Desafio Backend - Semana da Indústria

Criando seu primeiro Aplicativo de Gestão de Tarefas BackEnd com Node.Js, Express e MySQL

Descrição:

Você foi designado para criar uma aplicação simples de gerenciamento de tarefas. Esta aplicação deve permitir ao usuário criar, ler, atualizar e deletar (CRUD) tarefas. Para isso, você deve utilizar Node.js, o framework Express e o banco de dados MySQL.

Capacidades Trabalhadas:

- Aplicar linguagem de programação por meio do ambiente de desenvolvimento (ide)
- Aplicar métodos e técnicas de programação
- Empregar comentários para a documentação do código fonte.
- Integrar banco de dados por meio de linguagem de programação
- Identificar erros de acordo com o requisito do programa
- Aplicar os princípios de organização do trabalho estabelecidos no planejamento e no exercício de suas atividades

Conhecimentos:

- Conexão com banco de dados
- Preparação do ambiente: Ferramentas(funções,repositórios, IDE), instalação(configurações, requisitos mínimos), programação de aplicativos.
- Trabalho e profissionalismo: Planejamento da rotina, flexibilidade, resultados dos dados.
- Técnicas de programação: formatação, documentação de código, reutilização de código, técnicas de otimização de código.
- Ética profissional: Princípios de conduta ética do serviço(sigilo, prudência, imparcialidade, honestidade).

Requisitos do Projeto:

Modelo de Dados:

Crie um banco de dados no MySQL com o nome gestao_tarefas

Crie uma tabela tarefas no MySQL com os seguintes campos:

id: um número inteiro que se auto incrementa (este será o campo chave primária)
titulo: uma string para o título da tarefa
descricao: uma string para a descrição da tarefa
status: uma string para o status da tarefa (ex: "pendente", "em andamento", "concluído")
data_criacao: uma data para quando a tarefa foi criada
data_conclusao: uma data para quando a tarefa for concluída

API:

Crie as rotas para realizar operações CRUD na tabela tarefas. As rotas devem ser:

POST /tarefas: Cria uma nova tarefa
GET /tarefas: Lista todas as tarefas
GET /tarefas/:id: Mostra uma tarefa específica
PUT /tarefas/:id: Atualiza uma tarefa específica
DELETE /tarefas/:id: Deleta uma tarefa específica

Código:

O código deve estar bem organizado, comentado e fácil de entender. Cada função deve ter uma breve descrição do que faz.

Processo de Avaliação:

No final da semana, cada aluno apresentará o projeto para a turma. Durante a apresentação, o aluno deve:

Explicar o código que criou.
Discutir as dificuldades que enfrentou e como as superou.
Apresentar as funcionalidades adicionais que tenha implementado além dos requisitos básicos.
Demonstrar a funcionalidade do CRUD ao vivo.

Recurso Adicional:

É recomendado o uso do Postman ou similar para testar as rotas da API antes de implementar a interface do usuário, se houver.

Objetivo:

O objetivo deste desafio é familiarizar-se com a criação de uma API RESTful usando Node.js, Express e MySQL, ao mesmo tempo que ganha experiência prática com o desenvolvimento backend e gestão de banco de dados. Boa sorte!

Lembre-se: O foco é aprender e se divertir no processo. Não tenha medo de cometer erros, eles são uma parte valiosa do processo de aprendizado!

Documentação Para Desenvolvimento do Desafio:

-criar uma pasta chamada **app_gestao_tarefas**
-entrar na pasta e digitar:
npm init -y

-instalar o express:
npm i express

-abrir o vscode com a pasta **app_gestao_tarefas**

-criar o arquivo app.js
- para rodar o aplicação, salve tudo e digite no terminal:
node app.js

Como conectar um banco MySQL a uma aplicação Node.js?

Para conectar um banco de dados MySQL a uma aplicação Node.js, você pode usar um driver MySQL para Node.js, como o pacote `mysql` que é amplamente usado. Aqui estão os passos:

1. **Instalação do pacote:**

Primeiro, você precisará instalar o driver MySQL. Isso pode ser feito facilmente com o npm, que é o gerenciador de pacotes do Node.js. Abra um terminal no diretório do seu projeto e execute o seguinte comando:

npm install mysql

2. **Criando a conexão:**
Depois de instalar o pacote, você pode usar o seguinte código para iniciar uma conexão com o seu banco de dados MySQL:

```
//arquivo javascript app.js
var mysql      = require('mysql');
var connection = mysql.createConnection({
  host     : 'localhost',
  user     : 'meu_usuario',
  password : 'minha_senha',
  database : 'meu_banco_de_dados'
});

connection.connect(function(err) {
  if (err) {
    console.error('Erro ao conectar: ' + err.stack);
    return;
  }

  console.log('Conectado como id ' + connection.threadId);
});
```

Não se esqueça de substituir 'localhost', 'meu_usuario', 'minha_senha' e 'meu_banco_de_dados' pelos seus dados.

3. **Executando consultas:**
Agora você pode executar consultas SQL no seu banco de dados MySQL. Aqui está um exemplo de como fazer isso:

```
//codigo javascript
connection.query('SELECT * FROM minha_tabela', function (error, results, fields) {
  if (error) throw error;
  console.log('Os resultados são: ', results);
});
//fim codigo javascript
```

4. **Encerrando a conexão:**
Por último, quando terminar, você deve encerrar a conexão com o banco de dados:

```
//codigo javascript
connection.end();
```

Lembre-se de que gerenciar corretamente as conexões com o banco de dados é crucial para o desempenho do seu aplicativo. Não se esqueça de tratar os erros e de fechar as conexões quando não estiverem mais sendo usadas.

Como fazer as rotas CRUD simples usando o express para uma aplicação de livros?

Antes de prosseguirmos, é importante lembrar que precisamos do pacote 'express' e do 'mysql' que mencionei anteriormente. Vamos instalar o Express também:

npm install express

Agora, aqui está um exemplo básico de como você pode configurar rotas CRUD para uma aplicação de livros:

```
//código javascript em app.js
const express = require('express');
const app = express();
const mysql = require('mysql');

// Configurando o express para aceitar JSON
app.use(express.json());

// Conexão com o banco de dados
var connection = mysql.createConnection({
  host: 'localhost',
  user: 'meu_usuario',
  password: 'minha_senha',
  database: 'meu_banco_de_dados',
});

connection.connect(function (err) {
  if (err) throw err;
  console.log('Conectado!');
});

// CREATE
app.post('/livros', (req, res) => {
  const livro = req.body;

  const sql = 'INSERT INTO Livros SET ?';
  connection.query(sql, livro, (error, result) => {
```

```
    if (error) throw error;
    res.status(201).json({ id: result.insertId, ...livro });
  });
});

// READ
app.get('/livros', (req, res) => {
  const sql = 'SELECT * FROM Livros';

  connection.query(sql, (error, results) => {
    if (error) throw error;
    res.json(results);
  });
});

app.get('/livros/:id', (req, res) => {
  const id = req.params.id;
  const sql = 'SELECT * FROM Livros WHERE id = ?';

  connection.query(sql, id, (error, results) => {
    if (error) throw error;
    res.json(results[0]);
  });
});

// UPDATE
app.put('/livros/:id', (req, res) => {
  const id = req.params.id;
  const newBook = req.body;

  const sql = 'UPDATE Livros SET ? WHERE id = ?';
  connection.query(sql, [newBook, id], (error) => {
    if (error) throw error;
    res.status(204).end();
  });
});

// DELETE
app.delete('/livros/:id', (req, res) => {
  const id = req.params.id;

  const sql = 'DELETE FROM Livros WHERE id = ?';
  connection.query(sql, id, (error) => {
    if (error) throw error;
    res.status(204).end();
  });
});
```

```
// Configurando o servidor
const port = 3000;
app.listen(port, () => {
  console.log(`Servidor rodando na porta ${port}`);
});
```

Este é um exemplo simples e não contempla tratamento de erros apropriado, como verificar se um livro realmente existe antes de tentar atualizar ou excluir, e muitas outras considerações de uma aplicação real. Além disso, em uma aplicação real, você provavelmente gostaria de usar um pool de conexões e talvez até um ORM (Object-Relational Mapping) como Sequelize, mas espero que isso seja útil como um ponto de partida.

STATUS CODE HTTP

res.status(200) sinaliza que a operação foi bem sucedida.
res.status(400) sinaliza um erro.
status de 100 a 199 são respostas de informação.
200 a 299 são respostas de sucesso.
300 a 399 são para redirecionamentos.
400 a 499 são para indicar erros do cliente.
500 a 599 indicam erros do servidor.

Correção de bugs/erros

no app.js inserir no início do arquivo

```
//faz a conversão de JSON para javascript
const bodyParser = require('body-parser');
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
```

Ficha de Avaliação do Desafio: Gestão de Tarefas com Node.js, Express e MySQL

Nome do Aluno:

Data da Avaliação:

Avaliador:

Capacidades Avaliadas (Cada item será avaliado em uma escala de 1 a 5, sendo 1: Inadequado e 5: Excelente):

1. Aplicar linguagem de programação por meio do ambiente de desenvolvimento (IDE): _____

   Comentários:

2. Aplicar métodos e técnicas de programação: _____

   Comentários:

3. Empregar comentários para a documentação do código fonte: _____

   Comentários:

4. Integrar banco de dados por meio de linguagem de programação: _____

   Comentários:

5. Identificar erros de acordo com o requisito do programa: _____

   Comentários:

6. Aplicar os princípios de organização do trabalho estabelecidos no planejamento e no exercício de suas atividades: _____

   Comentários:

Apresentação Final:

7. Qualidade da explicação do código: _____

   Comentários:

8. Discussão sobre as dificuldades e desafios enfrentados: _____

   Comentários:

9. Demonstração de funcionalidades adicionais além dos requisitos básicos: _____

   Comentários:

10. Demonstração da funcionalidade do CRUD ao vivo: _____

    Comentários:

Avaliação Geral:

Notas (Avaliação Geral): _____

Comentários:

Feedback para o Aluno:
