React Native Build Real Time Chat Application

Chapter 7 - Topic 4

Selamat datang di Chapter 7 Topic 4 online course React Native dari Binar Academy!

Guyys!

Kita sudah di topik terakhir Chapter 7 nih! Nggak terasa yaaa. Nah, kalau di topik sebelumnya kamu sudah belakar tentang Real Time Database, maka sekarang kita akan membahas tentang pembuatan **Real Time Chat Application**.

Nggak usah pake lama, cus langsung kita bahas!

Detailnya, kita bakal bahas hal berikut ini:

- Cara membuat Real Time Chat Application dengan real time database

“Hape terooos!”

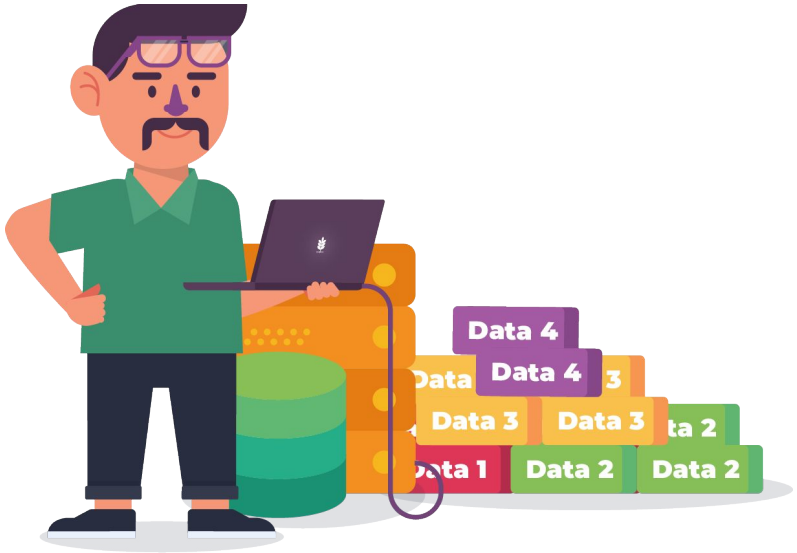
Begitu kira-kira ya kalau nyokap lihat kita lagi sibuk chatting sama gebetan hehe.

Berani maju duluan pas PDKT emang keren, tapi lebih keren lagi kalau kamu bisa bikin aplikasi chat!

Fufufu~ Another level of coolness.

Pada chapter sebelumnya, kamu sudah mempelajari bagaimana real time database dapat bekerja sebagai penyedia layanan backend kan yaa.

Kamu juga membuat database sebagai service yang memakai cloud hosting, sehingga kamu ataupun user dapat melakukan read, write terhadap keperluan data.



Nah, sekarang kamu akan belajar membuat suatu aplikasi hasil dari implementasi menggunakan real time database.

Yak benar! Aplikasi chat!

Zaman sekarang, aplikasi chat pasti sudah jadi salah satu aplikasi wajib yang ada dalam smart phone setiap orang, termasuk kamu.

Salah satu alasannya adalah aplikasi ini memungkinkanmu untuk berkirim pesan secara real time, cepat, dan efisien, baik untuk keperluan pribadi maupun profesional.

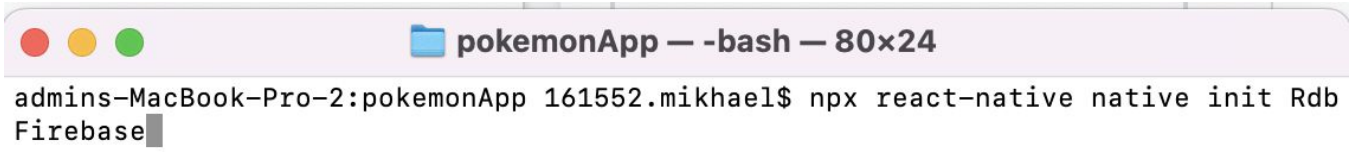
Okeee, daripada penasaran, langsung saja kita bahas cara pembuatannya yuk!

Langkah awal tentunya bikin project react native-nya dong~

Setelah itu kamu akan belajar membuat halaman login dan register untuk aplikasi chat-nya.

1. Create React Native Project

Buatlah project React Native dengan cara menjalankan command `React-native init project_name` di terminal. Selanjutnya lakukan **setup konfigurasi Firebase dan install module realtime database** atau kamu bisa menggunakan project pada chapter sebelumnya.

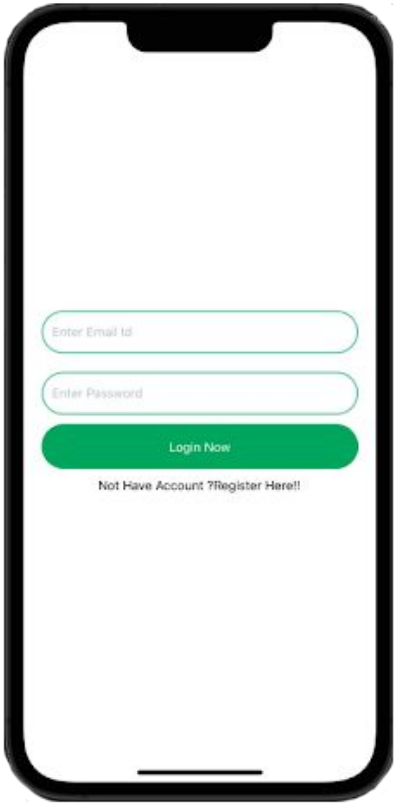


2. Login User

Oke, sekarang **buatlah Login Component** sederhana yang berisikan text input username/email dan password, serta button untuk redirect ke register.

```
<View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
    <View style={[styles.inputContainer, { marginTop: 10 }]}>
        <TextInput
            style={styles.inputs}
            placeholder="Enter Email Id"
            // keyboardType="number-pad"
            underlineColorAndroid="transparent"
            onChangeText={value => {
                setEmail(value)
            }}
            value={email}
            placeholderTextColor={COLORS.liteBlack}
        />
    </View>
    <View style={[styles.inputContainer, { marginTop: 10 }]}>
        <TextInput
            style={styles.inputs}
            placeholder="Enter Password"
            // keyboardType="number-pad"
            underlineColorAndroid="transparent"
            onChangeText={value => {
                setPassword(value)
            }}
            value={password}
            placeholderTextColor={COLORS.liteBlack}
        />
    </View>
    <TouchableOpacity
        style={styles.btn}
        onPress={() => onLoginRDB()}
    >
        <Text style={styles.btnText}>Login Now</Text>
    </TouchableOpacity>
    <View style={{ flexDirection: 'row', margin: 10 }}>
        <Text>Not Have Account ?</Text>
        <TouchableOpacity onPress={() => props.navigation.navigate('Register')}>
            <Text>Register Here!!</Text>
        </TouchableOpacity>
    </View>
```

Contoh hasilnya seperti gambar di samping ini ya.

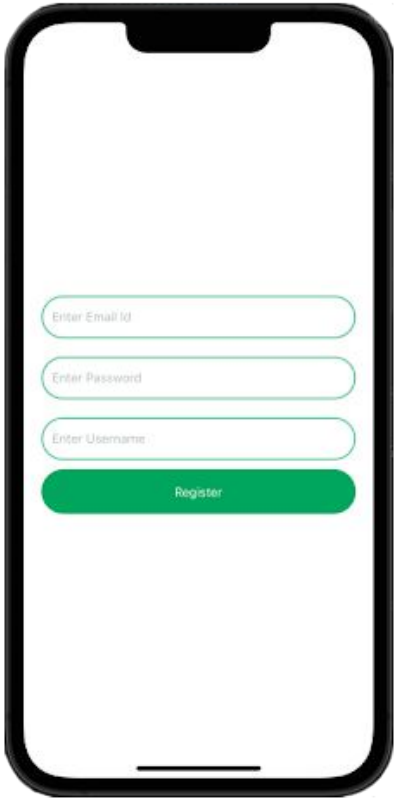


3. Register User

Selanjutnya, **buatlah halaman register** sederhana dimana user dapat memasukkan email, password, dan name.

```
return (
    <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
        <View style={[styles.inputContainer, { marginTop: 10 }]}>
            <TextInput
                style={styles.inputs}
                placeholder="Enter Email Id"
                keyboardType="number-pad"
                underlineColorAndroid="transparent"
                onChangeText={value => {
                    setEmail(value)
                }}
                value={email}
                placeholderTextColor={COLORS.liteBlack}
            />
        </View>
        <View style={[styles.inputContainer, { marginTop: 10 }]}>
            <TextInput
                style={styles.inputs}
                placeholder="Enter Password"
                keyboardType="number-pad"
                underlineColorAndroid="transparent"
                onChangeText={value => {
                    setPassword(value)
                }}
                value={password}
                placeholderTextColor={COLORS.liteBlack}
            />
        </View>
        <View style={[styles.inputContainer, { marginTop: 10 }]}>
            <TextInput
                style={styles.inputs}
                placeholder="Enter Username"
                keyboardType="number-pad"
                underlineColorAndroid="transparent"
                onChangeText={value => {
                    setName(value)
                }}
                value={name}
                placeholderTextColor={COLORS.liteBlack}
            />
        </View>
        <TouchableOpacity
```

Contoh hasilnya akan seperti gambar di samping yaaa.



Buat sekreatif mungkin, setidaknya untuk text input sederhana seperti pada slide sebelumnya.

Pastikan kamu **sudah menambahkan react-navigation** yang berguna untuk navigasi antar screen.

```
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
```

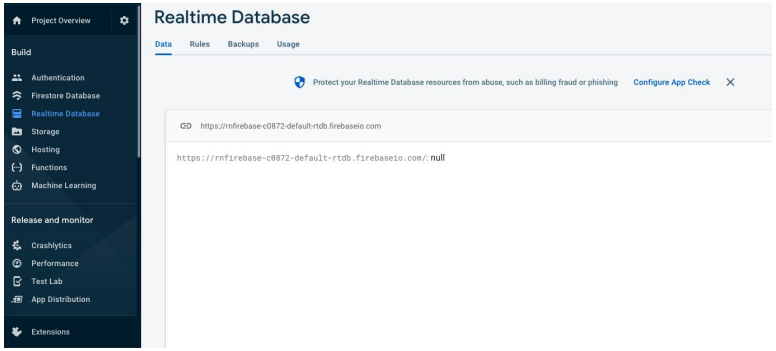
```
export default function App() {
  return (
    <NativeBaseProvider>
    <NavigationContainer>
      <Stack.Navigator
        detachInactiveScreens={false}
        initialRouteName="Auth"
        screenOptions={{
          headerShown: false,
          cardStyle: { backgroundColor: COLORS.white },
        }}>
        <Stack.Screen name="Login" component={Login} />
        <Stack.Screen name="ChatScreen" component={ChatScreen}/>
        <Stack.Screen name="Register" component={Register}/>
        <Stack.Screen name="DashboardUser" component={DashboardUser}/>
      </Stack.Navigator>
    </NavigationContainer>
    </NativeBaseProvider>
  )
}
```

Kalau tampilan login dan registernya sudah siap, sekarang kita siapin database terkait register dan loginnya yuk~

Makin penasaran kaann~

Sebagai langkah awal~

Pastikan kamu sudah menginstall module real time database seperti pada topik sebelumnya, dan **cek dashboard database kalian di Firebase** yaa**.**



Register With Real Time Database

Selanjutnya kamu akan membuat function. Kamu akan **menulis data ke database setiap kali user Register akun.**

Contoh pada code di samping adalah function bernama onRegisterWithRDB.

```
const onRegisterWithRDB = async () => {
    if (name == '' || email == '' || password == '') {
        Alert.alert('Error', 'Harap isi Semua field')
        return false;
    }
    let data = {
        id: uuid.v4(),
        name: name,
        emailId: email,
        password: password,
    };
    try {
        database()
            .ref('/users/' + data.id)
            .set(data)
            .then(() => {
                Alert.alert('Success', 'Register Successfully!'
                setEmail('')
                setPassword('')
                props.navigation.navigate('Login')
            });
    } catch (error) {
        Alert.alert('Error', error)
    }

}
```

Singkatnya, tahapan function tersebut terdiri dari :

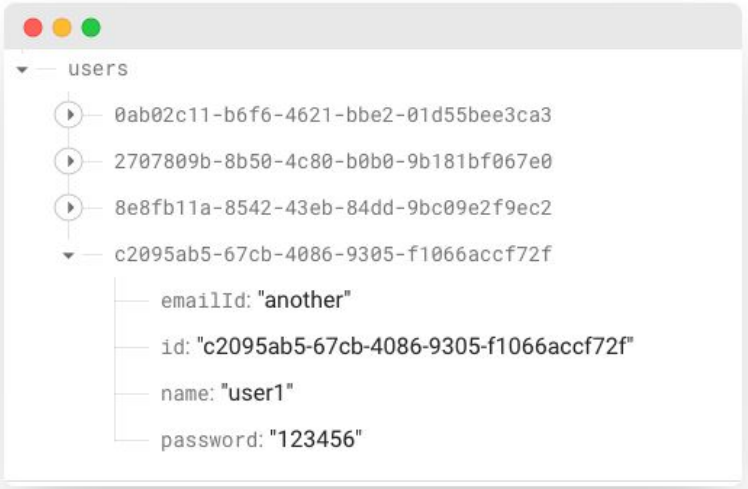
1. Pengecekkan bahwa semua data harus terisi.
2. Data yang dikirim ke database berbentuk object dengan schema, id, name, email Id dan password.
3. Id yang kamu kirim adalah hasil generate unique Id dari package react-native-uuid
4. Kamu menuliskan data pada database ke dalam field /users/

```
const onRegisterWithRDB = async () => {
    if (name == '' || email == '' || password == '') {
        Alert.alert('Error', 'Harap isi Semua field')
        return false;
    }
    let data = {
        id: uuid.v4(),
        name: name,
        emailId: email,
        password: password,
    };
    try {
        database()
            .ref('/users/' + data.id)
            .set(data)
            .then(() => {
                Alert.alert('Success', 'Register Successfully!'
                setEmail('')
                setPassword('')
                props.navigation.navigate('Login')
            });
    } catch (error) {
        Alert.alert('Error', error)
    }
}
```

Nah, begini kalau register berhasil~

Cobalah function berikut ini.

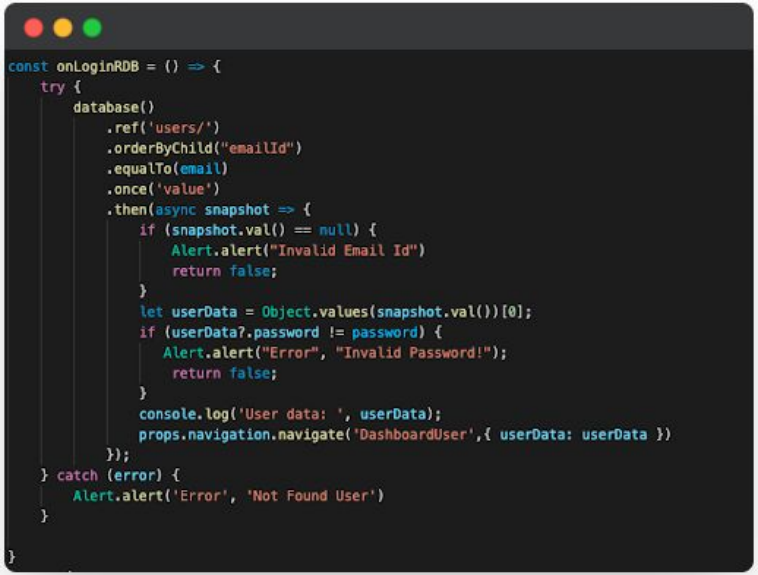
Setelah itu cek di database-mu. Seharusnya sudah tertulis seperti gambar di samping (sudah ada 4 data users yang dituliskan).



Login With Real Time Database

Selanjutnya adalah membuat function, di mana kamu akan **menulis data ke database setiap kali login** dan **melakukan read apabila data tersebut ada**.

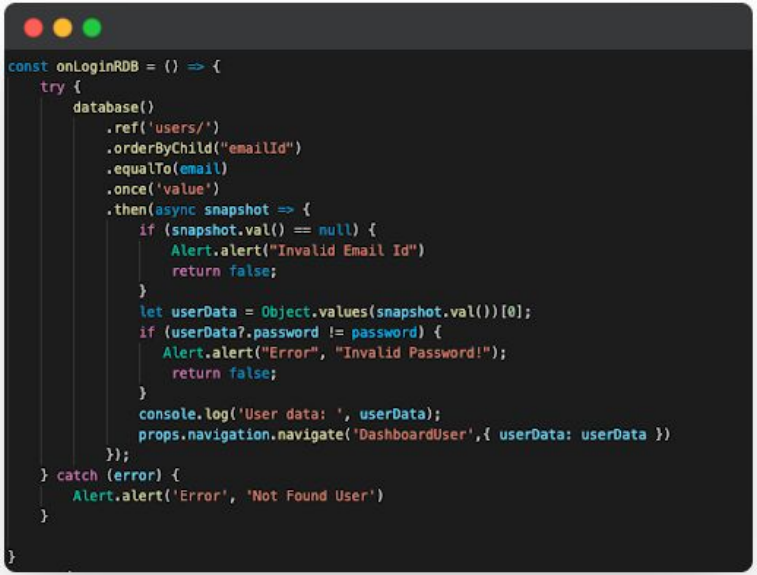
Contoh pada code di samping adalah function bernama onLoginWithRDB.



```
const onLoginRDB = () => {
    try {
        database()
            .ref('users/')
            .orderByChild("emailId")
            .equalTo(email)
            .once('value')
            .then(async snapshot => {
                if (snapshot.val() == null) {
                    Alert.alert("Invalid Email Id")
                    return false;
                }
                let userData = Object.values(snapshot.val())[0];
                if (userData?.password != password) {
                   Alert.alert("Error", "Invalid Password!");
                    return false;
                }
                console.log('User data: ', userData);
                props.navigation.navigate('DashboardUser',{ userData: userData })
            });
    } catch (error) {
        Alert.alert('Error', 'Not Found User')
    }

}
```

Singkatnya, tahapan function tersebut terdiri dari :

1. Memilih (select) tabel mana yang kamu ingin pilih.
2. Mengurutkan dari child yang bernama email id.
3. Melakukan pengecekan apakah email id yang dikirimkan terdapat pada tabel users-mu dengan child bernama email id.
4. Jika berhasil, kamu akan mengarah ke dashboard user.



```
const onLoginRDB = () => {
    try {
        database()
            .ref('users/')
            .orderByChild("emailId")
            .equalTo(email)
            .once('value')
            .then(async snapshot => {
                if (snapshot.val() == null) {
                    Alert.alert("Invalid Email Id")
                    return false;
                }
                let userData = Object.values(snapshot.val())[0];
                if (userData?.password != password) {
                   Alert.alert("Error", "Invalid Password!");
                    return false;
                }
                console.log('User data: ', userData);
                props.navigation.navigate('DashboardUser',{ userData: userData })
            });
    } catch (error) {
        Alert.alert('Error', 'Not Found User')
    }

}
```

Naaah, setelah realtime database beres, sekarang kamu akan merancang tampilan aplikasi chat untuk user.

Cekidot~

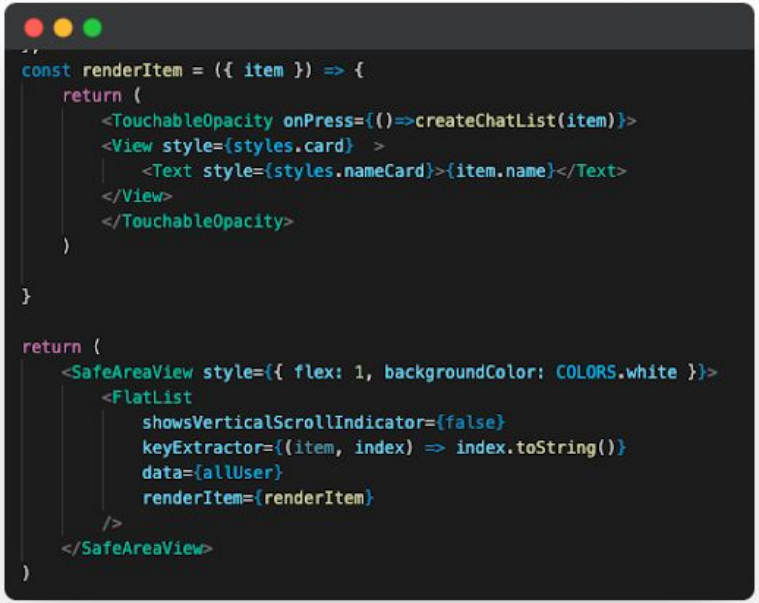
Bikin dulu nih, Dashboard User

Tahapan pertama dari proses merancang tampilan aplikasi chatting adalah **membuat screen dashboard user**.

Lewat dashboard ini, kamu dapat melihat dan memilih user apa saja yang tersedia untuk melakukan obrolan. Contoh tampilan sederhana yang akan kamu buat adalah seperti di samping.

Begini langkah-langkahnya:

Pertama, buatlah component sederhana menampilkan list user pada file baru DashboardUser.js



```
const renderItem = ({ item }) => {
    return (
        <TouchableOpacity onPress={()=>createChatList(item)}>
        <View style={styles.card}  >
            <Text style={styles.nameCard}>{item.name}</Text>
        </View>
        </TouchableOpacity>
    )

}

return (
    <SafeAreaView style={{ flex: 1, backgroundColor: COLORS.white }}>
        <FlatList
            showsVerticalScrollIndicator={false}
            keyExtractor={(item, index) => index.toString()}
            data={allUser}
            renderItem={renderItem}
        />
    </SafeAreaView>
)
```

Kedua, buatlah function dimana kamu akan **mengambil data user dari database app mu.** Kemudian simpan di variabel aplikasimu, contoh pada getAllUser function.

```
useEffect(() => {
    getAllUser();
}, []);


const getAllUser = () => {
    database()
        .ref('users/')
        .once('value')
        .then(snapshot => {
            console.log('all User data: ', Object.values(snapshot.val()));
            setallUser(
                Object.values(snapshot.val()).filter(it => it.id != userData.id),
            );
            setallUserBackup(
                Object.values(snapshot.val()).filter(it => it.id != userData.id),
            );
        });
};
```

Singkatnya, tahapan function tersebut terdiri dari :

1. Pilih file users dengan .ref() function
2. Gunakan .once() function yang artinya untuk setiap data yang kita terima kita eksekusi dengan function berikutnya.
3. Setiap data yang diterima, kamu akan filter atau buang data user yang userData.id sama dengan userData login di app mu. Sehingga pada dashboard user ini tidak menampilkan username account yang sudah login.

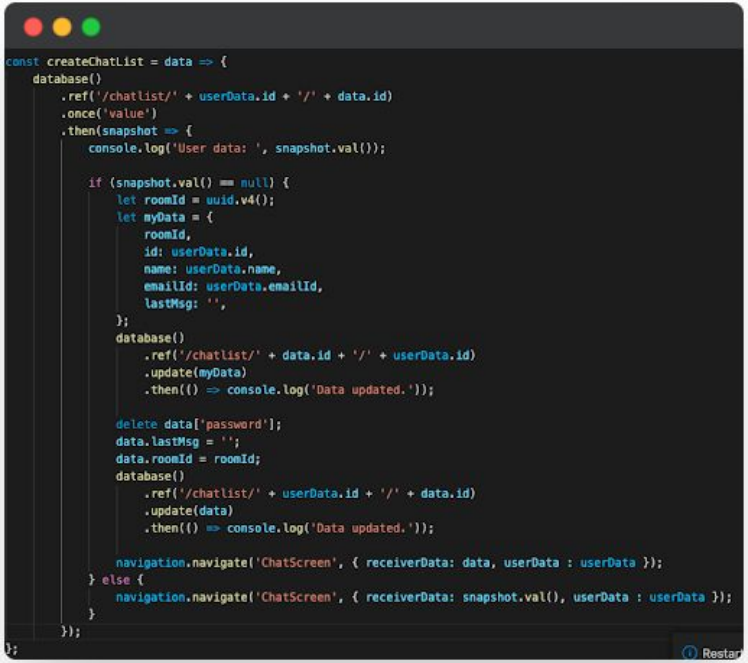
```
useEffect(() => {
    getAllUser();
}, []);


const getAllUser = () => {
    database()
        .ref('users/')
        .once('value')
        .then(snapshot => {
            console.log('all User data: ', Object.values(snapshot.val()));
            setallUser(
                Object.values(snapshot.val()).filter(it => it.id != userData.id),
            );
            setallUserBackup(
                Object.values(snapshot.val()).filter(it => it.id != userData.id),
            );
        });
};
```

Ketiga, buatlah function dimana ketika p**engguna mengklik salah satu dari list chat (user), aplikasi akan membuat room chat baru** sehingga pengguna dapat melakukan chat di screen baru tersebut. Seperti contoh pada function di samping, bernama createChatList()

```
const createChatList = data => {
    database()
        .ref('/chatlist/' + userData.id + '/' + data.id)
        .once('value')
        .then(snapshot => {
            console.log('User data: ', snapshot.val());

            if (snapshot.val() == null) {
                let roomId = uuid.v4();
                let myData = {
                    roomId,
                    id: userData.id,
                    name: userData.name,
                    emailId: userData.emailId,
                    lastMsg: '',
                };
                database()
                    .ref('/chatlist/' + data.id + '/' + userData.id)
                    .update(myData)
                    .then(() => console.log('Data updated.'));

                delete data['password'];
                data.lastMsg = '';
                data.roomId = roomId;
                database()
                    .ref('/chatlist/' + userData.id + '/' + data.id)
                    .update(data)
                    .then(() => console.log('Data updated.'));

                navigation.navigate('ChatScreen', { receiverData: data, userData : userData });
            } else {
                navigation.navigate('ChatScreen', { receiverData: snapshot.val(), userData : userData });
            }
        });
};
```

Singkatnya, tahapan function tersebut terdiri dari :

1. Tambahkan field baru dengan nama field /chatlist/
2. Setiap chatlist memiliki userData id yang di dalamnya ada data id, dimana data id memiliki room id
3. Setiap pengguna membuka room id dengan user tertentu, app kamu akan mengupdate dan mengubah data dengan data id yang baru seperti pada code di samping.



```
const createChatList = data => {
    database()
        .ref('/chatlist/' + userData.id + '/' + data.id)
        .once('value')
        .then(snapshot => {
            console.log('User data: ', snapshot.val());

            if (snapshot.val() == null) {
                let roomId = uuid.v4();
                let myData = {
                    roomId,
                    id: userData.id,
                    name: userData.name,
                    emailId: userData.emailId,
                    lastMsg: '',
                };
                database()
                    .ref('/chatlist/' + data.id + '/' + userData.id)
                    .update(myData)
                    .then(() => console.log('Data updated.'));

                delete data['password'];
                data.lastMsg = '';
                data.roomId = roomId;
                database()
                    .ref('/chatlist/' + userData.id + '/' + data.id)
                    .update(data)
                    .then(() => console.log('Data updated.'));

                navigation.navigate('ChatScreen', { receiverData: data, userData : userData });
            } else {
                navigation.navigate('ChatScreen', { receiverData: snapshot.val(), userData : userData });
            }
        });
};
```

Kira-kira kayak gini nih database akan tergenerate. Dari gambar di samping, kamu bisa memiliki field chatlist yang punya id untuk menyimpan room Id.

Next, membuat chat screen!

Setelah dashboard jadi, selanjutnya buatlah chat screen component, lalu tambahkan file baru bernama chatscreen.js. Komponen ini akan merender kegiatan room chat seperti pada contoh code di bawah ini!

```
return (
    <View style={styles.container}>
        <ChatHeader data={receiverData} />
        <ImageBackground
            source={require('../../Assets/background2.jpeg')}
            style={{ flex: 1 }}>
            <FlatList
                style={{ flex: 1 }}
                data={allChat}
                showsVerticalScrollIndicator={false}
                keyExtractor={(item, index) => index}
                inverted
                renderItem={({ item }) => {
                    return (
                        <MsgComponent
                            sender={item.from == userData.id}
                            item={item}
                        />
                    );
                }}
            />
        </ImageBackground>
        <View
            style={{
                backgroundColor: COLORS.theme,
                elevation: 5,
                // height: 60,
                flexDirection: 'row',
                alignItems: 'center',
                paddingVertical: 7,
                justifyContent: 'space-evenly',
                paddingBottom : 20
            }}>
```

```
                <TextInput
                    style={{
                        backgroundColor: COLORS.white,
                        width: '80%',
                        borderRadius: 25,
                        borderWidth: 0.5,
                        borderColor: COLORS.white,
                        paddingHorizontal: 15,
                        color: COLORS.black,
                        paddingBottom : 10
                    }}
                    placeholder="type a message"
                    placeholderTextColor={COLORS.black}
                    multiline={true}
                    value={msg}
                    onChangeText={val => setMsg(val)}
                />

                <TouchableOpacity disabled={disabled} onPress={()=>sendMsg()}>
                    <Text style={{ color: COLORS.white }}>Send!</Text>
                    {/* <Icon
                        style={{
                            // marginHorizontal: 15,
                            color: COLORS.white,
                        }}
                        name="paper-plane-sharp"
                        type="Ionicons"
                    /> */}
                </TouchableOpacity>
            </View>
        </View>
    );
```

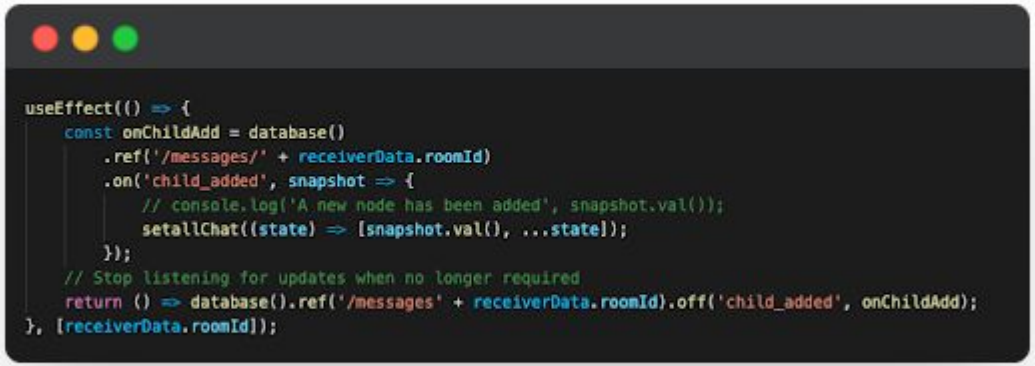
Penting nih, guys~

Buatlah 2 function di mana **ketika pengguna membuka chat screen, akan ada 2 kemungkinan.**

Kemungkinan pertama, kamu akan mendapatkan history chat sebelumnya (jika ada),kalau nggak ada, maka akan menampilkan chat room kosong, kemudian function untuk melakukan sending messages.

Singkatnya, tahapan function tersebut terdiri dari :

1. Kamu akan mengambil messages berdasarkan dari room id.
2. Kemudian hasil message tersebut akan kamu simpan ke dalam state.



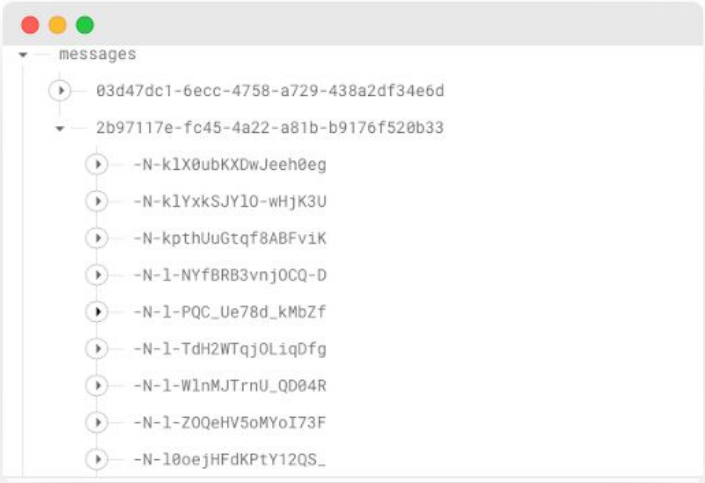
```
useEffect(() => {
    const onChildAdd = database()
        .ref('/messages/' + receiverData.roomId)
        .on('child_added', snapshot => {
            // console.log('A new node has been added', snapshot.val());
            setallChat((state) => [snapshot.val(), ...state]);
        });
    // Stop listening for updates when no longer required
    return () => database().ref('/messages' + receiverData.roomId).off('child_added', onChildAdd);
}, [receiverData.roomId]);
```

Singkatnya, tahapan function tersebut terdiri dari :

1. Melakukan pengecekan apakah coding sudah lengkap atau belum.
2. Mengirimkan beberapa informasi dari messages yang diperlukan akan dibungkus dalam variable msgData.
3. Menambahkan message yang baru ke dalam database/messages berdasarkan room Id dari recevier (penerima pesan).
4. Selanjutnya pada chat list kamu akan mengupdate dengan chatlist update, sehingga kamu dapat mengedit last message dan menambah message baru.

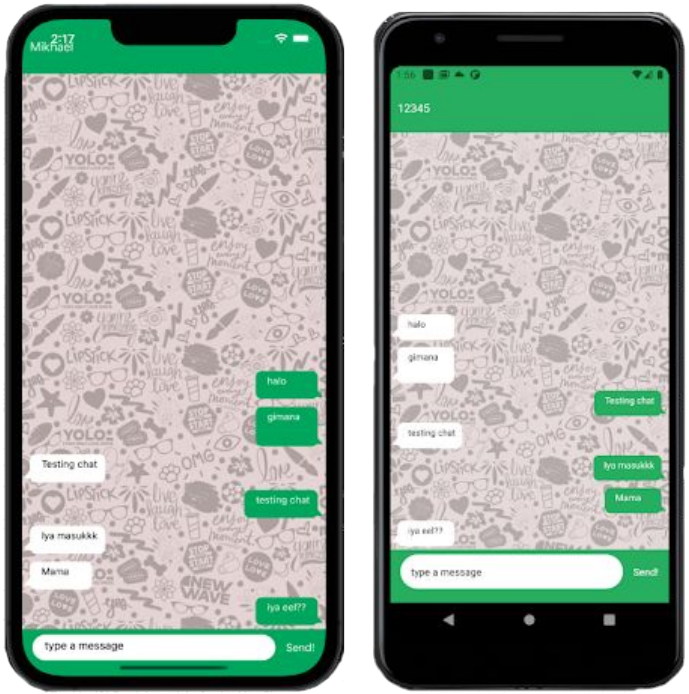
```
const msgvalid = txt => txt && txt.replace(/\s/g, '').length;
const sendMsg = () => {
    if (msg == '' || msgvalid(msg) == 0) {
        SimpleToast.show('Enter something....');
        return false;
    }
    setdisabled(true);
    let msgData = {
        roomId: receiverData.roomId,
        message: msg,
        from: userData?.id,
        to: receiverData.id,
        sendTime: moment().format(''),
        msgType: 'text',
    };

    const newReference = database()
        .ref('/messages/' + receiverData.roomId)
        .push();
    msgData.id = newReference.key;
    newReference.set(msgData).then(() => {
        let chatListupdate = {
            lastMsg: msg,
            sendTime: msgData.sendTime,
        };
        database()
            .ref('/chatlist/' + receiverData?.id + '/' + userData?.id)
            .update(chatListupdate)
            .then(() => console.log('Data updated.'));
        console.log("'/chatlist/' + userData?.id + '/' + data?.id", receiverData)
        database()
            .ref('/chatlist/' + userData?.id + '/' + receiverData?.id)
            .update(chatListupdate)
            .then(() => console.log('Data updated.'));

        setMsg('');
        setdisabled(false);
    });
};
```

Perhatikan database setelah kamu memanggil function send messages. Di situ kamu bisa lihat **tiap messages memiliki id messages**, di mana di dalamnya memiliki dari room id messages.

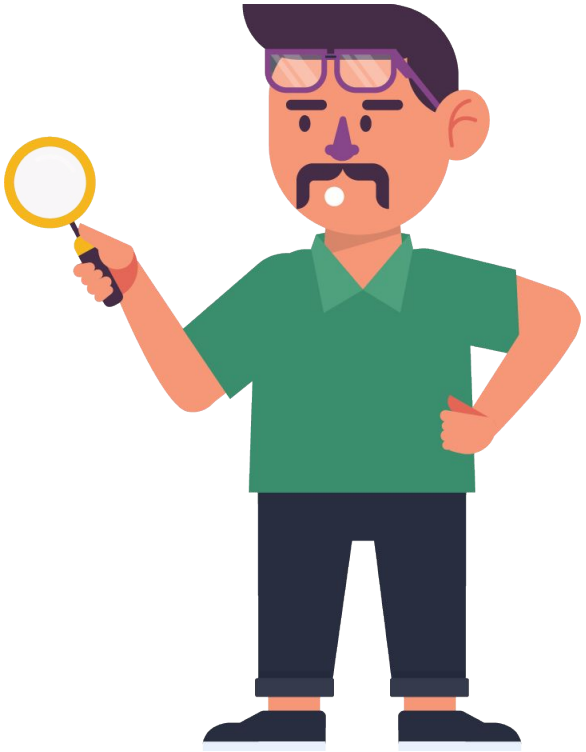


Saatnya membuat Chat Screen!

Selanjutnya, bersama teman kelompok, cobalah untuk membuat dua akun dari source code yang sama dan cobalah untuk melakukan chatting seperti gambar di samping.



Kalau kamu mengalami kesulitan untuk melihat isi function dan code, silakan melihat code di repo ini ya!



Saatnya kita

Quiz!

BINAR

1. Mengapa Chatting App menjadi salah satu bentuk implementasi Real Time Database?

A. Security firebase lebih baik dibanding database SQL

B. Firebase mensupport untuk melakukan penyimpanan data

C. Karena sifatnya yang real time sehingga data dapat diakses secara realtime

BINAR

2. Manakah function berikut yang berguna untuk merubah data pada database?

A. database().ref().update().

B. database().ref().read

C. database().ref().write

BINAR

3. Apa yang bisa kita gunakan untuk mengatur jam pengiriman pesan?

A. Menggunakan uuid

B. Menggunakan parameter untuk set roomID

C. Menggunakan moment untuk mendapatkan current time

BINAR

4. Dari hasil aplikasi yang kita buat, ada berapa field yang terbentuk di database?

A. 4

B. 3

C. 5

4. Berikut adalah contoh penerapan real time database yang benar pada fitur aplikasi chatting yang dibuat, kecuali...

A. Fitur Login

B. Fitur Chatting Screen

C. Fitur Dashboard User

Terima Kasih!

Chapter ✓

completed

BINAR