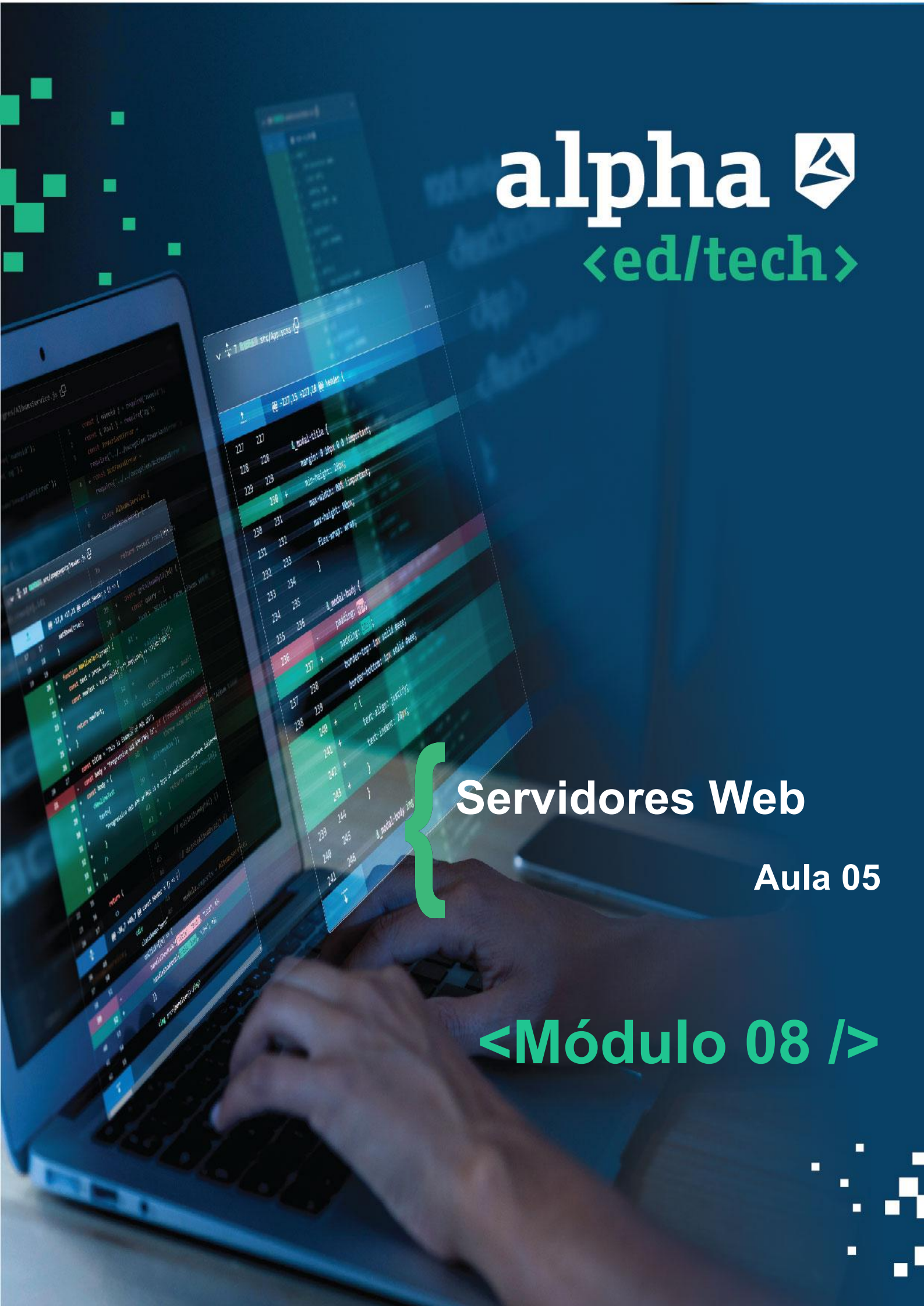

alpha
<ed/tech>
Servidores Web
Aula 05
<Módulo 08 />

Melhorando a aplicação nodejs

Introdução

Vamos agora estudar algumas melhorias que podemos realizar em nosso projeto e que nos dará um conhecimento de gestão de servidores com maior segurança.
Veremos a definição de variáveis de ambiente que são sensíveis e que não podem ser de conhecimento de demais programadores que tiverem acesso a código.
Também trataremos de gerar um hash de usuário que será utilizado para checar se o usuário, ao colocar nome e senha, a verificação seja feita do hash gerado e não da senha propriamente dita.
Aproveitaremos para armazenar os dados do usuário de login do lado do servidor de forma persistente em um banco de dados.
Animados? Vamos lá!

Variáveis de ambiente

Variáveis de ambiente são valores que podem ser configurados no sistema operacional e acessados por programas em execução, como os desenvolvidos em Node.js.
Elas são úteis para armazenar informações sensíveis, como senhas de banco de dados, chaves de API e outras configurações específicas do ambiente de execução, sem a necessidade de expor esses valores diretamente no código fonte.
Você pode definir variáveis de ambiente no arquivo '**.bashrc**', '**.bash_profile**', ou similar.
Basta adicionar as linhas '**export NOME_DA_VARIAVEL=VALOR**' para cada variável desejada.

Por exemplo, para definir uma variável de ambiente chamada '**PORT**' com o valor '**5000**', você pode usar o comando **export PORT="5000"** :

```
kenji@DESKTOP-0EELV37:~/exemplo-jwt$ pwd
/home/kenji/exemplo-jwt
kenji@DESKTOP-0EELV37:~/exemplo-jwt$ export PORT="5000"
kenji@DESKTOP-0EELV37:~/exemplo-jwt$ npm start

> exemplo-jwt@1.0.0 start
> node ./src/app.js

Servidor rodando em http://localhost:5000
```

Outra forma de passar a variável de ambiente é no próprio comando que inicia a aplicação:

```
kenji@DESKTOP-0EELV37:~/exemplo-jwt$ pwd
/home/kenji/exemplo-jwt
kenji@DESKTOP-0EELV37:~/exemplo-jwt$ PORT=6000 npm start

> exemplo-jwt@1.0.0 start
> node ./src/app.js

Servidor rodando em http://localhost:6000
```

Muitas variáveis de ambiente

Em sistemas normais, não é incomum termos diversas variáveis de ambiente que determinam como a nossa aplicação vai se comportar, por exemplo, qual o endereço do banco de dados e a porta, bem como o usuário e a senha para acesso a ele. Estas informações não devem ficar registradas em um arquivo no github mas devem ser passadas para a aplicação.
Também imagine o valor de SECRET_KEY não deve ficar em um arquivo de config facilmente legível e disponibilizado no repositório git por exemplo. Seria uma falha de segurança enorme!

Melhorando a aplicação nodejs

Variáveis de ambiente com 'dotenv'

Instale a partir da pasta do seu projeto o pacote **'dotenv':**

```
kenji@DESKTOP-0EELV37:~/exemplo-jwt$ pwd
/home/kenji/exemplo-jwt
kenji@DESKTOP-0EELV37:~/exemplo-jwt$ npm install dotenv --save

added 1 package, and audited 84 packages in 1s

13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
kenji@DESKTOP-0EELV37:~/exemplo-jwt$ cat package.json
{
  "name": "exemplo-jwt",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "start": "node ./src/app.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "cookie-parser": "^1.4.6",
    "dotenv": "^16.4.5",
    "express": "^4.18.3",
    "jsonwebtoken": "^9.0.2"
  }
}
```

Agora basta criar um arquivo chamado **'.env'** na 'raiz' do seu projeto (mesma pasta do 'package.json'):

Este arquivo está na pasta do projeto e normalmente está configurado dentro do **'.gitignore'**, igualmente com o **'node_modules'**, isto faz com que ninguém tenha acesso a suas variáveis de ambiente.

```
kenji@DESKTOP-0EELV37:~/exemplo-jwt$ pwd
/home/kenji/exemplo-jwt
kenji@DESKTOP-0EELV37:~/exemplo-jwt$ cat .env
NODE_ENV=development
SECRET_KEY="esta_eh_uma_chave_que_deve_ser_mantida_em_segredo"
```

Melhorando a aplicação nodejs

Variáveis de ambiente com ‘dotenv’

Agora basta alterar o arquivo **‘/src/config/index.js’** para:

Veja que o arquivo de configuração agora não ‘expõe’ as variáveis de ambiente e configuram os valores de config caso elas existam ou mantém o valor padrão para uso.

Para utilizar as variáveis de ambiente, basta importar no arquivo no qual será utilizado, por exemplo no **‘/src/app.js’**

```
const config = require('./config');
const express = require('express');
const app = express();
const port = config.PORT;
const cookieParser = require('cookie-parser');

// Middleware para analisar o corpo das requisições JSON
app.use(express.json());

// Middleware para lidar com cookies
app.use(cookieParser());

// Login Page
const loginPage = require('./pages/login');
app.get('/', loginPage);

// App Page
const appPage = require('./pages/app');
app.get('/app', appPage);

// Rotas
const routes = require('./routes');
app.use('/api', routes);

app.listen(port, () => {
    console.log(`Servidor rodando em http://localhost:${port}`);
});
```

Acima a constante ‘**port**’ é obtida de ‘**config.PORT**’ que neste caso será ‘3000’ porque o arquivo **‘.env’** não tem configurada a variável de ambiente **‘PORT’** configurado e permite que seja passado diretamente na chamada da aplicação.

Melhorando a aplicação nodejs

Variáveis de ambiente com ‘dotenv’

Note que não foi necessária nenhuma alteração no arquivo **‘/src/controllers/loginControler.js’** porque a importação da constante ‘**SECRET_KEY**’ já foi feita anteriormente:

src > controllers > loginController.js > ...

```
const { SECRET_KEY } = require("../config");
const jwt = require('jsonwebtoken');

let loginUsers = [
    {
        "username": "kenji",
        "name": "Kenji Taniguchi",
        "password": "123456",
        "userType": [ "admin", "user" ],
    },
    {
        "username": "samir",
        "name" : "Samir",
        "password" : "654321",
        "userType": [ "user" ],
    }
];

const getLogin = async (req, res) => {
    const user = req.user;
    return res.json(user);
};

const autenticate = async (req, res) => {
    const { username, password } = req.body;

    const error = "Usuário e/ou senha inválidos!";
    if (!username || !password) {
        res.cookie('session_id', '', { expires: new Date(0) });
        return res.status(400).json({ error });
    }

    // Procura a existência de um usuário e senha em loginUsers
    const foundUser = loginUsers.filter(user => user.username === username && user.password === password);
    if (foundUser.length === 0) {
        res.cookie('session_id', '', { expires: new Date(0) });
        return res.status(400).json({ error });
    }

    // Usuário encontrado!
    const user = {
        username: foundUser[0].username,
        name: foundUser[0].name,
        user_type: foundUser[0].userType,
    };

    // Gerando token JWT com informações personalizadas e enviando como cookie session_id
    try {
        const sessionToken = await jwt.sign({ user }, SECRET_KEY);
        res.cookie('session_id', sessionToken, { maxAge: 900000, httpOnly: true });
        res.json({ success: true });
    } catch (err) {
        res.status(500).json({ error: 'Erro ao gerar token JWT' });
    }

};

module.exports = {
    getLogin,
    autenticate,
};
```

Melhorando a aplicação nodejs

Biblioteca ‘bcrypt’

O **bcrypt** é uma biblioteca popular para ***‘hashing’*** de senhas em Node.js e em outras linguagens de programação.
Ele é amplamente utilizado devido à sua segurança e eficácia na proteção de senhas.

- **Hashing seguro de senhas**: O bcrypt é uma função de hash de senhas projetada para ser segura contra ataques de força bruta e ataques de dicionário. Ele utiliza um algoritmo de hash adaptativo que inclui um fator de custo ajustável, tornando mais difícil para os atacantes adivinharem senhas através de tentativa e erro.

- **Salting automático**: O bcrypt inclui a funcionalidade de adicionar um salt automaticamente ao hash da senha. O salt é um valor aleatório único que é adicionado à senha antes de hashear, o que torna cada hash único mesmo que duas senhas sejam idênticas. Isso aumenta a segurança, pois impede que os atacantes usem tabelas de hash precomputadas (rainbow tables) para quebrar senhas facilmente.

- **Custo ajustável**: O bcrypt permite ajustar o custo computacional do processo de hash, tornando-o mais lento e mais seguro conforme necessário. Isso é feito especificando o número de rounds de geração de salt ao criar o hash. Um número maior de rounds torna o processo de hash mais lento, mas também mais seguro contra ataques de força bruta.

- **Facilidade de uso em Node.js**: A biblioteca bcrypt é fácil de usar em aplicativos Node.js. Ela fornece métodos simples para gerar hashes de senhas, verificar senhas hashadas e trabalhar com salting de forma transparente.

- **Suporte a Promises e async/await**: O bcrypt oferece suporte para operações assíncronas, o que é útil em aplicativos Node.js onde a execução de operações de hash pode ser demorada. Isso permite que você use async/await ou promessas para lidar de forma eficiente com operações de hash em seu código.

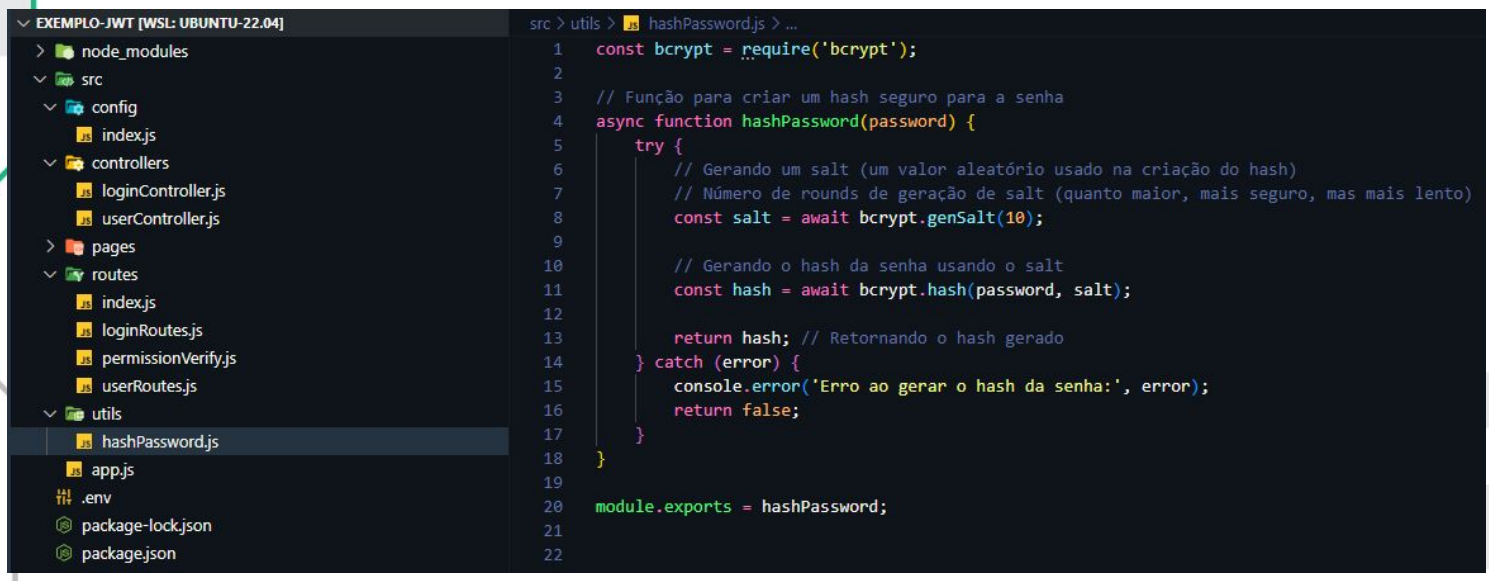
No geral, o **bcrypt** é uma escolha popular para proteger senhas em aplicativos Node.js devido à sua segurança comprovada, flexibilidade e facilidade de uso. Ele ajuda a garantir que as senhas dos usuários sejam armazenadas de forma segura e protegidas contra ataques maliciosos.

```
kenji@DESKTOP-0EELV37:~/exemplo-jwt$ pwd
/home/kenji/exemplo-jwt
kenji@DESKTOP-0EELV37:~/exemplo-jwt$ npm install bcrypt --save

added 57 packages, and audited 141 packages in 8s

16 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
kenji@DESKTOP-0EELV37:~/exemplo-jwt$ cat package.json
{
  "name": "exemplo-jwt",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "start": "node ./src/app.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.1.1",
    "cookie-parser": "^1.4.6",
    "dotenv": "^16.4.5",
    "express": "^4.18.3",
    "jsonwebtoken": "^9.0.2"
  }
}
```

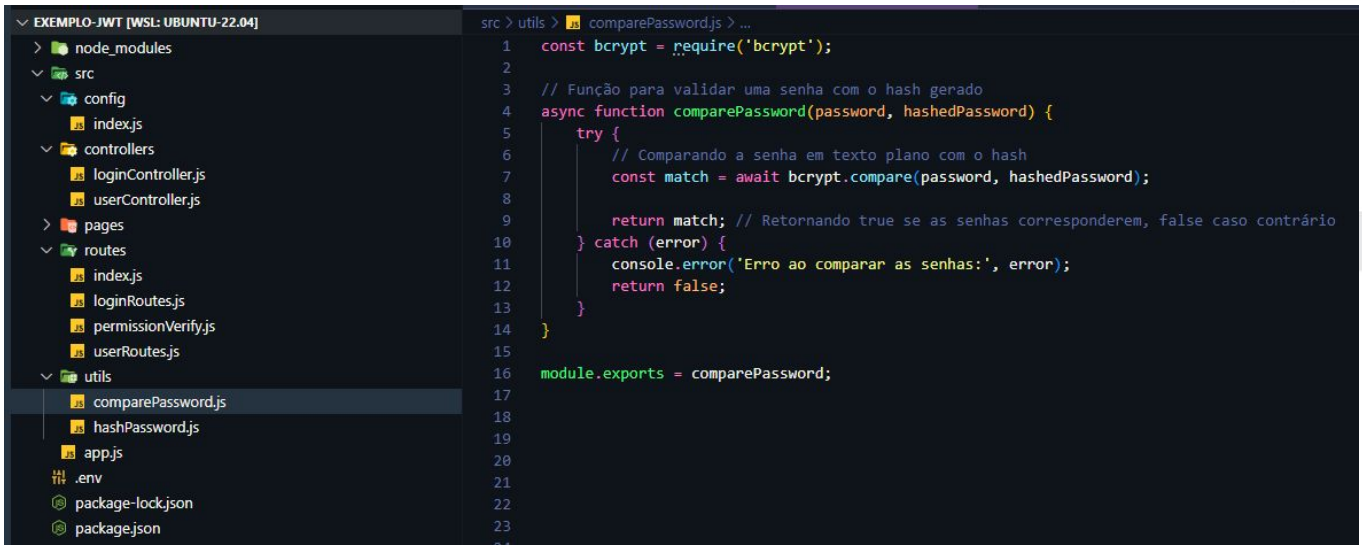
Melhorando a aplicação nodejs

Criando uma função para gerar ‘hash’ de senhas

Foi criada uma pasta ‘utils’ que conterá todas as funções e classes que serão utilizadas pelo nosso aplicativo.
Neste caso, foi criado o arquivo **‘hashPassword.js’**



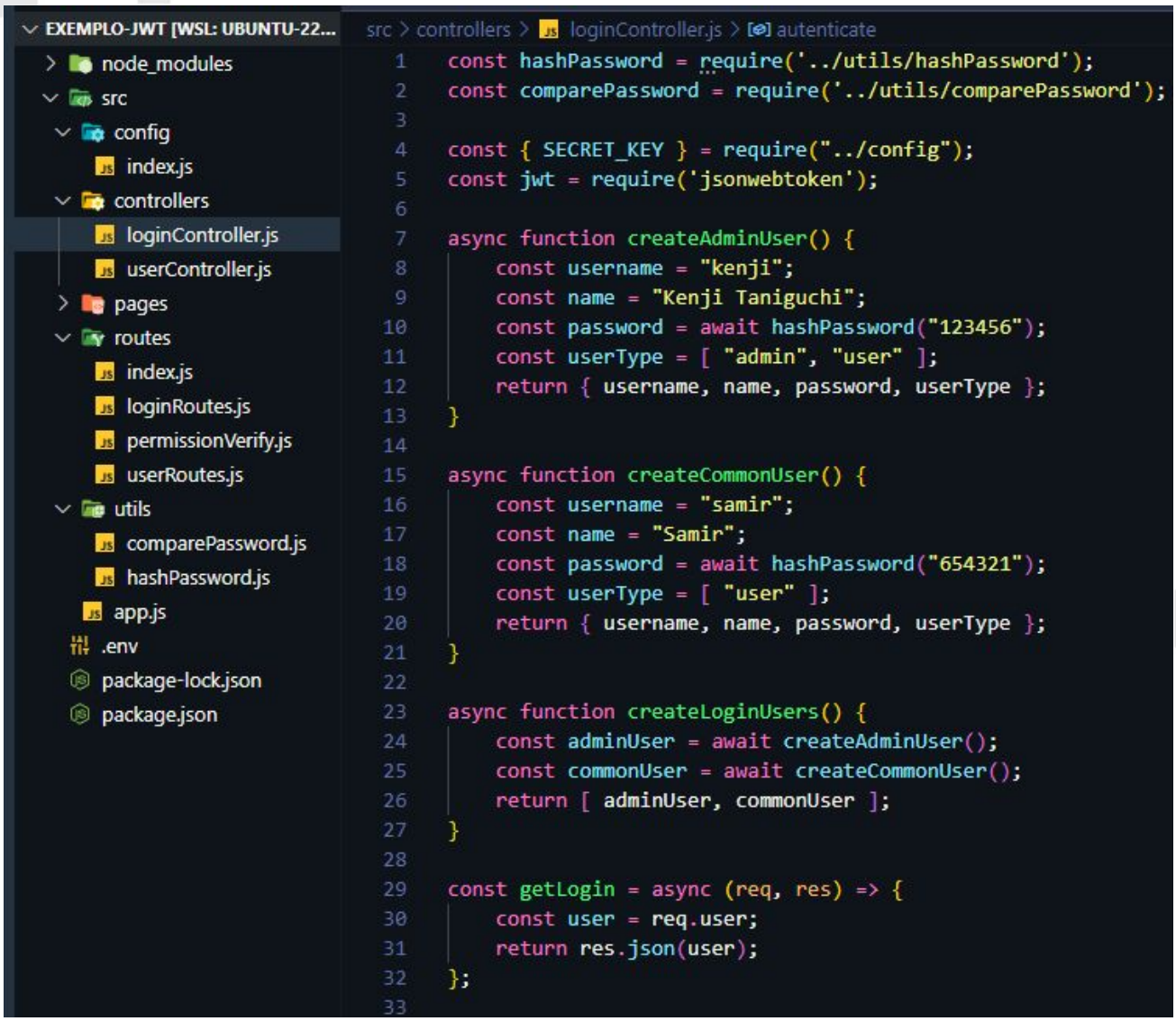
Agora crie um arquivo **‘comparePassword.js’**



Melhorando a aplicação nodejs

Ajustando '/src/controllers/loginController.js'

Vamos ajustar para criarmos os dois usuários agora com a senha criptografada:



Foram criadas funções assíncronas:

- **createAdminUser():** que retorna um objeto parecido com o que estava antes porém gerando um hash de um password que antes era do tipo 'plain text'
- **createCommonUser():** que retorna um objeto parecido com o que estava antes porém gerando um hash de um password que antes era do tipo 'plain text'
- **createLoginUsers():** esta função retorna o array composto pelos dois usuários gerados.

Melhorando a aplicação nodejs

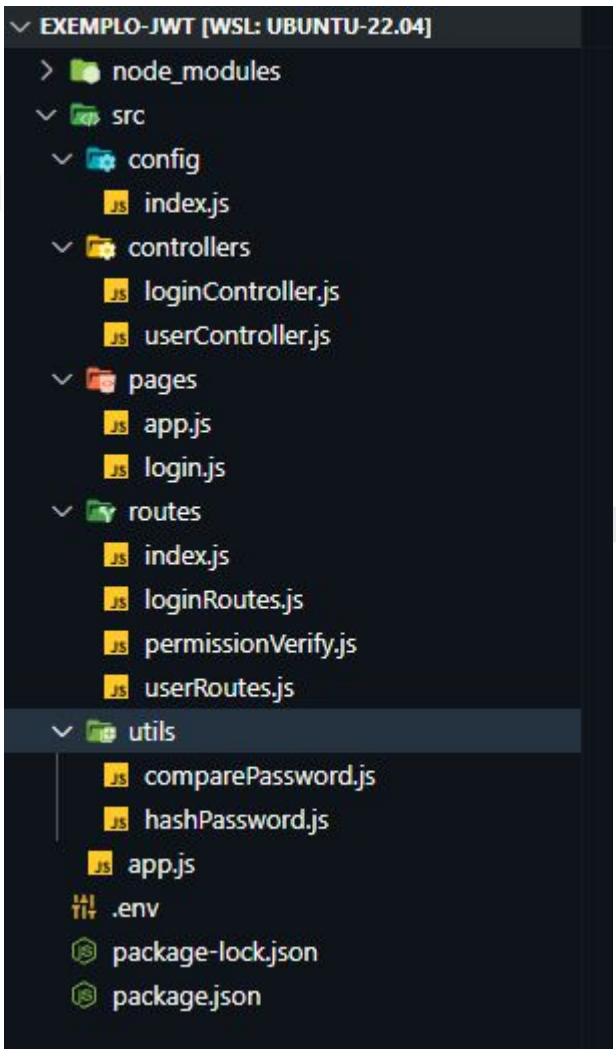
Ajustando '/src/controllers/loginController.js' (continuação)

```
33
34  const autenticate = async (req, res) => {
35      // Para fins de inicialização de 2 usuários com as senhas criptografadas
36      const loginUsers = await createLoginUsers();
37
38      const { username, password } = req.body;
39
40      const error = "Usuário e/ou senha inválidos!";
41      if (!username || !password) {
42          res.cookie('session_id', '', { expires: new Date(0) });
43          return res.status(400).json({ error });
44      }
45
46      // Procura a existência de um usuário e senha em loginUsers
47      const foundUser = loginUsers.filter(user => user.username === username);
48      if (foundUser.length === 0) {
49          res.cookie('session_id', '', { expires: new Date(0) });
50          return res.status(400).json({ error });
51      }
52
53      // Compara a senha fornecida com a senha criptografada
54      const match = await comparePassword(password, foundUser[0].password);
55
56      if (!match) {
57          res.cookie('session_id', '', { expires: new Date(0) });
58          return res.status(400).json({ error });
59      }
60
61      // Usuário encontrado!
62      const user = {
63          username: foundUser[0].username,
64          name: foundUser[0].name,
65          user_type: foundUser[0].userType,
66      };
67
68      // Gerando token JWT com informações personalizadas e enviando como cookie session_id
69      try {
70          const sessionToken = await jwt.sign({ user }, SECRET_KEY);
71          res.cookie('session_id', sessionToken, { maxAge: 900000, httpOnly: true });
72          res.json({ success: true });
73      } catch (err) {
74          res.status(500).json({ error: 'Erro ao gerar token JWT' });
75      }
76
77  };
78
79  module.exports = {
80      getLogin,
81      autenticate,
82  };
```

Veja que quando o usuário acessa o POST "/api/login" acessa o 'loginController.autenticate' que inicialmente cria os dois usuários para teste e depois procura se o usuário existe (linha 47).
Depois compara a senha passada com a senha armazenada no formato criptografado (linha 54) usando nossa função 'comparePassword' e, caso tenha sucesso, envia o Token JWT como na aula anterior.

Melhorando a aplicação nodejs

Estrutura de pastas e arquivos



Algumas considerações

Este exemplo de criação de usuários 'pré-configurados' bem como a demonstração de uso das funções assíncronas 'hashPassword' e 'comparePassword' visam o entendimento principalmente de que senhas não devem ser armazenadas em formato legível caso alguém tenha acesso não autorizado ao nosso banco de dados (neste caso representado por apenas um array de 2 usuários).
Também o uso de 'bcrypt' para criação de hashes 'fortes' e de maior dificuldade de quebras indevidas sendo a solução utilizando 'bcrypt' a mais indicada quando se tem a preocupação de segurança apesar de maior custo computacional.

Banco de Dados RocksDB

Introdução

O **RocksDB** é uma biblioteca de armazenamento de chave-valor altamente eficiente e robusta que é usada em produção por várias empresas em grande escala, incluindo o Facebook, onde foi originalmente desenvolvido.
Desde então, muitas outras empresas adotaram o RocksDB devido à sua performance, confiabilidade e escalabilidade.

Aqui estão alguns pontos que destacam por que o RocksDB é considerado adequado para uso em produção:

- **Performance**: O RocksDB é projetado para oferecer alto desempenho, com tempos de resposta rápidos e eficiência em termos de utilização de recursos. Ele é otimizado para operações de leitura e gravação rápidas, mesmo em grandes conjuntos de dados.

- **Escalabilidade**: O RocksDB é escalável e pode lidar com grandes volumes de dados sem sacrificar o desempenho. Ele é usado em ambientes de produção com grandes cargas de trabalho em várias empresas de tecnologia.

- **Confiabilidade**: O RocksDB é amplamente testado e usado em ambientes de produção de missão crítica. Ele é robusto e estável, com recursos como recuperação automática em caso de falha.

- **Manutenção ativa**: O projeto RocksDB é mantido ativamente pela comunidade e suportado pelo Facebook, garantindo que esteja atualizado com as melhores práticas e padrões de desenvolvimento.

- **Adoção pela comunidade**: Além do Facebook, muitas outras empresas e projetos de código aberto confiam no RocksDB como seu mecanismo de armazenamento de chave-valor de escolha, o que é um testemunho de sua adequação para uso em produção.

```
kenji@DESKTOP-0EELV37:~/exemplo-jwt$ pwd
/home/kenji/exemplo-jwt
kenji@DESKTOP-0EELV37:~/exemplo-jwt$ npm install rocksdb --save

added 12 packages, and audited 153 packages in 7s

21 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
kenji@DESKTOP-0EELV37:~/exemplo-jwt$ cat package.json
{
  "name": "exemplo-jwt",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "start": "node ./src/app.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.1.1",
    "cookie-parser": "^1.4.6",
    "dotenv": "^16.4.5",
    "express": "^4.18.3",
    "jsonwebtoken": "^9.0.2",
    "rocksdb": "^5.2.1"
  }
}
```

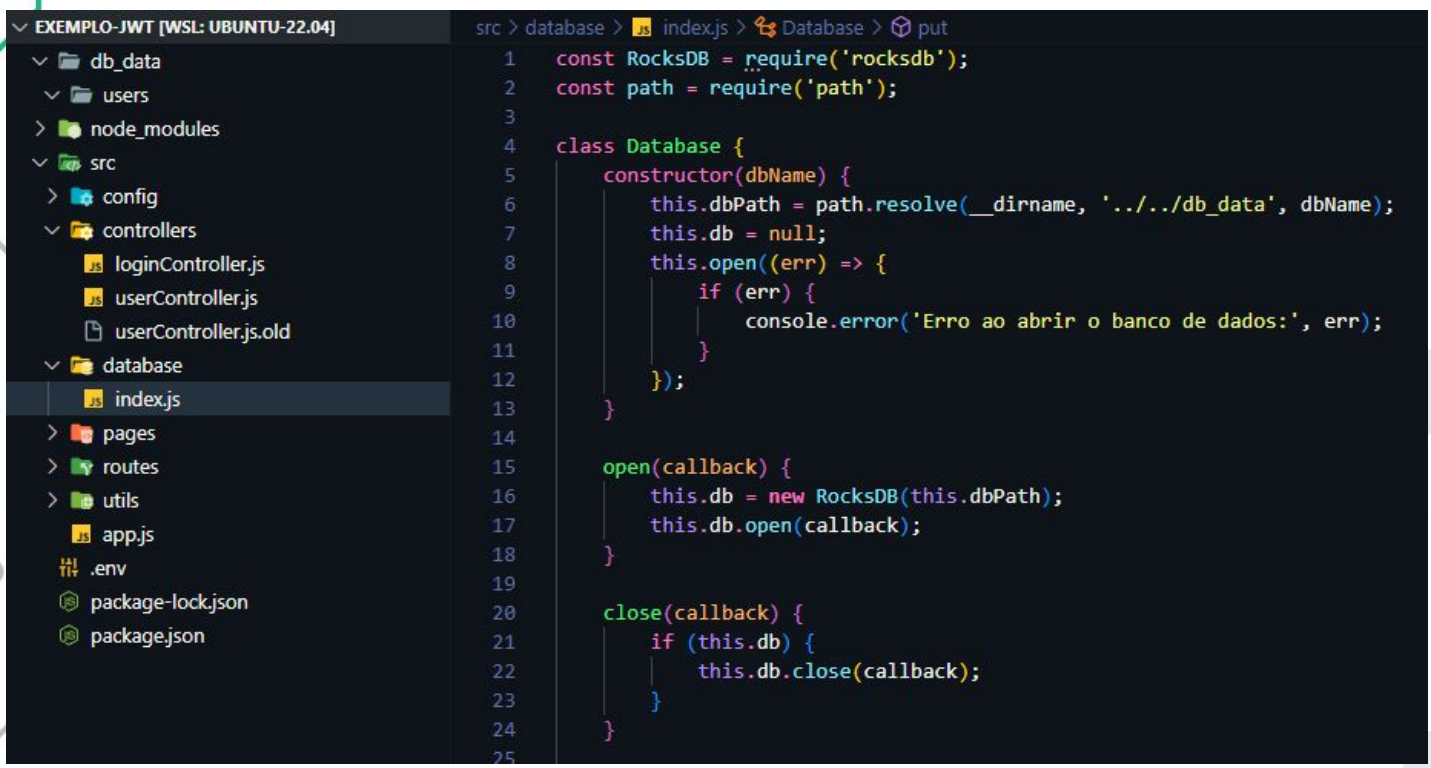
Banco de Dados RocksDB

'/src/database/index.js'

Este arquivo contém a classe de utilização do banco de dados.

Note que foi criada uma pasta na raiz do projeto '/db_data/users' que será a pasta que receberá os dados dos usuários.

A classe Database permite que você abra quantos databases quiser, isto é, basta passar o nome do database como parâmetro do construtor (dbName) que a pasta será criada.



Os métodos de criados nesta aplicação consistem em:

- **open(callback):** inicia a conexão com o banco de dados;
- **close(callback):** encerra a conexão ao banco de dados;
- **readAllData(callback):** retorna todos os elementos do banco de dados;
- **put(key, value, callback):** grava um registro baseado na key e value fornecidas;
- **get(key, callback):** retorna o valor da key fornecida;
- **del(key, callback):** apaga a key do banco de dados.

Outros métodos

Há diversas outras possibilidades de utilização do RocksDB que não se limitam a estas consultas básicas.
Você pode inclusive pesquisar por parte de chaves e criar estruturas mais complexas.

Visite o site oficial para maiores informações: https://rocksdb.org/

Banco de Dados RocksDB

'/src/database/index.js' (continuação)

```
26      readAllData(callback) {
27          if (!this.db) {
28              return callback(new Error('O banco de dados não está aberto'));
29          }
30
31          const data = [];
32
33          const iterator = this.db.iterator({});
34
35          const loop = () => {
36              iterator.next((err, key, value) => {
37                  if (err) {
38                      iterator.end(() => {
39                          callback(err);
40                      });
41                      return;
42                  }
43
44                  if (!key && !value) { // Verifica se chegou ao final do iterator
45                      iterator.end(() => {
46                          callback(null, data);
47                      });
48                      return;
49                  }
50
51                  data.push({ key: key.toString(), value: value.toString() });
52                  loop(); // Chama recursivamente loop para continuar iterando
53              });
54          };
55
56          loop(); // Inicia o loop inicialmente
57      }
58
59      put(key, value, callback) {
60          if (!this.db) {
61              return callback(new Error('O banco de dados não está aberto'));
62          }
63          this.db.put(key, value, callback);
64      }
65
66      get(key, callback) {
67          if (!this.db) {
68              return callback(new Error('O banco de dados não está aberto'));
69          }
70          this.db.get(key, callback);
71      }
72
73      del(key, callback) {
74          if (!this.db) {
75              return callback(new Error('O banco de dados não está aberto'));
76          }
77          this.db.del(key, callback);
78      }
79  }
80
81  module.exports = Database;
```

Banco de Dados RocksDB

‘/src/controllers/userController.js’

Agora precisamos alterar nosso controller para usuários fazendo a implementação da persistência de dados no RocksDB:

src > controllers > userController.js > updateUser > db.get() callback

```
const Database = require('../database');
const db = new Database('users');

const error = "Sem permissão para realizar esta requisição!";

const getAllUsers = (req, res) => {
    const admin = req.user.user_type.includes("admin");
    if (!admin) {
        res.status(403).json({error})
    }

    db.readAllData((err, data) => {
        if (err) {
            res.status(500).json({ error: 'Erro ao buscar usuários' });
            return;
        }
        res.json(data);
    });

};

const createUser = (req, res) => {
    const admin = req.user.user_type.includes("admin");
    if (!admin) {
        res.status(403).json({error})
    }

    const { username, email } = req.body;
    // Validação simples
    if (!username || !email) {
        return res.status(400).json({ error: 'O username e o email são obrigatórios' });
    }

    const newUser = { username, email };

    db.put(`user_${username}`, JSON.stringify(newUser), (err) => {
        if (err) {
            res.status(500).json({ error: 'Erro ao criar usuário' });
            return;
        }
        res.status(201).json(newUser);
    });
};

```

Banco de Dados RocksDB

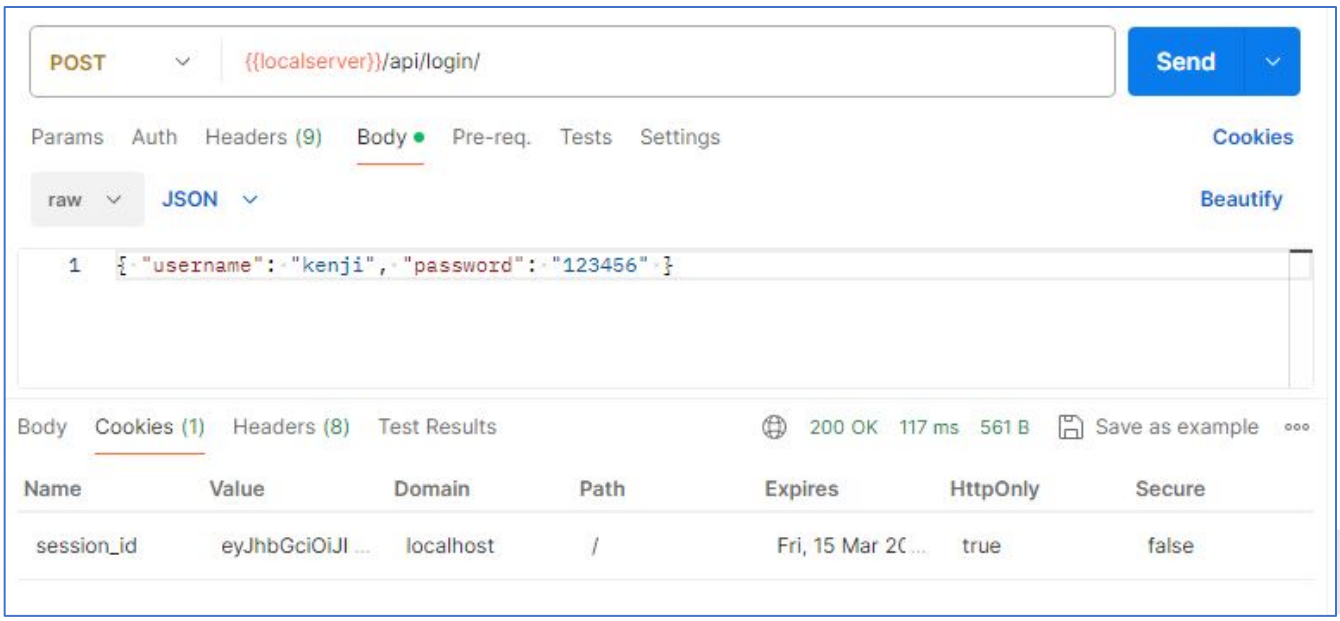
'/src/controllers/userController.js' (continuação)

```
45  const updateUser = (req, res) => {
46      const admin = req.user.user_type.includes("admin");
47      if (!admin) {
48          res.status(403).json({error})
49      }
50
51      const userId = req.params.id;
52      const { username, email } = req.body;
53
54      db.get(`user_${userId}`, (err, value) => {
55          if (err) {
56              res.status(404).json({ error: 'Usuário não encontrado' });
57              return;
58          }
59          const updatedUser = { username, email };
60          db.put(`user_${userId}`, JSON.stringify(updatedUser), (err) => {
61              if (err) {
62                  res.status(500).json({ error: 'Erro ao atualizar usuário' });
63                  return;
64              }
65              res.json({ message: `Usuário ${userId} atualizado` });
66          });
67      });
68  };
69
70  const deleteUser = (req, res) => {
71      const admin = req.user.user_type.includes("admin");
72      if (!admin) {
73          res.status(403).json({error})
74      }
75
76      const userId = req.params.id;
77
78      db.del(`user_${userId}`, (err) => {
79          if (err) {
80              res.status(500).json({ error: 'Erro ao excluir usuário' });
81              return;
82          }
83          res.json({ message: `Usuário ${userId} excluído` });
84      });
85  };
86
87  module.exports = {
88      getAllUsers,
89      createUser,
90      updateUser,
91      deleteUser,
92  };
```

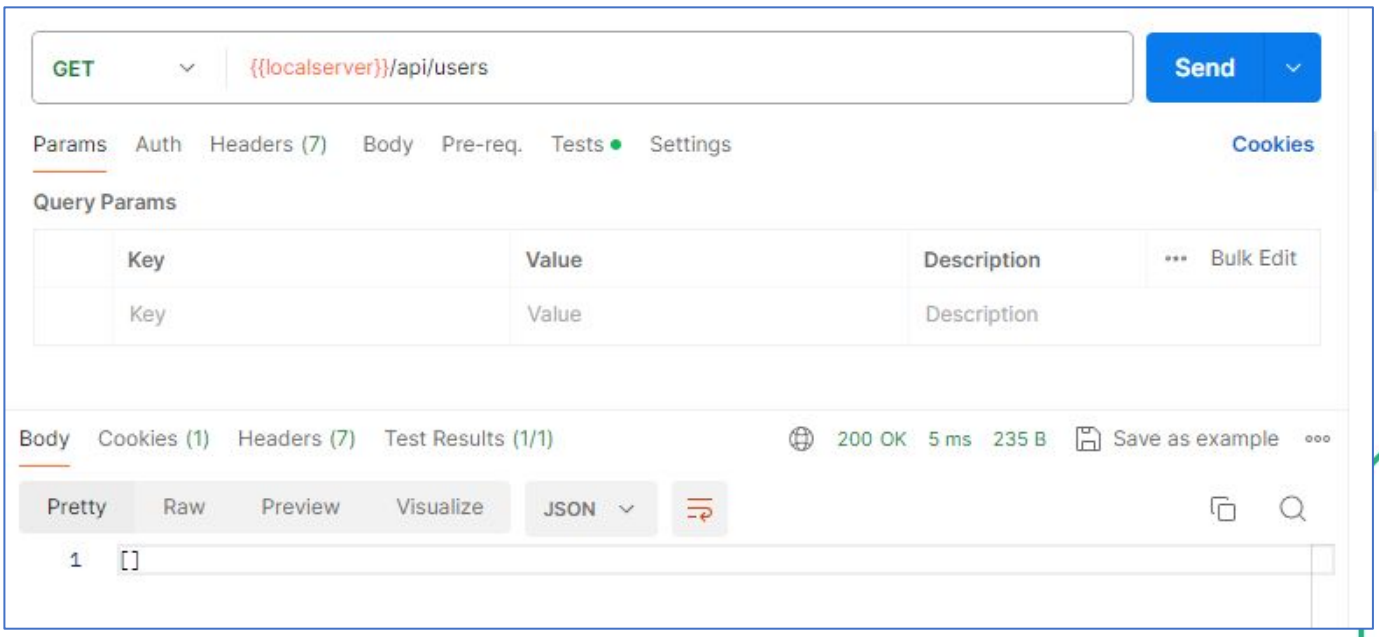
Banco de Dados RocksDB

Testes de Rotas

Para obter o Token JWT, precisamos fazer o login:



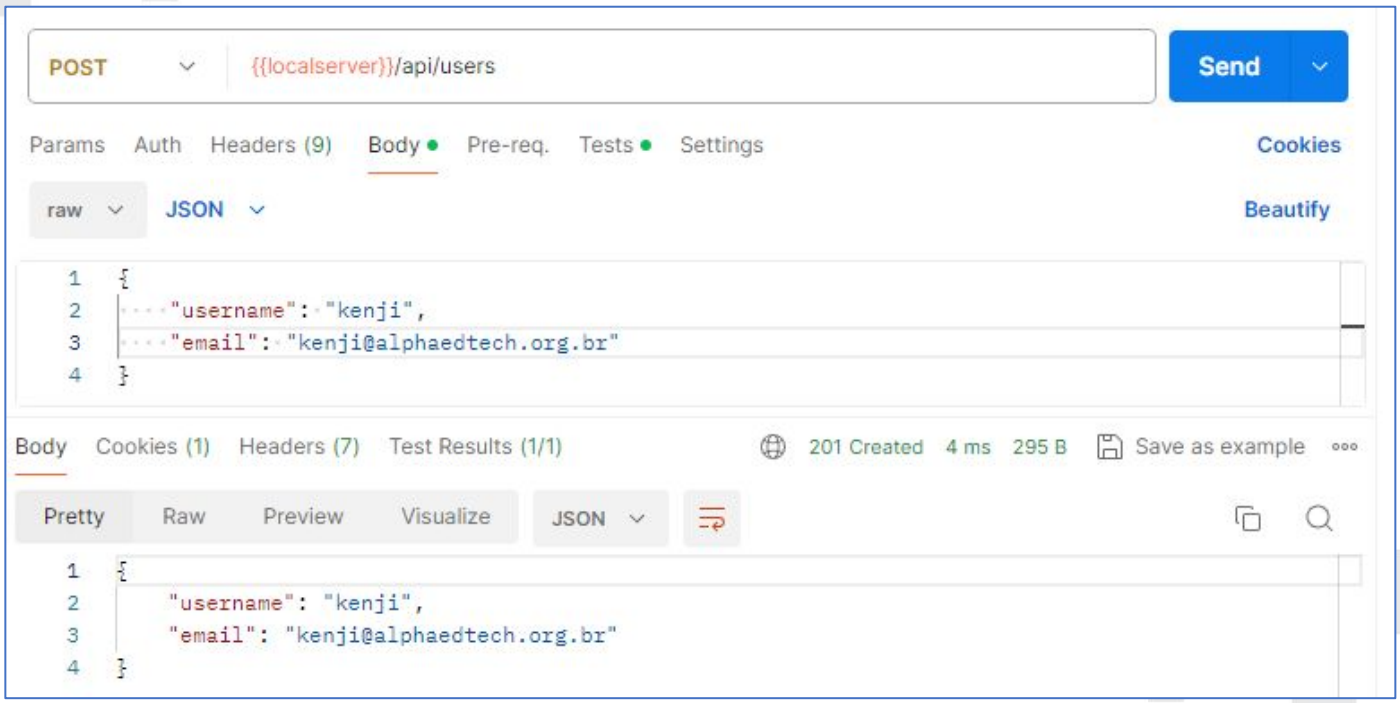
Busca inicial, sem usuários:



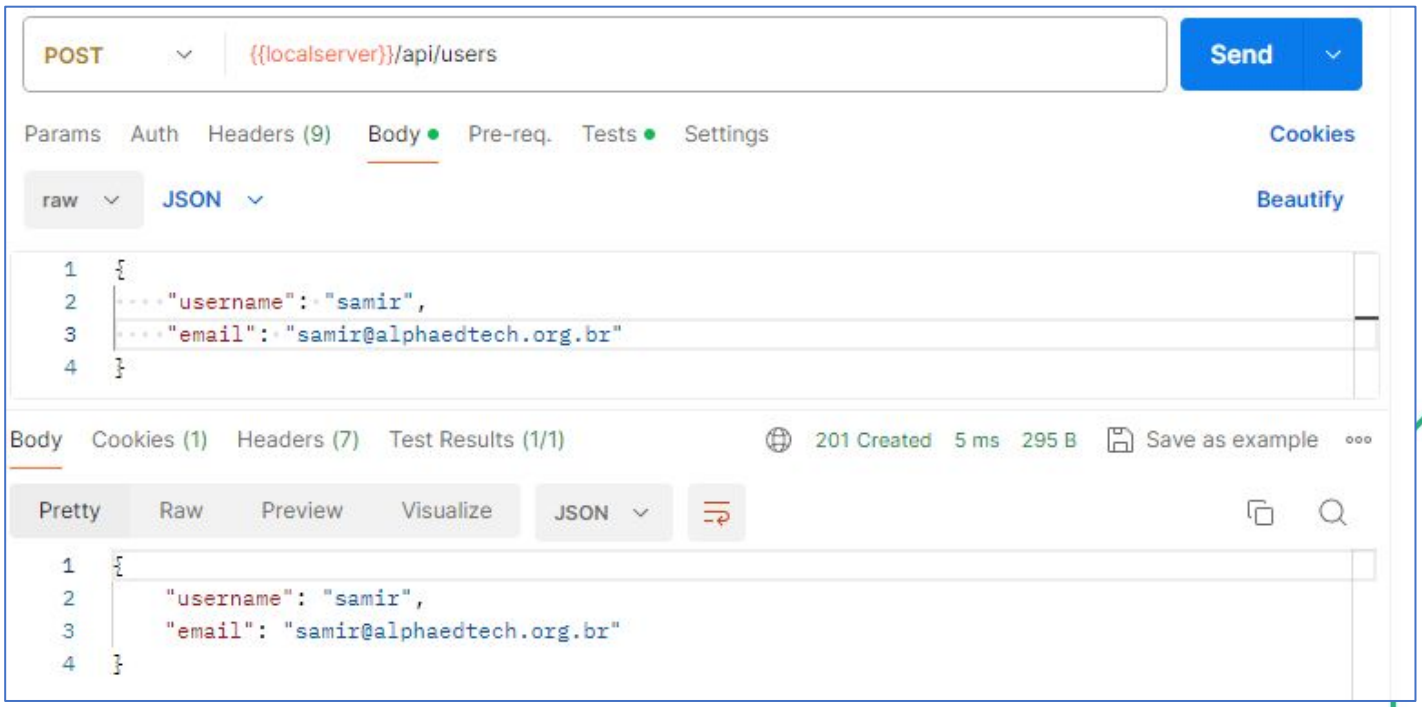
Banco de Dados RocksDB

Testes de Rotas

Vamos adicionar um usuário



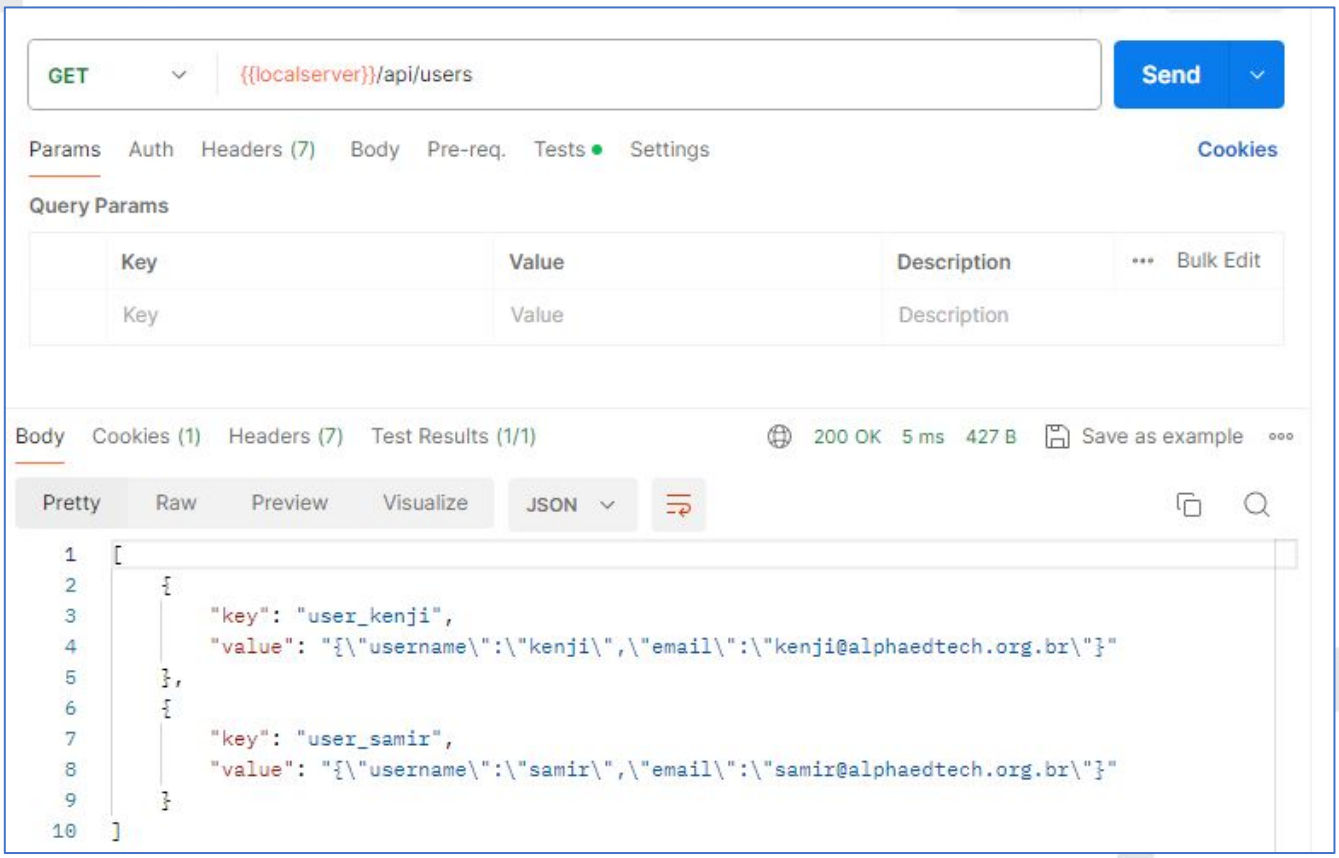
Mais um usuário:



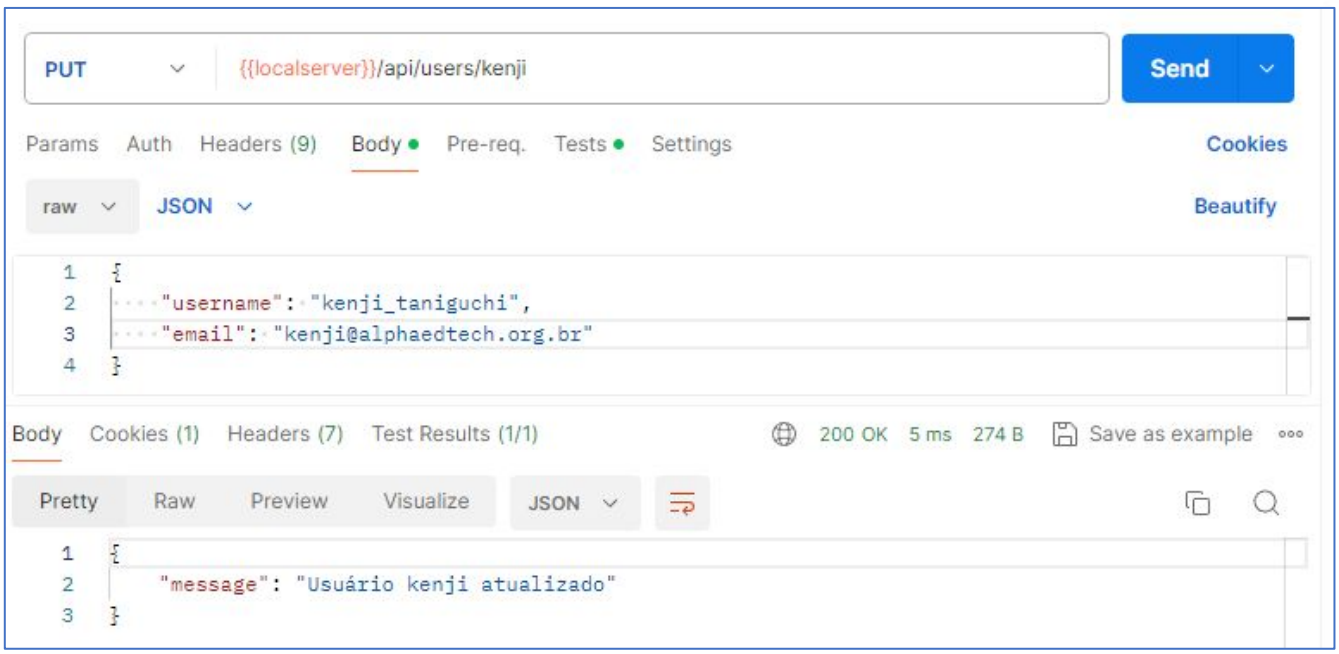
Banco de Dados RocksDB

Testes de Rotas

Listando usuários:



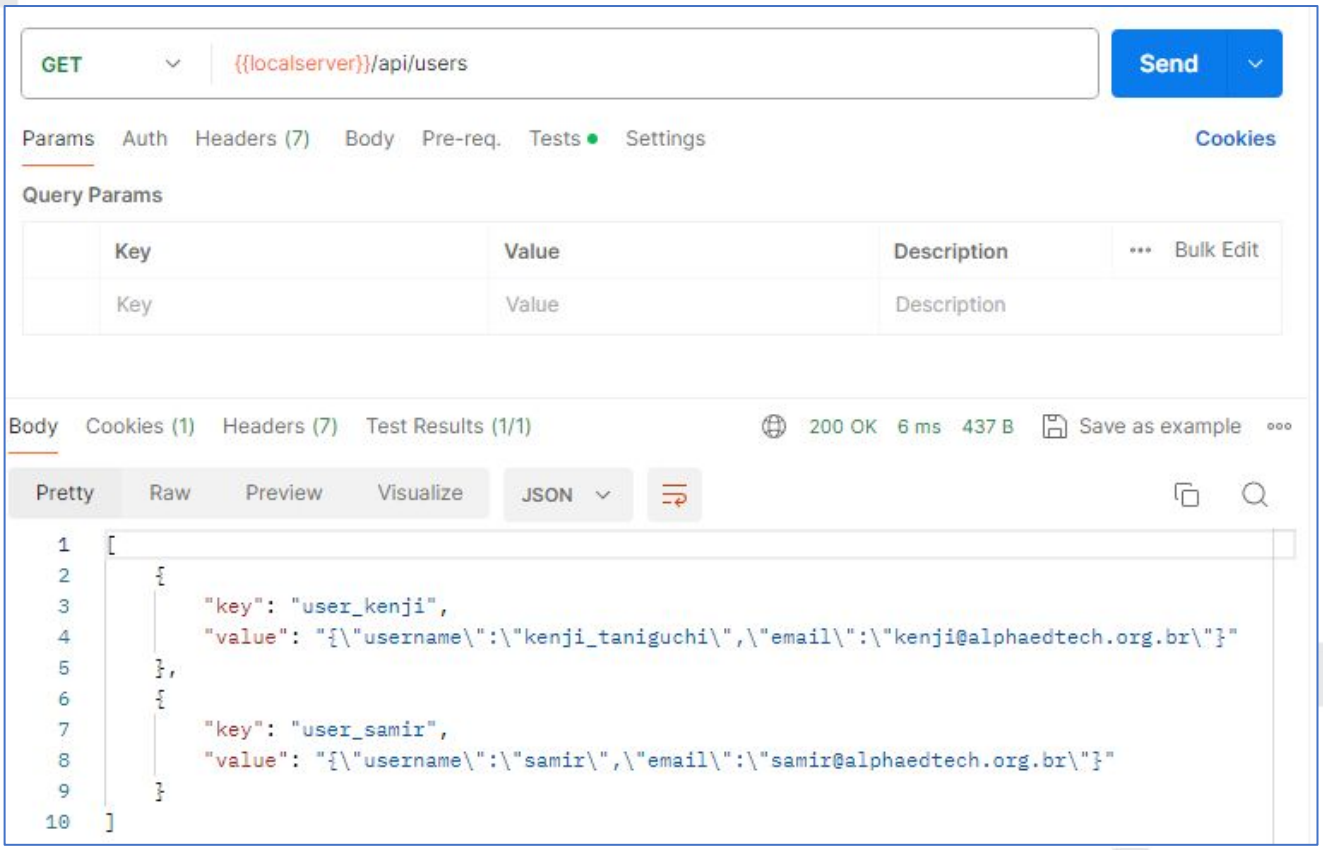
Alterando um usuário:



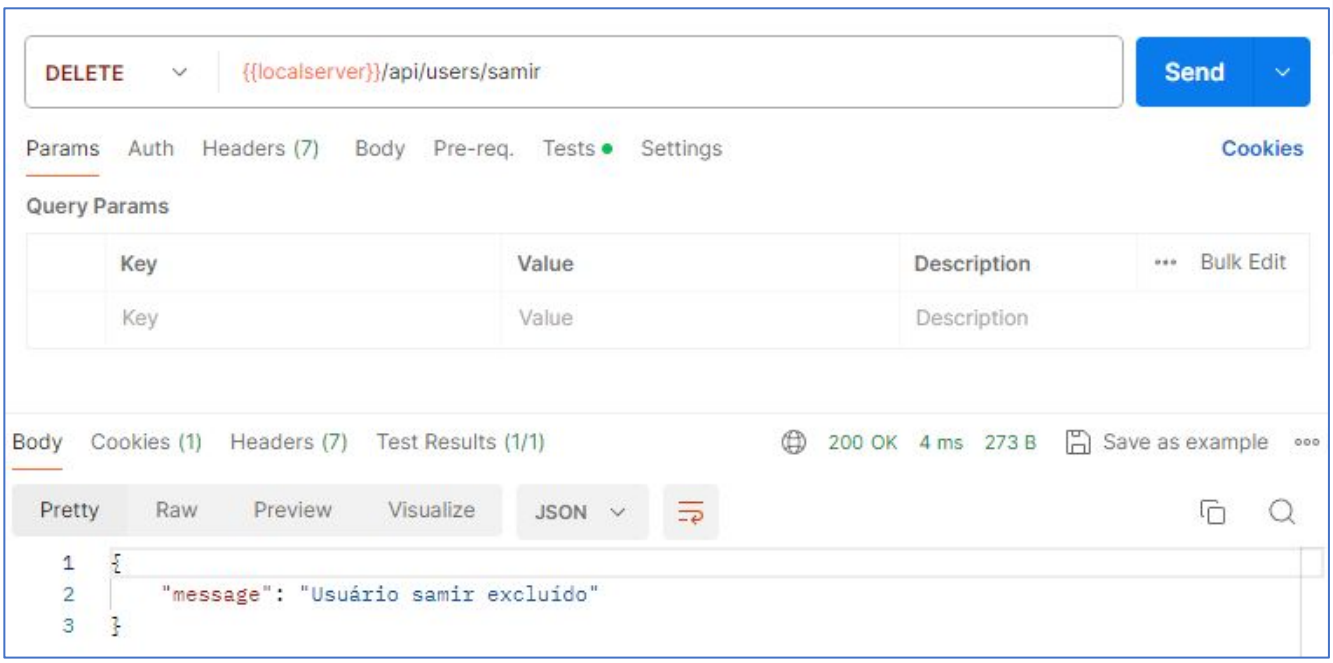
Banco de Dados RocksDB

Testes de Rotas

Listando usuários:



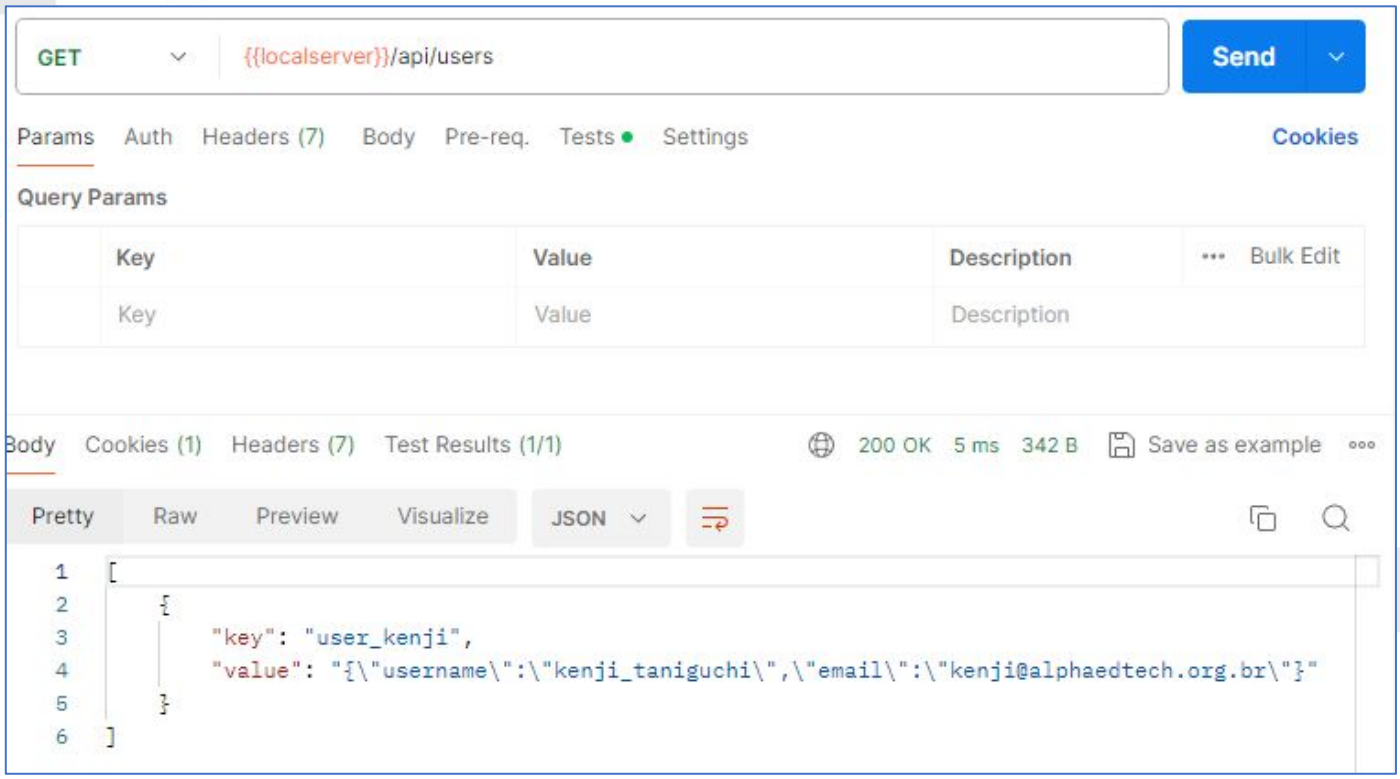
Apagando um usuário:



Banco de Dados RocksDB

Testes de Rotas

Listando usuários:



Agora os dados estão na pasta '/db_data/users':