ACESSO A BANCO

Estrutura do Diretório.

```
sistemaDeLoginMiguelGurgel2024.2/
├── css/
│   └── estilo.css
├── js/
│   └── script.js
├── server/
│   ├── conexao.js
│   ├── index.js
│   ├── rotas.js
├── views/
│   └── index.html
├── package.json
├── package-lock.json
└── README.md
```

OBS: CRIE ESSAS PASTAS NO C: DO SEU COMPUTADOR, ENTRE COM O USUÁRIO ADM.

ABRA A PASTA PELO VISUAL STUDIO CODE

ABRA O TERMINAL (CTRL + J) E DIGITE O SEGUINTE COMANDO

npm install express mysql bcrypt

CONFIGURAÇÃO DO BANCO DE DADOS:

ABRA O MySQL Command-Line Client

E DIGITE O SEGUINTE SUA SENHA (123456) OU (ALUNO)

APÓS ENTRAR NO BANCO CONFIGURE O BANCO DE DADOS:

```
CREATE DATABASE sistema_login;
USE sistema_login;

CREATE TABLE usuarios (
    id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) NOT NULL UNIQUE,
    senha VARCHAR(255) NOT NULL
);
```

Comandos de Autenticação MySQL:

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY
'123456';
FLUSH PRIVILEGES;
```

Inicialização do Servidor:

```
npm start
```

CRIAÇÃO DOS ARQUIVOS

1. Arquivo `conexao.js` (local: `server/conexao.js`)

```
// Importa o módulo mysql para lidar com a conexão ao banco de dados MySQL
const mysql = require('mysql');

// Configurações de conexão com o banco de dados MySQL
const connection = mysql.createConnection({
    host: 'localhost',        // O servidor MySQL está rodando localmente
    user: 'root',             // Nome de usuário do MySQL com permissões administrativas
    password: '123456',       // Senha configurada para o usuário root
    database: 'sistema_login' // Nome do banco de dados que criamos para este
projeto
});

// Tenta estabelecer uma conexão com o banco de dados MySQL
connection.connect((err) => {
    if (err) {
        // Caso ocorra um erro, a mensagem de erro será exibida no console
        console.error('Erro ao conectar ao banco de dados:', err);
        throw err; // Lança o erro para interromper a execução do programa
    }
    // Mensagem exibida se a conexão for bem-sucedida
    console.log('Conectado ao banco de dados MySQL!');
});

// Exporta a conexão para ser utilizada em outros arquivos do projeto
module.exports = connection;
```

2. Arquivo `rotas.js` (local: `server/rotas.js`)

```
// Importa os módulos necessários para lidar com as rotas
const express = require('express');  // Framework web para criação de rotas HTTP
const router = express.Router();     // Criador de rotas do Express
const bcrypt = require('bcrypt');    // Módulo para criptografar senhas
const connection = require('./conexao'); // Importa a configuração de conexão com o banco de dados

// Rota para cadastro de novos usuários
router.post('/cadastro', (req, res) => {
    // Extrai email e senha do corpo da requisição enviada pelo cliente
    const { email, senha } = req.body;

    // Criptografa a senha utilizando bcrypt com fator de custo 10
    bcrypt.hash(senha, 10, (err, hash) => {
        if (err) {
            // Retorna uma mensagem de erro em caso de falha na criptografia
            return res.json({ success: false, message: 'Erro ao criptografar a senha.' });
        }

        // Insere o novo usuário no banco de dados com a senha criptografada
        const sql = 'INSERT INTO usuarios (email, senha) VALUES (?, ?)';
        connection.query(sql, [email, hash], (err) => {
            if (err) {
                // Retorna uma mensagem de erro caso haja falha na inserção
                return res.json({ success: false, message: 'Erro ao cadastrar usuário.' });
            }
            // Retorna uma mensagem de sucesso se o usuário for cadastrado
            return res.json({ success: true });
        });
    });
});
```

```
// Rota para autenticação de usuários (login)
router.post('/login', (req, res) => {
    // Extrai email e senha do corpo da requisição enviada pelo cliente
    const { email, senha } = req.body;

    // Consulta SQL para buscar o usuário pelo email fornecido
    const sql = 'SELECT senha FROM usuarios WHERE email = ?';
    connection.query(sql, [email], (err, results) => {
        if (err) {
            // Retorna uma mensagem de erro caso a consulta falhe
            return res.json({ success: false, message: 'Erro na consulta.' });
        }

        // Verifica se um usuário foi encontrado
        if (results.length > 0) {
            // Compara a senha fornecida com o hash armazenado no banco de dados
            bcrypt.compare(senha, results[0].senha, (err, result) => {
                if (result) {
                    // Responde com sucesso se as senhas coincidirem
                    return res.json({ success: true });
                } else {
                    // Informa ao cliente que a senha está incorreta
                    return res.json({ success: false, message: 'Senha incorreta.' });
                }
            });
        } else {
            // Informa ao cliente que o usuário não foi encontrado
            return res.json({ success: false, message: 'Usuário não encontrado.' });
        }
    });
});

// Exporta as rotas para serem utilizadas no servidor principal
module.exports = router;
```

3. Arquivo `index.js` (local: `server/index.js`)

```
// Importa os módulos necessários
const express = require('express');      // Framework web para lidar com requisições HTTP
const app = express();                   // Inicializa o aplicativo Express
const rotas = require('./rotas');        // Importa as rotas definidas no arquivo rotas.js
const path = require('path');            // Módulo nativo para manipulação de caminhos de arquivos

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

// Configura as pastas de arquivos estáticos (HTML, CSS, JS)
app.use(express.static(path.join(__dirname, '..', 'views')));
app.use('/css', express.static(path.join(__dirname, '..', 'css')));
app.use('/js', express.static(path.join(__dirname, '..', 'js')));

// Usa as rotas importadas para lidar com as requisições de login e cadastro
app.use(rotas);

// Define a porta onde o servidor vai rodar
const PORT = 3000;
app.listen(PORT, () => {
    // Exibe uma mensagem no console confirmando que o servidor está rodando
    console.log(`Servidor rodando na porta ${PORT}`);
});
```

4. Arquivo `index.html` (local: `views/index.html`)

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <title>Sistema de Login</title>
    <link rel="stylesheet" href="css/estilo.css">
</head>
<body>
    <div class="container">
        <!-- Seção de Login -->
        <div id="loginSection">
            <h2>Login</h2>
            <form id="loginForm">
                <label for="loginEmail">Email:</label>
                <input type="email" id="loginEmail" name="email" required>
                <label for="loginSenha">Senha:</label>
                <input type="password" id="loginSenha" name="senha" required>
                <button type="submit">Entrar</button>
            </form>
            <p class="switchForm" id="showCadastro">Cadastrar</p>
        </div>

        <!-- Seção de Cadastro -->
        <div id="cadastroSection" style="display: none;">
            <h2>Cadastro</h2>
            <form id="cadastroForm">
                <label for="cadastroEmail">Email:</label>
                <input type="email" id="cadastroEmail" name="email" required>
                <label for="cadastroSenha">Senha:</label>
                <input type="password" id="cadastroSenha" name="senha" required>
                <button type="submit">Cadastrar</button>
```

```
            </form>
            <p class="switchForm" id="showLogin">Voltar para Login</p>
        </div>
    </div>

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

5. Arquivo `script.js` (local: `js/script.js`)

```
// Alterna entre as seções de login e cadastro
document.getElementById('showCadastro').addEventListener('click', function() {
    // Oculta a seção de login e exibe a seção de cadastro
    document.getElementById('loginSection').style.display = 'none';
    document.getElementById('cadastroSection').style.display = 'block';
});

document.getElementById('showLogin').addEventListener('click', function() {
    // Exibe a seção de login e oculta a seção de cadastro
    document.getElementById('loginSection').style.display = 'block';
    document.getElementById('cadastroSection').style.display = 'none';
});

// Manipula o envio do formulário de login
document.getElementById('loginForm').addEventListener('submit', function(e) {
    e.preventDefault(); // Previne o comportamento padrão de envio do formulário

    // Obtém os dados do formulário de login
    const email = document.getElementById('loginEmail').value;
    const senha = document.getElementById('loginSenha').value;

    // Envia uma requisição POST para a rota de login
    fetch('/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ email, senha })
    })
    .then(response => response.json())
    .then(data => {
        if (data.success) {
            // Exibe uma mensagem de sucesso
```

```
            alert('Login realizado com sucesso!');
        } else {
            // Exibe uma mensagem de erro se o login falhar
            alert('Login falhou: ' + data.message);
        }
    });
});

// Manipula o envio do formulário de cadastro
document.getElementById('cadastroForm').addEventListener('submit', function(e) {
    e.preventDefault(); // Previne o comportamento padrão de envio do formulário

    // Obtém os dados do formulário de cadastro
    const email = document.getElementById('cadastroEmail').value;
    const senha = document.getElementById('cadastroSenha').value;

    // Envia uma requisição POST para a rota de cadastro
    fetch('/cadastro', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ email, senha })
    })
    .then(response => response.json())
    .then(data => {
        if (data.success) {
            // Exibe uma mensagem de sucesso e alterna para a tela de login
            alert('Cadastro realizado com sucesso!');
            document.getElementById('showLogin').click();
        } else {
            // Exibe uma mensagem de erro se o cadastro falhar
            alert('Cadastro falhou: ' + data.message);
        }
    });
});
```

6. Comandos do Terminal (`comandos_terminal.txt`)

```
# Inicialize um novo projeto Node.js (no diretório raiz do projeto)
npm init -y

# Instale as dependências necessárias para o projeto
npm install express mysql bcrypt

# Crie o banco de dados MySQL e configure a tabela de usuários
# Execute esses comandos no terminal do MySQL
mysql -u root -p
CREATE DATABASE sistema_login;
USE sistema_login;
CREATE TABLE usuarios (
    id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) NOT NULL UNIQUE,
    senha VARCHAR(255) NOT NULL
);

# Se necessário, reconfigure o método de autenticação do MySQL
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY
'123456';
FLUSH PRIVILEGES;

# Inicie o servidor Node.js após configurar o banco de dados
node server/index.js

# Acesse o navegador no endereço http://localhost:3000 para visualizar a aplicação
```

7. Arquivo de Configuração do Banco de Dados (`configuracao_banco.sql`)

```
-- Criação do banco de dados
CREATE DATABASE sistema_login;

-- Seleção do banco de dados para uso
USE sistema_login;

-- Criação da tabela de usuários
CREATE TABLE usuarios (
    id INT AUTO_INCREMENT PRIMARY KEY, -- Identificador único para cada usuário
    email VARCHAR(255) NOT NULL UNIQUE, -- Armazena o email do usuário (valor único)
    senha VARCHAR(255) NOT NULL -- Armazena o hash da senha do usuário
);

-- Reconfiguração do método de autenticação do MySQL, se necessário
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY '123456';

-- Atualiza as permissões no banco de dados após as alterações
FLUSH PRIVILEGES;
```