APLICACIONES MÓVILES MULTIPLATAFORMA

LABORATORIO N° 08

Chat y Socket.IO

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

Criterio de Evaluación	*Excelente (4pts)*	*Bueno (3pts)*	*Requiere mejora (2pts)*	*No acept. (0pts)*	*Puntaje Logrado*
Agregar componentes de Chat					
Instalar y utilizar socket io client					
Consumir servicios en tiempo real					
Realiza con éxito lo propuesto en el laboratorio					
Es puntual y redacta el informe adecuadamente					

Laboratorio 08:
Chat y Socket.IO

Objetivos:

Al finalizar el laboratorio el estudiante será capaz de:

- Entender el funcionamiento del componente GiftedChat
- Desarrollar aplicaciones web enfocadas a componentes
- Consumo de servicios en tiempo real

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 API**
   1.1. En el proyecto **dawa_api**, instalaremos **socket.io**:

```
>npm install --save socket.io
```

   1.2. Ahora, modificaremos las líneas iniciales del archivo server.js para que podamos invocar a socket.io

```
const express = require('express');
const app = express();
var server = require('http').Server(app);
var io = require('socket.io')(server);
const bodyParser = require('body-parser');
```

   1.3. Finalmente, modificaremos las líneas finales de server.js para poder hacer uso de socket

```
server.listen(port);
console.log('La magia sucede en el puerto ' + port);

io.on('connection', function(socket) {
    console.log('Conectado');
});
```

2. **Configuración de proyecto**
   2.1. Copie el contenido del laboratorio 7 (la clase anterior) a excepción de la carpeta node_modules en una nueva carpeta llamada lab08 y reinstale todas las dependencias:

```
>npm install
```

2.2. En la nueva carpeta lab08, instalaremos las siguientes dependencias:

```
>npm install react-native-gifted-chat --save
```

```
>npm install --save socket.io-client
```

2.3. Modificaremos index.js, agregándole la dependencia de SocketIOClient y al mismo tiempo la invocación al socket. Lo hacemos en este archivo ya que es el entry point de todo el proyecto: **es invocado una sola vez y no necesitamos invocar a socket.io en cada componentDidMount**

```
import { AppRegistry } from 'react-native';
import App from './App';
import { name as appName } from './app.json';
import SocketIOClient from 'socket.io-client';

global.socket = SocketIOClient('http://192.168.1.139:5000');

AppRegistry.registerComponent(appName, () => App);
```

Una vez más, nótese la dirección en el ejemplo, es mi IP local, la cuál será reemplazada por la IP que tenga usted.

2.4. Lo arriba realizado servirá para mostrar en la consola de su servidor Express un mensaje como el siguiente:

```
C:\Users\favio\code\tecsup\dawa\dawa_api>npm start

> lab12@1.0.0 start C:\Users\favio\code\tecsup\dawa\dawa_api
> node server.js

La magia sucede en el puerto 5000
Conectado
```

2.5. Agregaremos los eventos de chat que consumiremos en la aplicación. Modificaremos a **server.js** para agregar lo siguiente al final del archivo:

```
server.listen(port);
console.log('La magia sucede en el puerto ' + port);

io.on('connection', function(socket) {
    console.log('Conectado');
    socket.on('message', message => {
        // El evento broadcast nos permite enviarle a todos menos al emisor
        console.log('recibo', message);
        socket.broadcast.emit('message', message);
    });
});
```

2.6. Como comentario adicional, no se ha mencionado antes, pero esto nos será muy útil, para poder ver errores o mensajes en nuestra consola, tal como sucede en express, haremos uso del siguiente comando (en otra consola situada en la carpeta del proyecto)

```
>react-native log-android
```

2.7. Para demostrar dicho funcionamiento, modificaremos **SignIn.js** en la parte de la respuesta del servidor (justo donde almacenamos el token) para agregar un **console.log** y ver lo que aparecerá en nuestra consola.

```
.then(async response => {
    console.log(response.data);
    await AsyncStorage.setItem('userId', response.data.data._id);
    await AsyncStorage.setItem('userName', response.data.data.username);
    await AsyncStorage.setItem('userToken', response.data.token);
    ToastAndroid.showWithGravity(
        response.data.message,
        ToastAndroid.LONG,
        ToastAndroid.TOP
    );
    this.props.navigation.navigate('App');
})
```

3. Creación de vista para Chat
   3.1. Modificaremos Chat.js, el contenido será el siguiente

```
import React from 'react';
import AsyncStorage from '@react-native-community/async-storage';
import { GiftedChat } from 'react-native-gifted-chat';

export default class HomeScreen extends React.Component {
    state = {
        messages: [],
        userId: null
    };
    async componentDidMount() {
        this.socket = global.socket;
        this.socket.on('message', this.onReceivedMessage);
        const userId = await AsyncStorage.getItem('userId');
        this.setState({ userId: userId });
    }
    onReceivedMessage = messages => {
        this._storeMessages(messages);
    };
    onSend = (messages = []) => {
        this.socket.emit('message', messages[0]);
        this._storeMessages(messages);
    };
```

Fíjese como en el componentDidMount llamamos a la variable global socket, a la cual le agregamos el evento message, es decir, le indicamos a nuestro componente que este a la escucha de un nueva message del socket. (Recuerde que este nombre de evento puede variar, si usted desea, llámelo de otra forma, por ejemplo “recibir-mensaje” y en el servidor deberá modificar el emit que se realiza a “message” por “recibir-mensaje”)
A continuación, el resto del contenido del archivo.

```
    onSend = (messages = []) => {
        this.socket.emit('message', messages[0]);
        this._storeMessages(messages);
    };
    _storeMessages = messages => {
        this.setState(previousState => {
            return {
                messages: GiftedChat.append(previousState.messages, messages)
            };
        });
    };
    render() {
        const user = { _id: this.state.userId || -1 };
        return (
            <GiftedChat
                messages={this.state.messages}
                onSend={this.onSend}
                user={user}
            />
        );
    }
}
```

3.2. Fíjese como el componente **GiftedChat** necesita pocas propiedades para poder armar un chat ya funcional. Dígale a su compañero que modifique la dirección IP de su proyecto de React Native hacia el suyo para que ambos puedan conversar (deben estar en el mismo servidor para que funcione el chat) Vea así mismo que pueden participar muchas personas sin problema. Para mayor detalle de la documentación del componente, puede acceder a:

https://github.com/FaridSafi/react-native-gifted-chat

3.3. El problema actual con el que nos encontramos es que no podemos regresar a la vista anterior ya que el componente cubre toda la pantalla. Usaremos la propiedad renderActions de GiftedChat para poder agregar un botón de acciones. Modifiquemos entonces el render:

```
<GiftedChat
    placeholder="Escribe algo..."
    renderActions={() => {
        return (
            <Icon
                color="#fff"
                style={{
                    padding: 5,
                    alignItems: 'center',
                    backgroundColor: '#46494f',
                    opacity: 0.8,
                    height: 40
                }}
                size={25}
                name={'md-settings'}
                onPress={this.settingsHandler}
            />
        );
    }}
    messages={this.state.messages}
    onSend={this.onSend}
    user={user}
/>
```

3.4. Modificaremos el estado del componente, para mostrar y/o ocultar un modal con las opciones del chat.

```
state = {
    messages: [],
    userId: null,
    modalVisible: false
};
```

3.5. Importaremos Fragment de React para encapsular dos componentes adyacentes, e importaremos el componente Modal de React Native

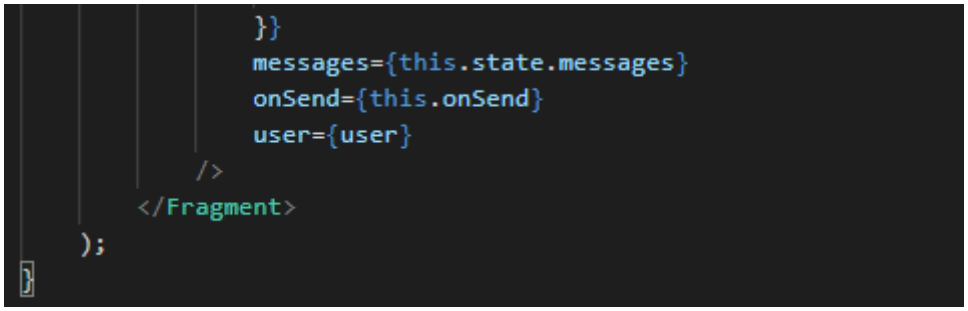
```
import React, { Fragment } from 'react';
import { Modal, View, Button } from 'react-native';
import Icon from 'react-native-ionicons';
```

3.6. Agregaremos los manejadores del evento de mostrar o no el modal

```
settingsHandler = () => {
    this.setState({ modalVisible: true });
};
chatHandler = () => {
    this.setState({ modalVisible: false });
};
backHandler = () => {
    this.setState({ modalVisible: false }, () => {
        this.props.navigation.navigate('Home');
    });
};
```

3.7. Modificamos nuevamente el render, tanto el inicio como el final (no modificaremos el GiftedChat)

```
render() {
    const user = { _id: this.state.userId || -1 };
    return (
        <Fragment>
            <Modal
                animationType="slide"
                transparent={false}
                visible={this.state.modalVisible}
            >
                <View>
                    <Button
                        onPress={this.chatHandler}
                        title="Regresar al Chat"
                        color="#841584"
                    />
                    <Button
                        onPress={this.backHandler}
                        title="Regresar al Inicio"
                        color="red"
                    />
                </View>
            </Modal>
            <GiftedChat
                placeholder="Escribe algo..."
                renderActions={() => {
```



3.8. Ahora ya tiene un ejemplo funcional de cómo crear un Chat con React Native y Socket IO, el cuál iremos mejorando con las siguientes clases.
3.9. Si desea modificar el estilo del Modal, puede hacerlo con la documentación oficial. https://facebook.github.io/react-native/docs/modal

4. Ejercicio Propuesto

4.1. Agregue imágenes de usuarios al chat, por ahora estáticas (siguiendo la documentación oficial proveída en el laboratorio)

4.2. Modifique el Modal para que sea solamente un cuadrado pequeño en el centro y haya una transparencia que permita ver el chat atrás (siguiendo la documentación oficial proveída en el laboratorio)

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.