

Trabalho de grupo

Bruno Jesus Sousa Mendes, Nº 20367

Vitor Hugo Dinis da Silva, Nº 20357

Xavier Alves Monteiro, Nº 20347

**Curso Técnico Superior Profissional em Desenvolvimento Web e Multimédia**

Polo de Famalicão, junho, 2021

Índice

1 Indice Figuras 3

2 Introdução 4

3 Projeto 5

3.1 Home 5

3.2 Choose Game 6

3.3 Black Jack 7

3.3.1 Cartas 7

3.3.2 Serviço 9

3.3.3 Deal 10

3.3.4 Hit 12

3.3.5 Stand 15

3.3.6 EndGame 16

3.4 Crypto Game 18

3.4.1 Serviço 18

3.4.2 TS 19

3.4.3 HTML e CSS 21

3.4.4 Filho 23

3.5 Pagina 404 25

4 Conclusão 27

# Indice Figuras

[Figura 1 - Home - Apresentação 5](#_Toc75468568)

[Figura 2 - Home – CSS 5](#_Toc75468569)

[Figura 3- ChooseGame – Apresentação 6](#_Toc75468570)

[Figura 4 - ChooseGame – Apresentação 6](#_Toc75468571)

[Figura 5- Black Jack - ViewChild 7](#_Toc75468572)

[Figura 6- Black Jack - Class 7](#_Toc75468573)

[Figura 7- Black Jack - Declaração de variáveis do tipo Carta 8](#_Toc75468574)

[Figura 8- Black Jack - Criar Cartas 8](#_Toc75468575)

[Figura 9- Black Jack - Serviço 9](#_Toc75468576)

[Figura 10- Black Jack - Deal 10](#_Toc75468577)

[Figura 11- Black Jack -Validar Cartas 10](#_Toc75468578)

[Figura 12- Black Jack - Validar Pontos 11](#_Toc75468579)

[Figura 13- Black Jack - Organizar 11](#_Toc75468580)

[Figura 14- Black Jack - Hit 12](#_Toc75468581)

[Figura 15- Black Jack - Pode Continuar 13](#_Toc75468582)

[Figura 16- Black Jack - Pode Continuar 2 13](#_Toc75468583)

[Figura 17- Black Jack - Contar Pontos 14](#_Toc75468584)

[Figura 18- Black Jack - Stand 15](#_Toc75468585)

[Figura 19- Black Jack - End Game 16](#_Toc75468586)

[Figura 20- Black Jack - End Game 2 17](#_Toc75468587)

[Figura 21- Black Jack - Reeinicar o jogo 17](#_Toc75468588)

[Figura 22- Crypto Game - Resultado 18](#_Toc75468589)

[Figura 23- Crypto Game - Service 18](#_Toc75468590)

[Figura 24- Crypto Game - Construtor e Variáveis 19](#_Toc75468591)

[Figura 25- Crypto Game - Função ngOnInit 19](#_Toc75468592)

[Figura 26- Crypto Game - Funções 20](#_Toc75468593)

[Figura 27- Crypto Game – HTML 21](#_Toc75468594)

[Figura 28- Crypto Game - Media queries 22](#_Toc75468595)

[Figura 29- Crypto Game - Filho – Resultado 23](#_Toc75468596)

[Figura 30- Crypto Game - Filho – Router 23](#_Toc75468597)

[Figura 31- Crypto Game - Filho - Funções 24](#_Toc75468598)

[Figura 32 - 404 – Resultado 26](#_Toc75468599)

# Introdução

Este trabalho foi desenvolvido no âmbito da Unidade Curricular de Programação Web II do curso de CTESP em Desenvolvimento Web e Multimédia do IPCA em Famalicão. A realização do mesmo ocorreu durante o mês de junho.

O tema para este trabalho foi o desenvolvimento de dois jogos (blackjack e crypto). O trabalho foi proposto pelo Professor Moisés Moreira, com o intuito de consolidar os conhecimentos adquiridos em aula.

No processo de desenvolvimento do trabalho, recorreu-se a ferramentas, tais como o Visual Studio Code com intuito de construir e depurar aplicativos modernos da web.

As linguagens que utilizamos foi Angular e TypeScript, pois foram linguagens dadas em aula presencial e online.

# Projeto

## Home

Na parte da home, para apresentar o nosso grupo, decidimos criar uma tela dividida em 3.



Figura 1 - Home - Apresentação

A classe SPLIT é correspondente á divisão da tela. Esta classe tem como propriedades a altura da tela, a largura da tela como no nosso grupo somos 3 e a tela da pagina é de 100px, não deu para dividir precisamente a pagina então ficou com está na imagem, a posição é fixa, propriedade z-index especifica a ordem da pilha de um elemento, a propriedade top afeta a posição vertical de um elemento posicionado e a propriedade overflow-x especifica se deve cortar o conteúdo, adicionar uma barra de rolagem ou exibir o conteúdo de estouro de um elemento de nível de bloco, quando ele estourar nas bordas esquerda e direita.

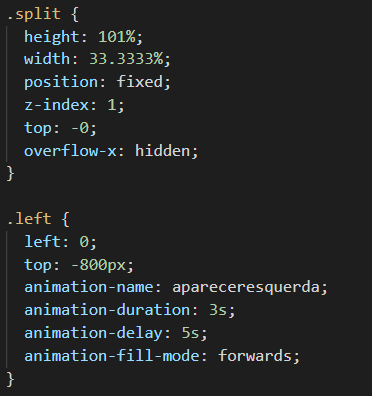


Figura 2 - Home – CSS

## Choose Game

Na parte de escolher o jogo, criamos uma tela dividida em 2, seguindo o padrão da home.



Figura 3- ChooseGame – Apresentação

A classe SPLIT é correspondente á divisão da tela. Esta classe tem como propriedades a altura da tela, a largura da tela, desta vez, diferente da home dividiu se em 2, então dividiu se a tela em 50%.

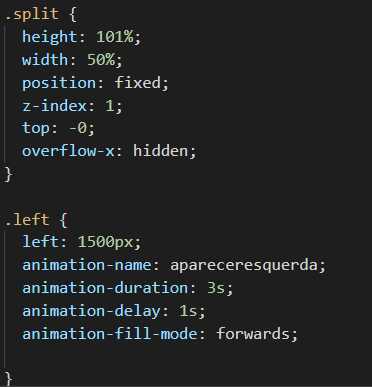


Figura 4 - ChooseGame – Apresentação

## Black Jack

### Cartas

Uma imagem com texto

Descrição gerada automaticamente

Figura - Black Jack - ViewChild

Em angular na parte do typescript uma das maneiras de aceder ao html é usando o ViewChild com uma referência.

Na imagem em cima podemos ver que queremos aceder as cartas do dealer e do player que no momento só tem uma imagem da capa de uma carta.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Black Jack - Class

Esta é a nossa classe Carta, decidimos criá-la porque a Carta é um objeto, e para alem de ter propriedades simples e de fácil entendimento também permite ter métodos próprios.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Black Jack - Declaração de variáveis do tipo Carta

Na imagem seguinte estamos a atribuir valores às variáveis criadas em cima e depois adicioná-las aos seus arrays específicos (array de cartas do player e array de cartas do dealer).

Iremos abordar o pedido á API e a função validarPontos mais á frente.

Uma imagem com texto, eletrónica, teclado

Descrição gerada automaticamente

Figura - Black Jack - Criar Cartas

### Serviço

Uma imagem com texto

Descrição gerada automaticamente

Figura - Black Jack - Serviço

Este é o serviço que criamos para poder fazer pedidos á API

Tem a função **getBaralho** que faz o pedido de um baralho de cartas, recebendo como resposta um id de baralho que usamos para poder fazer pedidos específicos ao nosso baralho

A função **getCards** é o pedido que dá início ao jogo, ele pede 4 cartas á API que nós depois como mostrado anteriormente atribuímos 2 ao dealer e 2 ao player.

A função **hit** é semelhante ao getCards sendo a diferença que pede apenas uma carta. O hit no blackjack é algo que o jogador tem opção ou não de pedir por isso está associado a um botão, também usamos essa função para o dealer pois ao dar Stand, o jogador quer terminar o jogo e ver as cartas do dealer, o dealer precisa de pedir novas cartas ou não, para tentar ter mais pontos que o player.

### Deal



Figura - Black Jack - Deal

O jogo começa quando a função **deal** é chamada (a partir do botão deal). Primeiro começamos por validar se o jogador apostou ou não só de seguida que fazemos e tratamos o pedido.

Começamos por fazer o pedido getBaralho para obter o id do nosso deck, se já tivermos um id (por que o jogador continuo a jogar) optamos por usar o pedido getCards de forma a não sobrecarregar a API com pedidos de novos baralhos quando podemos simplesmente “baralhá-lo”.

Depois de efetuado o pedido nos tratamos-lho em 3 partes:

-Usamos a função **validarNcartas** que verifica o número de cartas restantes para sabermos se precisamos de “baralhar” ou não.

-A função **criarCartas** como podemos ver uma parte na Figura 6

-A função **organize** que faz a “ligação” entre os elementos html respetivos às cartas e as cartas em si.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Black Jack -Validar Cartas

Uma imagem com texto

Descrição gerada automaticamente

Figura - Black Jack - Validar Pontos

Como dito anteriormente esta função é chamada na criação das Cartas, para “traduzir” o valor que a API nos dá da Carta para o valor que nós queremos. Exemplo:

Valor da API: QUEEN 🡪 dama

Valor que nós queremos: 10 (o normal é 2, mas no blackjack certas cartas valem 10 pontos)

O valor do às (ACE) é especial e também será abordado mais á frente.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Black Jack - Organizar

Nota: A variável contar, guarda o valor de quantos vezes o jogador fez o pedido hit (pedir uma nova carta).

### Hit

Uma imagem com texto, interior, captura de ecrã

Descrição gerada automaticamente

Figura - Black Jack - Hit

Como dito anteriormente o pedido hit tem as suas semelhanças ao deal, mas o que principal distingue na abordagem que fizemos é nós no hit precisamos de saber quem é a entidade que pede uma nova carta (usamos um parâmetro para saber quem é) para depois organizar e validar.

Como já abordamos a função **criarCartas** e a **organize**, agora vamos explicar a **PodeContinuar**.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Black Jack - Pode Continuar

Uma imagem com texto

Descrição gerada automaticamente

Figura - Black Jack - Pode Continuar 2

No blackjack perde-se automaticamente quando a soma dos valores das cartas excede os 21, ao fazer o pedido hit (pedir nova carta) precisamos verificar se quer o dealer ou o player pode continuar o jogo ou se perdem por excederem os pontos.

Iremos abordar a função **EndGame** que recebe como parâmetro o “código” de como acabou a partida mais para a frente.

Antes de passar para a função **Stand** iremos mostrar a função **contarPontos**.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Black Jack - Contar Pontos

A função de contar é relativamente simples, ela lê todas as cartas que estão na mão da entidade que recebemos (array de Cartas) e conta os seus pontos. Como dito anteriormente o ás é especial no blackjack ele pode ter valor 1 ou 11. O valor por defeito é 1, mas se na “mão” da entidade estiver ou um valete (JACK) ou uma dama (QUEEN) ou um rei (KING) o ás passa a valer mais 10 pontos.

### Stand

Uma imagem com texto

Descrição gerada automaticamente

Figura - Black Jack - Stand

Como dito no início, a função é chamada quando o jogador acha que tem uma boa “mão” e quer acabar o jogo e mostrar as cartas.

Começamos por contar os pontos de ambos e se os pontos do dealer forem menores ele tem de pedir outra carta (recorrendo á função hit), porque o objetivo do dealer é ter mais pontos que o jogador, senão o jogador perde automaticamente, chamando a função para terminar o jogo (EndGame) com o código que o jogador perdeu (p-lost: player lost).

### EndGame

Uma imagem com texto

Descrição gerada automaticamente

Figura - Black Jack - End Game

Uma imagem com texto

Descrição gerada automaticamente

Figura - Black Jack - End Game 2

Começamos por revelar as cartas do dealer (as cartas do player são reveladas 2 no início e sempre que ele faz o pedido de mais uma carta).

Depois baseado no “código” que recebemos mostramos a informação com o vencedor ou perdedor e os pontos, mas antes obviamente retiramos ou damos dinheiro para o jogador.

Depois de 2 segundos (tempo suficiente para ver as cartas) chamamos a função **Restart** para poder jogar outra vez.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Black Jack - Reeinicar o jogo

## Crypto Game



Figura 22- Crypto Game - Resultado

### Serviço

Para fazermos o jogo das criptomoedas começamos por criar um serviço, nele criamos duas variáveis e associamos um link, esse link é para buscar dados a uma api de criptomoedas, neste trabalho decidimos trabalhar com duas moedas, a Bitcoin e a Ethereum, temos outras duas variáveis que vão ser para guardar dados no decorrer do jogo.

Também temos duas funções com o objetivo de ir buscar os dados a api e retorna-los onde for chamado.



Figura 23- Crypto Game - Service

### TS

Depois de criado o serviço, vamos programar o typescript principal, começando pelo construtor, este é o responsável pela ligação do ficheiro typescript com o serviço e pelos serviços router. De seguida temos a criação das variáveis.

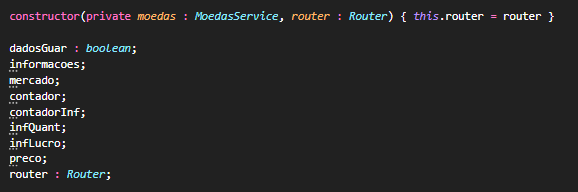


Figura 24- Crypto Game - Construtor e Variáveis

Em seguida na função ngOnInit() vamos colocar o que queremos que o jogo faça logo que é carregado. Então ele começa por executar as funções eth() e btc(), vê se existe o item “Informações” no localStorage e se caso não existir ele cria o mesmo, se não ele vai executar a função buscarInf().

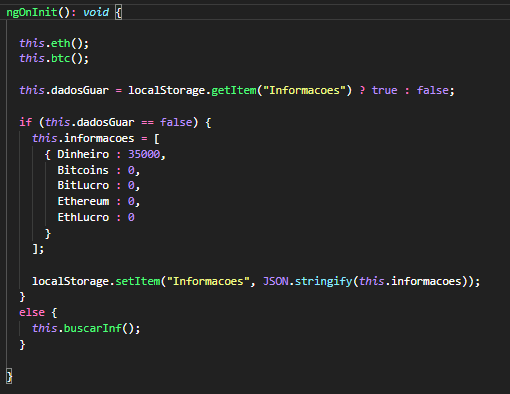


Figura 25- Crypto Game - Função ngOnInit

Agora temos as funções que achamos necessárias para o funcionamento do jogo em que cada uma tem um prepósito diferente.

* sair() – sai do jogo e volta para a página “/chooseGame”;
* buscarInf() – vai buscar e guardar os dados gravados em localstorage e ativa a função refreshInf();
* refreshInf() – é uma função que em 1 em 1 segundo ativa a função buscarInf(), com o propósito de manter as informações sempre atualizadas;
* btc() e eth() – estas funções vão buscar as funções do serviço, estas são responsáveis por guardar as informações das moedas para variáveis, como também guardar informações nas variáveis do serviço, elas também ativam as funções arredondar() e refresh();
* arredondar() – esta função vai arredondar os preços das moedas em cada loja;
* refresh() – a cada 30 segundos esta função vai ativar a função btc() ou a eth(), dependendo de qual valor está numa variável do serviço, que é a moeda que o utilizador escolheu, assim esta função mantem os dados o mais atualizado possivél;
* comprarVender(loja, preco) – por fim temos a função que permite ao utilizador escolher uma loja pelo qual está interessado em comprar ou vender moedas, esta função tem os parâmetros loja, que será passado para o filho, e preco, que será atribuído a uma variável no serviço.



Figura 26- Crypto Game - Funções

### HTML e CSS

Este é o nosso html, é através dele que o jogador vai interagir e por isso nós vamos mostrar os valores das variáveis, vindas do typescript, que achamos necessário, além de informar do que se trata, também tem como ativar funções e assim alterar os valores amostrados, abrir o filho para comprar/vender moedas e para sair do jogo.

Usamos várias divs para organizar as informações usando o CSS.

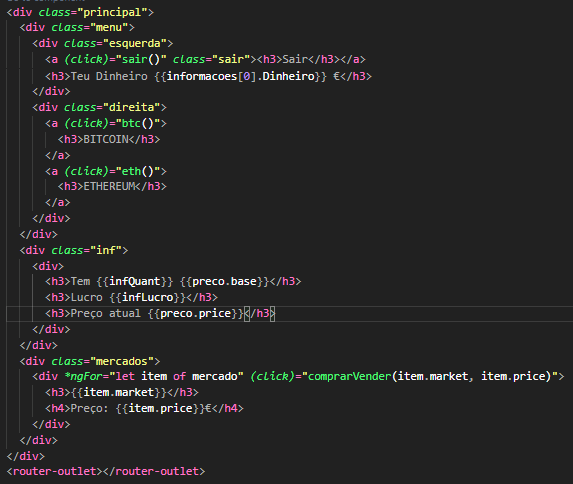


Figura 27- Crypto Game – HTML

Como já dissemos, usamos o CSS para organizar as informações através de class’s e também para colocar o jogo responsivo, neste caso para computador, tablet e telemóvel, para isso usamos media queries que permite definirmos os estilos de uma class para cada tipo de resolução.

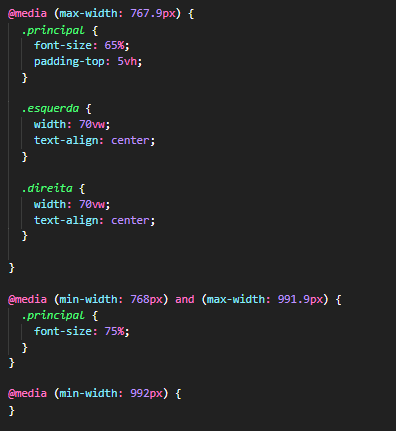


Figura 28- Crypto Game - Media queries

### Filho

O filho do jogo é onde o jogador vai comprar/vender uma moeda no mercado que escolheu e tem a responsabilidade de atualizar os dados do localStorage.

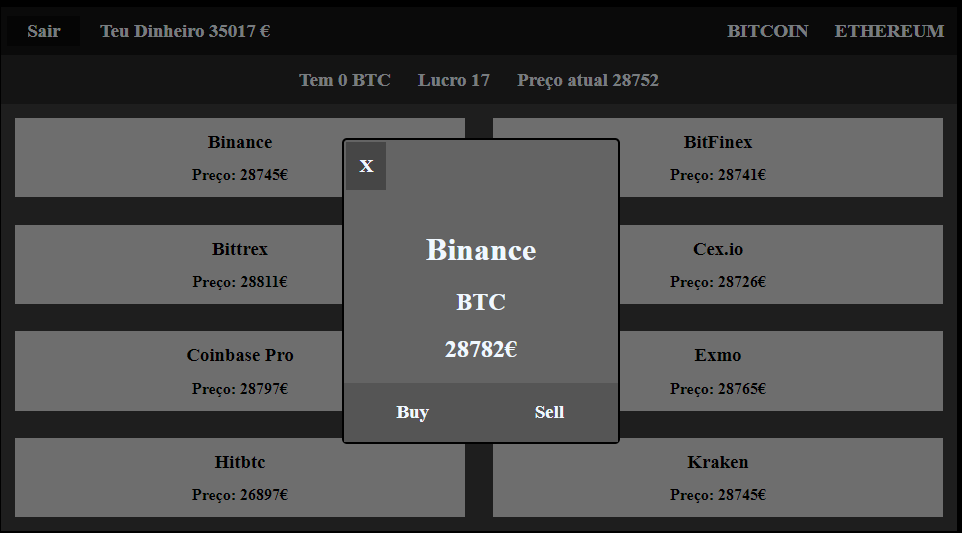


Figura 29- Crypto Game - Filho – Resultado

Para saber qual mercado o jogador escolheu nós definimos que ele ia receber um valor pelo router e no jogo dizemos qual é esse valor, neste caso passamos o nome da loja. O preço e o tipo de moeda vêm do serviço.

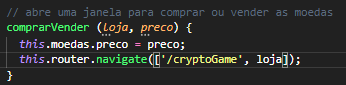


Figura 30- Crypto Game - Filho – Router

Para comprar e vender, nós criamos duas funções para, ficticiamente, fazer as transações. As funções são idênticas o que muda são as operações e as condições. A comprar() vê se temos dinheiro para a compra, e se sim, aumenta o numero da moeda selecionada e retira ao lucro e dinheiro o preço da mesma. Já a vender() vê se temos moedas para vender, e se sim, retira uma moeda e soma o lucro e o dinheiro com o preço da mesma.

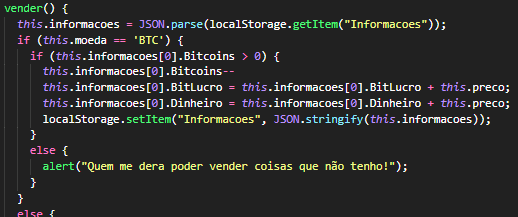
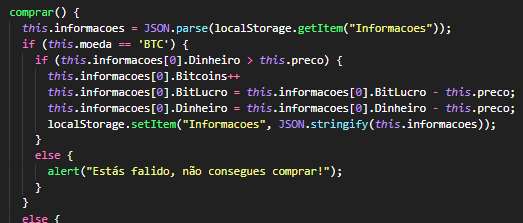


Figura 31- Crypto Game - Filho - Funções

## Pagina 404

Abrir o arquivo src / app / app-routing.module.ts, ele deve conter o seguinte código:

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

@NgModule({

declarations: [],

imports: [

CommonModule

]

})

export class AppRoutingModule { }

Este é um módulo Angular típico e precisamos adicionar a configuração de roteamento a ele.

Agora vamos adicionar a configuração do roteador ao módulo de roteamento. Primeiro importar Routes e RouterModule de @ angular / router da seguinte forma:

import { Routes, RouterModule } from '@angular/router';

Em seguida, definir uma matriz de routes que conterá nossas rotas da seguinte maneira:

const routes: Routes = [];

Em seguida, incluir a chamada do método forRoot () do RouterModule com a matriz de routes como o argumento a seguir:

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule { }

Agora, vamos ver como lidar com a página 404 não encontrada no Angular. Primeiro abrimos o terminal e executar o seguinte comando para gerar um componente:

#ng g component 404

Em seguida, abrir o arquivo src / app / app-routing.module.ts e adicionar estas duas routes:

const routes: Routes = [

// [...]

{path: '404', component: FourOrFourComponent},

{path: '\*\*', redirectTo: '/404'}

];

Não esquecer de importar o componente no módulo de roteamento da seguinte maneira:

import { FourOrFourComponent } from './components/four-or-four/four-or-four.component';

Resultado:

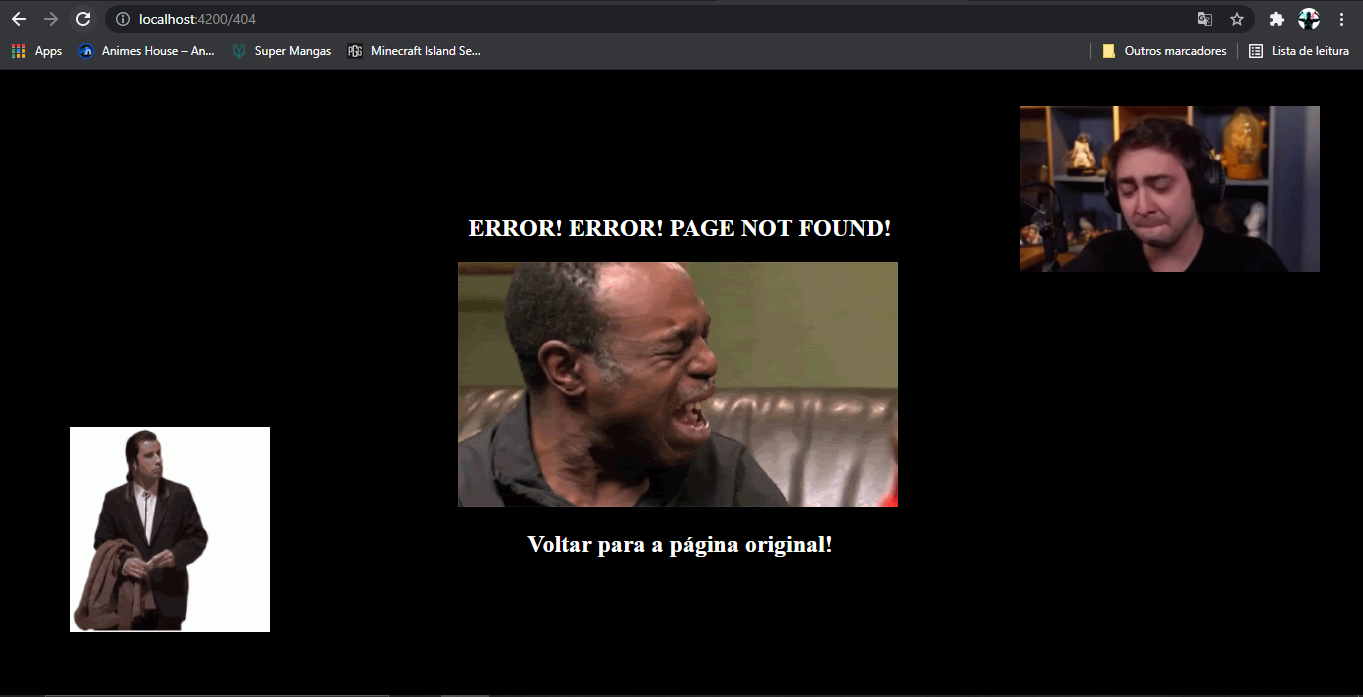


Figura 32 - 404 – Resultado

# Conclusão

Com este trabalho aplicamos os conhecimentos adquiridos em aula, adquiro-mos novos conhecimentos, recordamos conhecimentos do semestre anterior e aprimoramos os mesmos.

Foi um trabalho interessante e desafiador onde conseguimos ultrapassar dificuldades e assim concluir com o que foi proposto.