**TypeScript parte I – Evoluindo seu JavaScript**

1. **Aula 1 – Porque usar TypeScript:**
   1. A primeira coisa quando vamos codar em typescript é fazer a preparação do ambiente, ou seja, instalar node.js e depois dar um npm install no terminal dentro da pasta do projeto para instalar as dependências do server para rodar nossa aplicação.
      1. Npm run server: Starta o server quando rodar o comando no terminal dentro da pasta do projeto.
      2. No meu caso estou usando a extensão live server do VScode pois não funcionou de jeito nenhum.
   2. Negociação, modelagem e regras:
      1. #: variável privada, não se altera.
      2. Criamos uma classe de negociação para nossa aplicação onde possui as variáveis de data, quantidade e valor privadas com um # antes delas e um construtor que atribui esses valores ao ser criado.
      3. Teoricamente não poderíamos conseguir alterar os valores dessas variáveis diretamente, então fizemos o teste com uma atribuição direta fora do construtor.
      4. Ao fazer um console.log do antes e depois dessa atribuição direta, notamos que ao invés de alterar o valor da variável privada, ele criou uma nova variável com o valor novo atribuído:

// Módulo da negociação

export class Negociacao {

  #data;

  #quantidade;

  #valor;

  constructor(data, quantidade, valor) {

    this.#data = data;

    this.#quantidade = quantidade;

    this.#valor = valor;

  }

}

// Módulo do APP

import { Negociacao } from "./models/negociacao.js";

const negociacao = new Negociacao(new Date(), 10, 100);

console.log(negociacao)

negociacao.quantidade = 1000

console.log(negociacao)



* + 1. O módulo do app é o que está vinculado a nossa aplicação web.
  1. Finalização do modelo:
     1. Lembrando que se quiser visualizar uma variável privada, não conseguimos mesmo que tentemos colocar o # antes dela num console.log:

console.log(negociacao.#data)



* + 1. Se tentarmos visualizar sem o # ele dá como Undefined.
    2. Portanto precisamos criar getters para essas propriedades para que possamos visualizar elas, embora ainda não poderemos fazer alterações diretamente utilizando elas, pois são getters e não setters.
  1. Motivação do TypeScript:
     1. Quando utilizamos JS só vamos descobrir que cometemos um erro no código em run time, ou seja, em tempo de execução dele, seja já na produção ou então no ambiente de testes, tendo que fazer o código voltar para revisar o que deu errado.
     2. O TS trás tudo o que o JS possuí e ainda muito mais, fazendo com que esse tipo de erro não aconteça. O tipo de coding é igual, mas com muito mais ferramentas pois ela é um super do ECMA desenvolvido pela Microsoft. Uma das funcionalidades incríveis é que conseguimos identificar e pegar esses erros em tempo de desenvolvimento, adiantando muito caso algo dê errado.
  2. O que aprendemos:
     1. Introdução ao projeto e sua estrutura;
     2. Um pouco sobre módulos do ECMASCRIPT;
     3. Modelagem de uma Negociação em Javascript;
     4. Buracos em nossa modelagem por limitações da linguagem Javascript.

1. **Aula 2 – TypeScript e Compilador:**
   1. Instalando o TypeScript:
      1. npm install typescript --save-dev: Instalação da versão mais recente do TS via terminal no VScode.
      2. npm install typescript@n.n.n --save-dev: Instalação de uma versão específica do TS via terminal no VScode. Substituir ‘n’ pelos números da versão que quiser.
   2. Arquivos TS:
      1. A extensão de arquivos com código em TS é justamente essa: .ts.
      2. Simplesmente ao alterar a extensão do arquivo, antes mesmo de instalar o compilador e fazer a configuração do TS, ele já começa a nos mostrar os erros do nosso código e, ao colocar o mouse em cima do erro, ele exibe o motivo de estar errado.
      3. Se tentarmos carregar esse arquivo no navegador ele não irá reconhecer, pois o navegador nem sabe o que é o TS.
      4. Por causa disso temos o compilador e uma outra pasta chamada app. Na pasta chamada app nós deixamos tudo da nossa aplicação que será escrito em TS e, na pasta dist, onde estavam nossos arquivos que antes eram JS, ficará tudo o que o navegador consegue ler.
      5. O compilador serve justamente para converter tudo o que vamos escrever em TS para JS e jogar automaticamente nas pastas correspondentes dentro da pasta dist.
   3. Configuração básica do compilador: