**EVALUACIÓN MÓDULO 3 - PARTE 1**

**RECURSOS**

Los recursos para la evaluación se encuentran disponibles en la url <https://drive.google.com/drive/folders/18RAaYe5t7x-_d-CufsxRh6PVALW6JG_9?usp=drive_link>

**VALIDACIÓN DEL AMBIENTE**

BASE DE DATOS

1. Abrir pgAdmin4, ingresar con el usuario postgres y la base postgres
2. Hacer un select de confianza en la base de inventarios

ECLIPSE

1. Ingresar al directorio MODULO\_3, abrir eclipse, dejar el workspace configurado.
2. Ejecutar un maven install sobre el proyecto
3. Validar que el proyecto se genera correctamente

TOMCAT

1. En el apache tomcat de MODULO\_3, en la carpeta webapps, colocar el war generado en eclipse.
2. Levantar el tomcat
3. Abrir postman en windows, ejecutar las peticiones de la colección que consumen los web services de inventarios y validar la información contra la base de datos.

REACT NATIVE

1. Descargar y abrir desde Visual Studio Code el proyecto validarAmbiente
2. Por la consola de comandos ejecutar el comando npm install para instalar todos los módulos que requiere el proyecto.
3. Ejecutar el proyecto con npx expo start
4. Abrir el proyecto desde el teléfono, recuerde colocar al teléfono y el computador en la misma red.
5. Modificar un texto en el archivo App.js, guardar los cambios y validar que se muestren en el teléfono.

Una vez que haya comprobado que el ambiente está ok, avisar a uno de nuestros tutores.

**CONSUMO DE REST WEB SERVICES**

Existen varias formas para consumir REST web Services desde JavaScript, una de ellas es con el método **fetch**.

El llamado a fetch es muy peculiar, a continuación describimos con un ejemplo.

La versión más simple de **fetch**, recibe un solo parámetro que es la url del web service que vamos a consumir.

Ejemplos:

Ejemplo 1: invocando a nuestro rest web service

fetch('http://192.168.47.2:8080/inventarios-1.0.0/rest/tiposdocumento/recuperar').then((response) => { return response.json()))

.then((json) => { console.log(json)});

Ejemplo 2: invocando a un rest web service público

fetch('https://jsonplaceholder.typicode.com/todos/').then((response)=> {return response.json()}).then((json) => console.log(json));

**IMPORTANTE:**

Note que a continuación del ***fetch***, se encuentran dos invocaciones a la función ***then***

fetch().then().then()

A diferencia de otras funciones, **fetch** no retorna un valor que podamos capturar en el retorno.

Cuando se ha logrado invocar al Rest Web Service, se invoca a la función **then** que se encuentra junto al fetch.

Podemos observar el siguiente código que recibe como parámetro el primer **then**:

(response) => { return response.json())}

El código corresponde a una función, esta función será invocada una vez que el rest web service responda. Como podemos observar la función recibe un parámetro llamado response, en ese parámetro llegará el objeto de respuesta del llamado al rest web service.

Dentro de la función se invoca al método **json()** que nos retornará el valor de la respuesta formateada en json.

El valor retornado, es pasado a la segunda función **then,** se puede observar que el segundo **then** recibe como parámetro una función.

(json) => {console.log(json)}

Una vez que se ha invocado al rest web service, se invoca al primer then, este ejecuta una función que nos retorna la respuesta en json y con ese valor se invoca al segundo then.

El segundo then dispara una función, esta función recibe como parámetro el json de respuesta y en el cuerpo de la misma podemos hacer cualquier cosa con ese json que ya obtuvimos. En el ejemplo solamente imprimimos en consola el json que nos devolvió el rest web service.

La función fetch siempre se debe invocar con ambos then: **fetch().then().then()**

**EJERCICIO**

**PARTE 1 - Ejecutar el proyecto**

1. Descargar y descomprimir el archivo consumoRestWS.zip
2. Desde la consola de comandos, ingresar al directorio consumoRestWS
3. Ejecutar el comando npm install, para que se instalen todas las dependencias del proyecto.
4. Ejecutar el comando npx expo start, para ejecutar el proyecto en expo.
5. Asegurarse que el teléfono se encuentra en la misma red del computador
6. Desde el teléfono usando EXPO GO, abrir el proyecto.
7. Abrir el proyecto en Visual Studio Code. Modificar un texto del archivo **TestWebServices.js** y guardar, validar que el cambio se refleja en el teléfono.

**PARTE 2 - Consumir el servicio para recuperar posts**

**Datos del servicio:**

**Acción: GET**

**URL:** [**https://jsonplaceholder.typicode.com/posts**](https://jsonplaceholder.typicode.com/posts)

Paso 1:

Probar el servicio desde Postman

Paso 2:

En el archivo **TestServices.js** crear una función llamada getAllPostsService:

const getAllPostsService=()=>{ }

Para que esta función pueda ser consumida desde otros archivos, agregar **export**

export const getAllPostsService=()=>{ }

Dentro de la función, vamos a invocar al servicio usando **fetch**.

export const getAllPostsService=()=>{

fetch().then().then();

}

Como parámetro del fetch, le pasamos el url del rest web service

export const getAllPostsService=()=>{

fetch('https://jsonplaceholder.typicode.com/posts')

.then()

.then();

}

En el primer then, siempre colocamos una función que retorna la respuesta en formato json

export const getAllPostsService=()=>{

fetch('https://jsonplaceholder.typicode.com/posts')

.then((response) => {return response.json()})

.then();

}

Esa respuesta se pasa a la función que se ejecuta en el **segundo then**. Esta será la función que ejecutemos con la respuesta del rest web service. Por ahora solamente imprimimos la respuesta en consola.

export const getAllPostsService=()=>{

fetch('https://jsonplaceholder.typicode.com/posts')

.then((response) => {response.json()})

.then((json) => {console.log(json)});

}

Ahora vamos a invocar esta función desde un botón.

En el archivo TestWebServices crear una función getAllPosts, que será invocada desde un botón, por lo tanto no recibe y no retorna nada.

const getAllPosts=()=>{ }

En el archivo TestWebServices importar TestServices, para poder usar sus funciones, tomar en cuenta el path relativo

import {} from '../services/TestServices'

Importar la función getAllPostsService

import {getAllPostsService} from '../services/TestServices'

En la función getAllPosts, invocar a getAllPostsService

const getAllPosts=()=>{

getAllPostsService();

}

En el evento onPress del botón Recuperar Posts, colocar como parámetro la función getAllPosts

Ejecutar el proyecto, abrir en el celular, presionar el botón y verificar en la consola que trae e imprime la misma información que obtuvimos con postman.

Cuando haya completado este ejercicio, notificar a uno de nuestros tutores

**PARTE 3 - Consumir el servicio para insertar un post**

**Datos del Servicio**

**Acción: POST**

**URL: https://jsonplaceholder.typicode.com/posts**

Ejemplo de objeto que se envía en el post

{

"title":"foo",

"body":"bar",

"userId":1

}

Paso 1:

Probar el servicio desde postman

Luego de ejecutar este servicio que es para insertar un post, si volvemos a ejecutar la consulta, vemos que no se ha insertado nada, al ser un servicio público, dan la estructura del servicio, la respuesta correcta, pero en la realidad no insertan en su base de datos

Paso 2:

En TestServices, crear una función llamada createPostService, que pueda ser consumida desde otro archivo, la función no recibe parámetros, no retorna nada.

Desde esta función se invocará al web service.

**Consideraciones para el fetch:**

Para invocar a una acción diferente a GET, en el fetch debemos pasar un segundo parámetro con un objeto con los siguientes atributos:

**method**: el método http del rest web service

**body**: el objeto que viaja en el body

**headers**: las cabeceras http de la petición

En method, se coloca POST, PUT, PATCH o DELETE, en este caso POST

const config={

method:'POST',

}

En body, el objeto que queremos enviar como parte de la petición, en formato json. Dentro de la aplicación vamos a trabajar con objetos javascript, para convertir un objeto javascript a formato json, usamos el método **JSON.stringify**.

const config={

method:'POST',

body:JSON.stringify({

title: 'mensaje',

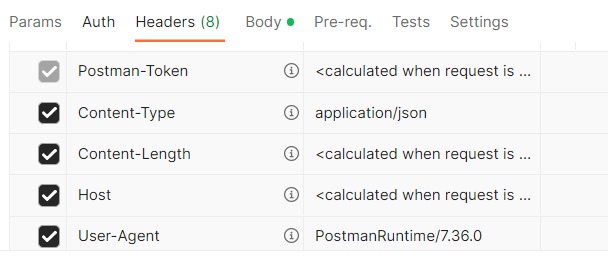
body: 'suerte en su evaluación',

userId: 1,

}),

}

En headers, debemos enviar el header Content Type, indicando que se va a enviar la trama en formato application/json. Hasta ahora, ese valor lo ha estado seteando POSTMAN en la cabecera http. Revisar la sección headers de postman



const config={

method:'POST',

body:JSON.stringify({

title: 'mensaje',

body: 'suerte en su evaluación',

userId: 1,

}),

headers: {

'Content-type': 'application/json',

}

}

Invocación al fetch

export const createPostService=()=>{

const config={

method:'POST',

body:JSON.stringify({

title: 'mensaje',

body: 'suerte en su evaluación',

userId: 1,

}),

headers: {

'Content-type': 'application/json',

}

};

fetch('https://jsonplaceholder.typicode.com/posts',config)

.then((response) => {return response.json()})

.then((json) => {console.log(json)});

}

Importar la función createPostService en TestWebServices.

Crear la función createPost, que invoca a la función createPostService

El onPress del botón Crear Post recibe como parámetro la función createPost

Probar y verificar la respuesta en consola

Cuando haya completado este ejercicio, notificar a uno de nuestros tutores

**PARTE 4 - Consumir el servicio para actualizar un post**

**Acción: PUT**

**URL: https://jsonplaceholder.typicode.com/posts/1**

Ejemplo de objeto que se envía en el post, para actualizar el registro con id 1

{

"id":1,

"title":"mensaje final",

"body":"hasta la vista baby",

"userId":1

}

Probar desde Postman

En TestServices, crear la función updatePostService e invocar al web service

En TestWebServcies crear la función updatePost que invoca a updatePostService, y se ejecuta en el botón Actualizar Post.

Probar y verificar la respuesta en consola

**PARTE 5 - Consumir el servicio para filtrar por id de usuario**

Acción: GET

URL: <https://jsonplaceholder.typicode.com/posts?userId=1>

Probar desde Postman

Crear la función getByUserIdService e invocar al web service

Crear la función getByUserId que invoca a getByUserIdService, invocar desde el botón.

Verificar que trae la respuesta

Cuando haya completado este ejercicio, notificar a uno de nuestros tutores

**PARTE 6 - Desde los botones XXX YYY ZZZ, consumir 3 servicios de web services públicos, puede usar los siguientes:**

| <https://fakestoreapi.com/docs> |
| --- |
| <https://api.chucknorris.io/> |
| <https://pokeapi.co/> |

Debe invocar una llamada a GET, una a POST y una a PUT, de cualquiera de ellos

**Cuando haya completado este ejercicio, notificar a uno de nuestros tutores.**