

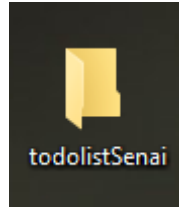
Conteúdo

Módulo 1: Projeto TodoList - Início

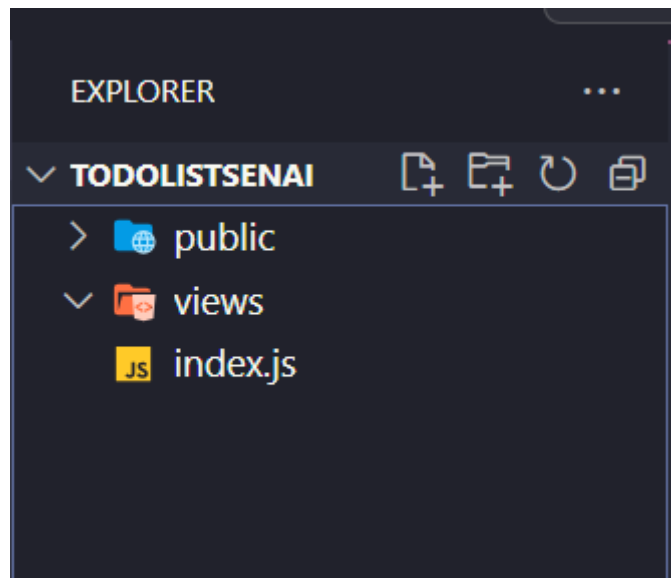
- Instalando Express, Ejs e demais dependências
- Criando rotas, configurando views e diretórios
- Desenvolvimento do projeto

Criando e configurando o nosso projeto

Primeiro vamos criar a pasta do projeto chamada “**TodolistSenai**”:

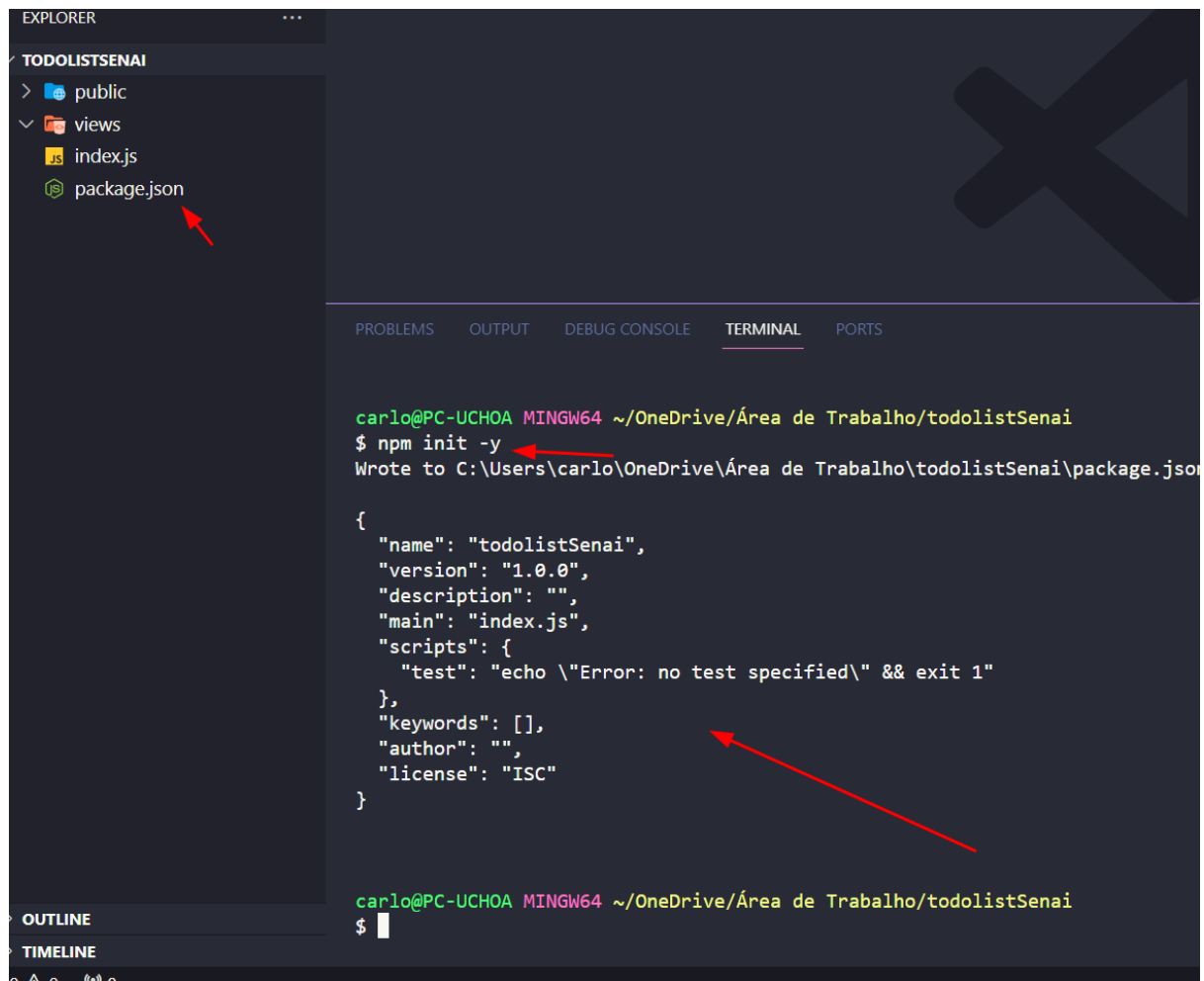


E vamos abrir com VsCode e vamos criar essa estrutura inicial:



Agora vamos criar nossos arquivos package.json com `npm init -y` como já sabemos e vimos nas aulas anteriores.. para instalarmos nossas dependências do projeto.

UniSENAI



```
carlo@PC-UCHOA MINGW64 ~/OneDrive/Área de Trabalho/todolistSenai
$ npm init -y
Wrote to C:\Users\carlo\OneDrive\Área de Trabalho\todolistSenai\package.json

{
  "name": "todolistSenai",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

vamos instalar também o express , o ejs e o nodemon como dependencias.

```
carlo@PC-UCHOA MINGW64 ~/OneDrive/Área de Trabalho/todolistSenai
$ npm install express
```

```
carlo@PC-UCHOA MINGW64 ~/OneDrive/Área de Trabalho/todolistSenai
$ npm install ejs
```

```
carlo@PC-UCHOA MINGW64 ~/OneDrive/Área de Trabalho/todolistSenai
$ npm install nodemon
```

Com estas dependências instaladas conseguimos realizar a criação de nosso projeto, agora vamos criar nosso arquivo de configuração:

```
index.js
index.js > ...
1  const express = require('express');
2
3  const app = express();
4
5  app.listen(5000, ()=>{
6    console.log('Servidor rodando em http://localhost:5000');
7  })
```

Agora vamos configurar nosso arquivo package.json e criar um script para iniciar nosso servidor:

```
Debug
{
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "dev": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "ejs": "^3.1.9",
    "express": "^4.18.2",
    "nodemon": "^3.0.1"
  }
}
```

Criamos o nosso script de atalho e rodamos o projeto com **"npm run dev"**

UniSENAI

```
index.js x
index.js > ...

1  const express = require('express');
2
3  const app = express();
4
5  Comment Code
6  app.listen(5000,()=>{
7  console.log('Servidor rodando em http://localhost:5000');
8  })

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH TERMINAL OUTPUT  COMMENTS

carlo@PC-UCHOA MINGW64 ~/OneDrive/Área de Trabalho/todolistSenai
$ npm run dev
> todolistSenai@1.0.0 dev C:\Users\carlo\OneDrive\Área de Trabalho\todolistSenai
> nodemon index.js

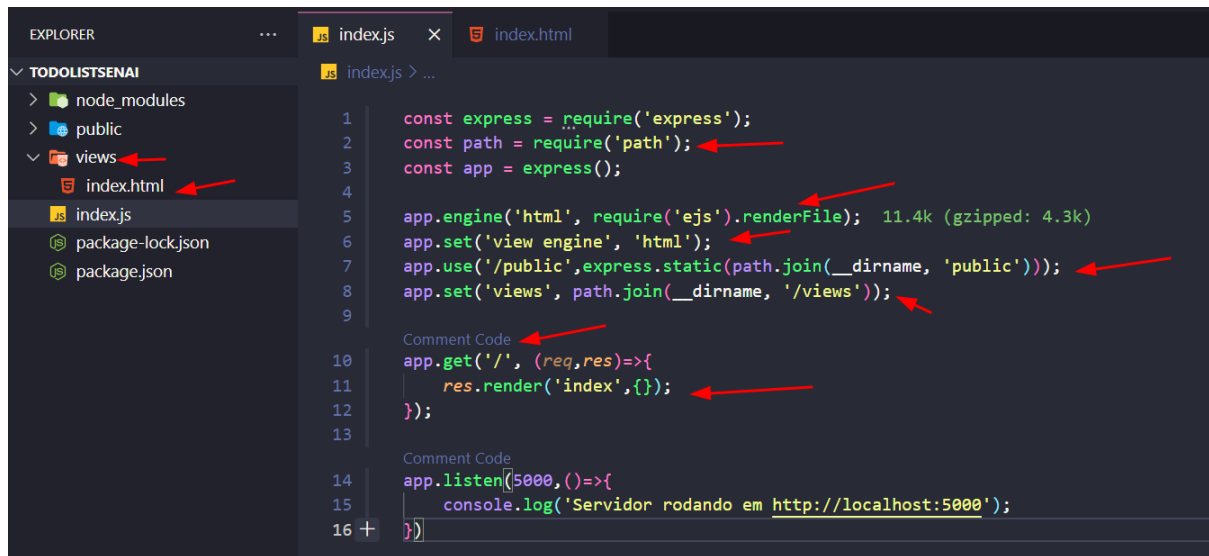
[nodemon] 3.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Servidor rodando em http://localhost:5000
```

Agora vamos criar as rotas, pois como podem ver não tem nada na rota localhost:5000



Cannot GET /

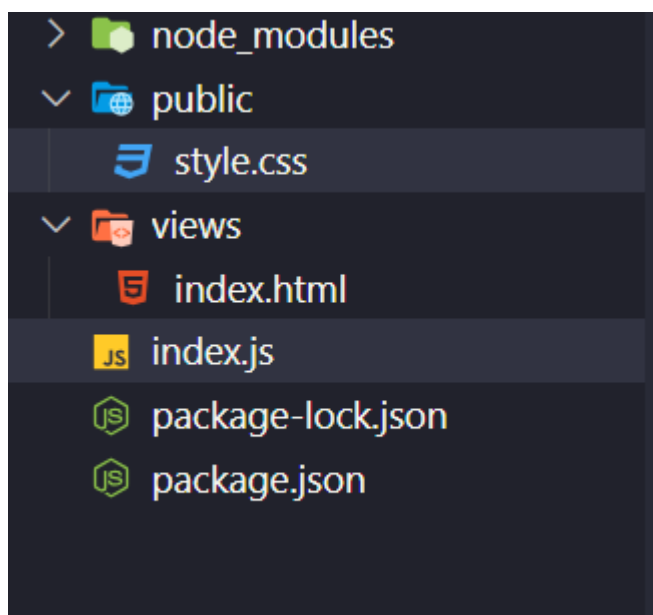
UniSENAI



```
1  const express = require('express');
2  const path = require('path');
3  const app = express();
4
5  app.engine('html', require('ejs').renderFile);
6  app.set('view engine', 'html');
7  app.use('/public', express.static(path.join(__dirname, 'public')));
8  app.set('views', path.join(__dirname, '/views'));
9
10 Comment Code
11 app.get('/', (req, res) => {
12   res.render('index', {});
13 });
14
15 Comment Code
16 app.listen(5000, () => {
17   console.log('Servidor rodando em http://localhost:5000');
18 });
```

Nossa index.js modificada agora com estas 4 linhas que são altamente necessárias para realizarem a leitura da pasta views e da pasta public, na pasta views criamos um arquivo index.html e iremos realizar a leitura do mesmo utilizando ejs e na pasta public criamos um arquivo style.css onde ficarão todos os arquivos estáticos de nossa aplicação.

Eu precisei instanciar o path lá em cima do projeto, pois utilizamos para navegar pelos diretórios que criamos.



Estrutura que foi criada e index.html da pasta views logo abaixo:

```
views > index.html > html
1  <!DOCTYPE html>
2  <html Lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title>TODO LIST NODE</title>
7  </head>
8  <body>
9    <h1>TODO LIST NODE</h1>
10 </body>
11 </html>
```

Ao verificarmos a rota agora:



TODOLIST NODE

Repararam que tem uma passagem de parâmetros aqui? Bem aqui é onde a mágica vai acontecer, iremos conseguir deixar tudo dinâmico passando parâmetros ok? vejam esse exemplo de sintaxe ejs.

```
Comment Code
app.get('/', (req, res) => {
  res.render('index', {});
});
Comment Code
```


Passo meu nome como parâmetro e recebo no arquivo index.html da pasta views e vejam como aparece;

Comment Code

```
app.get('/', (req, res) => {  
  res.render('index', { nome: 'Uchoa' });  
});
```

e na index.html eu consigo capturar dessa forma:

```
<!DOCTYPE html>  
<html lang="en">  
  <head>  
    <meta charset="UTF-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>TODO LIST NODE</title>  
  </head>  
  <body>  
    <h1>TODO LIST NODE</h1>  
    <h2>Olá <%= nome %>!</h2>  
  </body>  
</html>
```



Com isso se olharmos agora o nosso localhost:5000



TODOLIST NODE

Olá Uchoa!

Para mais detalhes sobre o ejs -> <https://ejs.co/>

```
index.js x index.html
index.js > ...

1  const express = require('express');
2  const path = require('path');
3  const app = express();
4
5  app.engine('html', require('ejs').renderFile); 11.4k (gzipped: 4.3k)
6  app.set('view engine', 'html');
7  app.use('/public', express.static(path.join(__dirname, 'public')));
8  app.set('views', path.join(__dirname, '/views'));
9
10
11 + let tasks = ['Passear com o dog', 'Ir ao mercado', 'Comprar pão'];
12
13 app.get('/', (req, res) => {
14   res.render('index', {tasksList: tasks});
15 });
16
```

Crio essa modificação onde irei pegar um array de lista de tarefas para manipularmos as informações que contém dentro do array e passo por parâmetro, como vimos anteriormente.

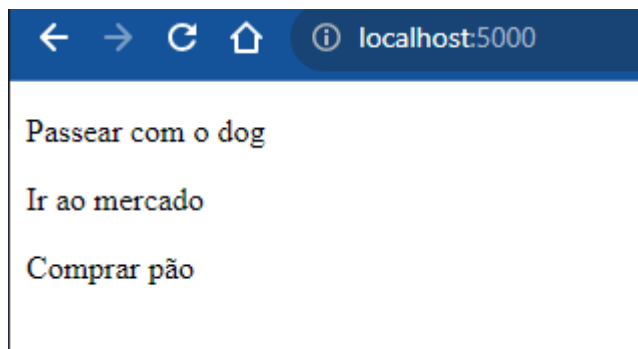
```
index.html > html > body > div.tarefas > ?

1  <!DOCTYPE html>
2  <html Lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title>TODOLIST NODE</title>
7  </head>
8  <body>
9    <div class="tarefas">
10      <% for(let i = 0; i < tasksList.length; i++){ %>
11
12 +
13      <% } %>
14    </div>
15  </body>
16  </html>
```

Criamos agora como estamos verificando um for para realizar a leitura desse nosso array e iremos popular com as informações dentro do bloco. Essa sintaxe como vimos é do EJS.

```
<div class="tarefas">
  <% for(let i = 0; i < tasksList.length; i++){ %>
    <p><%= tasksList[i] %></p>
  <% } %>
</div>
</body>
```

Dessa forma conseguimos receber toda a nossa listagem de tarefas.



Vamos melhorar o nosso front?

Modificando o index.html:

```
<html>

  <head>
    <title>Lista de tarefas NodeJS</title>
    <link href="/public/style.css" rel="stylesheet" />
    <!-- Adicione o Bootstrap CSS -->
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
  </head>

  <body class="container">
```

UniSENAI

```
<body class="container">

  <div class="tarefas">
    <h3>TODO LIST NODE</h3>
    <form class="d-flex" method="post" onsubmit="return validarFormulario()">
      <input type="text" class="form-control" name="tarefa" />
      <input type="submit" name="acao" class="btn btn-primary" value="Inserir" />
    </form>
    <% for(let i = 0; i <= tasksList.length; i++){ %>

      <p><%= tasksList[i] %> | <a href="/deletar/<%= i %>"><i class="bi bi-trash"></i></a></p>

    <% } %>
  </div>
  <!-- Adicione o Bootstrap JS (opcional, se precisar dos recursos JavaScript) -->
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
```

```
</body>
<script>
  function validarFormulario() {
    var tarefa = document.getElementsByName("tarefa")[0].value;
    if (tarefa.trim() === '') {
      alert("Por favor, preencha a tarefa antes de enviar.");
      return false; // Evita o envio do formulário se o campo estiver vazio
    }
    return true; // Permite o envio do formulário se o campo estiver preenchido
  }
</script>
</html>
```

e o css do arquivo style.css

realizamos a importação do bootstrap icons para utilizarmos:

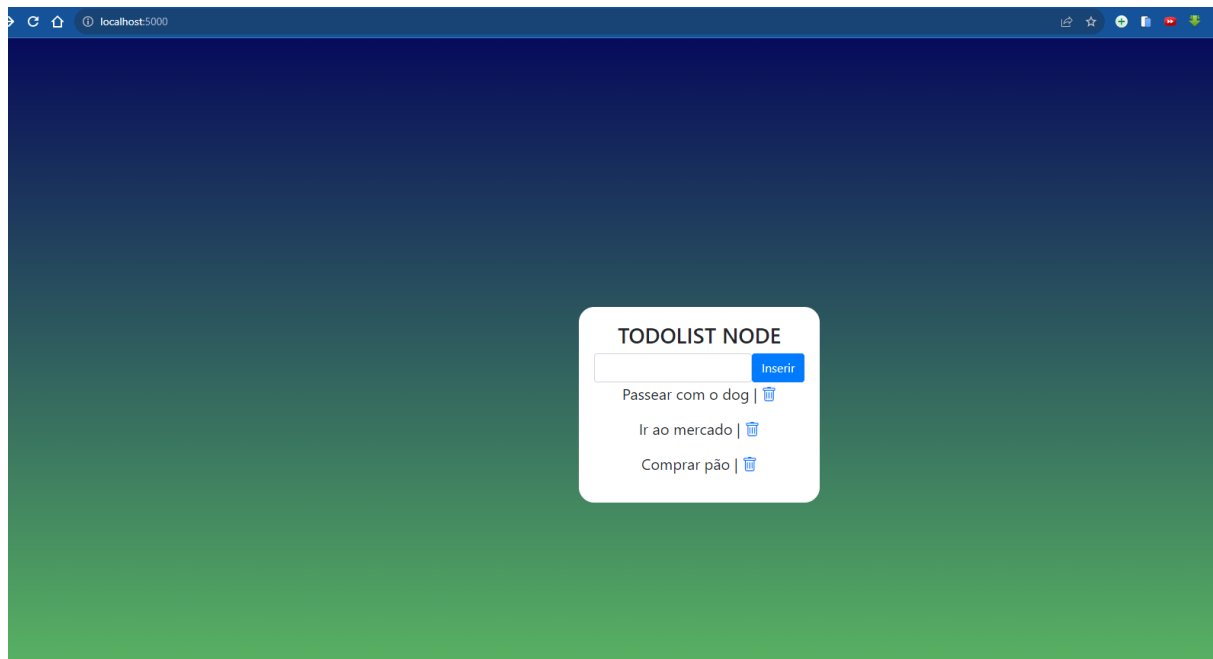
```
index.html style.css x
style.css > *
@import url("https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.2/font/bootstrap-icons.min.css");
Comment Code
*{
```

mas não se preocupe, segue o arquivo css completo para não perderem tempo:

<https://gist.github.com/uchoamaster/29ed0240c01670fc8cc6b936200b604b>

com tudo configurado se formos olhar como ficou o front do projeto, teremos algo parecido com isso:

Unisenai



Se perceberam já criamos a rota de deletar , no index.html, mas não criamos a rota no index.js então vamos criar a rota para deletarmos?

```
</form>
<% for(let i = 0; i <= tasksList.length; i++){ %>

  <p><%= tasksList[i] %> | <a href="/deletar/<%= i %>"><i class="bi bi-trash"></i></a></p>

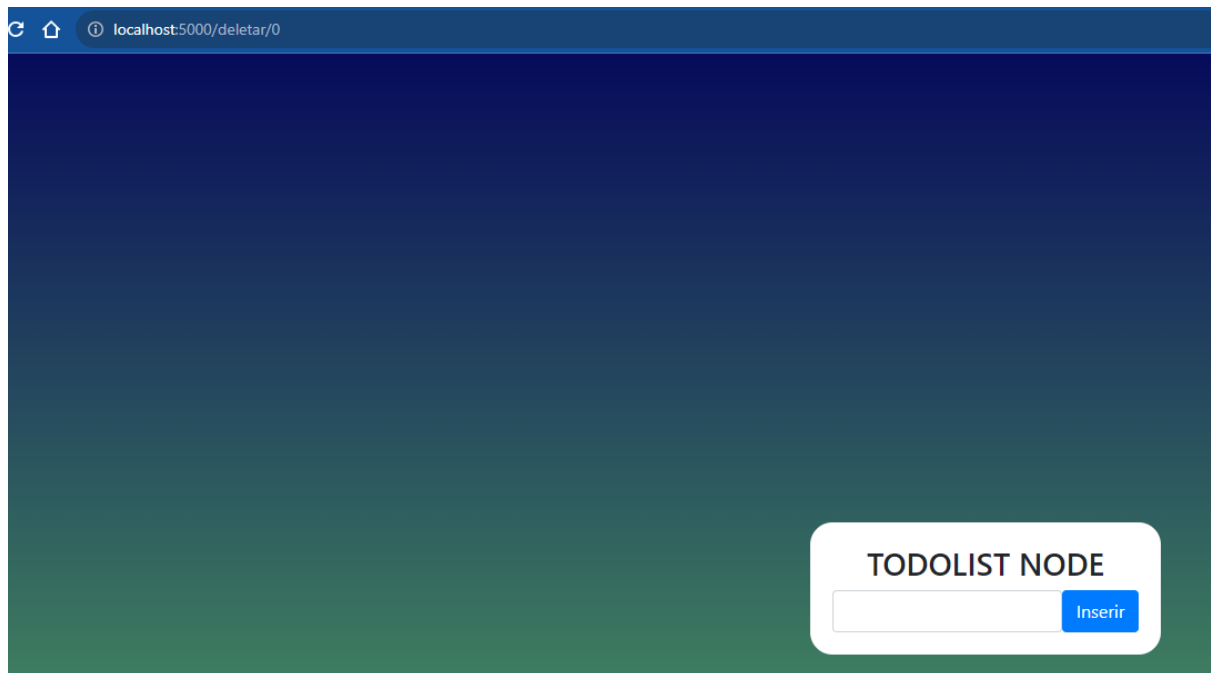
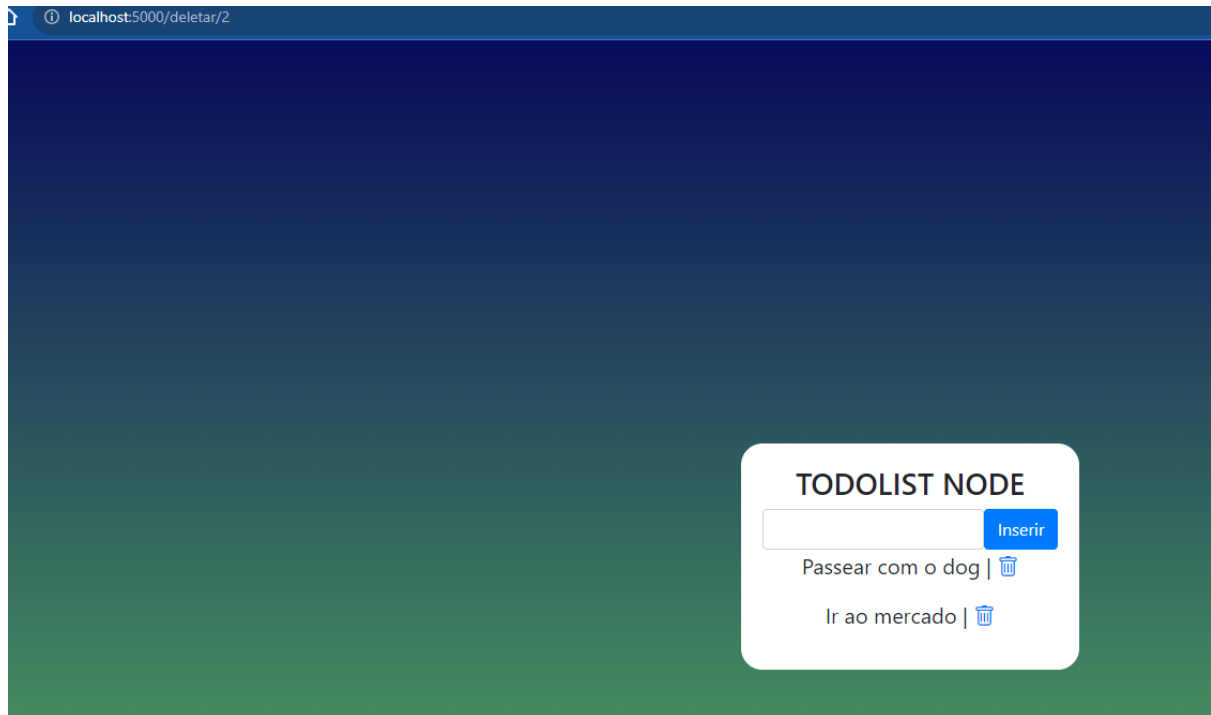
<% } %>
</div>
```

No index.js vamos criar a rota para enfim deletarmos pelo id do objeto.

```
app.get('/deletar/:id',(req,res)=>{
  tasks = tasks.filter(function(val,index){
    if(index !== req.params.id){
      return val;
    }
  })
  res.render('index',{tasksList:tasks});
})
```

UniSENAI

Dessa forma eu garanto conseguir deletar pelo id de cada elemento em sua definitiva posição no array. podemos testar?



UniSENAI

Notem que conseguimos deletar todos os objetos e diferente de uma outra linguagem como PHP ou Java , com Node o array permanece vazio até que seja reiniciado o servidor, pois ele basicamente fica em um Loop infinito aguardando novas instruções.

Também criamos uma função para validar a entrada de dados onde não deixaremos preencher nada em branco;

The screenshot displays a web application interface. At the top, a white modal box contains the text "localhost:5000 diz" and "Por favor, preencha a tarefa antes de enviar." with an "OK" button. Below this, the main content area has a dark blue header and a green body. In the center, a white rounded rectangle titled "TODOLIST NODE" contains a text input field and a blue "Inserir" button.

Agora para criarmos o método de adicionar uma tarefa nova, iremos precisar adicionar alguns códigos para que tudo funcione corretamente.

```
index.js  x  index.html  style.css
index.js > ...

1  const express = require('express');
2  var bodyParser = require('body-parser') 483.9k (gzipped: 211.2k)
3  const path = require('path');
4  const app = express();
5
6  app.engine('html', require('ejs').renderFile); 11.4k (gzipped: 4.3k)
7  app.set('view engine', 'html');
8  app.use('/public', express.static(path.join(__dirname, 'public')));
9  app.set('views', path.join(__dirname, '/views'));
10
11  app.use( bodyParser.json() );           // to support JSON-encoded bodies
12  app.use(bodyParser.urlencoded({       // to support URL-encoded bodies
13    extended: true
14  }));
15
16  let tasks = ['Passear com o dog', 'Ir ao mercado', 'Comprar pão'];
17
18  app.get('/', (req,res)=>{
```

Estes códigos garantem que eu consiga capturar as informações digitadas no corpo do documento e realize conversão de string para objeto e vice e versa para dentro da minha lista e a rota de add é:

```
app.post('/', (req,res)=>{
  tasks.push(req.body.task);
  res.render('index',{tasksList: tasks});
})
```

Este req.body.task , **task** é o nome dado ao nome dentro do input do tipo texto, então certifique-se de ser o mesmo nome ok?

The image shows a web application interface for a to-do list. At the top, the title 'TODOLIST NODE' is displayed in a large, bold, black font. Below the title is a white rectangular area containing a text input field and a blue button labeled 'Inserir'. Underneath the input field, there is a list of three items: 'teste 4', 'teste2', and 'UNISENAI'. Each item is followed by a vertical line and a trash can icon, suggesting a delete function. The entire interface is set against a dark blue background.