```
//ARQUIVOS PACKAGE JSON:

//Os arquivos package JSON são arquivos criados por através do Node com o objetivo de descrever um projeto criado em Node quanto também de atribuir alguns tipos de comportamento para o nosso projeto, como bibliotecas utilizadas, como controle de versionamento da biblioteca, palavras chave do projeto, quem é o autor do projeto, licensa e outras coisas do tipo.

//Outra grande vantagem dos arquivos package.json é a atribuição de dependências dentro do arquivo. Pois o nosso arquivo package.json irá acompanhar o nosso projeto para onde quer que ele for, assim quando vários desenvolvedores estão trabalhando num projeto que precisa de determinadas bibliotecas, o nosso package.json irá providenciar que essas bibliotecas sejam baixadas, e ainda melhor, podemos até escolher a versão de biblioteca, para evitar conflitos de versionamento caso um desenvolvedor baixe uma versão diferente da biblioteca que usamos atualmente no projeto.

//Existem formas diferentes de se criar um arquivo package JSON:

//CRIANDO ARQUIVO DE FORMA CONVENCIONAL:
//Para criar um arquivo package.json convencional com todas as alternativas básicas, basta o comando abaixo...
//      npm init -y
//  Esse comando quer dizer que estamos criando um arquivo normal e respondendo yes a todas as perguntas do node sobre os atributos do arquivo package.json...




//CRIANDO ARQUIVO PACKAGE.JSON DETALHADAMENTE:
//Podemos gerar um package.json colocando detalhadamente tudo o que queremos no arquivo, nesse método o Node pára a execução da criação do arquivo em partes específicas para nos perguntar coisas como nome do arquivo, versão, descrição, arquivo de entrada do projeto, teste de comando, repositório no git, palavras chave para chamar pelo arquivo, autor e licensa.
//Para gerarmos uma criação de package.json detalhada usamos o comando:
//      npm init




//INCLUÍNDO DEPENDÊNCIA A UM ARQUIVO PACKAGE.JSON:
//Para que possamos incluir uma biblioteca dentro do arquivo package.json podemos usar o comando --save quando formos baixar uma determinada biblioteca.
```

```
//OBS: Sempre certifique-se de que o arquivo package.json já existe dentro do seu projeto, e certifique-se de que o path esteja encaminhado para a biblioteca correta.
//Podemos incluir uma biblioteca assim:
//      npm i --save bibliotecaTal
//  Depois de fazer o download da biblioteca, perceba que a biblioteca foi inclusa em "dependences" no arquivo "package.json"



//BAIXANDO DEPENDÊNCIAS DO PACKAGE JSON EM UM NOVO ARQUIVO:
//Para que as dependências que salvamos no arquivo JSON sejam baixadas em num novo arquivo, basta para isso copiarmos ou baixarmos o nosso arquivo JSON á partir de um repositório dentro da nossa pasta de trabalho atual. E no terminal do Node usamos o comando:
//              npm i
//      Que significa "npm install", o Node irá procurar pelo arquivo "package.json" e irá automáticamente criar uma pasta "node-modules" contendo todas as bibliotecas que precisam ser baixadas no nosso projeto.



//INCLUINDO DEPENDENCIA SOMENTE EM PERÍODO DE DESENVOLVIMENTO:
//Podemos optar por incluir uma dependência que será utilizada somente durante o período de desenvolvimento do sistema, por exemplo dependências de testes, para isso podemos usar o comando "--save-dev", ele indica que após o desenvolvimento, quando o programa estiver rodando normal, essa dependência não será mais necessária.
//Usamos esse comando assim:
//      npm i --save-dev bibliotecaTal



//CONTROLE DE VERSIONAMENTO SEMÂNTICO DAS BIBLIOTECAS DO NODE:
//Quando baixamos uma biblioteca Node externa no nosso projeto e ela é inclusa nas nossas dependências, podemos observar que por padrão nossas bibliotecas estão configuradas apara aceitar atualizações de versionamentos. Porém existem 3 tipos de versionamento diferentes nas bibliotecas:
/*      * MAJOR: Quando as mudanças geram incompatibilidade entre a versão anterior e superior;
        * MINOR: Quando uma biblioteca teve melhorias, mas a compatibilidade entre versão anterior e superior são mantidas;
        * PATCH (ou FIX): Quando uma biblioteca teve correções, mas a compatibilidade entre versão anterior e superior são
                mantidas;
```

```
    Por padrão os números seguem a sequencia: MAJOR.MINOR.PATCH apresentada em números: 0.0.0
 */
//VERSIONAMENTO NO MINOR:
//Por padrão, quando incluímos uma biblioteca como dependência no nosso package.json ela permitirá atualizações na MINOR, ou seja, permite que novas melhorias implementadas na biblioteca sejam baixadas. No package.json isso é simbolizado pelo símbolo de "^", por exemplo: "^0.22.0". Quando está assim, queremos dizer que só aceitamos atualizações de bibliotecas, não aceitamos atualizações de API MAJOR, onde as mudanças são incompatíveis, nem correções de bugs pelo FIX.

//VERSIONAMENTO NO PATCH(FIX):
//Podemos mudar o estilo de versionamento de biblioteca para aceitar somente mudanças no "fix", para isso usamos o símbolo de "~", por exemplo: "~0.22.0". Quando deixamos assim, estamos dizendo para o Node que não aceitamos atualizações nem na MAJOR nem na MINOR, somente correções de bug na versão atual da API.

//VERSÃO EXATA:
//Quando desejamos que um projeto pegue somente uma versão exata, que ela não aceite versões atualizadas de forma alguma, basta não colocarmos símbolo nenhum, da seguinte forma: "0.22.0". Melhor ainda, podemos atribuir isso no momento que baixamos a API, por através da flag "-E", essa flag significa que deverá pegar uma versão exata que também precisa ser passada no comando, dessa forma:
//      npm i --save bibliotecaTal@.17.1 -E
//  Note que incluímos ao nome da biblioteca o "@" seguido pelo número de versão da biblioteca, obrigatóriamente temos que informar qual é o número de versão, e ao final a flag "-E" (Sempre em maíusculo) pois ela que vai dizer que o arquivo deve ser o Exato!



//GERANDO SCRIPTS PARA SEREM EXECUTADOS NO TERMINAL DO NODE Á PARTIR DO PACKAGE.JSON:
//É possível criar scripts que poderão ser executados no terminal do Node á partir do nosso arquivo package.json, existe um atributo dentro do nosso arquivo package JSON que se chama "scripts", dentro desse atributo temos um objeto onde podemos criar novos atributos contendo scripts em JSON que poderão ser executados no terminal do Node.

//Mas existe uma regra a ser seguida quanto a nomeação desses atributos, pois o Node já trabalha com alguns nomes padrão para atributos de script, esses nomes padrão podem ser executados diretamente no node por através do comando:
//              npm nomeDoAtributoDeScript
//      Para saber que scripts já estão pré-definidos pelo Node acesse: https://docs.npmjs.com/cli/v8/using-npm/scripts

//Ou podemos criar os nossos próprios nomes de atributos para que eles possam ser executados no Node, porém deles deverão ser executados de forma diferente por através do comando:
```

```
//              npm run nomeDoAtributoDeScript
//      Sempre usando a flag "run"...



//Para ver um exemplo prático disso entre na aula "135-Ex-JSON" na pasta "Exercicios/135-Funcionarios"
```

ARQUIVO PACKAGE.JSON (PERCEBA QUE OS SCRIPTS START E NOMECRIADOPORNOS FORAM CRIADOS)...

```
{
  "name": "135-funcionarios",
  "version": "1.0.0",
  "description": "Aqui segue uma descrição ficticia para o projeto",
  "main": "index.js",
  "scripts": {
    "start": "135-Ex-JSON.js",
    "nomeCriadoPorNos": "nodemon",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "palavraTal",
    "palavraTal",
    "palavraTal"
  ],
  "author": "Gabriel F. Nobre",
  "license": "ISC",
  "dependencies": {
    "axios": "^0.24.0"
  }
}
```

RESULTADO NO TERMINAL QUANDO O SCRIPT START É EXECUTADO...

Windows PowerShell

Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6

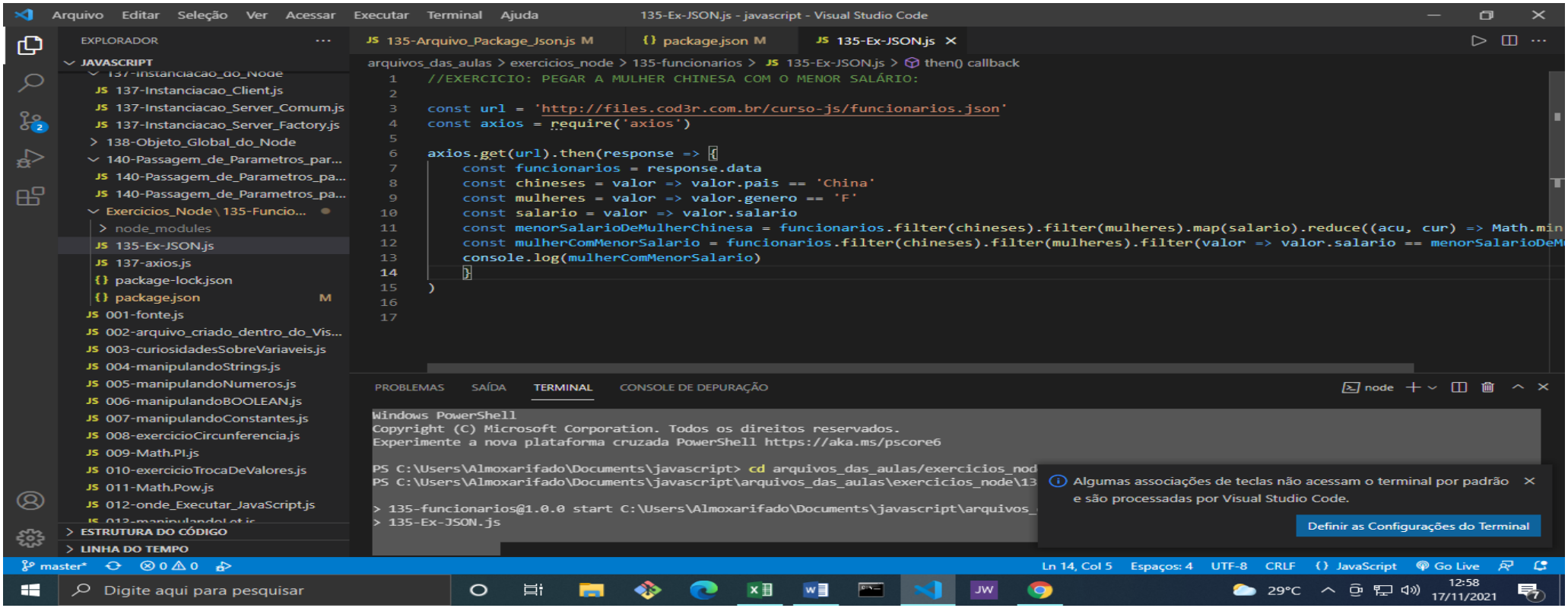
PS C:\Users\Almoxarifado\Documents\javascript> cd arquivos_das_aulas/exercicios_node/135-funcionarios

PS C:\Users\Almoxarifado\Documents\javascript\arquivos_das_aulas\exercicios_node\135-funcionarios> npm start

> 135-funcionarios@1.0.0 start C:\Users\Almoxarifado\Documents\javascript\arquivos_das_aulas\exercicios_node\135-funcionarios

> 135-Ex-JSON.js

QUANDO START É EXECUTADO O ARQUIVO ABRE NA IDE...



RESULTADO NO TERMINAL QUANDO O SCRIPT CRIADOPORNOS É EXECUTADO...

```
PS C:\Users\Almoxarifado\Documents\javascript\arquivos_das_aulas\exercicios_node\135-funcionarios> npm run nomeCriadoPorNos

> 135-funcionarios@1.0.0 nomeCriadoPorNos C:\Users\Almoxarifado\Documents\javascript\arquivos_das_aulas\exercicios_node\135-funcionarios
> nodemon

[nodemon] 2.0.15
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
internal/modules/cjs/loader.js:892
  throw err;
  ^

Error: Cannot find module 'C:\Users\Almoxarifado\Documents\javascript\arquivos_das_aulas\exercicios_node\135-funcionarios\index.js'
    at Function.Module._resolveFilename (internal/modules/cjs/loader.js:889:15)
    at Function.Module._load (internal/modules/cjs/loader.js:745:27)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:76:12)
    at internal/main/run_main_module.js:17:47 {
  code: 'MODULE_NOT_FOUND',
  requireStack: []
}
[nodemon] app crashed - waiting for file changes before starting...
```