**NODEJS**

yarn add express -> instala todas as dependências do express: node etc...

yarn add nodemon **-D** -> dependência responsável por fazer o node executar automaticamente

**"-D"** serve para instalar dependência em modo de desenvolvimento, não será utilizada quando a aplicação for para produção

**Comandos Essenciais:**

**SPLICE**: método que percorre o vetor, através do index, ao chegar o mesmo é deletado.

**res.JSON(name):** mostra a estrutura na tela.

**res.SEND():** retorna uma msg de statusCode na tela.

Find(): Mostra resultado do JSON em tela.

FindIndex(): Mostra o número index do array ex.: 0, 1, 2.

**server**.**listen**(**3000**) - > define rota

REQ/RES ->

**Req(Requisição)**: pega informações do frontend para o backend

**Res(Resultado)**: envia informações do do backend para o frontend

***package.json:*** *após aplicar o comando abaixo no package rodar o comando*

*“****yarn dev****” no terminal*

"scripts": {

    "dev": "nodemon index.js"

  },

***Middlewares:*** função que recebe os métodos (req, res), recebe outros parâmetros e faz alguma coisa dentro da aplicação. Manipula os dados de requisição e resposta de alguma forma. Pode ser identificado através do metodo **“NEXT”.**

***Middleware Global:*** pode ser trabalhada de forma global utilizando o método ***“use”***

 server.use((*req*, *res*, *next*) => {

        //console.log('A requisição foi chamada!');

        //criação de middleware de log

*console*.time('Request');

*console*.log(`Método: ${req.method}; URL: ${req.url}`);

        next();

*console*.timeEnd('Request');

      });

***Middleware Local:*** Este é aplicado nas rotas **(Post e Put)**

//Esta mid.. vai no corpo da requisição e ver se existe a informação "Name",

      //Se nao encontrar, ele vai retornar um erro p/ o usuário a msg informada

      //Se não entrar no IF, ele vai chamar o mid... da rota normalmente

      //Estes mid.. podem ser utilizados nas ROTAS, basta informar o nome da função

      //depois da rota

      function checkUserExists(*req*, *res*, *next*) {

        if (!req.body.name) {

          //STATUS - meto de identifica o tipo de error

          return res.status(400).json({ error: 'User name is required'});

        }

server.put('/users/:index', checkUserExists, checkUserInArray, (*req*, *res*) => {

        const { index } = req.params;

        const { name }  = req.body;

        users[index] = name;

        return res.json(users);

      });

***Middleware verifica se user name existe:*** Este é plicado nas rotas **(Get, Put e Delete)**

function checkUserInArray(*req*, *res*, *next*) {

        if (!users[req.params.index]) {

          return res.status(400).json({ error: 'User does not exists'});

        }

        return next();

      }

***Route params:***

// Route params = /users/1

    //const id = req.params.id;

    //Aplicando desestruturação.

    //Serve p/ impedir que se receba um mesmo objeto antes e depois do igual

      server.get('/users/:id', (*req*,*res*) => {

      const { id } = req.params;

      return res.json({message: `Buscando o usuário: ${id}`});

***Query Params:***

// Query params = ?teste=1

  server.get('/teste', (*req*, *res*) => {

    const nome  = req.query.nome;

      return res.json({message: `Hello ${nome}`})

  });

***CRUD - Create, Read, Update, Delete***

***CREATE***

server.post('/users', checkUserExists, (*req*, *res*) => {

        const { name } = req.body;

        users.push(name);

        return res.json(users);

      });

***READ***

//Lista todos os usuários

  server.get('/users', checkUserInArray, (*req*, *res*) => {

    return res.json(users);

  });

//Lista apenas o usuário pelo id

 server.get('/users/:index', checkUserInArray, (*req*, *res*) => {

   const { index }   = req.params;

   return res.json(users[index]);

 });

***UPDATE***

//Faz update nos usuários pelo ID

server.put('/users/:index', checkUserExists, checkUserInArray, (*req*, *res*) => {

        const { index } = req.params;

        const { name }  = req.body;

        users[index] = name;

        return res.json(users);

      });

***DELETE***

//Faz delete pelo ID

      server.delete('/users/:index', checkUserInArray, (*req*, *res*) => {

        const { index } = req.params;

        //SPLICE percorre o vetor, através do index, ao chegar o mesmo é deletado.

        users.splice(index, 1);

        //res.JSON, mostra a estrutura na tela, o res.SEND() retorna statusCode de certo

        return res.send();

      })