Introdução ao Desenvolvimento WEB - Calculadora Simples

Artur Moreno Duarte da Costa, Juliana Ferreira Gualorth
Universidade Federal de Rondônia - Campus BR 364, KM 9,5 (UNIR)
Porto Velho - RO - Brasil

Abstract. asdhjashd uhas udhuas hudhu ashduuas dhud huashduhasu dhuashduhasudhaushduashduhasudhuasduhasudhaushduhasudhuashudhaushduhasudhuashduash udasudhuashduashudhasuhduashduhasudhuashduashudhasudhuashduashduashduashudhasudhasu dhasudhughusdfghu usdhfu sdh fuhds ufdsufhusd fusdfusdh fuds hfu hdsuf hsdu fhsdu fhsdufh usd hfsdu fsdu fhudys hfusdhf sdufh usdhfuds dfh hdufh.

Resumo. Este relatório descreve o projeto final da disciplina de introdução ao desenvolvimento web, o qual se trata de uma aplicação de calculadora simples com histórico de operações, serão apresentadas as ferramentas utilizadas no desenvolvimento, funcionalidades implementadas, instruções para execução, link para repositório do projeto e apresentação.

1.Introdução

Conforme estudado na disciplina de introdução ao desenvolvimento WEB, aplicações WEB possuem características indispensáveis para seu funcionamento, segurança e boa experiência de usuário, sendo essas características a persistência de dados, design limpo, acessibilidade de informações e funcionalidades, integridade de dados, processos de autenticação, dentre muitas outras.

Para este projeto, fora desenvolvida uma aplicação de calculadora simples que mantém um histórico de operações para que outros usuários possam visualizar, usuários estes que poderão se conectar à aplicação através do front-end em uma área para logar ao site, também contendo persistência de dados como os usuários cadastrados e operações salvas no histórico.

2.Ferramentas utilizadas

No desenvolvimento de aplicações web, as ferramentas necessárias são diversas e a decisão fica a mercê dos, desenvolvedores, partes interessadas ou até mesmo do cliente, como as possibilidades são tamanhas, a seguir serão apresentadas aquelas que foram utilizadas neste projeto.

2.1.Linguagem

Nossa escolha para o desenvolvimento da aplicação foi a linguagem de programação javascript por ser leve e especializada em desenvolvimento web, além de no momento a principal linguagem do mercado, o que disponibilizada um ecossistema abrangente e uma comunidade enorme, facilitando a obtenção de informações e frameworks que auxiliam no desenvolvimento.

Para manipulação do backend da aplicação utilizamos o node, um software de código aberto que permite a conexão com banco de dados, interpretação de códigos javascript fora de um navegador, facilidade de instalação de novos frameworks e etc.

O front-end foi desenvolvido em HTML, a mais básica linguagem de marcação de hipertexto, que permite o anexo de links, determinação e áreas específicas, divisão de páginas web dentre muitas outras funcionalidades, também foi utilizado CSS (Cascading Style Sheets) para estilização das páginas.

2.2.MySQL

Para persistência de dados foi escolhido o banco MySQL para manipulação de dados relacionais pois foi o banco apresentado para os alunos na disciplina, sendo de simples obtenção e fácil manipulação.

2.3.Bcrypt.js

Para encriptação de dados sensíveis como senhas de usuários, a ferramenta bcrypt.js foi escolhida, com ela implementamos um método de criptografar as senhas através de uma função hash, fazendo com que apenas o usuário saiba de fato qual é a sua senha.

2.4.Express

O framework express foi utilizado para configuração básica do backend em junção com o Node e definição de rotas para a aplicação.

2.5.Visual Studio Code

Como IDE a escolha foi o visual studio code, desenvolvido pela microsoft, por ser a mais utilizada no mercado e permitir a instalação de aplicações auxiliares de forma simples e rápida para incremento de funcionalidades da aplicação.

3.Funcionalidades

Agora serão apresentadas todas as funcionalidades da aplicação, que se trata de uma calculadora simples com um histórico de operações, serão discutidos os arquivos mais importantes e serão explicados passo-a-passo.

3.1.Conexão com o banco de dados (db.js)

```
const mysql = require("mysql2");
const db = mysql.createConnection({
   host: "localhost", // Endereço do servidor MySQL
   user: "root",
   password: "root",
   database: "calculadora", // Nome do banco de dados
});
db.connect((err) => {
   if (err) {
      console.error("Erro ao conectar ao banco de dados:", err);
   } else {
      console.log("Conexão bem-sucedida ao banco de dados MySQL.");
   }
});
module.exports = db;
```

3.2.Manipulando dados no banco/servidor (api.js)

Atribuindo os métodos POST, GET e DELETE as rotas necessárias. POST para mandar dados de operações e novos usuários ao banco, GET para retornar dados de operações anteriores do banco para o front-end e DELETE para excluir operações presentes no banco MySQL.

```
// Salvar uma operação no banco de dados
router.post("/operacoes", (req, res) => {
   const { operacao, resultado } = req.body;
    const sql = "INSERT INTO operacoes (operacao, resultado) VALUES (?,
?)";
   db.query(sql, [operacao, resultado], (err, result) => {
      if (err) {
         console.error("Erro ao salvar a operação:", err);
         res.status(500).json({ error: "Erro ao salvar a operação" });
      } else {
           res.status(201).json({ message: "Operação salva com sucesso"
});
      }
```

```
    });
});

// Recuperar todas as operações do banco de dados
router.get("/operacoes", (req, res) => {
    const sql = "SELECT * FROM operacoes";
    db.query(sql, (err, result) => {
        if (err) {
            console.error("Erro ao recuperar as operações:", err);
             res.status(500).json({ error: "Erro ao recuperar as operações"
});
        } else {
            res.status(200).json(result);
        }
    });
});

// Rota para limpar os dados de operações do banco de dados
router.delete("/operacoes", (req, res) => {
    const sql = "DELETE FROM operacoes";
    db.query(sql, (err, result) => {
        if (err) {
            console.error("Erro ao limpar as operações:", err);
              res.status(500).json({ error: "Erro ao limpar as operações"
});
        } else {
                res.status(200).json({ message: "Operações excluídas com
sucesso" });
        }
    });
});

// Rota de Cadastro de Usuário
router.post("/cadastro", (req, res) => {
    const { nome, email, senha } = req.body;

    // Criptografar a senha antes de armazenar no banco de dados
    bcrypt.hash(senha, 10, (err, hash) => {
        if (err) {
            console.error("Erro na criptografia de senha:", err);
                res.status(500).json({ success: false, message: "Erro no
cadastro" });
        } else {
```

```
            const sql =
                  "INSERT INTO usuarios (nome, email, senha) VALUES (?, ?,
?)";
            db.query(sql, [nome, email, hash], (err, result) => {
               if (err) {
                  console.error("Erro no cadastro:", err);
                  res.status(500).json({
                     success: false,
                     message: "Erro no cadastro",
                  });
               } else {
                  res.status(201).json({
                     success: true,
                     message: "Cadastro bem-sucedido",
                  });
               }
            });
         }
      });
});

// Rota de Login de Usuário
router.post("/login", (req, res) => {
   const { email, senha } = req.body;
   const sql = "SELECT * FROM usuarios WHERE email = ?";

   db.query(sql, [email], (err, results) => {
      if (err) {
         console.error("Erro no login:", err);
             res.status(500).json({ success: false, message: "Erro no
login" });
      } else if (results.length === 0) {
         res.status(401).json({
            success: false,
            message: "Usuário não encontrado",
         });
      } else {
         const user = results[0];
         bcrypt.compare(senha, user.senha, (err, result) => {
            if (err || !result) {
               res.status(401).json({
                  success: false,
                  message: "Senha incorreta",
```

```
                });
            } else {
                res.status(200).json({
                    success: true,
                    message: "Login bem-sucedido",
                });
            }
        });
    }
  });
});

module.exports = router;
```

3.3.Definindo Rotas e arquivo de inicialização(app.js)

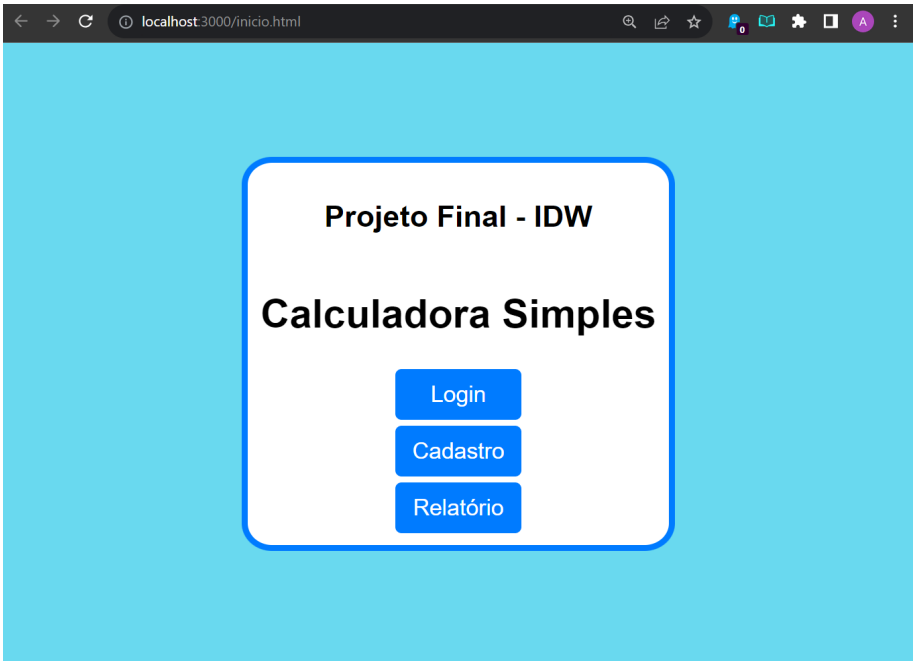
Método GET para definir as rotas “/” como página inicial e “/calculadora” como página da aplicação calculadora.

```
app.use(express.static(path.join(__dirname, "src")));

app.use(express.json());

app.get("/", (req, res) => {
  res.sendFile(path.join(__dirname, "src", "inicio.html"));
});

app.get("/calculadora", (req, res) => {
  res.sendFile(path.join(__dirname, "src", "index.html"));
});

app.use("/api", apiRouter);

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
    console.log(`Servidor em execução no endereço
http://localhost:3000/inicio.html`);
});
```

3.4.Página inicial (inicio.html)

Esta página é o “rosto” da aplicação e contém botões para navegação entre as páginas de login, cadastro e anexo do relatório.

```
<body>
      <div class="bem-vindo">
         <h2>Projeto Final - IDW</h2>
         <h1>Calculadora Simples</h1>
         <button class="btn-bv" onclick="location.href='login.html'">
            Login
         </button>
        <button class="btn-bv" onclick="location.href='cadastro.html'">
            Cadastro
         </button>
       <button class="btn-bv" onclick="location.href='relatorio.html'">
            Relatório
         </button>
      </div>
   </body>
```



3.5.Página de Cadastro(cadastro.html e cadastro.js)

Página destinada ao registro de usuários, etapa necessária para utilização da calculadora e persistência de dados salvos no banco.

Front-end da página:

```
<body>
      <div class="bem-vindo">
```

```
        <h1>Cadastro de Usuário</h1>
        <form id="cadastro-form">
           <label for="nome">Nome:</label>
               <input type="text" id="nome" name="nome" required /><br
/><br />
           <label for="email">Email:</label>
            <input type="email" id="email" name="email" required /><br
/><br />

           <label for="senha">Senha:</label>
           <input
              type="password"
              id="senha"
              name="senha"
              required
           /><br /><br />

           <button type="submit">Cadastrar</button>
         <button onclick="location.href='inicio.html'">Inicio</button>
    <button onclick="location.href='relatorio.html'">Relatório</button>

        </form>
             <p>Já possui uma conta? <a href="login.html">Faça login
aqui</a>.</p>

      </div>

      <script src="cadastro.js"></script>
      <script src="script.js"></script>
   </body>
```

Back-End da página:

```
document.addEventListener("DOMContentLoaded", () => {
    const cadastroForm = document.getElementById("cadastro-form");
    cadastroForm.addEventListener("submit", (e) => {
        e.preventDefault();
        const nome = document.getElementById("nome").value;
        const email = document.getElementById("email").value;
        const senha = document.getElementById("senha").value;

        fetch("/api/cadastro", {
            method: "POST",
            headers: {
                "Content-Type": "application/json",
            },
            body: JSON.stringify({ nome, email, senha }),
        })
            .then((response) => response.json())
            .then((data) => {
                if (data.success) {
                    window.location.href = "login.html";
                } else {
                    alert(data.message);
                }
            })
            .catch((error) => console.error("Erro no cadastro:", error));
    });
});
```

A função assíncrona acima é acionada quando o botão “cadastrar” é pressionado, primeiramente os dados inseridos nas caixas de nome, email e senha serão armazenados nas suas respectivas variáveis que por sua vez serão transferidos para o banco de dados através da “fetch” com o método HTTP “post”, se os dados chegaram ao seu destino corretamente, o usuário será transferido para a página de login, caso contrário uma mensagem de erro será exibida.

3.6.Página de login(login.html e login.js)

Front-End da página:

```
<body>
      <div class="bem-vindo">
         <h1>Login</h1>
         <form id="login-form">
            <label for="email">Email:</label>
              <input type="email" id="email" name="email" required /><br
/><br />

            <label for="senha">Senha:</label>
            <input
               type="password"
               id="senha"
               name="senha"
               required
            /><br /><br />

            <button type="submit">Entrar</button>
          <button onclick="location.href='inicio.html'">Inicio</button>
    <button onclick="location.href='relatorio.html'">Relatório</button>
         </form>
         <p>
            Ainda não possui uma conta?
            <a href="cadastro.html">Cadastre-se aqui</a>.
         </p>
      </div>
      <script src="login.js"></script>
   </body>
```

Back-End da página:

```
document.addEventListener("DOMContentLoaded", () => {
    const loginForm = document.getElementById("login-form");
    loginForm.addEventListener("submit", (e) => {
        e.preventDefault();
        const email = document.getElementById("email").value;
        const senha = document.getElementById("senha").value;
        fetch("/api/login", {
            method: "POST",
            headers: {
                "Content-Type": "application/json",
            },
            body: JSON.stringify({ email, senha }),
        })
            .then((response) => response.json())
            .then((data) => {
                if (data.success) {
                    window.location.href = "index.html";
                } else { alert(data.message);}
            })
            .catch((error) => console.error("Erro no login:", error));
    });
});
```

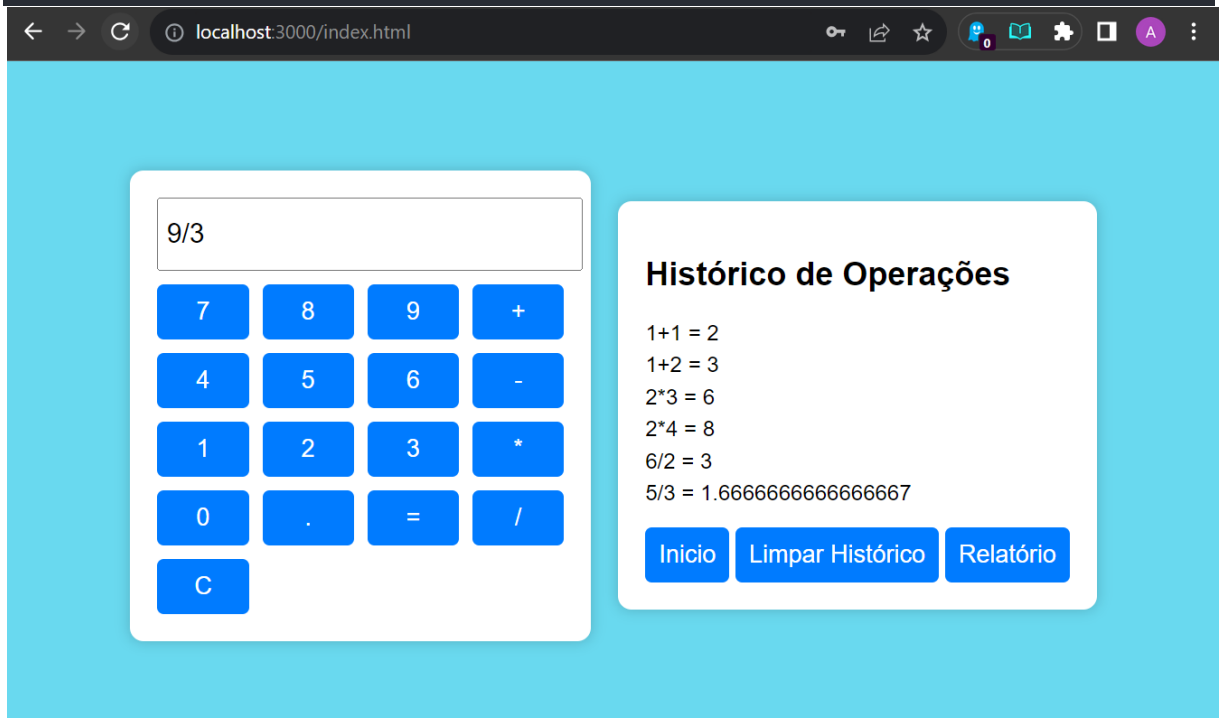
Quando o botão “entrar” é pressionado, os dados inseridos nas caixas de email e senha serão armazenados nas suas respectivas variáveis que por sua vez serão transferidos para o banco de dados através da “fetch” com o método HTTP “post”, se os dados chegaram ao seu destino corretamente e foram compatíveis com dados já armazenados no banco, o processo de login será efetuado corretamente e o usuário será transferido para a página da calculadora(index.html).

3.7.Página da Calculadora (index.html e script.js)

Front-End da página:

```
<body>
      <div class="pag-calculator">
         <div class="calculator">
            <input type="text" id="display" readonly />
            <div class="buttons">
               <button onclick="addToDisplay('7')">7</button>
               <button onclick="addToDisplay('8')">8</button>
               <button onclick="addToDisplay('9')">9</button>
               <button onclick="addToDisplay('+')">+</button>
               <button onclick="addToDisplay('4')">4</button>
               <button onclick="addToDisplay('5')">5</button>
               <button onclick="addToDisplay('6')">6</button>
               <button onclick="addToDisplay('-')">-</button>
               <button onclick="addToDisplay('1')">1</button>
               <button onclick="addToDisplay('2')">2</button>
               <button onclick="addToDisplay('3')">3</button>
               <button onclick="addToDisplay('*')">*</button>
               <button onclick="addToDisplay('0')">0</button>
               <button onclick="addToDisplay('.')">.</button>
               <button onclick="calculate()">=</button>
               <button onclick="addToDisplay('/')">/</button>
               <button onclick="clearDisplay()">C</button>
            </div>
         </div>
         <div class="history">
            <h2>Histórico de Operações</h2>
            <ul id="history-list"></ul>
            <div class="botoes">
          <button onclick="location.href='inicio.html'">Inicio</button>
           <button onclick="deleteOperationsFromServer()">
                  Limpar Histórico
               </button>
```

```
                <button onclick="location.href='relatorio.html'">
                    Relatório
                </button>
            </div>
        </div>
    </div>

    <script src="script.js"></script>
</body>
```



Back-End da página:

- Exibir valores na calculadora:

```
function addToDisplay(value) {
    currentInput += value;
    document.getElementById("display").value = currentInput;
}
```

Substitui a informação presente na caixa de id “display” para o valor obtido pela operação;

- Limpar valores na calculadora

```
function clearDisplay() {
    currentInput = "";
    document.getElementById("display").value = "";
}
```

Substitui a informação presente na caixa de id “display” por uma string vazia;

- Envia operações e seus respectivos resultados para o banco de dados e os armazena através do método HTTP “POST”:

```
function saveOperationToServer(operacao, resultado) {
   fetch("/api/operacoes", {
      method: "POST",
      headers: {
         "Content-Type": "application/json",
      },
      body: JSON.stringify({ operacao, resultado }),
   })
      .then((response) => response.json())
      .then((data) => console.log(data))
      .catch((error) =>
         console.error("Erro ao salvar a operação no servidor:", error)
      );
}
```

- Atualiza o histórico de operações sempre que a função é chamada:

```
function updateHistory() {
   const historyList = document.getElementById("history-list");
   historyList.innerHTML = "";
   history.forEach((item) => {
      const li = document.createElement("li");
      li.textContent = item;
      historyList.appendChild(li);
   });
}
```

A função forEach é utilizada para percorrer a lista de informações temporárias e as insere uma por uma no histórico;

- Limpar histórico:

```
function deleteOperationsFromServer() {
   fetch("/api/operacoes", {
      method: "DELETE",
   })
      .then((response) => response.json())
      .then((data) => {
         console.log(data);
         history = [];
         updateHistory();
      })
      .catch((error) =>
```

```
            console.error("Erro ao excluir as operações do servidor:",
error)
      );
}
```

Substitui as informações do histórico no front-end para um valor nulo e apaga as informações salvas no banco de dados atraves do metodo HTTP “DELETE”;

- Cálculos:

```
function calculate() {
   try {
      const result = eval(currentInput);
      history.push(`${currentInput} = ${result}`);
      saveOperationToServer(currentInput, result);
      currentInput = "";
      document.getElementById("display").value = result;
      updateHistory();
   } catch (error) {
      alert("Erro na expressão!");
      clearDisplay();
   }
}
```

Realiza as operações presentes do display através da função eval() e utiliza history.push() para mandar as novas informações para o histórico, atualiza as informações no banco atraves da funcao saveOperationToServer() e por fim, atualiza o histórico.

- Recebe informações do banco:

```
function fetchOperationsFromServer() {
   fetch("/api/operacoes")
      .then((response) => response.json())
      .then((data) => {
  history = data.map((item) => `${item.operacao} = ${item.resultado}`);
         updateHistory();
      })
      .catch((error) =>
           console.error("Erro ao recuperar as operações do servidor:",
error)
      );
}

fetchOperationsFromServer();
```

Sempre que a aplicação for iniciada, a funcao `fetchOperationsFromServer()` será chamada para receber todas as informações presentes no banco de dados, independentemente do usuário logado.

4.Execução do projeto

Primeiramente instale o node.js pelo site caso não o possua https://nodejs.org/en e o banco mysql https://dev.mysql.com/downloads/mysql/ , agora certifique-se de que as informações presentes no arquivo *db.js* condizem com as de sua máquina.

```
  host: "localhost", // Endereço do servidor MySQL
  user: "root",
  password: "root",
  database: "calculadora", // Nome do banco de dados
```

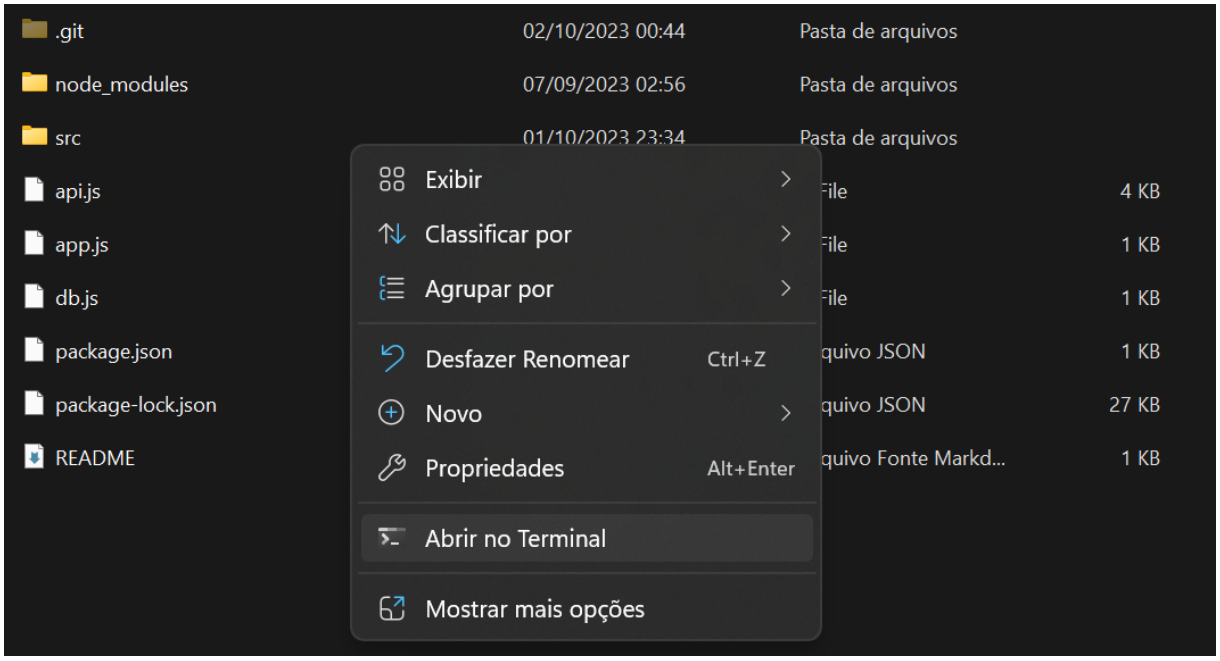
Crie o banco de dados calculadora no seu mysql, e crie as tabelas:

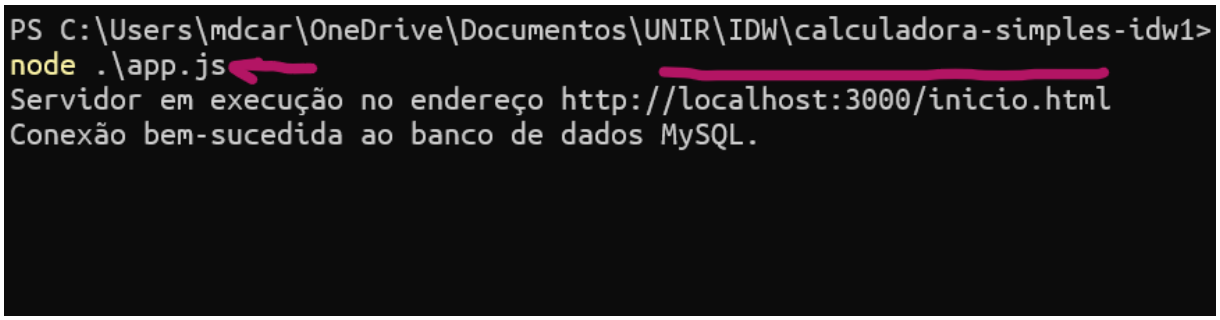
```
CREATE TABLE `usuarios` (
  `id` int NOT NULL AUTO_INCREMENT,
  `nome` varchar(255) NOT NULL,
  `email` varchar(255) NOT NULL,
  `senha` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci
```

e

```
CREATE TABLE `operacoes` (
  `id` int NOT NULL AUTO_INCREMENT,
  `operacao` text,
  `resultado` decimal(10,2) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=42 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci
```

Com o node instalado abra o terminal referente a pasta do projeto e insira o comando ***`node app.js`*** para inicializar a aplicação.





Copie o link obtido em seu navegador ou pressione a tecla ctrl e clique no link.

Pronto!! A aplicação já pode ser utilizada.

5.Considerações finais

A aplicação funciona corretamente no que se propõe (operacoes matematicas basicas, registro de usuários e histórico persistente), no futuro mais opções podem ser adicionadas e métodos de segurança de dados e proteção contra ataques podem ser considerados, no que diz a respeito ao trabalho final da disciplina de introdução ao desenvolvimento web a aplicação termina por aqui.

6. Referências

- https://www.npmjs.com/
- https://www.npmjs.com/package/bcryptjs
- https://expressjs.com/pt-br/
- https://nodejs.org/en
- https://www.mysql.com/
- https://www.oracle.com/br/database/what-is-a-relational-database/