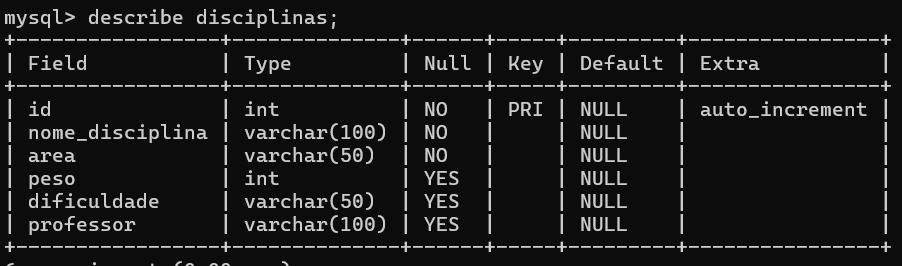
Documentação técnica para projeto BACK END

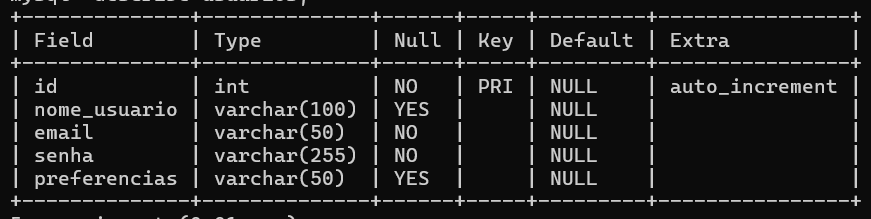
Uma aplicação voltada para o cenário dos concursos, em que o usuário pode logar na aplicação, obtendo autorização para adicionar, edital, visualizar, e deletar disciplinas que está estudando no momento.

Neste sistema, temos as seguintes tabelas:

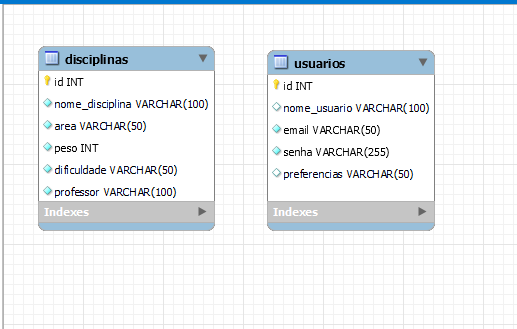
* Disciplinas: esta tabela é responsável por registrar todas as disciplinas que o usuário está cadastrado, tendo detalhes sobre as mesmas como dificuldade que o usuário sente perante esta e o peso que ela carrega para o mesmo. Assim, a tabela tem o seguinte formato:



* Usuários: Tabela responsável por registrar os usuários cadastrados no sistema, com colunas como nome, email, senha e preferencias (qual área deseja seguir). Desta forma, a tabela tem o formato:



Desta forma, a modelagem do banco de dados será:



Rotas API

1. LerDisciplinas:

Esta rota é responsável por retornar para o usuário todas as matérias registradas no banco de dados, identificando cada um por seu ID, Nome e Professor responsável por ministrá-la.

Na pasta controllers/controller.js:

const LerDisciplinas = (req, res) => {

db.query('SELECT\* FROM disciplinas', (err, results) => {

if(err) {

console.error('ERRO! rota LerDisciplinas', err)

res.status(500).send('erro');

return

}

res.json(results);

});

};

Diante disso, podemos testar aplicação, verificando quais dados já estão cadastrados. Executando a rota no insônia ou ThunderClient teremos:

Método: GET

url: http://localhost:5000/api/router

request: --

response: [

{

"id": 1,

"nome\_disciplina": "Português",

"area": "Linguagens",

"peso": 25,

"dificuldade": "Médio",

"professor": "Maria Silva"

},

{

"id": 3,

"nome\_disciplina": "Direito Administrativo",

"area": "Direito",

"peso": 2,

"dificuldade": "Média",

"professor": "Ana Costa"

},

{

"id": 4,

"nome\_disciplina": "Direito Administrativo",

"area": "Direito",

"peso": 2,

"dificuldade": "Média",

"professor": "Ana Costa"

},

{

"id": 5,

"nome\_disciplina": "Direito Civil",

"area": "Direito",

"peso": 2,

"dificuldade": "Média",

"professor": "Ana Costa"

}

]

1. AdicionarDisciplina:

Esta rota é responsável por adicionar disciplinas, permitindo que o usuário insira a informações a respeito desta matéria.

Na pasta controllers/controllers.js teríamos o seguinte código.

const AdicionarDisciplina = (req, res) => {

    const {nome\_disciplina, area, peso, dificuldade, professor} = req.body;

    db.query(

        'SELECT \* FROM disciplinas WHERE nome\_disciplina=? AND area=? AND peso=? AND dificuldade=? AND professor=?',

        [nome\_disciplina, area, peso, dificuldade, professor],

        (err, results) => {

            if(err) {

                console.error('ERRO. Não foi possível adicionar a disciplina', err);

                res.status(500).send('Erro!');

                return

            }

            if(results.length > 0) {

                res.status(400).send('DISCIPLINA JÁ EXISTENTE');

            }

            db.query(

                'INSERT INTO disciplinas(nome\_disciplina, area, peso, dificuldade, professor) VALUES(?,?,?,?,?)',

                [nome\_disciplina, area, peso, dificuldade, professor],

                (err, results) => {

                    if(err) {

                        console.error('NÃO FOI POSSÍVEL ADICIONAR DISCIPLINA', err);

                        res.status(500).send('ERROOO');

                    return

                    }

                    res.status(200).send('Disciplina adicionada');

                }

            )

        }

    )

};

Inicialmente, a rota de adicionar disciplina verifica uma possível existência da disciplina que deseja ser adicionada, retornando ao usuário a duplicidade de disciplinas no banco de dados. Desta forma, evita possíveis ambiguidades dentro da aplicação, no qual facilitaria a para o usuário gerenciar as matérias que está matriculado. Por fim, podemos realizar um teste para verificar se a rota está funcionando:

Método: POST

url: http://localhost:5000/api/router

request: {

"nome\_disciplina": "Português",

"area": "Linguagens",

"peso": 20,

"dificuldade": "Média",

"professor": "João Silva"

}

response: Disciplina adicionada

1. AtualizarDisciplina:

Esta rota é responsável por atualizar dados na tabela disciplinas, permitindo ao usuário editar o registro no banco de dados. O código desta rota, dentro da pasta controllers/controllers.js:

const AtualizarDisciplina = (req, res) => {

const { id } = req.params;

const { nome\_disciplina, area, peso, dificuldade, professor } = req.body;

db.query(

'UPDATE disciplinas SET nome\_disciplina = ?, area = ?, peso = ?, dificuldade = ?, professor = ? WHERE id = ?',

[nome\_disciplina, area, peso, dificuldade, professor, id],

(err,results) => {

if(err) {

console.error('CRASH. Não foi possível atualizar', err);

res.status(500).send('ERRO!');

return

}

if(results.affectedRows === 0) {

res.status(400).send('Disciplina não encontrada');

return

}

res.send('Atualização completa');

}

);

};

Inicialmente temos um corpo da requisição, convertendo para Json. Posteriormente, é realizada uma busca no banco de dados procurando pelo id, em que a atualização só ocorre se a disciplina estiver realmente cadastrada no banco de dados. Por fim, podemos testar a rota:

Método: PUT,

url: http://localhost:5000/api/router/3

request: {

"nome\_disciplina": "Matemática avançada",

"area": "Fiscal",

"peso": 5,

"dificuldade": "difícil",

"professor": "Ismael"

}

response: Atualização Completa

1. AtualizarDisciplinaParcial:

Tal rota é responsável por permitir ao usuário editar uma informação específica dentro do banco de dados, divergindo do que acontece na requisição anterior, no qual todas as informações podem ser alteradas. Na rota “patch”, apenas um elemento, fator (ou coluna da tabela) vai ser alterado, sem modificar os demais elementos. Desta forma, a rota AtualizarDisciplinaParcial no arquivo controller.js:

const AtualizarDisciplinaParcial = (req, res) => {

    const { id } = req.params;

    const fields = req.body;

    const query = [];

    const values = [];

    for (const [key, value] of Object.entries(fields)) {

        query.push(`${key} = ?`);

        values.push(value);

    }

    values.push(id);

    db.query(

        `UPDATE disciplinas SET ${query.join(',')} WHERE id=?`, values,

        (err,results) => {

            if(err) {

                console.error('ERRO!Atualização parcial falhou', err)

                res.status(400).send('ERROOO')

                return

            }

            if(results.affectedRows === 0) {

                res.status(404).send('Disciplina não encontrada')

                return;

            }

            res.status(200).send("Atualização bem-sucedida")

        }

    );

};

Podemos realizar o teste:

Método: PATCH,

url: <http://localhost:5000/api/router/2>

request: {  
 “nome\_disciplina”: “Informática”

}

Response: Atualização bem-sucedida

1. DeletarDisciplina:

A rota permite ao usuário deletar informações registradas no banco de dados. Na pasta controller/controllers.js teremos:

const DeletarDisciplina = (req, res) => {

    const {id} = req.params;

    db.query(

        'DELETE FROM disciplinas WHERE id=?',

        [id],

        (err, results) => {

            if(err) {

                console.error('ERRO. Não foi possível deletar a disciplina', err);

                res.status(500).send('ERRO')

                return

            }

            res.send('DISCIPLINA DELETADA');

        }

    );

};

A rota de Deletar é responsável por deletar alguma disciplina no banco de dados. Por fim, podemos testar a rota:

Método: DELETE,

URL: http://localhost:5000/api/router/5

request: --

response: Disciplina deletada

1. LoginUsuario:

Rota para o usuário entrar na aplicação (caso já tenha sido registrado anteriormente), em que temos um processo de autenticação, permitindo que apenas usuários autorizados podendo entrar e editar informações no banco de dados. Assim, em controllers/authController.js:

const LogarUsuario = async (req, res) => {

    const { email, senha } = req.body;

    try {

        const [user] = await db.promise().query(

            'SELECT \* FROM usuarios WHERE email=?', [email]);

        if(user.length === 0) {

            res.status(400).send('CREDENCIAIS INVÁLIDAS. Email e senha incorretos');

        return

        }

        const isMatch = await bcrypt.compare(senha, user[0].senha);

        if(!isMatch) {

            res.status(400).send('ERRO. Senha incorreta');

        return

        }

        const token = jwt.sign({userId: user[0].id}, process.env.JWT\_SECRET, {expiresIn: '1h'});

        res.json({token});

    } catch(err) {

        console.error('ERRO AO AUTENTICAR USUÁRIO USUÁRIO', err);

        res.status(500).send('ERRO! Não possível logar o usuário');

    }

};

A rota LoginUsuario é aquela que permite o usuário logar na aplicação, em que após a verificação das credenciais, a aplicação retorna um token de autenticação que autoriza o usuário realizar ações dentro da aplicação por tempo limitado. Por fim, temos um teste:

Método: POST,

url: <http://localhost:5000/api/auth/login>

request: {

“email”: joao@email.com,

“senha”: “senha123”

Response:

{

“token”: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjMsImlhdCI6MTcyODQ3ODEzOSwiZXhwIjoxNzI4NDgxNzM5fQ.gICJqxoHKp1S95VtMaKgkovaPlEDjo6Uw7mntz2WIPU”

}

Após o retorno do token, o usuário com o token ativo pode realizar alterações no banco de dados, permitindo editar, criar, atualizar e deletar disciplinas.

1. RegistroUsuario:

Rota responsável por permitir que o usuário se cadastre no banco de dados e tenha suas credenciais registradas no banco. Assim, em controllers/authController.js:

const RegistrarUsuario = async (req, res) => {

    const {nome\_usuario, email, senha, preferencias} = req.body;

    try {

        const [existingUser] = await db.promise().query(

            'SELECT \* FROM usuarios WHERE email=?', [email]);

            if(existingUser.length > 0) {

                res.status(400).send('USUÁRIO JÁ CADASTRADO');

            return

            };

            const hashedSenha = await bcrypt.hash(senha, 10);

            await db.promise().query(

                'INSERT INTO usuarios(nome\_usuario, email, senha, preferencias) VALUES (?,?,?,?)',

                [nome\_usuario, email, hashedSenha, preferencias]);

                res.status(201).send('USUÁRIO REGISTRADO');

    }

    catch (err) {

        console.error('ERRO. Não foi possível registrar o usuário', err);

        res.status(500).send('Erro ao registrar usuário');

    }

};

1. LerUsuarios:

Esta rota é responsável por retornar todos os usuários que tem cadastrado ativo no banco de dados. Assim, em controllers/controller.js:

const LerUsuarios = (req, res) => {

    db.query(

        'SELECT\* FROM usuarios',

        (err, results) => {

            if(err) {

                console.error('ERRO! Não foi ppssível visualizar os usuários', err);

                res.status(500).send('ERRO');

            return

            }

            res.json(results);

        }

    );

}

Integração de back-end com front-end