Documentação do Backend

Configuração Inicial

Este backend é um servidor Node.js usando o framework Express para gerenciar as rotas HTTP e SQLite como banco de dados para armazenamento de dados de usuários.

Dependências

1. **Express:** Framework para criar o servidor e gerenciar rotas.
2. **Body-parser:** Middleware para analisar o corpo das requisições HTTP.
3. **SQLite3:** Biblioteca para gerenciar o banco de dados SQLite.

Instalação

Certifique-se de ter o Node.js instalado. Em seguida, instale as dependências com o comando:

npm install express body-parser sqlite3

Código

```
const express = require("express");

const app = express();

var bodyParser = require("body-parser");

var port = process.env.PORT || 3000;

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: true }));



/**

 * Configuração do banco de dados SQLite.

 */

var sqlite3 = require("sqlite3").verbose();

var DBPATH = "appBD.db";

var db = new sqlite3.Database(DBPATH);
```

```
/**
 * Rota GET para retornar todos os registros da tabela 'users'.
 */
app.get("/tudo", function (req, res) {
  db.all(`SELECT * FROM users`, [], (err, rows) => {
    if (err) {
      res.send(err);
    }
    res.send(rows);
  });
});


/**
 * Criação da tabela 'users', se ainda não existir.
 */
db.serialize(function () {
  db.run(
    `CREATE TABLE IF NOT EXISTS users (
          id INTEGER PRIMARY KEY AUTOINCREMENT,
          email TEXT UNIQUE,
          senha TEXT,
          finalbom INTEGER DEFAULT 0,
          finalmedio INTEGER DEFAULT 0,
          finalruim INTEGER DEFAULT 0
        )`,
```

```
    function (err) {
      if (err) {
        console.error("Erro ao criar tabela:", err.message);
      }
    }
  );
});

/**
 * Rota GET para verificar se o servidor está funcionando.
 */
app.get("/", function (req, res) {
  res.send("Servidor funcionando!");
});

/**
 * Rota POST para obter o nome do usuário com base no email.
 */
app.post("/obterNomeUsuario", function (req, res) {
  var email = req.body.email;
  db.get(
    "SELECT nome FROM users WHERE email = ?",
    [email],
    function (err, row) {
      if (err) {
```

```
        res.status(500).send("Erro ao obter o nome do usuário");
      } else {
        if (row) {
          res.status(200).send(row.nome);
        } else {
          res.status(404).send("Usuário não encontrado");
        }
      }
    }
  );
});

/**
 * Rota POST para obter a pontuação do usuário com base no email.
 */
app.post("/obterPontuacaoUsuario", function (req, res) {
  var email = req.body.email;
  db.get(
    "SELECT pontuacao FROM users WHERE email = ?",
    [email],
    function (err, row) {
      if (err) {
        res.status(500).send("Erro ao obter a pontuação do usuário");
      } else {
        if (row) {
```

```
          res.status(200).send(row.pontuacao.toString());
        } else {
          res.status(404).send("Usuário não encontrado");
        }
      }
    }
  );
});

/**
 * Rota POST para cadastrar um novo usuário.
 */
app.post("/cadastrarUsuario", function (req, res) {
  var email = req.body.email;
  var senha = req.body.senha;

  db.get("SELECT * FROM users WHERE email = ?", [email], function (err, row) {
    if (err) {
      res.status(500).send("Erro ao verificar email");
    } else {
      if (row) {
        res.status(400).send("Este email já está registrado");
      } else {
        var sql = "INSERT INTO users (email, senha) VALUES (?, ?)";
        db.run(sql, [email, senha], function (err) {
```

```
          if (err) {
            res.status(500).send("Erro ao criar usuário");
          } else {
            res.status(201).send("Usuário registrado com sucesso");
          }
        });
      }
    }
  });
});

/**
 * Rota POST para autenticar um usuário.
 */
app.post("/login", function (req, res) {
  var email = req.body.email;
  var senha = req.body.senha;

  db.get(
    "SELECT * FROM users WHERE email = ? AND senha = ?",
    [email, senha],
    function (err, row) {
      if (err) {
        res.status(500).send("Erro ao autenticar usuário");
      } else {
```

```
        if (row) {
          res.status(200).send("Login bem-sucedido");
        } else {
          res.status(401).send("Usuário ou senha incorretos");
        }
      }
    }
  );
});

/**
 * Rota POST para registrar a pontuação do usuário.
 */
app.post("/enviarFinalBom", (req, res) => {
  const { email, senha, finalbom } = req.body;

  db.get(
    "SELECT * FROM users WHERE email = ? AND senha = ?",
    [email, senha],
    function (err, row) {
      if (err) {
        console.error("Erro ao verificar usuário:", err.message);
        res.status(500).send("Erro ao verificar usuário");
      } else {
        if (row) {
```

```
        db.run(
          "UPDATE users SET finalbom = finalbom + ? WHERE email = ? AND senha =
?",
          [finalbom, email, senha],
          function (err) {
            if (err) {
              console.error(
                "Erro ao atualizar a pontuação do usuário:",
                err.message
              );
              res.status(500).send("Erro ao atualizar a pontuação do usuário");
            } else {
              console.log(
                `Pontuação do usuário (${email}) atualizada para ${
                  row.finalbom + finalbom
                }`
              );
              res
                .status(200)
                .send("Pontuação do usuário atualizada com sucesso");
            }
          }
        );
      } else {
        res.status(404).send("Usuário não encontrado");
      }
```

```
    }
  }
 );
});

module.exports = app;

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

Rotas

1. Verificar o Servidor

- **Rota:** GET /
- **Descrição:** Verifica se o servidor está funcionando.
- **Resposta:** Servidor funcionando!

2. Listar Todos os Usuários

- **Rota:** GET /tudo
- **Descrição:** Retorna todos os registros da tabela users.
- **Resposta:** JSON com a lista de usuários.

3. Obter Nome do Usuário

- **Rota:** POST /obterNomeUsuario
- **Parâmetros:**
  - email: Email do usuário.
- **Descrição:** Retorna o nome do usuário baseado no email fornecido.
- **Resposta:**
  - **200:** Nome do usuário.
  - **404:** Usuário não encontrado.
  - **500:** Erro ao obter o nome do usuário.

4. Obter Pontuação do Usuário

- **Rota:** POST /obterPontuacaoUsuario
- **Parâmetros:**
  - email: Email do usuário.
- **Descrição:** Retorna a pontuação do usuário baseado no email fornecido.
- **Resposta:**
  - **200:** Pontuação do usuário.
  - **404:** Usuário não encontrado.
  - **500:** Erro ao obter a pontuação do usuário.

5. Cadastrar Usuário

- **Rota:** POST /cadastrarUsuario
- **Parâmetros:**
  - email: Email do usuário.
  - senha: Senha do usuário.
- **Descrição:** Cadastra um novo usuário.
- **Resposta:**
  - **201:** Usuário registrado com sucesso.
  - **400:** Este email já está registrado.
  - **500:** Erro ao criar usuário.

6. Autenticar Usuário

- **Rota:** POST /login
- **Parâmetros:**
  - email: Email do usuário.
  - senha: Senha do usuário.
- **Descrição:** Autentica um usuário.
- **Resposta:**
  - **200:** Login bem-sucedido.
  - **401:** Usuário ou senha incorretos.
  - **500:** Erro ao autenticar usuário.

7. Enviar Pontuação

- **Rota:** POST /enviarFinalBom
- **Parâmetros:**
  - email: Email do usuário.
  - senha: Senha do usuário.
  - finalbom: Pontuação a ser adicionada.
- **Descrição:** Atualiza a pontuação do usuário.
- **Resposta:**
  - **200:** Pontuação do usuário atualizada com sucesso.
  - **404:** Usuário não encontrado.
  - **500:** Erro ao atualizar a pontuação do usuário.

Configuração do Banco de Dados

A configuração do banco de dados SQLite é feita ao inicializar o servidor. Se a tabela users ainda não existir, ela será criada com a estrutura apropriada para armazenar informações do usuário.

Iniciar o Servidor

Para iniciar o servidor, execute o seguinte comando:

node <nome_do_arquivo>

Substitua <nome_do_arquivo> pelo nome do arquivo que contém o código do servidor.

Observações

- Certifique-se de ter o banco de dados appBD.db no mesmo diretório do código ou ajuste o caminho do banco de dados conforme necessário.
- As senhas não são criptografadas/descriptografadas neste exemplo. Para um ambiente de produção, recomenda-se fortemente implementar a criptografia de senhas usando bibliotecas como bcrypt.