PROJETO CRUD(CREATE, UPDATE e DELETE)

Vimos como podemos criar uma conta e fazer login. Agora vamos criar um projeto simples de como a gente coloca link no site, tipo linktree.
Primeiro temos que criar um model para esse crud. Da mesma forma, iremos criar funções para criar, limpar e validar.

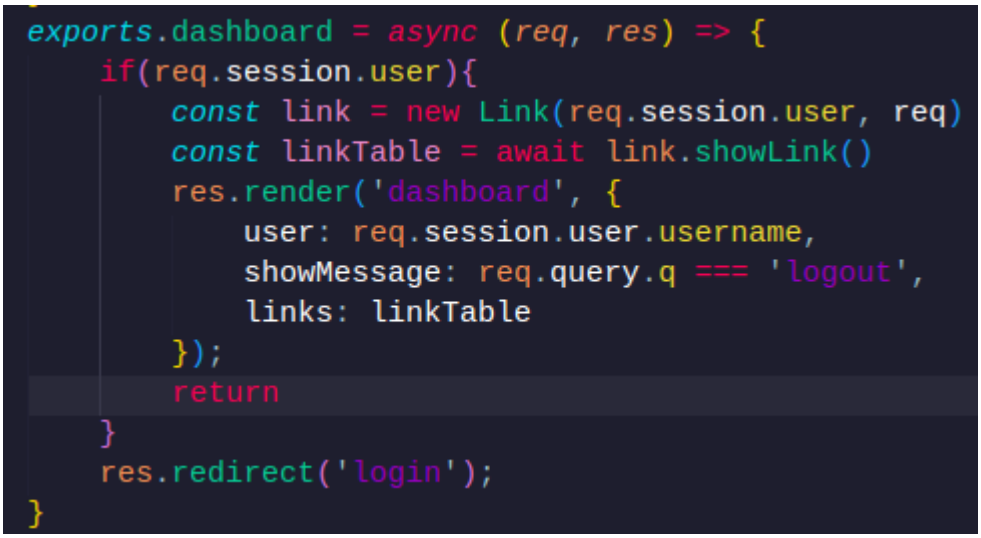
models > contatoModels.js > linkAdd > update

```
const mongo = require('mongoose');
const validador = require('validator');

const schema = new mongo.Schema({
    titulo: {type: String, required: true},
    link: {type: String, required: true},
    userId: {type: mongo.Schema.Types.ObjectId, ref: 'login', required: true}
});

const linkList = mongo.model('link', schema);

class linkAdd{
    constructor(body, req){
        this.body = body,
        this.req = req;
        this.link = {};
        this.error = [];
    }
    async add(){
        this.valida();
        if(this.error.length > 0) return;
        this.link = await linkList.create(this.link);

    }
    async showLink(){
        const link = await linkList.find({userId: this.req.session.user._id}).popu
        return link;
    }
    valida(){
        this.clean();
        if(!validador.isURL(this.body.link.link)) this.error.push('Link invalido')
    }
    clean(){
        for(const key in this.body.link){
            if(typeof this.body.link[key] !== 'string'){
                this.body.link[key] = '';
            }
        }
        this.link = {
            titulo: this.body.link.titulo,
            link: this.body.link.link,
            userId: this.req.session.user._id
        };
```

Aqui nos deparamos com algo novo na coluna **userId**, ela vai ser para que nós possamos referenciar o link com aquela conta/pessoa. Usamos o ‘ref’ para referenciar o model login e **mongo.Schema.types.ObjectId** serve para que ele seja preenchido com o id e precisa ser requirido

Em controls, a gente não salva nada em sessão

Como fizemos para aparecer os links ? Atribuimos mais uma variavel local para o .EJS



Na função showLink(), iremos usar .find({userID: req.body.link._id}) # id do usuário.

```
const link = await linkList.find({userId: this.req.session.user._id}).populate('userId');
```

Toda vez que atribuimos um link novo, é salvo o id do usuário, o .populate() é como um “JOIN” no SQL, ele junta duas tabelas. Então, tendo as duas no return, o .EJS vai fazer o resto

```
<div>
    <h2>Meus links</h2>
    <nav class="nav-link">
        <ul class="ul-link">
            <%if(links && links.length > 0){ %>
                <% links.forEach(l => { %>
                    <li data-id="<%= l._id %>">
                        <div class="link-show-list">
                            <div>
                                <h3><%= l.titulo %></h3>
                            </div>
                            <div>
                                <p><%= l.link %></p>
                            </div>
                        </div>
                        <button type="submit" class='btnEdit' data-id="<%= l._id %>"><img src="./assets/image/icon.png"></button>
                    </li>
                <% }) %>
            <% }%>
        </ul>
    </nav>
    <button class="btn-add-link">Adicionar</button>
```

Como ele retorna um objeto, então atribuimos acessando a propriedade .

! DICA: em html, Os atributos globais **data-*** formam uma classe de atributos conhecida como **custom data attributes**, a qual permite que informações proprietárias sejam trocadas via script entre o HTML e sua representação DOM . Por isso que usaremos para a logica do CRUD

Lembrando que para criar, atualizar e deletar estaremos criando rotas

```
router.post('/added', dashboardPage.toadd);
router.post('/edit', dashboardPage.edit);
router.post('/delete', dashboardPage.delete);
```

agora não tem segredo na parte de atualizar e deletar

```
async update(id, titulo, link){
    if(typeof id !== 'string') return;
    const edit = await linkList.findByIdAndUpdate(id, {titulo: titulo, link: link}, {new: true});

}
async delete(id){
    if(typeof id !== 'string') return;
    const deleteLink = await linkList.findByIdAndDelete(id);
}
```

usamos o “new: true” para retornar com o valor atualizado

e no delete é só isso e pronto

Na parte do front-end

```
document.querySelectorAll('.btnEdit').forEach(button => {
    button.addEventListener('click', e => {
        const btnEdit = e.currentTarget;
        btnEdit.style.display = 'none';

        const linkId = btnEdit.getAttribute('data-id');
        const listItem = document.querySelector(`li[data-id="${linkId}"]`);
        const linkShowList = listItem.querySelector('.link-show-list');

        const titulo = linkShowList.querySelector('h3').innerText;
        const link = linkShowList.querySelector('p').innerText;

        const token = document.querySelector('meta[name="token"]').getAttribute('value');

        linkShowList.innerHTML = `
            <form class="form-edit-delete" method="POST">
                <input type="hidden" name="_csrf" value="${token}">
                <input type="hidden" name="link[id]" value="${linkId}">
                <input value="${titulo}" type="text" name="link[titulo]">
                <input value="${link}" type="text" name="link[link]">
                <button name="edit" class='salvar' type="submit">Salvar</button>
                <button name="delete" class='delete' type="submit">Excluir</button>
                <button class="cancelar">Cancelar</button>
            </form>
        `;
```

Como vai existir diversos botões com a mesma classe, iremos usar o botão de editar só com aquele id do data-id que também tá atribuido no li

Na continuação do código, usamos essa lógica:

```
        const editar = listItem.querySelector('.salvar');
        editar.addEventListener('click', e => {
            const formMod = listItem.querySelector('.form-edit-delete');
            formMod.setAttribute('action', '/edit');

        });
        const del = listItem.querySelector('.delete');
        del.addEventListener('click', e => {
            const formMod = listItem.querySelector('.form-edit-delete');
            formMod.setAttribute('action', '/delete')
        });
        const btnCancelar = listItem.querySelector('.cancelar');
        btnCancelar.addEventListener('click', e => {
            e.preventDefault();
            btnEdit.style.display = 'block';
            linkShowList.innerHTML = `
                <div>
                    <h3>${titulo}</h3>
                </div>
                <div>
                    <p>${link}</p>
                </div>
            `;
        });
    });
});
```