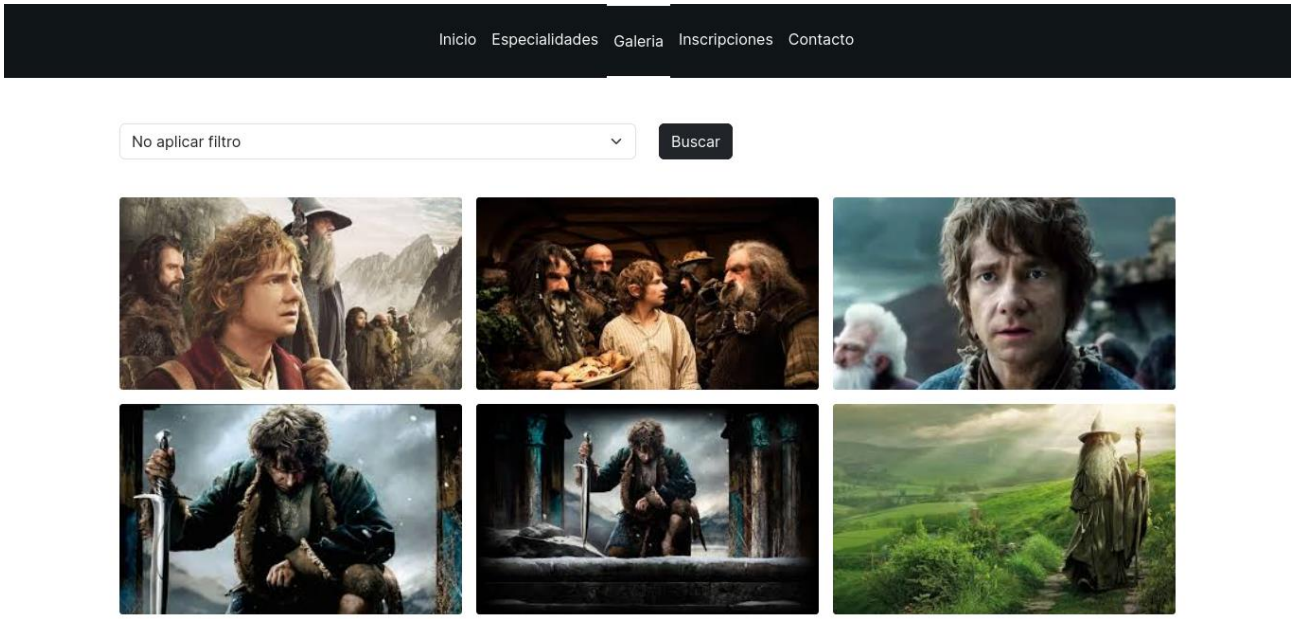
GALERÍA



ARCHIVOS Y ESTRUCTURACIÓN

En este módulo aprenderemos cómo funciona el código de la sección `Galería`. Primero veremos todos los archivos involucrados en esta sección:

/src/app/galeria
/galeria.css
/page.jsx

Dentro de la carpeta `/src/app/galeria` nos encontramos con el archivo `page.jsx` y el archivo `galeria.css`. Son los archivos básicos de la página:

- `page.jsx`: Se encarga de renderizar la interfaz de la página '/galeria'.
- `galeria.css`: Contiene todos los estilos utilizados en la interfaz de la página '/galeria', incluyendo los estilos de los componentes utilizados (`ImagenGaleria`).

/src/assets/static/Galeria/imagenes.js

Dentro de la carpeta `/src/assets/static/Galeria` nos encontramos con el archivo `imagenes.js`. Este archivo contiene la información de las imágenes. Si queremos agregar, modificar o eliminar imágenes, lo hacemos desde este archivo.

/src/assets/static/especialidades/especialidades.js

Dentro de la carpeta `/src/assets/static/especialidades` nos encontramos con el archivo `especialidades.js`. Este archivo contiene las especialidades existentes. Si en un futuro se agregan nuevas especialidades o se quitan las ya existentes, se podrán modificar desde este archivo. Por eso decidimos utilizar este archivo, para que en caso de que ocurra lo mencionado anteriormente, el filtro por especialidades pueda actualizarse de manera automática.

/src/components/ImagenGaleria/index.jsx

Dentro de la carpeta `/src/components/ImagenGaleria` nos encontramos con el archivo `index.jsx`. Se llama así para acortar la ruta a la hora de importarlo (`src/components/ImagenGaleria`). Dentro de este archivo encontraremos un código muy simple que renderiza un contenedor grid con la imagen proporcionada por medio de las `props`.

/public/Galeria/image-[id].jpg

Dentro de la carpeta `/public/Galeria` nos encontramos con todas las imágenes que se muestran en la página '/galeria'. Las imágenes deben estar en formato '.jpg' y deben llevar el nombre image-[id].jpg (ej: image-55.jpg).

ARCHIVO PRINCIPAL: PAGE.JSX

En esta sección examinaremos y comprenderemos el código principal de nuestra página '/galeria', para poder realizar modificaciones a futuro, o simplemente comprender la estructura del código.

A. Imports

```
"use client";
import { useState, useRef } from 'react';
import { IMAGENES } from '@/assets/static/Galeria/imagenes.js';
import { ESPECIALIDADES } from '@/assets/static/especialidades/especialidades.js';
import ImagenGaleria from '@/components/ImagenGaleria';
import './galeria.css';
```

1. Utilizamos la sentencia `"use client"` que el framework Next.js nos proporciona para que la página '/galeria' sea renderizada del lado del cliente. Hacemos esto para poder utilizar elementos a los que solamente se pueden acceder del lado del cliente, como `useState` y `useRef`.
2. Importamos `useState` para poder controlar el estado `filter` que utilizaremos para filtrar las imágenes por especialidad, y con `useRef` referenciamos el elemento `select` para poder obtener el valor de la opción seleccionado por el usuario.

3. Importamos la variable `IMAGENES` para obtener la información de todas las imágenes que queramos mostrar en la galería.
4. Importamos la variable `ESPECIALIDADES` para obtener la información de todas las especialidades que existan en la escuela para utilizarlas luego a la hora de mostrar las opciones en el elemento `select`.
5. Importamos el componente `ImagenGaleria` que utilizaremos para renderizar la lista de imágenes. Este componente contiene las clases necesarias para que se apliquen los estilos escritos en el archivo `galeria.css`.
6. Finalmente importamos el archivo `galeria.css` para que se aplique la hoja de estilo a la página.

B. Estructura

- Vamos a quitar todo lo que estorbe nuestro entendimiento para concentrarnos en la base del componente `GaleriaPage`.

```
export default function GaleriaPage() {
  return <div className="container">
      <form>
        <div className="row mt-5">
          <div className="col-8 col-sm-6">
            {/* SELECT | FILTRO */}
          </div>
          <div className="col-2">
            {/* BUTTON | FILTRO */}
          </div>
        </div>
      </form>

      <div className="row">
        <div className="grid-contenedor col-12">
          {/* RENDERIZADO DE IMAGENES */}
        </div>
      </div>
    </div>;
}
```

- El componente utiliza la grilla de Bootstrap y divide el contenedor en dos filas: La primera fila contiene el filtro, que selecciona las imágenes de una especialidad en específico; La segunda fila contiene las imágenes que se muestran en la página.
- La primera fila se divide en dos columnas: La primera que va a tener el elemento `select`; y la segunda que va a tener el `button` para confirmar nuestra selección. Esta fila está encerrada por un elemento `form` que utilizaremos más adelante para hacer funcionar nuestro filtrado de imágenes.

- La segunda fila contiene una columna, en la cual se realizará el renderizado de las imágenes. La clase “grid-contenedor” convierte nuestro contenedor en una grilla, para controlar la manera en que actúan las imágenes en relación con el tamaño de pantalla (Ver `galeria.css` Linea 1-27).

C. Renderizado de imágenes

- Las imágenes están localizadas en la carpeta `/public/Galeria`, y la información está en el archivo `/src/assets/galeria/imagenes.js`:

```
export const IMAGENES = [
  {
    id: 0,
    src: "/Galeria/image-1.jpg",
    alt: "Imagen de prueba!",
    especialidad: "informatica",
  },
]
```

- En la variable `IMAGENES` vamos a encontrarnos una lista que contiene objetos. Cada objeto es una imagen que posee cuatro propiedades: `id`, `src`, `alt` y `especialidad`. La variable `especialidad` la veremos más adelante cuando veamos el funcionamiento del filtro.

```
      <div className="row">
        <div className="grid-contenedor col-12">
          {IMAGENES.map(({id, src, alt}) => (
            <ImagenGaleria key={id} src={src} alt={alt} />
          ))}
        </div>
      </div>
```

- Utilizamos el renderizado de listas de React para automatizar la tarea y no tener que escribir una por una, y cada vez que agregamos una nueva modificar el código. Esta herramienta que React nos brinda nos facilita esta tarea.
- Nota que la función flecha está escrita de la siguiente manera: `() => **()**`, utilizamos paréntesis en vez de corchetes ({}), porque queremos renderizar lo que está dentro. Por esa razón no hace falta agregar un `return`.
- Lo que hacemos dentro de los parámetros de la función flecha es desestructurar el elemento en sus propiedades ({id, src, alt}), es lo mismo que hagamos: `(item) => (<ImagenGaleria key={item.id} />)`.
- Al componente `ImagenGaleria` le pasamos un atributo `key` que React nos pide para identificar al elemento.

- Básicamente estamos mapeando un array que renderiza el componente `ImagenGaleria` según la longitud de la lista. Si no comprendes lo que es un mapeo, te recomiendo este material: Array.prototype.map.

D. Filtrado de imágenes

- La primera fila contiene el elemento HTML que ayuda al usuario a filtrar las imágenes. Recordemos que esta fila está dentro de un formulario, que al ser enviado se ejecuta el evento `onSubmit`.
- Las opciones del select son las especialidades que existen dentro del archivo `especialidades.js`:

```
          <select className="form-control form-select" ref={selectRef}>
            <option default value="null">No aplicar filtro</option>
            {ESPECIALIDADES.map(({id, nombre, value}) => (
              <option key={id} value={value}>{nombre}</option>
            ))}
          </select>
```

- Tenemos la opción por defecto con un valor “null” (type: string), y luego un renderizado de lista, en la que renderizamos el elemento `option` con la especialidad existente.
- Tenemos la propiedad value y nombre, es necesario para que el valor del option no contenga carácteres en mayúsculas o acentos, pero en la UI esté bien escrito.
- Además agregamos una referencia al elemento `select` con la propiedad `ref` para poder tener una referencia de ese elemento y poder acceder al valor seleccionado cuando ejecutemos el manejador de eventos que nos proporciona React.

```
export default function GaleriaPage() {
  const [filter, setFilter] = useState(null);
  const selectRef = useRef();

  const changeFilter = (event) => {
    event.preventDefault();
    const newFilter = (selectRef.current.value === "null") ? null : selectRef.current.value;
    setFilter(newFilter);
  }

  return <div className="container">
      <form onSubmit={changeFilter}>
        {/* Code... */}
      </form>
    </div>;
}
```

- El formulario al enviarse ejecuta la función `changeFilter` que lo que hace esta función es detener la recarga de la página con `event.preventDefault()`, luego asigna a la variable `newFilter` el valor de la opción seleccionada en el elemento `select`, el cual obtenemos

por medio de la referencia que creamos con `useRef` que almacenamos en la variable `selectRef` para luego asignarla al elemento `select` por medio del atributo `ref`. (Te recomiendo para este punto ya haber visto la documentación de React sobre useRef).

- Tip: Nota que cuando llamamos `selectRef` le agregamos el `.current`, porque `selectRef` es un objeto, y la propiedad current nos da el elemento referenciado más reciente.
- La lógica dentro de la variable `newFilter` utiliza un operador ternario para validar si se ha seleccionado una opción o no. De esa manera el algoritmo es capaz de validar si el usuario desea filtrar las imágenes o apretó el botón por error.
- Ahora viene la magia, una vez que pasamos por lo anterior, el manejador de eventos ejecuta la función `setFilter` que el hook de estado nos proporciona para cambiar el estado `filter` y le avisa al componente que hubo cambios internos para que los chequee y renderice los cambios (funcionamiento básico de un componente de React) y aplique el filtro:

```
{IMAGENES.filter(
  ({ especialidad }) => filter === null || especialidad === filter
).map(({id, src, alt}) => (
  <ImagenGaleria key={id} src={src} alt={alt} />
))}
```

- Antes de mapear la lista, la filtramos y mapeamos solo los elementos que cumplan con la condición impuesta en el filter. La condición es que si el filtro es nulo, todos los elementos se rendericen. En caso de que exista un filtro, esta opción se vuelve `false`, y debido al operador lógico OR (||) se evalúa la condición de que el filtro coincida con la especialidad y si coincide lo renderiza.