Este workshop é referente a Next Level Week #3 da Rocket Seat, onde aprede-se mais sobre front-End e back-end utilizando HTML, JS e CSS.

Referências:

1. https://www.notion.so/NLW-Discovery-03-628a2c1b9ac744e28fad80046b699aab
2. https://discord.com/invite/as33qEE
3. https://nextlevelweek.com/inscricao/3
4. https://nextlevelweek.com/episodios/discovery/2/edicao/3?utm\_source=discord&utm\_medium=announcement&utm\_campaign=NLW3+Workshops&utm\_term=Leads+Discovery&utm\_content=Workshop+2
5. https://leafletjs.com/index.html

Dia 12/10 – Aula 01:

Configurações iniciais:

Antes de tudo configura-se o ambiente de desenvolvimento, no caso o VScode, e é feito o download dos arquivos citados em [I] nas referências conforme vídeo aula 01 [25:01].

Feito isso, devemos iniciar o arquivo index.html que servira de página inicial para a aplicação.

Um arquivo HTML é feita de tags, estas devem ser abertas em ordem correta pois cada uma exerce uma função, uma maneira rápida de fazer isso que o VScode permite fazer é digitar ‘!’ (exclamação) e apertar enter, e então o editor preenche a tela com as tags necessárias para se criar uma página. Assim como mostrado a seguir:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

</body>

</html>

As tags [<tag>] possuem duas características que devem ser observadas: a primeira é que muitas delas possuem atributos e a segunda que devem ser fechadas, algumas fogem a esta regra, como é o caso da tag <meta>, por exemplo. Conforme mostrado abaixo:

<tag atributo="valor"></tag>

HTML [00:38:56]:

Dito isto, dá-se seguimento para a construção da página em HTML. As instruções para a construção da página estão disponibilizadas na referencia [I].

Dica:

A ideia aqui é que a página é construída de cima para baixo, então as seções devem ser criadas levando isso em consideração, note como no código a seguir a <section> cria em <main> o botão no memso nível que a descrição escrita.

<!DOCTYPE html>

<html lang="pt\_br">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Happy</title>

    <!-- Coloca o icone no titulo da página quando colocado no <head>: -->

    <link rel="icon" href="./public/images/logo-icon.png" />

  </head>

  <body>

    <!-- Define a secao do 'cabeçalho' da página, no caso utilizamos apenas a imagem de logo do app -->

    <header>

      <img src="./public/images/logo.svg" alt="Logomarca de Happy" />

      <!-- Se trata de um container genérico para definir algum espaço a ser 'formatado'  -->

      <div>

        <strong>Taubaté</strong>

        <p>São Paulo</p>

      </div>

    </header>

    <!-- Define a secao que seria a 'principal' da página -->

    <main>

      <h1>Leve felicidade para o mundo</h1>

      <section>

        <p>Visite orfanatos e mude o dia de muitas crianças</p>

        <!-- Botao -->

        <a href="#" title="Visite orfanatos">

          <img src="./public/images/arrow.svg" alt="Ir para o mapa" />

        </a>

      </section>

    </main>

  </body>

</html>

CSS [1:07:18]:

Para adicionar um arquivo css à página HTML deve-se linkalo ao documento HTMl utilizando a tag <link> no <head>:

<link rel="stylesheet" href="./public/css/main.css" />

Dessa maneira fica mais fácil atualizar o estilo CSS da página conforme necessário através dos id’s dos elementos no arquivo HTML.

Para referenciar o elemento pelo ID no arquivo CS basta utilizar ‘#ID\_do\_elemento’; para referenciar por classe (i.e. <div class=”visit”>) basta acessar da seguinte maneira:

.classe\_verde, .classe\_azul p {}

No caso, a regra criada servirá para todos os elementos dentro da classe\_verde e para os elementos ‘p’ dentro de classe\_azul. Veja um exemplo mais prático:

.location,

h1,

.visit p {

  height: 16vh;

}

Aqui para a classe ‘location’, par todos os <h1> e para os <p> dentro da classe ‘visit’ setamos height com 16vh (viewPort height).

Fontes:

Para criar uma família de fontes, basta ir no fonts.google e selecionar as fontes desejadas, após isto, para inserir esta nova família no arquivo HTML temos que pegar seu código embed <link href...> e inseri-lo no <head> do HTML; após isto podemos utilizar esta família no arquivo css.

Quando trabalhamos com fontes é interessante utilizar conceitos de tamanhos em escala, para isto usamos a unidade ‘em’ e ‘rem’ (elemento e root elemento, respectivamente); onde o ‘em’ faz referencia à tag HTML anterior do escopo que estamos trabalhando no css e o ‘rem’ será sempre relativo ao ‘:root’ do arquivo HTML. Por exemplo, se o font-size do :root é 20px, 0.5rem é equivalente à 10px, e assim por diante... é interessante trabalhar nete conceito para que o aplicativo não perca a escala.

Animações:

Pode-se definir animações para serem aplicadas aos elementos da página para torna-la mais visualmente interessante. A Sintaxe para definir uma animação está a seguir:

@keyframes up {

    from {

        opacity: 0;

        transform: translateY(16px)

    }

    to {

        opacity: 1;

        transform: translateY(0)

    }

}

Aqui definimos a animação ‘up’, que faz os objetos aparecerem na tela utilizando a função ‘translateY()’.

É interessante alterar alguns valores desta animação, então para poder fazer isto:

.animate-up {

    animation-name: up;

    animation-duration: 1000ms;

    animation-fill-mode: backwards;

}

Para que o código reconheça onde deve atuar, deve-se definir a calsse ‘animate-up’ como definimos seu nome acima onde queremos que ela atue:

<p class="animate-up">

Ou

<div class="location animate-up"> /\* no caso de já existir uma classe na tag\*/

E então, para que fique ainda mais interessante, deve-se alterar o parâmetro animation-delay em cada um dos blocos em que ela será executada:

#logo {

    animation-delay: 50ms;

}

.location {

    animation-delay: 100ms;

}

main h1 {

    animation-delay: 150ms;

}

.visit p {

    animation-delay: 200ms;

}

.visit a {

    animation-delay: 250ms;

}

Sumarizando, a ideia é criar no documento HTML campos manipuláveis, através de classes e id’s nas tags, para que possamos aplicas regras CSS de maneira que se comportem conforme desejado. Deve-se obedecer as sintaxes, regras de especificidades e de cadeia dos elementos inseridos tanto no documento HTL quanto nas regras CSS.

Dia 13/10 – Aula 02:

Finalizando a landing page para desktop [00:01:14]:

Primeiro adiciona-se uma regra para que o cliente entenda quando se está trabalhando com dispositivos mobile e desktop, essa diferença se dará distância em x, no caso, quando x for maior que 760px estará lidando com dispositivos desktop e tratará a página de maneira diferente. Para isto fazemos o que segue no documento landing-page.css:

/\* Desktop version \*/

@media(min-width: 760px){

  header {

    display: flex; /\* flexibiliza o display \*/

    align-items: center; /\* alinha ele com o eixo horizontal médio entre os elementos contidos no header \*/

    justify-content: space-between; /\* coloca os items nos dois extremos da página \*/

  }

  .location, h1, .visit p {

    height: initial;

  }

}

Tudo isso no final do documento.

OBS: os @something funcionam como regras css que são aplicadas caso alguma condição seja atingida, é como se fosse um ‘if condição atingida, aplique as seguintes regras’.

Agora em ‘#container’ aplicamos uma função para que os elementos contidos nele fiquem espaçados com o mínimo entre 90% da página ou 1120px, ou neste caso, 112rem:

  #container {

    margin: auto;

    width: min(90%, 112rem);}

Procede-se com a adição de regras...

Quanto a organização do código é interessante que se trabalhe de cima para baixo, seguindo a ordem dos elementos como apresentado no conteúdo HTML, desta maneira fica mais fácil a leitura dos itens, além de permitir uma maior navegabilidade.

Page-orphanges.html [00:32:30]:

Para começar a segunda página do aplicativo é mais fácil copiar o index.html e fazer as alterações necessárias nela, de primeira mão basta remover todo o body da página, alterar os links no head necessário e começar a colocar os elementos

Lembrando aqui que o ‘display: flex’ felibiliza o display para os filhos, o que faz com que a pagina coloque um elemento do lado do outro.

Após configurado todos os elementos da página, procede-se à animação ‘animate-right’ que após escrita deve ser colocada como classse do elemento ‘aside’ no html.

JavaScript [00:57:11]:

Pulando a introdução a JS, vamos direto para os finalmentes.

A biblioteca a ser utilizada no app é a LeafLetJS, ela utiliza mapas para pdoer trabalhar na sua pagina web. Para coloca-la no site basta seguir as etapas descritas no tutorail do link [V].

Montando a próxima página [01:51:56]:

Deve-se proceder com o mesmo modelo de montagem das outras páginas, primeiro coloca-se todo o conteúdo da página no html para que então se possa formatar este conteúdo utilizando css.

Quando se utiliza o tipo de display grid, a pagina entende que será montado um grid mesmo para poder posicionar os elementos. neste caso o programa libera a opção ‘grid-template-column’ na qual pode-se usar a função ‘repeat(arg1, arg2)’ que faz com que este display funcione de maneira flexível de acordo o numero de elementos que queremos inserir [arg1] e o numerod e fracos que se deseja [arg2]. Também é liberado o collum-gap, que permite criar lacunas entre estes elementos.

Interação entre página e HTML [02:46:06]:

Existem várias maneiras de criar funções para interagir com elementos da página, a maneira mostrada a seguir cria uma função para executar uma acao no mapa quando se clica na imagem do orfanato selecionado.

No arquivo orphanage.js, cria-se a função:

function selectImage (event) {

    console.log('funcionou')

}

E no arquivo html, no botão, adiciona-se:

onclick="selectImage(event)"

ficanod assim:

<button onclick="selectImage(event)" class="active" type="button">AAA</button>

Então toda vez que esse botão for clicado, ele ativara a função ‘selecImage()’ enviando o valor ‘event’; no caso a única coisa que ela faz é logar no console ‘funcinou’, porem diversas outras funções podem ser programadas.

Dia 14/10 – Aula 03:

Nesta aula continua-se com a construção do site, agora com a implementação da página para adição de orfanatos, utilizando formulários.

Alguns elementos desta página são idênticos à elementos já criados, por isso basta copiar as partes do código referente a estes no novo arquivo HTML.

Aqui cabe uma observação importante com relação a organização do código, sempre que for utilizar algo igual é interessante separar este snip de código em algum arquivo, trecho a parte para que possa ser utilizado depois posteriormente, desta maneira além de economizarmos tempo economizamos linhas de código, facilita a leitura do mesmo e o deixa menos redundante; aqui esta técnica está presente no ‘sidebar’, já que se tinha o mesmo sidebar para duas paginas diferentes, foi conveniente separar o trecho de código deste no arquivo sidebar.css e importar este nas duas paginas que o usavam, orphanage.html e create-orphanage.html.

/\* Sidebar \*/

aside.sidebar {

    position: fixed;

    height: 100%;

    padding: 3.2rem 2.4rem;

    background: #15c3d6;

    display: flex;

    flex-direction: column;

    justify-content: space-between;

    align-items: center;

}

aside.sidebar footer button {

    width: 48px;

    height: 48px;

    border: none;

    background: #12afcb;

    border-radius: 1.6rem;

    cursor: pointer;

    display: flex;

    align-items: center;

    justify-content: center;

    transition: background-color 200ms;

}

aside.sidebar footer button:hover {

    background: #17d6eb;

}

.with-sidebar {

    padding-left: 110px;

}

Para se trabalhar com formulários, basta adicionar as tags referentes no documento html e formatar as características desejadas via css; este é sempre o modus operandi de montagem de páginas web, primeiro monta-se o layout, coloca os elementos existentes no arquivo HTML, smepre separados corretamente, e então formata-se no css.

Mapa [00:59:49]:

Agora no Javascript escrevemos um função curta para poder adicionar o marcador de local ao mapa:

// create and add marker

mymap.on("click", (event) => {

  const lat = event.latlng.lat;

  const lng = event.latlng.lng;

  // remove icon from map

  marker && mymap.removeLayer(marker); //se marker = True (existir) então faça mymap.removeLayer(marker)

  // add icon layer

  marker = L.marker([lat, lng], { icon })

  .addTo(mymap);

});

Agora, percebe-se que ao se enviar o formulário, na barra de navegação não aparecem os dados referentes a localização do mapa, para poder tratar disso devemos inserir um input do tipo ‘hidden’ no HTML que recebe o valor ‘lat’ e ‘lng’ do marker setado no JS.

// create and add marker

mymap.on("click", (event) => {

    const lat = event.latlng.lat;

    const lng = event.latlng.lng;

    // remove icon from map

    marker && mymap.removeLayer(marker); //se marker = True (existir) então faça mymap.removeLayer(marker)

    // add icon layer

    marker = L.marker([lat, lng], { icon })

        .addTo(mymap);

    // gets marker lat and lng value to HTML doc

    document.querySelector('[name=lat]').value = lat;

    document.querySelector('[name=lng]').value = lng;

});

Fotos [01:17:00]:

Feito isso agora resta trabalhar o campo de adicionar fotos, para isso utilizaremos também o JS.

// add photo field

function addPhotoField() {

    //pegar o container de gotos #images

    const container = document.querySelector('#images');

    //pegar o container para duplicar .new-upload

    const fieldsContainer = document.querySelectorAll('.new-upload');

    //realizar o clone da ultima imagem adicionadas

    const newFieldContainer = fieldsContainer[fieldsContainer.length - 1]

        .cloneNode(true);

    //verificar se o campo está vazio, se sim não adicionar ao container de imagens

    const input = newFieldContainer.children[0];

    // console.log(input);

    if (input.value == '') {

        return

    };

    //limpeza do campo

    input.value = '';

    //adicionar o clone ao container de #images

    container.appendChild(newFieldContainer);

}

Agora para adicionar a funcionalidade de remover os campos das fotos a ideia é basicamente a mesma.

Perceba que sempre quando mexemos no documento HTML via JS a ideia é trabalhar com Nodes pais e filhos e a ideia de index dentro destes elementos, para isso agora fazemos com que ao clicar no botão no documento HTML ele ative a função ‘deleteField(event)’ já com seu argumento:

<span onclick="deleteField(event)">

E então na função usamos o querySelectorAll(‘.new-upload’) para selecionar o campo desejado dentro do documento HTML e então nele removemos o nodo pai, e caso só exista 1 elemento dentro deste campo seu conteúdo deve ser limpo (‘’):

//remove photo field

function deleteField(event) {

    const span = event.currentTarget;

    const fieldsContainer = document.querySelectorAll('.new-upload');

    if (fieldsContainer.length < 2) {

        //limpar o valor do campo

        span.parentNode.children[0].value = '';

        return

    }

    //deletar o campo

    span.parentNode.remove()

}

Freezando: a lógica é bem simples, conssite em acessar um elemento, acessar seus nodos pais e/ou filhos para remover/modificar estes campos de alguma maneira. Basta pensar no HTML como uma série de campos que são selecionados através do querySelector e dentro deste listas com valores a serem acessado através de índices...

Dia 15/10 – Aula 04:

Aqui é começado a trabalhar o back-end da aplicação e com isso alguns conceitos são inseridos, dentre eles, o de regra de negócio, que é basicamente as acoes pelas quais o back-end é responsável. Por exemplo, a validação de um cadastro é de total responsabilidade do back-end, por isto se trata de uma regra de negocio.

O back-end é responsável por:

* Receber os pedidos do cliente;
* Preparar os pedidos (regra de ngócio);
* Enviar respostas;
* Lidar com dados da aplicação;

Em suma, o fron-end apresenta e o back-end processa.

Para trabalhar com o back-end é necessário instalar:

* Noje.js;
* Git Bash;

Após instalados, devemos, via terminal bash, executar o seguinte comando dentro da pasta nlw:

* npm init -y

Desta forma um arquivo .json será colocado na pasta c/nlw. Agora instalaremos as bibliotecas necessárias para utilizar o server:

* npm install express

Após instalados estes pacotes é possível iniciar o servidor, no entanto, toda vez que for feita uma atualização no código seria necessário fechar o servidor e abri-lo novamente mas para automatizar este processo e facilitar este cmainho, podemos instalar o pacote nodemon:

* npm install nodemon

Então se faz necessário modificiar o arquivo .json inicial para ficar desta maneira:

{

  "name": "nlw",

  "version": "1.0.0",

  "description": "",

  "main": "index.js",

  "scripts": {

    "start": "nodemon src/server.js"

  },

  "keywords": [],

  "author": "",

  "license": "ISC",

  "dependencies": {

    "express": "^4.17.1",

    "nodemon": "^2.0.6"

  }

}

Dia 14/10 – Aula 03:

Banco de dados:

A tecnologia utilizada será um banco de dados baseado em SQL chamado SQLite.