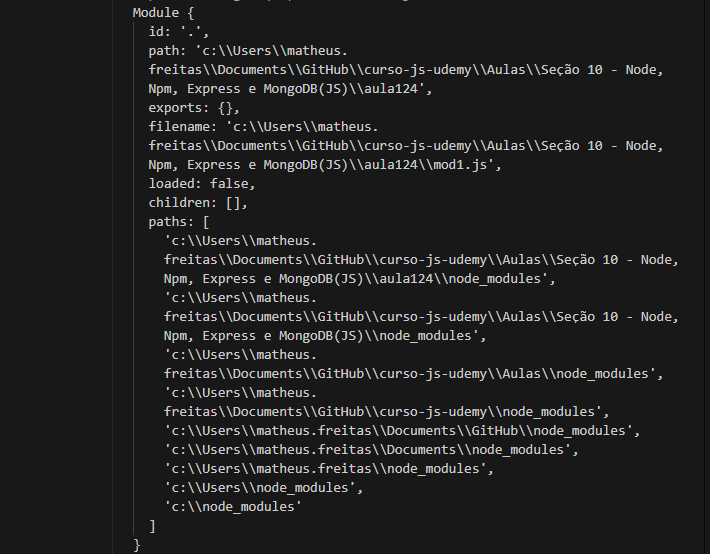
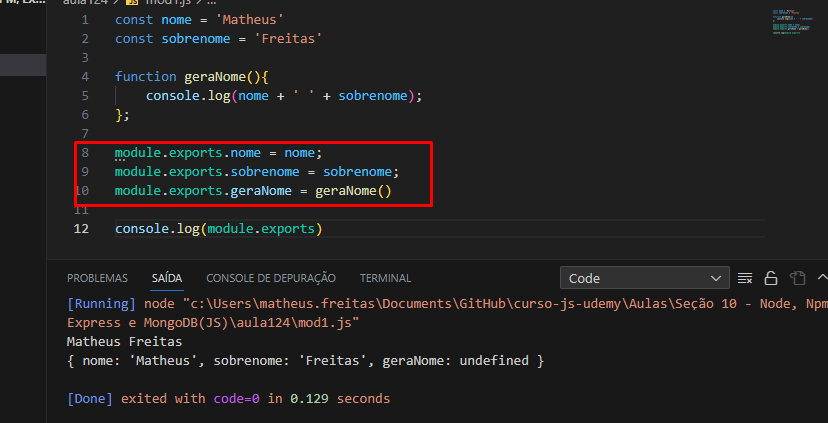
**AULA 124**

Nesta aula iremos ver como funciona os módulos no Node.js. No Node, um módulo é como um objeto que possui vários índices.

Exportar arquivos

Um desses indicies é o “exports”. Quando exportamos algo no Node, esse elemento vai para esse objeto vazio, criando um índice no objeto.

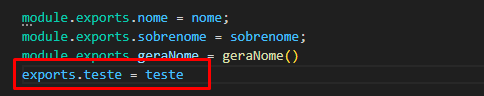
Para exportar algum elemento, precisamos atribuir esse elemento como um índice deste objeto. Exemplo:



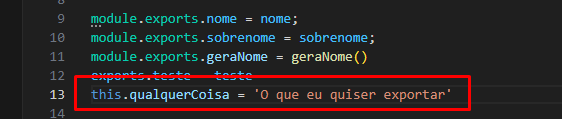
1. Inserimos o module.exports.
2. Colocamos o nome do índice que desejamos criar (recomendado utilizar o mesmo nome do elemento para facilitar identificação)
3. Inserimos o sinal de atribuição (=)
4. Inserimos o elemento que desejamos exportar

Podemos exportar todos os elementos, como variáveis, arrays e objetos.

Podemos também exportar elementos utilizando o comando “exports” direto. Funciona como um “atalho”, passando o valor por referência direto para o module.exports. Exemplo:

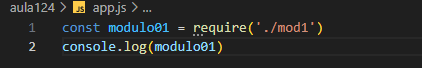


E por último, também podemos utilizar o comando “this” para realizar a exportação. Exemplo:

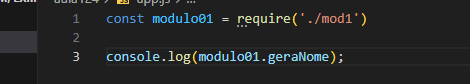


Importar arquivos

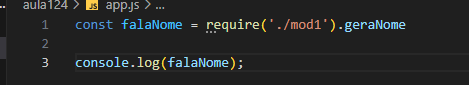
Quando importamos elementos dentro do node, acabamos importando toda o objeto “exports”. A forma mais simples de realizarmos a importação, é utilizando o comando require. Exemplo:



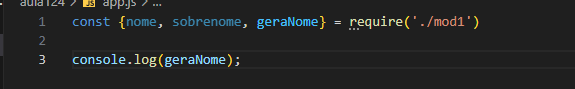
Assim, podemos utilizar os elementos criados no primeiro módulo. Exemplo:



Podemos importar também apenas um elemento do primeiro módulo, para isto, basta colocar um “.” e nome do elemento que você deseja importar. Exemplo:



Podemos também exportar elemento já atribuindo, vindo atribuição em desestruturação. Exemplo:



**AULA 125**

Esta aula é uma continuação direta da aula anterior, sobre módulos no Node.js. Vimos como utilizar os símbolos “./” para indicar algum caminho de um módulo ou um outro arquivo na linha de código

**AULA 126**

Esta aula é sobre o npm do node. Quando iniciamos o NPM, automaticamente ele cria o arquivo “package.json” que mostra os complementos que vamos utilizar no Node e suas versões

Para instalarmos uma aplicação de terceiro, utilizamos o comando “npm install + nome do app”. Exemplo:

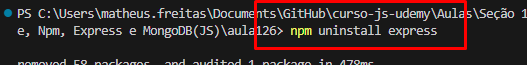


Quando instalamos dessa forma, o node vai realizar a instalação da versão mais recente do aplicativo e caso o app tenha receba alguma nova atualização, o node irá atualizar o app automaticamente

Para instalarmos uma versão especifica de um app, basta inserir um “@ + versão do aplicativo”

Quando possuímos um app voltado somente para o desenvolvimento do software, geralmente instalamos ele no campo “devDependencies”, para isso, basta colocar o comando “—save-dev” na linha de comando da instalação do app.

Para desinstalar um app, podemos utilizar o comando “npm uninstall + nome do app”. Exemplo:



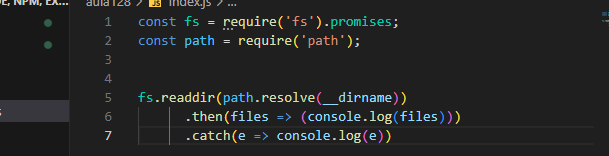
**AULA 128**

Esta aula fala sobre o File System, que é um modo de interagir o com o sistema de arquivos. Nesta aula vimos o comando “fs.readdir” que possui muitas funções, mas nesta aula utilizamos para descobrir o caminho do arquivo que estamos trabalhando.

O comando “fs.readdir” retorna uma array, no nosso exercício ele deverá retornar uma array que contém o caminho do nosso arquivo.

1. Criamos duas varáveis para podemos uitlizar o file system como uma promises e o path
2. Acionamos o file system com o comando readdir
3. Passamos o caminho do nosso arquivo para o comando “fs.readdir”
4. Acionamos o .then e .catch para visualizarmos os resultados

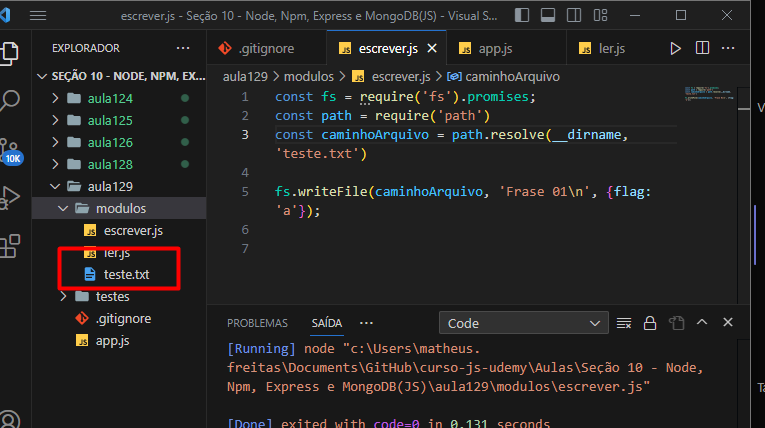
Com ele podemos executar um comando para descobrir o caminho de um arquivo, utilizando o método resolve e path. Exemplo:



**AULA 129**

Nesta aula utilizamos o file system para criar um arquivo txt.

1. Criamos as variáveis “fs” e “path” para utilizarmos esses objetos
2. Criamos uma variável que armazena o caminho do nosso arquivo atual e passamos o nome do arquivo que será criado
3. Utilizamos o comando fs.writeFile para criar o arquivo, passando os parâmetros
   1. Caminho de onde irá ficar o arquivo e o seu nome
   2. Um texto que estará dentro do nosso arquivo (\n é usado para o texto não ficar sendo criado na mesma linha quando executarmos o código mais de uma vez)
   3. “Flag: ‘a’” é usado para quando executarmos o código novamente, ele apenas acrescentar o texto mais uma vez



**AULA 130**

Nesta aula vamos aprender a criar um servidor web utilizando Express (é um micro framework que auxilia o trabalho com a web).

O Express auxilia nas rotas da nossa aplicação ficando responsável por direcionar as solicitações (quando acessamos um site, nosso navegador faz uma requisição para o servidor utilizando o método get)

Como Express podemos realizar algumas operações, as mais famosas delas são as operações CRUD

Operações CRUD – Create, Read, Update, Delete

No Express, essas operações correspondem aos seguintes comandos:

* Create => Post | Criar
* Read => Get | Ler
* Update => Put | Atualizar
* Delete => Delete | Apagar

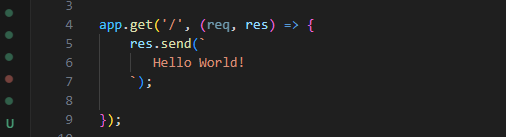
Invocamos o express através de uma variável e criamos a variável app para conseguirmos manipular o Express



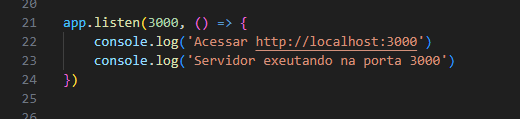
Utilizamos o método GET – Quando utilizamos o método GET, precisamos passar dois parâmetros, o primeiro é o caminho da página que você deseja fazer as solicitações. O segundo parâmetro é uma função anônima que recebe mais dois parâmetros:

* Requisição => Referente ao que você está solicitando
* Resposta => O que será enviado ao servidor

Dentro da função anônima, criamos uma resposta para a solicitação do GET, ou seja, quando o método for acionado, a resposta enviada ao solicitante será uma página com a mensagem “Hello World!”

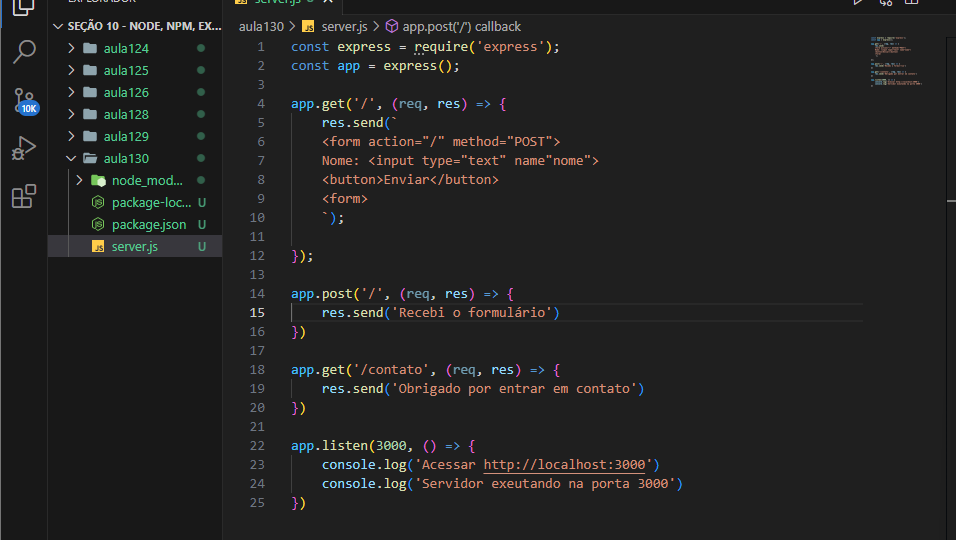


Utilizamos o método “listen” para monitorar uma porta do nosso servidor, para que quando haja alguma atividade nela



Dentro do método listen, criamos uma função anônima que manda uma mensagem para o console de que a execução iniciou e que entrega o link do que está sendo executado

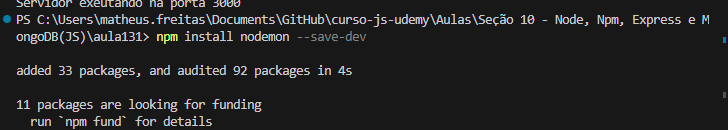
Criamos uma segunda página para testar e um formulário



**AULA 131**

Nesta aula conhecemos o Nodemon. É utilizado para atualizar nossa página em tempo real quando utilizamos o get e express.

Para realizar a instalação do software, abrimos o terminal e executamos o comando “npm install nodemon –save-dev”



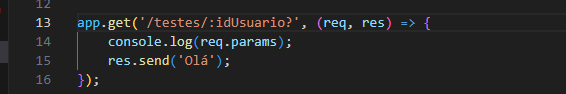
**AULA 132**

Nesta aula vamos falar sobre a manipulação das rotas de uma aplicação web através do Express.js

Para realizamos consultas, alterações ou criações de rotas podemos utilizar algumas funções, a seguir, vamos ver algumas delas.

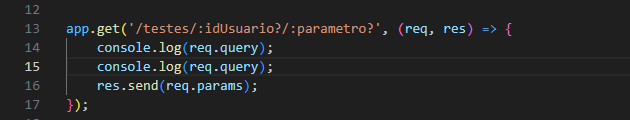
Req.params

Uma forma de manipularmos uma URL. Quando acessamos uma aplicação via internet, fazemos uma solicitação ao servidor da aplicação que nos retorna com a URL da aplicação, utilizando Req.params podemos manipular a URL adicionando alguns parâmetros. Para passamos um parâmetro para nossa url, basta colocar o sinal de dois pontos na frente da nossa URL e o nome do parâmetro. Exemplo:



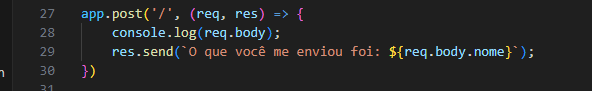
Req.query

O query params é outra forma de passar valores pela url mas, dessa forma ao invés de passar apenas um parâmetro por rota, podemos passar vários! Normalmente utilizado em filtros de pesquisa, usado após o sinal de interrogação ("?"). Exemplo:



Req.body

Armazena as informações mais cruciais (como o nome e a senha de usuário) que não devem ficar visíveis na URL. Exemplo: Quando utilizamos um formulário com o método “POST”, as informações são enviadas para esta “chave”, que pode ser acessada através do Express com comando req.body



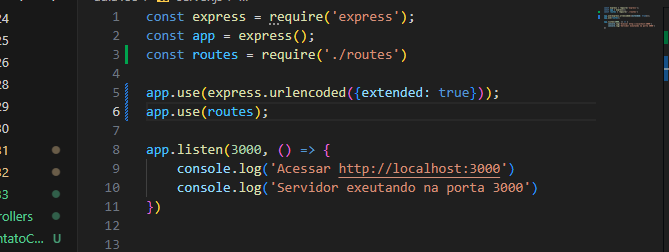
**AULA 133**

Na aula, vimos que se usarmos um arquivo para toda uma aplicação web, o arquivo ficaria enorme e se tornaria inviável para se trabalhar, para isso, nesta aula, vamos ver como podemos utilizar modularização junto com o roteamento do Express.js.

Criamos alguns arquivos que exportaram suas funções, veremos eles a seguir:

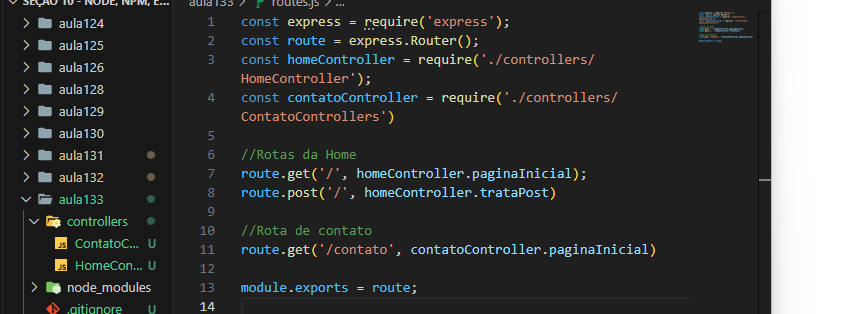
Arquivo “server.js”

O primeiro arquivo “server.js” é o arquivo do servidor da nossa aplicação, onde fica sendo executado uma verificação de alterações no nosso código utilizando o “nodemon” e que importa os roteamentos do arquivo “routes.js”



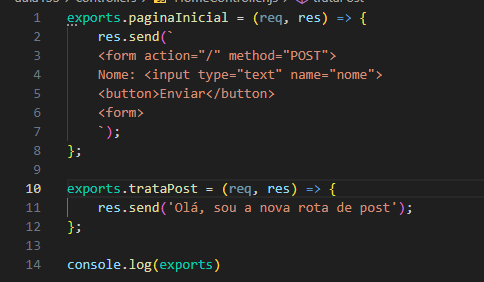
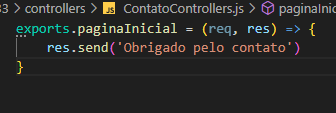
Arquivo “routes.js”

O segundo arquivo ficou responsável pelas rotas da nossa aplicação web. Ele recebe a estrutura das pastas dos arquivos de controle e fica responsável por criar a rota desses arquivos



Arquivos de controle

Os arquivos de controle ficaram responsáveis pelas páginas da nossa aplicação, dentro delas, ficam a estrutura das páginas

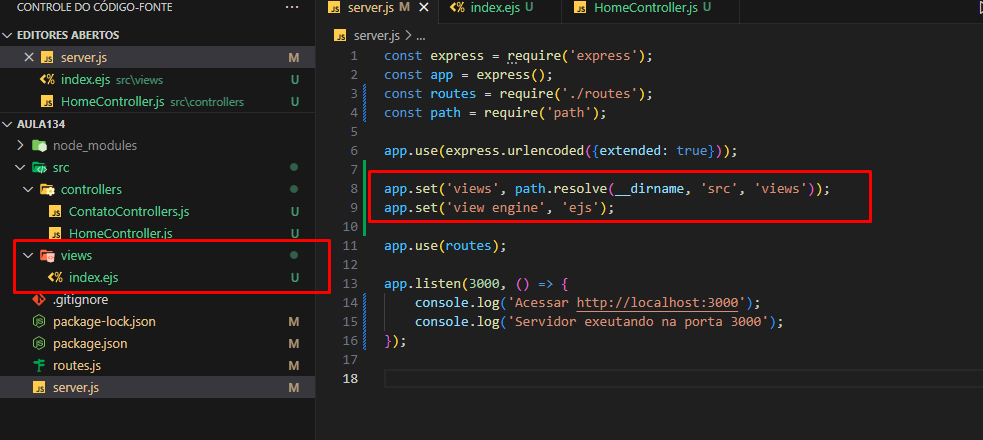


**AULA 134**

Na aula, vimos o conceito do MVC, um padrão para construção de aplicações. **O MVC é utilizado em muitos projetos devido a arquitetura que possui**, o que possibilita a divisão do projeto em camadas muito bem definidas. Cada uma delas, o Model, o Controller e a View, executa o que lhe é definido e nada mais do que isso.

Criamos a pasta “src”, onde colocamos nossas pastas das camadas “Controller” e “View”.

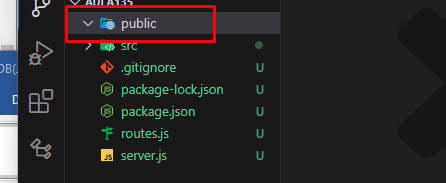
Construímos nossa camada view e atrelamos os nossos controles criados anteriormente ao nosso arquivo “ejs”



**AULA 135**

Nesta aula falamos sobre conteúdo estático. Um conteúdo estático é qualquer arquivo armazenado em um servidor e sempre é exibido aos usuários da mesma forma.

Criamos uma pasta onde iremos colocar nosso conteúdo estático.



**AULA 136**

Nesta aula, fizemos a instalação do webpack na pasta da aula anteior

**AULA 137**

Aula sobre middleware