INTRODUÇÃO

O projeto é composto por:

#HTML

Index.html: O Index é composto por uma estrutura básica de html, e é através dele que iremos construir o esqueleto de nosso site, onde trabalhamos com chamada de scripts, chamada de arquivos de estilização e afins.

É importante realizar a chamada da função **getNumber**, responsável por fazer o get da api para que tudo funcione conforme esperado. O tipo do script é module, fazendo com que não tenhamos problemas em esperar a promessa de resposta, por parte da api. A declaração de window é necessária para garantir que as variáveis tenham escopo global.

#CSS

Grid.css: Arquivo responsável pela declaração e configuração do nosso display grid;

Reset.css: Reset simples de css, que retira estilizações padrões de elementos e declara o box-sizing.

Style.css: Responsável pela estilização geral da index.

O grande segredo está na utilização de containers, com display do tipo grid. A utilização do Flex foi importante para centralizar os componentes e garantir que tudo fique responsivo, com o apoio é claro das mediasqueries, que permitem realizar "quebras" em pontos no qual podemos estilizar condicionalmente elementos específicos para garantir a responsividade. A Responsividade foi trabalhada até o ponto de 300 de largura tela, com uma margem de erro, simulando telas dos menores celulares disponíveis atualmente, que em grande parte nem disponíveis no mercado estão.

#JAVASCRIPT

Main.js: Responsável por realizar o get na api, tratar e receber o input do usuário e dar início a funções especificas, de acordo com determinadas condições.

Game.js: Responsável por conter a lógica do game (É maior... É menor...).

ExibirNumeros.js: Responsável por exibir os números no front em formato de display.

// Lógica utilizada.

No ato do carregamento da página chama-se a função **getNumber()** , que consome a API e retorna dois valores que utilizaremos como base, o *number* e o *statusCode*. O *number* é o número aleatório, que pode ser de 1 a 300 e o *statusCode* é o status de nossa requisição. A ideia é ter esses retornos gravados em variáveis, facilitando o uso delas sempre que preciso.

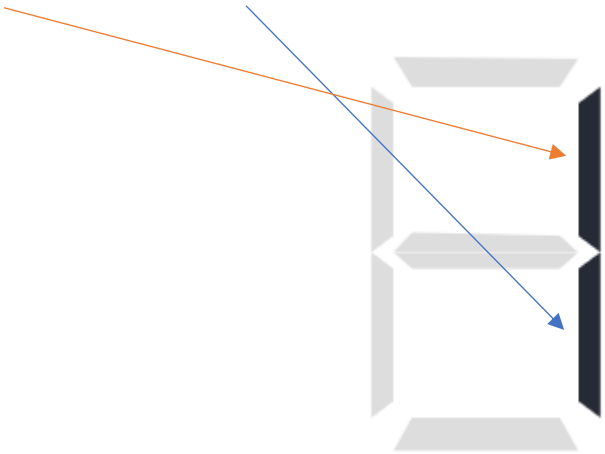
Nesse momento o nosso display já estará renderizado na tela. Utilizamos no total 3 containers, onde seu display foi setado como Grid e sua renderização na tela é condicional, dependendo de qual número o usuário irá inputar. O grid literalmente insere um sistema de grades, atendendo as nossas expectativas por conta de sua robustez e dimensões que podem ser fixas ou flexíveis. No nosso caso utilizamos dimensões fixas, definindo o tamanho em pixels. Como a api irá nos fornecer um número de 1 a 300 vamos considerar um container, que será um grid, para cada número, ou seja, se o usuário digitar o número 300 o sistema deverá colocar o primeiro número no primeiro container, o segundo número no segundo e por aí vai... Ex:

#CONTAINER	#CONTAINER2	#CONTAINER3
3	**0**	**0**

Sabendo-se que cada número deverá ser apresentado em formato de led, vamos definir um layout de 3x4 em cada container. Lembrando que o led disponibiliza 7 segmentos. Vamos desenhar um trapézio em cada posição, tentando chegar o mais próximo de um segmento digital. É necessário criar uma classe para cada posição do grid, ex:

.	Topo1	.
Topo1-esquerda	Meio1-alto	Topo1-direita
Baixo1-esquerda	Meio1-baixo	Baixo1-direita
.	Baixo1	.

Através desta classe será possível estilizar cada posição. A ideia é pintar de preto os segmentos que formarão o número imputado e pintar de cinza os que não serão “utilizados”. Seguindo a tabela acima, se fossemos apresentar o número 1 teríamos que colorir de preto a classe ***Topo1-direita*** e ***Baixo1-direita***.



Basicamente teremos uma função, que irá identificar qual o primeiro, segundo e terceiro número, e o colocará dentro do seu devido container.

Se for unidade, apenas um grid será exibido;

Se for dezena dois grids serão exibidos;

Se for centena, todos os 3 containers grids serão exibidos.

Mas como exibir isso de forma condicional?

A resposta está em nossa função **percorreInput()** que recebe o valor digitado pelo usuário e identifica se é unidade, dezena, ou centena.
Ao carregar a página, o container2 e container3 estarão com display:none, o que impede sua renderização. A função **percorreInput()** seta um display grid nos containers que deverão aparecer, e/ou display:none caso o número imputado caia em alguma condição, citada acima (unidade, dezena...).

E quanto as cores?

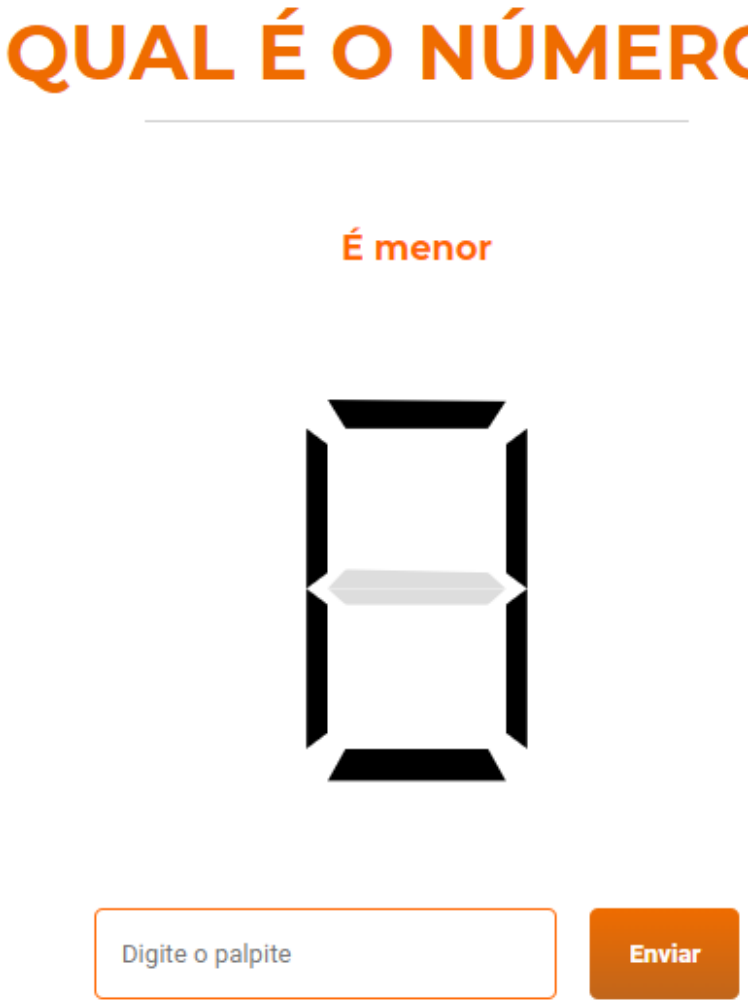
Em nossa função **exibirNumeros()**, responsável por exibir os números na tela, temos uma variável, que se chama background. A variável background recebe uma cor, com base no número recebido via parâmetro, que é o input do usuário. Se o número recebido pela função for maior ou igual a 400 significa que é um erro, e seja ele qual for deverá ser apresentado na cor vermelha, ou seja, background receberá cor vermelha. Bem no inicio da função verificamos qual o texto está sendo apresentado em nosso parágrafo, responsável por dizer se o número é maior, menor ou até mesmo se o usuário acertou. Vamos aproveitar isso, e dizer que sempre que o paragrafo conter a frase “Você acertou!!!” background receberá a cor verde. Desta forma os números são exibidos na cor correspondente.

Obvio que temos outras funções no meio do caminho que fazem com que a funcionalidade seja executada de uma maneira mais inteligente. Uma boa forma de entender é literalmente percorrer a lógica, desde o início, focando na lógica como um todo e tentando é claro deixar o algoritmo o mais claro possível.

Na mente do computador!

Vamos fazer uma espécie de passo a passo e entender como o sistema toma as decisões.

Ao iniciar o sistema veremos a seguinte tela:



A pergunta é: O que já aconteceu, por trás dos panos?

A Api é consumida, e os dois retornos são salvos em variáveis, sendo eles: number e statusCode. A Função chamada log recebe essas duas variáveis, imprime-as no console e verifica o status. Se o status for diferente de 200, há algo de errado, certo? Pois bem, neste caso passaremos a responsabilidade para a função de erro, que receberá o statusCode como parâmetro. A Função erro encaminha o número para a função responsável por imprimir os números na tela, e em seguida executa uma serie de ações, com o apoio de outras funções, como por exemplo:

- Desabilitar input / botão enviar;
- Disponibilizar o botão que permite o início de um novo jogo;
- Exibe o parágrafo ERRO....

Caso tudo tenha dado certo, e nenhum tipo de erro for retornado começaremos o jogo, aguardando que o usuário digite e envie o primeiro palpite.

Quando o usuário começar a digitar uma função será executada, escutando tudo o que for digitado e caso ele digite um número inferior a 1 ou maior que 300 o botão de enviar será desabilitado. O Input está preparado para impedir que caracteres especiais sejam enviados, e caso isso aconteça por algum motivo nada acontecerá.

Primeiro número enviado pelo usuário. Neste momento a função do evento onclick será acionada, e ela será responsável por executar a logica do game, verificando o número recebido pela api com o número digitado pelo usuário. Com base nessa comparação, decisões condicionais serão tomadas exibindo parágrafos personalizados, com frases do tipo: “É menor”, “É maior”, e “Você acertou!!!”.

Dentro dessas decisões terão a chamativa de funções especificas, que farão ações importantes dentro do contexto proposto.