

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 > js contatos/models.js > linkAdd > update
1  const mongo = require('mongoose');
2  const validador = require('validator');
3
4  const schema = new mongo.Schema({
5    titulo: {type: String, required: true},
6    link: {type: String, required: true},
7    userId: {type: mongo.Schema.Types.ObjectId, ref: 'login', required: true}
8  });
9
10 const linkList = mongo.model('link', schema);
11
12 class linkAdd{
13   constructor(body, req){
14     this.body = body,
15     this.req = req;
16     this.link = {};
17     this.error = [];
18   }
19   async add(){
20     this.valida();
21     if(this.error.length > 0) return;
22     this.link = await linkList.create(this.link);
23   }
24
25   async showLink(){
26     const link = await linkList.find({userId: this.req.session.user._id}).popu
27     return link;
28   }
29   valida(){
30     this.clean();
31     if(!validador.isURL(this.body.link.link)) this.error.push('Link invalido')
32   }
33   clean(){
34     for(const key in this.body.link){
35       if(typeof this.body.link[key] !== 'string'){
36         this.body.link[key] = '';
37       }
38     }
39     this.link = {
40       titulo: this.body.link.titulo,
41       link: this.body.link.link,
42       userId: this.req.session.user._id
43     };
44   }
45 }
```

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 requerido

Em controls, a gente não salva nada em sessão

```
controls > dashboard.js > delete > delete
1  const link = require('../models/contatoModels');
2  const { csrfToken } = require("../middlewares/middleware");
3
4  exports.toadd = async (req, res) => {
5    const add = new link(req.body, req);
6    await add.add();
7    if(add.error.length > 0) return;
8    req.session.save(async function(){
9      res.redirect('dashboard');
10     add.showLink();
11   });
12 }
13 exports.edit = async(req, res) => {
14   try{
15     const {id, titulo, url} = req.body.link
16     const edit = new link(req.body.link);
17     await edit.update(id, titulo, url);
18     res.redirect('dashboard');
19   }catch(e){
20     console.log('erro em atualizar: ', e)
21   }
22 }
23
24 exports.delete = async(req, res) =>{
25   try{
26     const{id} = req.body.link;
27     const del = new link(req.body.link);
28     await del.delete(id);
29     res.redirect('dashboard');
30   }catch(e){
31     console.log('deu erro em deletar', e);
32   }
33 }
```

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

```
exports.dashboard = async (req, res) => {
  if(req.session.user){
    const link = new Link(req.session.user, req)
    const linkTable = await link.showLink()
    res.render('dashboard', {
      user: req.session.user.username,
      showMessage: req.query.q === 'logout',
      links: linkTable
    });
    return
  }
  res.redirect('login');
}
```

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 atribuímos 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 %>"></button>
          </li>
        <% }) %>
      <%}%>
    </ul>
  </nav>
  <button class="btn-add-link">Adicionar</button>
</div>
```

Como ele retorna um objeto, então atribuímos 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

```

7 document.querySelectorAll('.btnEdit').forEach(button => {
8   button.addEventListener('click', e => {
9     const btnEdit = e.currentTarget;
10    btnEdit.style.display = 'none';
11
12    const linkId = btnEdit.getAttribute('data-id');
13    const listItem = document.querySelector(`li[data-id="${linkId}"]`);
14    const linkShowList = listItem.querySelector('.link-show-list');
15
16    const titulo = linkShowList.querySelector('h3').innerText;
17    const link = linkShowList.querySelector('p').innerText;
18
19    const token = document.querySelector('meta[name="token"]').getAttribute('value');
20
21    linkShowList.innerHTML = `
22      <form class="form-edit-delete" method="POST">
23        <input type="hidden" name="_csrf" value="${token}">
24        <input type="hidden" name="link[id]" value="${linkId}">
25        <input value="${titulo}" type="text" name="link[titulo]">
26        <input value="${link}" type="text" name="link[link]">
27        <button name="edit" class='salvar' type="submit">Salvar</button>
28        <button name="delete" class='delete' type="submit">Excluir</button>
29        <button class="cancelar">Cancelar</button>
30      </form>
31    `;
32  });
33 }

```

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 atribuído 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>
    `;
});
});
});

```