Mobile Application Design and Development- CW2 (Logbook)

Student name: Nguyễn Quang Hải Ninh

**ID: 001201689**

Contents

[I. Exercise 1 4](#_Toc88058298)

[1. Basic Information 4](#_Toc88058299)

[2. Exercise answer 5](#_Toc88058300)

[2.1. Screen shots demonstrating what achieved 5](#_Toc88058301)

[2.2. Code 6](#_Toc88058302)

[II. Exercise 2 8](#_Toc88058303)

[1. Basic Information 8](#_Toc88058304)

[2. Exercise answer 9](#_Toc88058305)

[2.1. Screen shots demonstrating what achieved 9](#_Toc88058306)

[2.2. Code 22](#_Toc88058307)

[III. Exercise 3 38](#_Toc88058308)

[1. Basic Information 38](#_Toc88058309)

[2. Exercise answer 39](#_Toc88058310)

[2.1. Screen shots demonstrating what achieved 39](#_Toc88058311)

[2.2. Code 49](#_Toc88058312)

[IV. Exercise 4 53](#_Toc88058313)

[1. Basic Information 53](#_Toc88058314)

[2. Exercise answer 53](#_Toc88058315)

[2.1. Screen shots demonstrating what achieved 53](#_Toc88058316)

[2.2. Code 62](#_Toc88058317)

[Link Google drive to download source code 68](#_Toc88058318)

[Figure 1: Native functions page 5](#_Toc87997446)

[Figure 2: Entry data page 9](#_Toc87997447)

[Figure 3: Notice that address is required field 10](#_Toc87997448)

[Figure 4: Notice that property type is required field 11](#_Toc87997449)

[Figure 5: Notice that bedrooms is required field 12](#_Toc87997450)

[Figure 6: Notice that price is required field 13](#_Toc87997451)

[Figure 7: Notice that name of reporter is required field 14](#_Toc87997452)

[Figure 8: Notice that the address description cannot contain special characters 15](#_Toc87997453)

[Figure 9: Notice that the bedrooms description cannot contain special characters 16](#_Toc87997454)

[Figure 10: Notice that If user selects more for bedrooms, user is required to enter more description for bedrooms 17](#_Toc87997455)

[Figure 11: Notice that the data type of price can only be number 18](#_Toc87997456)

[Figure 12: Notice about required fields are displayed below input 19](#_Toc87997457)

[Figure 13: Notice that if a user enters a property address that matches a previously added address 20](#_Toc87997458)

[Figure 14: Notice that rental has been successfully added 21](#_Toc87997459)

[Figure 15: Home page 40](#_Toc87997460)

[Figure 16: Notice that rental has been successfully added 41](#_Toc87997461)

[Figure 17: Home page after added 42](#_Toc87997462)

[Figure 18: Details page 43](#_Toc87997463)

[Figure 19: Details page after change price 44](#_Toc87997464)

[Figure 20: Home page after edit 45](#_Toc87997465)

[Figure 21: Notice for users to confirm deletion of real estate 46](#_Toc87997466)

[Figure 22: Notification of successful deletion 47](#_Toc87997467)

[Figure 23: Home page after delete 48](#_Toc87997468)

[Figure 24: Check input data when editing 49](#_Toc87997469)

[Figure 25: Android app data entry screen 54](#_Toc87997470)

[Figure 26: Android app data input screen when error message is reported 55](#_Toc87997471)

[Figure 27: Notice that address is required field 56](#_Toc87997472)

[Figure 28: Notice that property type is required field 57](#_Toc87997473)

[Figure 29: Notice that bedrooms is required field 58](#_Toc87997474)

[Figure 30: Notice that price is required field 59](#_Toc87997475)

[Figure 31: Notice that name of reporter is required field 60](#_Toc87997476)

[Figure 32: Android app data added screen 61](#_Toc87997477)

# I. Exercise 1

## 1. Basic Information

|  |  |
| --- | --- |
| 1.1. Student name | **Nguyễn Quang Hải Ninh** |
| 1.2. Who did you work with? | **Name:**  **Login id:** |
| 1.3. Which Exercise is this? | Create a PhoneGap App or any other similar platform utilizing the Notification API |
| 1.4. How well did you complete the exercise? | I did everything that was asked |
| 1.5. Briefly explain your answer to question | The application is installed with the react-audio-player library. A page that allows the user to play a music with 2 buttons play to play the favorite song and pause to stop the currently playing ringtone. A button vibrates when the user presses the device vibrates for 3 seconds. |

## 2. Exercise answer

### 2.1. Screen shots demonstrating what achieved

In the native tab there are 3 buttons in which play button and pause button to play and stop the ringtone and vibrate to ring the bell

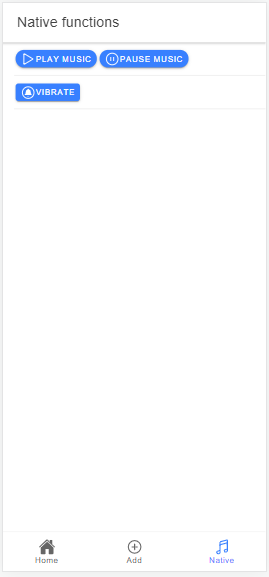


Figure 1: Native functions page

### 2.2. Code

The components of ionic used include IonButton, IonContent, IonHeader, IonIcon, IonItem, IonPage, IonTitle and IonToolbar. Imported in the first line to be able to use. 3 icons include playOutline that is used to play button, pauseCircleOutline that is used to pause button and notificationsCircleOutline that is used to vibrate.

import { IonButton, IonContent, IonHeader, IonIcon, IonItem, IonPage, IonTitle, IonToolbar } from '@ionic/react';

import ReactAudioPlayer from 'react-audio-player';

import { playOutline, pauseCircleOutline, notificationsCircleOutline} from 'ionicons/icons';

const Native: React.FC = () => {

    var musicPlayer: ReactAudioPlayer | null

    return (

        <IonPage>

          <IonHeader>

            <IonToolbar>

              <IonTitle>Native functions</IonTitle>

            </IonToolbar>

          </IonHeader>

          <IonContent fullscreen>

            <IonItem>

              <IonButton shape="round"  onClick={() => musicPlayer?.audioEl.current?.play()}>

              <IonIcon slot="icon-only" icon={playOutline}></IonIcon>

                  Play music</IonButton>

              <IonButton shape="round" onClick={() => musicPlayer?.audioEl.current?.pause()}>

              <IonIcon slot="icon-only" icon={pauseCircleOutline}></IonIcon>

                  Pause music</IonButton>

              <ReactAudioPlayer

                ref={(element) => { musicPlayer = element; }}

                src="./assets/LetItBeMe-SteveAokiBackstreetBoys.mp3"

              />

            </IonItem>

            <IonItem>

              <IonButton onClick={() => navigator.vibrate(3000)}>

              <IonIcon slot="icon-only" icon={notificationsCircleOutline}></IonIcon>

                  Vibrate</IonButton>

            </IonItem>

            </IonContent>

    </IonPage>

    );

}

export default Native;

The event onClick() of the play button is assigned the function to play the ringtone so when the user presses the play button the ringtone will be played. Similarly, when the pause button is pressed, the ringtone will stop playing because the onClick() event of the pause button is used for the pause function. with Vibrate button when user presses vibrate will be activated 3000 milliseconds.

# II. Exercise 2

## 1. Basic Information

|  |  |
| --- | --- |
| 1.1. Student name | **Nguyễn Quang Hải Ninh** |
| 1.2. Who did you work with? | **Name:**  **Login id:** |
| 1.3. Which Exercise is this? | Create a PhoneGap App data entry screen |
| 1.4. How well did you complete the exercise? | I did more than was asked for |
| 1.5. Briefly explain your answer to question | A form allows the user to enter all the required data in part 1 a) of the coursework section. The user will receive a notification if the required fields are left blank. In addition, the input data is also checked, the user cannot enter special characters in the name of reporter, price, notes. In principally, the user can only enter the number in the price and the value must be greater than zero, otherwise the system will notify |

## 2. Exercise answer

### 2.1. Screen shots demonstrating what achieved

Information page with fields including address, property type, bedrooms, date and time of adding the property, monthly rent price, furniture types, notes, name of the reporter.

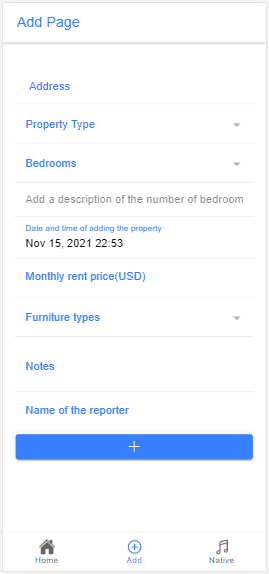


Figure 2: Entry data page

Notify when user press add without entering address that is required field.

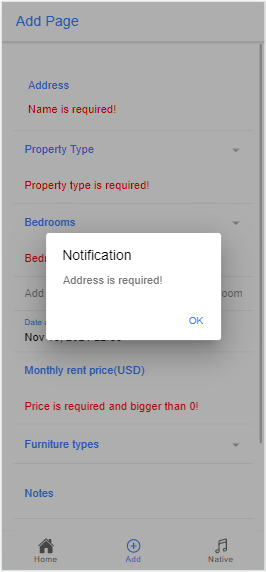


Figure 3: Notice that address is required field

Notify when user press add without entering property type that is required field.

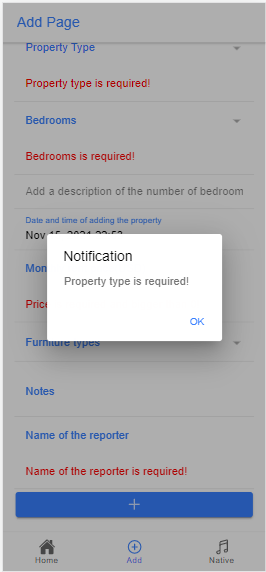


Figure 4: Notice that property type is required field

Notify when user press add without entering bedrooms that is required field.

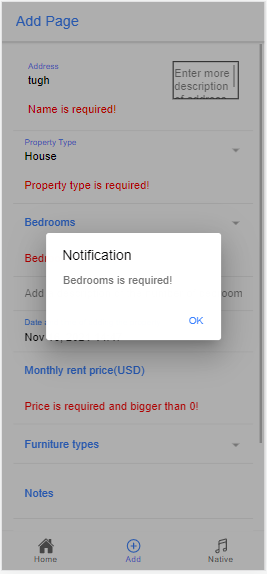


Figure 5: Notice that bedrooms is required field

Notify when user press add without entering bedrooms that is required field or value is less than zero

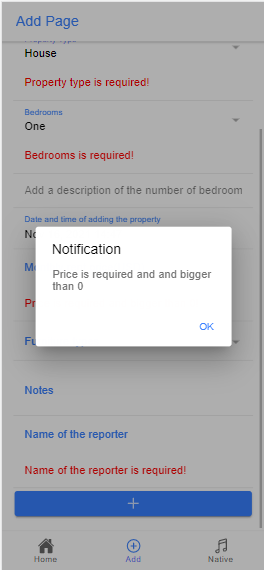


Figure 6: Notice that price is required field

Notify when user press add without entering name of reporter that is required field.

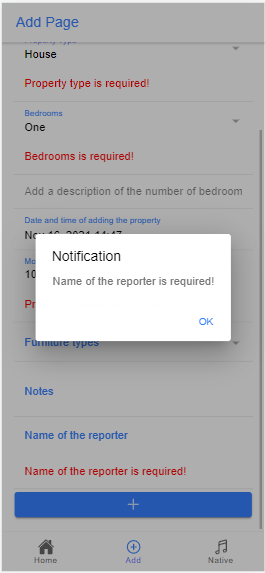


Figure 7: Notice that name of reporter is required field

Notify when the user enters special characters in the notes for the address

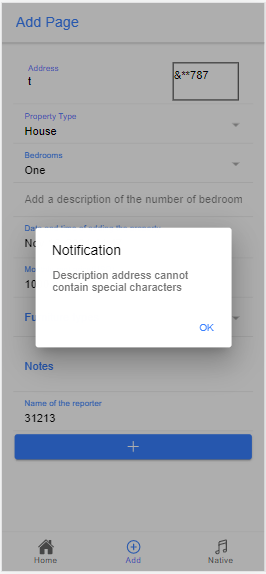


Figure 8: Notice that the address description cannot contain special characters

Notify when the user enters special characters in the notes for the bedrooms

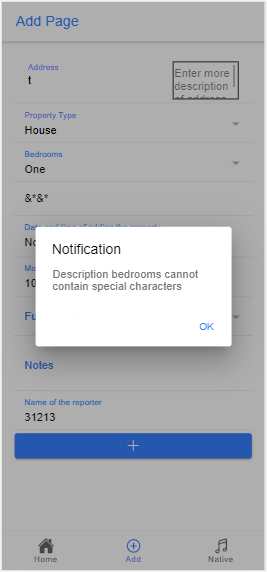


Figure 9: Notice that the bedrooms description cannot contain special characters

If the user selects more in the bedrooms section, the extra description of the bedroom will become required field and if the user leaves it blank and add will get the message as below.

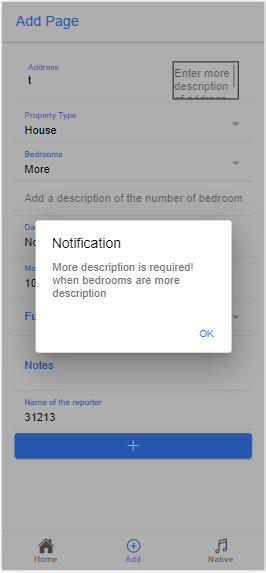


Figure 10: Notice that if user selects more for bedrooms, user is required to enter more description for bedrooms

Report when the user enters non-numeric data in the price section

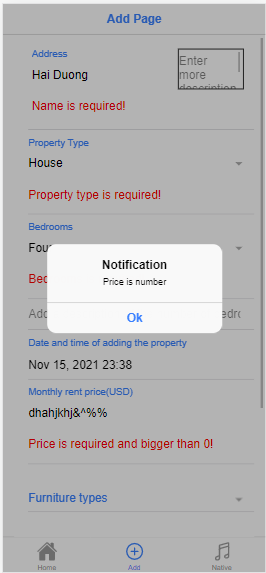


Figure 11: Notice that the data type of price can only be number

The message below the input tells the user which is required fields need to enter correctly and completely to add

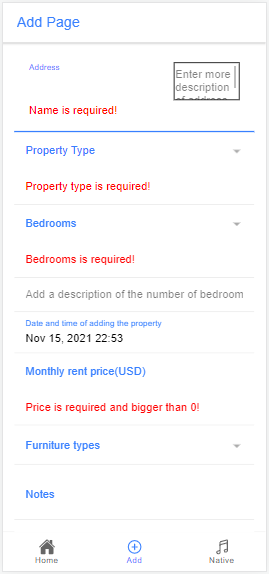


Figure 12: Notice about required fields are displayed below input

Notify if the property address has been previously added

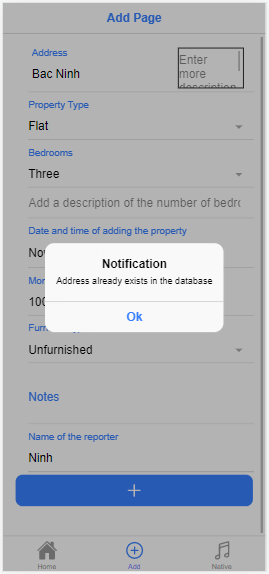


Figure 13: Notice that if a user enters a property address that matches a previously added address

Notice that the property has been successfully added

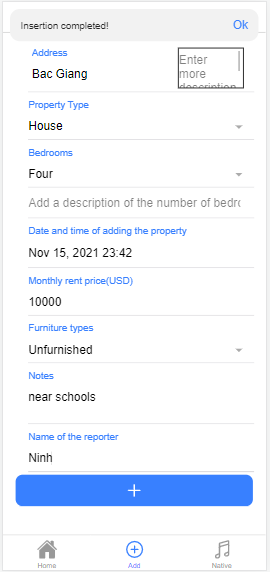


Figure 14: Notice that rental has been successfully added

### 2.2. Code

Libraries, ionic components, hooks, routers, insertRental functions, getAllRental page to get data to compare input and add data to data. Hooks are used to declare additional values as requested from 1 a) in the coursework. Refresh to refresh the page after adding to update the newly added data. History to return to the homepage after adding. presentToast to display the successfully added message. presentAlert displays a message in case of invalid input. specialCharacters are used to check for special characters when entering notes. checkNameCharacters to check name of reporter when user enters number or special character.

import { IonButton, IonCol, IonContent, IonDatetime, IonHeader, IonIcon, IonInput, IonItem, IonLabel, IonNote, IonPage, IonRow, IonSelect, IonSelectOption, IonTextarea, IonTitle, IonToolbar, useIonAlert, useIonToast } from '@ionic/react';

import { addOutline } from 'ionicons/icons';

import React from 'react';

import { useState, useEffect } from 'react';

import { useHistory } from 'react-router';

import { insertRental, getAllRental } from '../databaseHandler'

import { Rental } from '../models';

const Add: React.FC = () => {

  const [address, setAddress] = useState<string>('');

  const [propertyType, setPropertyType] = useState('');

  const [bedrooms, setBedrooms] = useState('');

  const [dateAndTime, setDateAndTime] = useState(new Date().toISOString());

  const [price, setPrice] = useState<number>();

  const [furnitureTypes, setFurnitureTypes] = useState('');

  const [notes, setNotes] = useState<string>('');

  const [name, setName] = useState<string>('');

  const [descriptionA, setDescriptionA] = useState<string>('');

  const [descriptionB, setDescriptionB] = useState<string>('');

  const [rentals, setRentals] = useState<Rental[]>([])

  const [checkValid, setCheckValid] = useState(false)

  const [refresh, setRefresh] = useState(false)

  const history = useHistory();

  const [presentToast, dismissToast] = useIonToast();

  const [presentAlert, dismissAlert] = useIonAlert();

  var specialCharacters = /[`!@#$%^&\*()\_+\-=\[\]{};':"\\|,.<>\/?~]/

  var checkNameCharacters = /[0123456789`!@#$%^&\*()\_+\-=\[\]{};':"\\|,.<>\/?~]/

Check input if required field is left blank. The announcement will display that color message line below the input. The function mainly compares the length of the input data if length == 0, the system will report an error. For price because it is a number type, the condition is price = 0

function check() {

    if (address.length == 0 && checkValid) {

      return false

    }

    else if (propertyType.length == 0 && checkValid) {

      return false

    }

    else if (bedrooms.length == 0 && checkValid) {

      return false

    }

    else if (price == 0 || price == null && checkValid) {

      return false

    }

    else if (name.length == 0 && checkValid) {

      return false

    }

    else

      return true;

  }

Function to check if the property address already exists. If the property address already exists, an error will be reported. A checkDuplicate variable is declared with a value equal to Not duplicate. forEach() is used to compare each address of previously added properties in the array with the data being added. If they match then checkDuplicate = 'Duplicate' and will give an error.

 function checkForDuplicate() {

    var checkDuplicate = "Not duplicate"

    rentals.forEach((r) => {

      if (r.address == address) {

        checkDuplicate = 'Duplicate';

      }

    })

    return checkDuplicate;

  }

Function to add data, in this function, if the data of the required fields is left blank, the system will report an error through IonAlert. If the input has met all the conditions, the property will be successfully added and the user will be notified via IonToast. When the user presses ok, the user will be returned to the home page by typing on the command line onDidDismiss: () => history.push("/Home").

const addClick = () => {

    setCheckValid(true)

    if (address.trim().length == 0) {

      presentAlert({

        header: 'Notification',

        message: 'Address is required!',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (specialCharacters.test(descriptionA)) {

      presentAlert({

        header: 'Notification',

        message: 'Description address cannot contain special characters',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (propertyType.trim().length == 0) {

      presentAlert({

        header: 'Notification',

        message: 'Property type is required!',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (bedrooms.trim().length == 0) {

      presentAlert({

        header: 'Notification',

        message: 'Bedrooms is required!',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (bedrooms == "More description" && descriptionB.trim().length == 0) {

      presentAlert({

        header: 'Notification',

        message: 'More description is required! when bedrooms are more description',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (specialCharacters.test(descriptionB)) {

      presentAlert({

        header: 'Notification',

        message: 'Description bedrooms cannot contain special characters',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (price == 0 || price == null) {

      presentAlert({

        header: 'Notification',

        message: 'Price is required and and bigger than 0',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (Number.isNaN(price)) {

      presentAlert({

        header: 'Notification',

        message: 'Price is number',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (specialCharacters.test(notes)) {

      presentAlert({

        header: 'Notification',

        message: 'Notes cannot contain special characters',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (name.trim().length == 0) {

      presentAlert({

        header: 'Notification',

        message: 'Name of the reporter is required!',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (checkNameCharacters.test(name)) {

      presentAlert({

        header: 'Notification',

        message: 'Name of the reporter cannot be numeric and contain special characters',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (checkForDuplicate() == 'Duplicate') {

      presentAlert({

        header: 'Notification',

        message: 'Address already exists in the database',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else {

      const newRental = { address: address, propertyType: propertyType, bedrooms: bedrooms, dateAndTime: dateAndTime, price: price, furnitureTypes: furnitureTypes, notes: notes, name: name, descriptionA: descriptionA, descriptionB: descriptionB }

      insertRental(newRental);

      setRefresh(!refresh)

      presentToast({

        position: "top", buttons: [{ text: 'Ok', handler: () => dismissToast() }],

        message: 'Insertion completed!',

        onDidDismiss: () => history.push("/Home"),

      })

    }

  }

Get the data in the array to compare the input data to see if the address already exists

 async function fetchData() {

    const rentals = await getAllRental() as Rental[]

    setRentals(rentals)

  }

  useEffect(() => {

    fetchData();

  }, [])

UI ionic components display labels, inputs for users to enter data, buttons.

return (

    <IonPage>

      <IonHeader>

        <IonToolbar>

          <IonTitle color="primary">Add Page</IonTitle>

        </IonToolbar>

      </IonHeader >

      <IonContent className="ion-padding">

        <IonItem>

          <IonRow>

            <IonCol size='8'>

              <IonLabel position="floating" color="primary" >Address</IonLabel>

              <IonInput value={address} onIonChange={event => setAddress(event.detail.value!)}></IonInput>

              {!check() && <p style={{ color: "red" }}>Name is required!</p>}

            </IonCol>

            <IonCol>

              <IonTextarea className="textarea" value={descriptionA} placeholder="Enter more description of address" onIonChange={e => setDescriptionA(e.detail.value!)}></IonTextarea>

            </IonCol>

          </IonRow>

        </IonItem>

        <IonItem>

          <IonLabel color="primary" position="floating">Property Type</IonLabel>

          <IonSelect interface="popover" value={propertyType} onIonChange={event => setPropertyType(event.detail.value)}>

            <IonSelectOption value="flat">Flat</IonSelectOption>

            <IonSelectOption value="house">House</IonSelectOption>

            <IonSelectOption value="bungalow">Bungalow</IonSelectOption>

          </IonSelect>

          {!check() && <p style={{ color: "red" }}>Property type is required!</p>}

        </IonItem>

        <IonItem >

          <IonLabel color="primary" position="floating">Bedrooms</IonLabel>

          <IonSelect interface="popover" value={bedrooms} onIonChange={event => setBedrooms(event.detail.value)}>

            <IonSelectOption value="studio">Studio</IonSelectOption>

            <IonSelectOption value="One">One</IonSelectOption>

            <IonSelectOption value="Two">Two</IonSelectOption>

            <IonSelectOption value="Three">Three</IonSelectOption>

            <IonSelectOption value="Four">Four</IonSelectOption>

            <IonSelectOption value="Five">Five</IonSelectOption>

            <IonSelectOption value="More description">More</IonSelectOption>

          </IonSelect>

          {!check() && <p style={{ color: "red" }}>Bedrooms is required!</p>}

        </IonItem>

        <IonItem>

          <IonInput value={descriptionB} placeholder="Add a description of the number of bedrooms" onIonChange={event => setDescriptionB(event.detail.value!)}></IonInput>

        </IonItem>

        <IonItem>

          <IonLabel color="primary" position="floating">Date and time of adding the property</IonLabel>

          <IonDatetime readonly={true} displayFormat="MMM DD, YYYY HH:mm" onIonChange={e => setDateAndTime(e.detail.value!)} value={dateAndTime}></IonDatetime>

        </IonItem>

        <IonItem>

          <IonLabel position="floating" color="primary">Monthly rent price(USD)</IonLabel>

          <IonInput onIonChange={event => setPrice(Number(event.detail.value!))}></IonInput>

          {!check() && <p style={{ color: "red" }}>Price is required and bigger than 0!</p>}

        </IonItem>

        <IonItem>

          <IonLabel color="primary" position="floating">Furniture types</IonLabel>

          <IonSelect interface="popover" value={furnitureTypes} onIonChange={event => setFurnitureTypes(event.detail.value)}>

            <IonSelectOption value="furnished">Furnished</IonSelectOption>

            <IonSelectOption value="unfurnished">Unfurnished</IonSelectOption>

            <IonSelectOption value="partFurnished">Part Furnished</IonSelectOption>

          </IonSelect>

        </IonItem>

        <IonItem>

          <IonLabel color="primary" position="floating">Notes</IonLabel>

          <IonTextarea value={notes} onIonChange={e => setNotes(e.detail.value!)}></IonTextarea>

        </IonItem>

        <IonItem>

          <IonLabel color="primary" position="floating">Name of the reporter</IonLabel>

          <IonInput value={name} onIonChange={e => setName(e.detail.value!)}></IonInput>

          {!check() && <p style={{ color: "red" }}>Name of the reporter is required!</p>}

        </IonItem>

        <IonButton expand="block" onClick={addClick} >

          <IonIcon slot="icon-only" icon={addOutline}></IonIcon>

        </IonButton>

      </IonContent>

    </IonPage>

  );

};

export default Add;

Code of details page to review added rental information based on ID. getAllRental() to get data from rental and compare if there are duplicate addresses, getRentalById() is used to get information of previously added property based on ID. handleDelete() is a function to delete rental if the user needs it when clicking on delete buttons the function will be called and the user will receive an alert to confirm the operation.

import { IonBackButton, IonButton, IonButtons, IonCol, IonContent, IonDatetime, IonHeader, IonIcon, IonInput, IonItem, IonLabel, IonNote, IonPage, IonRow, IonSelect, IonSelectOption, IonTextarea, IonTitle, IonToolbar, useIonAlert, useIonToast } from '@ionic/react';

import { addOutline, trashSharp } from 'ionicons/icons';

import { useEffect, useState } from 'react';

import { useHistory, useParams } from 'react-router';

import { updateRental, getRentalById, deleteRental, getAllRental } from '../databaseHandler'

import { Rental } from '../models';

interface MyParams {

  id: string

}

const Details: React.FC = () => {

  const { id } = useParams<MyParams>()

  const [address, setAddress] = useState<string>('')

  const [propertyType, setPropertyType] = useState('')

  const [bedrooms, setBedrooms] = useState('')

  const [dateAndTime, setDateAndTime] = useState(new Date().toISOString());

  const [price, setPrice] = useState<number>();

  const [furnitureTypes, setFurnitureTypes] = useState('');

  const [notes, setNotes] = useState<string>('')

  const [name, setName] = useState<string>('')

  const [descriptionA, setDescriptionA] = useState<string>('')

  const [descriptionB, setDescriptionB] = useState<string>('')

  const [rentals, setRentals] = useState<Rental[]>([])

  const [checkValid, setCheckValid] = useState(false)

  const [refresh, setRefresh] = useState(false)

  const [presentToast, dismissToast] = useIonToast();

  const [presentAlert, dismissAlert] = useIonAlert();

  var specialCharacters = /[`!@#$%^&\*()\_+\-=\[\]{};':"\\|,.<>\/?~]/

  var checkNameCharacters = /[0123456789`!@#$%^&\*()\_+\-=\[\]{};':"\\|,.<>\/?~]/

  const history = useHistory();

  function checkForDuplicate(id: string) {

    var checkDuplicate = "Not duplicate"

    rentals.forEach((r) => {

      if (r.address == address && r.id != Number.parseInt(id)) {

        checkDuplicate = 'Duplicate'

      }

    })

    return checkDuplicate;

  }

  function check() {

    if (address.length == 0 && checkValid) {

      return false

    }

    if (propertyType.length == 0 && checkValid) {

      return false

    }

    if (bedrooms.length == 0 && checkValid) {

      return false

    }

    if (price == 0 || price == null && checkValid) {

      return false

    }

    if (name.length == 0 && checkValid) {

      return false

    }

    else

      return true;

  }

  const handleUpdate = () => {

    setCheckValid(true)

    if (address.trim().length == 0) {

      presentAlert({

        header: 'Notification',

        message: 'Address is required!',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (specialCharacters.test(descriptionA)) {

      presentAlert({

        header: 'Notification',

        message: 'Description address cannot contain special characters',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (propertyType.trim().length == 0) {

      presentAlert({

        header: 'Notification',

        message: 'Property type is required!',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (bedrooms.trim().length == 0) {

      presentAlert({

        header: 'Notification',

        message: 'Bedrooms is required!',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (bedrooms == "More description" && descriptionB.trim().length == 0) {

      presentAlert({

        header: 'Notification',

        message: 'More description is required! when bedrooms are more description',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (specialCharacters.test(descriptionB)) {

      presentAlert({

        header: 'Notification',

        message: 'Description bedrooms cannot contain special characters',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (price == 0 || price == null) {

      presentAlert({

        header: 'Notification',

        message: 'Price is required and and bigger than 0',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (Number.isNaN(price)) {

      presentAlert({

        header: 'Notification',

        message: 'Price is number',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (specialCharacters.test(notes)) {

      presentAlert({

        header: 'Notification',

        message: 'Notes cannot contain special characters',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (name.trim().length == 0) {

      presentAlert({

        header: 'Notification',

        message: 'Name of the reporter is required!',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (checkNameCharacters.test(name)) {

      presentAlert({

        header: 'Notification',

        message: 'Name of the reporter cannot be numeric and contain special characters',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else if (checkForDuplicate(id) == 'Duplicate') {

      presentAlert({

        header: 'Notification',

        message: 'Address already exists in the database',

        buttons: [{ text: 'Ok', handler: () => dismissAlert() }]

      })

    }

    else {

      const newRental = {

        id: Number.parseInt(id),

        address: address, propertyType: propertyType,

        bedrooms: bedrooms, dateAndTime: dateAndTime, price: price,

        furnitureTypes: furnitureTypes, notes: notes, name: name, descriptionA: descriptionA, descriptionB: descriptionB

      }

      updateRental(newRental);

      setRefresh(!refresh)

      presentToast({

        position: "top", buttons: [{ text: 'Ok', handler: () => dismissToast() }],

        message: 'Update done!',

        onDidDismiss: () => history.push("/Home"),

      })

    }

  }

  function handleDelete() {

    deleteRental(Number.parseInt(id))

    presentToast({

      position: 'top',

      buttons: [{ text: 'OK', handler: () => dismissToast() }],

      message: 'Deleteion done!',

      onDidDismiss: () => history.goBack(),

    })

  }

  async function fetchData() {

    const rentals = await getAllRental() as Rental[]

    setRentals(rentals)

    const rental = await getRentalById(Number.parseInt(id)) as Rental

    setAddress(rental.address);

    setPropertyType(rental.propertyType)

    setBedrooms(rental.bedrooms)

    setDateAndTime(rental.dateAndTime)

    setPrice(rental.price)

    setFurnitureTypes(rental.furnitureTypes)

    setNotes(rental.notes)

    setName(rental.name)

    setDescriptionA(rental.descriptionA)

    setDescriptionB(rental.descriptionB)

  }

  useEffect(() => {

    fetchData();

  }, [])

  return (

    <IonPage>

      <IonHeader>

        <IonToolbar>

          <IonButtons slot="start">

            <IonBackButton />

          </IonButtons>

          <IonTitle color="primary" >Details {id}</IonTitle>

          <IonButton onClick={() =>

            presentAlert({

              header: 'Notification',

              message: 'Are you sure to delete?',

              buttons: ['Cancel', { text: 'Confirm', handler: () => handleDelete() }],

            })

          } size="small" color="danger" slot="end">

            <IonIcon slot="icon-only" icon={trashSharp}></IonIcon>

          </IonButton>

        </IonToolbar>

      </IonHeader>

      <IonContent className="ion-padding">

        <IonItem>

          <IonRow>

            <IonCol size='8'>

              <IonLabel position="floating" color="primary" >Address</IonLabel>

              <IonInput value={address} onIonChange={event => setAddress(event.detail.value!)}></IonInput>

              {!check() && <p style={{ color: "red" }}>Name is required!</p>}

            </IonCol>

            <IonCol>

              <IonTextarea className="textarea" value={descriptionA} placeholder="Enter more description of address" onIonChange={e => setDescriptionA(e.detail.value!)}></IonTextarea>

            </IonCol>

          </IonRow>

        </IonItem>

        <IonItem>

          <IonLabel color="primary" position="floating">Property Type</IonLabel>

          <IonSelect interface="popover" value={propertyType} onIonChange={event => setPropertyType(event.detail.value)}>

            <IonSelectOption value="flat">Flat</IonSelectOption>

            <IonSelectOption value="house">House</IonSelectOption>

            <IonSelectOption value="bungalow">Bungalow</IonSelectOption>

          </IonSelect>

          {!check() && <p style={{ color: "red" }}>Property type is required!</p>}

        </IonItem>

        <IonItem >

          <IonLabel color="primary" position="floating">Bedrooms</IonLabel>

          <IonSelect interface="popover" value={bedrooms} onIonChange={event => setBedrooms(event.detail.value)}>

            <IonSelectOption value="studio">Studio</IonSelectOption>

            <IonSelectOption value="One">One</IonSelectOption>

            <IonSelectOption value="Two">Two</IonSelectOption>

            <IonSelectOption value="Three">Three</IonSelectOption>

            <IonSelectOption value="Four">Four</IonSelectOption>

            <IonSelectOption value="Five">Five</IonSelectOption>

            <IonSelectOption value="More description">More</IonSelectOption>

          </IonSelect>

          {!check() && <p style={{ color: "red" }}>Bedrooms is required!</p>}

          <IonItem>

          <IonInput value={descriptionB} placeholder="Add a description of the number of bedrooms" onIonChange={event => setDescriptionB(event.detail.value!)}></IonInput>

        </IonItem>

        </IonItem>

        <IonItem>

          <IonLabel color="primary" position="floating">Date and time of adding the property</IonLabel>

          <IonDatetime readonly={true} displayFormat="MMM DD, YYYY HH:mm" onIonChange={e => setDateAndTime(e.detail.value!)} value={dateAndTime}></IonDatetime>

        </IonItem>

        <IonItem>

          <IonLabel position="floating" color="primary">Monthly rent price(USD)</IonLabel>

          <IonInput value={price} onIonChange={event => setPrice(Number(event.detail.value!))}></IonInput>

          {!check() && <p style={{ color: "red" }}>Price is required and bigger than 0!</p>}

        </IonItem>

        <IonItem>

          <IonLabel color="primary" position="floating">Furniture types</IonLabel>

          <IonSelect interface="popover" value={furnitureTypes} onIonChange={event => setFurnitureTypes(event.detail.value)}>

            <IonSelectOption value="furnished">Furnished</IonSelectOption>

            <IonSelectOption value="unfurnished">Unfurnished</IonSelectOption>

            <IonSelectOption value="partFurnished">Part Furnished</IonSelectOption>

          </IonSelect>

        </IonItem>

        <IonItem>

          <IonLabel color="primary" position="floating">Notes</IonLabel>

          <IonTextarea value={notes} onIonChange={e => setNotes(e.detail.value!)}></IonTextarea>

        </IonItem>

        <IonItem>

          <IonLabel color="primary" position="floating">Name of the reporter</IonLabel>

          <IonInput value={name} onIonChange={e => setName(e.detail.value!)}></IonInput>

          {!check() && <p style={{ color: "red" }}>Name of the reporter is required!</p>}

        </IonItem>

        <IonButton color="primary" expand="block" onClick={handleUpdate} >

          <IonIcon slot="icon-only" icon={addOutline}></IonIcon>

        </IonButton>

      </IonContent>

    </IonPage>

  );

};

export default Details;

# III. Exercise 3

## 1. Basic Information

|  |  |
| --- | --- |
| 1.1. Student name | **Nguyễn Quang Hải Ninh** |
| 1.2. Who did you work with? | **Name:**  **Login id:** |
| 1.3. Which Exercise is this? | Create an SQLite database to store the event details entered into the RentalZ App |
| 1.4. How well did you complete the exercise? | I did everything that was asked |
| 1.5. Briefly explain your answer to question | The idb library is installed and to store data in the Rental array containing the fields required in the coursework section 1 a) and declared in the models.ts file. To store data some functions in databaseHandler file are also created to store data such as insertRental(), initDB(), getAllRental(), getRentalById(), updateRental() and deleteRental(). Moreover, users can also edit and delete the added data |

## 2. Exercise answer

### 2.1. Screen shots demonstrating what achieved

The home page displays a list of added real estate addresses with monthly rental rates

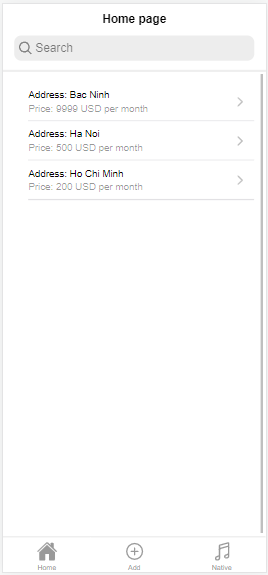


Figure 15: Home page

A property with the information entered as shown in the photo has been successfully notified

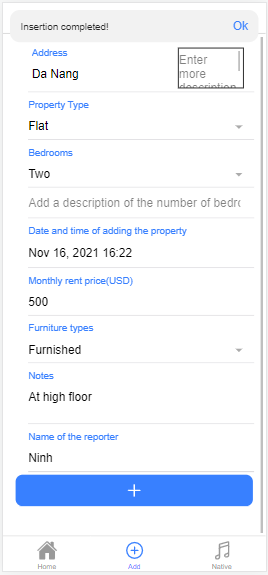


Figure 16: Notice that rental has been successfully added

Real estate "Da Nang" has been added to the system and appears in the list on the home page

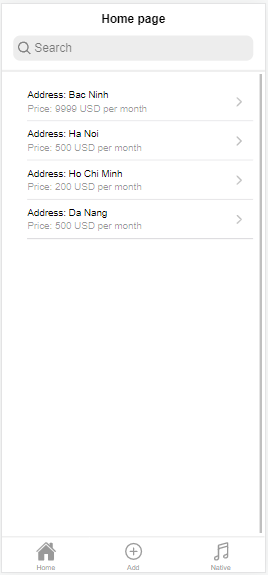


Figure 17: Home page after added

The application allows users to review the information entered by clicking on the label with the name of the property address in the list on the home page.

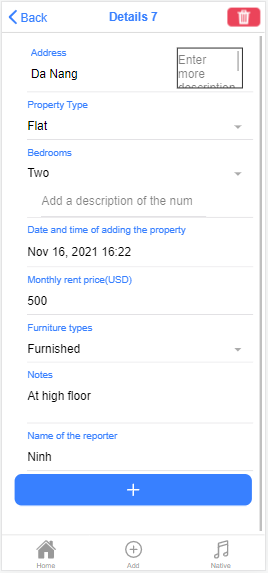


Figure 18: Details page

Users can edit the saved information, for example, fix the price of Da Nang real estate from 500 to 600 then press update to edit the property's data.

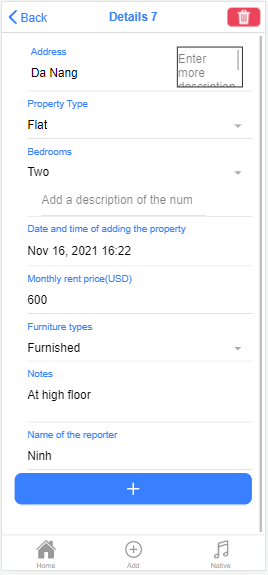


Figure 19: Details page after change price

Return to the listing page on the homepage to see that the price of Da Nang real estate has been fixed to 600

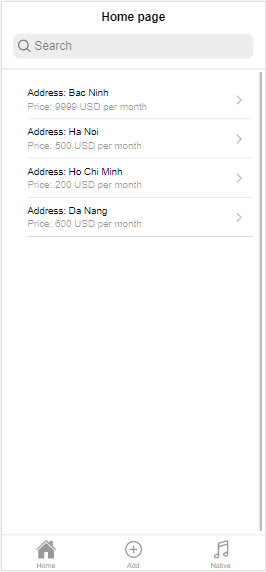


Figure 20: Home page after edit

If the user wants to delete the data, click the red trash can delete button in the top right corner, a message will appear for the user to confirm. Users can press cancel to cancel the operation or press confirm to confirm deletion

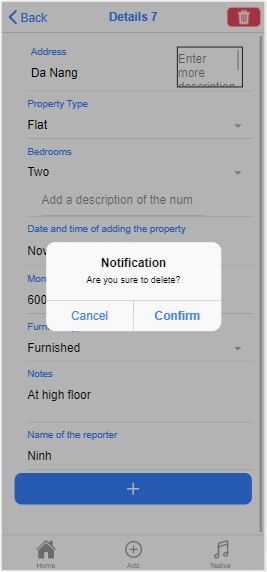


Figure 21: Notice for users to confirm deletion of real estate

The application notices that the deletion has been successful, press ok to confirm the message and return to the homepage

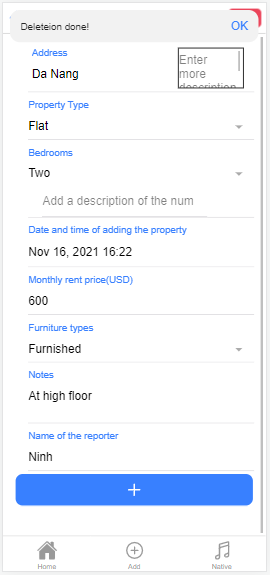


Figure 22: Notification of successful deletion

Back to the homepage, Da Nang real estate has been removed from the data

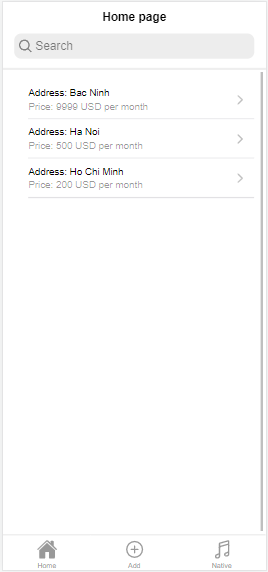


Figure 23: Home page after delete

Similar to add, editing is also checked for input data. For example, price input data is not numeric, user will receive message

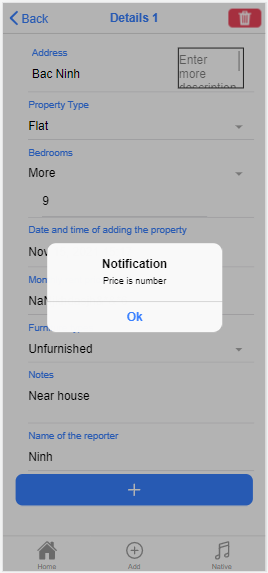


Figure 24: Check input data when editing

### 2.2. Code

Rental array with fields such as address: string, propertyType: string, bedrooms: string,  dateAndTime: string, price? : number, furnitureTypes: string, notes: string, name: string, descriptionA: string, descriptionB: string.

export interface Rental{

    id? : number,

    address: string,

    propertyType: string,

    bedrooms: string,

    dateAndTime: string,

    price? : number,

    furnitureTypes: string,

    notes: string,

    name: string,

    descriptionA: string,

    descriptionB: string

}

idp library is used, the database on the Rental DB is declared.

import { openDB } from 'idb'

import { Rental } from './models';

const DATABASE\_NAME = "RentalZDB";

initDB().then(() => {

    console.log("database initialized!")

  })

Update information of rentals function

export async function updateRental(rentalToUpdate:Rental) {

    const db = await openDB(DATABASE\_NAME, 1)

    const ren= await db.transaction("rentals").

          objectStore("rentals").get(rentalToUpdate.id!) as Rental

    ren.address = rentalToUpdate.address

    ren.propertyType = rentalToUpdate.propertyType

    ren.bedrooms = rentalToUpdate.bedrooms

    ren.dateAndTime = rentalToUpdate.dateAndTime

    ren.price = rentalToUpdate.price

    ren.furnitureTypes = rentalToUpdate.furnitureTypes

    ren.notes = rentalToUpdate.notes

    ren.name =  rentalToUpdate.name

    ren.descriptionA = rentalToUpdate.descriptionA

    ren.descriptionB = rentalToUpdate.descriptionB

    await db.put("rentals",ren);

  }

Delete information of rentals function

export async function deleteRental(id:number) {

    const db = await openDB(DATABASE\_NAME, 1)

    await db.delete("rentals",id)

  }

Get the rental data added by id to display the listing on the homepage and edit the data

export async function getRentalById(id:number) {

    const db = await openDB(DATABASE\_NAME, 1);

    const cus= await db.transaction("rentals").objectStore("rentals").get(id);

    return cus;

  }

Get all information of rental in database

 export async function getAllRental() {

    const db = await openDB(DATABASE\_NAME, 1);

    var cursor = await db.transaction("rentals").

              objectStore("rentals").openCursor();

    var customers = [];

    while (cursor) {

      customers.push(cursor.value);

      cursor = await cursor.continue();

    }

    return customers;

  }

Add information of rental to database

 export async function insertRental(rental: Rental) {

    const db = await openDB(DATABASE\_NAME, 1)

    const tx = db.transaction('rentals', 'readwrite');

    const store = tx.objectStore('rentals');

    await store.put(rental)

  }

  async function initDB() {

    const db = await openDB(DATABASE\_NAME, 1, {

      upgrade(db) {

        const store = db.createObjectStore('rentals', {

          keyPath: 'id',

          autoIncrement: true,

        });

      },

    });

  }

# IV. Exercise 4

## 1. Basic Information

|  |  |
| --- | --- |
| 1.1. Student name | **Nguyễn Quang Hải Ninh** |
| 1.2. Who did you work with? | **Name:**  **Login id:** |
| 1.3. Which Exercise is this? | Create Android data entry screen |
| 1.4. How well did you complete the exercise? | I did everything that was asked |
| 1.5. Briefly explain your answer to question | An android data entry screen for people to enter all the fields specified in the coursework section 1 a). The data is checked input data if required fields are left blank, the system will notify |

## 2. Exercise answer

### 2.1. Screen shots demonstrating what achieved

Screen to user to enter data for fields including property type, bedrooms, date and time of adding the property, monthly rent price, furniture types, notes, name of the reporter. In addition, a button and a textview display the information the user has just entered.

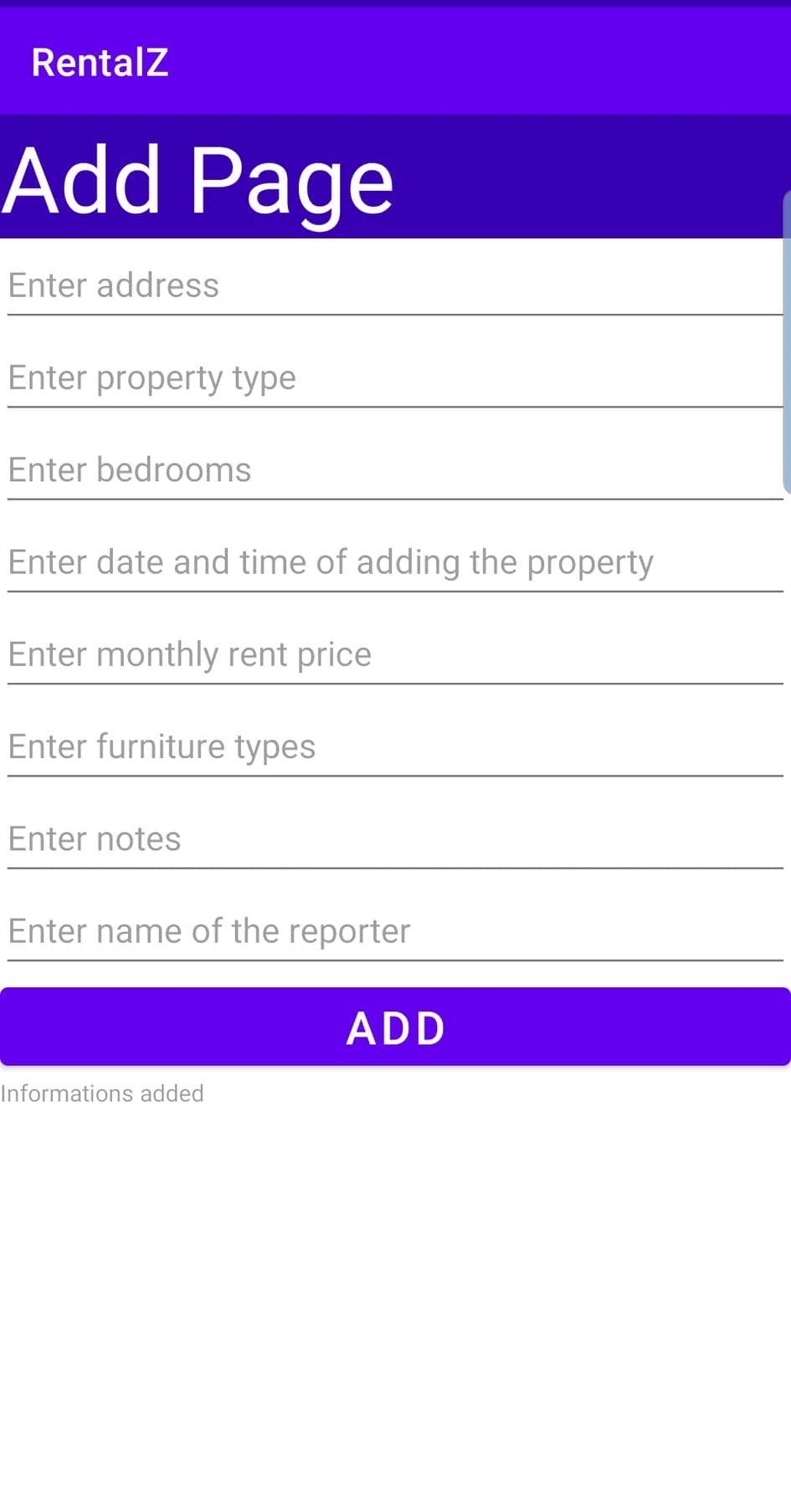


Figure 25: Android app data entry screen

When the user presses add but leaves the required fields blank, the user will see red dots at the end of the edit text of the required fields.

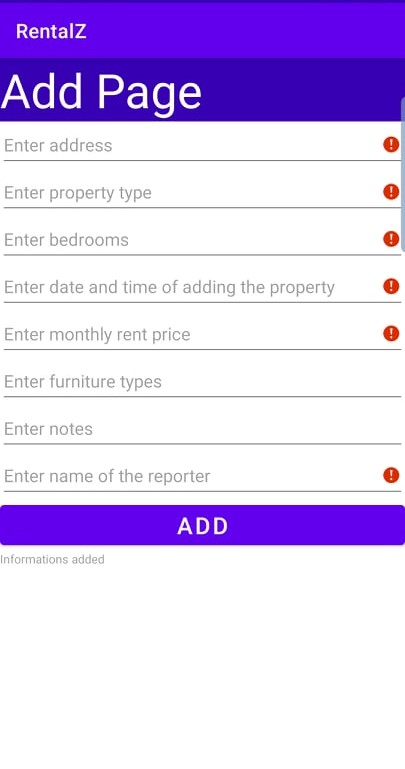


Figure 26: Android app data input screen when error message is reported

Notice that address is a required field. The notification will appear when the user clicks on the red dot the notification will appear.

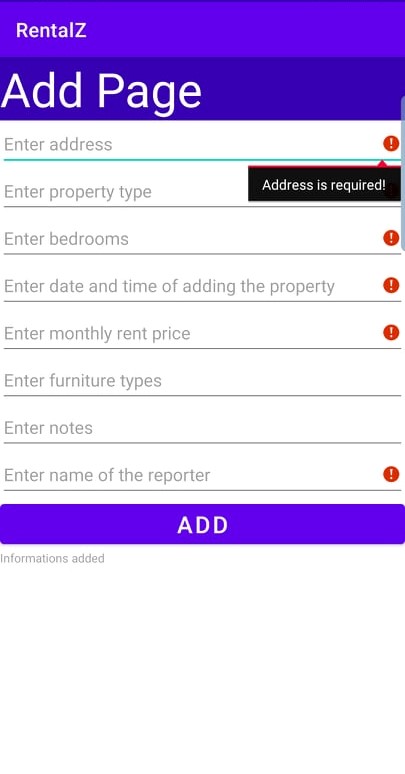


Figure 27: Notice that address is required field

Notice that property type is a required field. The notification will appear when the user clicks on the red dot the notification will appear.

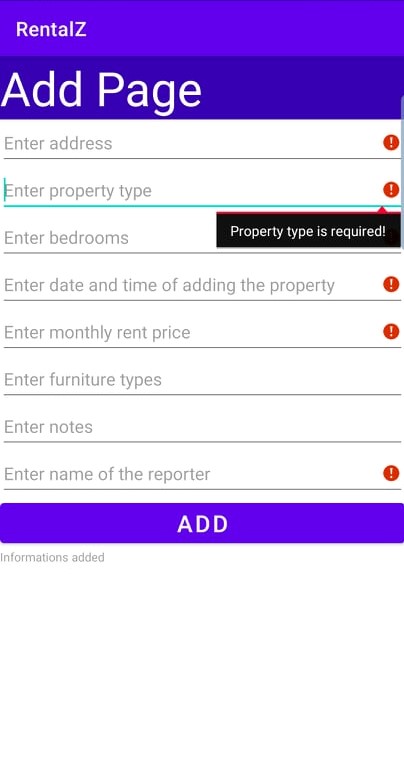


Figure 28: Notice that property type is required field

Notice that bedrooms is a required field. The notification will appear when the user clicks on the red dot the notification will appear.

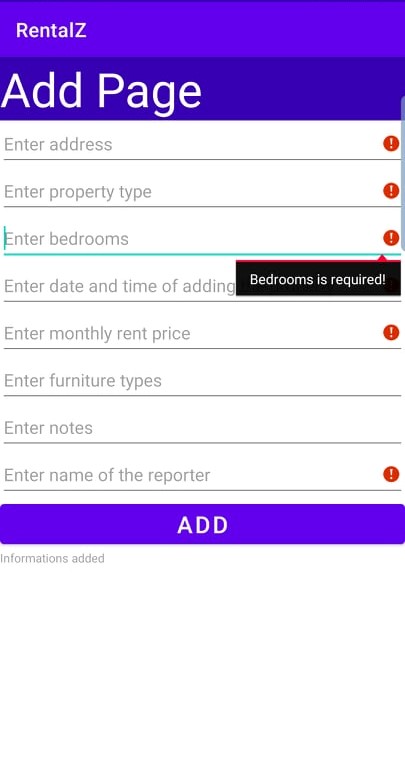


Figure 29: Notice that bedrooms is required field

Notice that price is a required field. The notification will appear when the user clicks on the red dot the notification will appear.

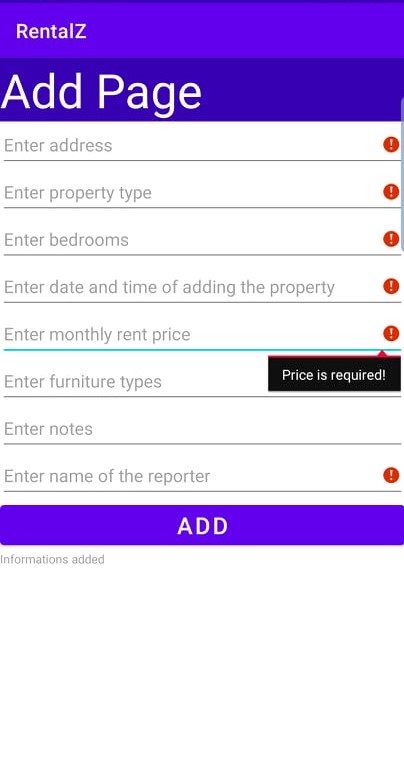


Figure 30: Notice that price is required field

Notice that name of reporter is a required field. The notification will appear when the user clicks on the red dot the notification will appear.

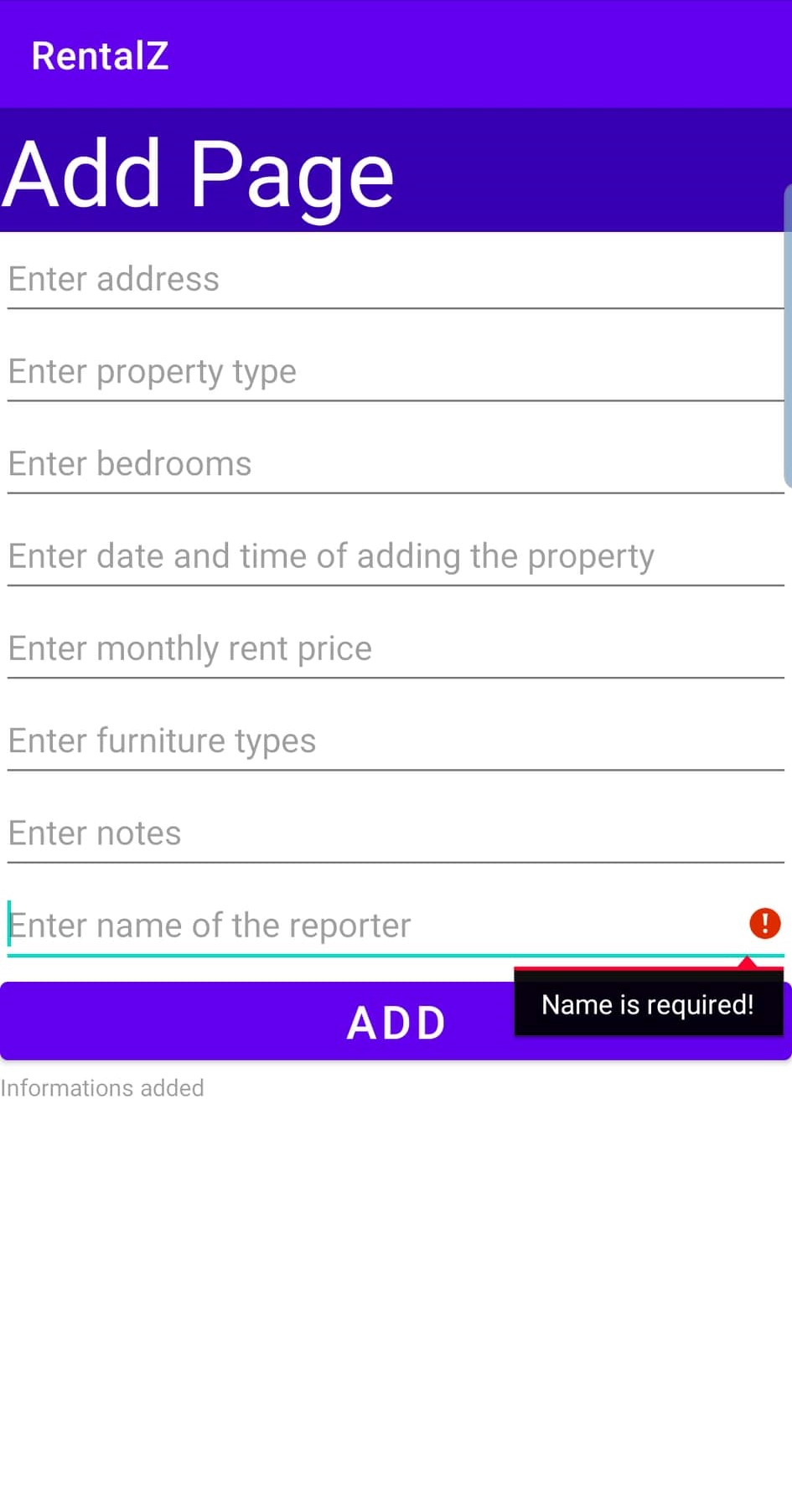


Figure 31: Notice that name of reporter is required field

Try entering all the fields and press add. The user can see the newly added information displayed below the add button with a text view.



Figure 32: Android app data added screen

### 2.2. Code

A scroll view is used so that the user can scroll up or down the data entry page if there are too many rows and the screen cannot display all of them based on the screen size of the machine. In design, scroll view has width equal to 409dp and height equal to 729dp, constraints of scroll view on the four sides are matched to the border of the full screen. In the scroll view contains a LinearLayout whose width and height coincide with the scroll view. The edit text in the linearlayout for the user to enter data, each edit text corresponds to a field. In addition, there is a button that is attached to the function of adding onClick events and a text view to display the newly added information.

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".MainActivity">  
  
 <ScrollView  
 android:layout\_width="409dp"  
 android:layout\_height="729dp"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 tools:ignore="SpeakableTextPresentCheck">  
  
 <LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:orientation="vertical" />  
 </ScrollView>  
  
 <LinearLayout  
 android:id="@+id/linearLayout"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:orientation="vertical"  
 app:layout\_constraintBottom\_toBottomOf="parent">  
  
 <TextView  
 android:id="@+id/textView"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:background="@color/design\_default\_color\_primary\_variant"  
 android:text="Add Page"  
 android:textColor="@color/white"  
 android:textSize="48sp" />  
  
 <EditText  
 android:id="@+id/txtAddress"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:ems="10"  
 android:hint="Enter address"  
 android:inputType="textPersonName"  
 android:minHeight="48dp" />  
  
 <EditText  
 android:id="@+id/txtPropertyType"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:ems="10"  
 android:hint="Enter property type "  
 android:inputType="textPersonName"  
 android:minHeight="48dp" />  
  
 <EditText  
 android:id="@+id/txtBedrooms"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:ems="10"  
 android:hint="Enter bedrooms"  
 android:inputType="textPersonName"  
 android:minHeight="48dp" />  
  
 <EditText  
 android:id="@+id/txtTime"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:ems="10"  
 android:hint="Enter date and time of adding the property "  
 android:inputType="textPersonName"  
 android:minHeight="48dp" />  
  
 <EditText  
 android:id="@+id/txtPrice"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:ems="10"  
 android:hint="Enter monthly rent price "  
 android:inputType="textPersonName"  
 android:minHeight="48dp" />  
  
 <EditText  
 android:id="@+id/txtFurnitureTypes"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:ems="10"  
 android:hint="Enter furniture types "  
 android:inputType="textPersonName"  
 android:minHeight="48dp" />  
  
 <EditText  
 android:id="@+id/txtNotes"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:ems="10"  
 android:hint="Enter notes"  
 android:inputType="textPersonName"  
 android:minHeight="48dp" />  
  
 <EditText  
 android:id="@+id/txtName"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:ems="10"  
 android:hint="Enter name of the reporter "  
 android:inputType="textPersonName"  
 android:minHeight="48dp" />  
  
 <Button  
 android:id="@+id/button"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:onClick="okClickHandler"  
 android:text="ADD"  
 android:textSize="24sp" />  
  
 <TextView  
 android:id="@+id/textView3"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:hint="Informations added"  
 android:textSize="12sp" />  
  
 </LinearLayout>  
</androidx.constraintlayout.widget.ConstraintLayout>

The code on the main java page shows how the add function works. The data entered into the editText is retrieved by assigning them specifically to the IDs of pre-set editTexts, for example, txtAddress is the id of the editText for the user to enter the address field, textPrice is the id of the editText. editText for the user to enter the price field. Input data is checked by IF statements, if length() that is the length of an input is zero, the application will report an error to the user when the user presses add. In case all required fields are satisfied, the entered data will be displayed in textView with ID textView3.

package fpt.schools.fgw;  
  
import androidx.appcompat.app.AppCompatActivity;  
  
import android.os.Bundle;  
  
import android.os.Bundle;  
import android.view.View;  
import android.widget.EditText;  
import android.widget.TextView;  
  
public class MainActivity extends AppCompatActivity {  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
 }  
  
 public void okClickHandler(View view) {  
  
 EditText txtAddress = findViewById(R.id.*txtAddress*);  
 EditText txtPropertyType = findViewById(R.id.*txtPropertyType*);  
 EditText txtBedrooms = findViewById(R.id.*txtBedrooms*);  
 EditText txtTime = findViewById(R.id.*txtTime*);  
 EditText txtPrice = findViewById(R.id.*txtPrice*);  
 EditText txtFurnitureTypes = findViewById(R.id.*txtFurnitureTypes*);  
 EditText txtNotes = findViewById(R.id.*txtNotes*);  
 EditText txtName = findViewById(R.id.*txtName*);  
  
 String address = txtAddress.getText().toString();  
 String propertyType = txtPropertyType.getText().toString();  
 String bedrooms = txtBedrooms.getText().toString();  
 String time = txtTime.getText().toString();  
 String price = txtPrice.getText().toString();  
 String furnitureTypes = txtFurnitureTypes.getText().toString();  
 String notes = txtNotes.getText().toString();  
 String name = txtName.getText().toString();  
  
 TextView myLabel = findViewById(R.id.*textView3*);  
 boolean isError = false;  
 if (address.length() == 0) {  
 txtAddress.setError("Address is required!");  
 isError = true;  
 }  
 if (propertyType.length() == 0) {  
 txtPropertyType.setError("Property type is required!");  
 isError = true;  
 }  
 if (bedrooms.length() == 0) {  
 txtBedrooms.setError("Bedrooms is required!");  
 isError = true;  
 }  
 if (time.length() == 0) {  
 txtTime.setError("Time is required!");  
 isError = true;  
 }  
 if (price.length() == 0) {  
 txtPrice.setError("Price is required!");  
 isError = true;  
 }  
 if (name.length() == 0) {  
 txtName.setError("Name is required!");  
 isError = true;  
 }  
 if (!isError) {  
 myLabel.setText("Address: " + address +  
 "Property type: " + propertyType +  
 "Bedrooms: " + bedrooms +  
 "Date and time of adding the property: " + time +  
 "Monthly rent price: " + price +  
 "Furniture ypes: " + furnitureTypes +  
 "Name of reporter: " + name +  
 "Notes: " + notes);  
 }  
 }  
}

# Link Google drive to download source code

Link to download ionic project:

<https://drive.google.com/file/d/1WUvzZvj8Z2aIBcBrJMsNU1UfZ3euu8Y_/view?usp=sharing>

Link to download android project:

https://drive.google.com/file/d/1WUvzZvj8Z2aIBcBrJMsNU1UfZ3euu8Y\_/view?usp=sharing