APLICACIONES MÓVILES MULTIPLATAFORMA

LABORATORIO N° 07

Esquema de aplicación

Alumno(s):		***Nota***	
Grupo:		***Ciclo:V***	

Criterio de Evaluación	*Excelente (4pts)*	*Bueno (3pts)*	*Requiere mejora (2pts)*	*No acept. (0pts)*	*Puntaje Logrado*
Agregar componentes y estilos					
Instalar y utilizar axios					
Consumir APIs de terceros					
Realiza con éxito lo propuesto en el laboratorio					
Es puntual y redacta el informe adecuadamente					

Laboratorio 07:
Esquemas de aplicación

Objetivos:

Al finalizar el laboratorio el estudiante será capaz de:

- Entender el funcionamiento de los estilos en componentes nativos
- Desarrollar aplicaciones web enfocadas a componentes
- Manejar de manera básica axios para consumir APIs

Seguridad:

- Ubicar maletines y/o mochilas en el gabinete del aula de Laboratorio.
- No ingresar con líquidos, ni comida al aula de Laboratorio.
- Al culminar la sesión de laboratorio apagar correctamente la computadora y la pantalla, y ordenar las sillas utilizadas.

Equipos y Materiales:

- Una computadora con:
  - Windows 7 o superior
  - VMware Workstation 10+ o VMware Player 7+
  - Conexión a la red del laboratorio
- Máquinas virtuales:
  - Windows 7 Pro 64bits Español - Plantilla
- Instalador de node.js

Procedimiento:

Lab Setup

1. **Configuración de proyecto**
   1.1. En su carpeta de trabajo, inicie un nuevo proyecto de react-native. En caso de usar una instalación nativa, utilice:

```
>react-native init lab07
```

   1.2. Iniciemos nuestro proyecto. En caso de usar la instalación nativa, utilizar:

```
>cd lab07
```

   1.3. Procederemos a instalar las dependencias para nuestra librería de navegación. Utilizaremos los siguientes tres comandos.

```
>npm install --save react-navigation
```

```
>npm i -S react-native-gesture-handler
```

```
>react-native link react-native-gesture-handler
```

   1.4. Ahora configuraremos la parte gestual de nuestra aplicación (para habilitar los menús). Esto se hará solamente para Android. Modificamos el archivo **MainActivity.java** ubicado en **android/app/src/main/java/com/lab06**

```
import com.facebook.react.ReactActivity;
import com.facebook.react.ReactActivityDelegate;
import com.facebook.react.ReactRootView;
import com.swmansion.gesturehandler.react.RNGestureHandlerEnabledRootView;

public class MainActivity extends ReactActivity {

    /**
     * Returns the name of the main component registered from JavaScript.
     * This is used to schedule rendering of the component.
     */
    @Override
    protected String getMainComponentName() {
        return "lab06";
    }

    @Override
    protected ReactActivityDelegate createReactActivityDelegate() {
        return new ReactActivityDelegate(this, getMainComponentName()) {
            @Override
            protected ReactRootView createRootView() {
                return new RNGestureHandlerEnabledRootView(MainActivity.this);
            }
        };
    }
```

1.5. Instalaremos el paquete async-storage y realizaremos su respectivo linkeado al proyecto

```
>npm install --save @react-native-community/async-storage
>react-native link @react-native-community/async-storage
```

1.6. Así mismo, añadiremos la siguiente línea en android/app/build.gradle para agregar soporte a imágenes GIF

```
dependencies {
    implementation project(':react-native-vector-icons')
    implementation project(':@react-native-community_async-storage')
    implementation project(':react-native-gesture-handler')
    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation "com.android.support:appcompat-v7:${rootProject.ext.supportLibVersion}"
    implementation "com.facebook.react:react-native:+"  // From node_modules
    implementation 'com.facebook.fresco:animated-gif:1.10.0'
}
```

1.7. Instalaremos así mismo, los siguientes paquetes que se utilizarán en efectos estéticos de la aplicación, a excepción de axios, que al igual que su uso en express, servirá para gestionar las llamadas a nuestra API.

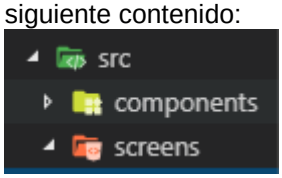
```
>npm install --save react-native-vector-icons
>react-native link react-native-vector-icons
>npm install --save react-native-textinput-effects
>npm install --save react-native-ionicons
>npm install --save axios
```

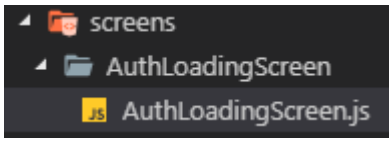
1.8. Crearemos el archivo axios.js con el siguiente contenido dentro de la carpeta lib dentro de src. La IP que aparece es referencial, en este caso, es la IP de mi computadora y el puerto de mi express. Modifíquelo por los valores necesarios para su laboratorio.

```
import axios from 'axios';

const instance = axios.create({
    baseURL: 'http://192.168.1.139:5000'
});

export default instance;
```

2. Pantalla de carga y navegación Básica

2.1. Crearemos la carpeta src, y dentro de ella las carpetas components y screens. Dentro de screens crearemos la carpeta AuthLoadingScreen con un archivo del mismo nombre y el siguiente contenido:






```
import React from 'react';
import { ActivityIndicator, StatusBar, View, Text } from 'react-native';
import AsyncStorage from '@react-native-community/async-storage';

export default class AuthLoadingScreen extends React.Component {
    componentDidMount() {
        this._bootstrapAsync();
    }
    _bootstrapAsync = async () => {
        const userToken = await AsyncStorage.getItem('userToken');
        this.props.navigation.navigate(userToken ? 'App' : 'Auth');
    };
    render() {
        return (
            <View
                style={{
                    flex: 1,
                    flexDirection: 'column',
                    justifyContent: 'center',
                    alignItems: 'stretch'
                }}
            >
                <ActivityIndicator />
                <StatusBar barStyle="default" />
                <Text
                    style={{
                        textAlign: 'center',
                        fontWeight: 'bold'
                    }}
                >
                    Cargando...
                </Text>
            </View>
        );
    }
}
```

2.2. Reemplazaremos el contenido del archivo App.js por el siguiente.

```
import {
    createSwitchNavigator,
    createDrawerNavigator,
    createAppContainer,
    createBottomTabNavigator
} from 'react-navigation';

import AuthLoadingScreen from './src/screens/AuthLoadingScreen/AuthLoadingScreen';
import SignInScreen from './src/screens/SignIn/SignIn';
import SignUpScreen from './src/screens/SignUp/SignUp';
import HomeScreen from './src/screens/Home/Home';
import ChatScreen from './src/screens/Chat/Chat';

const AppStack = createDrawerNavigator({ Home: HomeScreen, Other: ChatScreen });
const AuthStack = createBottomTabNavigator({
    SignIn: SignInScreen,
    SignUp: SignUpScreen
});

export default createAppContainer(
    createSwitchNavigator(
        {
            AuthLoading: AuthLoadingScreen,
            App: AppStack,
            Auth: AuthStack
        },
        {
            initialRouteName: 'AuthLoading'
        }
    )
);
```

Lo que se busca con esta implementación es crear dos navegaciones, una mientras no estamos autenticados y otra que se habilita cuando ya estamos logueados.

2.3. Crearemos dentro de screens la carpeta SignIn con el archivo SignIn y el mismo caso para SignUp con un archivo idéntico. Ambos tendrán por el momento el siguiente contenido:

```
import React from 'react';
import { View, Button, Image } from 'react-native';
import Ionicons from 'react-native-vector-icons/Ionicons';
import AsyncStorage from '@react-native-community/async-storage';

export default class SignInScreen extends React.Component {
    static navigationOptions = {
        title: 'Registrarse',
        tabBarIcon: ({ focused, horizontal, tintColor }) => {
            return <Ionicons name="ios-clipboard" size={25} color={tintColor} />;
        }
    };
    _signInAsync = async () => {
        await AsyncStorage.setItem('userToken', 'abc');
        this.props.navigation.navigate('App');
    };
    render() {
        return (
            <View>
                <Button title="Inicie sesión!" onPress={this._signInAsync} />
            </View>
        );
    }
}
```

2.4. Crearemos dentro de screens la carpeta Home con el archivo Home y el mismo caso para Chat con un archivo idéntico. Ambos tendrán por el momento el siguiente contenido:

```
import React from 'react';
import { View, Button } from 'react-native';
import Ionicons from 'react-native-vector-icons/Ionicons';

export default class HomeScreen extends React.Component {
    static navigationOptions = {
        title: 'Bienvenido a la App!',
        tabBarIcon: ({ focused, horizontal, tintColor }) => {
            return <Ionicons name="ios-clipboard" size={25} color={tintColor} />;
        }
    };
    _showMoreApp = () => {
        this.props.navigation.navigate('Chat');
    };
    _signOutAsync = async () => {
        await AsyncStorage.clear();
        this.props.navigation.navigate('Auth');
    };
    render() {
        return (
            <View>
                <Button title="Muestrame el Chat" onPress={this._showMoreApp} />
                <Button title="Mejor, cierra sesión :)" onPress={this._signOutAsync} />
            </View>
        );
    }
}
```

2.5. Compruebe el funcionamiento de la aplicación. Por el momento solamente debería poder navegar entre las dos vistas de inicio de sesión y de creación de cuenta, sin posibilidad de la vista chat y la vista de inicio.

3. Pantalla de inicio

3.1. Reemplace el contenido de SingIn.js por el siguiente

```
import React from 'react';
import { View, Text, ImageBackground } from 'react-native';
import Ionicons from 'react-native-vector-icons/Ionicons';

import imgBackground from '../../assets/img/sala-cine.jpg';

export default class SignInScreen extends React.Component {
    static navigationOptions = {
        title: 'Inicie sesión',
        tabBarIcon: ({ focused, horizontal, tintColor }) => {
            return <Ionicons name="ios-contact" size={25} color={tintColor} />;
        }
    };
    render() {
        return (
            <View style={{ flex: 1 }}>
                <ImageBackground
                    source={imgBackground}
                    style={{ width: '100%', height: '100%' }}
                >
                    <Text>Tecsup Videos</Text>
                </ImageBackground>
            </View>
        );
    }
}
```

3.2. El componente ImageBackground es de mucha utilidad para poner una imagen de fondo, pero las letras no se notan. Para modificar el estilo del texto de la app, usaremos el atributo style y las mismas propiedades que utilizamos en el navegador, con la excepción de que, si tuvieran un guion en el medio, este se reemplaza por una mayúscula.
Reemplacemos el único texto de la vista para ver cómo queda mejor el título.

```
<Text
    style={{
        textAlign: 'center',
        fontWeight: 'bold',
        fontSize: 48,
        color: '#fff'
    }}
>
    Tecsup Videos
</Text>
```

3.3. Ahora agregaremos los controles para ingresar usuario y contraseña. Modifiquemos las dependencias del inicio de sesión para que luzcan de la siguiente manera.

3.4.

```
import { View, Text, ImageBackground, TextInput } from 'react-native';
```

3.5. Agreguemos el componente TextInput después de nuestro Text, agregue una imagen de como se ve la aplicación con este control.

3.6.

```
<TextInput
    style={{ height: 40, borderColor: 'gray', borderWidth: 1 }}
/>
```

3.7. Al ver la presentación que nos entrega, podríamos agregar estilos para hacerlo lucir mejor. En este caso, optaremos por usar un componente ya construido. Agregamos la siguiente línea al bloque de importaciones del archivo.

```
import { Fumi } from 'react-native-textinput-effects';
```

3.8. Reemplazamos el código del TextInput por el siguiente y apreciamos el cambio.

```
<Fumi
    style={{
        backgroundColor: '#46494f',
        opacity: 0.8,
        marginBottom: 10
    }}
    label={'Usuario'}
    iconClass={Icon}
    keyboardType="email-address"
    iconName={'person'}
    iconColor={'#fff'}
    labelStyle={{ color: 'white' }}
    iconSize={30}
    iconWidth={40}
    inputPadding={16}
/>
```

3.9. A pesar de que la aplicación se ve mucho mejor, podemos centrar ese control de texto y darle margen para tener una mejor estética. Reemplazamos dicha porción de código (la que hace referencia a Fumi) e insertamos la siguiente vista que incluirá el control Fumi de usuario y al mismo tiempo, el de la contraseña.

```
<View style={{ padding: 10 }}>
    <View style={{ marginTop: 10 }}>
        <View>
            <Fumi
                style={{
                    backgroundColor: '#46494f',
                    opacity: 0.8,
                    marginBottom: 10
                }}
                label={'Usuario'}
                iconClass={Icon}
                keyboardType="email-address"
                iconName={'person'}
                iconColor={'#fff'}
                labelStyle={{ color: 'white' }}
                iconSize={30}
                iconWidth={40}
                inputPadding={16}
            />
        </View>
        <View style={{ flexDirection: 'row' }}>
            <Fumi
                style={{
                    width: '82%',
                    backgroundColor: '#46494f',
                    opacity: 0.8
                }}
                label={'Contraseña'}
                labelStyle={{ color: 'white' }}
                secureTextEntry={true}
                iconClass={Icon}
                iconName={'key'}
                iconColor={'#fff'}
                iconSize={30}
                iconWidth={40}
                inputPadding={16}
            />
        </View>
    </View>
</View>
```

4. Conexión con API

4.1. Agregaremos un estado al componente, como el mostrado a continuación. Nos ayudará a manejar el usuario, su contraseña, si hay alguna carga y si se debe mostrar o no la contraseña (más adelante, agregaremos una característica al control para ocultarla o mostrarla)

```
export default class SignInScreen extends React.Component {
    static navigationOptions = {
        title: 'Inicie sesión',
        tabBarIcon: ({ focused, horizontal, tintColor }) => {
            return <Ionicons name="ios-contact" size={25} col
        }
    };
    state = {
        user: '',
        password: '',
        loading: false,
        showPassword: false
    };
    render() {
        return (
            <View style={{ flex: 1 }}>
```

4.2. Así mismo, añadimos una función que recibirá dos parámetros: el campo a modificar en el estado, y el valor que deseamos tenga, de tal manera que una sola función me sirve para n campos.

```
    state = {
        user: '',
        password: '',
        loading: false,
        showPassword: false
    };
    inputHandler = (field, value) => {
        this.setState({ [field]: value });
    };
    render() {
        return (
            <View style={{ flex: 1 }}>
```

4.3. Ahora agregamos el value y el onChangeText en los respectivos Fumi. Para el caso del Fumi de contraseña, agregaremos la propiedad secureTextEntry, que dependiendo si es verdadera o falsa, mostrará u ocultará la contraseña digitada.

```
    label={'Usuario'}
    iconClass={Icon}
    keyboardType="email-address"
    onChangeText={text => this.inputHandler('user', text)}
    value={this.state.user}
    iconName={'person'}
    iconColor={'#fff'}
    labelStyle={{ color: 'white' }}
```

```
            label={'Contraseña'}
            labelStyle={{ color: 'white' }}
            onChangeText={text => this.inputHandler('password', text)}
            secureTextEntry={!this.state.showPassword}
            value={this.state.password}
            iconClass={Icon}
            iconName={'key'}
            iconColor={'#fff'}
```

No se olvide de añadir capturas de como luce la aplicación hasta el momento.

4.4. Nos falta un botón para iniciar sesión. Primero, modifiquemos nuestras importaciones del inicio del archivo, para luego agrega después de los Fumi un TouchableOpacity, es decir, un componente equivalente a una vista pero que reacciona cuando se dispara el evento onPress

```
import React from 'react';
import {
    View,
    Text,
    ImageBackground,
    TouchableOpacity,
    ToastAndroid
} from 'react-native';
import Ionicons from 'react-native-vector-icons/Ionicons';
```

```
                    iconSize={30}
                    iconWidth={40}
                    inputPadding={16}
                />
            </View>
            <TouchableOpacity
                onPress={this.onSubmitHandler}
                style={{
                    marginTop: 20,
                    padding: 15,
                    justifyContent: 'center',
                    alignItems: 'center',
                    borderRadius: 25,
                    backgroundColor: '#dcdcdc'
                }}
            >
                <Text
                    style={{
                        color: '#46494f',
                        fontSize: 15,
                        fontWeight: 'bold'
                    }}
                >
                    Iniciar Sesión
                </Text>
            </TouchableOpacity>
        </View>
    </View>
</ImageBackground>
```

4.5. Fíjese que nuestro TouchableOpacity hace referencia a una función onSubmitHandler. Pasaremos a crearla. Para esto, primero importaremos axios desde la parte de arriba de nuestra cabecera, donde están las importaciones, para luego añadir la función onSubmitHandler tal como está mostrada en las imágenes.

```
import axios from '../../lib/axios';
```

```
    inputHandler = (field, value) => {
        this.setState({ [field]: value });
    };
    onSubmitHandler = () => {
        if (this.state.user === '' || this.state.password === '') {
            return ToastAndroid.showWithGravity(
                'Falta ingresar datos!',
                ToastAndroid.SHORT,
                ToastAndroid.TOP
            );
        }
        this.setState({ loading: true });
        axios({
            method: 'POST',
            url: 'api/user/signin',
            data: {
                username: this.state.user,
                password: this.state.password
            }
        })
            .then(async response => {
                ToastAndroid.showWithGravity(
                    response.data.message,
                    ToastAndroid.LONG,
                    ToastAndroid.TOP
                );
                await AsyncStorage.setItem('userToken', response.data.token);
                this.props.navigation.navigate('App');
            })
            .catch(err => {
                ToastAndroid.showWithGravity(
                    'Hubo un error en el registro',
                    ToastAndroid.LONG,
                    ToastAndroid.TOP
                );
                console.warn(err);
                this.setState({ loading: false });
            });
    };
    render() {
        return (
```

4.6. Agregaremos un link en la parte inferior, para las personas que no tienen cuenta (como nosotros por el momento) que les lleve a la otra vista de inscripción. Para eso, antes del render, añadiremos la función registerHandler.

```
        });
    };
    registerHandler = () => {
        this.props.navigation.navigate('SignUp');
    };
    render() {
        return (
            <View style={{ flex: 1 }}>
```

4.7. Agregamos después de nuestro TouchableOpacity la siguiente vista con el enlace a nuestro formulario de registro.

```
                </TouchableOpacity>
                <View
                    style={{
                        marginTop: 10,
                        justifyContent: 'center',
                        alignItems: 'center',
                        alignSelf: 'center'
                    }}
                >
                    <Text style={{ color: '#fff', fontSize: 14 }}>
                        No tienes una cuenta?
                        <Text
                            onPress={this.registerHandler}
                            style={{
                                color: '#fff',
                                fontSize: 16,
                                fontWeight: 'bold'
                            }}
                        >
                            {' '}
                            Registrate aquí
                        </Text>
                    </Text>
                </View>
            </View>
        </View>
    </ImageBackground>
```

4.8. Finalmente, agregaremos un ícono que indique si es seguro o no mostrar la contraseña.

```
        label={'Contraseña'}
        labelStyle={{ color: 'white' }}
        onChangeText={text => this.inputHandler('password', text)}
        secureTextEntry={!this.state.showPassword}
        value={this.state.password}
        iconClass={Icon}
        iconName={'key'}
        iconColor={'#fff'}
        iconSize={30}
        iconWidth={40}
        inputPadding={16}
    />
    <Icon
        color="#fff"
        style={{
            padding: 20,
            alignItems: 'center',
            backgroundColor: '#46494f',
            opacity: 0.8,
            height: 65
        }}
        size={25}
        name={this.state.showPassword ? 'md-eye' : 'md-eye-off'}
        onPress={this.showPassword}
    />
</View>
<TouchableOpacity
    onPress={this.onSubmitHandler}
```

4.9. Y para cerrar el funcionamiento del formulario, añadiremos la función showPassword, que modificará el estado, haciendo visible o no la contraseña ingresada.

```
    state = {
        user: '',
        password: '',
        loading: false,
        showPassword: false
    };
    showPassword = () => {
        this.setState({ showPassword: !this.state.showPassword });
    };
    inputHandler = (field, value) => {
        this.setState({ [field]: value });
    };
    onSubmitHandler = () => {
```

5. Acomodación automática.
   5.1. Tal vez no lo note ahora, pero en dispositivos con pantalla pequeña, o inclusive, aplicaciones con pantallas muy cargadas, sufren de que cuando se va a utilizar el teclado del celular, la pantalla deja de ser visible en ciertos sectores. Para evitar eso, añadiremos el componente KeyBoardAvoidingView de react-native.

```
import React from 'react';
import {
    View,
    Text,
    TouchableOpacity,
    ImageBackground,
    ToastAndroid,
    KeyboardAvoidingView
} from 'react-native';
import Icon from 'react-native-ionicons';
```

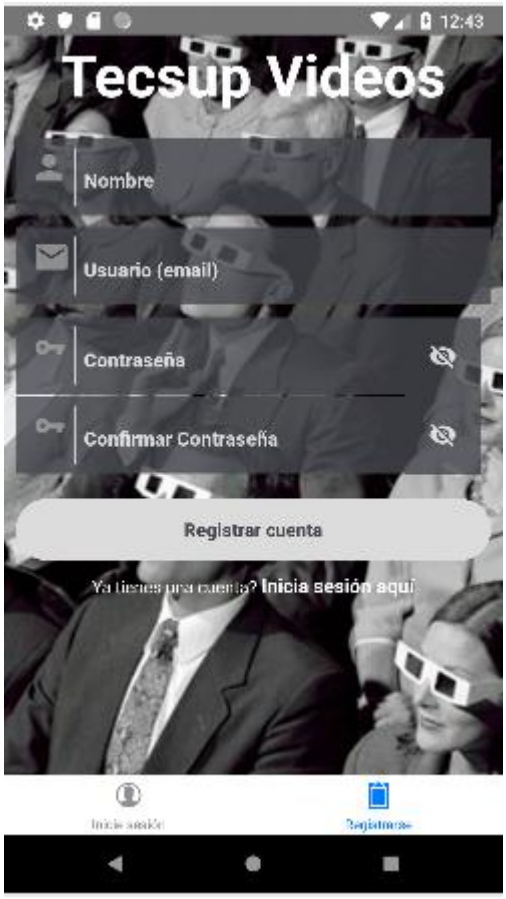
   5.2. Este componente deberá envolver todo dentro de nuestra imagen de fondo (tiene que ser dentro y envolviendo, para que no se altere el fondo)

```
render() {
    return (
        <View style={{ flex: 1 }}>
            <ImageBackground
                source={imgBackground}
                style={{ width: '100%', height: '100%' }}
            >
                <KeyboardAvoidingView
                    behavior="position"
                    style={{ justifyContent: 'center' }}
                    enabled
                >
                    <Text
                        style={{
                            textAlign: 'center',
```

```
                    </View>
                </View>
                </KeyboardAvoidingView>
            </ImageBackground>
        </View>
    );
    }
}
```

6. Ejercicio propuesto

6.1. Tomando como referencia la siguiente imagen, copiar el contenido de SignIn y copiarlo en SignUp. Se debe modificar el fondo (ambos fondos, están proveídos por el docente, como adjuntos a este laboratorio)



6.2. Se debe modificar la función de navegación, ya no me llevará a SignUp, sino a SignIn.

6.3. Así mismo, se debe modificar el onSubmitHandler para que aparezca el campo name y el campo confirmar contraseña.

Así mismo, la URL del servicio variará. A continuación, se adjunta una sugerencia de los cambios dentro de onSubmitHandler. Tome en cuenta que deben realizarse los cambios necesarios del render para que este envío de información no tenga inconvenientes.

```
};
onSubmitHandler = () => {
    if (
        this.state.user === '' ||
        this.state.name === '' ||
        this.state.password === ''
    ) {
        return ToastAndroid.showWithGravity(
            'Falta ingresar datos!',
            ToastAndroid.SHORT,
            ToastAndroid.TOP
        );
    }
    if (this.state.password !== this.state.password2) {
        return ToastAndroid.showWithGravity(
            'Las contraseñas no coinciden',
            ToastAndroid.SHORT,
            ToastAndroid.TOP
        );
    }
    this.setState({ loading: true });
    axios({
        method: 'POST',
        url: 'api/user/signup',
        data: {
            email: this.state.user,
            username: this.state.name,
            password: this.state.password
        }
    })
        .then(async response => {
            ToastAndroid.showWithGravity(
                response.data.message,
```

7. **Finalizar la sesión**
   7.1. Apagar el equipo virtual
   7.2. Apagar el equipo

Conclusiones:

Indicar las conclusiones que llegó después de los temas tratados de manera práctica en este laboratorio.