

# Mobile Application & Development

## LOGBOOK

Ngo Hoang Thanh

COMP1786

GCD18591

### Table of Content

I. Create a PhoneGap App or any other similar platform utilizing the Notification API .....	2
1. Basic information .....	2
2. Exercise Answer: .....	3
2.1. Screenshots demonstrating what is achieved: .....	3
2.2. Code and explanation .....	4
II. Create React Native App data entry screen.....	20
1. Basic information .....	20
2. Exercise Answer .....	20
2.1. Screenshots demonstrating what is achieved .....	20
2.2 Code and explanation .....	26
III. Create Android data entry screen.....	42
1. Basic information .....	42
2. Exercise answer:.....	42
2.1. Screenshots demonstrating what is achieved: .....	42
2.2. Code and Explanation .....	46
IV. Conclusion.....	69

### Table of Figure

Figure 1: Confirmation Dialog Box .....	4
Figure 2: Form Input Screen.....	21
Figure 3: Error Required Select .....	22

Figure 4: Choose Date Dialog.....	23
Figure 5: Error Required Input .....	24
Figure 6: Some other errors.....	24
Figure 7: Validation successful.....	25
Figure 8: Check Information.....	26
Figure 9: Submit successful.....	26
Figure 10: Form input Screen.....	43
Figure 11: Select drop down .....	44
Figure 12: Date.....	44
Figure 13: Choose Date dialog .....	44
Figure 14: Poster error .....	45
Figure 15: Price error .....	45
Figure 16: Note error .....	45
Figure 17: Validation Successful .....	46

## I. Create a PhoneGap App or any other similar platform utilizing the Notification API

### 1. Basic information

1.1 Student name	Ngo Hoang Thanh
1.2 Who did you work with? Note that for logbook exercises you are allowed to work with one other person as long as you give their name and login id and both contribute to the work.	I just do it alone
1.3 Which Exercise is this? Tick as appropriate.	Create a PhoneGap App or any other similar platform utilizing the Notification API.
1.4 How well did you complete the exercise? Tick as appropriate.	I tried but couldn't complete it <input type="checkbox"/> I did it but I feel I should have done better <input type="checkbox"/> I did everything that was asked <input type="checkbox"/> I did more than was asked for <input checked="" type="checkbox"/>
1.5 Briefly explain your answer to question	I have created a react native app that shows a confirmation dialog with 2 buttons. One button to ring the bell and another button to vibrate.

## 2. Exercise Answer:

### 2.1. Screenshots demonstrating what is achieved:

This section has a fairly simple interface, only one button "Show Dialog Box" and it will display a dialog box displaying 2 functions, "Ring" and "Vibrate". If user want to play music press "Ring" and if user want to vibrate phone press "Vibrate"

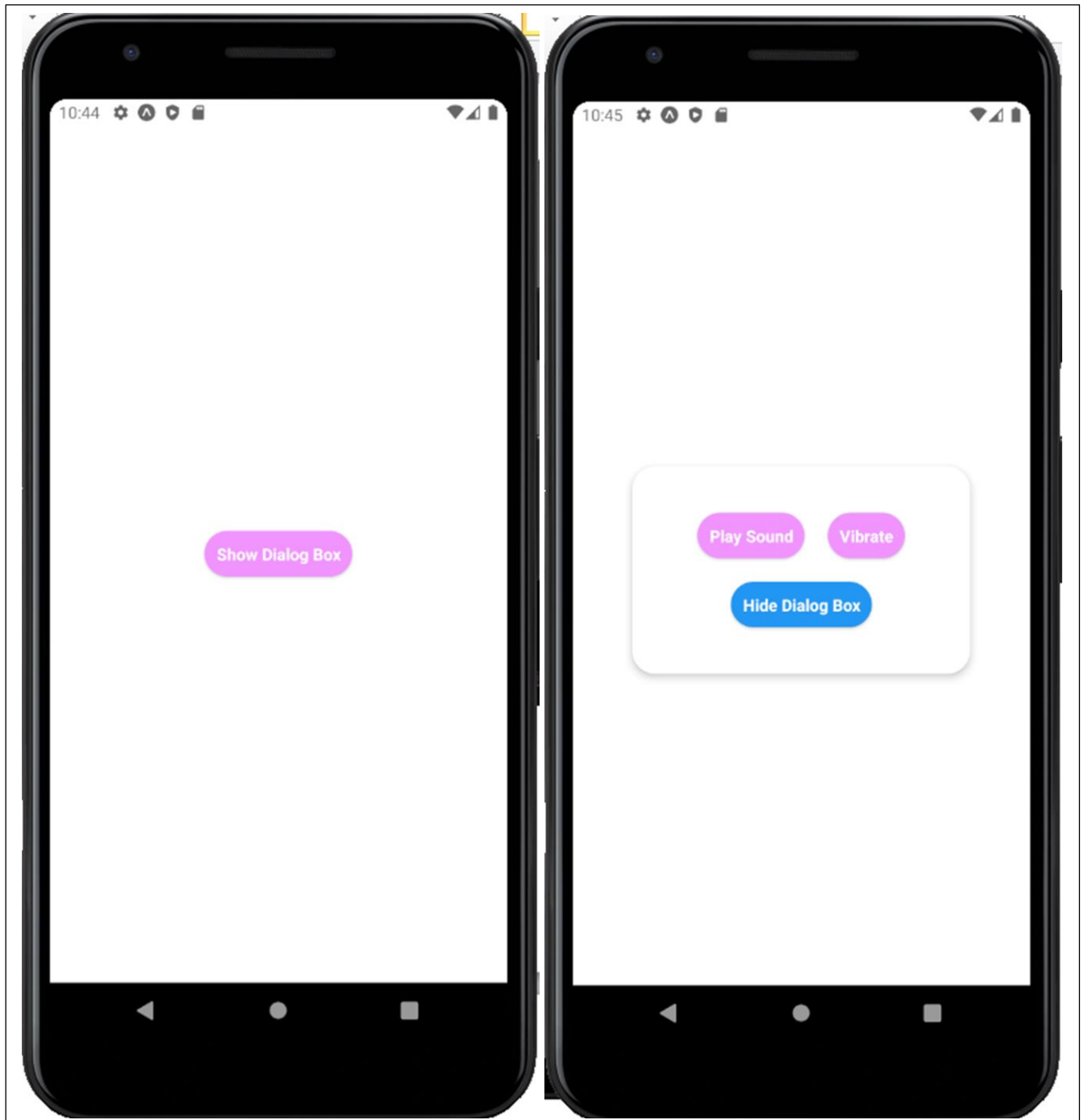


Figure 1: Confirmation Dialog Box

## 2.2. Code and explanation

### 2.2.1. Code

This is code of create react native app utilizing the Notification API:

#### ring.js

This is the file that contains the code that implements the "Ring" and "Vibrate" functions for the application.

```
import { styles } from './styles';
import * as React from 'react';
import { Text, View, StyleSheet, Button, Pressable, Alert, Vibration } from 'react-native';
import { Audio } from 'expo-av';

const RingAudio = () => {

  const [sound, setSound] = React.useState();
  const [count, setCount] = React.useState(0);
  const [vibrate, setVibrate] = React.useState(false);
  var time = 5000;

  const timerRef = React.useRef(null);
  React.useEffect(() => {
    if (count % 2 == 1){
      timerRef.current = setTimeout(() => {
        sound.unloadAsync();
        setSound();
      }, 7000);
    }
    setCount(count + 1)
    console.log(count)
    return () => {
      if (timerRef.current) {
        clearTimeout(timerRef.current);
      }
    };
  }, [sound]);

  async function playSound() {
```

```

const { sound } = await Audio.Sound.createAsync(
  require('./hello.mp3')
);
setSound(sound);
await sound.playAsync();
}

function pauseSound(){
  sound.unloadAsync();
  setSound();
}

function vibra(){
  setVibrate(true)
  Vibration.vibrate(time);
  setTimeout(() => {
    setVibrate(false)
  }, time);
}

return (
  <View style={styles.containerRing}>
    <View style={styles.fixToText}>
      <Pressable
        style={[styles.button, styles.buttonRing]}
        onPress={sound === undefined ? playSound : pauseSound}
      >
        <Text style={styles.textStyle}>{sound === undefined ? "Play Sound"
: "Pause Sound"}</Text>
      </Pressable>

      <Pressable
        style={[styles.button, styles.buttonVibrate]}
        onPress={vibra}
      >
        <Text style={styles.textStyle}>{vibrate === false ? "Vibrate" :
"Vibrating"}</Text>
      </Pressable>

    </View>

  </View>
);
}

```

```
export default RingAudio;
```

## modal.js

This is the file that contains the code that implements show and hide dialog box function

```
import React, { useState } from "react";
import { Alert, Modal, StyleSheet, Text, Pressable, View } from "react-native";
import { styles } from "../styles";
import RingAudio from '../ring.js';

const ModalNoti = () => {
  const [modalVisible, setModalVisible] = useState(false);
  return (
    <View style={styles.centeredView}>
      <Modal
        animationType="slide"
        transparent={true}
        visible={modalVisible}
        onRequestClose={() => {
          Alert.alert("Modal has been closed.");
          setModalVisible(!modalVisible);
        }}
      >
        <View style={styles.centeredView}>
          <View style={styles.modalView}>
            <RingAudio/>
            <Pressable
              style={[styles.button, styles.buttonClose]}
              onPress={() => setModalVisible(!modalVisible)}
            >
              <Text style={styles.textStyle}>Hide Dialog Box</Text>
            </Pressable>
          </View>
        </View>
      </Modal>
      <Pressable
        style={[styles.button, styles.buttonOpen]}
        onPress={() => setModalVisible(true)}
      >
```

```

        <Text style={styles.textStyle}>Show Dialog Box</Text>
      </Pressable>
    </View>
  );
};

export default ModalNoti;

```

## styles.js

This is the file containing the CSS code for the above files

```

import { StyleSheet, Platform } from "react-native";

const HEADER_BACKGROUND = "#3498db";
const CONTENT_BACKGROUND = "#f9f9f9";
export const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: CONTENT_BACKGROUND,
  },
  topSafeArea: {
    backgroundColor: HEADER_BACKGROUND,
  },
  header: {
    height: 60,
    justifyContent: "center",
    alignItems: "center",
    backgroundColor: HEADER_BACKGROUND,
    paddingTop: 50,
    paddingBottom: 30,
  },
  headerText: {
    color: "#fff",
    fontSize: 20,
  },
  content: {
    padding: 20,
    backgroundColor: CONTENT_BACKGROUND,
  },
  formGroup: {
    marginBottom: 10,
  },
});

```

```
label: {
  color: "#7d7e79",
  fontSize: 16,
  lineHeight: 30,
  paddingBottom: 10
},
input: {
  height: 50,
  paddingHorizontal: 20,
  borderRadius: 5,
  borderWidth: 2,
  borderColor: "#e3e3e3",
  backgroundColor: "#fff",
},
inputNote: {
  paddingHorizontal: 20,
  borderRadius: 5,
  borderWidth: 2,
  borderColor: "#e3e3e3",
  backgroundColor: "#fff",
},
buttonSubmit: {
  marginTop: 20,
  backgroundColor: "#2980b9",
  padding: 15,
  borderRadius: 15,
},
buttonText: {
  color: "#fff",
  fontWeight: "bold",
  fontSize: 18,
  textAlign: "center",
},
errorContainer: {
  marginVertical: 5,
},
errorText: {
  color: "#ff7675",
},
pickedDateContainer: {
  padding: 20,
  backgroundColor: '#eee',
  borderRadius: 10,
},
pickedDate: {
```



```
    fontSize: 18,
    color: 'black',
  },

  btnContainer: {
    padding: 30,
  },
  datePicker: {
    width: 320,
    height: 260,
    display: 'flex',
    justifyContent: 'center',
    alignItems: 'flex-start',
  },
  containerPicker: {
    paddingTop: 10,
  },
  centeredView: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
    marginTop: 22
  },
  modalView: {
    margin: 20,
    backgroundColor: "white",
    borderRadius: 20,
    padding: 40,
    alignItems: "center",
    shadowColor: "#000",
    shadowOffset: {
      width: 0,
      height: 2
    },
    shadowOpacity: 0.25,
    shadowRadius: 4,
    elevation: 5
  },
  button: {
    borderRadius: 20,
    padding: 10,
    elevation: 2
  },
  buttonOpen: {
    backgroundColor: "#F194FF",
```

```
},
buttonRing: {
  backgroundColor: "#F194FF",
  marginRight: 20,
},
buttonVibrate: {
  backgroundColor: "#F194FF",
},
buttonClose: {
  backgroundColor: "#2196F3",
},
textStyle: {
  color: "white",
  fontWeight: "bold",
  textAlign: "center"
},
modalText: {
  marginBottom: 15,
  textAlign: "center"
},
containerRing: {
  justifyContent: 'center',
  marginHorizontal: 16,
  marginBottom: 20,
},
fixToText: {
  flexDirection: 'row',
  justifyContent: 'space-between',
},
spacer: {
  paddingBottom: 50
},
requireName: {
  position: 'absolute',
  left: 48,
  fontSize: 20,
  color: 'red',
},
requirePrice: {
  position: 'absolute',
  left: 42,
  fontSize: 20,
  color: 'red',
},
requirePoster: {
```

```

        position: 'absolute',
        left: 52,
        fontSize: 20,
        color: 'red',
      },
      requireProp:{
        position: 'absolute',
        left: 103,
        fontSize: 20,
        color: 'red',
      },
      requireBed:{
        position: 'absolute',
        left: 81,
        fontSize: 20,
        color: 'red',
      },
      requireLocation:{
        position: 'absolute',
        left: 67,
        fontSize: 20,
        color: 'red',
      },
    },
  })

```

### 2.2.2. Explanation

#### ring.js

For the app to ring a bell, I need to use the library expo-av.

```
import { Audio } from 'expo-av';
```

First, I will declare state variables for later processing.

```

const [sound, setSound] = React.useState();
const [vibrate, setVibrate] = React.useState(false);
const [count, setCount] = React.useState(0);

```

For example, in the screen display, I will check the sound variable if its value is undefined, then when pressed it will run the playSound function and its displayed text is "Play Sound", otherwise, the pauseSound function will be run and the text will be displayed. Its market is "Pause Sound". Similar to vibrate button

```

    <Pressable
      style={[styles.button, styles.buttonRing]}
      onPress={sound === undefined ? playSound : pauseSound}
    >
      <Text style={styles.textStyle}>{sound === undefined ? "Play Sound"
: "Pause Sound"}</Text>
    </Pressable>

    <Pressable
      style={[styles.button, styles.buttonVibrate]}
      onPress={vibra}
    >
      <Text style={styles.textStyle}>{vibrate === false ? "Vibrate" :
"Vibrating"}</Text>
    </Pressable>

```

This is a function that automatically turns off the music for a specified time if the user does not press to stop the music. I have used useRef, useEffect, setTimeout and clearTimeout along with logic handlers to make this functionality.

```

const [count, setCount] = React.useState(0);
const timerRef = React.useRef(null);
React.useEffect(() => {
  if (count % 2 == 1){
    timerRef.current = setTimeout(() => {
      sound.unloadAsync();
      setSound();
    }, 7000);
  }
  setCount(count + 1)
  console.log(count)
  return () => {
    if (timerRef.current) {
      clearTimeout(timerRef.current);
    }
  };
}, [sound]);

```

This is the playSound function used to turn on the music and pauseSound function to turn off the music anytime you want. In the playSound function I have created and loaded a new sound by path then I will assign that value to the sound variable finally run the playAsync function to play the song. As for the pauseSound function, I simply run the unloadAsync function to stop the song and set the value of the sound variable to undefined

```

async function playSound() {
  const { sound } = await Audio.Sound.createAsync(
    require('./hello.mp3')
  );
  setSound(sound);
  await sound.playAsync();
}

function pauseSound(){
  sound.unloadAsync();
  setSound();
}

```

Next comes the Vibration function, it is already provided by the react-native library you just need to import to use.

```

import { Text, View, StyleSheet, Button, Pressable, Alert, Vibration } from 'react-native';

```

In the vibra function I will run the function Vibration.vibrate and assign the time value to it as a pre-created time variable, specifically 5s here. Also while performing the vibrate function, I will set the value of the variable vibrate to true to notify the user that the vibrate function is being performed and when the vibrate function is finished, I will return its value return to false with the setTimeout function

```

var time = 5000;
function vibra(){
  Vibration.vibrate(time);
  setVibrate(true)
  setTimeout(() => {
    setVibrate(false)
  }, time);
}

```

This is a code showing 2 buttons to perform 2 functions “Ring” and “Vibrate” for the user.

```

return (
  <View style={styles.containerRing}>
    <View style={styles.fixToText}>
      <Pressable
        style={[styles.button, styles.buttonRing]}
        onPress={sound === undefined ? playSound : pauseSound}
      >

```

```

        <Text style={styles.textStyle}>{sound === undefined ? "Play Sound"
: "Pause Sound"}</Text>
      </Pressable>

      <Pressable
        style={[styles.button, styles.buttonVibrate]}
        onPress={vibra}
      >
        <Text style={styles.textStyle}>{vibrate === false ? "Vibrate" :
"Vibrating"}</Text>
      </Pressable>

    </View>

  </View>
);

```

### modal.js

This file uses the modal property provided in the react-native library to perform the show and hide dialog box functionality. At the same time also import 2 function buttons "Ring" and "Vibrate" from ring.js file to display with dialog box on screen for users.

```

import React, { useState } from "react";
import { Alert, Modal, StyleSheet, Text, Pressable, View } from "react-native";
import { styles } from "../styles";
import RingAudio from '../ring.js';

const ModalNoti = () => {
  const [modalVisible, setModalVisible] = useState(false);
  return (
    <View style={styles.centeredView}>
      <Modal
        animationType="slide"
        transparent={true}
        visible={modalVisible}
        onRequestClose={() => {
          Alert.alert("Modal has been closed.");
          setModalVisible(!modalVisible);
        }}
      >
        <View style={styles.centeredView}>

```

```

    <View style={styles.modalView}>
      <RingAudio/>
      <Pressable
        style={[styles.button, styles.buttonClose]}
        onPress={() => setModalVisible(!modalVisible)}
      >
        <Text style={styles.textStyle}>Hide Dialog Box</Text>
      </Pressable>
    </View>
  </View>
</Modal>
<Pressable
  style={[styles.button, styles.buttonOpen]}
  onPress={() => setModalVisible(true)}
>
  <Text style={styles.textStyle}>Show Dialog Box</Text>
</Pressable>
</View>
);
};

export default ModalNoti;

```

## style.js

As mentioned above, this file only performs CSS functions

```

import { StyleSheet, Platform } from "react-native";

const HEADER_BACKGROUND = "#3498db";
const CONTENT_BACKGROUND = "#f9f9f9";
export const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: CONTENT_BACKGROUND,
  },
  topSafeArea: {
    backgroundColor: HEADER_BACKGROUND,
  },
  header: {
    height: 60,
    justifyContent: "center",
    alignItems: "center",
    backgroundColor: HEADER_BACKGROUND,
  },
});

```

```
paddingTop: 50,
paddingBottom: 30,
},
headerText: {
  color: "#fff",
  fontSize: 20,
},
content: {
  padding: 20,
  backgroundColor: CONTENT_BACKGROUND,
},
formGroup: {
  marginBottom: 10,
},
label: {
  color: "#7d7e79",
  fontSize: 16,
  lineHeight: 30,
  paddingBottom: 10
},
input: {
  height: 50,
  paddingHorizontal: 20,
  borderRadius: 5,
  borderWidth: 2,
  borderColor: "#e3e3e3",
  backgroundColor: "#fff",
},
inputNote: {
  paddingHorizontal: 20,
  borderRadius: 5,
  borderWidth: 2,
  borderColor: "#e3e3e3",
  backgroundColor: "#fff",
},
buttonSubmit: {
  marginTop: 20,
  backgroundColor: "#2980b9",
  padding: 15,
  borderRadius: 15,
},
buttonText: {
  color: "#fff",
  fontWeight: "bold",
```



```
        fontSize: 18,
        textAlign: "center",
    },
    errorContainer: {
        marginVertical: 5,
    },
    errorText: {
        color: "#ff7675",
    },
    pickedDateContainer: {
        padding: 20,
        backgroundColor: '#eee',
        borderRadius: 10,
    },
    pickedDate: {
        fontSize: 18,
        color: 'black',
    },
    },

    btnContainer: {
        padding: 30,
    },
    datePicker: {
        width: 320,
        height: 260,
        display: 'flex',
        justifyContent: 'center',
        alignItems: 'flex-start',
    },
    containerPicker: {
        paddingTop: 10,
    },
    centeredView: {
        flex: 1,
        justifyContent: "center",
        alignItems: "center",
        marginTop: 22
    },
    modalView: {
        margin: 20,
        backgroundColor: "white",
        borderRadius: 20,
        padding: 40,
        alignItems: "center",
        shadowColor: "#000",
```

```
        shadowOffset: {
          width: 0,
          height: 2
        },
        shadowOpacity: 0.25,
        shadowRadius: 4,
        elevation: 5
      },
      button: {
        borderRadius: 20,
        padding: 10,
        elevation: 2
      },
      buttonOpen: {
        backgroundColor: "#F194FF",
      },
      buttonRing: {
        backgroundColor: "#F194FF",
        marginRight: 20,
      },
      buttonVibrate: {
        backgroundColor: "#F194FF",
      },
      buttonClose: {
        backgroundColor: "#2196F3",
      },
      textStyle: {
        color: "white",
        fontWeight: "bold",
        textAlign: "center"
      },
      modalText: {
        marginBottom: 15,
        textAlign: "center"
      },
      containerRing: {
        justifyContent: 'center',
        marginHorizontal: 16,
        marginBottom: 20,
      },
      fixToText: {
        flexDirection: 'row',
        justifyContent: 'space-between',
      },
      spacer: {
```

```
        paddingBottom: 50
      },
      requireName:{
        position: 'absolute',
        left: 48,
        fontSize: 20,
        color: 'red',
      },
      requirePrice:{
        position: 'absolute',
        left: 42,
        fontSize: 20,
        color: 'red',
      },
      requirePoster:{
        position: 'absolute',
        left: 52,
        fontSize: 20,
        color: 'red',
      },
      requireProp:{
        position: 'absolute',
        left: 103,
        fontSize: 20,
        color: 'red',
      },
      requireBed:{
        position: 'absolute',
        left: 81,
        fontSize: 20,
        color: 'red',
      },
      requireLocation:{
        position: 'absolute',
        left: 67,
        fontSize: 20,
        color: 'red',
      },
    },
  })
})
```

## II. Create React Native App data entry screen

### 1. Basic information

1.1 Student name	Ngô Hoàng Thành
1.2 Who did you work with? Note that for logbook exercises you are allowed to work with one other person as long as you give their name and login id and both contribute to the work.	I just do it alone
1.3 Which Exercise is this? Tick as appropriate.	Create a React Native App data entry screen
1.4 How well did you complete the exercise? Tick as appropriate.	I tried but couldn't complete it <input type="checkbox"/> I did it but I feel I should have done better <input type="checkbox"/> I did everything that was asked <input type="checkbox"/> I did more than was asked for <input checked="" type="checkbox"/>
1.5 Briefly explain your answer to question	I have created a React Native App that allows the user to create a form from all the fields specified in lesson part 1 a). The application must do some data input validation and display an error message to the user if the data is invalid. If all the data is valid then it will be saved to the database

### 2. Exercise Answer

#### 2.1. Screenshots demonstrating what is achieved

This is the application's data entry form. Includes Property type, Bedrooms, Furniture type, Date Time, Poster, Price, Note. There are 2 fields, Furniture type and Note, where the user can not select or enter data, and all the remaining fields must be entered by the user and must be entered correctly otherwise the Submit button will be disabled and not sent data goes.

4:20

### Create new Post

Property type \*

Select Property Type

Bedrooms \*

Select Bedroom

Furniture Types

Select Furniture Type

11/16/2021

CHOOSE DATE

Poster \*

Enter The Poster's Name

Price \*

Enter Price

Note

Enter Your Note

4:20

### Create new Post

Select Bedroom

Furniture Types

Select Furniture Type

11/16/2021

CHOOSE DATE

Poster \*

Enter The Poster's Name

Price \*

Enter Price

Note

Enter Your Note

SUBMIT

Figure 2: Form Input Screen

For 2 required fields of select type, Property type and Bedrooms, if you just accessed the application and did nothing, it will look like the picture above, but if you have selected a certain attribute and then reselected the default value of it will appear a message asking you to choose

The screenshot shows a mobile application interface for creating a new post. The title bar is blue with the text "Create new Post". The form contains several fields:

- Property type \***: A dropdown menu with the placeholder text "Select Property Type". Below the dropdown, the error message "Please select this field" is displayed in red.
- Bedrooms \***: A dropdown menu with the placeholder text "Select Bedroom". Below the dropdown, the error message "Please select this field" is displayed in red.
- Furniture Types**: A dropdown menu with the placeholder text "Select Furniture Type".
- Date**: A date picker showing "11/16/2021" with a purple "CHOOSE DATE" button below it.
- Poster \***: A text input field with the placeholder text "Enter The Poster's Name". Below the input field, the error message "Please Enter This Field!" is displayed in red.
- Price \***: A text input field with the placeholder text "Enter Price".

The bottom of the screen shows the standard Android navigation bar with back, home, and recent apps buttons.

Figure 3: Error Required Select

For Date time, when you click the Choose Date button, a dialog box will appear for you to choose the date and the date you can only choose the date from today and earlier.

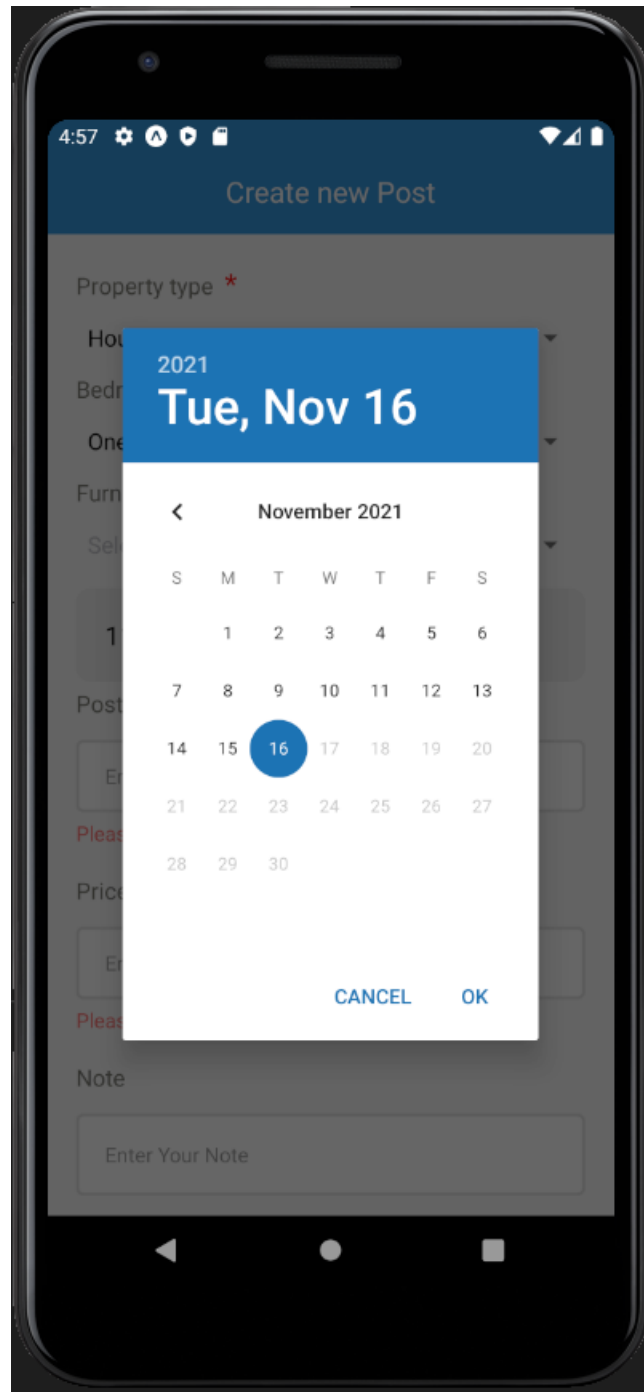
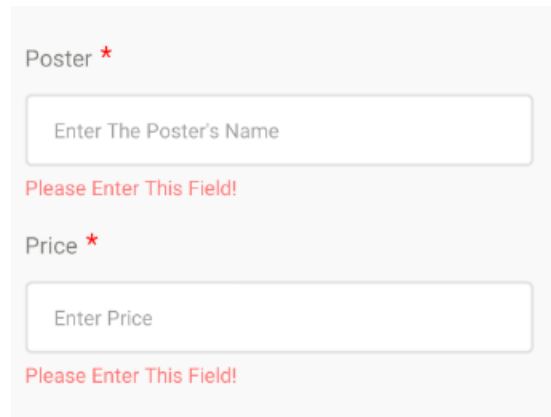


Figure 4: Choose Date Dialog

As for the two required input fields, Poster and Price, only the user clicks on the input box but does not enter data, a message will be displayed asking the user to enter data.



Poster \*

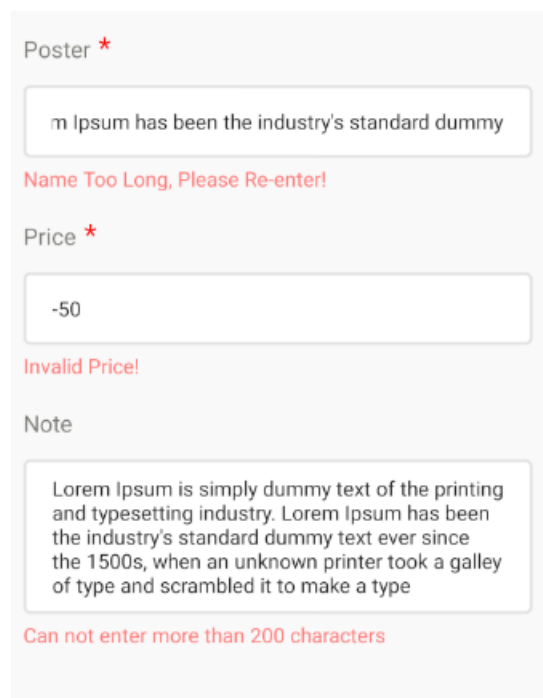
Please Enter This Field!

Price \*

Please Enter This Field!

Figure 5: Error Required Input

In addition to the required conditions for the input fields, I also added some validation conditions for them like with Poster, only allowing users to enter 100 characters or less, for Price, only numbers can be entered and must be large than 0, and for the note field, it only allows users to enter less than 200 characters. If the user violates, an error will appear to report.



Poster \*

Name Too Long, Please Re-enter!

Price \*

Invalid Price!

Note

Can not enter more than 200 characters

Figure 6: Some other errors

Once all the conditions are fulfilled and the validation is passed then the Submit button will be activated so that the user can submit the entered form.



5:16

Create new Post

Furniture Types

Select Furniture Type

11/16/2021

CHOOSE DATE

Poster \*

rinting and typesetting industry. Lorem Ipsum has

Price \*

50000

Note

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s

SUBMIT

Figure 7: Validation successful

When the user clicks the Submit button, the screen will display a message dialog box displaying all the information that the user has entered in each field so that he can check if he feels there is anything wrong, the user will click and Cancel button and go back to editing and if all data is correct then user clicks Ok button. Then the data will be saved to the database and display a success message.

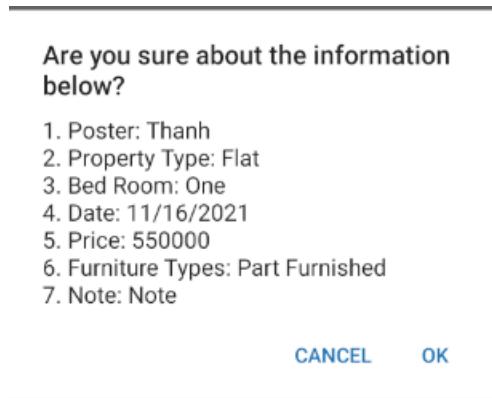


Figure 8: Check Information

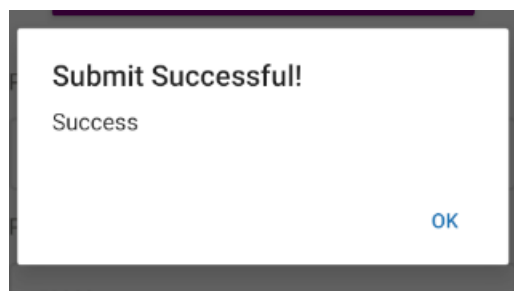


Figure 9: Submit successful

## 2.2 Code and explanation

### 2.2.1. Code

#### Form.js

Contains the code to create and validate the form for the application

```
import { StatusBar } from 'expo-status-bar';
import React, { useState } from 'react';
import DateTimePicker from '@react-native-community/datetimepicker';
import { Formik } from 'formik';
import { SignupSchema } from './validation';
import { styles } from './styles';
import RNPickerSelect from 'react-native-picker-select';
import { KeyboardAwareScrollView } from "react-native-keyboard-aware-scroll-view";
import { StyleSheet, Text, View, SafeAreaView, TextInput, TouchableOpacity, Alert, Button, } from 'react-native';
```

```

const FormValidate = ({ navigation }) => {
  function onSubmitHandler(values) {
    Alert.alert(
      "Are you sure about the information below?",
      `1. Poster: ${values.name}
2. Property Type: ${values.propertyType}
3. Bed Room: ${values.bedRoom}
4. Date: ${GetFormattedDate(date)}
5. Price: ${parseInt(values.price)}
6. Furniture Types: ${values.furnitureType}
7. Note: ${values.note}`,

      [
        {
          text: "Cancel",
          style: "cancel"
        },
        { text: "OK", onPress: () => postData(values) }
      ]
    );
  }
}

function postData(data) {
  const url = "http://192.168.1.6:5001/api/Logbook/CreateLogbook";
  fetch(url, {
    method: 'POST',
    headers: {
      'Accept': '/',
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      name: data.name,
      propertyType: data.propertyType,
      bedRoom: data.bedRoom,
      dateCreated: date,
      price: parseInt(data.price),
      furnitureTypes: data.furnitureType,
      note: data.note
    })
  }).then((response) => response.json())
  .then((responseJson) => {

    if (responseJson) {
      Alert.alert(

```

```

        "Submit Successful!",
        "Success",
        [
            { text: "OK", onPress: () => navigation.push('Form') }
        ]
    );
}
else {
    Alert.alert(
        "Submit Fail!",
        "Fail",
        [
            { text: "OK" }
        ]
    )
}
})
.catch((error) => {
    console.error(error);
}));

}
function isValidForm(isValid, touched) {
    return isValid && Object.keys(touched).length !== 0 && !propertyTypeErr &&
!bedRoomErr;
}

//Date Picker
const [isPickerShow, setIsPickerShow] = useState(false);
const [date, setDate] = useState(new Date(Date.now()));
const maxDate = new Date(Date.now());
function GetFormattedDate(dateTime) {

    var month = dateTime.getMonth() + 1;
    var day = dateTime.getDate();
    var year = dateTime.getFullYear();
    return month + "/" + day + "/" + year;
}
const showPicker = () => {
    setIsPickerShow(true);
};

const onChange = (event, value) => {
    const currentDate = value || date;
    setDate(currentDate);
};

```

```

    if (Platform.OS === 'android') {
      setIsPickerShow(false);
    }
  };

  // select property type
  const [propertyType, setPropertyType] = useState();
  const [propertyTypeErr, setPropertyTypeErr] = useState();
  function setErrorPropertyType() {
    if (propertyType === '') {
      setPropertyTypeErr('Please select this field')
    }
    else {
      setPropertyTypeErr();
    }
  }
  React.useEffect(() => {
    return setErrorPropertyType();
  }, [propertyType]);

  //bedrooms
  const [bedRoom, setBedRoom] = useState();
  const [bedRoomErr, setBedRoomErr] = useState();
  function setErrorBedRoom() {
    if (bedRoom === '') {
      setBedRoomErr('Please select this field')
    }
    else {
      setBedRoomErr();
    }
  }
  React.useEffect(() => {
    return setErrorBedRoom();
  }, [bedRoom]);

  return (
    <>
      <SafeAreaView style={styles.topSafeArea} />
      <StatusBar style="auto" />
      <SafeAreaView style={styles.container}>

```

```

<View style={styles.header}>
  <Text style={styles.headerText}>Create new Post</Text>
</View>

<KeyboardAwareScrollView
  style={styles.content}
  showsVerticalScrollIndicator={false}
  keyboardShouldPersistTaps="handled"
  extraScrollHeight={50}
  enableOnAndroid={true}
>
  <Formik
    initialValues={{ name: '', date: date, propertyType: undefined,
bedRoom: undefined, price: undefined, furnitureType: '', note: '' }}
    validationSchema={SignupSchema}
    onSubmit={onSubmitHandler}
  >
    {({ handleChange, handleBlur, handleSubmit, values, errors, touched,
isValid }) => (
      <>
        <View style={styles.formGroup}>
          <Text style={styles.label}>Property type</Text>
          <Text style={styles.requireProp}>*</Text>
          <RNPickerSelect
            onChange={(value) => { values.propertyType = value;
setPropertyType(value) }}
            items={[
              { label: 'Flat', value: 'Flat', },
              { label: 'House', value: 'House' },
              { label: 'Bungalow', value: 'Bungalow' },
            ]}
            placeholder={{ label: "Select Property Type", value: '' }}
            style={{ inputAndroid: { color: 'black' } }}
          />

          {propertyTypeErr ?
            (<View style={styles.errorContainer}>
              <Text style={styles.errorText}>{propertyTypeErr}</Text>
            </View>) : null}
        </View>

        <View style={styles.formGroup}>
          <Text style={styles.label}>Bedrooms</Text>

```

```

        <Text style={styles.requireBed}>*</Text>
        <RNPickerSelect
          onChange={(value) => { values.bedRoom = value;
setBedRoom(value) }}
          items={[
            { label: 'Studio', value: 'Studio', },
            { label: 'One', value: 'One' },
            { label: 'Two', value: 'Two' },
          ]}
          placeholder={{ label: "Select Bedroom", value: '' }}
          style={{ inputAndroid: { color: 'black' } }}
        />
        {bedRoomErr ?
          (<View style={styles.errorContainer}>
            <Text style={styles.errorText}>{bedRoomErr}</Text>
          </View>) : null}
      </View>

    <View style={styles.formGroup}>
      <Text style={styles.label}>Furniture Types</Text>
      <RNPickerSelect
        onChange={(value) => values.furnitureType = value}
        items={[
          { label: 'Furnished', value: 'Furnished', },
          { label: 'Unfurnished', value: 'Unfurnished' },
          { label: 'Part Furnished', value: 'Part Furnished' },
        ]}
        placeholder={{ label: "Select Furniture Type", value: '' }}
        style={{ inputAndroid: { color: 'black' } }}
      >
    </RNPickerSelect>

  </View>

  <View style={styles.containerPicker}>
    <View style={styles.pickedDateContainer}>
      <Text
style={styles.pickedDate}>{GetFormattedDate(date)}</Text>
    </View>
    {!isPickerShow && (
      <View style={styles.btnContainer}>
        <Button title="Choose Date" color="purple"
onPress={showPicker} />
      </View>
    )
  }

```

```

    })
    {isPickerShow && (
      <DateTimePicker
        value={date}
        mode={'date'}
        display={Platform.OS === 'ios' ? 'spinner' : 'default'}
        is24Hour={true}
        onChange={onChange}
        style={styles.datePicker}
        maximumDate={maxDate}
      />
    )}
  </View>

  <View style={styles.formGroup}>
    <Text style={styles.label}>Poster</Text>
    <Text style={styles.requirePoster}>*</Text>
    <TextInput style={styles.input}
      onChangeText={handleChange('name')}
      onBlur={handleBlur('name')}
      value={values.name}
      placeholder="Enter The Poster's Name" />

    {errors.name && touched.name ?
      (<View style={styles.errorContainer}>
        <Text style={styles.errorText}>{errors.name}</Text>
      </View>) : null}
  </View>

  <View style={styles.formGroup}>
    <Text style={styles.label}>Price</Text>
    <Text style={styles.requirePrice}>*</Text>
    <TextInput style={styles.input}
      keyboardType='numeric'
      onBlur={handleBlur('price')}
      value={values.price}
      onChangeText={handleChange('price')}
      placeholder="Enter Price" />

    {errors.price && touched.price ?
      (<View style={styles.errorContainer}>
        <Text style={styles.errorText}>{errors.price}</Text>
      </View>
    ) : null}

```



```

        </View>

        <View style={styles.formGroup}>
          <Text style={styles.label}>Note</Text>
          <TextInput
            style={styles.inputNote}
            onBlur={handleBlur('note')}
            value={values.note}
            onChangeText={handleChange('note')}
            multiline={true}
            numberOfLines={3}
            placeholder="Enter Your Note" />

          {errors.note && touched.note ?
            (<View style={styles.errorContainer}>
              <Text style={styles.errorText}>{errors.note}</Text>
            </View>
            ) : null}
        </View>

        <TouchableOpacity
          disabled={!isFormValid(isValid, touched)}
          onPress={handleSubmit}
        >
          <View
            style={[
              styles.buttonSubmit,
              {
                opacity: isFormValid(isValid, touched) ? 1 : 0.5,
              },
            ]}
          >
            <Text style={styles.buttonText}>SUBMIT</Text>
          </View>
        </TouchableOpacity>

      </>
    )}
  </Formik>
  <View style={styles.spacer}></View>
</KeyboardAwareScrollView>
</SafeAreaView>
</>
);
};

```

```
export default FormValidate;
```

### Validate.js

Contains code that performs validation for the fields

```
import * as Yup from 'yup';

export const SignupSchema = Yup.object().shape({
  propertyType: Yup.string()
    .required('Please Select This Field!'),
  bedRoom: Yup.string()
    .required('Please Select This Field!'),
  name: Yup.string()
    .max(100, 'Name Too Long, Please Re-enter!')
    .required('Please Enter This Field!'),
  price: Yup.number()
    .required('Please Enter This Field!')
    .typeError('You Must Specify a Number!')
    .min(0, 'Invalid Price!'),
  note: Yup.string()
    .max(200, 'Can not enter more than 200 characters')
});
```

### 2.2.2. Explanation

In addition to the libraries supported by react native, here are the libraries I added to use to create forms and below are their functions:

- @react-native-community/datetimepicker: Serves the application's date and time selection function.
- formik: Combined with the yup library (which will be described in more detail later) helps me to implement validation functionality for the application.
- react-native-picker-select: this library help me to make dropdown menu function for app

```
import { StatusBar } from 'expo-status-bar';
import React, { useState } from 'react';
import DateTimePicker from '@react-native-community/datetimepicker';
import { Formik } from 'formik';
import { SignupSchema } from './validation';
import { styles } from './styles';
import RNPickerSelect from 'react-native-picker-select';
import { KeyboardAwareScrollView } from "react-native-keyboard-aware-scroll-view";
```

```
import { StyleSheet, Text, View, SafeAreaView, TextInput, TouchableOpacity, Alert, Button, } from 'react-native';
```

I use formik for form processing because it provides me with many very convenient functions. Here I don't need to declare state variables but just declare initialValues for formik. After the values are changed, we will get those values through the values variable

```
<Formik
  initialValues={{ name:'', date: date , propertyType: undefined,
  bedRoom : undefined, price: undefined, furnitureType: '', note:''}}
  validationSchema={SignupSchema}
  onSubmit={onSubmitHandler}
  >
  {({ handleChange, handleBlur, handleSubmit, values, errors, touched,
  isValid }) => (
    // Form in here)
  </Formik>
```

2 fields Property Type and Bedrooms will be displayed as a drop-down menu for users to choose. The default value of these fields is empty until the user selects a value, that value will be assigned via the onChange function.

```
<View style={styles.formGroup}>
  <Text style={styles.label}>Property type</Text>
  <Text style={styles.requireProp}>*</Text>
  <RNPickerSelect
    onChange={(value) => {values.propertyType = value;
  setPropertyType(value)}}
    items={[
      { label: 'Flat', value: 'Flat', },
      { label: 'House', value: 'House' },
      { label: 'Bungalow', value: 'Bungalow' },
    ]}
    placeholder={{ label: "Select Property Type", value: ''
  }}
    style={{ inputAndroid: { color: 'black' } }}
  />

  {propertyTypeErr ?
    (<View style={styles.errorContainer}>
      <Text style={styles.errorText}>{propertyTypeErr}</Text>
    </View>) : null}
</View>
```

```

    <View style={styles.formGroup}>
      <Text style={styles.label}>Bedrooms</Text>
      <Text style={styles.requireBed}>*</Text>
      <RNPickerSelect
        onChange={(value) => {values.bedRoom = value;
setBedRoom(value)}}
        items={[
          { label: 'Studio', value: 'Studio', },
          { label: 'One', value: 'One' },
          { label: 'Two', value: 'Two' },
        ]}
        placeholder={{ label: "Select Bedroom", value: '' }}
        style={{ inputAndroid: { color: 'black' } }}
      />
      {bedRoomErr ?
      (<View style={styles.errorContainer}>
        <Text style={styles.errorText}>{bedRoomErr}</Text>
      </View>) : null}
    </View>

```

In addition, I also consider that if the user has selected a field other than the default field and then selects the default field again with a null value, it will assign an error to the two variables `propertyTypeError` and `bedRoomErr` and display it directly below.

```

// select property type
const [propertyType, setPropertyType] = useState();
const [propertyTypeError, setPropertyTypeError] = useState();
function setErrorPropertyType() {
  if (propertyType === '') {
    setPropertyTypeError('Please select this field')
  }
  else {
    setPropertyTypeError();
  }
}
}
React.useEffect(() => {
  return setErrorPropertyType();
}, [propertyType]);

//bedrooms

```

```

const [bedRoom, setBedRoom] = useState();
const [bedRoomErr, setBedRoomErr] = useState();
function setErrorBedRoom() {
  if (bedRoom === '') {
    setBedRoomErr('Please select this field')

  }
  else {
    setBedRoomErr();
  }
}
React.useEffect(() => {
  return setErrorBedRoom();
}, [bedRoom]);

```

The Furniture Type field is also displayed as a drop-down menu, but this field is not required for the user to select, so if the user selects a value, it will receive the value selected by the user through the onChange function, and if the user does not select it, it will receive default value is null.

```

<View style={styles.formGroup}>
  <Text style={styles.label}>Furniture Types</Text>
  <RNPickerSelect
    onChange={(value) => values.furnitureType = value}
    items={[
      { label: 'Furnished', value: 'Furnished', },
      { label: 'Unfurnished', value: 'Unfurnished' },
      { label: 'Part Furnished', value: 'Part Furnished' },
    ]}
    placeholder={{ label: "Select Furniture Type", value: '' }}
    style={{ inputAndroid: { color: 'black' } }}
  >
</RNPickerSelect>

</View>

```

Next is the function that handles the date selection function. GetFormattedDate function to handle displaying date as mm/dd/yyyy to be more user friendly. The showPicker function is used to set the display state of the date picker dialog for the user. Finally, the onChange function is used to get the date selected by the user and close the date selection dialog.

```

//Date Picker
const [isPickerShow, setIsPickerShow] = useState(false);

```

```

const [date, setDate] = useState(new Date(Date.now()));
const maxDate = new Date(Date.now());
function GetFormattedDate(dateTime) {

    var month = dateTime.getMonth() + 1;
    var day = dateTime.getDate();
    var year = dateTime.getFullYear();
    return month + "/" + day + "/" + year;
}

const showPicker = () => {
    setIsPickerShow(true);
};

const onChange = (event, value) => {
    const currentDate = value || date;
    setDate(currentDate);
    if (Platform.OS === 'android') {
        setIsPickerShow(false);
    }
};

```

The remaining fields including Poster, Price and Note are all input fields, but each field has its specific differences as follows: The Poster field is just an input tag that receives user input and the onChangeText property will receive and save the data to formik's poster variable. The Price field only allows the user to enter a number with keyboardType="numeric" and also uses the onChangeText property to get the value and store it in the price variable. Finally, the Note field, this field is similar to the Poster field, which will receive input data and use the onChangeText property and save the value in the note variable, but here is another thing that is if the user enters too many characters then will automatically bring a line break with the multiline={true} attribute. Under each field there is also a row to display the error, if the user enters it incorrectly, the error will be displayed immediately.

```

<View style={styles.formGroup}>
    <Text style={styles.label}>Poster</Text>
    <Text style={styles.requirePoster}>*</Text>
    <TextInput style={styles.input}
        onChangeText={handleChange('name')}
        onBlur={handleBlur('name')}
        value={values.name}
        placeholder="Enter The Poster's Name" />

    {errors.name && touched.name ?
        (<View style={styles.errorContainer}>
            <Text style={styles.errorText}>{errors.name}</Text>

```

```

        </View>) : null}
    </View>

    <View style={styles.formGroup}>
      <Text style={styles.label}>Price</Text>
      <Text style={styles.requirePrice}>*</Text>
      <TextInput style={styles.input}
        keyboardType='numeric'
        onBlur={handleBlur('price')}
        value={values.price}
        onChangeText={handleChange('price')}
        placeholder="Enter Price" />

      {errors.price && touched.price ?
        (<View style={styles.errorContainer}>
          <Text style={styles.errorText}>{errors.price}</Text>
        </View>
        ) : null}
    </View>

    <View style={styles.formGroup}>
      <Text style={styles.label}>Note</Text>
      <TextInput
        style={styles.inputNote}
        onBlur={handleBlur('note')}
        value={values.note}
        onChangeText={handleChange('note')}
        multiline={true}
        numberOfLines={3}
        placeholder="Enter Your Note" />

      {errors.note && touched.note ?
        (<View style={styles.errorContainer}>
          <Text style={styles.errorText}>{errors.note}</Text>
        </View>
        ) : null}
    </View>

```

The `isFormValid` function is used to check if all information entered by the user is correct, if all is correct, it returns true, otherwise, it returns false.

```
function isFormValid(isValid, touched) {
```

```

    return isValid && Object.keys(touched).length !== 0 && !propertyTypeErr &&
    !bedRoomErr;
  }

```

Rely on the `isFormValid` function to check that all fields have not been entered correctly, otherwise the Submit button will be disabled and its opacity will be reduced to 0.5.

```

<TouchableOpacity
  disabled={!isFormValid(isValid, touched)}
  onPress={handleSubmit}
>
  <View
    style={[
      styles.buttonSubmit,
      {
        opacity: isFormValid(isValid, touched) ? 1 : 0.5,
      },
    ]}
  >
    <Text style={styles.buttonText}>SUBMIT</Text>
  </View>
</TouchableOpacity>

```

Next is the `onSubmitHandler` function used to display a dialog box containing entered information for the user to check. If the user clicks OK, the `postData` function will be called

```

function onSubmitHandler(values) {
  Alert.alert(
    "Are you sure about the information below?",
    `
1. Poster: ${values.name}
2. Property Type: ${values.propertyType}
3. Bed Room: ${values.bedRoom}
4. Date: ${GetFormattedDate(date)}
5. Price: ${parseInt(values.price)}
6. Furniture Types: ${values.furnitureType}
7. Note: ${values.note}`,
    [
      {
        text: "Cancel",
        style: "cancel"
      },
      { text: "OK", onPress: () => postData(values) }
    ]
  );
}

```



```

    ]
  );
}

```

The postData function is used to push the user's information to the server and save it to the database and display a success message to the user.

```

function postData(data) {
  const url = "http://192.168.1.6:5001/api/Logbook/CreateLogbook";
  fetch(url, {
    method: 'POST',
    headers: {
      'Accept': '/',
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      name: data.name,
      propertyType: data.propertyType,
      bedRoom: data.bedRoom,
      dateCreated: date,
      price: parseInt(data.price),
      furnitureTypes: data.furnitureType,
      note: data.note
    })
  }).then((response) => response.json())
    .then((responseJson) => {

      if (responseJson) {
        Alert.alert(
          "Submit Successful!",
          "Success",
          [
            { text: "OK", onPress: () => navigation.push('Form') }
          ]
        );
      }
      else {
        Alert.alert(
          "Submit Fail!",
          "Fail",
          [
            { text: "OK" }
          ]
        );
      }
    });
}

```

```

    )
  }
})
.catch((error) => {
  console.error(error);
});
}

```

### III. Create Android data entry screen

#### 1. Basic information

1.1 Student name	Ngo Hoanh Thanh
1.2 Who did you work with? Note that for logbook exercises you are allowed to work with one other person as long as you give their name and login id and both contribute to the work.	I just do it alone
1.3 Which Exercise is this? Tick as appropriate.	Create an android data entry screen that displays a form that allows a user to enter all the fields specified in the coursework section
1.4 How well did you complete the exercise? Tick as appropriate.	I tried but couldn't complete it <input type="checkbox"/> I did it but I feel I should have done better <input type="checkbox"/> I did everything that was asked <input checked="" type="checkbox"/> I did more than was asked for <input type="checkbox"/>
1.5 Briefly explain your answer to question	I created an android data entry screen that displays a form that allows a user to enter all the fields specified in the coursework section 1 a) and the app perform some validation of the data input and display an error message to the user if the data is invalid.

#### 2. Exercise answer:

##### 2.1. Screenshots demonstrating what is achieved:

In this part I will use android studio with java language to create RentalZ application. This application also provides 7 fields: Property Type, Bedrooms, Furniture Type, Date, Poster, Price, Note. In which there are 2 fields, Furniture Type and Note, which are not required for the user to enter data, the rest of the fields are required.

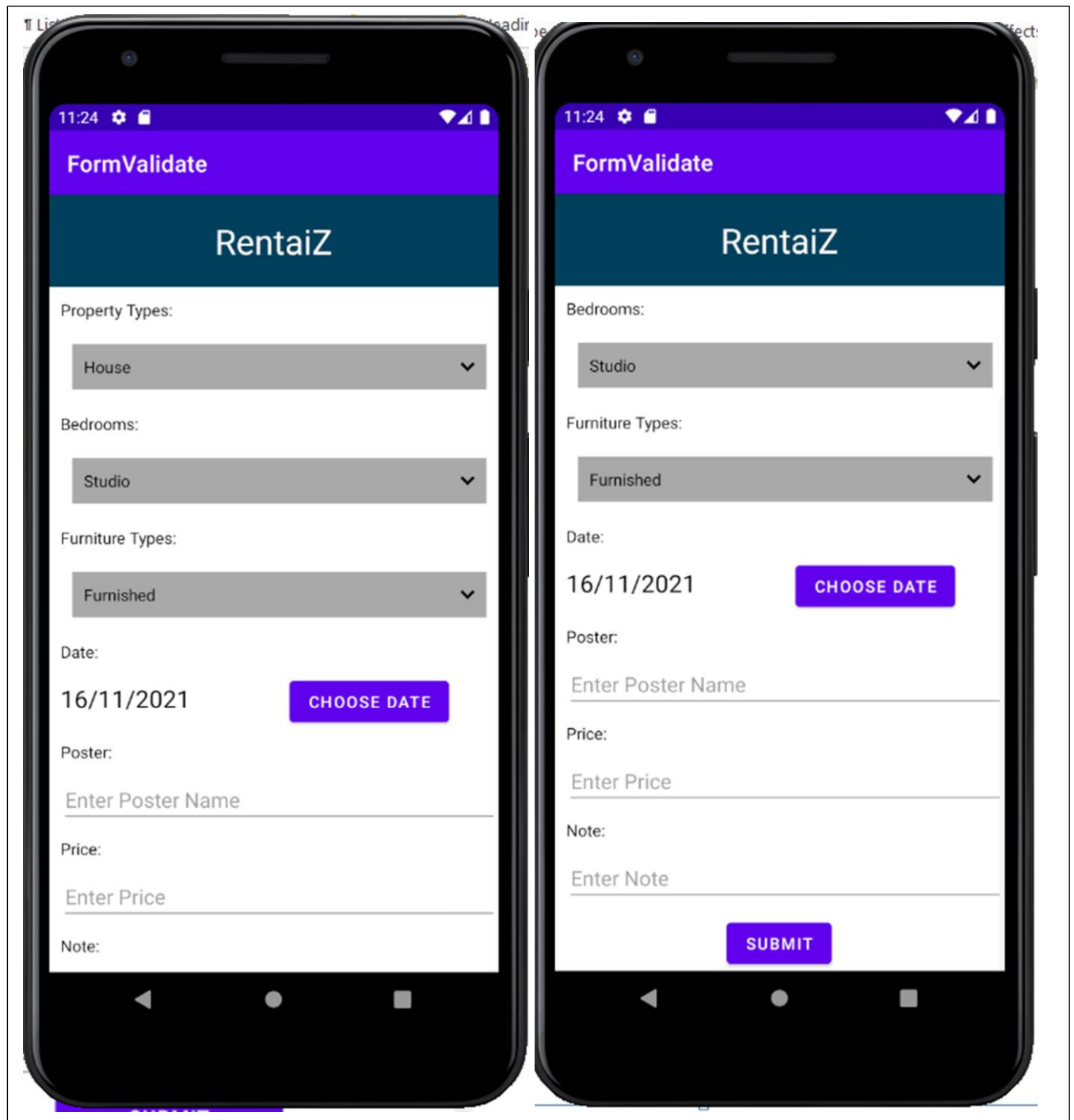


Figure 10: Form input Screen

The first 3 fields including Property Type, Bedrooms, Furniture Types are drop-down lists and they all have a default value of a specific value, so these fields are not required for the user to select.

Property Types:

House ▼

Bedrooms:

Studio ▼

Furniture Types:

Furnished ▼

Figure 11: Select drop down

As for the Date field, there will be 1 field to display the date and its default value is today's date.

Date:

16/11/2021

CHOOSE DATE

Figure 12: Date

If the user clicks on the choose date button, a dialog box will be displayed for the user to choose the date they want but can only choose from today and earlier.

2021  
Tue, Nov 16

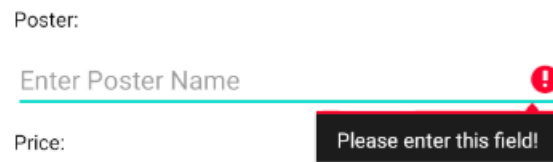
< November 2021

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

CANCEL OK

Figure 13: Choose Date dialog

The Poster field requires the user to enter data or it will display an error message.



Poster:

Enter Poster Name

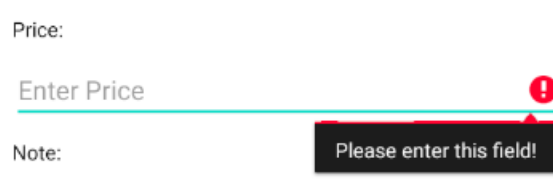
Price:

Please enter this field!

The screenshot shows a form with two fields. The first field, labeled 'Poster:', has a text input with the placeholder 'Enter Poster Name'. A red exclamation mark icon is positioned to the right of the input. Below this, the label 'Price:' is visible, followed by a black error message box that says 'Please enter this field!'.

Figure 14: Poster error

The Price field only allows the user to enter a positive integer and requires the user to enter the data otherwise an error message will be displayed.



Price:

Enter Price

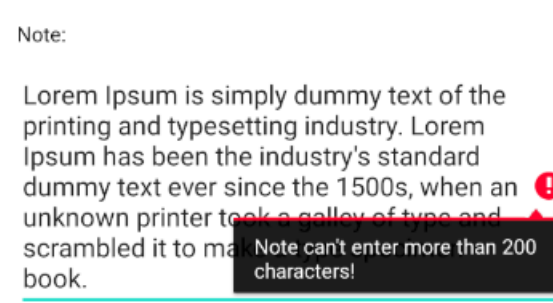
Note:

Please enter this field!

The screenshot shows a form with two fields. The first field, labeled 'Price:', has a text input with the placeholder 'Enter Price'. A red exclamation mark icon is positioned to the right of the input. Below this, the label 'Note:' is visible, followed by a black error message box that says 'Please enter this field!'.

Figure 15: Price error

The Note field does not force the user to enter, but if the user intentionally enters too long over 200 characters, an error message will also be displayed



Note:

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a book.

Note can't enter more than 200 characters!

The screenshot shows a form with a single field labeled 'Note:'. The text input contains a long paragraph of Lorem Ipsum text. A red exclamation mark icon is positioned to the right of the input. Below the input, a black error message box says 'Note can't enter more than 200 characters!'.

Figure 16: Note error

If the user has passed the authentication process, after clicking the submit button, a message will appear that the authentication is successful.

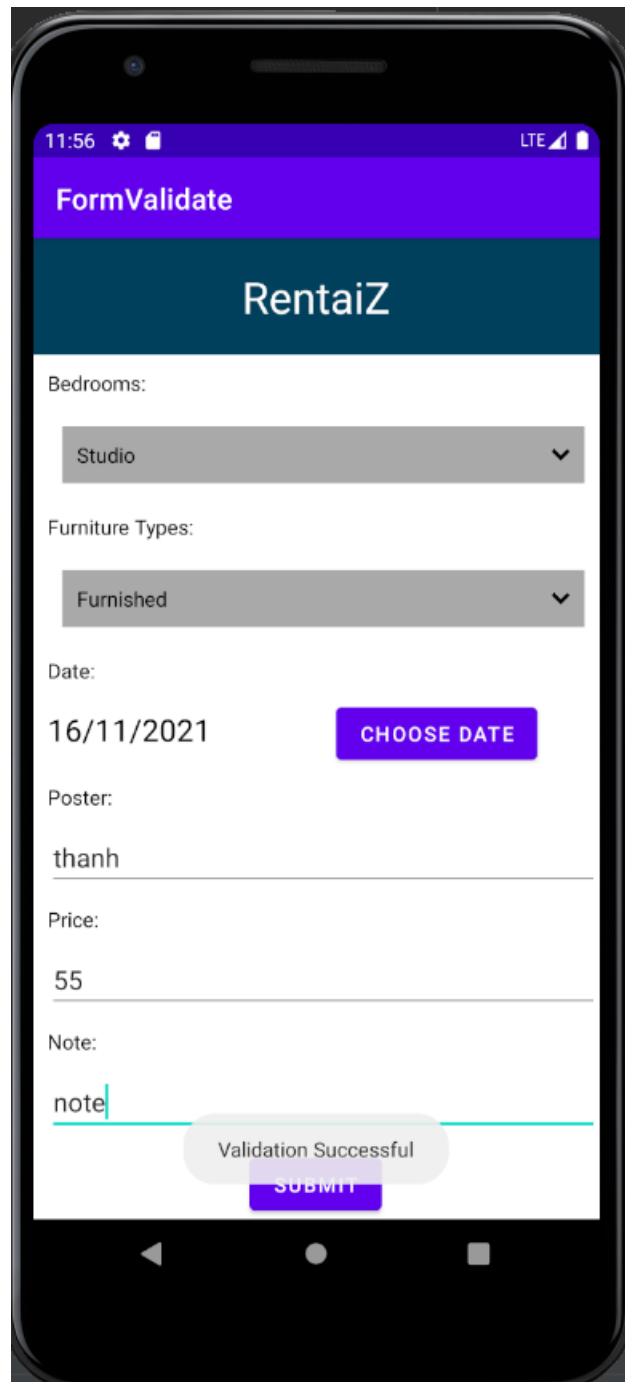


Figure 17: Validation Successful

## 2.2. Code and Explanation

### 2.2.1. Code

Here is the code that shows the user interface of the application.

**Activity\_main.xml**

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
>

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="RentaiZ"
        android:textColor="@color/white"
        android:textSize="30sp"
        android:textAlignment="center"
        android:background="#003f5c"
        android:paddingTop="20dp"
        android:paddingBottom="20dp"
        android:layout_marginBottom="10dp"
    />

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical">

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginLeft="10dp"
                android:orientation="vertical"
                android:paddingBottom="10dp">

                <TextView
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:paddingBottom="10dp"
                    android:text="Property Types:"
                    android:textColor="@color/black"
                    android:textFontWeight="600" />

                <Spinner
                    android:id="@+id/spnProperty"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:layout_margin="10dp"
                    android:background="@android:color/transparent" />

            </LinearLayout>

        <LinearLayout
            android:layout_width="match_parent"

```

```
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dp"
        android:orientation="vertical"
        android:paddingBottom="10dp">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:paddingBottom="10dp"
            android:text="Bedrooms:"
            android:textColor="@color/black"
            android:textFontWeight="600" />

        <Spinner
            android:id="@+id/spnBedroom"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:background="@android:color/transparent" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dp"
        android:orientation="vertical"
        android:paddingBottom="10dp">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:paddingBottom="10dp"
            android:text="Furniture Types:"
            android:textColor="@color/black"
            android:textFontWeight="600" />

        <Spinner
            android:id="@+id/spnFurniture"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:background="@android:color/transparent" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dp"
        android:orientation="vertical"
        android:paddingBottom="10dp">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:paddingBottom="10dp"
```



```

        android:text="Date:"

        android:textColor="@color/black"
        android:textFontWeight="600" />

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"

    >

    <TextView
        android:id="@+id/tvDate"
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:textColor="@color/black"
        android:textSize="21dp" />

    <Button
        android:id="@+id/btnDate"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Choose Date" />
</LinearLayout>

</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="10dp"
    android:orientation="vertical"
    android:paddingBottom="10dp">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingBottom="10dp"
        android:text="Poster:"
        android:textColor="@color/black"
        android:textFontWeight="600" />

    <EditText
        android:id="@+id/edtPoster"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter Poster Name"
        android:text="" />
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="10dp"
    android:orientation="vertical"
    android:paddingBottom="10dp">

```

```

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:paddingBottom="10dp"
            android:text="Price:"
            android:textColor="@color/black"
            android:textFontWeight="600" />

        <EditText
            android:id="@+id/edtPrice"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Enter Price"
            android:inputType="number"
            android:text=""

        />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dp"
        android:orientation="vertical"
        android:paddingBottom="10dp">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:paddingBottom="10dp"
            android:text="Note:"
            android:textColor="@color/black"
            android:textFontWeight="600" />

        <EditText
            android:id="@+id/edtNote"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Enter Note"
            android:inputType="text|textMultiLine"
            android:text=""

        />
    </LinearLayout>

    <Button
        android:id="@+id/btnSubmit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="150dp"
        android:text="Submit" />
</LinearLayout>

</ScrollView>

</LinearLayout>

```

### property\_selected.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#a9a9a9">

    <TextView
        android:id="@+id/tvProperty_selected"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:text="@string/app_name"
        android:textColor="@color/black"
        android:layout_toStartOf="@id/imgProperty"
    />

    <ImageView
        android:id="@+id/imgProperty"
        android:layout_width="23dp"
        android:layout_height="23dp"
        android:layout_alignParentEnd="true"
        android:layout_centerVertical="true"
        android:src="@drawable/down_arrow"
        android:paddingRight="10dp"
    />
</RelativeLayout>
```

### bedroom\_selected.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#a9a9a9">

    <TextView
        android:id="@+id/tvBedroom_selected"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:text="@string/app_name"
        android:textColor="@color/black"
        android:layout_toStartOf="@id/imgBedroom"
    />

    <ImageView
        android:id="@+id/imgBedroom"
        android:layout_width="23dp"
        android:layout_height="23dp"
        android:layout_alignParentEnd="true"
        android:layout_centerVertical="true"
    />
</RelativeLayout>
```

```
        android:src="@drawable/down_arrow"
        android:paddingRight="10dp"
    />
</RelativeLayout>
```

### furniture\_selected.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#a9a9a9">

    <TextView
        android:id="@+id/tvFurniture_selected"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:text="@string/app_name"
        android:textColor="@color/black"
        android:layout_toStartOf="@id/imgFurniture"
    />

    <ImageView
        android:id="@+id/imgFurniture"
        android:layout_width="23dp"
        android:layout_height="23dp"
        android:layout_alignParentEnd="true"
        android:layout_centerVertical="true"
        android:src="@drawable/down_arrow"
        android:paddingRight="10dp"
    />
</RelativeLayout>
```

### item\_property.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/tvProperty"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="16sp"
        android:text="@string/app_name"
        android:textColor="@color/black"
        android:padding="10dp"
    />

    <View
        android:layout_width="match_parent"
```

```
        android:layout_height="0.5dp"
        android:background="#a9a9a9"
    />
</LinearLayout>
```

### item\_bedroom.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/tvBedroom"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="16sp"
        android:text="@string/app_name"
        android:textColor="@color/black"
        android:padding="10dp"
    />

    <View
        android:layout_width="match_parent"
        android:layout_height="0.5dp"
        android:background="#a9a9a9"
    />
</LinearLayout>
```

### item\_furniture.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/tvFurniture"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="16sp"
        android:text="@string/app_name"
        android:textColor="@color/black"
        android:padding="10dp"
    />

    <View
        android:layout_width="match_parent"
        android:layout_height="0.5dp"
        android:background="#a9a9a9"
    />
</LinearLayout>
```

Next is the code for creating the drop-down list fields and performing the validation.

### Property

```
package com.example.formvalidate;

public class Property {
    private String name;

    public Property(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

### Property Adapter

```
package com.example.formvalidate;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import java.util.List;

public class PropertyAdapter extends ArrayAdapter<Property> {

    public PropertyAdapter(@NonNull Context context, int resource, @NonNull
List<Property> objects) {
        super(context, resource, objects);
    }

    @NonNull
    @Override
    public View getView(int position, @Nullable View convertView, @NonNull
ViewGroup parent) {
        convertView =
```

```

LayoutInflater.from(parent.getContext()).inflate(R.layout.property_selected, parent, false);
        TextView tvPropertySelected =
convertView.findViewById(R.id.tvProperty_selected);

        Property property = this.getItem(position);
        if (property != null ) {
            tvPropertySelected.setText(property.getName());
        }
        return convertView;
    }

    @Override
    public View getDropDownView(int position, @Nullable View convertView,
@NonNull ViewGroup parent) {
        convertView =
LayoutInflater.from(parent.getContext()).inflate(R.layout.item_property, parent, false);
        TextView tvProperty = convertView.findViewById(R.id.tvProperty);

        Property property = this.getItem(position);
        if (property != null ) {
            tvProperty.setText(property.getName());
        }

        return convertView;
    }
}

```

## Bedroom

```

package com.example.formvalidate;

public class Bedroom {
    private String name;

    public Bedroom(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}

```

## BedroomAdapter

```

package com.example.formvalidate;

import android.content.Context;

```

```
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import java.util.List;

public class BedroomAdapter extends ArrayAdapter<Bedroom> {

    public BedroomAdapter(@NonNull Context context, int resource, @NonNull
List<Bedroom> objects) {
        super(context, resource, objects);
    }

    @NonNull
    @Override
    public View getView(int position, @Nullable View convertView, @NonNull
ViewGroup parent) {
        convertView =
LayoutInflater.from(parent.getContext()).inflate(R.layout.bedroom_selected,parent,false);
        TextView tvBedroomSelected =
convertView.findViewById(R.id.tvBedroom_selected);

        Bedroom bedroom = this.getItem(position);
        if (bedroom != null ){
            tvBedroomSelected.setText(bedroom.getName());
        }
        return convertView;
    }

    @Override
    public View getDropDownView(int position, @Nullable View convertView,
@NonNull ViewGroup parent) {
        convertView =
LayoutInflater.from(parent.getContext()).inflate(R.layout.item_bedroom,parent,false);
        TextView tvBedroom = convertView.findViewById(R.id.tvBedroom);

        Bedroom bedroom = this.getItem(position);
        if (bedroom != null ){
            tvBedroom.setText(bedroom.getName());
        }

        return convertView;
    }
}
```

## Furniture



```

package com.example.formvalidate;

public class Furniture {
    private String name;

    public Furniture(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}

```

## FurnitureAdapter

```

package com.example.formvalidate;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import java.util.List;

public class FurnitureAdapter extends ArrayAdapter<Furniture> {
    public FurnitureAdapter(@NonNull Context context, int resource, @NonNull
    List<Furniture> objects) {
        super(context, resource, objects);
    }

    @NonNull
    @Override
    public View getView(int position, @Nullable View convertView, @NonNull
    ViewGroup parent) {
        convertView =
        LayoutInflater.from(parent.getContext()).inflate(R.layout.furniture_selected,
        parent, false);
        TextView tvFurnitureSelected =
        convertView.findViewById(R.id.tvFurniture_selected);

        Furniture furniture = this.getItem(position);
        if (furniture != null ) {
            tvFurnitureSelected.setText(furniture.getName());
        }
    }
}

```

```

        return convertView;
    }

    @Override
    public View getDropDownView(int position, @Nullable View convertView,
@NonNull ViewGroup parent) {
        convertView =
LayoutInflater.from(parent.getContext()).inflate(R.layout.item_furniture,pare
nt,false);
        TextView tvFurniture = convertView.findViewById(R.id.tvFurniture);

        Furniture furniture = this.getItem(position);
        if (furniture != null ){
            tvFurniture.setText(furniture.getName());
        }

        return convertView;
    }
}

```

## MainActivity

```

package com.example.formvalidate;

import androidx.appcompat.app.AppCompatActivity;

import android.app.DatePickerDialog;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.CalendarView;
import android.widget.DatePicker;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;

public class MainActivity extends AppCompatActivity {

    TextView tvDate;
    Button btnDate,btnSubmit;
    EditText edtPoster, edtPrice,edtNote;
    Spinner spnProperty, spnBedroom, spnFurniture;
    private PropertyAdapter propertyAdapter;
    private BedroomAdapter bedroomAdapter;
    private FurnitureAdapter furnitureAdapter;
    @Override
    protected void onCreate(Bundle savedInstanceState) {

```

```

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //Date
        tvDate = (TextView) findViewById(R.id.tvDate);
        Calendar calendar = Calendar.getInstance();
        SimpleDateFormat simpleDateFormat = new
SimpleDateFormat("dd/MM/yyyy");
        tvDate.setText(simpleDateFormat.format(calendar.getTime()));

        btnDate = (Button) findViewById(R.id.btnDate);
        btnDate.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                ChooseDate();
            }
        });

        // spinner property
        spnProperty = findViewById(R.id.spnProperty);
        propertyAdapter = new
PropertyAdapter(this,R.layout.property_selected, getListProperty());
        spnProperty.setAdapter(propertyAdapter);
        spnProperty.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener(){
            @Override
            public void onItemSelected(AdapterView<?> adapterView, View view,
int i, long l){
                Toast.makeText(MainActivity.this,
propertyAdapter.getItem(i).getName(), Toast.LENGTH_SHORT).show();
            }

            @Override
            public void onNothingSelected(AdapterView<?> adapterView){

            }
        });

        // spinner bedroom
        spnBedroom = findViewById(R.id.spnBedroom);
        bedroomAdapter = new BedroomAdapter(this,R.layout.bedroom_selected,
getListBedroom());
        spnBedroom.setAdapter(bedroomAdapter);
        spnBedroom.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener(){
            @Override
            public void onItemSelected(AdapterView<?> adapterView, View view,
int i, long l){
                Toast.makeText(MainActivity.this,
bedroomAdapter.getItem(i).getName(), Toast.LENGTH_SHORT).show();
            }

            @Override
            public void onNothingSelected(AdapterView<?> adapterView){

            }
        });

```

```

        // spinner furniture
        spnFurniture = findViewById(R.id.spnFurniture);
        furnitureAdapter = new
FurnitureAdapter(this,R.layout.furniture_selected, getListFurniture());
        spnFurniture.setAdapter(furnitureAdapter);
        spnFurniture.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener(){
            @Override
            public void onItemSelected(AdapterView<?> adapterView, View view,
int i, long l){
                Toast.makeText(MainActivity.this,
furnitureAdapter.getItem(i).getName(), Toast.LENGTH_SHORT).show();
            }

            @Override
            public void onNothingSelected(AdapterView<?> adapterView){

            }
        });

        //validate
        btnSubmit = (Button) findViewById(R.id.btnSubmit);
        edtPoster = (EditText) findViewById(R.id.edtPoster);
        edtPrice = (EditText) findViewById(R.id.edtPrice);
        edtNote = (EditText) findViewById(R.id.edtNote);

        btnSubmit.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                final String poster = edtPoster.getText().toString();
                final String priceStr = edtPrice.getText().toString();
                final String note = edtNote.getText().toString();
                if (poster.length() == 0){
                    edtPoster.requestFocus();
                    edtPoster.setError("Please enter this field!");
                }
                else if(priceStr.length() == 0){
                    edtPrice.requestFocus();
                    edtPrice.setError("Please enter this field!");
                }

                else if(note.length() > 200){
                    edtNote.requestFocus();
                    edtNote.setError("Note can't enter more than 200
characters!");
                }
                else
                {
                    Toast.makeText(MainActivity.this,"Validation
Successful",Toast.LENGTH_LONG).show();
                }
            }
        });
    }

    private void ChooseDate() {
        Calendar calendar = Calendar.getInstance();
        int date = calendar.get(Calendar.DATE);

```

```

        int month = calendar.get(Calendar.MONTH);
        int year = calendar.get(Calendar.YEAR);

        DatePickerDialog datePickerDialog = new DatePickerDialog(this, new
        DatePickerDialog.OnDateSetListener() {
            @Override
            public void onDateSet(DatePicker datePicker, int i, int i1, int
i2) {
                calendar.set(i,i1,i2);
                SimpleDateFormat simpleDateFormat = new
SimpleDateFormat("dd/MM/yyyy");
                tvDate.setText(simpleDateFormat.format(calendar.getTime()));
            }
        }, year, month, date);

datePickerDialog.getDatePicker().setMaxDate(System.currentTimeMillis());
datePickerDialog.show();
    }

    private List<Property> getListProperty() {
        List<Property> properties = new ArrayList<>();
        properties.add(new Property("House"));
        properties.add(new Property("Flat"));
        properties.add(new Property("Bungalow"));
        return properties;
    }
    private List<Bedroom> getListBedroom() {
        List<Bedroom> bedrooms = new ArrayList<>();
        bedrooms.add(new Bedroom("Studio"));
        bedrooms.add(new Bedroom("One"));
        bedrooms.add(new Bedroom("Two"));
        return bedrooms;
    }
    private List<Furniture> getListFurniture() {
        List<Furniture> furniture = new ArrayList<>();
        furniture.add(new Furniture("Furnished"));
        furniture.add(new Furniture("Unfurnished"));
        furniture.add(new Furniture("Part Furnished"));
        return furniture;
    }
}

```

### 2.2.2. Explanation

First, I create variables tvDate, btnDate, btnSubmit, edtPoster, edtPrice, edtNote, spnBedroom, spnBedroom, spnFurniture, propertyAdapter, bedroomAdapter, furnitureAdapter in the MainActivity class according to the ids declared from the .xml files. They will find views by id and to handle different tasks

```

TextView tvDate;
Button btnDate, btnSubmit;
EditText edtPoster, edtPrice, edtNote;
Spinner spnProperty, spnBedroom, spnFurniture;

```

```
private PropertyAdapter propertyAdapter;  
private BedroomAdapter bedroomAdapter;  
private FurnitureAdapter furnitureAdapter;
```

Next is the process of creating the drop-down list. Take the property type field as an example and do the same for the bedrooms and furniture type fields.

First I will create 2 interface, 1 interface is used to display the interface displayed in the form to the user and 1 interface displays the list of values after clicking. Each interface has corresponding ids to connect to MainActivity class

This is the view displayed in the interface for the user to choose

```
<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:background="#a9a9a9">  
  
    <TextView  
        android:id="@+id/tvProperty_selected"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:padding="10dp"  
        android:text="@string/app_name"  
        android:textColor="@color/black"  
        android:layout_toStartOf="@id/imgProperty"  
    />  
  
    <ImageView  
        android:id="@+id/imgProperty"  
        android:layout_width="23dp"  
        android:layout_height="23dp"  
        android:layout_alignParentEnd="true"  
        android:layout_centerVertical="true"  
        android:src="@drawable/down_arrow"  
        android:paddingRight="10dp"  
    />  
</RelativeLayout>
```

This is the interface is display list for the users.

```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:orientation="vertical">  
  
    <TextView  
        android:id="@+id/tvProperty"  
        android:layout_width="match_parent"
```

```

        android:layout_height="wrap_content"
        android:textSize="16sp"
        android:text="@string/app_name"
        android:textColor="@color/black"
        android:padding="10dp"
    />
    <View
        android:layout_width="match_parent"
        android:layout_height="0.5dp"
        android:background="#a9a9a9"
    />
</LinearLayout>

```

Class Property contains name value, 1 constructor, 2 methods getName and setName as below.

```

package com.example.formvalidate;

public class Property {
    private String name;

    public Property(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}

```

When the above 2 interface are created, they will not have any data. The PropertyAdapter class will rely on the two pre-created views and add display information to them.

```

package com.example.formvalidate;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import java.util.List;

public class PropertyAdapter extends ArrayAdapter<Property> {

```

```

    public PropertyAdapter(@NonNull Context context, int resource, @NonNull
List<Property> objects) {
        super(context, resource, objects);
    }

    @NonNull
    @Override
    public View getView(int position, @Nullable View convertView, @NonNull
ViewGroup parent) {
        convertView =
LayoutInflater.from(parent.getContext()).inflate(R.layout.property_selected,p
arent,false);
        TextView tvPropertySelected =
convertView.findViewById(R.id.tvProperty_selected);

        Property property = this.getItem(position);
        if (property != null ) {
            tvPropertySelected.setText(property.getName());
        }
        return convertView;
    }

    @Override
    public View getDropDownView(int position, @Nullable View convertView,
@NonNull ViewGroup parent) {
        convertView =
LayoutInflater.from(parent.getContext()).inflate(R.layout.item_property,paren
t,false);
        TextView tvProperty = convertView.findViewById(R.id.tvProperty);

        Property property = this.getItem(position);
        if (property != null ) {
            tvProperty.setText(property.getName());
        }

        return convertView;
    }
}

```

Tiếp theo trong lớp MainActivity, chúng ta sẽ tạo một danh sách thuộc tính

```

private List<Property> getListProperty() {
    List<Property> properties = new ArrayList<>();
    properties.add(new Property("House"));
    properties.add(new Property("Flat"));
    properties.add(new Property("Bungalow"));
    return properties;
}

```

Finally in the onCreate function we will display that list thanks to the propertyAdapter class and catch the event when the user selects any value, it will display the selected value.



```

spnProperty = findViewById(R.id.spnProperty);
propertyAdapter = new PropertyAdapter(this,R.layout.property_selected,
getListProperty());
spnProperty.setAdapter(propertyAdapter);
spnProperty.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int i,
long l) {
        Toast.makeText(MainActivity.this,
propertyAdapter.getItem(i).getName(), Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onNothingSelected(AdapterView<?> adapterView) {

    }
});

```

Next is the Date field, I use a TextView to display the selected date and a button for the user to click and bring up a dialog to select the date. The TextView and Button tags will have a matching id so that the MainActivity class can find the id to connect to the Activity\_main.xml

```

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="10dp"
    android:orientation="vertical"
    android:paddingBottom="10dp">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingBottom="10dp"
        android:text="Date:"

        android:textColor="@color/black"
        android:textFontWeight="600" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"

        >

        <TextView
            android:id="@+id/tvDate"
            android:layout_width="200dp"
            android:layout_height="wrap_content"
            android:textColor="@color/black"
            android:textSize="21dp" />

        <Button
            android:id="@+id/btnDate"

```

```

        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Choose Date" />
    </LinearLayout>
</LinearLayout>

```

In the MainActivity class, create a ChooseDate function that will execute when the user selects the Choose Date button, this function will display a date selection dialog box with the default selected date being today.

```

private void ChooseDate() {
    Calendar calendar = Calendar.getInstance();
    int date = calendar.get(Calendar.DATE);
    int month = calendar.get(Calendar.MONTH);
    int year = calendar.get(Calendar.YEAR);

    DatePickerDialog datePickerDialog = new DatePickerDialog(this, new
    DatePickerDialog.OnDateSetListener() {
        @Override
        public void onDateSet(DatePicker datePicker, int i, int i1, int i2) {
            calendar.set(i, i1, i2);
            SimpleDateFormat simpleDateFormat = new
            SimpleDateFormat("dd/MM/yyyy");
            tvDate.setText(simpleDateFormat.format(calendar.getTime()));
        }
    }, year, month, date);
    datePickerDialog.getDatePicker().setMaxDate(System.currentTimeMillis());
    datePickerDialog.show();
}

```

Finally in the function onCreate perform the function to assign the selected date value to the TextView or assign the function to display the date picker dialog to the Choose Date button

```

tvDate = (TextView) findViewById(R.id.tvDate);
Calendar calendar = Calendar.getInstance();
SimpleDateFormat simpleDateFormat = new SimpleDateFormat("dd/MM/yyyy");
tvDate.setText(simpleDateFormat.format(calendar.getTime()));

btnDate = (Button) findViewById(R.id.btnDate);
btnDate.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        ChooseDate();
    }
});

```

For the rest of the fields I use EditText to show an input interface to the user that can enter data from their keyboard. For the Price field I have added the android:inputType="number" attribute to specify that the user is only allowed to enter numbers. For the Poster and Note fields, the user can enter both

letters and numbers, but for the Note field, I have added the `android:inputType="text|textMultiLine"` attribute so that if the user enters too long characters, it will automatically newline. All `EditText` tags will have a matching id so that the `MainActivity` class can find the id to connect to `Activity_main.xml`

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="10dp"
    android:orientation="vertical"
    android:paddingBottom="10dp">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingBottom="10dp"
        android:text="Poster:"
        android:textColor="@color/black"
        android:textFontWeight="600" />

    <EditText
        android:id="@+id/edtPoster"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter Poster Name"
        android:text="" />
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="10dp"
    android:orientation="vertical"
    android:paddingBottom="10dp">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingBottom="10dp"
        android:text="Price:"
        android:textColor="@color/black"
        android:textFontWeight="600" />

    <EditText
        android:id="@+id/edtPrice"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Enter Price"
        android:inputType="number"
        android:text=""

    />
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

```

        android:layout_marginLeft="10dp"
        android:orientation="vertical"
        android:paddingBottom="10dp">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:paddingBottom="10dp"
            android:text="Note:"
            android:textColor="@color/black"
            android:textFontWeight="600" />

        <EditText
            android:id="@+id/edtNote"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Enter Note"
            android:inputType="text|textMultiLine"
            android:text=""

            />
    </LinearLayout>

    <Button
        android:id="@+id/btnSubmit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="150dp"
        android:text="Submit" />
</LinearLayout>

```

In the onCreate function, find the views by id from the activity\_main.xml file and save the corresponding variables.

```

btnSubmit = (Button) findViewById(R.id.btnSubmit);
edtPoster = (EditText) findViewById(R.id.edtPoster);
edtPrice = (EditText) findViewById(R.id.edtPrice);
edtNote = (EditText) findViewById(R.id.edtNote);

```

Below is the code that handles the validation that requires the user to enter the correct input in order to pass the validation process. Here, the Poster and Price fields are required to be entered, and the Note field cannot be more than 200 characters. Finally, if the user has passed the confirmation process, when the submit button is pressed, the message Validation Successful will appear

```

btnSubmit = (Button) findViewById(R.id.btnSubmit);
edtPoster = (EditText) findViewById(R.id.edtPoster);
edtPrice = (EditText) findViewById(R.id.edtPrice);
edtNote = (EditText) findViewById(R.id.edtNote);

btnSubmit.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

```

```

final String poster = edtPoster.getText().toString();
final String priceStr = edtPrice.getText().toString();
final String note = edtNote.getText().toString();
if (poster.length() == 0){
    edtPoster.requestFocus();
    edtPoster.setError("Please enter this field!");
}
else if(priceStr.length() == 0){
    edtPrice.requestFocus();
    edtPrice.setError("Please enter this field!");
}

else if(note.length() > 200){
    edtNote.requestFocus();
    edtNote.setError("Note can't enter more than 200 characters!");
}
else
{
    Toast.makeText(MainActivity.this, "Validation
Successful", Toast.LENGTH_LONG).show();
}
}
});

```

## IV. Conclusion

After completing this LogBook, I started to get the basics of developing mobile apps. Thank you, Ms. Phan Thanh Tra, for helping me during my studies.