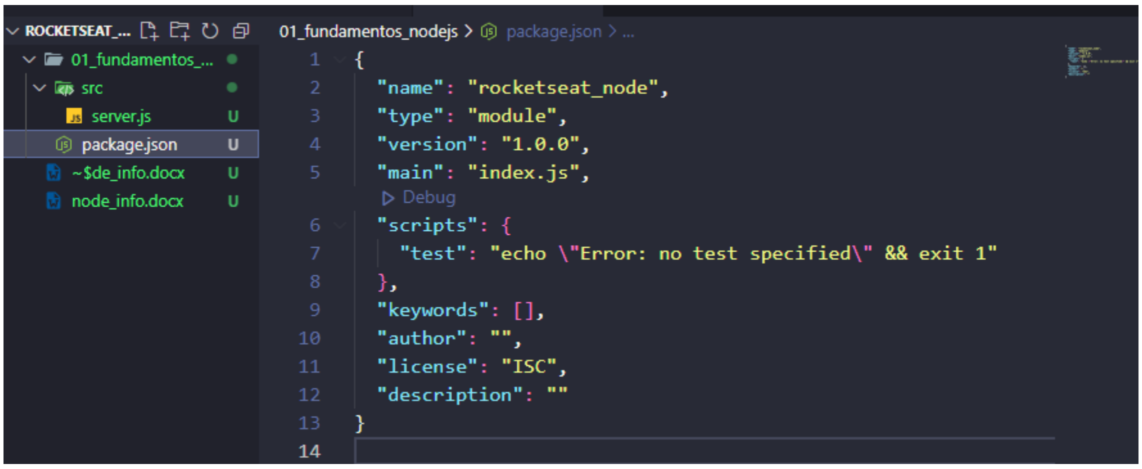
FUNDAMENTOS NODE JS

01 – Criando um projeto node js

Criando o projeto em node = npm init -y

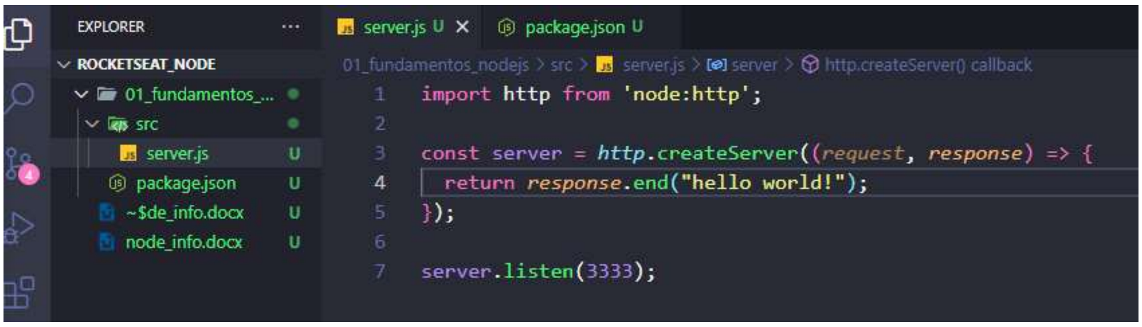
Para fazer o importe coloca o “type”: “module” no package.json



```
ROCKETSEAT_...
  01_fundamentos_...
    src
      server.js        U
    package.json       U
  ~$de_info.docx       U
  node_info.docx       U
```

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

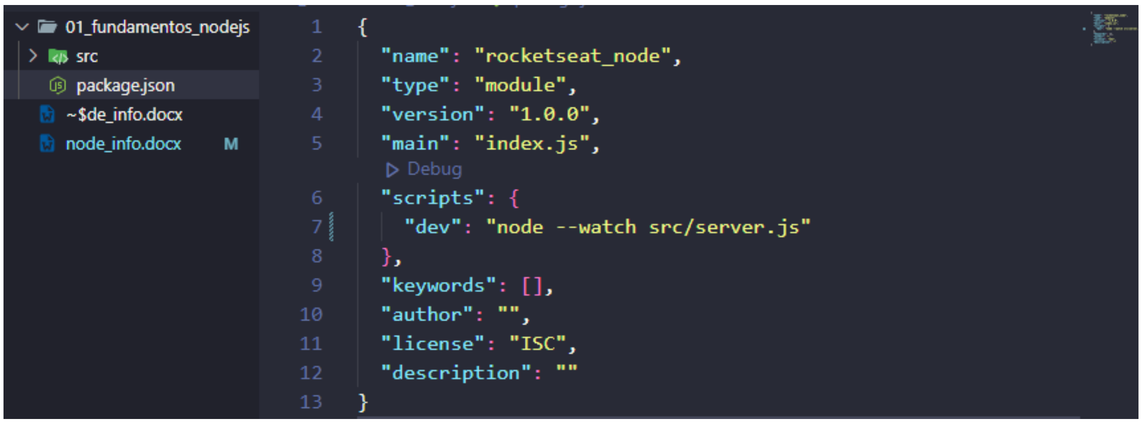
Iniciando o servidor em node, rodar o projeto digita no terminal node src/server.js



```js
import http from 'node:http';

const server = http.createServer((request, response) => {
  return response.end("hello world!");
});

server.listen(3333);
```

02 – Node watch

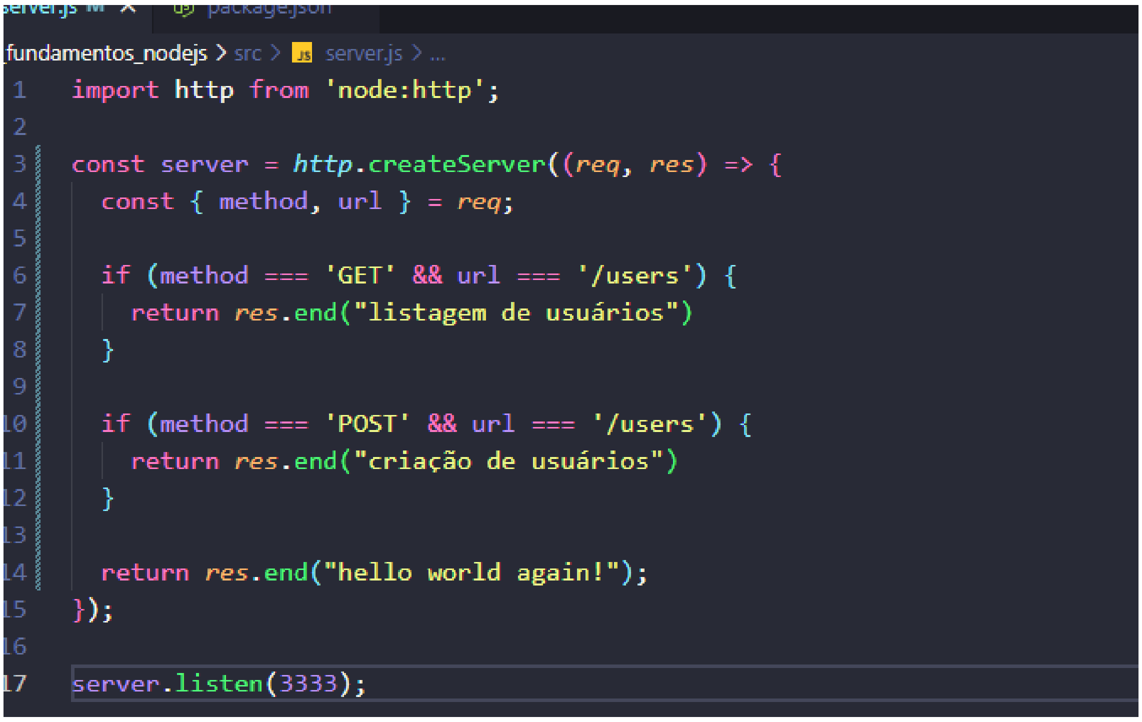
Bastas digitar no terminal node –watch src/server.js ou coloca no package para ele rodar para vc



```json
{
  "name": "rocketseat_node",
  "type": "module",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "dev": "node --watch src/server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

03 – Rotas de criação e listagem métodos HTTP

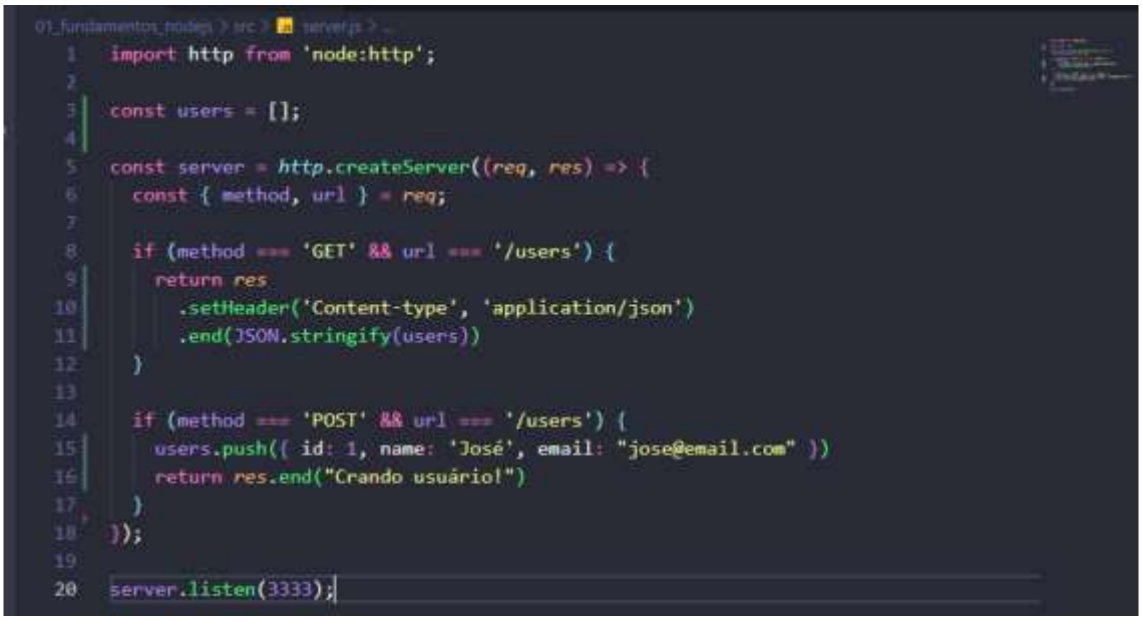
São GET, POST, PUT, PATCH, DELETE



```js
import http from 'node:http';

const server = http.createServer((req, res) => {
  const { method, url } = req;

  if (method === 'GET' && url === '/users') {
    return res.end("listagem de usuários")
  }

  if (method === 'POST' && url === '/users') {
    return res.end("criação de usuários")
  }

  return res.end("hello world again!");
});

server.listen(3333);
```

04 – Salvando usuários em memória Headers

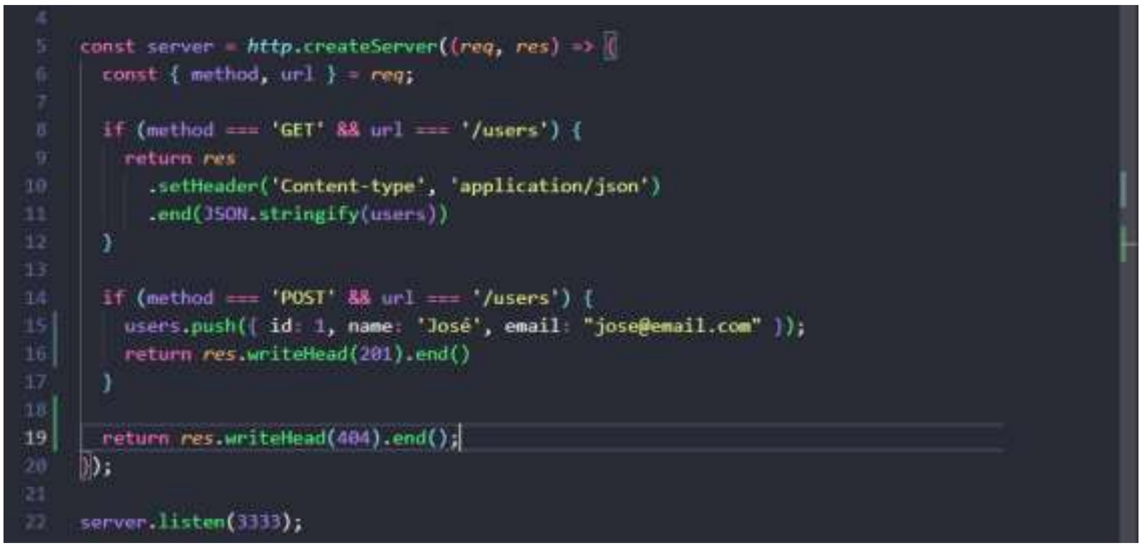
Retorna um JSON



```js
import http from 'node:http';

const users = [];

const server = http.createServer((req, res) => {
  const { method, url } = req;

  if (method === 'GET' && url === '/users') {
    return res
      .setHeader('Content-type', 'application/json')
      .end(JSON.stringify(users))
  }

  if (method === 'POST' && url === '/users') {
    users.push({ id: 1, name: 'José', email: "jose@email.com" })
    return res.end("Crando usuário!")
  }
});

server.listen(3333);
```

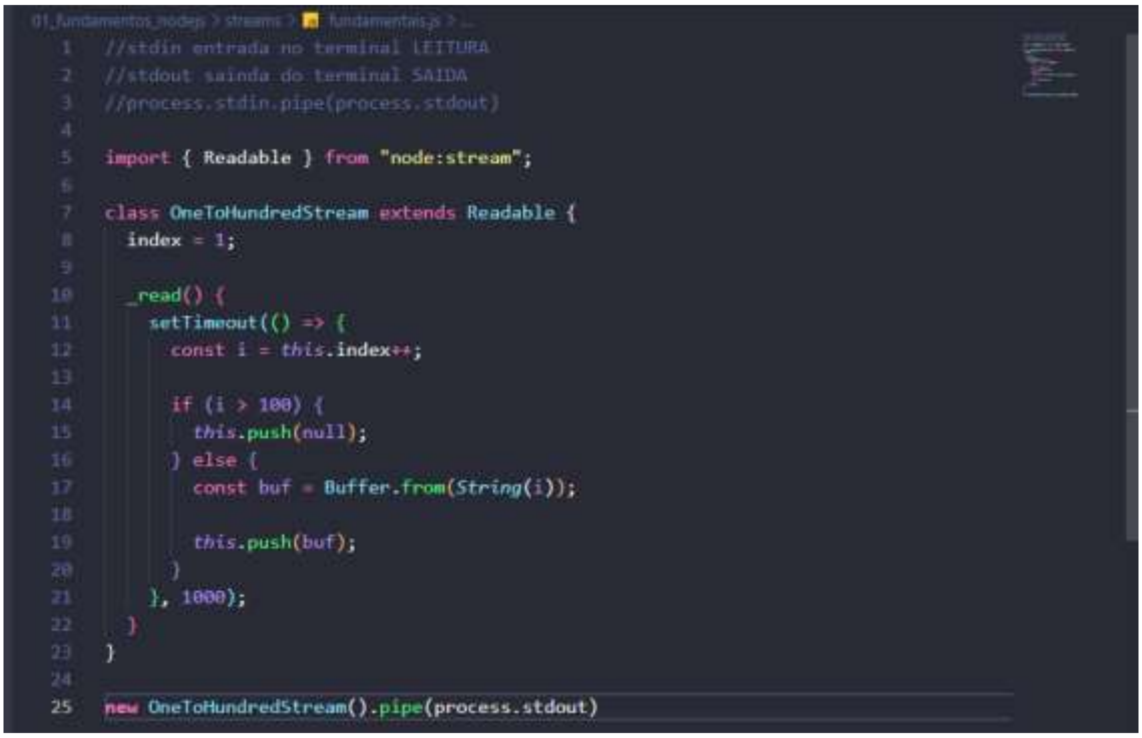
05 – Conhecendo HTTP status codes

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


```
const server = http.createServer((req, res) => {
  const { method, url } = req;

  if (method === 'GET' && url === '/users') {
    return res
      .setHeader('Content-type', 'application/json')
      .end(JSON.stringify(users))
  }

  if (method === 'POST' && url === '/users') {
    users.push({ id: 1, name: 'José', email: "jose@email.com" });
    return res.writeHead(201).end()
  }

  return res.writeHead(404).end();
});

server.listen(3333);
```

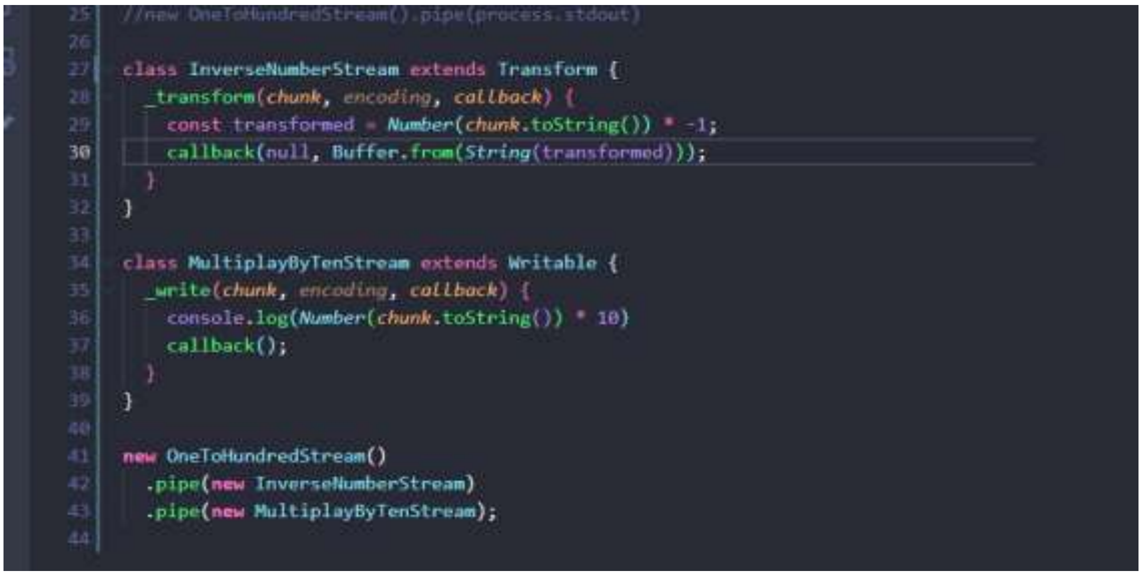
06 - Criando streams de leitura

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


01_fundamentos_nodejs > streams > fundamentals.js > ...
```
//stdin entrada no terminal LEITURA
//stdout saida do terminal SAIDA
//process.stdin.pipe(process.stdout)

import { Readable } from "node:stream";

class OneToHundredStream extends Readable {
  index = 1;

  _read() {
    setTimeout(() => {
      const i = this.index++;

      if (i > 100) {
        this.push(null);
      } else {
        const buf = Buffer.from(String(i));

        this.push(buf);
      }
    }, 1000);
  }
}

new OneToHundredStream().pipe(process.stdout)
```

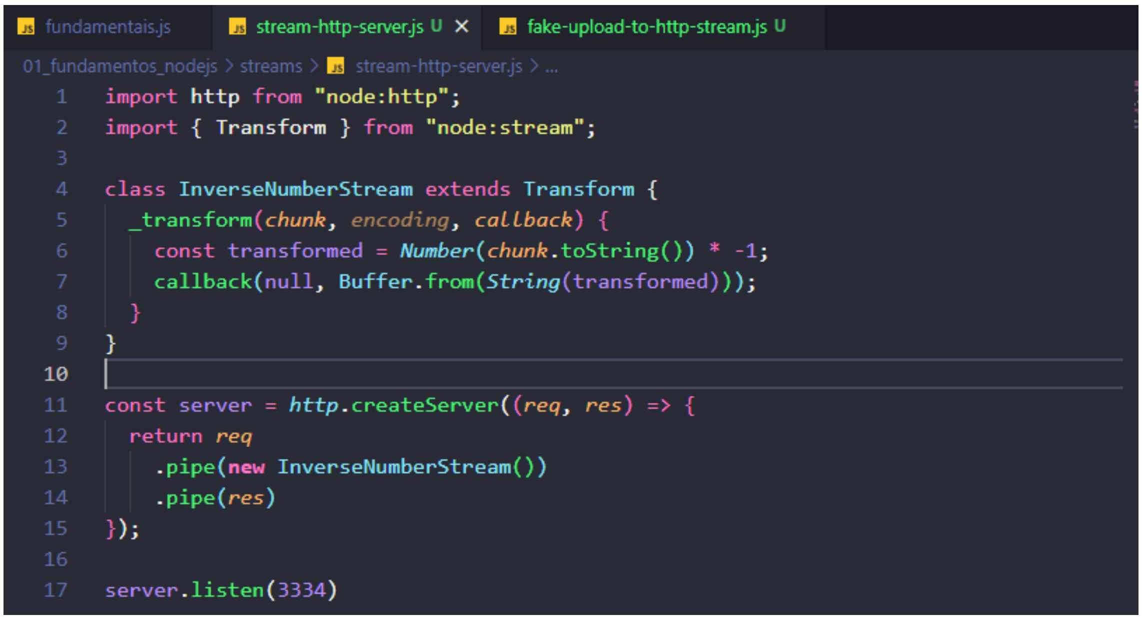
07 - Stream de escrita e transformação

É uma stream que processa dados



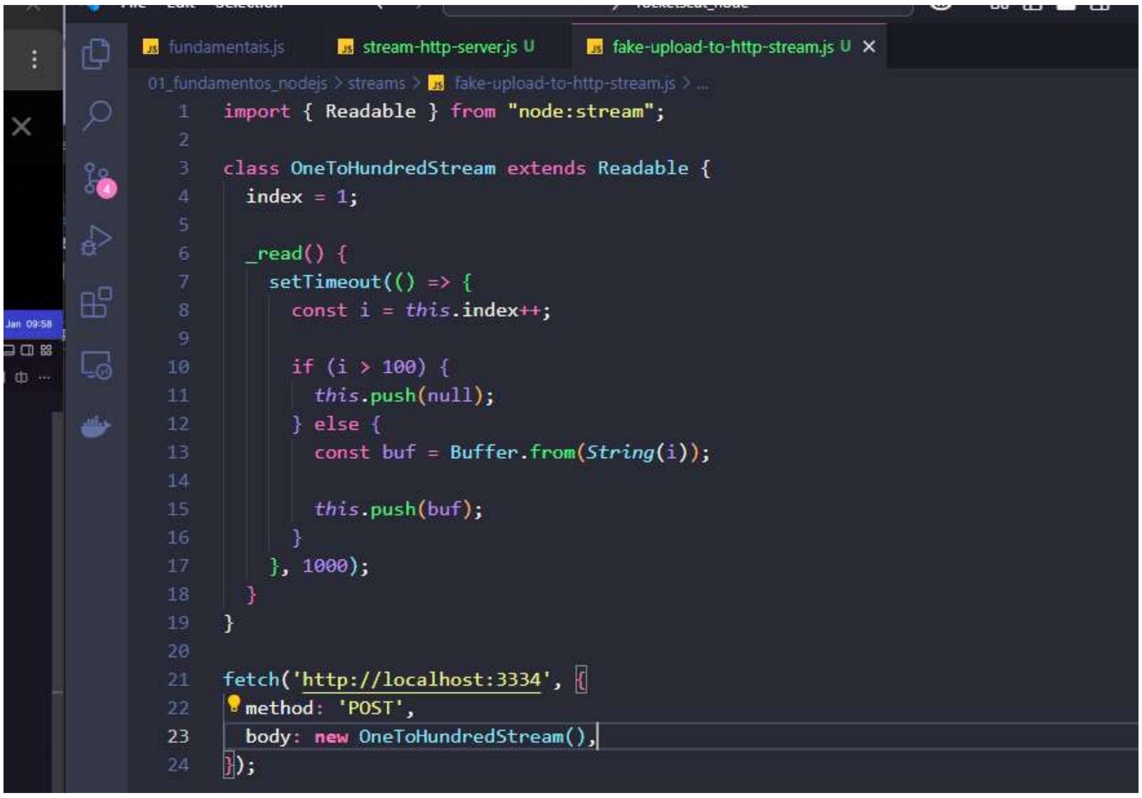
```javascript
//new OneToHundredStream().pipe(process.stdout)

class InverseNumberStream extends Transform {
  _transform(chunk, encoding, callback) {
    const transformed = Number(chunk.toString()) * -1;
    callback(null, Buffer.from(String(transformed)));
  }
}

class MultiplayByTenStream extends Writable {
  _write(chunk, encoding, callback) {
    console.log(Number(chunk.toString()) * 10)
    callback();
  }
}

new OneToHundredStream()
  .pipe(new InverseNumberStream)
  .pipe(new MultiplayByTenStream);
```

08 – Aplicando streams no modulo HTTP



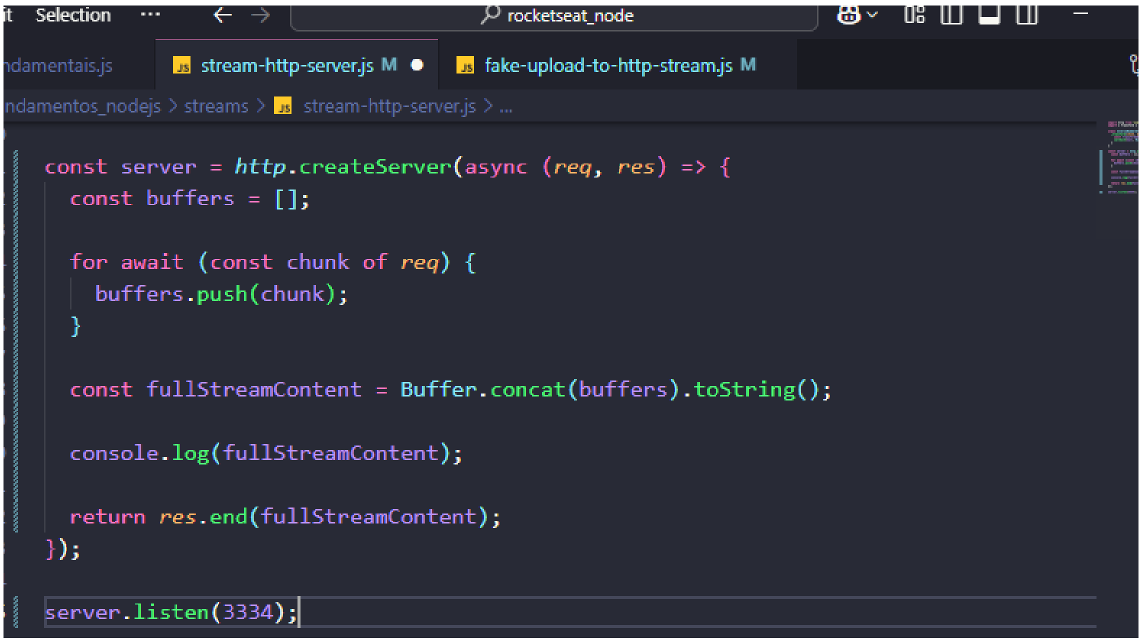
fundamentais.js
stream-http-server.js U
fake-upload-to-http-stream.js U
01_fundamentos_nodejs > streams > stream-http-server.js > ...

```javascript
import http from "node:http";
import { Transform } from "node:stream";

class InverseNumberStream extends Transform {
  _transform(chunk, encoding, callback) {
    const transformed = Number(chunk.toString()) * -1;
    callback(null, Buffer.from(String(transformed)));
  }
}

const server = http.createServer((req, res) => {
  return req
    .pipe(new InverseNumberStream())
    .pipe(res)
});

server.listen(3334)
```

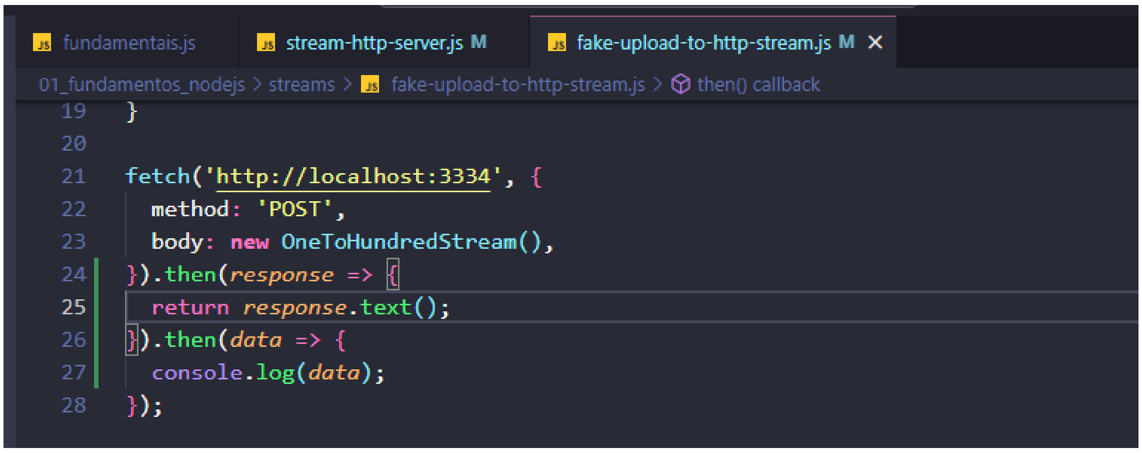

fundamentais.js
stream-http-server.js U
fake-upload-to-http-stream.js U
01_fundamentos_nodejs > streams > fake-upload-to-http-stream.js > ...

```javascript
import { Readable } from "node:stream";

class OneToHundredStream extends Readable {
  index = 1;

  _read() {
    setTimeout(() => {
      const i = this.index++;

      if (i > 100) {
        this.push(null);
      } else {
        const buf = Buffer.from(String(i));

        this.push(buf);
      }
    }, 1000);
  }
}

fetch('http://localhost:3334', {
  method: 'POST',
  body: new OneToHundredStream(),
});
```

09 - Consumindo uma stream completa

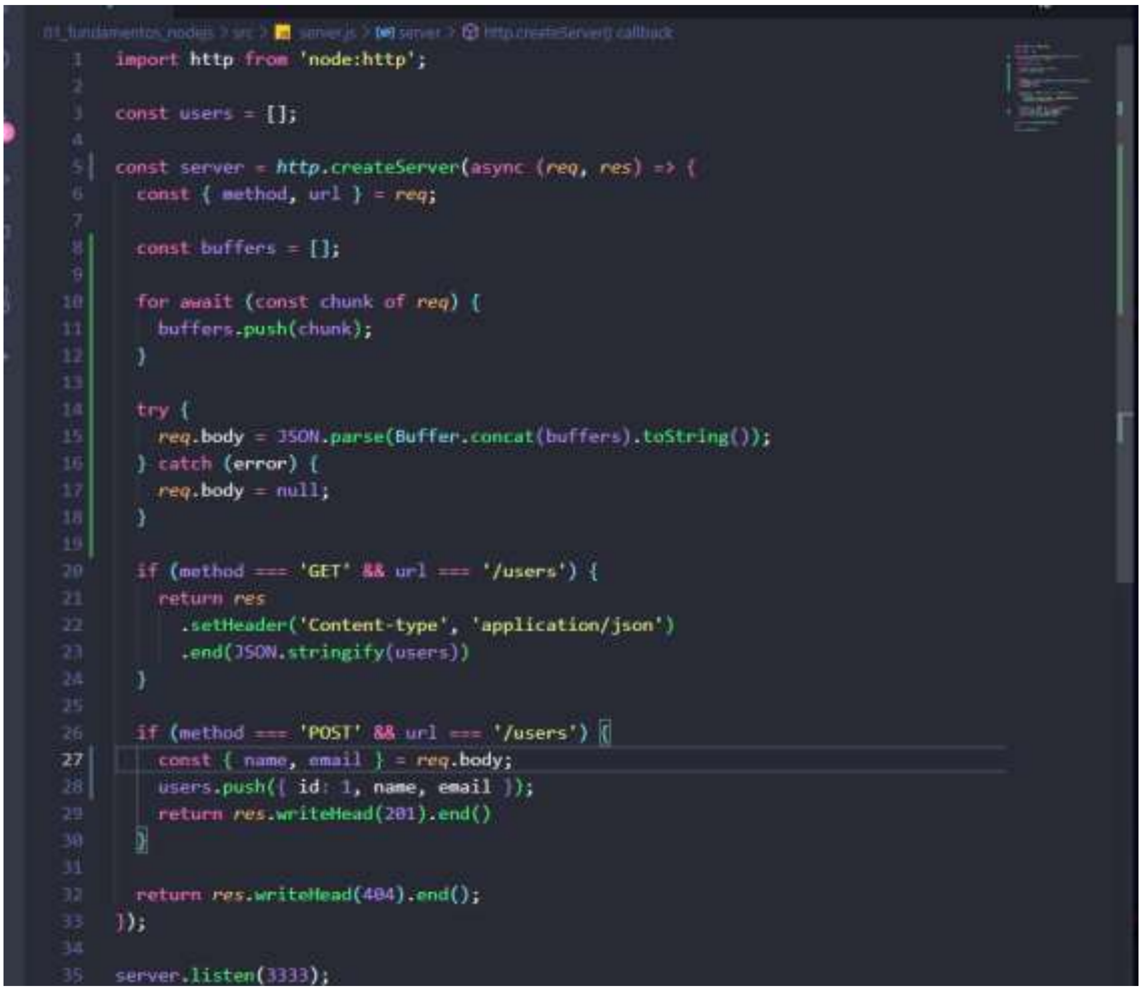

Selection
rocketseat_node
ndamentais.js
stream-http-server.js M
fake-upload-to-http-stream.js M
ndamentos_nodejs > streams > stream-http-server.js > ...

```javascript
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
01_fundamentos_nodejs > streams > fake-upload-to-http-stream.js > then() callback

```
}

fetch('http://localhost:3334', {
  method: 'POST',
  body: new OneToHundredStream(),
}).then(response => {
  return response.text();
}).then(data => {
  console.log(data);
});
```

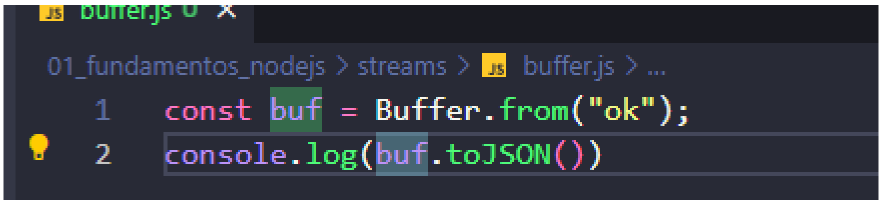
10 - Corpo da requisição em JSON


01_fundamentos_nodejs > src > server.js > server > http.createServer() callback

```
import http from 'node:http';

const users = [];

const server = http.createServer(async (req, res) => {
  const { method, url } = req;

  const buffers = [];

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

  try {
    req.body = JSON.parse(Buffer.concat(buffers).toString());
  } catch (error) {
    req.body = null;
  }

  if (method === 'GET' && url === '/users') {
    return res
      .setHeader('Content-type', 'application/json')
      .end(JSON.stringify(users))
  }

  if (method === 'POST' && url === '/users') {
    const { name, email } = req.body;
    users.push({ id: 1, name, email });
    return res.writeHead(201).end()
  }

  return res.writeHead(404).end();
});

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.

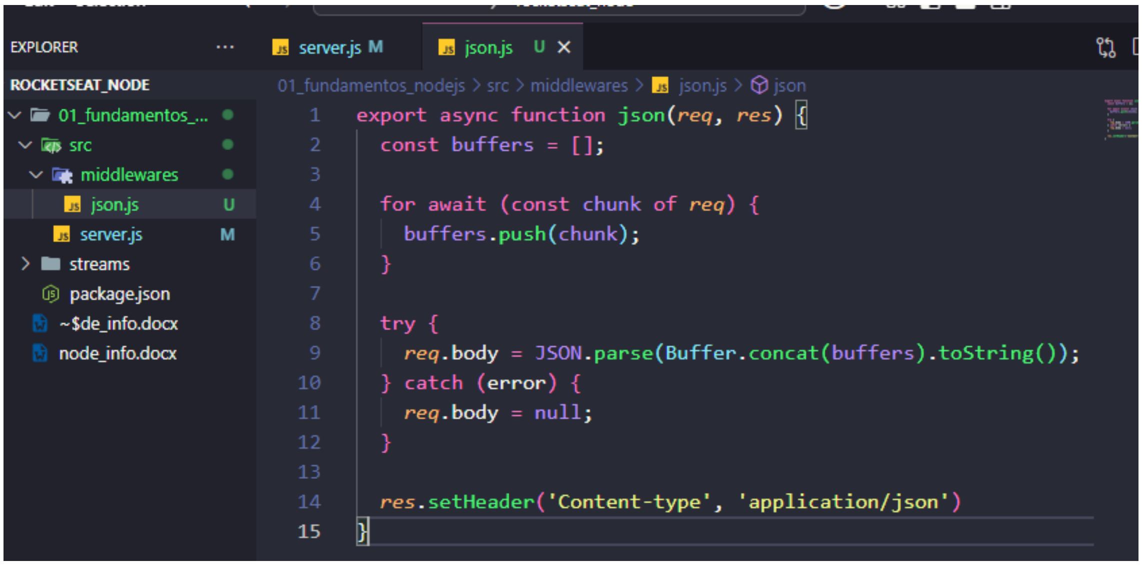


buffer.js U

01_fundamentos_nodejs > streams > buffer.js > ...

```js
const buf = Buffer.from("ok");
console.log(buf.toJSON())
```

12 - Criando middleware de JSON



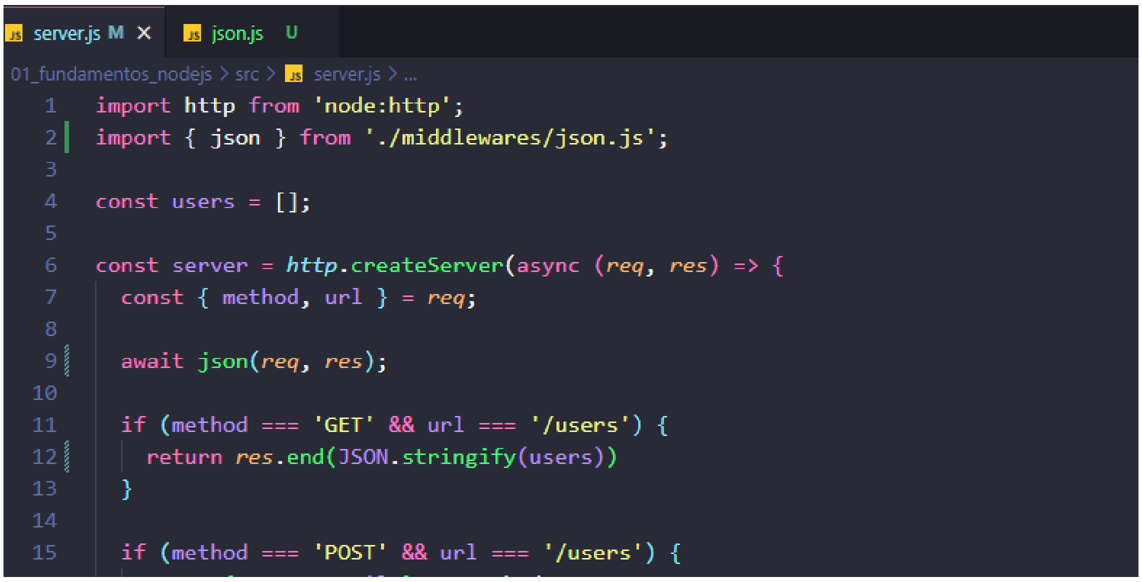
EXPLORER

ROCKETSEAT_NODE
- 01_fundamentos_...
  - src
    - middlewares
      - json.js U
    - server.js M
  - streams
  - package.json
- ~$de_info.docx
- node_info.docx

server.js M | json.js U

01_fundamentos_nodejs > src > middlewares > json.js > json

```js
export async function json(req, res) {
  const buffers = [];

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

  try {
    req.body = JSON.parse(Buffer.concat(buffers).toString());
  } catch (error) {
    req.body = null;
  }

  res.setHeader('Content-type', 'application/json')
}
```

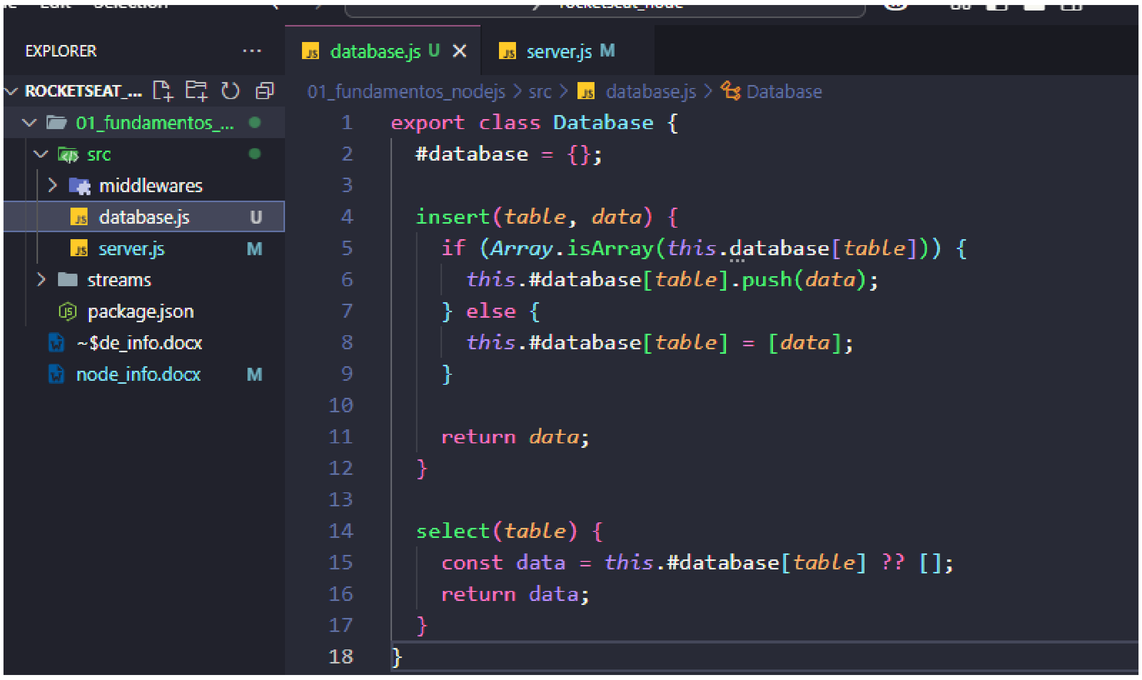


server.js M | json.js U

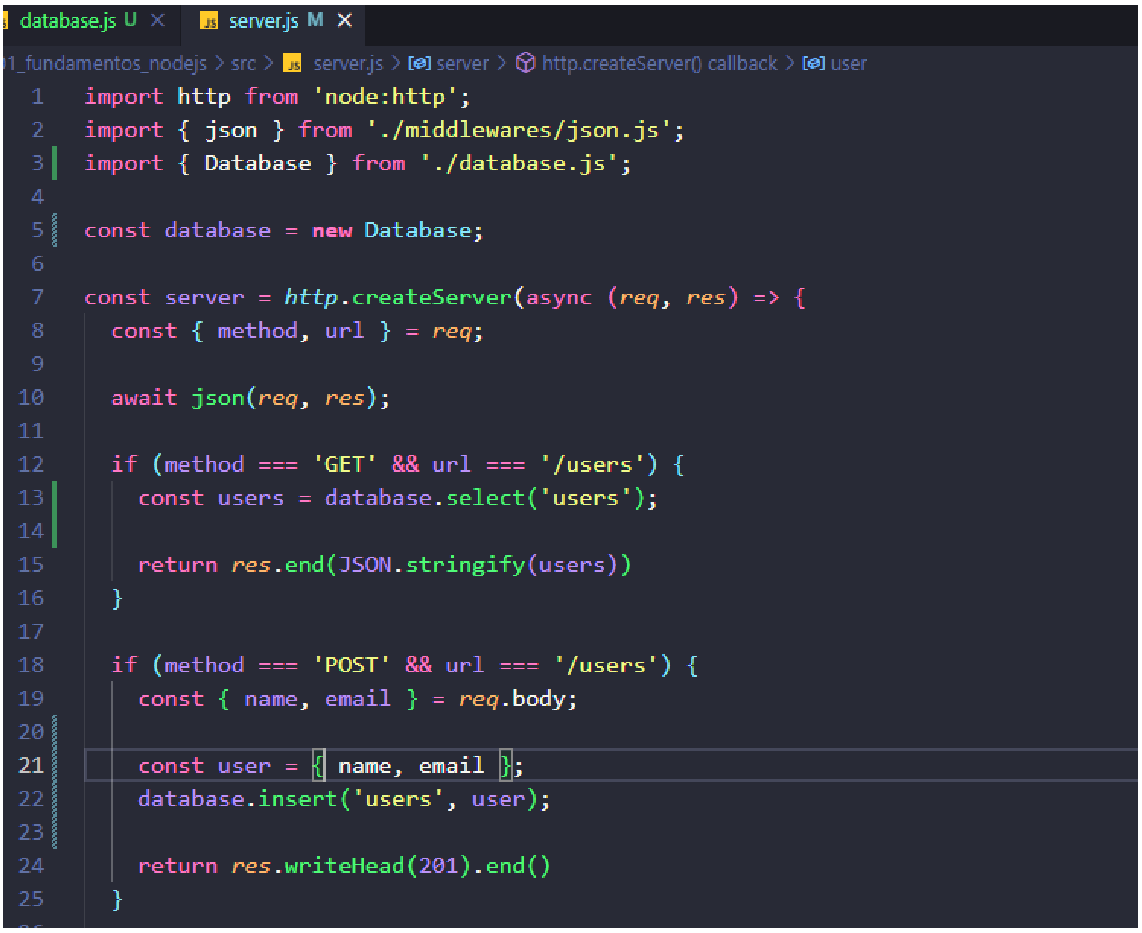
01_fundamentos_nodejs > src > server.js > ...

```js
import http from 'node:http';
import { json } from './middlewares/json.js';

const users = [];

const server = http.createServer(async (req, res) => {
  const { method, url } = req;

  await json(req, res);

  if (method === 'GET' && url === '/users') {
    return res.end(JSON.stringify(users))
  }

  if (method === 'POST' && url === '/users') {
```

13 - Criando banco de dados JSON

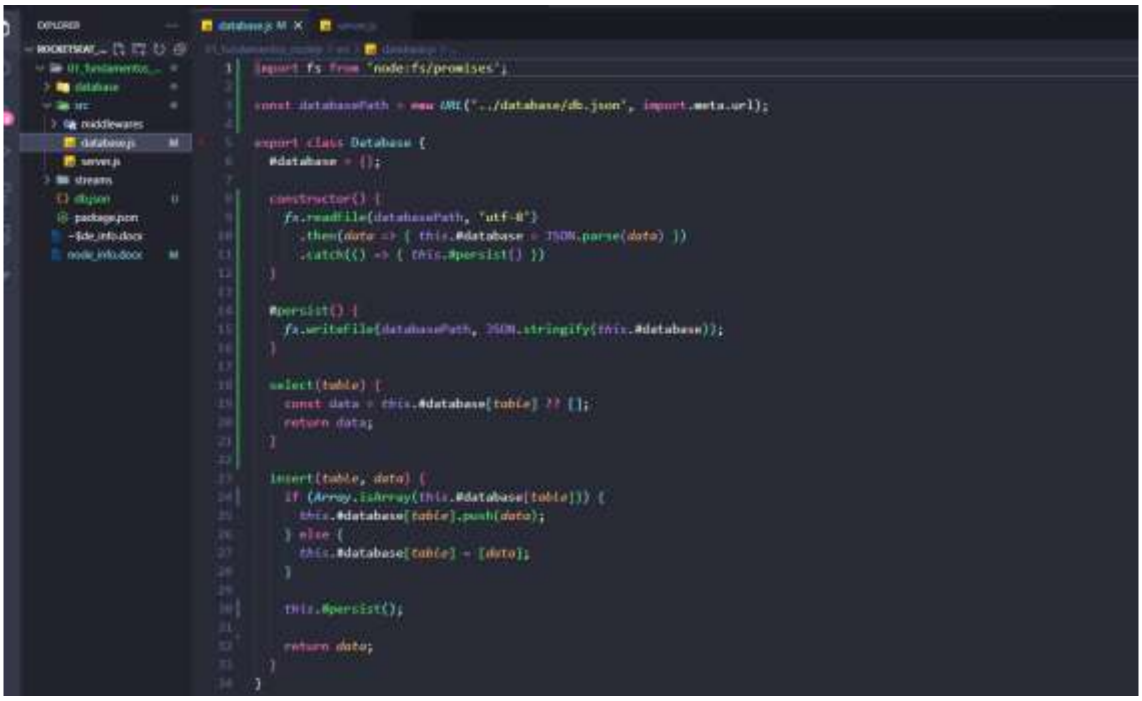


```
export class Database {
  #database = {};

  insert(table, data) {
    if (Array.isArray(this.database[table])) {
      this.#database[table].push(data);
    } else {
      this.#database[table] = [data];
    }

    return data;
  }

  select(table) {
    const data = this.#database[table] ?? [];
    return data;
  }
}
```


database.js U
server.js M
01_fundamentos_nodejs > src > server.js > server > http.createServer() callback > user

```
import http from 'node:http';
import { json } from './middlewares/json.js';
import { Database } from './database.js';

const database = new Database;

const server = http.createServer(async (req, res) => {
  const { method, url } = req;

  await json(req, res);

  if (method === 'GET' && url === '/users') {
    const users = database.select('users');

    return res.end(JSON.stringify(users))
  }

  if (method === 'POST' && url === '/users') {
    const { name, email } = req.body;

    const user = { name, email };
    database.insert('users', user);

    return res.writeHead(201).end()
  }
```

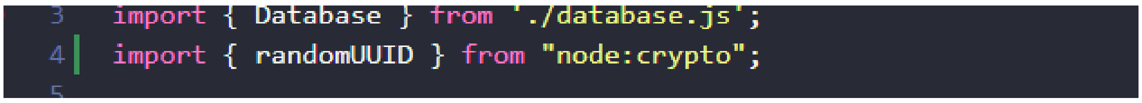
14 - Persistindo banco de dados



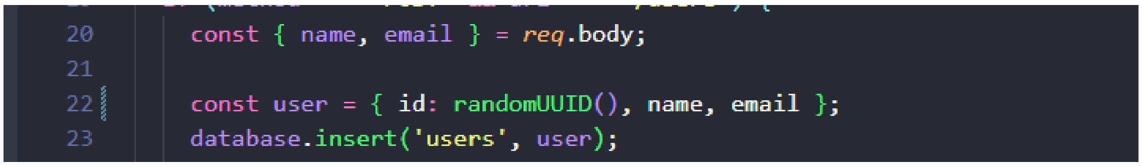
```
import fs from 'node:fs/promises';

const databasePath = new URL('../database/db.json', import.meta.url);

export class Database {
  #database = {};

  constructor() {
    fs.readFile(databasePath, 'utf-8')
      .then(data => { this.#database = JSON.parse(data) })
      .catch(() => { this.#persist() })
  }

  #persist() {
    fs.writeFile(databasePath, JSON.stringify(this.#database));
  }

  select(table) {
    const data = this.#database[table] ?? [];
    return data;
  }

  insert(table, data) {
    if (Array.isArray(this.#database[table])) {
      this.#database[table].push(data);
    } else {
      this.#database[table] = [data];
    }

    this.#persist();

    return data;
  }
}
```

15 - Criando ID único com universal UUID

Gera um id aleatório com vários caracteres

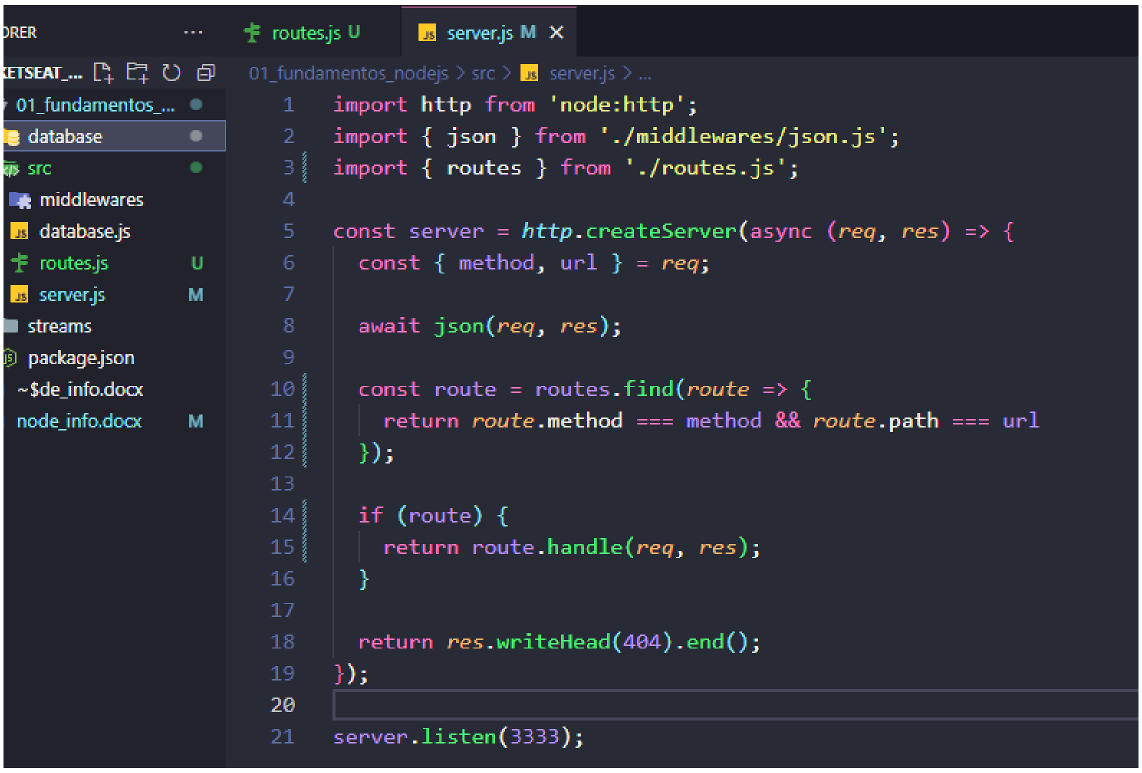


```
import { Database } from './database.js';
import { randomUUID } from "node:crypto";
```

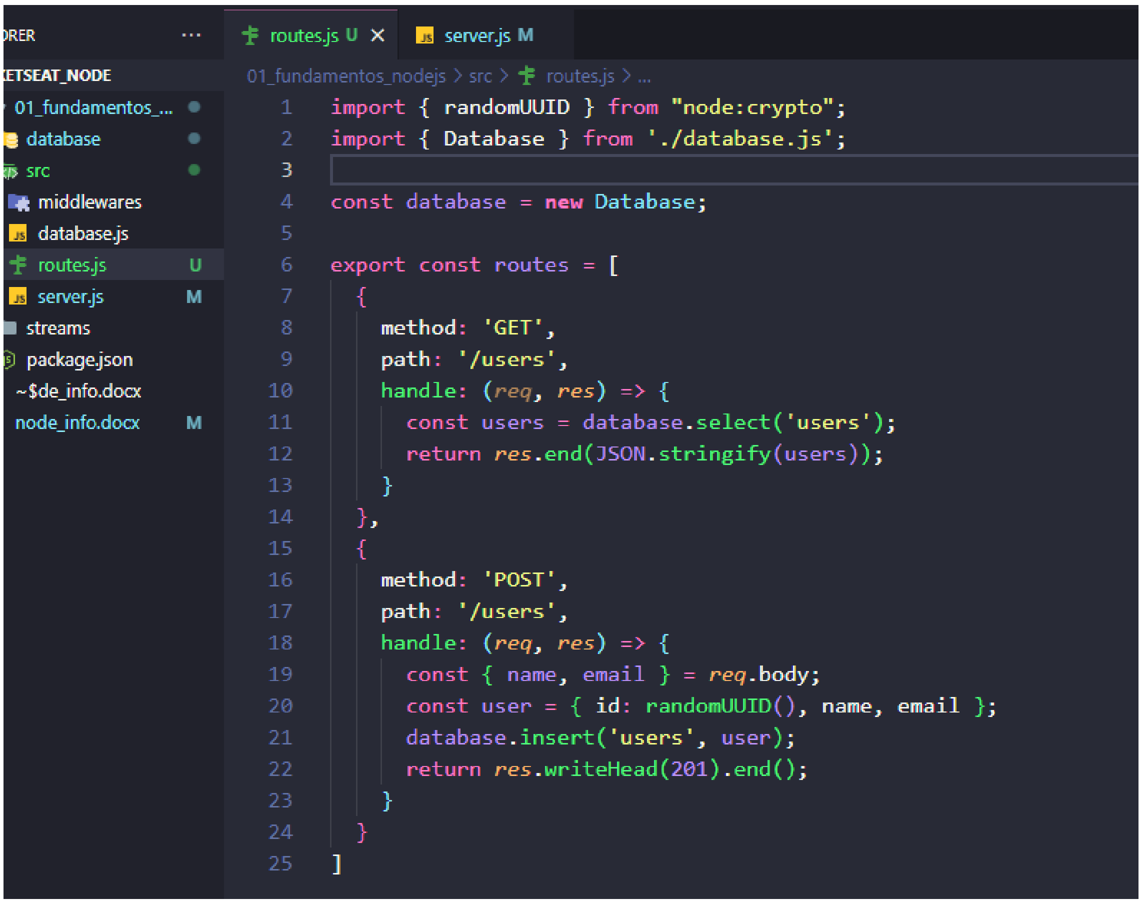


```
    const { name, email } = req.body;

    const user = { id: randomUUID(), name, email };
    database.insert('users', user);
```

16 - Separando rotas da aplicação

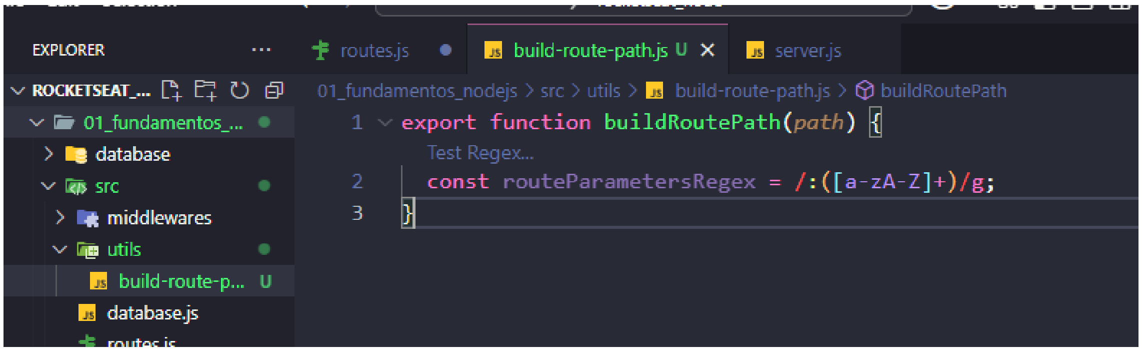


```js
import http from 'node:http';
import { json } from './middlewares/json.js';
import { routes } from './routes.js';

const server = http.createServer(async (req, res) => {
  const { method, url } = req;

  await json(req, res);

  const route = routes.find(route => {
    return route.method === method && route.path === url
  });

  if (route) {
    return route.handle(req, res);
  }

  return res.writeHead(404).end();
});

server.listen(3333);
```

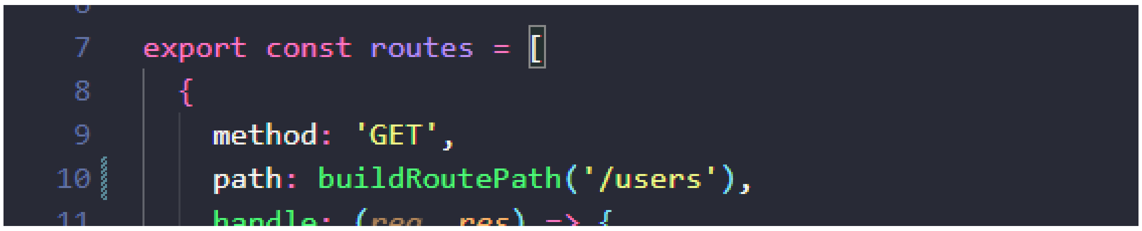


```js
import { randomUUID } from "node:crypto";
import { Database } from './database.js';

const database = new Database;

export const routes = [
  {
    method: 'GET',
    path: '/users',
    handle: (req, res) => {
      const users = database.select('users');
      return res.end(JSON.stringify(users));
    }
  },
  {
    method: 'POST',
    path: '/users',
    handle: (req, res) => {
      const { name, email } = req.body;
      const user = { id: randomUUID(), name, email };
      database.insert('users', user);
      return res.writeHead(201).end();
    }
  }
]
```

17 - Route e Query parameters

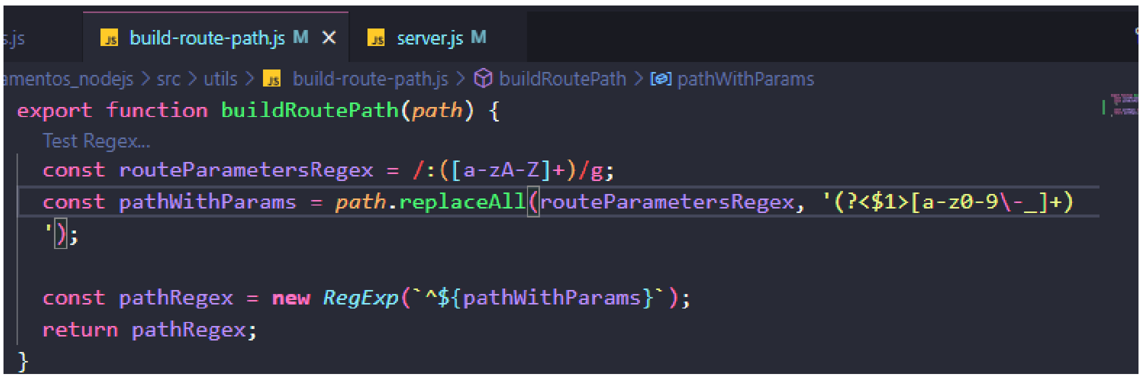


```
export function buildRoutePath(path) {
  Test Regex...
  const routeParametersRegex = /:([a-zA-Z]+)/g;
}
```

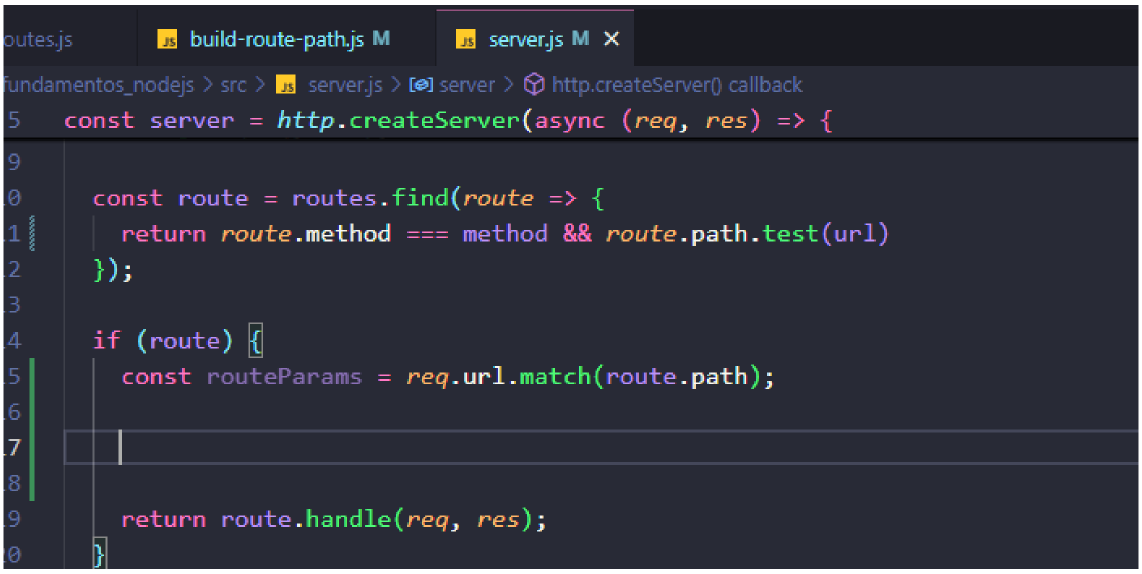


```
export const routes = [
  {
    method: 'GET',
    path: buildRoutePath('/users'),
    handle: (req, res) => {
```

18 – Rotas com parâmetros

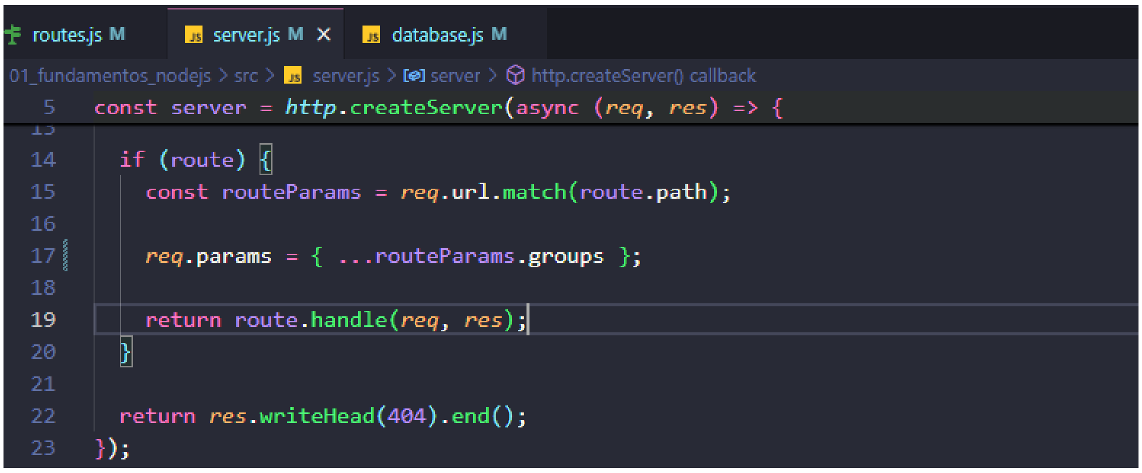

build-route-path.js M
server.js M
amentos_nodejs > src > utils > build-route-path.js > buildRoutePath > pathWithParams

```
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;
}
```


outes.js
build-route-path.js M
server.js M
fundamentos_nodejs > src > server.js > server > http.createServer() callback

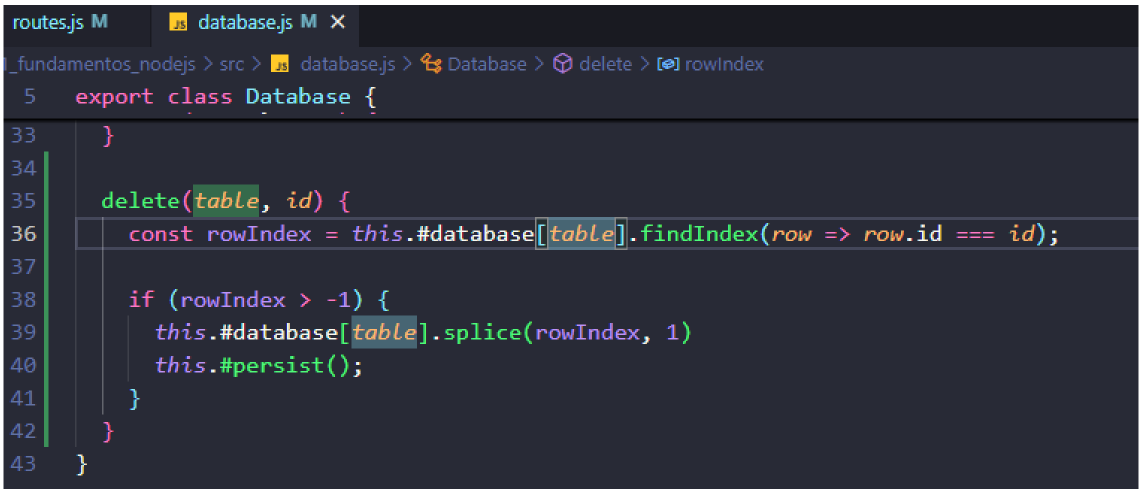
```
const server = http.createServer(async (req, res) => {

  const route = routes.find(route => {
    return route.method === method && route.path.test(url)
  });

  if (route) {
    const routeParams = req.url.match(route.path);


    return route.handle(req, res);
  }
```

19 – Remoção de registros


routes.js M | server.js M × | database.js M

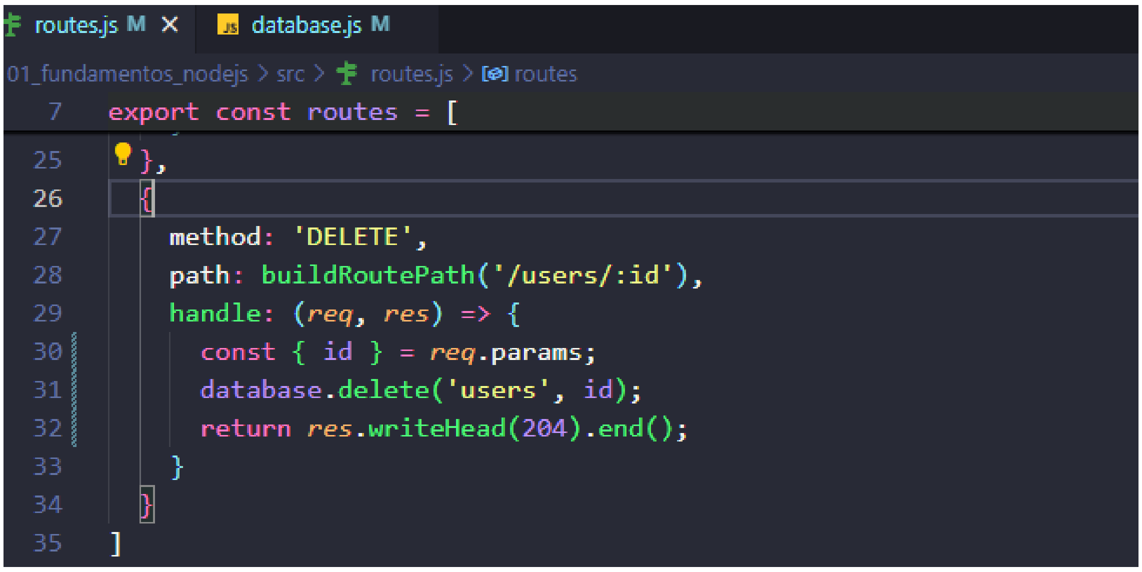
01_fundamentos_nodejs > src > server.js > server > http.createServer() callback

```
const server = http.createServer(async (req, res) => {

  if (route) {
    const routeParams = req.url.match(route.path);

    req.params = { ...routeParams.groups };

    return route.handle(req, res);
  }

  return res.writeHead(404).end();
});
```


routes.js M | database.js M ×

_fundamentos_nodejs > src > database.js > Database > delete > rowIndex

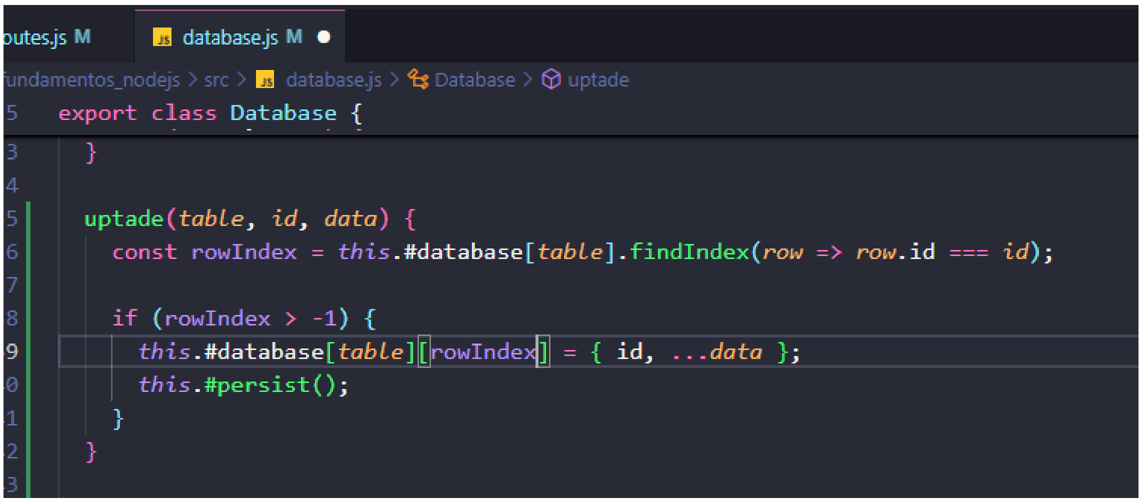
```
export class Database {
  }

  delete(table, id) {
    const rowIndex = this.#database[table].findIndex(row => row.id === id);

    if (rowIndex > -1) {
      this.#database[table].splice(rowIndex, 1)
      this.#persist();
    }
  }
}
```

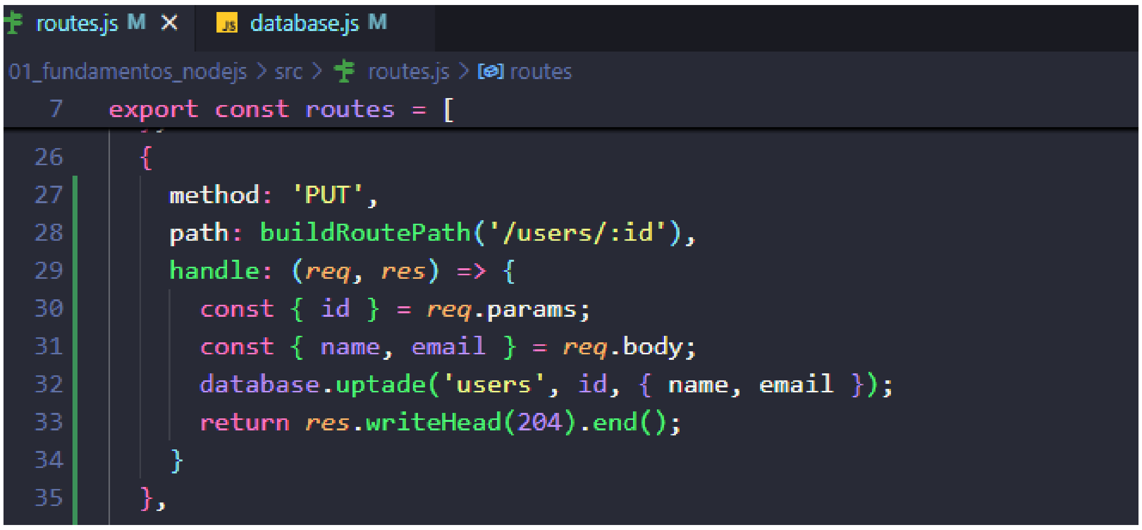

routes.js M × | database.js M

01_fundamentos_nodejs > src > routes.js > routes

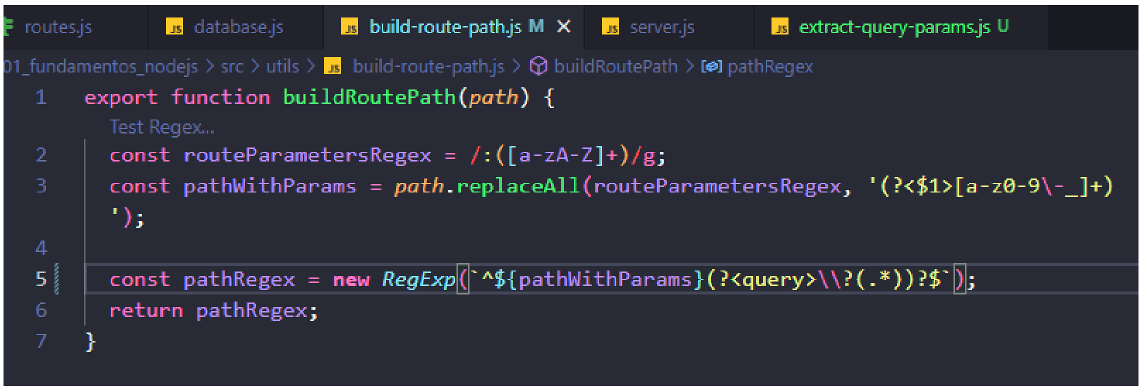
```
export const routes = [
  },
  {
    method: 'DELETE',
    path: buildRoutePath('/users/:id'),
    handle: (req, res) => {
      const { id } = req.params;
      database.delete('users', id);
      return res.writeHead(204).end();
    }
  }
]
```

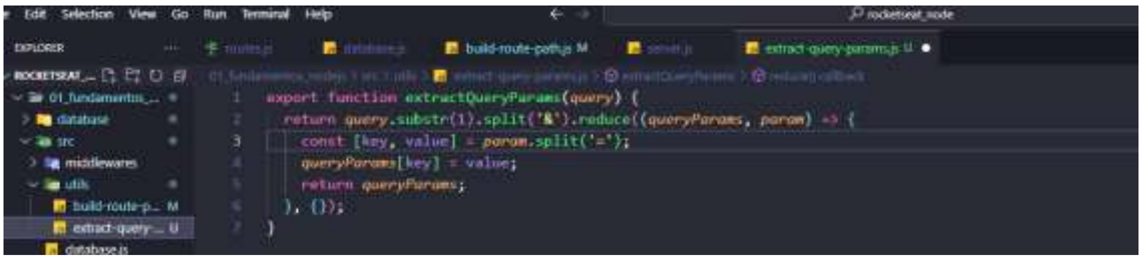
20 – Atualização de registros

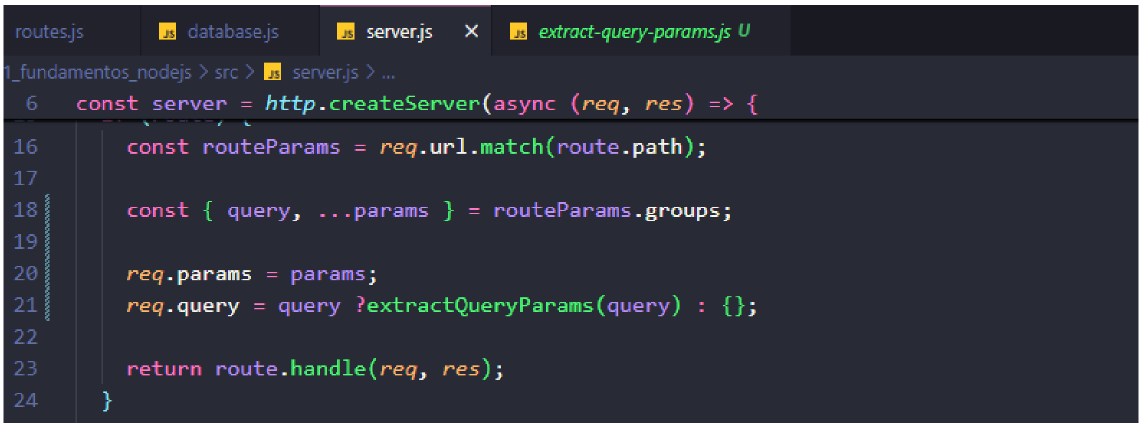

```
export class Database {
  }

  uptade(table, id, data) {
    const rowIndex = this.#database[table].findIndex(row => row.id === id);

    if (rowIndex > -1) {
      this.#database[table][rowIndex] = { id, ...data };
      this.#persist();
    }
  }
```


```
export const routes = [
  {
    method: 'PUT',
    path: buildRoutePath('/users/:id'),
    handle: (req, res) => {
      const { id } = req.params;
      const { name, email } = req.body;
      database.uptade('users', id, { name, email });
      return res.writeHead(204).end();
    }
  },
```

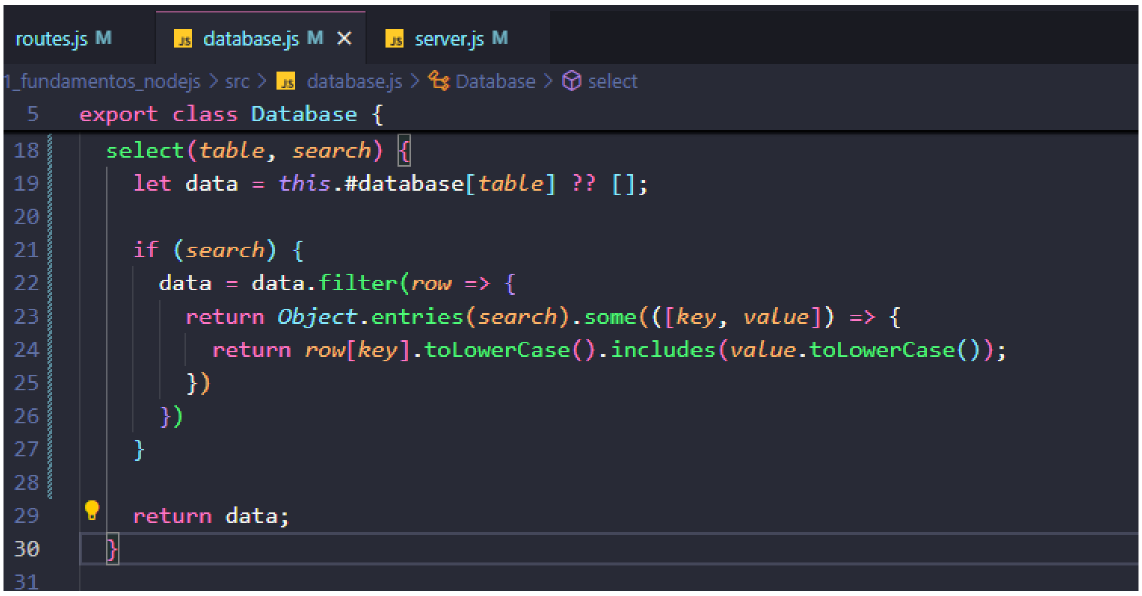
21 – Capturando query parameters


```
export function buildRoutePath(path) {
  const routeParametersRegex = /:([a-zA-Z]+)/g;
  const pathWithParams = path.replaceAll(routeParametersRegex, '(?<$1>[a-z0-9\-_]+)');

  const pathRegex = new RegExp(`^${pathWithParams}(?<query>\\?(.*))?$`);
  return pathRegex;
}
```

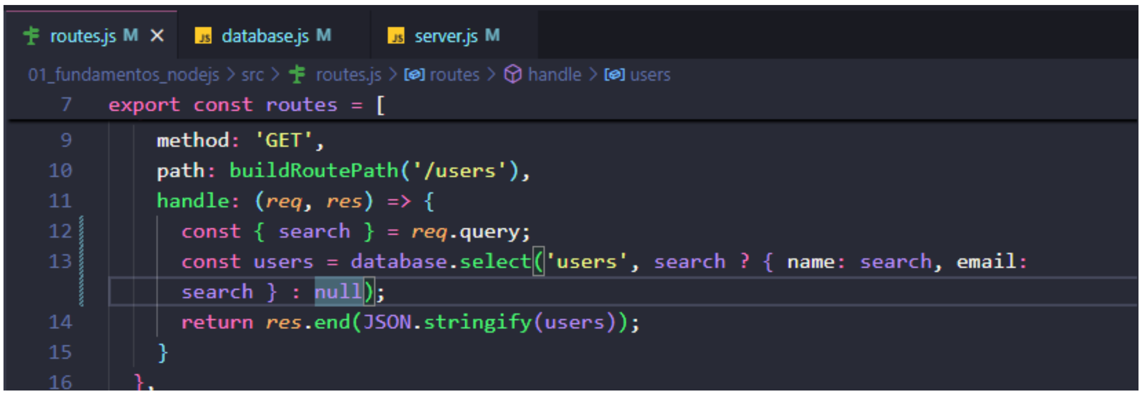

```
export function extractQueryParams(query) {
  return query.substr(1).split('&').reduce((queryParams, param) => {
    const [key, value] = param.split('=');
    queryParams[key] = value;
    return queryParams;
  }, {});
}
```


routes.js
database.js
server.js
extract-query-params.js U
1_fundamentos_nodejs > src > server.js > ...

```
const server = http.createServer(async (req, res) => {
    const routeParams = req.url.match(route.path);

    const { query, ...params } = routeParams.groups;

    req.params = params;
    req.query = query ?extractQueryParams(query) : {};

    return route.handle(req, res);
  }
```


routes.js M
database.js M
server.js M
1_fundamentos_nodejs > src > database.js > Database > select

```
export class Database {
  select(table, search) {
    let data = this.#database[table] ?? [];

    if (search) {
      data = data.filter(row => {
        return Object.entries(search).some(([key, value]) => {
          return row[key].toLowerCase().includes(value.toLowerCase());
        })
      })
    }

    return data;
  }
```


routes.js M
database.js M
server.js M
01_fundamentos_nodejs > src > routes.js > routes > handle > users

```
export const routes = [
    method: 'GET',
    path: buildRoutePath('/users'),
    handle: (req, res) => {
      const { search } = req.query;
      const users = database.select('users', search ? { name: search, email:
      search } : null);
      return res.end(JSON.stringify(users));
    }
  },
```