**EVALUACIÓN MÓDULO 3 - PARTE 2**

**PARTE 1**

**ENVIAR INFORMACIÓN DESDE FORMULARIO**

1. En la carpeta screens del proyecto consumoRestWS, agregar el archivo PostForm.js
2. En el archivo App.js, importar PostForm, para determinar su navegación

import {PostForm} from './screens/PostForm'

1. Agregar un nuevo Screen que permita navegar a PostForm

<StackProducts.Screen name="PostFormView" component={PostForm} options={{ title: 'MENSAJES' }}/>

1. En StackProducts.Navigator, agregar el atributo initialRouteName, para configurar el screen inicial al cual va a navegar

initialRouteName='PostFormView'

1. Ejecutar el proyecto y validar que la pantalla inicial es PostForm
2. Revisar el código de PostForm, tiene dos cajas de texto atadas a dos variables de estado y un botón que cuando se lo presiona ejecuta la función createPost. Por ahora la función solo muestra en consola los valores que toma de las cajas de texto. Ejecutar y probar.
3. Modificar la función createPost, para que luego de imprimir en consola invoque al método createPostService (recuerde primero importar el método). Probar que funcione la invocación

**PARTE 2**

**CAMBIAR createPostService**

Al momento createPostService, llama al servicio, pero siempre manda la misma información del objeto.

Cambiamos la función createPostService, para que reciba un parámetro llamado post, en este parámetro le llegarán los datos que tomamos desde las cajas de texto.

export const createPostService = (post) => {

Cuando armemos este objeto desde la pantalla, tendrá dos atributos, llamados *title* y *body*.

Así cuando vayamos a invocar al rest web service, en lugar de pasarle siempre la misma información como está ahorita, vamos a tomar los valores que llegan en el objeto post:

body: JSON.stringify({

title: post.title,

body: post.body,

userId: 1,

}),

**PARTE 3**

**CAMBIAR la invocación**

En PostForm, en el llamado a createPostService, debemos pasarle el objeto, con los atributos title y body. El valor que toman **title** y **body**, es el valor que ingresó el usuario en las cajas de texto y que lo tenemos en las variables de estado **subject** y **message**

createPostService(

{

title:subject,

body:message

}

);

Validar la invocación hasta este punto.

**PARTE 4**

**MOSTRAR EN PANTALLA**

Luego de que el rest web service haya respondido, vamos a mostrar un Alert en pantalla.

Tomar en cuenta lo siguiente: el Alert lo podemos mostrar en el archivo PostForm, que tiene la parte visual, no podemos usar un Alert en el archivo TestServices, porque ahí no está la parte visual.

Lo que haremos entonces es pasar a **createPostService** un segundo parámetro que va a ser una función, esta función será la que ejecute el Alert. Por tanto **createPostService**, recibe un segundo parámetro llamado **fnExito** que es una función.

export const createPostService = (post, fnExito) => {

Cuando el web service haya respondido, y tengamos el contenido de la respuesta, es decir en el **segundo then,** luego de imprimir en consola ejecutamos la función fnExito

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

Finalmente en la invocación a createPostService, pasamos la función como segundo parámetro. Tomar en cuenta que es una función que no recibe nada. La función simplemente muestra un Alert.

createPostService(

{

title:subject,

body:message

},

()=>{

Alert.alert("CONFIRMACION","Se ha ingresado un nuevo POST");

}

);

De esta forma cuando el rest web service responda exitoso, se va a ejecutar esta función que tiene acceso a los elementos de pantalla.

Validar el funcionamiento

**PARTE 5**

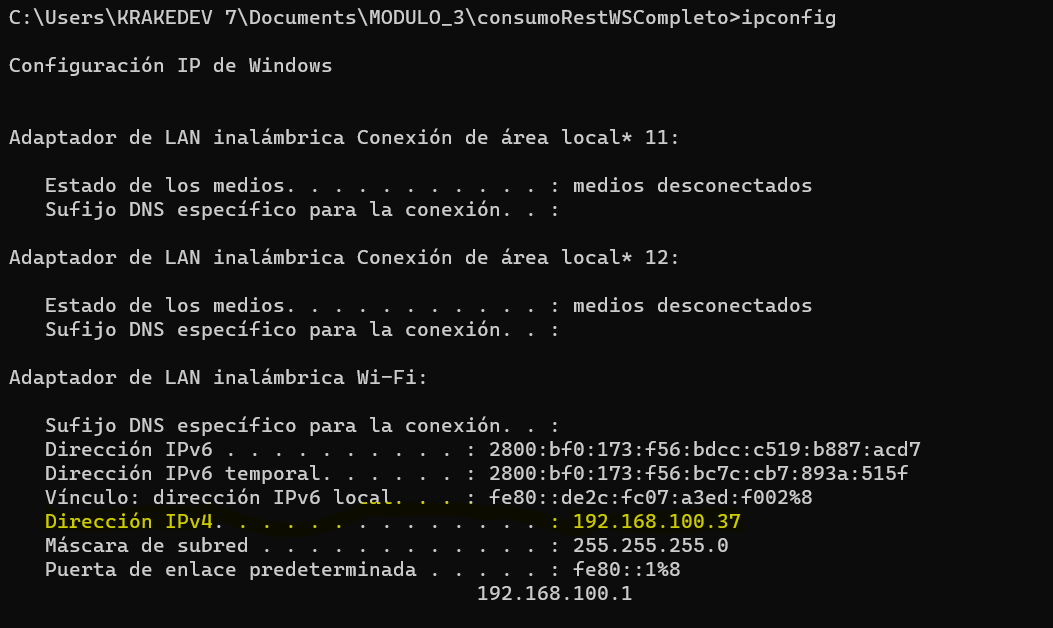
**CONSUMIR UN REST WEB SERVICE PUBLICADO POR NOSOTROS EN TOMCAT**

Levantar el tomcat y probar desde postman el consumo del rest web service:

URL: <http://localhost:8080/inventarios-1.0.0/rest/tiposdocumento/recuperar>

ACCION: GET

En la URL, vamos a reemplazar localhost, por el IP del computador. Para obtener el IP, en una consola de comandos se puede ejecutar el comando ipconfig



De esta forma, en mi caso, la nueva URL sería

[http://192.168.100.37:8080/inventarios-1.0.0/rest/tiposdocumento/recuperar](http://localhost:8080/inventarios-1.0.0/rest/tiposdocumento/recuperar)

Probar desde postman, reemplazando con su propia IP

En TestServices, agregar una función **getDocumentTypes** que permita invocar a este web service.

En TestWebServices, agregar un botón TIPOS DE DOCUMENTOS y hacer lo necesario para invocar a getDocumentTypes

Validar que trae los datos y muestra en consola

**PARTE 7**

Crear en Java un rest web service que permita insertar un registro en la base de datos, en la tabla tipo\_documentos, para este propósito debería:

Agregar un método para insertar en la clase TipoDocumentoBDD

Agregar un método en la clase ServiciosTiposDocumento

Generar y publicar el war y probar desde postman, validar que inserta en la base de datos.

**PARTE 8**

Modificar la pantalla PostForm, para que permita ingresar un tipo de documento en la base.

Puede usar los mismos métodos que existen createPost y createPostService, solo cambia la llamada al rest web service.

1. errores
2. proyecto actual
3. cambio en base