El futuro digital es de todos

MinTIC

UNIVERSIDAD DEL NORTE

Vigilada Mineducación

CICLO IV:

Desarrollo de Aplicaciones Web

MisionTIC2022

El futuro digital es de todos | MinTIC

UNIVERSIDAD DEL NORTE
Vigilada Mineducación

Sesión 09:

Desarrollo de Aplicaciones Web

Desarrollo de Front-End web con React - Consumo de servicios RESTful

MisiónTIC2022

Objetivos de la sesión

Al finalizar esta sesión estarás en capacidad de:

1. Consumir servicios desde una aplicación web diseñada con React de una API RESTful.

React - Consumo de Servicios RESTful

- Es posible que necesitemos acceder a algún tipo de bases de datos, ya sea nuestra o de algún otro servicio con al que tengamos acceso.
- Por lo general este tipo de integraciones se realizacion por medio de servicios REST via peticiones HTTP.
- Para ello consideraremos hacer una web app a modo de demostración.
    - Consideremos una aplicación web que liste todos los pokémon, por páginas, con su respectiva imagen.
    - Tome en cuenta la API: PokéAPI.
    - Estilos con Bootstrap.

React - Consumo de Servicios RESTful

- Primero que todo analicemos los siguientes endpoint de PokéAPI:
  - Listar: https://pokeapi.co/api/v2/pokemon
  - Detalle: https://pokeapi.co/api/v2/pokemon/1
- Para la solicitud de listado consideremos los siguientes query params:
  - **`offset`**: Agrega un desfase de N elementos al resultado.
  - **`limit`**: Agrega un límite de elementos consultados al resultado.
- Estos parámetros suelen ser conocidos como paginación, ya que nos permiten segmentar la consulta en diferentes páginas o grupos de contenido.

React - Consumo de Servicios RESTful

- Analicemos la consulta de listado sin los query params mediante el gestor de peticiones REST, insomnia:


```
{
  "count": 1118,
  "next": "https://pokeapi.co/api/v2/pokemon?offset=20&limit=20",
  "previous": null,
  "results": [
    {
      "name": "bulbasaur",
      "url": "https://pokeapi.co/api/v2/pokemon/1/"
    },
    {
      "name": "ivysaur",
      "url": "https://pokeapi.co/api/v2/pokemon/2/"
    },
    {
      "name": "venusaur",
      "url": "https://pokeapi.co/api/v2/pokemon/3/"
    },
    {
      "name": "charmander",
      "url": "https://pokeapi.co/api/v2/pokemon/4/"
    },
    {
      "name": "charmeleon",
      "url": "https://pokeapi.co/api/v2/pokemon/5/"
    },
    {
      "name": "charizard",
      "url": "https://pokeapi.co/api/v2/pokemon/6/"
    },
```



```
{
  "count": 1118,
  "next": "https://pokeapi.co/api/v2/pokemon?offset=5&limit=5",
  "previous": null,
  "results": [
    {
      "name": "bulbasaur",
      "url": "https://pokeapi.co/api/v2/pokemon/1/"
    },
    {
      "name": "ivysaur",
      "url": "https://pokeapi.co/api/v2/pokemon/2/"
    },
    {
      "name": "venusaur",
      "url": "https://pokeapi.co/api/v2/pokemon/3/"
    },
    {
      "name": "charmander",
      "url": "https://pokeapi.co/api/v2/pokemon/4/"
    },
    {
      "name": "charmeleon",
      "url": "https://pokeapi.co/api/v2/pokemon/5/"
    }
  ]
}
```


- Izquierda: Sin query params.
- Derecha: Con query params.
  - **`offset`**: 5
  - **`limit`**: 5

React - Consumo de Servicios RESTful

- En la respuesta de nuestra solicitud observamos que no importa si se especifican o no los query params de paginación.
- Esto lo podemos determinar porque nuestra respuesta incluye las siguientes propiedades:
  - **`count`**: Total de elementos agrupando todas las páginas.
  - **`next`**: Siguiente página de la paginación.
  - **previous**: Página anterior de la paginación.
- Por lo general los listados de paginación suelen incluir estas propiedades, también pueden ser incluidas o expuestas otras como lo son:
  - **`page`**: Página actual o consultada.
  - **`elements`**: Cantidad de elementos por página.
  - **`totalPages`**: Total de páginas disponibles.

React - Consumo de Servicios RESTful

- Analizando nuestra solicitud de detalle de Pokémon, notamos lo siguiente:

- Nos trae mucha información relacionada con el pokémon.
- El vector de `sprites` contiene una propiedad llamada `front_default` la cual contiene la ruta de una imagen.

React - Consumo de Servicios RESTful

- Detalle de la propiedad `sprites`:


```
"sprites": {
  "back_default": "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/back/1.png",
  "back_female": null,
  "back_shiny": "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/back/shiny/1.png",
  "back_shiny_female": null,
  "front_default": "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/1.png",
  "front_female": null,
  "front_shiny": "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/shiny/1.png",
  "front_shiny_female": null,
  "other": { ↔ 2 ↦ },
  "versions": { ↔ 8 ↦ }
},
```

React - Consumo de Servicios RESTful

- Para poder realizar estas solicitudes a través de nuestra web app estaremos introduciendo a la fetch API.
- Para ello trabajaremos sobre StackBlitz.
- Inicialmente, crearemos un archivo de configuración `config.json` sobre la carpeta `src/`. Esto se considera como archivo de variables de entorno y es una buena práctica:

```
{
  "POKE_API": {
    "HOST": "pokeapi.co",
    "VERSION": "2"
  }
}
```

React - Consumo de Servicios RESTful

- Luego, crearemos un archivo para nuestra api llamado `poke-api.fetch.js` bajo la carpeta **`src/api/`**:

```
import CONFIG from '../config.json';

const getURL = (resource = null) =>
  new URL(`https://${CONFIG.POKE_API.HOST}/api/v${CONFIG.POKE_API.VERSION}/${resource || ''}`);

export const list = async (params = { offset: 0, limit: 10 }) => {
    const resource = getURL('pokemon');
    resource.search = new URLSearchParams(params).toString();
    const results = await fetch(resource);
    return await results.json();
};

export const detail = async (resource) => {
    const results = await fetch(resource);
    return await results.json();
};
```

React - Consumo de Servicios RESTful

- Como podemos ver, fetch API por defecto envia solicitudes con el método **`HTTP GET`**.
- Adicionalmente, definimos nuestros métodos para consultar el listado y el detalle de pokemon.
- Fetch es un evento asíncrono motivo por el cual definimos nuestros métodos de listar y detalle como async, y realizamos fetch con el keyword **`await`** antes de llamar al método.
- Fetch inicialmente nos entrega un buffer de lectura con respecto a nuestra solicitud, por lo que para poder leerlo de forma correcta como un objeto de JS, es decir un JSON tenemos que ejecutar el método `.json()` propio de la clase Response.
- Ya que el método **`.json()`** también retorna una Promise o promesa debemos usar el keyword **`await`**.

React - Consumo de Servicios RESTful

- Para seguir con el proyecto, instalaremos Bootstrap ya que no configuraremos ninguna clase de estilos con css.
- Para esto seguiremos la guía de instalación de la documentación de Bootstrap.
- Una vez instalado añadiremos los siguientes imports a nuestro archivo `src/index.js`:

```
import 'bootstrap';
import 'bootstrap/dist/css/bootstrap.min.css';
```

- Con esto ya deberíamos tener Bootstrap correctamente instalado y configurado en nuestro proyecto.

React - Consumo de Servicios RESTful

- Inicialmente definiremos los siguientes componentes:
  - `<PaginationHelper />`.
  - `<Dropdown />`.
  - `<Button />`.
  - `<PokemonList />`.
  - `<PokemonItem />`.
  - `<GrowingSpinner />`.
- Seguidamente en nuestro archivo **`src/App.js`** reemplazamos nuestro return por lo siguiente:

```
<div className="container">
  <div className="row">
    <PaginationHelper listProvider={PokemonList} />
  </div>
</div>
```

React - Consumo de Servicios RESTful

- Para nuestro <PaginationHelper />, donde ubicamos toda la lógica de paginar nuestra solicitud de listado. Ubicado en src/components/PaginationHelper/PaginationHelper.js. Agregaremos los siguientes estados y efectos:

```
const [page, setPage] = useState(1);
const [elements, setElements] = useState(5);
const [currentPage, setCurrentPage] = useState(null);
const maxPage = Math.ceil(currentPage?.count / elements);
useEffect(() => {
  const fetch = async () => {
    const payload = await PokeAPI_with_fetch.list({
      offset: (page - 1) * elements,
      limit: elements,
    });
    setCurrentPage(payload);
  };
  fetch();
}, [page, elements]);
```

React - Consumo de Servicios RESTful

- Inicialmente configuraremos nuestra página y nuestros elementos por página para que tengan el valor de 1 y 5 respectivamente.
- Adicionalmente, controlaremos la página actual con un valor inicial de nulo puesto que no contamos inicialmente con esta información hasta que la solicitud sea realizada.
- Tomando en cuenta los Life-cycle Hooks de un componente en React, la forma apropiada de realizar o llamar eventos asíncronos es dentro del Hook **`useEffect`**.
- Por lo que, definimos nuestro método fetch para hacer la consulta dentro del mismo y lo ejecutamos sin el await ya que no podemos detener la ejecución de nuestro sitio web solo para hacer la consulta.
- Finalmente, tenemos de dependencia a nuestros estados, page y elements.

React - Consumo de Servicios RESTful

- De igual forma definimos los siguientes métodos:

```
const dropdownSelectHandler = (newSelection) => {
  if (page != 1) {
    const approximatedCount = page * elements;
    const count = Math.min(approximatedCount, currentPage?.count);
    const bookmark = Math.ceil(count / newSelection);
    setPage(bookmark);
  }
  setElements(newSelection);
};
```

- Este metodo sera el controlador o handler del evento `onSelect` que definiremos para nuestro `<Dropdown />`.
- Como podemos ver usaremos este dropdown para controlar la cantidad de elementos que tendremos por página.

React - Consumo de Servicios RESTful

- De igual forma definimos los siguientes métodos:

```
const generatePaginationActionButtons = () => {
  let focus = [false, true, false];
  let indexes = [page - 1, page, page + 1];
  if (page == 1) {
    indexes = [1, 2, 3];
    focus = [true, false, false];
  }
  if (page === maxPage) {
    indexes = [maxPage - 2, maxPage - 1, maxPage];
    focus = [false, false, true];
  }
  const buttons = [0, 1, 2].map((i) => (
    <Button key={`pmb-${i}`} withoutBorders={i %2==0} isFocused={focus[i]} onClick={() =>
setPage(indexes[i])}>
      {indexes[i]}
    </Button>
  ));
  return buttons;
};
```

React - Consumo de Servicios RESTful

```
const renderPaginationActions = () => {
  const [first, second, third] = generatePaginationActionButtons();
  return (
    <>
      <div className="col g-0">
        <Button key="pb-0" borderRadius="5px 0px 0px 5px" onClick={() => setPage(1)} >
          &#8810;
        </Button>
      </div>
      <div className="col g-0">{first}</div>
      <div className="col g-0">{second}</div>
      <div className="col g-0">{third}</div>
      <div className="col g-0">
        <Button key="pb-1" borderRadius="0px 5px 5px 0px" onClick={() => setPage(maxPage)}>
          &#8811;
        </Button>
      </div>
    </>
  );
};
```

- Estos métodos generan los botones de nuestra paginación.

React - Consumo de Servicios RESTful

- Ya que nuestro `<PaginationHelper />` depende de los componentes `<Dropdown />`, `<Button />` y `<PokemonList />` (con el alias **`<ListProvider />`**).
- En `<Dropdown />`, que se encuentra en `src/components/Dropdown/Dropdown.js`, encontramos que:

```
const Dropdown = ({ options, onSelect, children }) => {
  const classes = "btn btn-outline-primary dropdown-toggle";
  const renderOptions = () => options.map((op) => (
      <li key={`option-${op}`} onClick={() => onSelect(op)}> <a className="dropdown-item" href="#"> {op} </a> </li>
    ));
  return (
    <div className="dropdown">
      <button className={classes} type="button" id="dropdownMenuButton" data-bs-toggle="dropdown" aria-expanded="false" >
        {children}
      </button>
      <ul className="dropdown-menu" aria-labelledby="dropdownMenuButton">
        {renderOptions()}
      </ul>
    </div>
  );
};
```

- Componente adaptado de Bootstrap.

React - Consumo de Servicios RESTful

- En **<Button />**, que se encuentra en **src/components/Button/Button.js**, encontramos que:

```
const Button = ({ isFocused, withoutBorders, borderRadius = null, onClick, children }) => {
  const decorator = isFocused ? 'primary' : 'outline-secondary';
  const borderDecorator = withoutBorders ? `border-end-0  border-start-0` : ``;

  return (
    <button
      className={`btn btn-${decorator} w-100 ${borderDecorator}`}
      onClick={onClick}
      style={{ borderRadius: borderRadius || '0px' }}
    >
      {children}
    </button>
  );
};
```

- Este componente lo usamos más que todo para los botones de la barra inferior de nuestra paginación.

React - Consumo de Servicios RESTful

- En **<PokemonList />**, que se encuentra en **src/components/Pokemon/PokemonList.js**, encontramos que:

```
const PokemonList = ({ items = [] }) => {
  if (!Boolean(items) || items.length === 0) return <p>No hay Pokemon.</p>

  const renderItems = () =>
    items.map((pokemon) => <PokemonItem key={pokemon.name} pokemon={pokemon} />));

  return <ul className="list-group">{renderItems()}</ul>;
};
```

- Este componente lo usamos más que todo para listar los Pokémon de una forma sencilla.

Adaptado de Bootstrap.

React - Consumo de Servicios RESTful

- Observamos que **<PokemonList />** depende de **<PokemonItem />**, que se encuentra en **src/components/Pokemon/PokemonItem.js**, definimos los siguientes estados y efectos:

```
const [detail, setDetail] = useState(null);
const [showSpinner, setShowSpinner] = useState(null);
const { url } = pokemon;
useEffect(() => {
  const fetchDetail = async () => {
    setShowSpinner(true);
    const payload = await PokeAPI_with_fetch.detail(url);
    setDetail(payload);
    setShowSpinner(false);
  };
  fetchDetail();
}, [url]);
```

- Esta solicitud, la cual se envía con cada cambio a url, obtiene el detalle del Pokémon y con esto su imagen.

React - Consumo de Servicios RESTful

- Adicionalmente definimos el siguiente método:

```
const renderImage = () => {
  if (showSpinner === null) {
    return;
  } else if (showSpinner) {
    return <GrowingSpinner />;
  } else {
    return (
      <img
        width={96}
        height={96}
        alt={pokemon.name}
        src={detail?.sprites?.front_default}
      />
    );
  }
};
```

- El cual se encarga de mostrar la imagen, un spinner o nada dependiendo del estado del detalle

React - Consumo de Servicios RESTful

- Luego nuestro componente genera el siguiente JSX:

```
return (
  <li className="list-group-item g-0">
    <div className="row g-0 pl-4 align-items-center">
      <div
        className="col-2 col-sm-3 col-lg-2 col-xl-1"
        style={{ height: '96px' }}
      >
        {renderImage()}
      </div>
      <div className="col-10 col-sm-9 col-lg-10 col-xl-11">
        <h2>{pokemon.name}</h2>
      </div>
    </div>
  </li>
);
```

- Aquí simplemente mostramos la imagen y el nombre del Pokémon.

React - Consumo de Servicios RESTful

- Por último hay que resaltar que nuestro componente se exporta de una forma diferente, la cual es la siguiente:

```
export default React.memo(PokemonItem);
```

- Esto se debe a que no queremos que este componente se vuelva a generar o computar con cada cambio de nuestra aplicación.
- Por ejemplo, si no añadiremos React.memo, cuando cambiaramos la cantidad de elementos con nuestro dropdown. Entonces, en todos los casos enviaríamos nuevas peticiones HTTP.
- Esto no es lo que deseamos ya que si cambiamos de una cantidad de elementos mayor a una menor, por ejemplo de 10 a 5, ya contamos con la información de los primeros 5 Pokémon.
- Por lo que no necesitamos volver a consultar esta información.
- Esto es posible a las validaciones internas de React.memo

React - Consumo de Servicios RESTful

- Por último, tenemos a <GrowingSpinner>, ubicado en src/components/Spinner/GrowingSpinner.js:

```
const GrowingSpinner = () => {
  return (
    <div className="spinner-grow text-secondary" role="status" style={{ width: '96px',
height: '96px' }} >
      <span className="visually-hidden">Loading...</span>
    </div>
  );
};
```

- Este componente lo utilizamos para decirle al usuario que las imagenes estan cargado debido a la solicitud de detalle que estamos realizando.
- Componente adaptado de Bootstrap.
- Para ver la aplicación demo en vivo, visitar este enlace.

El futuro digital es de todos

MinTIC

UNIVERSIDAD DEL NORTE

Vigilada Mineducación

Ejercicios de práctica

Mision TIC 2022

El futuro digital es de todos MinTIC

UNIVERSIDAD DEL NORTE
Vigilada Mineducación

Referencias

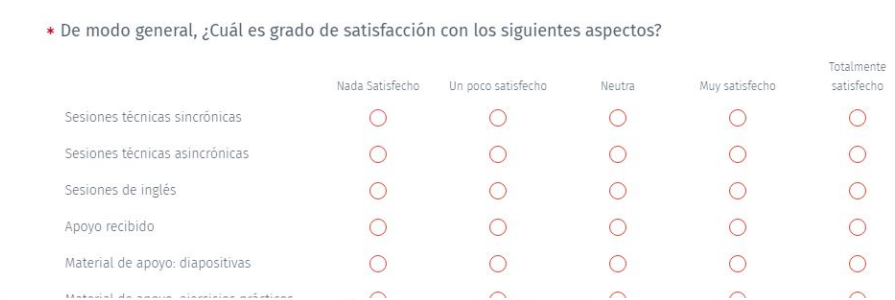
- https://pokeapi.co/docs/v2#pokemon-section
- https://getbootstrap.com/docs/5.1/getting-started/download/#npm
- https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
- https://stackblitz.com/edit/react-h3h7mc?file=package.json
- https://developer.mozilla.org/en-US/docs/Web/API/fetch#return_value
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function
- https://developer.mozilla.org/en-US/docs/Web/API/fetch
- https://developer.mozilla.org/en-US/docs/Web/API/Response/json
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise
- https://reactjs.org/docs/react-api.html#reactmemo

MisionTIC2022

El futuro digital es de todos | MinTIC

UNIVERSIDAD DEL NORTE
Vigilada Mineducación

Seguimiento Habilidades Digitales en Programación



* De modo general, ¿Cuál es grado de satisfacción con los siguientes aspectos?

	Nada Satisfecho	Un poco satisfecho	Neutra	Muy satisfecho	Totalmente satisfecho
Sesiones técnicas sincrónicas	○	○	○	○	○
Sesiones técnicas asincrónicas	○	○	○	○	○
Sesiones de inglés	○	○	○	○	○
Apoyo recibido	○	○	○	○	○
Material de apoyo: diapositivas	○	○	○	○	○

Completa la siguiente encuesta para darnos retroalimentación sobre esta semana ▼▼▼

https://www.questionpro.com/t/ALw8TZlxOJ

MisionTIC2022

El futuro digital es de todos

MinTIC

UNIVERSIDAD DEL NORTE

Vigilada Mineducación

¡GRACIAS

POR SER PARTE DE ESTA EXPERIENCIA DE APRENDIZAJE!

Mision TIC 2022