**Instalando o Node**

Entrar no site oficial do Node atualmente é o node.org

Ir no botão de instalação e baixar a versão LTS que é a mais recomendada.

Baixe e acordo com seu sistema operacional.

**Testando o Node e o NPM no terminal e verificando versões**

Comando: node -v

Comando: node --version

Comando:npn -version

**Criando o primeiro projeto**

No terminal na pasta do projeto roda o comando:

Comando: npm init

Próximo passo é só dar enter nas perguntas relacionadas ao projeto ou nomear se desejável.

Após todos os passos sucedidos criara um arquivo chamado package.json que é os dados básicos informados relacionados ao projeto.

Caso queira ignorar as perguntas relacionadas ao projeto rode o comando abaixo:

Comando: npm init -y

O arquivo package.json é onde estará todas as informações e bibliotecas importantes do projeto.

**Executando um código js no terminal.**

**Comando:** node nomeDoArquivo.js

**Instalando o Typescript no projeto**

Comando: npm install -g typescript

**Iniciando o typescript**

Comando: tsc --init

**Separando os arquivos por pastas**

Criar a pasta dist: Onde ficara os arquivos de distribuição (compilado, traduzido pelo js)

Criar a pasta src: Onde ficara o código fonte do projeto

**Instalando o arquivo de configuração Typescript:**

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

Todas a configurações do typescript estão armazenadas no arquivo **tsconfig.json**

**Dentro deste arquivo definidos algumas configurações importantes para o nosso projeto**

**Interface gráfica do usuário, Texto

Descrição gerada automaticamente**

**target é onde dizemos a versão do ecmascript que queremos usar para traduzir o arquivo js para a versão desejada.**

**ModuleResolution é onde dizemos para o nosso projeto que vamos trabalhar com node, para não ter variações de comportamento é uma boa pratica informar esta configuração, caso contrário deixar no classic.**

**Após devemos definir as pastas (src, dist)que criamos para o nosso projeto:**

**Tela de celular

Descrição gerada automaticamente com confiança média**

**Descomentar o rootDir e informar que a pasta do projeto é a ‘src’**

**Descomentar o outDir e informar que a pasta final do projeto é a ‘dist’**

**Salve e o projeto já estará configurado.**

**Instalando uma dependência de inteligência para o typescript (auto complete e etc)**

O comando: npm install --save-dev @types/node

--save-dev informa que a biblioteca será apenas para desenvolvimento para criar o projeto.

Vai criar a pasta node\_modules onde contém os módulos Node e as bibliotecas que baixamos, esta pasta não pode ser alterada, é a pasta mais pesada do projeto, pode conter vários arquivos.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Não incluir a pasta node\_modules no repositório, pois ela pode ser criada novamente quando baixado o projeto, esta biblioteca esta nas dependências do package.json, com o comando **npm install** a pasta node\_modules será criada.

**Deixando o typescript monitorando a pasta do projeto Wath mode**

Comando: tsc -w

Deve deixar o typescript monitorando a pasta do projeto, para sempre que houver modificações ele cria na pasta dist o código onde contém o js puro.

Para isso vamos abrir outro terminal ao lado e digitar o comando > tsc -w , após isso ele ficará executando e monitorando as modificações que fizer no typescript. O comando tsc -w significa wath mode.

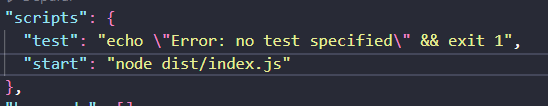
Deve rodar o node em outro terminal e deixar o wath mode monitorando ao lado.

Para rodar o node usa o comando: node dist/index.js

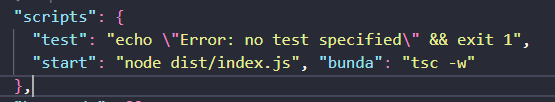
**Scripts do package.json**

Package.json e não pode ser alterada, dentro desta pasta no “scripts” podemos colocar atalhos para rodar(scripts de atalho) comandos personalizados. Exemplo abaixo colocamos o comando “"start": "node dist/index.js"

Executamos esse comando no terminal como> npm run start , o comando start já existe como no caso de >npm start é a mesma coisa. Só pra saber podemos personalizar outros comandos no scripts.



Pode criar outro atalho por exemplo para iniciar o wath mode do tyscript com o comando bunda.



Pode criar atalhos que executam vários códigos, que rodam ouros atalhos por exemplo vários comandos usando o && entre os comandos dentro da virgula.

**Entendendo o import/export em Commonjs (usando de forma padrão)**

Exportar o conceito é habilidade para uso em outros arquivos

Primeiro habilitamos para uso externo o arquivo e depois importando para o arquivo atual

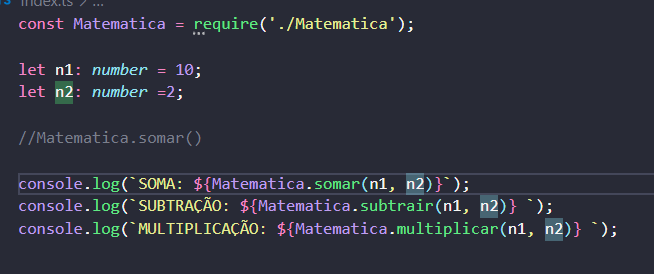
Imagem: arquivo Matemática.js

Texto

Descrição gerada automaticamente

Vou para o arquivo que eu quero importar uso o const para importar o arquivo

Imagem: arquivo index.js



Roda com o comando node dist/index.js

Resultado:

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

**Técnicas mais modernas de fazer essas importação usando o ES6**

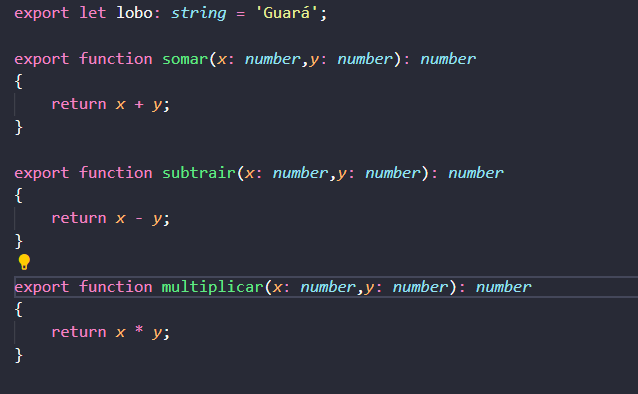
Apenas colocando o export antes da variável ou função o que quiser exportar

Exportando uma variável :



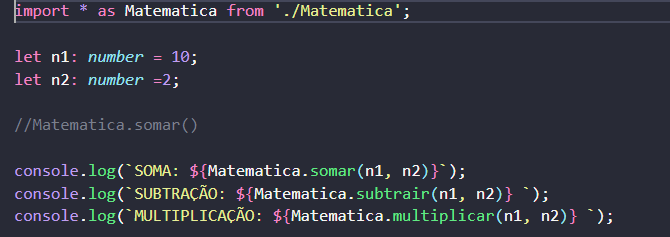
Habilitando para uso externo(exportando) um variável e três funções:

Imagem arquivo: Matematica.ts



Importando tudo:

Imagem: aquivo index.ts



Importando apenas algumas funções:

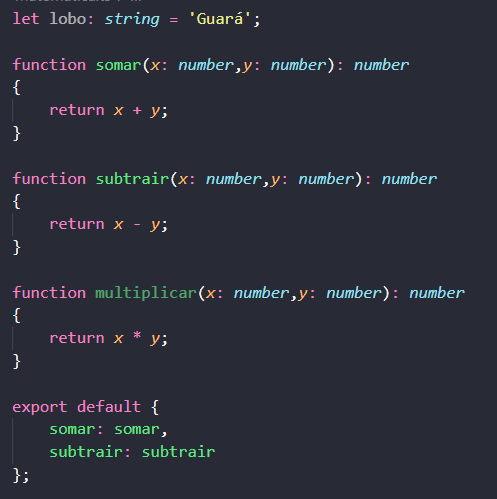
Imagem: arquivo index.ts

Texto

Descrição gerada automaticamente

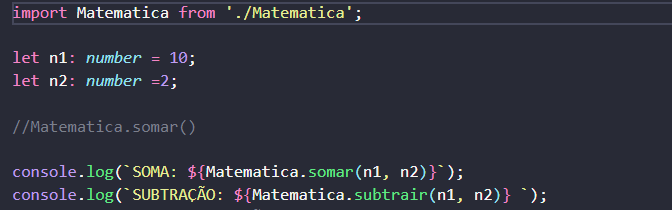
Exportando apenas uma única coisa:

Imagem: arquivo Matematica.ts



Importando padrão

Imagem: arquivo index.ts



**Instalando biblioteca na comunidade npm.js**

Instalando a biblioteca validator

<https://www.npmjs.com/package/validator>

para instalar digite o comando:

comando: npm install validator

usando a biblioteca

no arquivo index.js devemos importar a biblioteca

import validator from ‘validator’;

**instalando as declarações da biblioteca validator**

npm install --save-dev @types/validator

Usando a biblioteca nodemom executa o codigo sem dar start

Instalando: npm install -g nodemon

Rodando: nodemon dist/index.js

Usando o ts-node que faz um comdando em 2, converte ts em js e ts -w

Instalando: npm install -g ts-node

rodando: ts-node src/index.ts ou se intalado o nodemon: nodemon src/index.ts

com ts-node podemos rodar direto o codigo fonte sendo a pasta src

se instalado o nodemon podemos apenas rodar o nodemon que suporta o ts-node

rodando: nodemon src/index.ts

**Como criar nosso servidor para rodar no navegador**

**Instalando o Express e criando um servidor**

Criar o arquivo server.ts e dentro devemos criar todo o funcionamento do servidor, instalar a biblioteca express

npm install express

**instalando os types do express**

npm install @types/express

**Para usar a biblioteca express devemos dentro do arquivo server.ts importar :** import express from ‘express’;

**Para usar:**

const server = express();

**gerando o servidor:**

server.listen(3000); //o primeiro parâmetro devemos colocar a porta que vamos executar no caso estamos usando a 3000.

Ao usar a porta 3000 devemos especificar no navegador <http://localhost:3000>, somente a porta 80 não tem necessidade de especificar, colocando <http://localhost> rodara a pagina porque a porta 80 é porta padrão do navegador.

Atenção: Para o node reconhecer o express devemos instalar o types da biblioteca express, instalando o types ao colocar o ‘ server. ’ aparecera as opções da biblioteca:

Exemplo: Criando uma pagina principal na porta 80

Texto

Descrição gerada automaticamente

Para executar rodar o comando nodemon no server.ts, abrir o navegador e digitar o endereço: <http://localhost> e ira aparecer a mensagem ‘Olá mundo!’

**Rotas e métodos**

Get acessa a pagina

Post recebe dados internamente

**Exemplo de Rota estática:**

server.get(‘/’, (req: Resquest, res: Response)=>{

Res.send(‘Olá mundo !’);

} );

Server.get recebe dois parâmetros a rota e segundo é o call back a função

A função recebe por padrão alguns parâmetros sendo o mais comuns req e o res; o req é a requisição da página, o res é responsável pela resposta que o servidor dará para quem fez a requisição.

req = controle da requisição

res = controle da resposta

**Exemplo de rota dinâmica:**

Rota dinâmica é quando tem algum dado que pode mudar mas tem que ir pra mesma página.

server.get(‘/noticia/:slug ’, (req: Resquest, res: Response)=>{

let slug : string = req.params.slug;

Res.send(‘Noticia: ${slug}’);

} );

/noticia/: = após o : colocamos uma referência para a página dinâmica no caso colocamos o nome slug. ‘/noticia/:slug’ após isso essa rota se transformara em uma rota dinâmica, na requisição colocamos os dados que precisamos para receber, no caso declaramos a variável slug para saber qual é o slug com req.params. após o . colocamos o nome que queremos receber no caso colocamos o slug ficando a rota dessa forma: req.params.slug;

Para acessar vamos no navegador: <http://localhost/noticia/aqui> digitamos a página dinâmica no :slug que pega qualquer coisa que tenhamos digitado após noticia/.

Na rota dinâmica podemos colocar vários itens, no exemplo abaixo vamos direcionar uma rota dinâmica para o seguinte site // site.com/voo/gru-rec

O ‘gru’ é uma rota e ‘rec’ será outra então faremos:

server.get(‘/voo/:origem-:destino’, (req: Request, res: Response)=>{

//let origem = req.params.origem;

//let destino = req.params.destino;

//fazendo a mesma coisa abaixo

let { origem, destino } = req.params;

res.send(`Procurando voos de ${origem.toUpperCase()} até ${destino.toUpperCase}`);

});

Então criamos uma rota dinâmica com dois valores diferentes, usamos toUpperCase para exibir letra maiúscula na tela.

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

Criando o erro 404

//quando a pagina não encontrou nenhuma das rotas

server.use((*req*: *Request*, *res*: *Response*)=>{

*res*.status(404).send('Pagina não encontrada!');

});

Acessando uma pasta publica em nossa aplicação



//como disponibilizar uma pasta como publicamente acessivel como arquivo estatico

server.use(express.static(path.join(\_\_dirname, '../public')));//pegando o endereço absoluto ate a pasta public

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Motores de templates engines

Instalando e configurando o mustache

npm install mustache-express

npm install --save-dev @types/mustache-express

showWelcome

Tela de celular com aplicativo aberto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Configurando o nodemon no package.json para rodar o mustache

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Exibindo condicional

Tela de computador com letras e números em fundo preto

Descrição gerada automaticamente

O exemplo abaixo somente sera exibido se a variável showCond for true

Texto

Descrição gerada automaticamente

Exibindo uma lista de objetos no mustache

Texto

Descrição gerada automaticamente

No html

Texto

Descrição gerada automaticamente

No navegador

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Exibindo uma lista simples no routes/index.ts

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

No html home.mustache

Texto

Descrição gerada automaticamente

No navegador

Texto

Descrição gerada automaticamente

Else de liste (quando eu não tenho uma lista na pagina)

Lista vazia no routes/index.ts

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Home.mustache usa o ^ para o else

Texto

Descrição gerada automaticamente

Exibindo na tela

Tabela

Descrição gerada automaticamente

**Node com Banco de dados Mysql**

ORM - OBJECT RELATIONAL MAPPER

MAPEAMENTO OBJETO-RELACIONAL

**Biblioteca ORM Sequelize**

npm install --save sequelize

**Sequelize para rodar precisa das seguintes bibliotecas**

@types/node

@types/validator

@types/sequelize = npm install --save-dev @types/sequelize

**Fazendo a conexão no MySql**

Instalar a biblioteca de gerenciamento do MySql

npm install mysql2

**Configurando no env os dados para conexão do banco de dados**

MYSQL\_DB=teste

MYSQL\_USER=root

MYSQL\_PASSWORD=root

MYSQL\_PORT=3306

**Tipos de Consultas no banco de dados**

//pegando todos os usuários no banco de dados

let users = await Users.findALL();

**//pegando apenas os dados que vou ultilizar no caso name e age**

let users = await Users.findALL({

attibutes: ['name', 'age']

});

**//mudando o nome de algum campo sem mudar no bdo**

let users = await Users.findALL({

attibutes: ['name', ['age', 'idade']]

});

**//Excluindo da exibição o que eu nao quero no caso 'id' e 'age'**

let users = await Users.findALL({

attibutes: {exclude: ['id', 'age']}

});

**//Filtrando resultados**

let users = await Users.findALL({

where: { name: 'Paulo'} //onde o name for 'Paulo'

});

**//Filtrando apenas quem tem 90 anos**

let users = await Users.findALL({

where: { age: 90}

});

**//Filtrando apenas quem tem o id 3**

let users = await Users.findALL({

where: { id: 3}

});

**//Filtrando duas condiçoes**

let users = await Users.findALL({

where: { age: 30, name: 'Paulo' }

});

#####################################################################################

**//Filtrando duas condições com a operação or (ou)**

import {Op} from 'sequelize'; //Op faz operações

let users = await Users.findALL({

where: {

[Op.or]: [

{age: 55}

{name: 'Paulo'}

]

}

});

//O outra forma com or

let users = await Users.findALL({

where: {

age: [55, 30, 90]

}

//Cosultando o nome pela letra

User.finALL({

where: {

name: {

[Op.like]: "%M", // "%M%" que tem m em qualquer lugar // que tem "%pa"

},

},

});

**//Operadores para fazer a filtragem**

let users = await User.findAll({

where: {

age: {

[Op.gt]: 40, //> 40 maior que 40

[Op.gte]: 40, // >=40 maior igual a 40

[Op.lt]: 40, // < 40 menor que 40

[Op.lte]: 40, // <=40 menor igual 40

[Op.between]: [40, 100] // pegando quem entre as duas idades

[Op.notbetween]: [40, 100] // pegando quem não tem entre as duas idades

[Op.in]: [30, 40] //pega todos que tem esta idade

[Op.notIn]: [30, 40] //pega todos que não tem esta idade

[

}

}

});

**Ordenação e limitação de resultados**

**Como organizar a lista**

// order: ['name'] / ordena pelo nome crescente ou order: ['name', 'ASC']

let users = await User.findAll({

where: {

age: {

[Op.gte]: 18 //pegando todos que tem acima de 18

}

},

order: ['name']

});

**// ordena pelo nome Decrescente order: [['name', 'DESC']]**

let users = await User.findAll({

where: {

age: {

[Op.gte]: 18 //pegando todos que tem acima de 18

}

},

order: [['name', 'DESC']]

});

**//Ordena pela idade**

order: [['age', 'ASC']]

**LIMITANDO RESULTADOS**

offset: 2 //pula a qtd 2 em 2 paginas

limit: 2 // quantidade a exibir

**Inserindo dados no banco com Sequelize, duas formas de inserir com 'build + save' ou 'create'**

**//Inserindo com build + save**

**cria a instancia do usuario**

const user = User.build({

name: 'Fulaninho',

age: 25

});

await user.save(); //salva no banco

**//Inserindo user com create sem save**

const user = await User.create({

name: 'Ciclano',

age: 39

});

**//Atualizando o nome e a idade com sequelize**

//1 parametro dados a serem alterados

//2 parametro Condição para encontrar o(s) tem(s)

await User.update({name: 'Barbara', age: 13}, {

where: {

id:10

}

});

**//Atualizando todos menores de idade para 18**

await User.update({age: 18}, {

where: {

age: {

[Op.lt]: 18

}

}

});

**Selecionando um usuario pelo id**

let result = await User.findAll({ where: {id: 7} });

**Alterando um item especifico**

let results = await User.findAll({ where: {id: 7} });

if(results.length > 0){

let usuario = results[0];

usuario.name = 'Testador';

usuario.age = 18;

await usuario.save();

}

**Deletando todos usuarios menor e igual a 18 anos**

await User.destroy({

where: {

age: {

[Op.lte]: 18

}

}

});

**Outra forma de deletar apenas um item**

let results = await User.findAll({ where: {name: 'Ciclano' } });

if(results.length > 0 {

let usuario = results[0];

await usuario.destroy();

}

**5 formas de encontrar um registro no banco de dados**

Usando findOne

let usuario = await User.findOne({

where: {

name: "Sandro"

}

});

if(usuario){ //exibindo no console

console.log(`O usuario ${usuario.name} possui ${usuario.age} anos`);

}else {

console.log('Usuario nao encontrado');

}

**Encontrando apenas com id**

let usuario = await User.findByPk(1);

**Usuario sera criado apenas se nao existir com o findORCreate() // colocar no banco o autoImcrement para nao dar erro**

const [ usuario,created ] = await User.findOrCreate({

where: { name: 'Sandro' },

defaults: {

age: 80

}

});

if(created){

console.log("Usuario criado com sucesso");

} else {

concole.log("Usuario existe");

}

console.log("Nome: ", usuario.nome);