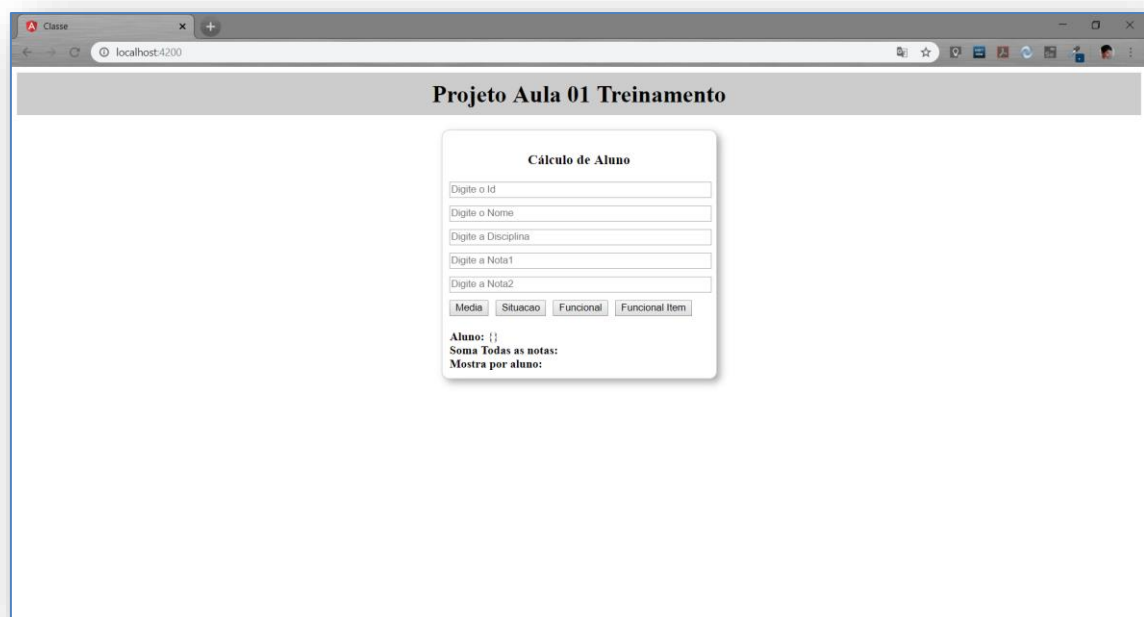
## Para rodar o projeto

Digitar no terminal:

“**ng s -o**” ou

“**ng s -o --port 1234**” para caso já tenha a porta padrão (4200) em uso.

<http://localhost:4200/>



Digitando os dados para o calculo e clicando nos botões

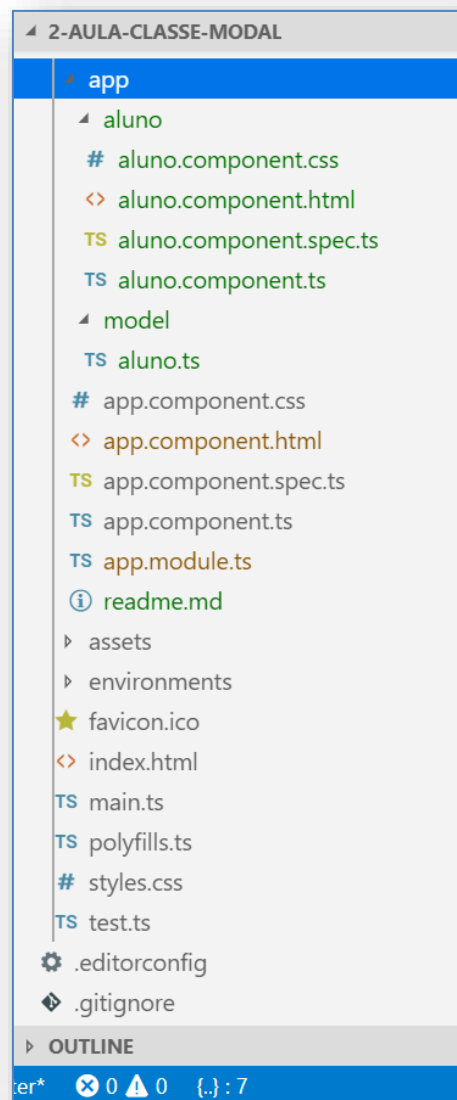
The screenshot shows a web browser window with the address bar set to 'localhost:4200'. The page title is 'Projeto Aula 01 Treinamento'. In the center, there is a form titled 'Cálculo de Aluno'. The form contains five input fields: 'id' (containing '3'), 'nome' (containing 'lu'), 'disciplina' (containing 'java'), 'nota1' (containing '9'), and 'nota2' (containing '8'). Below these fields are four buttons: 'Media', 'Situacao', 'Funcional', and 'Funcional Item'. The 'Funcional Item' button is highlighted. Below the buttons, the following JSON object is displayed: 

```
Aluno: { "id": 3, "nome": "lu", "disciplina": "java", "nota1": 9, "nota2": 8, "media": 8.5, "situacao": "aprovado" }
```

 Below the JSON, the text 'Soma Todas as notas: 32' is shown. At the bottom, the text 'Mostra por aluno: [ 17, 15 ]' is displayed.

## NOVO PROJETO MDB:

Estrutura do projeto depois de finalizado:



Criar o pacote "model"

Criar a classe "aluno.ts"

## aluno.ts

```
export class Aluno {

    nome: string;
    disciplina: string;
    nota1: number;
    nota2: number;
    media: number;
    situacao: string;

    //INICIAR AS VARIÁVEL NO CONSTRUTOR COLOCANDO COMO
    //OPCIONAIS NÃO INCLUIR MEDIA E SITUAÇÃO POIS SERÃO CALCULADAS
    //AUTOMATICAMENTE
    constructor(nome?: string, disciplina?: string, nota1?:
number, nota2?: number) {
        this.nome = nome;
        this.disciplina = disciplina;
        this.nota1 = nota1;
        this.nota2 = nota2;
    }

    //METODO DE GERAR MEDIA RETORNANDO ALUNO (:ALUNO)
    public gerarMedia(): Aluno {
        this.media = (this.nota1 + this.nota2) / 2;
        return this;
    }

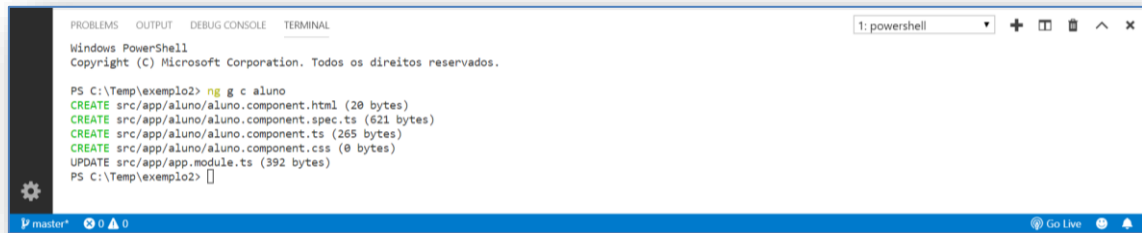
    //METODO DE GERAR A SITUAÇÃO
    //UMA OUTRA FORMA DE FAZER O METODO
    public gerarSituacao() {
        return this.situacao = (this.media >= 7 ? 'aprovado' :
'reprovado');
    }
}
```

---

## Criar o componente “aluno”

Digitar no terminal:

“ng g c aluno”



## aluno.component.ts

(Classe que programa os eventos da tela)

```
import { Component, OnInit, ViewChild, ElementRef } from
'@angular/core';
import { Aluno } from '../model/aluno';

@Component({
  selector: 'app-aluno', //NOME PELO QUAL A CLASSE É CHAMADA
  templateUrl: './aluno.component.html', //HTML CORRESPONDENTE
  DESSA CLASSE
  styleUrls: ['./aluno.component.css'] //CSS CORRESPONDENTE
})
export class AlunoComponent implements OnInit {

  //CRIANDO AS VARIÁVEIS
  aluno: Aluno;
  msg: string;
  msg2: string = "wait..."; //VARIÁVEL JÁ COM UM VALOR

  //NO CONSTRUTOR DAR ESPAÇO DE MEMÓRIA PARA ALUNO
  constructor() {
    this.aluno = new Aluno();
  }

  //MÉTODO INICIALIZADOR (NÃO USAMOS)
  ngOnInit() {

  }
}
```



```

//METODO DO BOTÃO PARA CALCULAR A SITUACAO
public calculaSituacao() {
    if (this.aluno.nome) { //SE O NOME FOR PREENCHIDO
        this.aluno.gerarMedia(); //CHAMANDO O METODO DA
CLASSE ALUNO
        this.aluno.gerarSituacao();
        //NO CONSOLE IRÁ MOSTRAR A MENSAGEM DE RESULTADO
        console.log("Resultado: " + this.aluno.disciplina +
" - " + this.aluno.situacao);
        this.msg = "dados ok"; //MOSTRA MSG CASO SEJA
PREENCHIDO O ALUNO
        this.msg2 = ""; //NAO MOSTRA A 2ª MSG
    } else { //SE O NOME NAO FOR PREENCHIDO
        alert('Nome vazio') //ABRE UM ALERTA COM A MSG DE
NOME VAZIO
        this.msg = ""; //NÃO MOSTRA A 1ª MSG
        this.msg2 = "Dados indefinidos"; //MOSTRA A MSG
    }
}

//ALERTA DO BOOTSTRAP COM JAVASCRIPT
@ViewChild('alert', { static: true }) alert: ElementRef;

//METODO DO BOTÃO PARA FECHAR O ALERTA
closeAlert() {
    this.alert.nativeElement.classList.remove('show');
}
}

```

---

## Material Design Bootstrap

O Material Design for Bootstrap é um kit de ferramentas de código aberto baseado no Bootstrap para desenvolver aplicativos de Material Design com HTML, CSS e JS. Crie rapidamente protótipos de suas ideias ou construa seu aplicativo inteiro com nossas variáveis e mixins Sass, sistema de grade responsivo, extensos componentes pré-construídos e poderosos plugins construídos em jQuery.

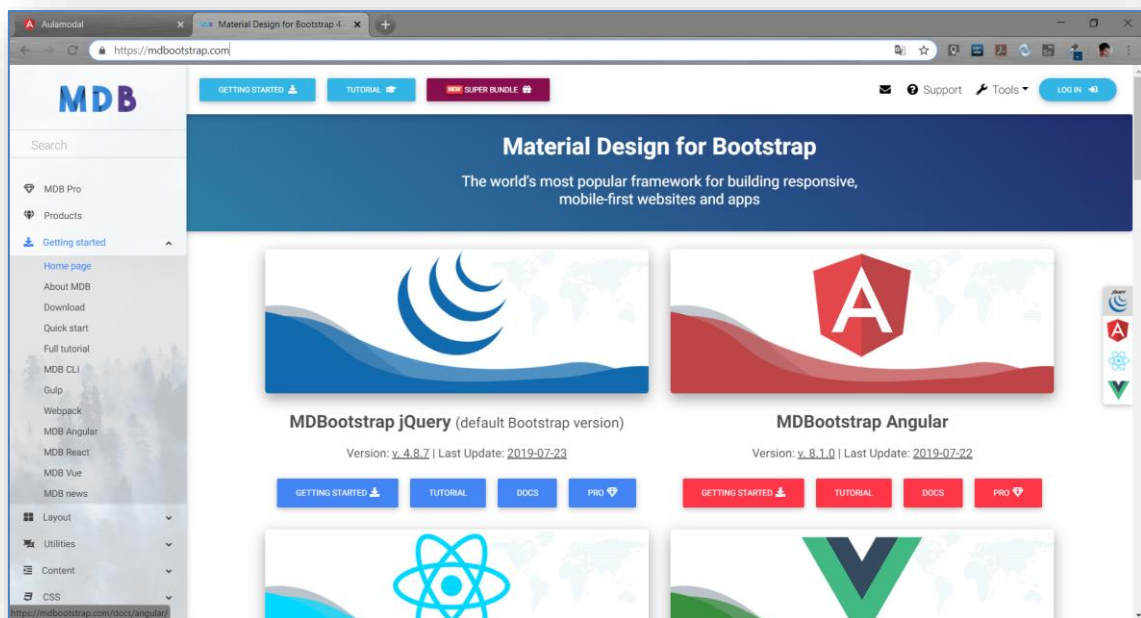
Desenvolvido em 2014 pelo Google, o **Material Design** é uma “**linguagem de design**” . Ele é baseado nos motivos de "cartão" que foram introduzidos pela primeira vez no Google Now e expandiu isso, incluindo transições e animações responsivas, efeitos como iluminação e sombras e layouts baseados em grade.

Sendo uma linguagem de design, o Material Design define um conjunto de diretrizes que mostram como melhor projetar um website. Ele informa quais botões são para uso e quais você deve usar, como animar ou movê-lo, bem como onde e como ele deve ser colocado, etc.

É uma linguagem de design para **dispositivos móveis** , que pode ser usada em todas as versões compatíveis do Android, proporcionando uma experiência consistente e de qualidade em todos os aplicativos e plataformas Android, que também podem ser expandidos para outras plataformas e aplicativos, principalmente por meio de APIs para desenvolvedores de terceiros. Além da facilidade de implementação e uso, outro grande recurso dessa plataforma de design é a **falta de dependência de estruturas e bibliotecas JavaScript** .

Em vez disso, ele se baseia em uma estrutura de design de materiais que facilita a utilização e fornece modelos personalizáveis e proporcionam muita liberdade criativa, ajudando a criar sites que se destacam.

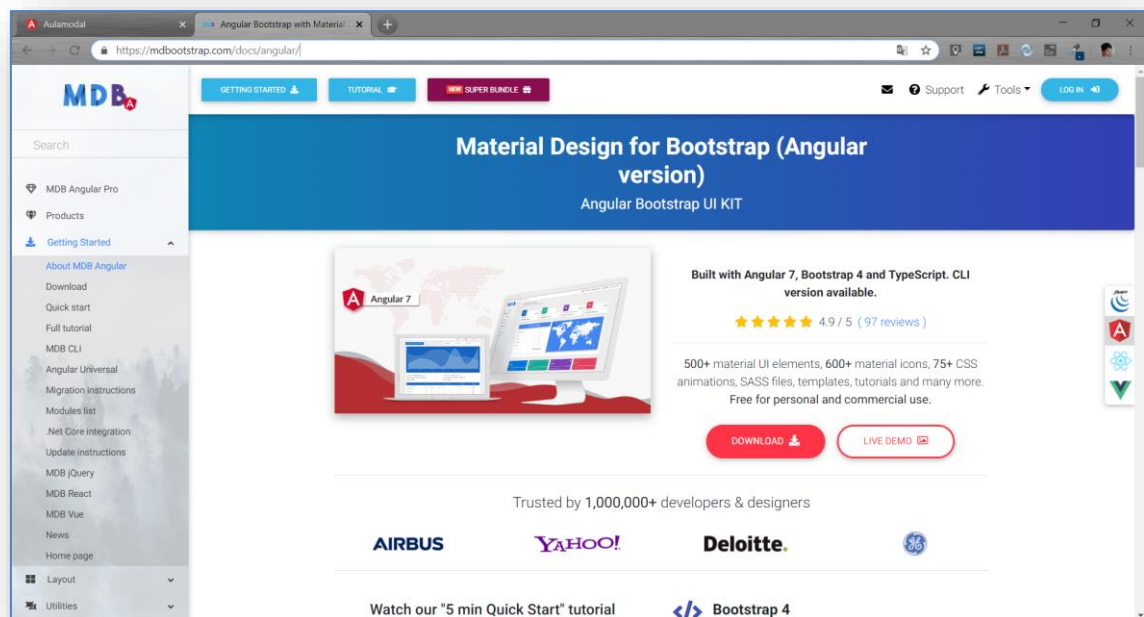
<https://mdbootstrap.com/>



Para angular:

Documentação oficial para estudar:

<https://mdbootstrap.com/docs/angular/>



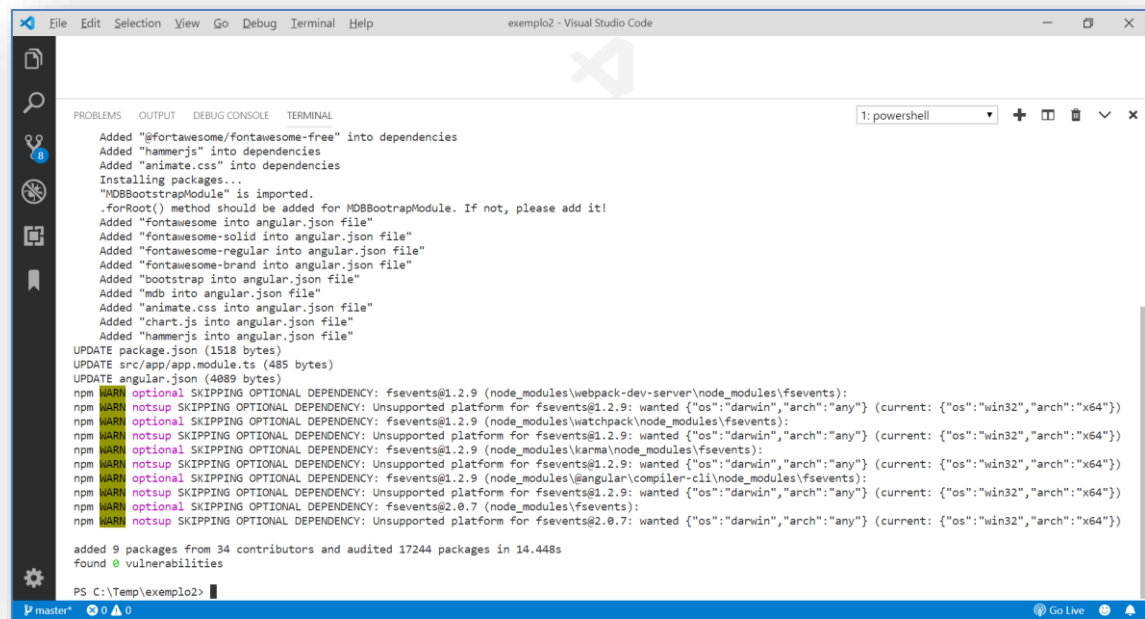
Para instalar no projeto.

Digitar no terminal:

**“npm i npm-registry-client”**



## “ng add angular-bootstrap-md”



```
File Edit Selection View Go Debug Terminal Help
exemplo2 - Visual Studio Code

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: powershell

Added "@fortawesome/fontawesome-free" into dependencies
Added "hammerjs" into dependencies
Added "animate.css" into dependencies
Installing packages...
"@fortawesome/fontawesome-free" is imported.
.forRoot() method should be added for MDBBootstrapModule. If not, please add it!
Added "fontawesome into angular.json file"
Added "fontawesome-solid into angular.json file"
Added "fontawesome-regular into angular.json file"
Added "fontawesome-brand into angular.json file"
Added "bootstrap into angular.json file"
Added "mdb into angular.json file"
Added "animate.css into angular.json file"
Added "chart.js into angular.json file"
Added "hammerjs into angular.json file"
UPDATE package.json (1518 bytes)
UPDATE src/app/app.module.ts (485 bytes)
UPDATE angular.json (4089 bytes)
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\webpack-dev-server\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\webpack\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\angular\compiler-cli\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\angular\compiler\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.9 (node_modules\angular\cli\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.9: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})
added 9 packages from 34 contributors and audited 17244 packages in 14.448s
found 0 vulnerabilities

PS C:\Temp\exemplo2>
```

## app.component.html

```
<!-- CONTAINER PARA O CONTEUDO DA TELA COM UMA MARGEM INTERNA
(PADDING DE 50PX) -->
```

```
<div class="container-fluid p-5">
```

```
<h3>Calculo Aluno com Modal</h3>
```

```
<!-- BOTÃO PARA ABRIR O MODAL, MDBBTN PARA INCORPORAR O
LAYOUT DO MDB, COR AZUL, OUTLINE PARA TER SOMENTE A BORDA DO
BOTÃO, CLICK PARA ACIONAR O MODAL, MDBWAVESEFFECT É O EFEITO DE
ONDA NO BOTÃO -->
```

```
<button type="button" mdbBtn color="primary" outline="true"
(click)="modal.show()" mdbWavesEffect>
```

```
Abrir Modal</button>
```

```
<!-- DIV DO MODAL, TRALHA COM A ATIVAÇÃO DO MODAL, CLASS COM
O ESTILO DE MODAL E EFEITO DE SURGIR NA TELA (FADE), CONFIRMA
QUE É UMA JANELA DE DIALOGO -->
```

```
<div mdbModal #modal="mdbModal" class="modal fade"
role="dialog">
```

```
<!-- ESTAMOS CONFIRMANDO QUE O ESTILO É DE UMA MODAL-
DIALOG -->
```

```
<div class="modal-dialog">
```

```

<!-- DEFININDO O CONTEUDO DA MODAL -->
<div class="modal-content">
  <!-- DEFININDO CABEÇALHO E BACKGROUND AZUL -->
  <div class="modal-header bg-info">
    <!-- BOTÃO DE FECHAR ALINHADO PELA DIREITA E
COM O CLICK DO BOTÃO PAR ESCONDER -->
    <button type="button" class="close pull-
left" (click)="modal.hide()">
      <!-- MOSTRAR O "X" PARA FECHAR -->
      <span>x</span>
    </button>
    <!-- TITULO -->
    <h4 class="modal-title">Calcule a Média</h4>
  </div>
  <!-- DEFININDO O CORPO DA MODAL -->
  <div class="modal-body">
    <!-- CAMPOS DE INPUT PARA O USUARIO DIGITAR,
MDBINOUT PARA DEFINIR O LAYOUT, NAME COM O NOME DO CAMPO,
[(ngModel)] PARA ACESSAR O DADO DA CLASSE, PLACEHOLDER PARA
MOSTRAR UMA MSG DENTRO DO CAMPO DE DIGITAÇÃO, CLASS DO FORM PARA
O LAYOUT, MB PARA MARGEM BOTTOM (INFERIOR)-->
    <input mdbInput type="text" name="nome"
[(ngModel)]="aluno.nome" placeholder="Nome"
class="form-control mb-3">
    <input mdbInput type="text"
name="disciplina" [(ngModel)]="aluno.disciplina"
placeholder="Disciplina" class="form-
control mb-3">
    <input mdbInput type="number" name="nota1"
[(ngModel)]="aluno.nota1" placeholder="Nota1"
class="form-control mb-3">
    <input mdbInput type="text" name="nota2"
[(ngModel)]="aluno.nota2" placeholder="Nota2"
class="form-control mb-3">

    <!-- BOTÃO COM O CLICK DO METODO, MDBBTN
PARA DEFINIR QUE É UM BOTÃO DO MDB, COR AZUL, EFEITO DE ONDA -->
    <button (click)="calculaSituacao()" mdbBtn
color="primary" mdbWavesEffect>Calcula</button>
  </div>
  <!-- DEFININDO O RODAPE DO MODAL -->
  <div class="modal-footer">

```

```

        <!-- DEFININDO O ALERTA COM A MSG, COR AZUL,
ALERTA QUE PODE DESAPARECER, EFEITO DE SURGIR, TIPO ALERT -->
        <div #alert class="alert alert-info alert-
dismissible fade show" role="alert">
            <!-- BOTÃO DE FCHAR O ALERTA, ESTILO
CLOSE DO MDB, CLICK DO METODO DE FECHAR -->
            <button type="button" class="close"
(click)="closeAlert()">
                <!-- "X" PARA O CLICK -->
                <span>&times;</span>
            </button>
            <!-- MENSAGEM DE DENTRO DO ALERTA -->
            <strong>Aluno Json : {{aluno |
json}}</strong>
        </div>
        <!-- MENSAGEM DEFINIDA NA CLASSE, JA ABRE A
JANELA MOSTRANDO WAIT... -->
        <div id="resposta1" *ngIf="aluno.nome">
            {{msg}}
        </div>
        <!-- OUTRA MSG -->
        <div id="resposta2" *ngIf="!aluno.nome">
            {{msg2}}
        </div>
    </div>
</div>
</div>
</div>
</div>
</div>

```

---

## app.module.ts

```

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { AlunoComponent } from './aluno/aluno.component';
import { FormsModule } from '@angular/forms';
import { MDBBootstrapModule } from 'angular-bootstrap-md';

```

```

@NgModule({
  declarations: [
    AppComponent,
    AlunoComponent //COMPONENTE IMPORTADO
  ],
  imports: [
    BrowserModule,
    FormsModule, //HABILITANDO O USO DO FORMULARIO
    MDBBootstrapModule.forRoot() //HABILITANDO O MATERIAL
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

---

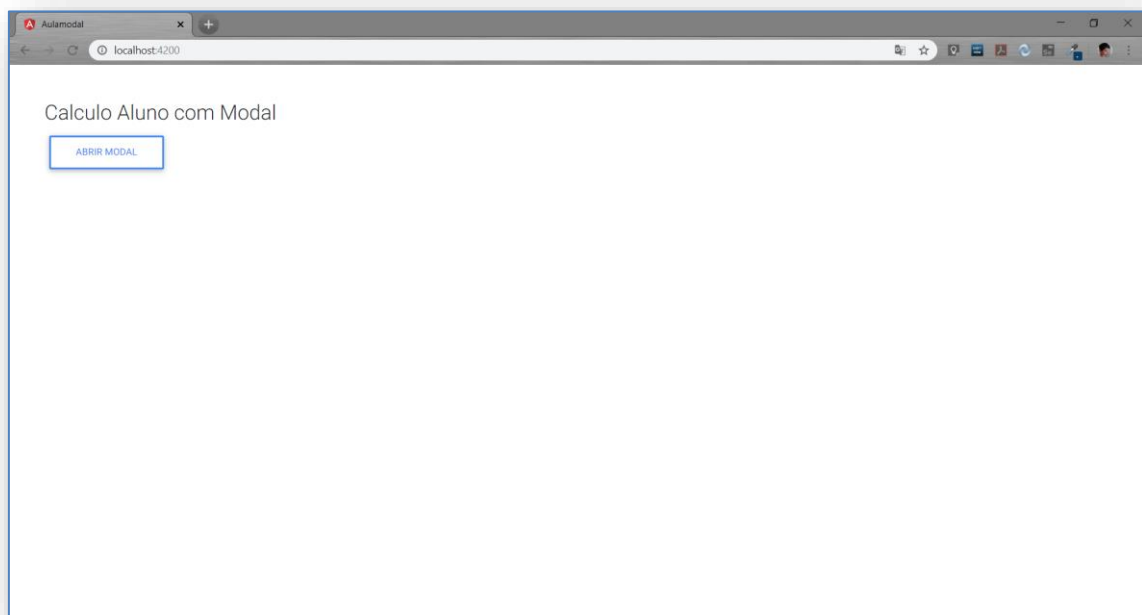
## Para rodar o projeto

Digitar no terminal:

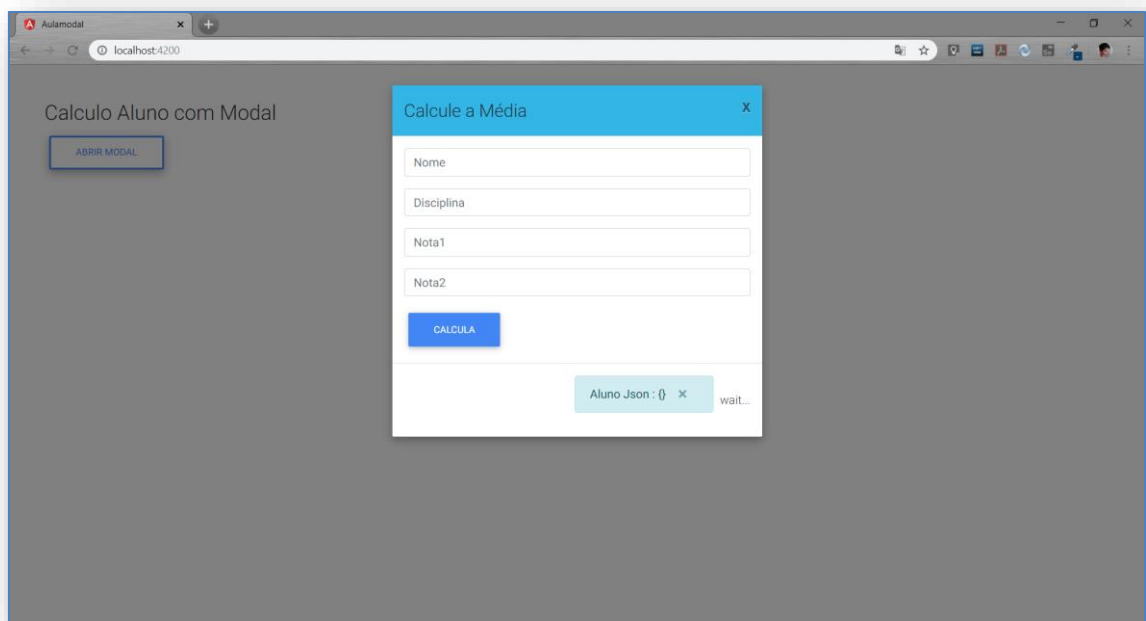
“ng s -o” ou

“ng s -o --port 1234” para caso já tenha a porta padrão (4200) em uso.

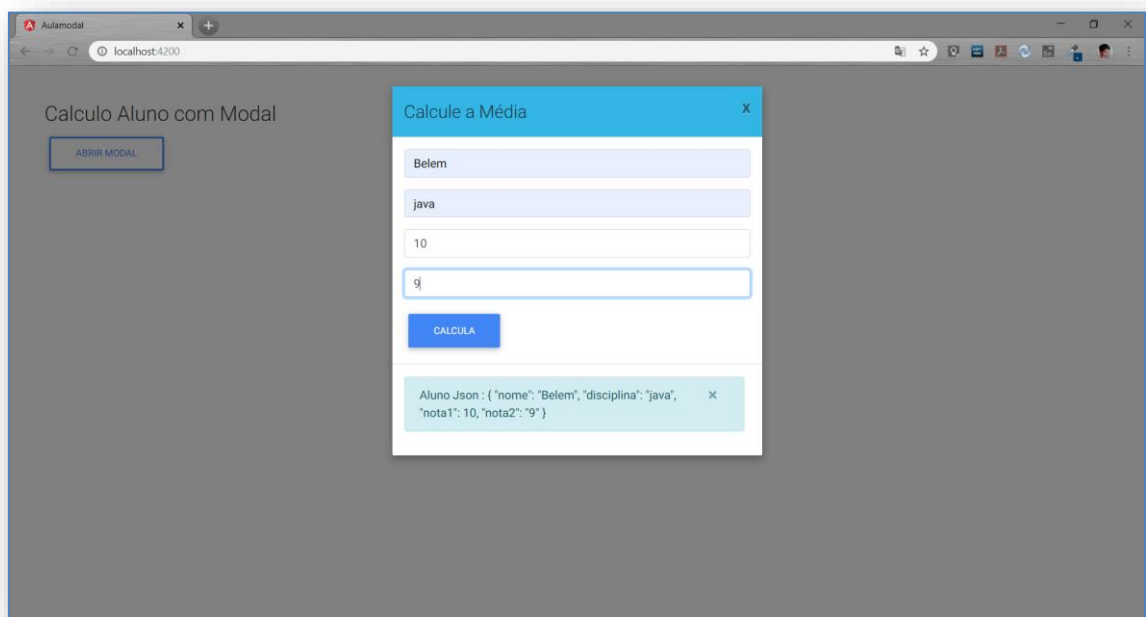
<http://localhost:4200/>



Clicando no botão

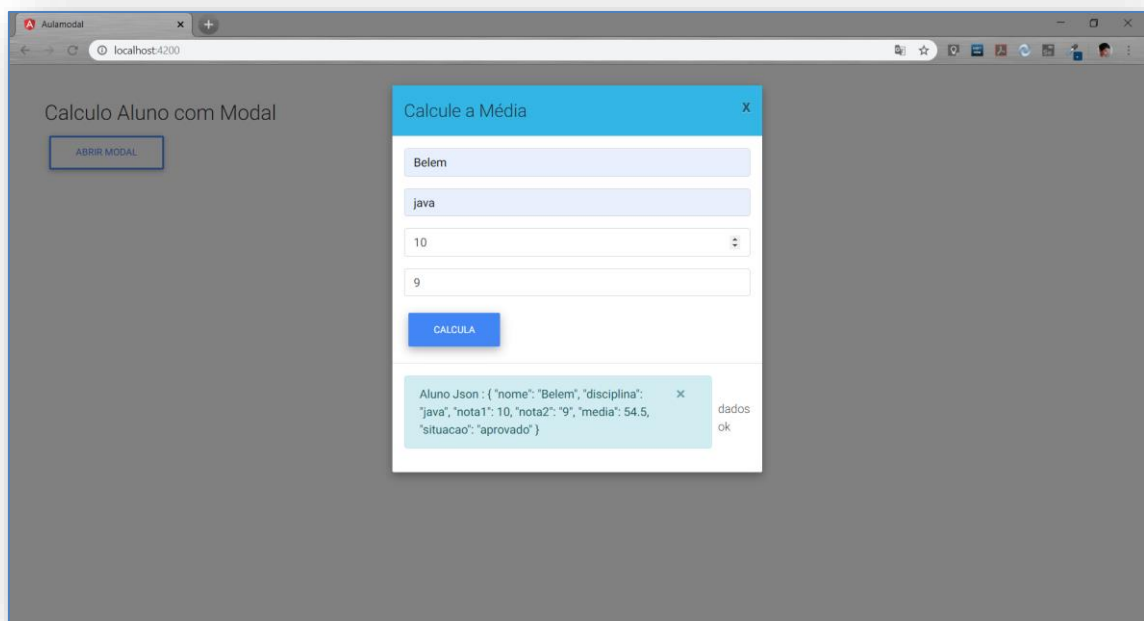


Preenchendo os campos

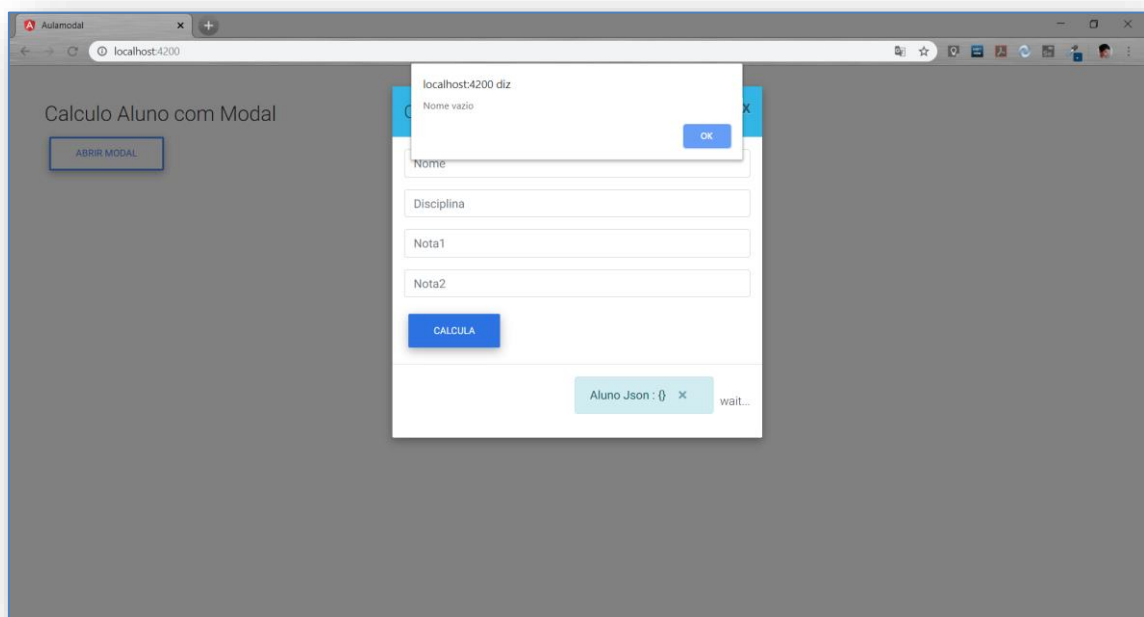




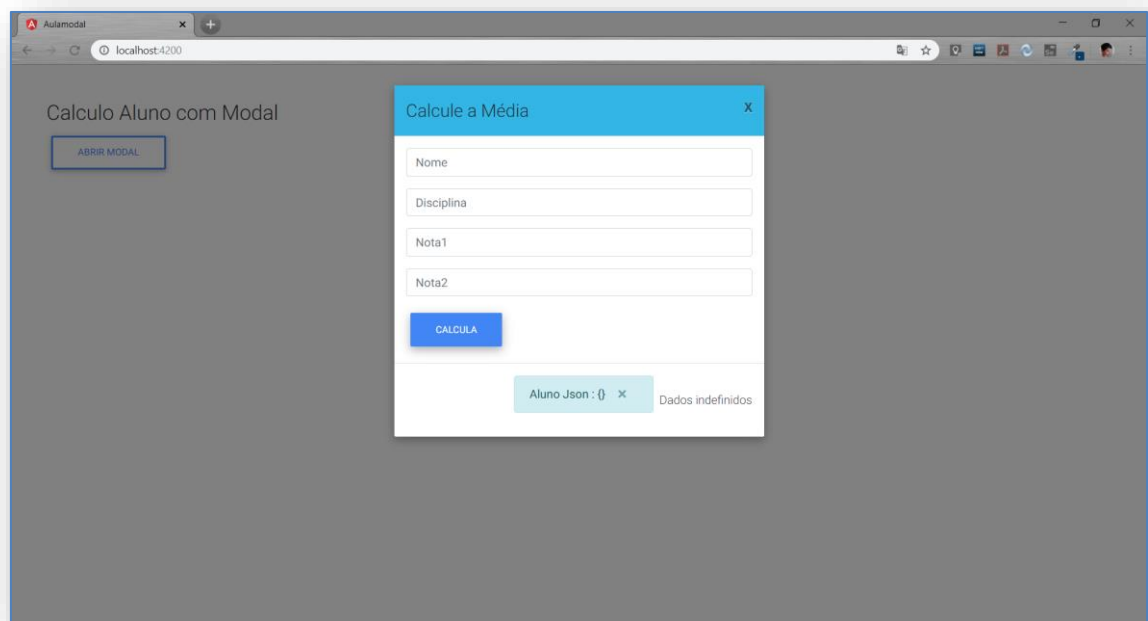
Clicando no botão



Se não preencher as informações e clicar no botão mostra o alerta



E a msg de dados indefinidos



## Módulo Programação Funcional

### model/pessoa.ts

```
export class Pessoa{
  id : number;
  nome : string;
  idade : number;

  constructor(id ? : number, nome ? : string, idade ? : number){
    this.id = id;
    this.nome = nome;
    this.idade = idade;
  }

  public dados(): string{
    return this.id + "," + this.nome + "," + this.idade;
  }
}
```

## peessoa-controll.ts

```
export const dados=[
  {
    id:10,
    nome:"biel",
    idade:20
  },
  {
    id:11,
    nome:"bel",
    idade:15
  },{
    id:12,
    nome:"belem",
    idade:40
  }
];

export const pessoaMaisvelho= dados.reduce( (a,b) =>a.idade > b.idade? a: b );

export const pessoaMaisNovo= dados.reduce( (a,b) =>a.idade < b.idade? a: b );

export const pessoaSomaldade= dados.reduce( function(total,dados){return total +
dados.idade},0);

export const pessoaQua= dados.reduce( function(conta){return conta + 1},0);

export const pessoaMedia = pessoaSomaldade / pessoaQua;

export const ordena = dados.sort((a,b)=>{return a.idade - b.idade});
```

---

## app.component.ts

```
import { Component } from '@angular/core';
import { pessoaSomaldade, ordena, pessoaMaisvelho, pessoaMaisNovo, pessoaQua,
pessoaMedia } from './controll/controll-pessoa';
```

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'funcional2';

  public totais(){
    console.log('soma', pessoaSomaldade);
    console.log('Ordenar', ordena);
    console.log('mais velho', pessoaMaisvelho);
    console.log('pessoaMaisNovo',pessoaMaisNovo );
    console.log('Quantidade',pessoaQua);
    console.log('Media',pessoaMedia)
  }
}
```

app.component.ts

```
<h2>Funcional</h2>
```

```
<button (click)="totais();">Totais</button>
```

---

Rodando o projeto:

No Terminal

**“ng s -o - -port 1234”**

---

## Resultado

Angular is running in the development mode. Call enableProdMode() to enable the production mode.

soma 75

Ordenar (3) [{...}, {...}, {...}]0: {id: 11, nome: "bel", idade: 15}1: {id: 10, nome: "biel", idade: 20}2: {id: 12, nome: "belem", idade: 40}length: 3\_\_proto\_\_: Array(0)

mais velho {id: 12, nome: "belem", idade: 40}id: 12idade: 40nome: "belem"\_\_proto\_\_: Object

pessoaMaisNovo {id: 11, nome: "bel", idade: 15}

Quantidade 3

pessoaMaisNovo {id: 11, nome: "bel", idade: 15}

id: 11idade: 15nome: "bel"\_\_proto\_\_: Objectconstructor: f Object()hasOwnProperty: f

Media 25

## Exercícios

- 1) O que é uma Arquitetura MVC ?
- 2) O que é o Angular ?
- 3) O que é um Model ?
- 4) Qual a Diferença entre Js e ts ?
- 5) Qual o comando do angular para :
  - Criar um projeto
  - Executar um Projeto
- 6) Faça uma classe Cliente em typescript com id, nome e idade .
- 7) Faça uma Interface de Cliente para mostrar a maior idade
- 8) Implemente a Lógica para mostrar a Cliente com a maior idade dentro de um vetor de 3 posições ?
- 9) Faça o HTML para mostrar ao digitar os dados no Formulário, imprimir os dados de Pessoa em Json.
- 10) Qual a Diferença entre `componente cliente.component.ts`, `cliente.component.html`, `cliente.component.css`, `cliente.ts`