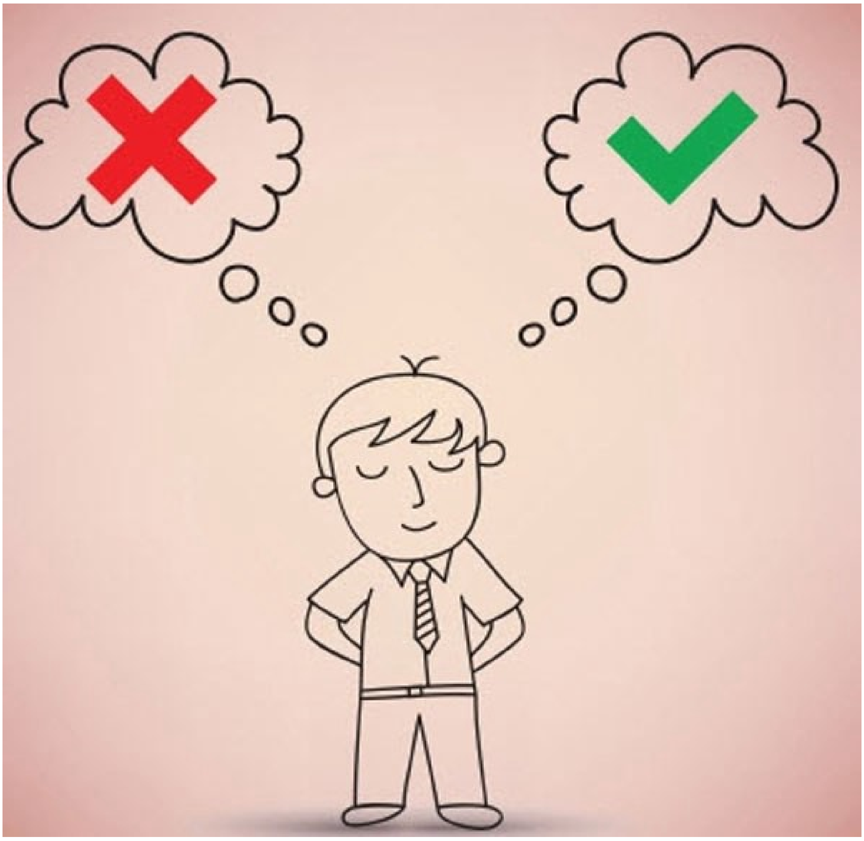
How (and why) you should use TypeScript with Node and Express.

During the beginning of my career, I was a desktop developer, where strong typed languages dominate the market.

When I migrated to web development, I was fascinated by every new feature on languages like JavaScript and Python. The fact that I didn't have to declare the type of variables brought me productivity and my work became more fun.

So the first time I heard about TypeScript, the idea seemed like a step backwards in the evolution of the language.



Did I change my mind?

Yes, but it depends on the case. For personal projects, in which I work alone, I still prefer the productivity of plain JavaScript. But, for larger projects, where you work as a team, I recommend using TypeScript. Throughout this article I will explain how and why.

TypeScript

If you still do not know what TypeScript is, I recommend reading this overview: https://www.tutorialspoint.com/typescript/typescript_overview.htm
And the official 5-minute tutorial: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

But we can say, superficially, that it is ECMAScript 6 with strong typing.

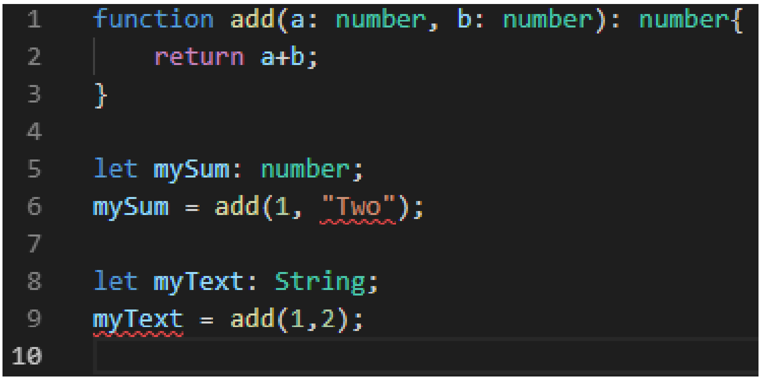
Productivity vs maintenance

By definition, “TypeScript is JavaScript for application-scale development”. That is, the work we have with the initial setup of the project is compensated with the maintainability of complex projects. Let’s point out why this happens:

Type safe = less errors

By defining types in your code, you allow the IDE to acknowledge errors in the use of classes and functions that would only be perceived at runtime.

Example:



```
function add(a: number, b: number): number{
    return a+b;
}

let mySum: number;
mySum = add(1, "Two");

let myText: String;
myText = add(1,2);

```

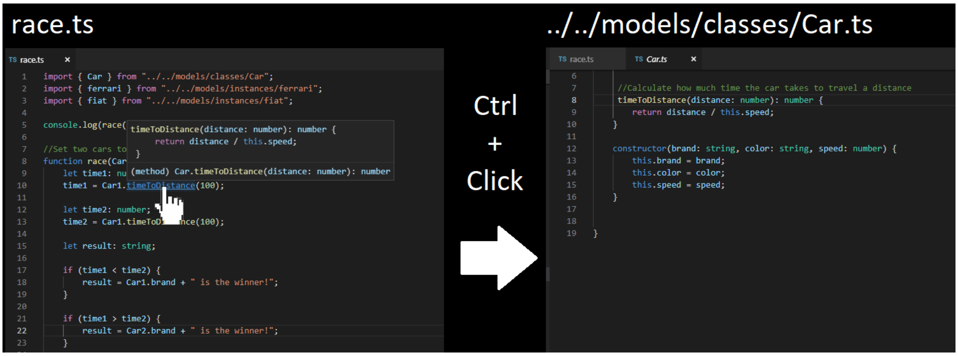
Here I am using Visual Studio Code which points put two errors:

On line 6: we are trying to pass a string parameter to a function that only accepts numbers.
On line 9: we are trying to assign, to a string, the result of a function that returns a number.
These two errors would go unnoticed, without Typescript, resulting in some bug on the final application.

The IDE exposes the project modules more easily

In complex projects we have hundreds of classes distributed across multiple files. When we define types, the IDE is able to relate objects and functions to the files that gave origin to them.

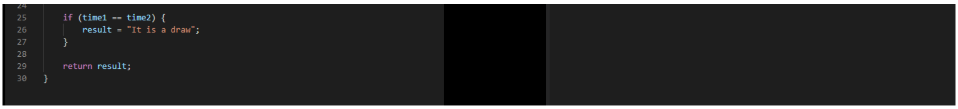
When using control + click on a method or class, imported from another file, the IDE will automatically navigate to the imported file, highlighting the line where the reference was defined.



```
import { Car } from "../../models/classes/Car";
import { ferrari } from "../../models/instances/ferrari";
import { fiat } from "../../models/instances/fiat";

console.log(race(

//Set two cars to
function race(Car
    let time1: nu
    time1 = Car1.timeToDistance(100);

    let time2: number;
    time2 = Car1.timeToDi   nce(100);

    let result: string;

    if (time1 < time2) {
        result = Car1.brand + " is the winner!";
    }

    if (time1 > time2) {
        result = Car2.brand + " is the winner!";
    }
```

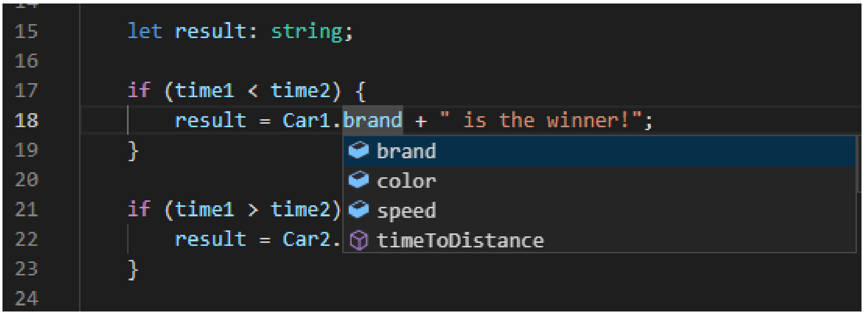
```
timeToDistance(distance: number): number {
     return distance / this.speed;
 }
(method) Car.timeToDistance(distance: number): number
```

```

        //Calculate how much time the car takes to travel a distance
        timeToDistance(distance: number): number {
            return distance / this.speed;
        }

        constructor(brand: string, color: string, speed: number) {
            this.brand = brand;
            this.color = color;
            this.speed = speed;
        }


}
```



```
    if (time1 == time2) {
        result = "It is a draw";
    }

    return result;
}
```

We can use autocomplete in classes that have been imported from other files.



```
    let result: string;

    if (time1 < time2) {
        result = Car1.brand + " is the winner!";
    }

    if (time1 > time2) {
        result = Car2.
    }

```

The difficulty of maintenance is one of the main reasons why Java and C # developers avoid migrating large projects to JS. We can say that Typescript is an enterprise language that overcomes this barrier.

How to setup an Express project with Typescript

Let’s now take a step-by-step to create an environment for using the Typescript language within an Express.js project.

```
npm init
```

Let’s now install the `typescript` package.

```
npm install typescript -s
```

About the Typescript node package

Node.js is an engine that runs Javascript and not Typescript. The node Typescript package allows you to transpile your `.ts` files to `.js` scripts. Babel can also be used to transpile Typescript, however the market standard is to use the official Microsoft package.

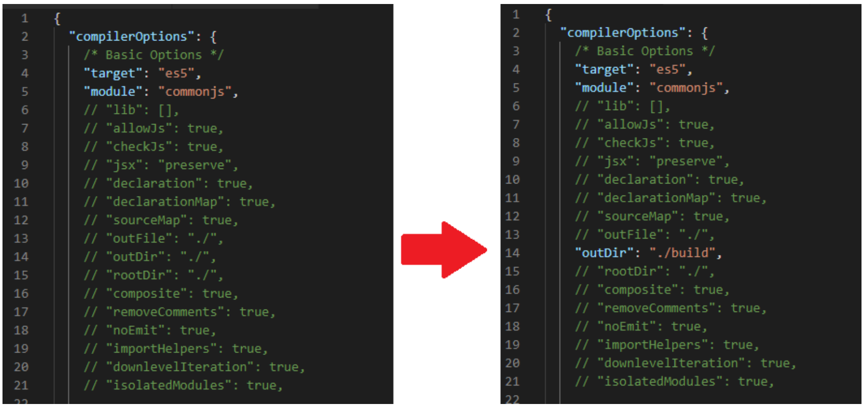
Inside our `package.json` we will put a script called `tsc`:

```
"scripts": {
    "tsc": "tsc"
},
```

This modification allows us to call typescript functions from the command line in the project’s folder. So we can use the following command:

```
npm run tsc -- --init
```

This command initializes the typescript project by creating the `tsconfig.json` file. Within this file we will uncomment the `outDir` option and choose a location for the transpiled .js files to be delivered:



```
{
  "compilerOptions": {
    /* Basic Options */
    "target": "es5",
    "module": "commonjs",
    // "lib": [],
    // "allowJs": true,
    // "checkJs": true,
    // "jsx": "preserve",
    // "declaration": true,
    // "declarationMap": true,
    // "sourceMap": true,
    // "outFile": "./",
    // "outDir": "./",
    // "rootDir": "./",
    // "composite": true,
    // "removeComments": true,
    // "noEmit": true,
    // "importHelpers": true,
    // "downlevelIteration": true,
    // "isolatedModules": true,
```

```
{
  "compilerOptions": {
    /* Basic Options */
    "target": "es5",
    "module": "commonjs",
    // "lib": [],
    // "allowJs": true,
    // "checkJs": true,
    // "jsx": "preserve",
    // "declaration": true,
    // "declarationMap": true,
    // "sourceMap": true,
    // "outFile": "./",
    "outDir": "./build",
    // "rootDir": "./",
    // "composite": true,
    // "removeComments": true,
    // "noEmit": true,
    // "importHelpers": true,
    // "downlevelIteration": true,
    // "isolatedModules": true,
```

Installing express.js

```
npm install express -s
```

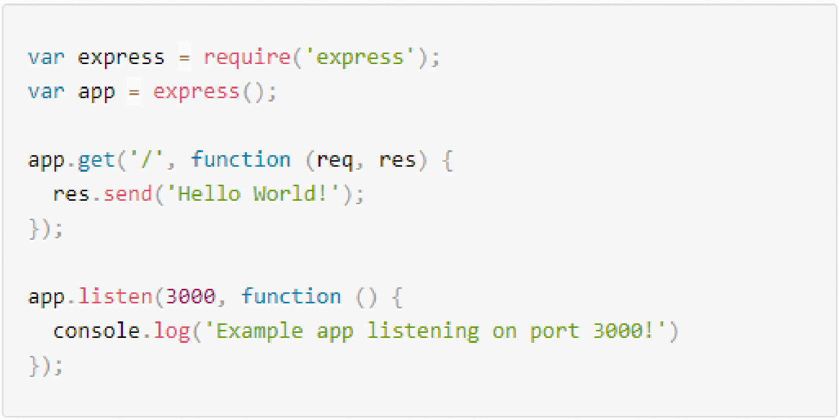
Express and Typescript packages are independent. The consequence of this is that Typescript does not “know” types of Express classes. There is a specific npm package for the Typescript to recognize the Express types.

```
npm install @types/express -s
```

Hello world

To have the simplest application as possible, I'll use the hello world example of the express.js tutorial:

https://expressjs.com/pt-br/starter/hello-world.html



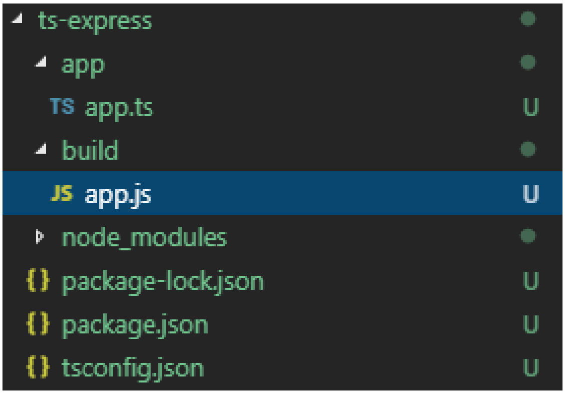
```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello World!');
});

app.listen(3000, function () {
  console.log('Example app listening on port 3000!')
});
```

Inside our project, we will create a folder called `app`. Inside this folder we will create a file called `app.ts` with the following content:

Compiling our first application:

```
npm run tsc
```

As you can see, the command automatically created the build folder and the .js file.



Running the express:

```
node build/app.js
```

With that, we have a server already running on port 3000:

Hello World!

Running TypeScript without transpiling

You can run typescript directly on the node with the `ts-node` package.

This package is recommended for development only. To make the final deploy in production, always use the javascript version of your project.

The `ts-node` is already included as a dependency on another package, t`ts-node-dev`. After installing, `ts-node-dev` we can run commands that restarts the server whenever a project file changes.

```
npm install ts-node-dev -s
```

Inside our `packege.json` we will add two more scripts:

```
"scripts": {
    "tsc": "tsc",
    "dev": "ts-node-dev --respawn --
transpileOnly ./app/app.ts",
    "prod": "tsc && node ./build/app.js"
},
```

To start the development environment:

```
npm run dev
```

To run the server in production mode:

```
npm run prod
```