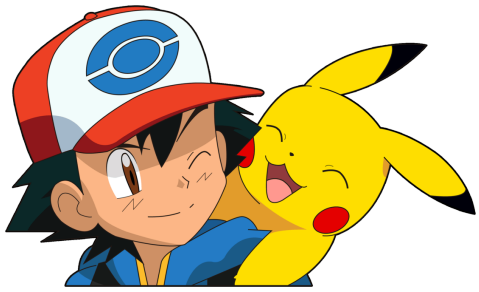
Individual Project - Henry Pokemon



Objetivos del Proyecto

- Construir una App utlizando React, Redux, Node y Sequelize.
- Afirmar y conectar los conceptos aprendidos en la carrera.
- Aprender mejores prácticas.
- Aprender y practicar el workflow de GIT.
- Usar y practicar testing.

Horarios y Fechas

El proyecto tendrá una duración máxima de tres semanas. En el caso de que completan todas las tareas antes de dicho lapso podrán avisar a su Instructor para coordinar una fecha de presentación del trabajo (DEMO).

Comenzando

1. Forkear el repositorio para tener una copia del mismo en sus cuentas
2. Clonar el repositorio en sus computadoras para comenzar a trabajar

Tendrán un `boilerplate` con la estructura general tanto del servidor como de cliente.

IMPORTANTE: Es necesario contar minimamente con la última versión estable de Node y NPM. Asegurarse de contar con ella para poder instalar correctamente las dependecias necesarias para correr el proyecto.

Actualmente las versiónes necesarias son:

- **Node**: 12.18.3 o mayor
- **NPM**: 6.14.16 o mayor

Para verificar que versión tienen instalada:

```
node -v
npm -v
```

ACLARACIÓN: Las dependencias actuales se encuentran en las versiones que venimos trabajando durante el bootcamp.

Versiones:

- **react**: 17.0.1
- **react-dom**: 17.0.1
- **react-router-dom**: 5.2.0
- **redux**: 4.0.5
- **react-redux**: 7.2.3

Está permitido, **bajo su responsabilidad**, actualizar las dependencias a versiones más actuales.

> **IMPORTANTE:** Versiones mas actuales podrían presentar configuraciones diferentes respecto a las versiones en las que venimos trabajando durante el bootcamp.

BoilerPlate

El boilerplate cuenta con dos carpetas: `api` y `client`. En estas carpetas estará el código del back-end y el front-end respectivamente.

En `api` crear un archivo llamado: `.env` que tenga la siguiente forma:

```
DB_USER=usuariodepostgres
DB_PASSWORD=passwordDePostgres
DB_HOST=localhost
```

Reemplazar `usuariodepostgres` y `passwordDePostgres` con tus propias credenciales para conectarte a postgres. Este archivo va ser ignorado en la subida a github, ya que contiene información sensible (las credenciales).

Adicionalmente será necesario que creen desde psql una base de datos llamada `pokemon`

El contenido de `client` fue creado usando: Create React App.

Enunciado

La idea general es crear una aplicación en la cual se puedan ver los distintos Pokemon utilizando la api externa pokeapi y a partir de ella poder, entre otras cosas:

- Buscar pokemons
- Filtrarlos / Ordenarlos
- Crear nuevos pokemons

IMPORTANTE: Para las funcionalidades de filtrado y ordenamiento NO pueden utilizar los endpoints de la API externa que ya devuelven los resultados filtrados u ordenados sino que deben realizarlo ustedes mismos. En particular alguno de los ordenamientos o filtrados debe si o si realizarse desde el frontend.

Únicos Endpoints/Flags que pueden utilizar

- GET https://pokeapi.co/api/v2/pokemon
- GET https://pokeapi.co/api/v2/pokemon/{id}
- GET https://pokeapi.co/api/v2/pokemon/{name}
- GET https://pokeapi.co/api/v2/type

Requerimientos mínimos

A continuación se detallaran los requerimientos mínimos para la aprobación del proyecto individial. Aquellos que deseen agregar más funcionalidades podrán hacerlo. En cuanto al diseño visual no va a haber wireframes ni prototipos prefijados sino que tendrán libertad de hacerlo a su gusto pero tienen que aplicar los conocimientos de estilos vistos en el curso para que quede agradable a la vista.

IMPORTANTE: No se permitirá utilizar librerías externas para aplicar estilos a la aplicación. Tendrán que utilizar CSS con algunas de las opciones que vimos en dicha clase (CSS puro, CSS Modules o Styled Components)

Tecnologías necesarias

- [ ] React
- [ ] Redux
- [ ] Express
- [ ] Sequelize - Postgres

Frontend

Se debe desarrollar una aplicación de React/Redux que contenga las siguientes pantallas/rutas.

Pagina inicial: deben armar una landing page con

- [ ] Alguna imagen de fondo representativa al proyecto
- [ ] Botón para ingresar al home (`Ruta principal`)

Ruta principal: debe contener

- [ ] Input de búsqueda para encontrar pokemons por nombre (La búsqueda será exacta, es decir solo encontrará al pokemon si se coloca el nombre completo)
- [ ] Área donde se verá el listado de pokemons. Al iniciar deberá cargar los primeros resultados obtenidos desde la ruta `GET /pokemons` y deberá mostrar su:
  - Imagen
  - Nombre
  - Tipos (Electrico, Fuego, Agua, etc)
- [ ] Botones/Opciones para filtrar por tipo de pokemon y por pokemon existente o creado por nosotros
- [ ] Botones/Opciones para ordenar tanto ascendentemente como descendentemente los pokemons por orden alfabético y por ataque

- [ ] Paginado para ir buscando y mostrando los siguientes pokemons, 12 pokemons por pagina.

IMPORTANTE: Dentro de la Ruta Principal se deben mostrar tanto los pokemons traidos desde la API como así también las de la base de datos. Por otro lado, si revisan el endpoint que trae todos los pokemons verán que no muestra la información del pokemon sino una URL para hacer un subrequest y obtener los datos de allí. Tendrán que por cada pokemon que van a mostrar hacer otro request a esa URL para obtener su imagen y tipos. Debido a que esto puede hacer que la búsqueda sea muy lenta limitar el resultado total a 40 pokemons totales.

Ruta de detalle de Pokemon: debe contener

- [ ] Los campos mostrados en la ruta principal para cada pokemon (imagen, nombre y tipos)
- [ ] Número de Pokemon (id)
- [ ] Estadísticas (vida, ataque, defensa, velocidad)
- [ ] Altura y peso

Ruta de creación: debe contener

- [ ] Un formulario **controlado con JavaScript** con los campos mencionados en el detalle del Pokemon
- [ ] Posibilidad de seleccionar/agregar más de un tipo de Pokemon
- [ ] Botón/Opción para crear un nuevo Pokemon

> Es requisito que el formulario de creación esté validado con JavaScript y no sólo con validaciones HTML. Pueden agregar las validaciones que consideren. Por ejemplo: Que el nombre del Pokemon no pueda contener caracteres numéricos, que la altura no pueda ser superior a determinado valor, etc.

Base de datos

El modelo de la base de datos deberá tener las siguientes entidades (Aquellas propiedades marcadas con asterísco deben ser obligatorias):

- [ ] Pokemon con las siguientes propiedades:
  - ID (Número de Pokemon) * : No puede ser un ID de un pokemon ya existente en la API pokeapi
  - Nombre *
  - Vida
  - Ataque
  - Defensa
  - Velocidad
  - Altura
  - Peso
- [ ] Tipo con las siguientes propiedades:
  - ID
  - Nombre

La relación entre ambas entidades debe ser de muchos a muchos ya que un pokemon puede pertenecer a más de un tipo y, a su vez, un tipo puede incluir a muchos pokemons.

IMPORTANTE: Pensar como modelar los IDs de los pokemons en la base de datos. Existen distintas formas correctas de hacerlo pero tener en cuenta que cuando hagamos click en alguno, este puede provenir de la API o de la Base de Datos por lo que cuando muestre su detalle no debería haber ambigüedad en cual se debería

mostrar. Por ejemplo si en la API el pokemon `Bulbasaur` tiene id = 1 y en nuestra base de datos creamos un nuevo pokemon `Henry` con id = 1, ver la forma de diferenciarlos cuando querramos acceder al detalle del mismo.

Backend

Se debe desarrollar un servidor en Node/Express con las siguientes rutas:

IMPORTANTE: No está permitido utilizar los filtrados, ordenamientos y paginados brindados por la API externa, todas estas funcionalidades tienen que implementarlas ustedes.

- ☐ **GET /pokemons**:
  - Obtener un listado de los pokemons desde pokeapi.
  - Debe devolver solo los datos necesarios para la ruta principal
- ☐ **GET /pokemons/{idPokemon}**:
  - Obtener el detalle de un pokemon en particular
  - Debe traer solo los datos pedidos en la ruta de detalle de pokemon
  - Tener en cuenta que tiene que funcionar tanto para un id de un pokemon existente en pokeapi o uno creado por ustedes
- ☐ **GET /pokemons?name="..."**:
  - Obtener el pokemon que coincida exactamente con el nombre pasado como query parameter (Puede ser de pokeapi o creado por nosotros)
  - Si no existe ningún pokemon mostrar un mensaje adecuado
- ☐ **POST /pokemons**:
  - Recibe los datos recolectados desde el formulario controlado de la ruta de creación de pokemons por body
  - Crea un pokemon en la base de datos relacionado con sus tipos.
- ☐ **GET /types**:
  - Obtener todos los tipos de pokemons posibles
  - En una primera instancia deberán traerlos desde pokeapi y guardarlos en su propia base de datos y luego ya utilizarlos desde allí

Testing

- ☐ Al menos tener un componente del frontend con sus tests respectivos
- ☐ Al menos tener una ruta del backend con sus tests respectivos
- ☐ Al menos tener un modelo de la base de datos con sus tests respectivos