Nome: Leonardo Rodrigues de Oliveira

QUESTÕES

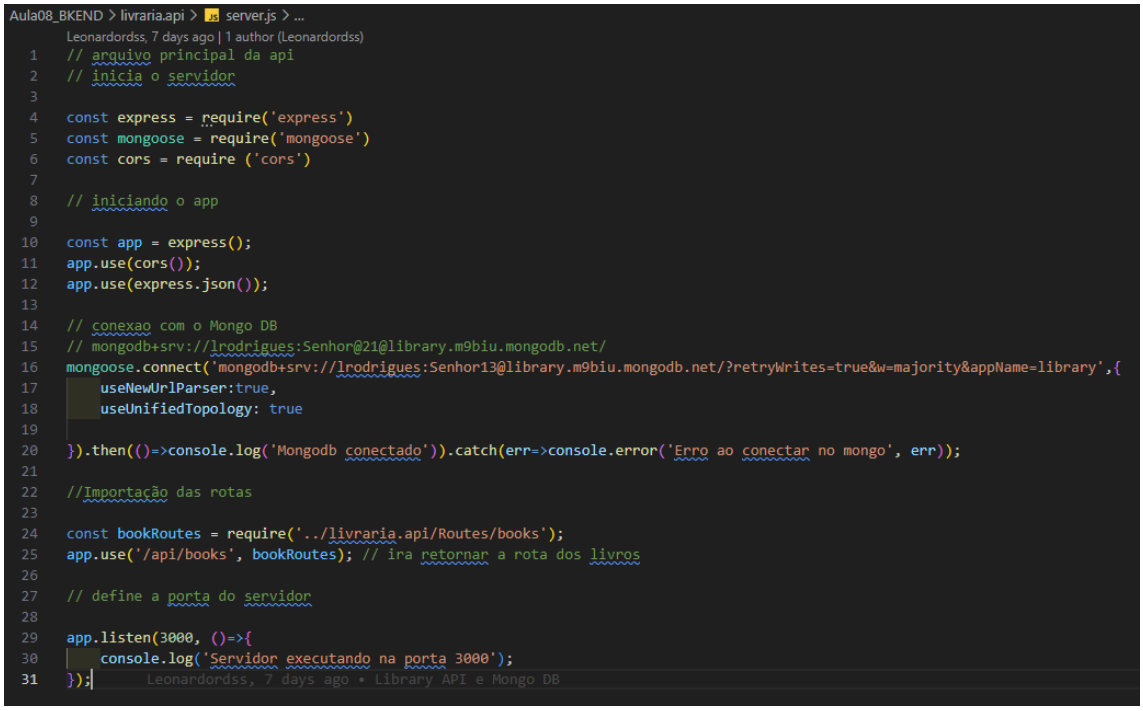
1. **Identificar as partes principais de um CRUD em Node.js e Express.**

Identifique e explique cada uma das seguintes partes:

- Configuração do servidor.
- Rota para listar os itens.
- Rota para adicionar um item.
- Conexão com o banco de dados.

RESPOSTA:

Esta parte do código é responsável por inicializar o servidor Express e configurar middlewares, como o uso de CORS e o parser de JSON.



```
Aula08_BKEND > livraria.api > server.js > ...
      Leonardordss, 7 days ago | 1 author (Leonardordss)
1     // arquivo principal da api
2     // inicia o servidor
3
4     const express = require('express')
5     const mongoose = require('mongoose')
6     const cors = require ('cors')
7
8     // iniciando o app
9
10    const app = express();
11    app.use(cors());
12    app.use(express.json());
13
14    // conexao com o Mongo DB
15    // mongodb+srv://lrodrigues:Senhor@21@library.m9biu.mongodb.net/
16    mongoose.connect('mongodb+srv://lrodrigues:Senhor13@library.m9biu.mongodb.net/?retryWrites=true&w=majority&appName=library',{
17        useNewUrlParser:true,
18        useUnifiedTopology: true
19
20    }).then(()=>console.log('Mongodb conectado')).catch(err=>console.error('Erro ao conectar no mongo', err));
21
22    //Importação das rotas
23
24    const bookRoutes = require('../livraria.api/Routes/books');
25    app.use('/api/books', bookRoutes); // ira retornar a rota dos livros
26
27    // define a porta do servidor
28
29    app.listen(3000, ()=>{
30        console.log('Servidor executando na porta 3000');
31    });      Leonardordss, 7 days ago • Library API e Mongo DB
```

Explicação:

Express: O Express é utilizado para criar o servidor e definir rotas.

Mongoose: Usado para conectar ao MongoDB, o banco de dados no qual os dados serão armazenados.

CORS: Permite que o servidor aceite requisições de diferentes origens, o que é útil em aplicações web.

JSON Parser: O express.json() permite que o servidor interprete o corpo das requisições no formato JSON.

Porta do Servidor: O servidor escuta na porta 3000.

- **Rota para Listar os Itens (GET)**

Essa rota permite recuperar todos os livros do banco de dados.

```
router.get('/', async(req,res)=>{
try{
    const books = await Book.find(); // busca todos os livros com o metodo find
    res.status(200).json(books) // retorna a lista de livros
}catch(error){
    res.status(500).json({message: 'Erro ao buscar os livros',error}) //retorna erro ao buscar o livro
}
});
```

Explicação:

Método GET: Utilizado para buscar recursos.

Book.find(): Método do Mongoose que busca todos os documentos da coleção "books".

Resposta: Retorna um status 200 e a lista de livros em formato JSON.

- **Rota para Adicionar um Item (POST)**

Essa rota permite adicionar um novo livro ao banco de dados.

```
router.post('/',async(req,res)=>{
const {title, author,year} = req.body; // Extrai os dados da requisicao
try{
    const newBook = new Book({title, author, year});
    await newBook.save();
    // 201 - ok - codigo de status
    res.status(201).json(newBook);
}catch(error){
    res.status(500).json({message:'Erro ao cadastrar livro'});
}

});
```

Explicação:

Método POST: Utilizado para criar novos recursos.

Criação do Livro: Um novo objeto Book é criado e salvo no banco de dados.

Resposta: Retorna um status 201, indicando que o recurso foi criado com sucesso, junto com os dados do novo livro.

- **Conexão com o Banco de Dados**

Esta parte do código configura a conexão com o MongoDB usando o Mongoose.

```
// conexao com o Mongo DB
// mongodb+srv://lrodrigues:Senhor@21@library.m9biu.mongodb.net/
mongoose.connect('mongodb+srv://lrodrigues:Senhor13@library.m9biu.mongodb.net/?retryWrites=true&w=majority&appName=library',{
    useNewUrlParser:true,
    useUnifiedTopology: true

}).then(()=>console.log('Mongodb conectado')).catch(err=>console.error('Erro ao conectar no mongo', err));
```

Explicação:

String de Conexão: Fornece as credenciais e o endereço do banco de dados.

useNewUrlParser e useUnifiedTopology: Opções de configuração do Mongoose para garantir que a conexão seja realizada corretamente.

Tratamento de Erros: O uso de .catch permite lidar com falhas na conexão, informando no console.

2. **O que é um servidor e qual sua função no back-end de uma aplicação?**

Um servidor, no contexto de desenvolvimento de back-end, é um software que processa e responde a solicitações feitas por clientes (geralmente navegadores web, aplicativos móveis, etc.). Ele funciona como o intermediário que recebe requisições, executa operações no banco de dados ou em outros sistemas, e devolve uma resposta ao cliente, sejam na forma de dados, mensagens de erro ou outros conteúdos.

O servidor é criado usando o framework Express, que é uma ferramenta popular para construir servidores web no Node.js. O servidor tem várias funções principais no back-end de uma aplicação.

3. **Qual a função do Node.js no desenvolvimento back-end?**

O Node.js desempenha um papel fundamental no desenvolvimento de back-end, especialmente por trazer o JavaScript (tradicionalmente uma linguagem usada apenas no front-end) para o lado do servidor. Ele oferece uma plataforma robusta e eficiente para construir servidores e executar código JavaScript fora do navegador.

4. **Como funcionam as rotas no Express.js e como são definidas?**

As rotas no Express.js são uma das funcionalidades mais importantes desse framework, pois permitem definir como o servidor vai responder a diferentes solicitações HTTP (GET, POST, PUT, DELETE, etc.) feitas pelos clientes. Cada rota corresponde a uma URL ou caminho específico, e a função associada à rota define o que acontecerá quando essa URL for acessada.

Como as Rotas Funcionam no Express.js

Uma rota no Express.js tem duas partes principais:

1. **Método HTTP**: Indica o tipo de requisição que a rota vai lidar, como GET, POST, PUT, DELETE.
2. **Caminho da URL**: O caminho que o cliente acessa para disparar a rota, como `/books` ou `/books/:id`.

5. **O que são controladores (controllers) no contexto do back-end e como eles organizam a lógica de negócio?**

Os controladores são componentes que têm a responsabilidade de lidar com a lógica de negócio de uma aplicação, recebendo as requisições HTTP dos usuários, processando esses dados e retornando a resposta apropriada. Eles atuam como intermediários entre os modelos (dados) e as rotas (caminhos que os usuários acessam).

Organização do Código**:** Em vez de incluir toda a lógica de negócio nas próprias rotas (o que poderia resultar em rotas longas e confusas), os controladores dividem as responsabilidades, organizando o código em funções específicas para cada operação, como criação, leitura, atualização e exclusão de dados (CRUD).

6. **O que é uma requisição HTTP e quais são suas partes principais?**

Uma **requisição HTTP (HyperText Transfer Protocol)** é o mecanismo que permite a comunicação entre clientes (geralmente navegadores ou aplicações) e servidores web. Ela é usada para enviar e receber dados através da web. Quando um cliente faz uma requisição a um servidor, ele está solicitando algum tipo de recurso ou serviço (por exemplo, uma página da web, um arquivo, dados de uma API). O servidor, por sua vez, processa a requisição e retorna uma **resposta HTTP** correspondente, que pode incluir o recurso solicitado ou uma mensagem indicando o status do processamento.

Partes Principais de uma Requisição HTTP

Uma requisição HTTP é composta por várias partes que carregam informações essenciais sobre o que o cliente deseja e como ele está enviando a solicitação ao servidor. Essas partes incluem:

Método HTTP: Indica o tipo de operação desejada, como:

- GET: Solicita a obtenção de um recurso (por exemplo, buscar uma página da web ou dados).
- POST: Envia dados ao servidor, geralmente para criar ou atualizar um recurso.
- PUT: Atualiza um recurso existente.
- DELETE: Remove um recurso.
- Outros métodos como PATCH, HEAD, OPTIONS, etc.

7. **O que são middlewares e como eles podem ser aplicados em rotas específicas no Express?**

Middlewares são funções intermediárias no Express.js que têm acesso ao **objeto de requisição (req)**, ao **objeto de resposta (res)** e à função **next()** na pilha de processamento de requisições. Eles são fundamentais para processar requisições antes de enviá-las para a lógica final (controladores) ou para fazer algum tipo de manipulação nas respostas antes de enviá-las ao cliente.

8. **Como o Mongoose é utilizado para MongoDB?**

O Mongoose é amplamente utilizado em aplicações Node.js para lidar com o MongoDB. Ele oferece um **modelo de objetos** que simplifica a manipulação de documentos do MongoDB, garantindo que os dados sejam tratados de forma consistente e segura.

Aqui está o fluxo básico de como o Mongoose é utilizado com o MongoDB:

1. **Conectar ao MongoDB**
2. **Definir um Schema (Esquema)**
3. **Criar um Model (Modelo) baseado no Schema**
4. **Realizar operações CRUD (Create, Read, Update, Delete)** no banco de dados.

9. **O que é validação de dados no back-end e como ela pode ser feita?**

A **validação de dados** no back-end é o processo de garantir que os dados recebidos de um cliente (por meio de uma requisição HTTP) atendam a certos critérios antes de serem processados ou armazenados no banco de dados. A validação protege a aplicação contra entradas incorretas, incompletas ou maliciosas, garantindo a integridade e segurança dos dados.

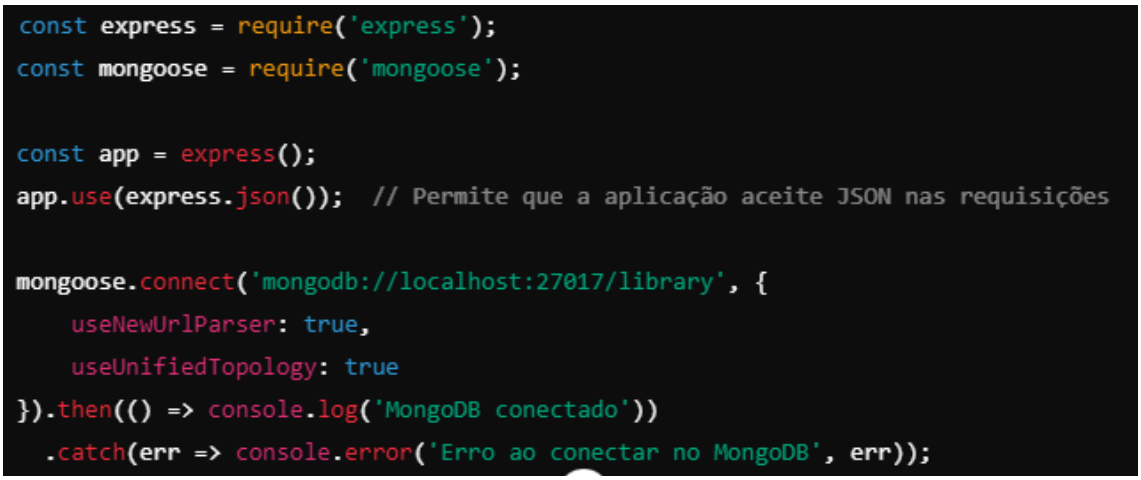
10. **O que é validação de dados no back-end e como ela pode ser feita com Mongoose?**

A **validação de dados** no back-end é o processo de garantir que os dados recebidos de uma requisição atendam a critérios específicos antes de serem processados ou salvos no banco de dados, protegendo a aplicação de entradas inválidas ou maliciosas.

Com **Mongoose**, a validação de dados é feita diretamente nos **schemas** (esquemas) definidos para os modelos. No esquema, você pode definir regras como tipos de dados, campos obrigatórios, limites numéricos e outras restrições.

11. **Como o Express.js é utilizado para criar rotas que permitem realizar as operações CRUD no banco de dados MongoDB?**

O **Express.js** é uma estrutura de servidor web no Node.js que facilita a criação de APIs e a definição de rotas para operações de **CRUD** (Create, Read, Update, Delete) em um banco de dados, como o MongoDB. Em conjunto com o **Mongoose**, que oferece uma interface para interagir com o MongoDB, o Express permite criar rotas que realizam essas operações de forma eficiente.



```
const express = require('express');
const mongoose = require('mongoose');

const app = express();
app.use(express.json());  // Permite que a aplicação aceite JSON nas requisições

mongoose.connect('mongodb://localhost:27017/library', {
    useNewUrlParser: true,
    useUnifiedTopology: true
}).then(() => console.log('MongoDB conectado'))
  .catch(err => console.error('Erro ao conectar no MongoDB', err));
```

12. **Qual é a função do modelo Book.js (usando Mongoose) no sistema, e como ele define a estrutura de um documento no MongoDB?**

Funções do Modelo Book.js

1. **Definição da Estrutura de Dados**: O modelo Book.js define como os dados de um livro serão organizados no MongoDB. Isso inclui quais campos são obrigatórios, quais tipos de dados são aceitos e quais validações devem ser aplicadas.
2. **Interação com o Banco de Dados**: O modelo permite realizar operações CRUD (Create, Read, Update, Delete) no banco de dados de forma estruturada e com menos chances de erro. Com ele, você pode facilmente criar novos livros, buscar livros existentes, atualizar informações e deletar registros.
3. **Validação**: Ele fornece a capacidade de validar dados antes de serem salvos no banco de dados. Por exemplo, você pode garantir que o campo de título não esteja vazio ou que o ano esteja dentro de um intervalo aceitável.

```
const mongoose = require('mongoose');

const bookSchema = new mongoose.Schema({
    title: {
        type: String,
        required: true // Campo obrigatório
    },
    author: {
        type: String,
        required: true
    },
    year: {
        type: Number,
        min: 1500, // Ano mínimo
        max: new Date().getFullYear() // Ano máximo é o ano atual
    },
    genre: {
        type: String,
        default: 'Fiction' // Gênero padrão
    },
    // Outros campos podem ser adicionados conforme necessário
});

// Cria o modelo 'Book' baseado no esquema definido
const Book = mongoose.model('Book', bookSchema);

module.exports = Book; // Exporta o modelo para uso em outras partes da aplicação
```

13. **Como a função findByIdAndUpdate é utilizada na rota de atualização (PUT) para modificar os dados de um livro?**

A função findByIdAndUpdate é uma maneira eficaz de atualizar documentos no MongoDB. Ela permite que você localize um documento pelo seu ID e o modifique em uma única operação, simplificando a lógica na rota de atualização (PUT) e melhorando a legibilidade do código.

```
Model.findByIdAndUpdate(id, update, options, callback);
```

14. **Por que utilizamos try...catch nas operações CRUD e como tratamos erros em requisições que falham?**

O uso de try...catch nas operações CRUD é fundamental para garantir que sua aplicação possa lidar com erros de forma robusta e amigável ao usuário. Ele permite que você capture exceções, registre problemas para análise posterior e envie respostas adequadas aos usuários, melhorando a experiência geral e a estabilidade da aplicação.

```
app.put('/api/books/:id', async (req, res) => {
    const { id } = req.params; // Extrai o ID do livro
    const { title, author, year } = req.body; // Extrai os dados do corpo da requisição

    try {
        // Tenta atualizar o livro no banco de dados
        const updatedBook = await Book.findByIdAndUpdate(id, { title, author, year }, { ne

        // Verifica se o livro foi encontrado
        if (updatedBook) {
            res.status(200).json(updatedBook); // Retorna o livro atualizado
        } else {
            res.status(404).json({ message: 'Livro não encontrado' }); // Retorna erro se
        }
    } catch (error) {
        // Captura e trata o erro
        console.error(error); // Log do erro no console para diagnóstico
        res.status(500).json({ message: 'Erro ao atualizar livro', error }); // Retorna um
    }
```

15. **Como o método find() no Mongoose funciona para retornar a lista de todos os livros na rota GET, e como ele interage com o MongoDB?**

O método find() do Mongoose permite consultar documentos em uma coleção do MongoDB de forma simples e intuitiva. Ao ser chamado em uma rota GET, ele busca todos os livros e retorna os dados ao cliente em formato JSON. A interação com o MongoDB é feita de maneira automática, e o Mongoose cuida de toda a lógica de consulta, o que simplifica muito o desenvolvimento de aplicações que usam banco de dados.

```
app.get('/api/books', async (req, res) => {
    try {
        const books = await Book.find(); // Busca todos os livros
        res.status(200).json(books); // Retorna a lista de livros
    } catch (error) {
        res.status(500).json({ message: 'Erro ao buscar livros', error }); // Retorna erro
    }
});
```

16. **O que o comando npm init -y faz ao criar um novo projeto Node.js?**

O comando npm init -y é utilizado para criar um novo projeto Node.js e inicializar um arquivo package.json automaticamente. Aqui estão os detalhes sobre o que esse comando faz:

- **Criação do arquivo package.json**: O package.json é um arquivo essencial para qualquer projeto Node.js. Ele contém informações sobre o projeto, como o nome, a versão, a descrição, as dependências, os scripts, entre outros.
- **Uso da opção -y**: Ao adicionar a opção -y, você está dizendo ao npm para aceitar todas as configurações padrão ao criar o package.json. Isso significa que ele será gerado sem solicitar interativamente informações como o nome do projeto, a versão, a descrição, etc. Com isso, o npm criará um arquivo package.json básico com valores padrão.

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

17. **O que o método app.get() faz na API Express?**

O método app.get() é uma função fornecida pelo framework Express.js que é usada para definir rotas para lidar com requisições HTTP do tipo **GET**. Aqui está uma visão geral do que ele faz:

- **Definição de Rota**: app.get() permite que você especifique uma rota na sua aplicação Express. Quando um cliente faz uma requisição GET para essa rota, a função de callback associada é executada.
- **Execução da Função de Callback**: A função de callback recebe dois parâmetros principais: req (o objeto de requisição) e res (o objeto de resposta). Você pode usar esses objetos para processar a requisição e enviar uma resposta ao cliente.

```
const express = require('express');
const app = express();

app.get('/api/books', (req, res) => {
    // Simula a recuperação de livros
    const books = [
        { id: 1, title: '1984', author: 'George Orwell' },
        { id: 2, title: 'Brave New World', author: 'Aldous Huxley' }
    ];
    res.status(200).json(books); // Retorna a lista de livros em formato JSON
});

// Inicia o servidor
app.listen(3000, () => {
    console.log('Servidor executando na porta 3000');
});
```

18. Qual a função da linha app.listen(port, () => {...}); no código da API?

A linha app.listen(port, () => {...}); é crucial porque inicia o servidor Express, permitindo que ele escute requisições HTTP na porta especificada. O callback é usado para informar que o servidor está funcionando corretamente, proporcionando um feedback ao desenvolvedor. Essa linha é a porta de entrada para sua API, permitindo a comunicação entre clientes e seu aplicativo.

```
const express = require('express');
const app = express();
const port = 3000; // Define a porta que o servidor irá escutar

app.get('/', (req, res) => {
    res.send('Hello World!'); // Responde com "Hello World!" para requisições na raiz
});

// Inicia o servidor
app.listen(port, () => {
    console.log(`Servidor executando na porta ${port}`); // Mensagem exibida no console
});
```

19. Qual é o papel do require('express') no início do código?

A linha require('express') é crucial porque importa o módulo Express, permitindo que você utilize suas funcionalidades na construção de aplicações web. A partir dessa importação, você pode criar uma instância da aplicação, definir rotas, aplicar middleware, e implementar a lógica da sua API ou aplicação web de forma eficiente e organizada. O Express é amplamente utilizado devido à sua simplicidade e flexibilidade, tornando o desenvolvimento de aplicações Node.js mais fácil e produtivo.

```
const express = require('express'); // Importa o módulo Express
const app = express(); // Cria uma instância da aplicação Express

// Define uma rota para a raiz
app.get('/', (req, res) => {
    res.send('Hello World!'); // Responde com "Hello World!" quando acessa a raiz
});

// Inicia o servidor
app.listen(3000, () => {
    console.log('Servidor executando na porta 3000');
});
```

20. **Como você pode testar se a API está funcionando corretamente sem o usar o navegador?**

Para testar se a API está funcionando corretamente sem usar o navegador, você pode utilizar ferramentas de teste de API como Postman, Insomnia ou cURL. Também é possível escrever testes automatizados com bibliotecas como Jest ou Mocha, ou utilizar ferramentas de linha de comando como HTTPie. Monitorar logs do servidor pode ajudar a identificar problemas e entender como as requisições estão sendo tratadas.

```
const request = require('supertest');
const app = require('../app'); // Importa seu aplicativo Express

describe('GET /api/books', () => {
    it('should return a list of books', async () => {
        const response = await request(app).get('/api/books');
        expect(response.statusCode).toBe(200);
        expect(response.body).toBeInstanceOf(Array);
    });
});
```