Unifacs

Ciência da computação

Usabilidade, desenvolvimento web, mobile e jogos

João Vitor Domingues

(1272229434)

João Padin

(1272224769)

João Luiz

(1272226624)

Lucas Freire

(12722217127)

Pedro Lopes

(1272018234)

Rebeca Azevedo

(1272226342)

Loja de jogos online, AV3



Salvador Bahia

2023

**Relatório/A3 Unifacs**

Introdutoriamente iniciamos o projeto em que foi estabelecido pela faculdade a realização de um trabalho que consiste em criar uma aplicação WEB para gerenciar catálogo de jogos de um usuário.

Nos baseando em alguns sites de nossa escolha como forma de inspiração para termos uma base e iniciarmos o projeto.

**<LOGIN>**

Criamos uma <div class= “banner”> juntamente com uma <div id= login> juntamente com a tag <form> sendo utilizada nesse caso para agrupar elementos relacionados em um formulário e é usada para criar vários tipos de campos de entrada, como caixas de texto, botões de rádio e caixas de seleção. Dentro dela, inserimos a <div class=¨conteudo¨> com label(for) e input(type, id, name) finalizando com a tag<button> para o usuário ter acesso através da senha, do email(login), mantendo-se logado e direcionado para a página em html web: Xgames.

Em **JavaScrip,** com a tag <script> utilizamos a função function, sendo funções em JavaScript, blocos de código que realiza uma tarefa, uma operação, executada quando é chamada por alguém ou invocada. Ela permitirá no nosso código que agrupe um conjunto de instruções em uma única unidade lógica, tornando nosso código mais organizado, modular e fácil de manter. Os parâmetros utilizados (ou argumentos) são variáveis definidas dentro dos parênteses na declaração de uma função que recebem valores quando a função é invocada sendo utilizadas no nosso código, (email e senha). Ao declararmos essa função, podemos especificar um ou mais parâmetros entre parênteses. Cada parâmetro utilizado nesse caso é um identificador (um nome) que representa um valor que será passado para a função quando ela for chamada.

**#login.css**

Estilizamos o login, na parte do (**body{background-image** em sua **margin,padding** e **height** assim como **(.banner, #login, .conteudo, h1 , label)** nas entradas **(Input, Input:not, button e a).**

**Pegando elementos através da DOM (Document Object Model)**

Seguindo a linha de raciocínio acima, precisávamos pegar os elementos do nosso documento HTML, para que pudéssemos manipulá-los. Primeiro precisamos pegar nosso documento, tendo acesso a ele através da variável global “document”, depois utilizamos um dos métodos disponíveis.

O **getElementById** vai nos trazer o primeiro elemento que ele encontrar com o id que passarmos para ele. No nosso código, vai nos trazer o primeiro elemento que possuir o id id do elemento (email e senha).

**<Cabeçalho/Barra de navegação>**

De início, começamos pela barra de navegação, para facilitar o acesso de funções específicas pelo usuário com a tag <header> definindo o cabeçalho da seção e dentro dele introduzimos a tag <nav> que define um conjunto de links de navegação ou um menu. Juntamente com uma class denominada (“nav-list”) em que colocamos para listas desordenadas, conteúdo da lista, os itens, são definidos com a tag <li> juntamente com <a href>. A tag define um hiperlink que permite que se navegue entre páginas como (LOJA, COMUNIDADE, SOBRE e SUPORTE). Seu atributo mais importante é o href, que define o destino dos links. Em seguida finalizando essa parte, concluímos com a tag <input> para inserirmos o campo de entrada de texto juntamente com a tag <button> para os botões.

Importante destacar também, que criamos uma logo (X GAMES) como marca registada do nosso site através de inteligência artificial e introduzimos na barra de navegação.

**<Carousel>**

Iniciamos criando uma classe (“slider”) com a tag <div> criando blocos que se estendem até a borda horizontalmente pos vemos como uma forma interessante de agrupar informações na nossa página HTML reunindo os conteúdos em blocos e divisões em forma de carousel.

Criamos outra classe (“navigation-auto”) para a navegação automática sem necessitar clicar nas imagens para que elas percorram passando uma por uma.

Sendo assim colocamos a tag <img src> inserindo imagens que serão apresentadas junto com os textos. Seu atributo mais importante e obrigatório é o src, que define a origem das figuras obtidas pela internet do site da Google.

**<Card de Jogos>**

Criamos uma Tag **<div id= “linha1”>** com a **<div class= “card”>** no mesmo formato e tamanho para deixarmos alinhados lado a lado cada jogo em destaque com a imagem, texto e botão encaixado como um cartão de amostra para visualização do usuário.

**#style.css**

Para o estilo, colocamos em(**.flex-container>div)** e estilizamos o (**background-color, width, margin, text-align , line-height e font-size).**

**JavaScript das principais funções da página**

**Card de Jogos**

**var nome = document.getElementById("nomeJogo").value;**  
**var genero = document.getElementById("generoJogo").value;**  
**var fxE = document.getElementById("faixaEtaria").value;**  
**var dt\_lancamento = document.getElementById("dtLancamento").value;**  
**var desenvolvedora = document.getElementById("desenvolvedora").value;**  
**var valor = document.getElementById("valor").value;**  
**var avaliacao = document.getElementById("avaliacao").value;**  
**var img = document.getElementById("logoImg").value;**  
**var overview = document.getElementById("overview").value;**

Utilizamos esse trecho do código para obter os valores dos campos de entrada do formulário de cadastro de jogo. Cada variável corresponde a um campo de entrada diferente.

**if(nome == "") {**  
 **alert("Digite o Nome do Jogo.");**  
 **document.getElementById("nomeJogo").focus();**  
**} else if(genero == "") {**  
 **alert("Digite o Gênero do Jogo.")**  
 **document.getElementById("generoJogo").focus();**  
**} ...**  
Esta é uma série de verificações para garantir que todos os campos de entrada foram preenchidos. Se um campo estiver vazio, um alerta será exibido para o usuário e o foco será movido para o campo vazio.

**fetch('http://localhost:3000/getDataJogos')**  
 **.then(response => response.json())**

**.then(data=>{**

**let x = true;**  
 **let count = 0;**  
 **var nomeJ = [];**  
 **var cadastrar = true;**  
  
 **while (x) {**  
 **try {**  
 **nomeJ.push(data[count].nome);**  
 **count += 1;**  
 **} catch {**  
 **x = false;**  
 **}**  
 **}**

**for (let i = 0; i < nomeJ.length; i++) {**  
 **if(nomeJ[i] == nome){**  
 **alert("Este jogo já esta cadastrado ou já existe um jogo com esse nome cadastrado.")**  
 **cadastrar = false;**  
 **break;**  
 **}**  
 **}**

Neste trecho de código faz uma solicitação GET para o **endpoint /getDataJogos** do servidor para obter todos os jogos existentes. Em seguida, verifica se o nome do jogo que está sendo cadastrado já existe. Se existir, um alerta será exibido para o usuário e o jogo não será cadastrado.

**if(cadastrar){**  
**vardata = {**  
 **nome: nome,**  
 **genero: genero,**  
 **faixa\_etaria: fxE,**  
 **dt\_lancamento: dt\_lancamento,**  
 **desenvolvedora: desenvolvedora,**  
 **valor: valor,**  
 **avaliacao: avaliacao,**  
 **img: img,**  
 **overview: overview**  
 **};**  
  
 **fetch('http://localhost:3000/cadastrar\_jogo' , {**  
 **method: 'POST',**  
 **headers: {**  
 **'Content-Type': 'application/json'**  
 **},**  
 **body: JSON.stringify(data)**  
 **})**  
 **.then(response => response.json())**  
 **.then(data => {**  
 **console.log('Resposta:', data);**  
 **})**  
 **.catch(error => {**  
 **console.error('Erro:', error);**  
 **});**  
 **alert('Jogo Cadastrado.');**  
 **location.reload();**  
**}**  
Se o nome do jogo não existir, o jogo será cadastrado. Os dados do jogo são enviados como uma solicitação POST para o endpoint /cadastrar\_jogo do servidor. Se a solicitação for bem-sucedida, um alerta será exibido para o usuário informando que o jogo foi cadastrado e a página será recarregada.

**Loja/JS**

**fetch('http://localhost:3000/getDataJogos')**  
 **.then(response => response.json())**  
 **.then(jogos => {**  
 **const listaJogos = document.getElementById('objeto-list');**

Neste trecho de código faz uma solicitação GET para o endpoint /getDataJogos do servidor para obter todos os jogos existentes. Em seguida, obtém uma referência ao elemento HTML com o ID objeto-list.

**jogos.forEach(jogo => {**  
 **const div = document.createElement('div');**  
 **div.className = 'conteudo';**  
 **div.innerHTML = `**  
 **<img src="${jogo.img}" class="card-img-top" height="300">**  
  
 **<h4 class="card-title">${jogo.nome}</h4>**  
 **<p>R$ ${jogo.valor}</p>**  
 **<button type="button" class="button" onclick="pagJogos(${jogo.id})">Comprar</button>**  
 **`;**  
 **listaJogos.appendChild(div);**  
**});**

Neste trecho de código percorremos cada jogo retornado pelo servidor. Para cada jogo, criamos um novo elemento **div**, define sua classe como **conteudo** e preenche seu conteúdo interno com os detalhes do jogo. Em seguida, anexa o div ao elemento **listaJogos.**

**.catch(error => console.error('Error ao obter objetos:', error));**  
Para captura e registrar quaisquer erros que possam ocorrer durante a solicitação GET.

**function pagJogos(id) {**  
 **localStorage.setItem('valor', (id - 1));**  
 **open('paginaJogo.html');**  
**}**  
Esta é uma função que é chamada quando o botão “Comprar” é clicado. Ela armazena o ID do jogo (menos um) no armazenamento local do navegador e, em seguida, abre a página **paginaJogo.html.**

**Para a página do Jogo**

**fetch('http://localhost:3000/getDataJogos')**  
 **.then(response => response.json())**  
 **.then(data => {**  
 **const id = localStorage.getItem('valor');**  
 **document.getElementById('imagem').outerHTML = `<img src=${data[id].img}>`**  
 **document.getElementById('nomeJogo').innerHTML = data[id].nome;**  
 **document.getElementById('overview').innerHTML = data[id].overview;**  
 **document.getElementById('valor').innerHTML = `R$ ${data[id].valor}`;**  
 **}).catch(error => console.error('Erro ao obter dados:', error));**

Com essa parte do código, faz uma solicitação GET para o endpoint /getDataJogos do servidor para obter todos os jogos existentes. Em seguida, obtém o ID do jogo do armazenamento local do navegador e usa esse ID para acessar os detalhes do jogo. Os detalhes do jogo são então exibidos na página.

**function comprar() {**  
 **var add = '';**  
 **const idUser = localStorage.getItem('idUser');**

Esta é a função comprar, que é chamada quando o usuário clica no botão “Comprar”. Ela obtém o ID do usuário do armazenamento local do navegado

**fetch('http://localhost:3000/getDataJogos')**  
 **.then(response => response.json())**  
 **.then(data => {**  
 **const idJogo = localStorage.getItem('valor');**  
 **const nomeJogo = data[idJogo].nome;**

Nesse trecho de código faz uma solicitação GET para o endpoint /getDataJogos do servidor para obter todos os jogos existentes. Em seguida, obtém o ID do jogo do armazenamento local do navegador e usa esse ID para acessar o nome do jogo.

**fetch('http://localhost:3000/getData')**  
 **.then(response => response.json())**  
 **.then(user => {**  
 **if (user[idUser].jogo\_user == null) {**  
 **add = nomeJogo;**  
 **} else {**  
 **add = `${user[idUser].jogo\_user}, ${nomeJogo}`;**  
 **}**

O código faz uma solicitação GET para o endpoint /getData do servidor para obter todos os usuários existentes. Em seguida, verifica se o usuário já tem algum jogo. Se não, o nome do jogo é adicionado à variável add. Se sim, o nome do jogo é adicionado à lista de jogos do usuário.

**JavaScript/Login**

**function validacaoLogin() {**  
 **var email = document.getElementById("email").value;**  
 **var senha = document.getElementById("senha").value;**  
Esta é a função validacaoLogin, que é chamada quando o usuário tenta fazer login. Ela obtém o email e a senha inseridos pelo usuário.

**if (email == "") {**  
 **alert("Coloque seu email");**  
 **document.getElementById("email").focus();**  
**} else if (senha == "") {**  
 **alert("Coloque sua senha");**  
 **document.getElementById("senha").focus();**  
**} else {**

Esta função de código verifica se o email e a senha foram inseridos. Se algum deles estiver vazio, um alerta será exibido para o usuário e o foco será movido para o campo vazio.

**fetch('http://localhost:3000/getData')**  
 **.then(response => response.json())**  
 **.then(data => {**

Este trecho de código faz uma solicitação GET para o endpoint /getData do servidor para obter todos os usuários existentes.

**let x = true**  
**let count = 0**  
**var emails = []**  
**while (x) {**  
 **try {**  
 **emails.push(data[count].email);**  
 **count += 1;**  
 **} catch {**  
 **x = false;**  
 **}**  
**}**

Este trecho de código percorre todos os usuários retornados pelo servidor e extrai seus emails.

**for (let i = 0; i < emails.length; i++) {**  
 **if (emails[i] == email) {**  
 **if (senha == data[i].senha) {**  
 **localStorage.setItem('idUser', i);**  
 **open('XGames.html');**  
 **close('login.html');**  
 **break;**  
 **} else {**  
 **alert("Senha Incorreta!");**  
 **document.getElementById('senha').focus();**  
 **break;**  
 **}**  
 **}**  
 **if (i == (emails.length - 1)) {**  
 **alert("Email não cadastrado!");**  
 **document.getElementById('email').focus();**  
 **}**  
Este trecho de código verifica se o email inserido pelo usuário corresponde a algum dos emails extraídos. Se corresponder, verifica se a senha inserida corresponde à senha do usuário correspondente. Se a senha estiver correta, o ID do usuário é armazenado no armazenamento local do navegador e a página XGames.html é aberta. Se a senha estiver incorreta, um alerta será exibido para o usuário. Se o email não corresponder a nenhum dos emails extraídos, um alerta será exibido para o usuário.

A função **cadastrando\_novo\_user** segue um padrão semelhante, mas é usada para cadastrar novos usuários em vez de fazer login.

Este trecho de código é a parte final da função cadastrando\_novo\_user(). Se a variável cadastrar for verdadeira, o que significa que o email do usuário não está em uso e todas as validações foram aprovadas, o código prossegue para criar um novo usuário.

var data = {  
 nome: nome,  
 email: n\_email,  
 senha: senha,  
 dt\_nascimento: dt\_nascimento,  
 sexo: sexo,  
 n\_celular: n\_celular  
};

Aqui, um objeto data é criado com os detalhes do novo usuário.

**fetch('http://localhost:3000/cadastrar\_user', {**  
 **method: 'POST',**  
 **headers: {**  
 **'Content-Type': 'application/json'**  
 **},**  
 **body: JSON.stringify(data)**  
**})**  
 **.then(response => response.json())**  
 **.then(data => {**  
 **console.log('Resposta:', data);**  
 **})**  
 **.catch(error => {**  
 **console.error('Erro:', error);**  
 **});**

Este trecho de código faz uma solicitação POST para o endpoint /cadastrar\_user do servidor para cadastrar o novo usuário. Os detalhes do usuário são enviados no corpo da solicitação. Se a solicitação for bem-sucedida, a resposta do servidor é registrada no console. Se ocorrer um erro, o erro é registrado no console.

**open('XGames.html');**  
**close('cadastrar\_user.html');**  
Finalmente, se o usuário for cadastrado com sucesso, a página XGames.html é aberta e a página cadastrar\_user.html é fechada.

**Pasta Nodes\_modules**

**Obs: O Css foi estilizado baseado na página principal para encaixar com as características fidedignas.**

**Cadastrar Jogo**

Referente ao cadastro de jogos em que o usuário consegue cadastrar o jogo que lhe deseja,

**<nav>**  
 **<img src="imagens/Logo1.png"><a class="logo" href="XGames.html">XGames</a>**  
 **<ul class="nav-list">**  
 **<li><a href="XGames.html">Loja</a></li>**  
 **<li><a href="XGames.html">Comunidade</a></li>**  
 **<li><a href="XGames.html">Sobre</a></li>**  
 **<li><a href="XGames.html">Suporte</a></li>**  
 **</ul>**  
 **<ul class="nav-list">**  
 **<li><input type="text" id="pesq" name="pesq"></li>**  
 **<li><button type="submit">Pesquisar</button></li>**  
 **</ul>**  
**</nav>**

Esta é a barra de navegação do nosso site. Ela contém um logotipo e links para várias páginas do site (Loja, Comunidade, Sobre, Suporte). Além disso, há uma barra de pesquisa que os usuários podem usar para pesquisar no site.

**<center>**  
 **<h1>Cadastrando Novo Jogo</h1>**  
**</center>** Usamos uma simples tag de título que exibe “Cadastrando Novo Jogo” no centro da página.

**<form method="post" action="JavaScript: cadastrar\_novo\_jogo()">**  
Este é o início do formulário de cadastro de jogo. Quando o formulário é enviado (ou seja, quando o botão “Cadastrar Jogo” é clicado), a função JavaScript **cadastrar\_novo\_jogo()** é chamada.

Utilizamos o campo de entrada onde os usuários podem inserir o nome do jogo. O rótulo para este campo é “Nome do Jogo”. **<div class="conteudo">**  
 **<label for="nome">Nome do Jogo:</label><br>**  
 **<input type="text" name="nomeJogo" id="nomeJogo">**  
**</div>**

O restante do formulário segue um padrão semelhante, solicitando informações como Gênero, Faixa Etária, Data de Lançamento, Desenvolvedora, Valor, Avaliação, URL da Logo do Jogo e Overview.

**<button class="enviar" type="submit">Cadastrar Jogo</button>**  
Este é o botão que os usuários clicam para enviar o formulário. Quando clicado, ele chama a função JavaScript especificada na tag do formulário **(cadastrar\_novo\_jogo()).**

**<script src="JS/cadJogo.js"></script>** Este script é onde a função cadastrar\_novo\_jogo() está definida. Quando o formulário é enviado, os dados inseridos pelo usuário são enviados para esta função para processamento.

**Cadastrando Usuário**

**<div class="video-container">**  
 **<video autoplay loop muted>**  
 **<source src="imagens/background.mp4" type="video/mp4">**  
 **</video>**  
 **</div>**  
Esta parte do código define um vídeo para ser reproduzido automaticamente, em loop e sem som, como plano de fundo da página. O vídeo está localizado em imagens/background.mp4.

**<div class="banner">**  
 **<div id="login">**  
 **<form method="post" action="JavaScript: cadastrando\_novo\_user()">**  
Aqui começamos o formulário de cadastro de usuário. Quando o formulário é enviado (ou seja, quando o botão “Logar” é clicado), a função JavaScript cadastrando\_novo\_user() é chamada.

**<div class="conteudo">**  
 **<label for="nome">Seu nome:\*</label>**  
 **<input type="text" id="nome" name="nome">**  
**</div>**

Este é um campo de entrada onde os usuários podem inserir seu nome. O rótulo para este campo é “Seu nome:\*”.

O restante do formulário segue um padrão semelhante, solicitando informações como Email, Nova senha, Confirmar senha, Data de nascimento, Gênero e Número do Celular.

**<button type="submit">Logar</button>**

Este é o botão que os usuários clicam para enviar o formulário. Quando clicado, ele chama a função JavaScript especificada na tag do formulário (cadastrando\_novo\_user()).

**<script src="JS/scriptLogin.js"></script>**

Este script é onde a função cadastrando\_novo\_user() está definida. Quando o formulário é enviado, os dados inseridos pelo usuário são enviados para esta função para processamento.

**Servidor JS**

**const express = require('express');**  
**const sqlite3 = require('sqlite3').verbose();**  
**const bodyParser = require('body-parser');**  
**const app = express();**  
**const port = 3000;**  
Neste trecho de código importamos os módulos necessários para o servidor. O módulo **express** é usado para criar o servidor web, **sqlite3** é usado para interagir com o banco de dados SQLite, e **body-parser** é usado para analisar o corpo das solicitações HTTP.

**const db = new sqlite3.Database('db\_xgames.sqlite');**  
Aqui, uma nova instância do banco de dados **SQLite** é criada. O banco de dados está localizado no arquivo **db\_xgames.sqlite**.

**app.use((req, res, next) => {**  
 **res.header('Access-Control-Allow-Origin', '\*');**  
 **res.header('Access-Control-Allow-Methods', 'GET, PUT, POST, DELETE');**  
 **res.header('Access-Control-Allow-Headers', 'Content-Type');**  
 **next();**  
**});**  
Neste middleware, usamos para configurar os cabeçalhos de controle de acesso CORS. Isso permite que qualquer origem acesse o servidor e permite métodos HTTP específicos (GET, PUT, POST, DELETE).

**app.use(express.static('public'));**  
**app.use(bodyParser.json());**  
Com isso,o primeiro middleware serve arquivos estáticos da pasta public. O segundo middleware analisa o corpo das solicitações HTTP como JSON.

**app.get('/getData', (req, res) => {**  
 **db.all('SELECT \* FROM tb\_user', (err, rows) => {**  
 **if (err) {**  
 **res.status(500).send('Erro ao obter dados do banco de dados.');**  
 **} else {**  
 **const data = rows;**  
 **res.json(data);**  
 **}**  
 **});**  
**});**  
Este é um endpoint GET que retorna todos os dados da tabela tb\_user. Se ocorrer um erro ao obter os dados, uma mensagem de erro é enviada com o status 500. Caso contrário, os dados são enviados como uma resposta JSON.

**app.post('/cadastrar\_user', (req, res) => {**  
 **const { nome, email, senha, dt\_nascimento, sexo, n\_celular } = req.body**;  
  **db.run('INSERT INTO tb\_user (nome, email, senha, dt\_nascimento, sexo, n\_celular) VALUES (?, ?, ?, ?, ?, ?)', [nome, email, senha, dt\_nascimento, sexo, n\_celular], function (err) {**  
 **if (err) {**  
 **return res.status(500).json({ error: err.message });**  
  
 **const userID = this.lastID;**  
 **// Responder com os dados do objeto cadastrado**  
 **res.status(201).json({ id: userID, nome, email, senha, dt\_nascimento, adm: 0, sexo, n\_celular, jogo\_user: null });**  
 **});**  
**});**

Um endpoint POST que insere um novo usuário na tabela tb\_user foi criado. Os dados do usuário são extraídos do corpo da solicitação. Se ocorrer um erro durante a inserção, uma mensagem de erro é enviada com o status 500. Caso contrário, os dados do usuário são enviados como uma resposta JSON com o status 201.

Os outros endpoints seguem um padrão semelhante, lidando com diferentes tabelas e operações.

**app.listen(port, () => {**  
 **console.log(`Servidor está ouvindo na porta ${port}`);**  
**});**  
E finalmente, o servidor começa a ouvir na porta especificada, que neste caso é a porta 3000.