

## FUNDAMENTOS NODE JS

### 01 – Criando um projeto node js

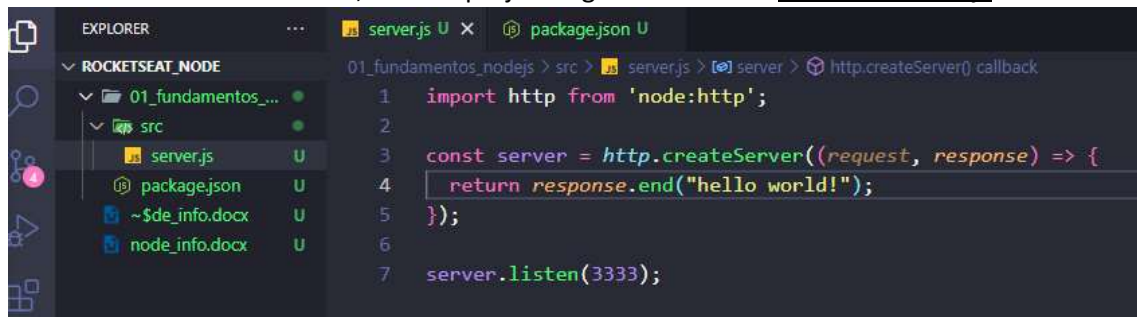
Criando o projeto em node = `npm init -y`

Para fazer o importe coloca o “type”: “module” no package.json



```
1 {
2   "name": "rocketseat_node",
3   "type": "module",
4   "version": "1.0.0",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \\\"Error: no test specified\\\" && exit 1"
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC",
12  "description": ""
13 }
```

Iniciando o servidor em node, rodar o projeto digita no terminal `node src/server.js`



```
1 import http from 'node:http';
2
3 const server = http.createServer((request, response) => {
4   return response.end("hello world!");
5 });
6
7 server.listen(3333);
```

### 02 – Node watch

Bastar digitar no terminal `node --watch src/server.js` ou coloca no package para ele rodar para vc



```
1 {
2   "name": "rocketseat_node",
3   "type": "module",
4   "version": "1.0.0",
5   "main": "index.js",
6   "scripts": {
7     "dev": "node --watch src/server.js"
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC",
12  "description": ""
13 }
```

### 03 – Rotas de criação e listagem métodos HTTP

São GET, POST, PUT, PATCH, DELETE

```

fundamentos_nodejs > src > server.js > ...
1  import http from 'node:http';
2
3  const server = http.createServer((req, res) => {
4    const { method, url } = req;
5
6    if (method === 'GET' && url === '/users') {
7      return res.end("listagem de usuários")
8    }
9
10   if (method === 'POST' && url === '/users') {
11     return res.end("criação de usuários")
12   }
13
14   return res.end("hello world again!");
15 });
16
17 server.listen(3333);

```

#### 04 – Salvando usuários em memória Headers

Retorna um JSON

```

01_fundamentos_nodejs > src > server.js > ...
1  import http from 'node:http';
2
3  const users = [];
4
5  const server = http.createServer((req, res) => {
6    const { method, url } = req;
7
8    if (method === 'GET' && url === '/users') {
9      return res
10        .setHeader('Content-type', 'application/json')
11        .end(JSON.stringify(users))
12    }
13
14    if (method === 'POST' && url === '/users') {
15      users.push({ id: 1, name: 'José', email: "jose@email.com" })
16      return res.end("Criando usuário!")
17    }
18  });
19
20 server.listen(3333);

```

#### 05 – Conhecendo HTTP status codes

São a resposta de retorno da requisição os 200 são os de sucessos, 400 são os de erros da requisição, 500 são os erros de servidor

```

4
5 const server = http.createServer((req, res) => {
6   const { method, url } = req;
7
8   if (method === 'GET' && url === '/users') {
9     return res
10      .setHeader('Content-type', 'application/json')
11      .end(JSON.stringify(users))
12   }
13
14   if (method === 'POST' && url === '/users') {
15     users.push({ id: 1, name: 'José', email: "jose@email.com" });
16     return res.writeHead(201).end()
17   }
18
19   return res.writeHead(404).end();
20 });
21
22 server.listen(3333);

```

## 06 - Criando streams de leitura

Trabalhar com os dados sem precisa que ele esteja carregado por inteiro

```

01_fundamentos_nodejs > streams > fundamentos.js > ...
1 //stdin entrada no terminal LEITURA
2 //stdout saída do terminal SAIDA
3 //process.stdin.pipe(process.stdout)
4
5 import { Readable } from "node:stream";
6
7 class OneToHundredStream extends Readable {
8   index = 1;
9
10  _read() {
11    setTimeout(() => {
12      const i = this.index++;
13
14      if (i > 100) {
15        this.push(null);
16      } else {
17        const buf = Buffer.from(String(i));
18
19        this.push(buf);
20      }
21    }, 1000);
22  }
23 }
24
25 new OneToHundredStream().pipe(process.stdout)

```

## 07 - Stream de escrita e transformação

É uma stream que processa dados

```

25 //new OneToHundredStream().pipe(process.stdout)
26
27 class InverseNumberStream extends Transform {
28   _transform(chunk, encoding, callback) {
29     const transformed = Number(chunk.toString()) * -1;
30     callback(null, Buffer.from(String(transformed)));
31   }
32 }
33
34 class MultiplayByTenStream extends Writable {
35   _write(chunk, encoding, callback) {
36     console.log(Number(chunk.toString()) * 10)
37     callback();
38   }
39 }
40
41 new OneToHundredStream()
42   .pipe(new InverseNumberStream)
43   .pipe(new MultiplayByTenStream);
44

```

## 08 – Aplicando streams no modulo HTTP

```

fundamentais.js  stream-http-server.js U X  fake-upload-to-http-stream.js U
01_fundamentos_nodejs > streams > stream-http-server.js > ...
1  import http from "node:http";
2  import { Transform } from "node:stream";
3
4  class InverseNumberStream extends Transform {
5    _transform(chunk, encoding, callback) {
6      const transformed = Number(chunk.toString()) * -1;
7      callback(null, Buffer.from(String(transformed)));
8    }
9  }
10
11 const server = http.createServer((req, res) => {
12   return req
13     .pipe(new InverseNumberStream())
14     .pipe(res)
15 });
16
17 server.listen(3334)

```

```
01_fundamentos_nodejs > streams > fake-upload-to-http-stream.js > ...
1  import { Readable } from "node:stream";
2
3  class OneToHundredStream extends Readable {
4    index = 1;
5
6    _read() {
7      setTimeout(() => {
8        const i = this.index++;
9
10         if (i > 100) {
11           this.push(null);
12         } else {
13           const buf = Buffer.from(String(i));
14           this.push(buf);
15         }
16       }, 1000);
17     }
18   }
19
20
21   fetch('http://localhost:3334', {
22     method: 'POST',
23     body: new OneToHundredStream(),
24   });
```

## 09 - Consumindo uma stream completa

```
01_fundamentos_nodejs > streams > stream-http-server.js > ...
const server = http.createServer(async (req, res) => {
  const buffers = [];

  for await (const chunk of req) {
    buffers.push(chunk);
  }

  const fullStreamContent = Buffer.concat(buffers).toString();

  console.log(fullStreamContent);

  return res.end(fullStreamContent);
});

server.listen(3334);
```

```
fundamentais.js  stream-http-server.js M  fake-upload-to-http-stream.js M X
01_fundamentos_nodejs > streams > fake-upload-to-http-stream.js > then() callback
19 }
20
21 fetch('http://localhost:3334', {
22   method: 'POST',
23   body: new OneToHundredStream(),
24 }).then(response => {
25   return response.text();
26 }).then(data => {
27   console.log(data);
28 });
```

## 10 - Corpo da requisição em JSON

```
01_fundamentos_nodejs > src > server.js > server > http.createServer() callback
1 import http from 'node:http';
2
3 const users = [];
4
5 const server = http.createServer(async (req, res) => {
6   const { method, url } = req;
7
8   const buffers = [];
9
10  for await (const chunk of req) {
11    buffers.push(chunk);
12  }
13
14  try {
15    req.body = JSON.parse(Buffer.concat(buffers).toString());
16  } catch (error) {
17    req.body = null;
18  }
19
20  if (method === 'GET' && url === '/users') {
21    return res
22      .setHeader('Content-type', 'application/json')
23      .end(JSON.stringify(users))
24  }
25
26  if (method === 'POST' && url === '/users') {
27    const { name, email } = req.body;
28    users.push({ id: 1, name, email });
29    return res.writeHead(201).end()
30  }
31
32  return res.writeHead(404).end();
33 });
34
35 server.listen(3333);
```

## 11 - Entendendo Buffers no Node

É uma representação de espaço na memória do computador, usado para transitar dados rápido, representa os dados em hexadecimal.

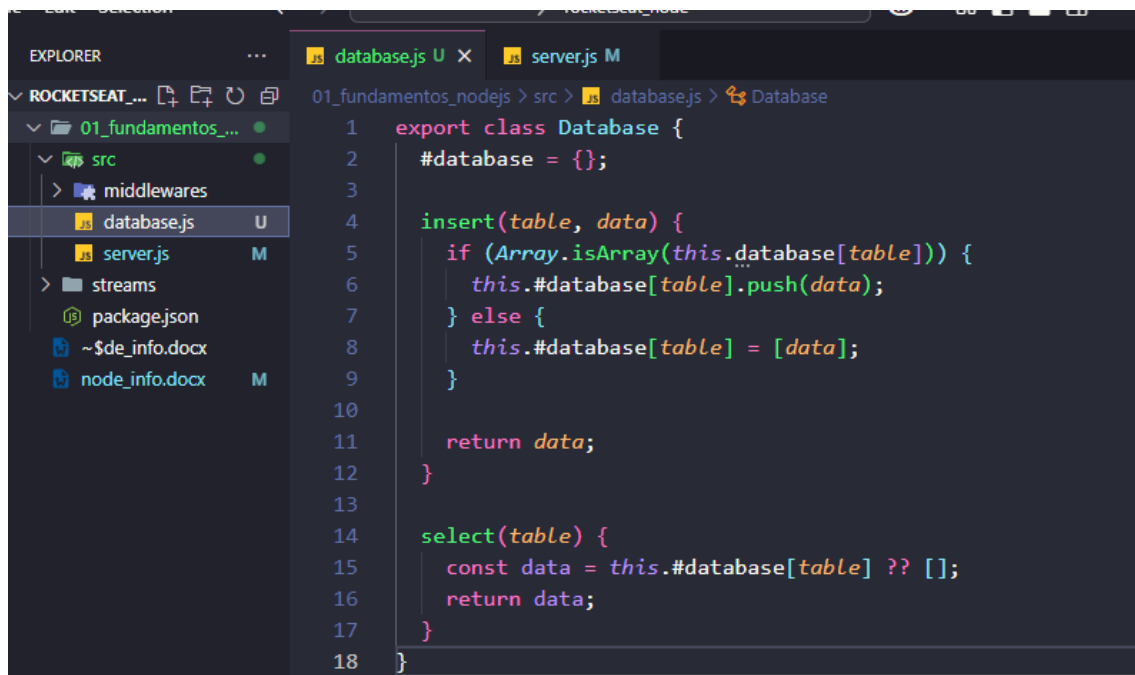
```
01_fundamentos_nodejs > streams > buffer.js > ...
1  const buf = Buffer.from("ok");
2  console.log(buf.toJSON())
```

## 12 - Criando middleware de JSON

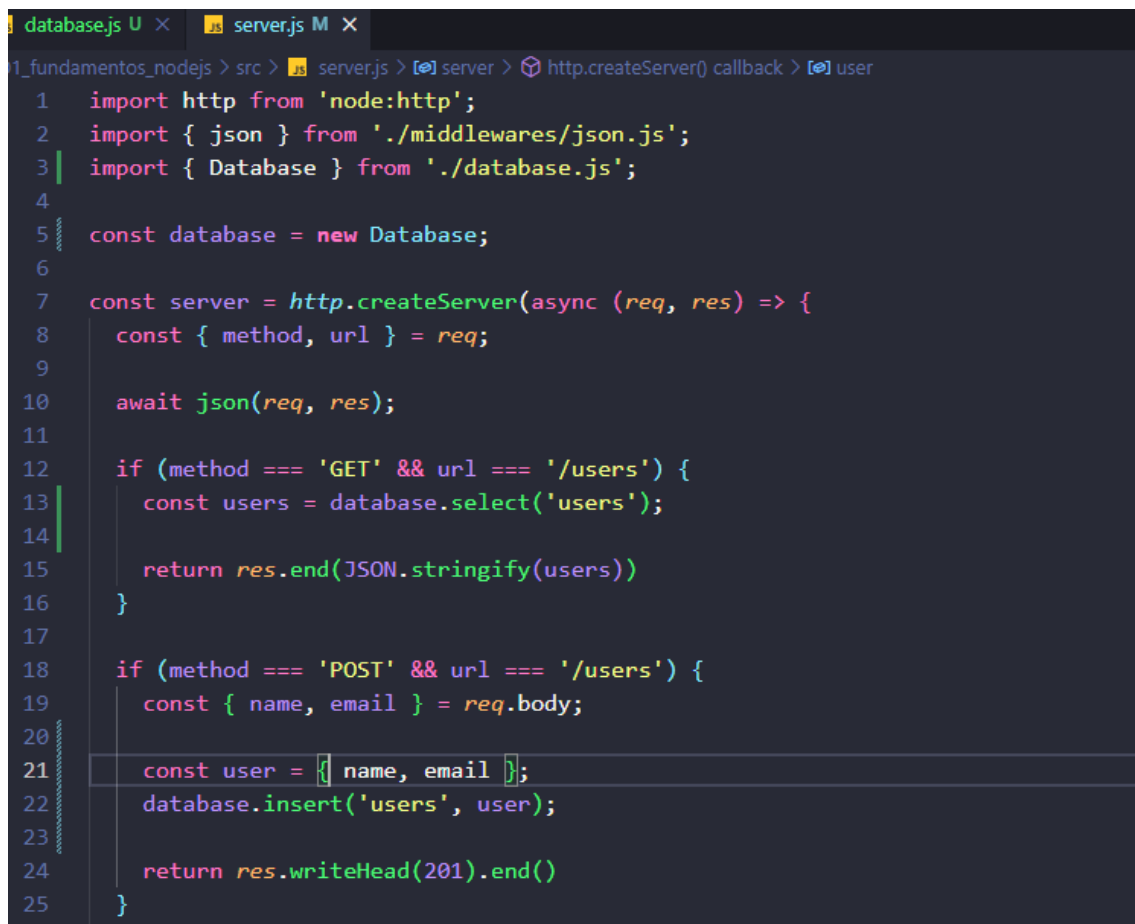
```
EXPLORER 01_fundamentos_nodejs > src > middlewares > json.js > json
1  export async function json(req, res) {
2    const buffers = [];
3
4    for await (const chunk of req) {
5      buffers.push(chunk);
6    }
7
8    try {
9      req.body = JSON.parse(Buffer.concat(buffers).toString());
10   } catch (error) {
11     req.body = null;
12   }
13
14   res.setHeader('Content-type', 'application/json')
15 }
```

```
server.js M  json.js U
01_fundamentos_nodejs > src > server.js > ...
1  import http from 'node:http';
2  import { json } from './middlewares/json.js';
3
4  const users = [];
5
6  const server = http.createServer(async (req, res) => {
7    const { method, url } = req;
8
9    await json(req, res);
10
11    if (method === 'GET' && url === '/users') {
12      return res.end(JSON.stringify(users))
13    }
14
15    if (method === 'POST' && url === '/users') {
```

## 13 - Criando banco de dados JSON



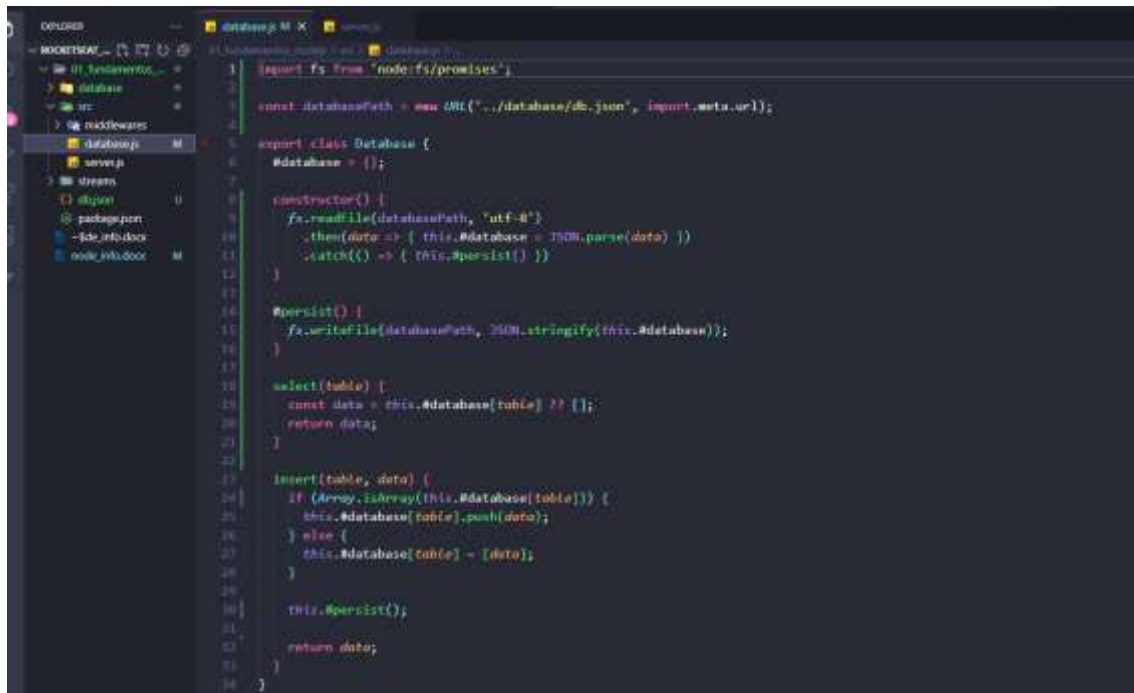
```
1 export class Database {
2   #database = {};
3
4   insert(table, data) {
5     if (Array.isArray(this.#database[table])) {
6       this.#database[table].push(data);
7     } else {
8       this.#database[table] = [data];
9     }
10
11     return data;
12   }
13
14   select(table) {
15     const data = this.#database[table] ?? [];
16     return data;
17   }
18 }
```



```
1 import http from 'node:http';
2 import { json } from './middlewares/json.js';
3 import { Database } from './database.js';
4
5 const database = new Database;
6
7 const server = http.createServer(async (req, res) => {
8   const { method, url } = req;
9
10   await json(req, res);
11
12   if (method === 'GET' && url === '/users') {
13     const users = database.select('users');
14
15     return res.end(JSON.stringify(users))
16   }
17
18   if (method === 'POST' && url === '/users') {
19     const { name, email } = req.body;
20
21     const user = { name, email };
22     database.insert('users', user);
23
24     return res.writeHead(201).end()
25   }
26 }
```

## 14 - Persistindo banco de dados





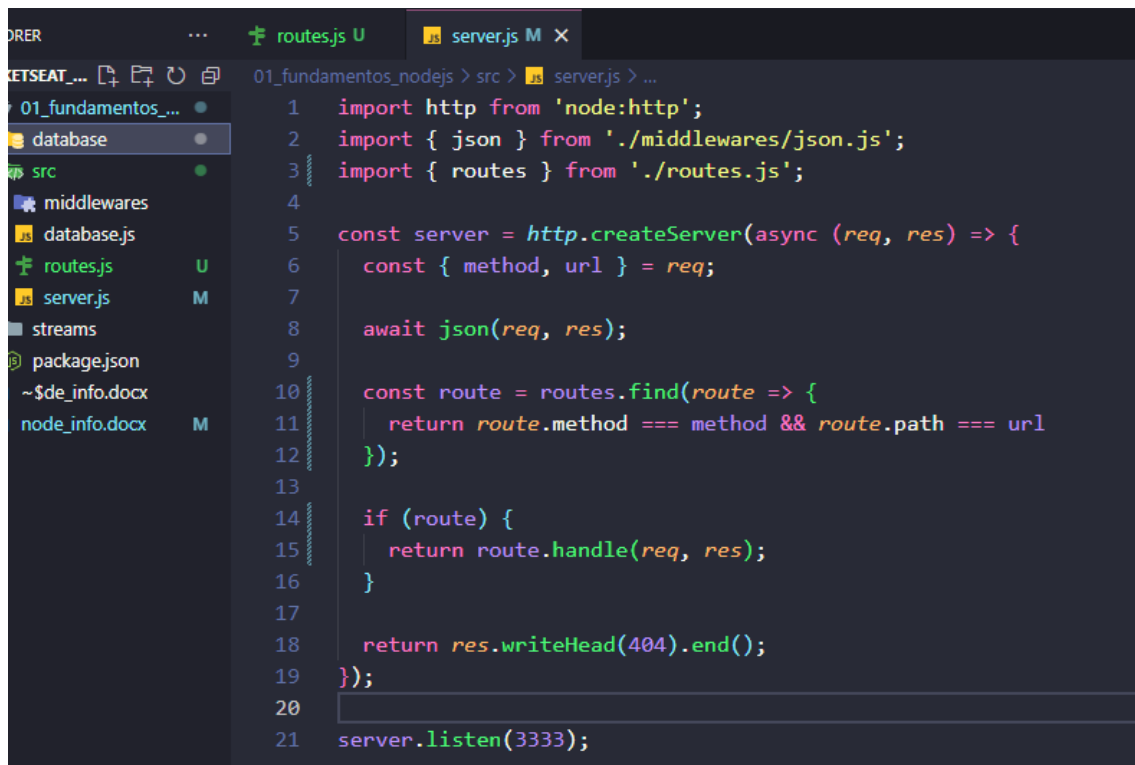
```
1 import fs from "node:fs/promises";
2
3 const databasePath = new URL("../database/db.json", import.meta.url);
4
5 export class Database {
6   #database = {};
7
8   constructor() {
9     fs.readFile(databasePath, "utf-8")
10       .then(data => { this.#database = JSON.parse(data) })
11       .catch(() => { this.#persist() });
12   }
13
14   #persist() {
15     fs.writeFile(databasePath, JSON.stringify(this.#database));
16   }
17
18   select(table) {
19     const data = this.#database[table] ?? [];
20     return data;
21   }
22
23   insert(table, data) {
24     if (Array.isArray(this.#database[table])) {
25       this.#database[table].push(data);
26     } else {
27       this.#database[table] = [data];
28     }
29
30     this.#persist();
31
32     return data;
33   }
34 }
```

## 15 - Criando ID único com universal UUID

Gera um id aleatório com vários caracteres

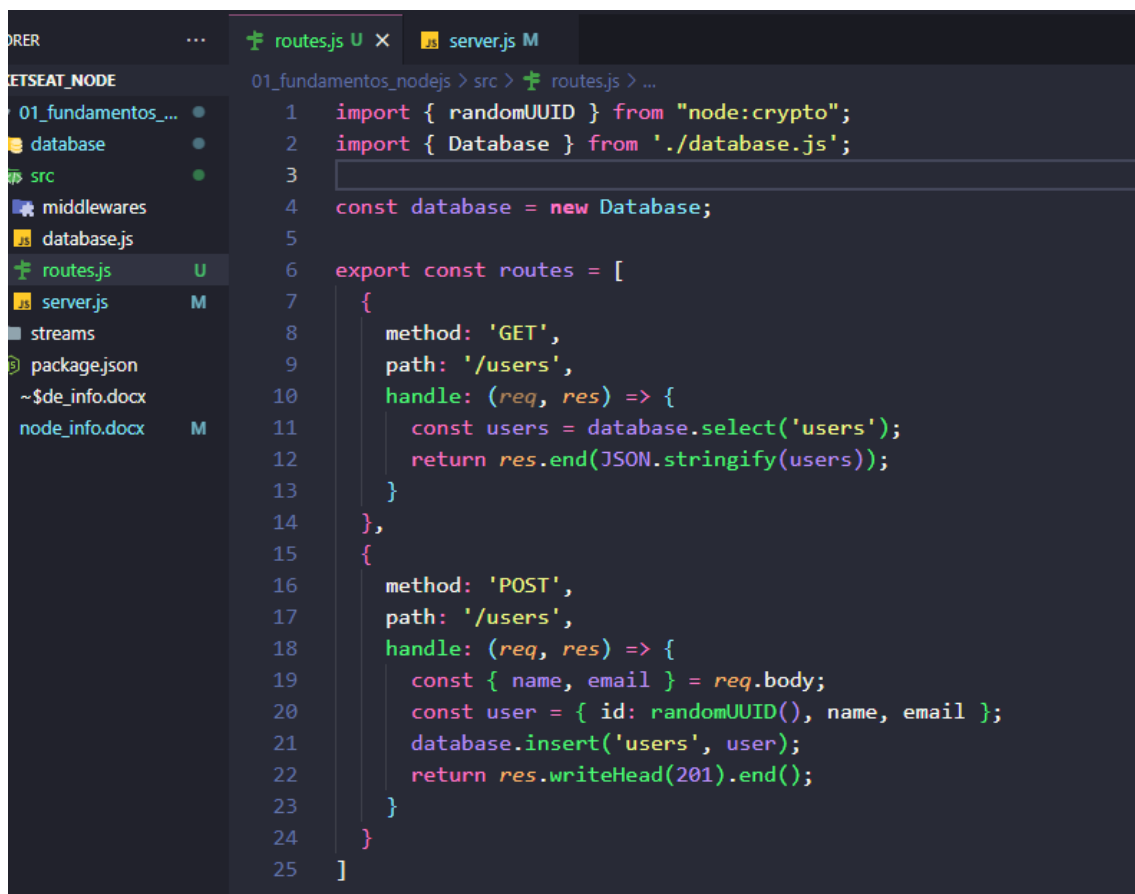
```
3 import { Database } from './database.js';
4 import { randomUUID } from "node:crypto";
5
6 // ...
7
8 // ...
9
10 // ...
11
12 // ...
13
14 // ...
15
16 // ...
17
18 // ...
19
20 const { name, email } = req.body;
21
22 // ...
23
24 const user = { id: randomUUID(), name, email };
25 database.insert('users', user);
```

## 16 - Separando rotas da aplicação



The screenshot shows a VS Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with files like database.js, routes.js, server.js, and package.json. The code editor displays the content of server.js, which imports http, json, and routes, creates a server, and listens on port 3333.

```
01_fundamentos_nodejs > src > server.js > ...
1 import http from 'node:http';
2 import { json } from './middlewares/json.js';
3 import { routes } from './routes.js';
4
5 const server = http.createServer(async (req, res) => {
6   const { method, url } = req;
7
8   await json(req, res);
9
10  const route = routes.find(route => {
11    return route.method === method && route.path === url
12  });
13
14  if (route) {
15    return route.handle(req, res);
16  }
17
18  return res.writeHead(404).end();
19 });
20
21 server.listen(3333);
```



The screenshot shows a VS Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with files like database.js, routes.js, server.js, and package.json. The code editor displays the content of routes.js, which imports randomUUID and Database, creates a database instance, and exports an array of routes for GET and POST methods on the /users path.

```
01_fundamentos_nodejs > src > routes.js > ...
1 import { randomUUID } from "node:crypto";
2 import { Database } from './database.js';
3
4 const database = new Database;
5
6 export const routes = [
7   {
8     method: 'GET',
9     path: '/users',
10    handle: (req, res) => {
11      const users = database.select('users');
12      return res.end(JSON.stringify(users));
13    }
14  },
15   {
16     method: 'POST',
17     path: '/users',
18     handle: (req, res) => {
19       const { name, email } = req.body;
20       const user = { id: randomUUID(), name, email };
21       database.insert('users', user);
22       return res.writeHead(201).end();
23     }
24   }
25 ]
```

## 17 - Route e Query parameters

```
1 export function buildRoutePath(path) {  
  Test Regex...  
  2   const routeParametersRegex = /:([a-zA-Z]+)/g;  
  3 }
```

```
7 export const routes = [  
8   {  
9     method: 'GET',  
10    path: buildRoutePath('/users'),  
11    handle: (req, res) => {
```

## 18 – Rotas com parâmetros

```
export function buildRoutePath(path) {  
  Test Regex...  
  const routeParametersRegex = /:([a-zA-Z]+)/g;  
  const pathWithParams = path.replaceAll(routeParametersRegex, '(<$1>[a-z0-9\\-\\_]+)');  
  const pathRegex = new RegExp(`^${pathWithParams}$`);  
  return pathRegex;  
}
```

```
5 const server = http.createServer(async (req, res) => {  
9  
10   const route = routes.find(route => {  
11     return route.method === method && route.path.test(url)  
12   });  
13  
14   if (route) {  
15     const routeParams = req.url.match(route.path);  
16  
17  
18  
19     return route.handle(req, res);  
20   }
```

## 19 – Remoção de registros

```
routes.js M  server.js M  database.js M
01_fundamentos_nodejs > src > server.js > http.createServer() callback
5  const server = http.createServer(async (req, res) => {
13
14    if (route) {
15      const routeParams = req.url.match(route.path);
16
17      req.params = { ...routeParams.groups };
18
19      return route.handle(req, res);
20    }
21
22    return res.writeHead(404).end();
23  });
```

```
routes.js M  database.js M
_fundamentos_nodejs > src > database.js > Database > delete > rowIndex
5  export class Database {
33
34
35    delete(table, id) {
36      const rowIndex = this.#database[table].findIndex(row => row.id === id);
37
38      if (rowIndex > -1) {
39        this.#database[table].splice(rowIndex, 1)
40        this.#persist();
41      }
42    }
43  }
```

```
routes.js M  database.js M
01_fundamentos_nodejs > src > routes.js > routes
7  export const routes = [
25    },
26    {
27      method: 'DELETE',
28      path: buildRoutePath('/users/:id'),
29      handle: (req, res) => {
30        const { id } = req.params;
31        database.delete('users', id);
32        return res.writeHead(204).end();
33      }
34    }
35  ]
```

## 20 – Atualização de registros

```
routes.js M database.js M
fundamentos_nodejs > src > database.js > Database > update
5 export class Database {
3   }
4
5   update(table, id, data) {
6     const rowIndex = this.#database[table].findIndex(row => row.id === id);
7
8     if (rowIndex > -1) {
9       this.#database[table][rowIndex] = { id, ...data };
10      this.#persist();
11    }
12  }
13}
```

```
routes.js M X database.js M
01_fundamentos_nodejs > src > routes.js > routes
7 export const routes = [
26   {
27     method: 'PUT',
28     path: buildRoutePath('/users/:id'),
29     handle: (req, res) => {
30       const { id } = req.params;
31       const { name, email } = req.body;
32       database.uptade('users', id, { name, email });
33       return res.writeHead(204).end();
34     }
35   },
```

## 21 – Capturando query parameters

```
routes.js database.js build-route-path.js M X server.js extract-query-params.js U
01_fundamentos_nodejs > src > utils > build-route-path.js > buildRoutePath > pathRegex
1 export function buildRoutePath(path) {
  Test Regex...
2   const routeParametersRegex = /:([a-zA-Z]+)/g;
3   const pathWithParams = path.replaceAll(routeParametersRegex, '(<$1>[a-z0-9\\-_]+)');
4
5   const pathRegex = new RegExp(`^${pathWithParams}(?<query>\\?(.*)?)?$`);
6   return pathRegex;
7 }
```

```
Edit Selection View Go Run Terminal Help
01_fundamentos_nodejs > src > utils > extract-query-params.js > extractQueryParams > extractQueryParams
1 export function extractQueryParams(query) {
2   return query.substr(1).split('&').reduce((queryParams, param) => {
3     const [key, value] = param.split('=');
4     queryParams[key] = value;
5     return queryParams;
6   }, {});
7 }
```

```
routes.js database.js server.js X extract-query-params.js U
1_fundamentos_nodejs > src > server.js > ...
6 const server = http.createServer(async (req, res) => {
16   const routeParams = req.url.match(route.path);
17
18   const { query, ...params } = routeParams.groups;
19
20   req.params = params;
21   req.query = query ? extractQueryParams(query) : {};
22
23   return route.handle(req, res);
24 }
```

```
routes.js M database.js M X server.js M
1_fundamentos_nodejs > src > database.js > Database > select
5 export class Database {
18   select(table, search) {
19     let data = this.#database[table] ?? [];
20
21     if (search) {
22       data = data.filter(row => {
23         return Object.entries(search).some(([key, value]) => {
24           return row[key].toLowerCase().includes(value.toLowerCase());
25         })
26       })
27     }
28
29     return data;
30   }
31 }
```

```
routes.js M X database.js M server.js M
01_fundamentos_nodejs > src > routes.js > routes > handle > users
7 export const routes = [
9   method: 'GET',
10   path: buildRoutePath('/users'),
11   handle: (req, res) => {
12     const { search } = req.query;
13     const users = database.select('users', search ? { name: search, email:
14       search } : null);
15     return res.end(JSON.stringify(users));
16   },
17 }
```