APLICACIONES MÓVILES MULTIPLATAFORMA

LABORATORIO N° 12

Settings y Flatlist

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

Criterio de Evaluación	*Excelente (4pts)*	*Bueno (3pts)*	*Requiere mejora (2pts)*	*No acept. (0pts)*	*Puntaje Logrado*
Agregar componentes de estilos					
Reutiliza componentes de UI					
Crea sus propios componentes de UI					
Realiza con éxito lo propuesto en el laboratorio					
Es puntual y redacta el informe adecuadamente					

Laboratorio 12:
Settings y Flatlist

Objetivos:

Al finalizar el laboratorio el estudiante será capaz de:

- Entender el funcionamiento de los estilos en React Native
- Desarrollar componentes reutilizables visuales para toda la aplicación
- Importar con éxito una librería creada para un proyecto web y acomodarla al proyecto móvil

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. Copie el contenido del laboratorio 11 (la clase anterior) a excepción de la carpeta node_modules en una nueva carpeta llamada lab12 y reinstale todas las dependencias:

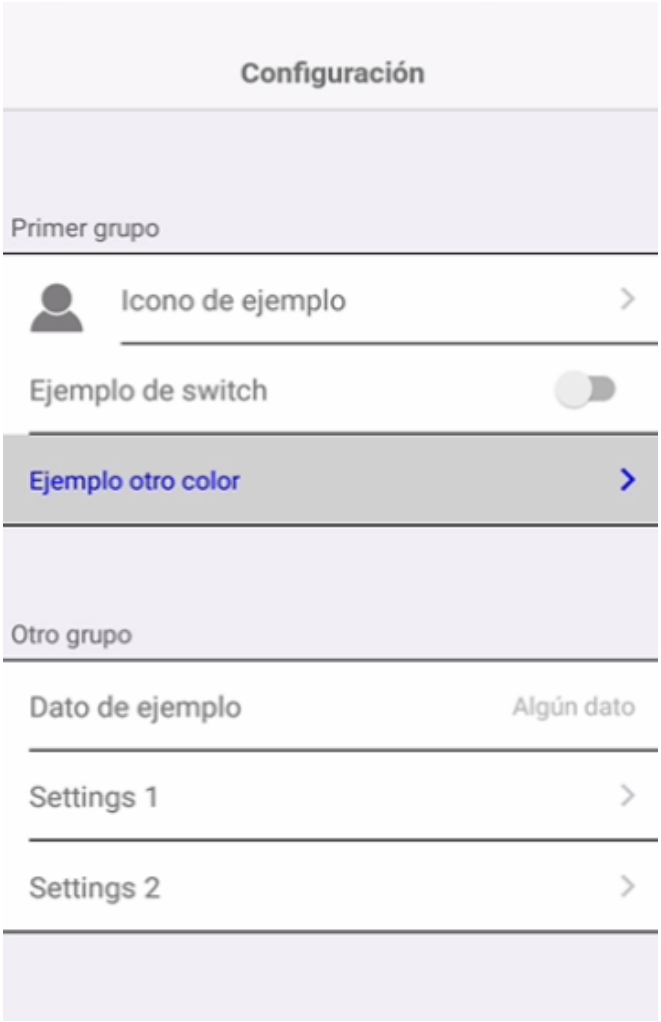
```
>npm install
```

   1.2. En la nueva carpeta lab12, instalaremos las siguientes dependencias:

```
>npm install --save react-native-settings-list
```

2. Vista de Configuración

2.1. Casi todas nuestras aplicaciones suelen tener una vista de configuración para modificar distintos detalles de su comportamiento. Nosotros queremos lograr una vista como la siguiente presentada:



2.2. Para lograr esto, modificaremos el archivo **Settings.js** creado en el laboratorio anterior y pondremos el siguiente contenido (iré comentando cada bloque, pero el código es íntegro un solo archivo hasta llegar al siguiente punto del laboratorio)

```
import React, { Component } from 'react';
import { View, Image, Alert, Text } from 'react-native';
import SettingsList from 'react-native-settings-list';

class Settings extends Component {
    state = {
        switchValue: false
    };
    onValueChange = value => {
        this.setState({ switchValue: value });
    };
    render() {
        return (
            <View style={{ backgroundColor: '#EFEFF4', flex: 1 }}>
                <View
                    style={{
                        borderBottomWidth: 1,
                        backgroundColor: '#f7f7f8',
                        borderColor: '#c8c7cc'
                    }}
                >
                    <Text
                        style={{
                            alignSelf: 'center',
                            marginTop: 30,
                            marginBottom: 10,
                            fontWeight: 'bold',
                            fontSize: 16
                        }}
                    >
                        Configuración
                    </Text>
                </View>
```

Ahora, utilizaremos la dependencia recién instalada, **SettingsList** que nos permitirá crear un componente con todo el look and feel de una sección de configuración. Fíjese que tenemos desde nuestros headers, opciones con íconos, inclusive opciones con switchs.

```
                <View style={{ flex: 1, marginTop: 50 }}>
                    <SettingsList>
                        <SettingsList.Header
                            headerText="Primer grupo"
                            headerStyle={{ color: '#666' }}
                        />
                        <SettingsList.Item
                            icon={
                                <View
                                    style={{ height: 30, marginLeft: 10, alignSelf: 'center' }}
                                >
                                    <Image
                                        style={{ alignSelf: 'center', height: 40, width: 40 }}
                                        source={require('../../assets/img/icon-profile.png')}
                                    />
                                </View>
                            }
                            itemWidth={50}
                            title="Icono de ejemplo"
                            onPress={() => Alert.alert('Icono de ejemplo presionado')}
                        />
                        <SettingsList.Item
                            hasNavArrow={false}
                            switchState={this.state.switchValue}
                            switchOnValueChange={this.onValueChange}
                            hasSwitch={true}
                            title="Ejemplo de switch"
                        />
```

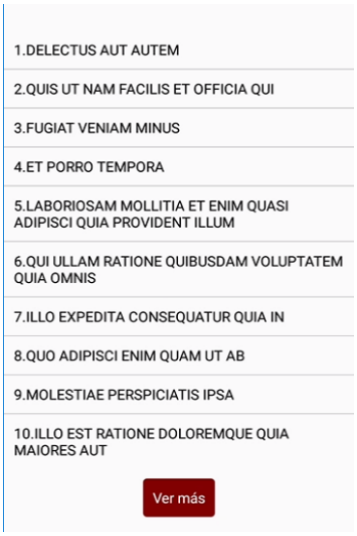
Fíjese que estos elementos inclusive tienen el evento onPress, para poder determinar que cosas sucederán cuando sean presionados. Podremos por ejemplo mostrar un modal, llevarnos a otra pantalla, etc.

```
                        <SettingsList.Item
                            title="Ejemplo otro color"
                            backgroundColor="#D1D1D1"
                            titleStyle={{ color: 'blue' }}
                            arrowStyle={{ tintColor: 'blue' }}
                            onPress={() => Alert.alert('Ejemplo de otro color presionado!')}
                        />
                        <SettingsList.Header
                            headerText="Otro grupo"
                            headerStyle={{ color: '#666', marginTop: 50 }}
                        />
                        <SettingsList.Item
                            titleInfo="Algún dato"
                            hasNavArrow={false}
                            title="Dato de ejemplo"
                        />
                        <SettingsList.Item title="Settings 1" />
                        <SettingsList.Item title="Settings 2" />
                    </SettingsList>
                </View>
            </View>
        );
    }
}

export default Settings;
```

3. Listado con paginación

3.1. Ya hemos utilizado el componente Flatlist en laboratorios anteriores, pero ahora, no solamente lo integraremos a una API, sino que queremos funcione la paginación para cargar más datos (recuerde que es una mala práctica cargar toda la información de golpe, debido a que puede hacer esperar mucho al usuario y tal vez este no la use toda). Queremos obtener un listado como el siguiente, con un botón al final de Ver más.



3.2. Modificaremos el archivo **Lists.js** creado en el laboratorio anterior.

```
import React, { Component } from 'react';
import {
    View,
    Text,
    TouchableOpacity,
    StyleSheet,
    FlatList,
    ActivityIndicator
} from 'react-native';

export default class Lists extends Component {
    state = {
        loading: true,
        serverData: [],
        fetching_from_server: false,
        offset: 1
    };

    componentDidMount() {
        fetch('http://aboutreact.com/demo/getpost.php?offset=' + this.state.offset)
            .then(response => response.json())
            .then(responseJson => {
                this.setState({
                    serverData: [...this.state.serverData, ...responseJson.results],
                    offset: this.state.offset + 1,
                    loading: false
                });
            })
            .catch(error => {
                console.error(error);
            });
    }
```

Note que, en el componentDidMount, es decir, en cuanto carga la aplicación, hacemos una llamada a nuestra API, en este caso, una API que nos devuelve nombres aleatorios. Así mismo, dentro del query de la URL enviamos un parámetro offset, que vendría a equivaler en que página estamos. Inicialmente, al renderizar nuestro componente, estaremos siempre en la página 1, por eso se inicializa así en el estado, pero en cuanto hay una carga, se incrementa dicho valor.

```
    loadMoreData = () => {
        this.setState({ fetching_from_server: true }, () => {
            fetch(
                'http://aboutreact.com/demo/getpost.php?offset=' + this.state.offset
            )
                .then(response => response.json())
                .then(responseJson => {
                    this.setState({
                        serverData: [...this.state.serverData, ...responseJson.results],
                        offset: this.state.offset + 1,
                        fetching_from_server: false
                    });
                })
                .catch(error => {
                    console.error(error);
                });
        });
    };
```

De igual manera, crearemos una función loadMoreData, que será la encargada de traer la siguiente página cada vez que se nos solicite. Note como en el setState de serverData hacemos una deestructuración del state actual […] y la respuesta obtenida. Otra opción sería hacer un merge de arrays, pero esa sintaxis equivale a unir ambos arrays.

```
renderFooter = () => {
    return (
        <View style={styles.footer}>
            <TouchableOpacity
                activeOpacity={0.9}
                onPress={this.loadMoreData}
                style={styles.loadMoreBtn}
            >
                <Text style={styles.btnText}>Ver más</Text>
                {this.state.fetching_from_server ? (
                    <ActivityIndicator color="white" style={{ marginLeft: 8 }} />
                ) : null}
            </TouchableOpacity>
        </View>
    );
};
```

Ahora estamos creando un método renderFooter. Este **NO ES** un método de React como es el caso de render o componentDidMount, sino es una función que nosotros estamos creando para renderizar el footer de nuestro componente de listado. Recuerde que usted es libre de cambiar los nombres de estas funciones, pero para poder identificarla rápidamente, le ponemos un nombre que describa lo que hace (renderizar el footer del listado)

```
render() {
    return (
        <View style={styles.container}>
            {this.state.loading ? (
                <ActivityIndicator size="large" />
            ) : (
                <FlatList
                    style={{ width: '100%' }}
                    keyExtractor={(item, index) => index}
                    data={this.state.serverData}
                    renderItem={({ item, index }) => (
                        <View style={styles.item}>
                            <Text style={styles.text}>
                                {item.id}
                                {'.'}
                                {item.title.toUpperCase()}
                            </Text>
                        </View>
                    )}
                    ItemSeparatorComponent={() => <View style={styles.separator} />}
                    ListFooterComponent={this.renderFooter}
                />
            )}
        </View>
    );
  }
}
```

Finalmente, y el truco de todo el ejercicio, en nuestro componente FlatList incluiremos la propiedad ListFooterComponent, que a su vez llamará al renderFooter. De esta forma, nuestro FlatList podrá incluir dicho pie de lista y podremos generar un mejor aspecto de la aplicación. Solamente faltan los estilos de nuestra lista para acabar y ya deberíamos tener el componente funcionando.

```
const styles = StyleSheet.create({
    container: {
        flex: 1,
        justifyContent: 'center',
        alignItems: 'center',
        paddingTop: 30
    },
    item: {
        padding: 10
    },
    separator: {
        height: 0.5,
        backgroundColor: 'rgba(0,0,0,0.4)'
    },
    text: {
        fontSize: 15,
        color: 'black'
    },
```

```
    footer: {
        padding: 10,
        justifyContent: 'center',
        alignItems: 'center',
        flexDirection: 'row'
    },
    loadMoreBtn: {
        padding: 10,
        backgroundColor: '#800000',
        borderRadius: 4,
        flexDirection: 'row',
        justifyContent: 'center',
        alignItems: 'center'
    },
    btnText: {
        color: 'white',
        fontSize: 15,
        textAlign: 'center'
    }
});
```

4. Ejercicio propuesto

4.1. La vista **Lists.js** deberá apuntar a nuestra API de usuarios de **dawa_api**. Haga las modificaciones necesarias para que renderice dicha información.

5. **Finalizar la sesión**

5.1. Apagar el equipo virtual

5.2. Apagar el equipo

Conclusiones:

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