**Node**

O node surgiu como uma forma de solucionar os problemas que as aplicações single-trhead tinham como Java, Python, C# etc.

**Instalação:**

Para instalar o node acessamos o site <http://nodejs.org> baixamos e instalamos, para vermos a versão instalada usamos o comando **npm -v** no terminal.

**Gerenciador de módulos com NPM:**

O node possui seu próprio gerenciador de pacotes, ele se chama NPM (Node Package Module), alguns comandos do npm:

**Npm install nome\_do\_modulo:** Instala um modulo no projeto.

**Npm install -g nome\_do\_Modulo:** Instala um modulo global.

**Npm install nome\_do\_modulo –save:** Instala o módulo no projeto, atualizando o package.json na lista de dependências.

**Npm list:** Lista todos os módulos do projeto.

**Npm list -g:** Lista todos os módulos globais.

**Npm remove nome\_do\_modulo:** Desinstala um módulo do projeto.

**npm remove -g nome\_do\_módulo**: Desinstala um módulo global.

**Npm update nome\_do\_modulo:** atualiza a versão do modulo.

**Npm update -g nome\_do\_modulo:** atualiza a versão do modulo global.

**npm -v**: exibe a versão atual do npm.

**npm adduser nome\_do\_usuário**: cria uma conta no npm, através do site

(<https://npmjs.org>).

**npm whoami**: exibe detalhes do seu perfil público npm (é necessário criar

uma conta antes).

**npm publish**: publica um módulo no site do npm, é necessário ter uma

conta antes.

**Criando uma api**

**Iniciando o projeto:**

Para iniciarmos o projeto vamos usar typeScript por isso, devemos criar o arquivo **tsconfig.json** através do comando **tsc –init:**

Com o arquivo criado vamos adicionar algumas configurações:

{

"compilerOptions": {

"target": "es6",

"module": "commonjs",

"esModuleInterop": true,

"outDir": "dist",

"typeRoots": [

"../node\_modules/@types"

],

"types": [

"node"

]

},

"include": [

"\*\*/\*.ts",

"\*.ts",

],

"exclude": [

"node\_modules"

]

}

O que alteramos nele? Dentro de compilerOptions nós passamos:

**target**: versão do ECMAScript que vamos utilizar no projeto.

**module**: biblioteca que vamos utilizar para trabalhar com os

módulos.

**esModuleInterop**: permite importarmos os módulos CommonJS em

conformidade com as especificações do target es6.

**outDir**: diretório de destino onde serão transpilados os nossos

arquivos.

**typeRoots**: caminho dos pacotes @types que nós importaremos

para trabalhar com TypeScript; veremos com mais detalhes no

decorrer do capítulo.

Fora de compilerOptions, temos:

**include**: local onde o compilador deve procurar os nossos

arquivos para fazer o transpile;

**exclude**: os diretórios que devem ser ignorados no momento do

transpile.

Após configurarmos o typeScript do projeto, vamos criar o nosso **package.json** através do comando **npm init -y**, após isso o vamos instalar as bibliotecas necessárias para usarmos o typeScript:

npm i typescript express --save

npm i @types/express --save-dev

É importante entender que temos que instalar a biblioteca necessária para usarmos o banco de dados que queremos, além dos types também, exemplo:

npm i mysql2

npm i @types/mysql2 --save-dev

Vamos também instalar a biblioteca **sequelize** que nos permite integrar com qualquer tipo de banco de dados existente:

Npm i sequelize nomeDoBanco

npm install sequelize-cli -d

Após isso vamos adicionar os seguintes scripts no package.json:

"scripts": {

"dev": "echo \"Error: no test specified\" && exit 1",

"compile": "tsc -w",

"start": "node ./dist/program.js"

},

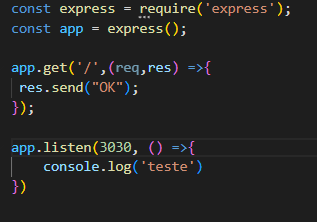
Nesse código, temos duas instruções:

**compile**: esse parâmetro já foi visto na introdução do livro. Ao adicionar o -w depois da instrução tsc, o compilador passa a monitorar os arquivos .ts do nosso projeto e, em caso de alteração, ele faz o transpile dos nossos arquivos.

**start**: aqui estamos passando qual será a instrução de inicialização da nossa API.

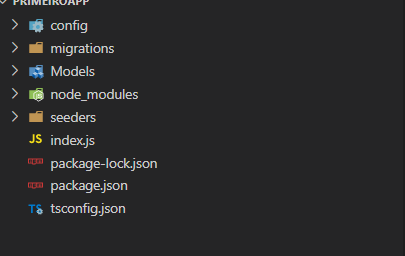
**Criando a estruturação do projeto:**

Primeiramente vamos criar o arquivo index.js na raiz do projeto e ele conterá as configurações das rotas:



Depois disso vamos criar a estrutura do projeto através do sequelize-cli:

npx sequelize-cli init



Nessa estrutura temos algumas pastas:

**Config:** Aqui terá o arquivo config.json, onde teremos a configuração do banco de dados:

