

MOBILE APPLICATION DESIGN AND DEVELOPMENT

Student’s name: Phung Ngoc Tan

Accessor’s name:Ho Nguyen Phu Bao

Student ID : GCS190619

Banner ID:

Class: TCS2203

Subject code: Comp1786

Contents

[**I/ Introduction** 3](#_Toc120043588)

[**II/ Implementation** 3](#_Toc120043589)

[**III/ Reflection** 4](#_Toc120043590)

[**III/ Evaluation** 5](#_Toc120043591)

[**Human Computing Interaction** 5](#_Toc120043592)

[**Security** 6](#_Toc120043593)

[**Screen Adjusting** 6](#_Toc120043594)

[**Changes** 7](#_Toc120043595)

[**IV/ Screenshot and demonstration** 7](#_Toc120043596)

[**Native app** 7](#_Toc120043597)

[**2. Cordova** 28](#_Toc120043598)

# **I/ Introduction**

There are a various amount of employee that works in London but may have to travel to Plymouth for meeting. They need a clear application where shows expense of travel for their job. This application will cover transport cost, food and other accommodations.

# **II/ Implementation**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Feature | Fully Implemented | Create but data is not being stored | Implemented but the app throws an exception | I have created a Cordova prototype of feature | Were not implemented in the Cordova prototype |
| Enter details of trips | ✓ |  |  | ✓ |  |
| View and delete trip details | ✓ |  |  | ✓ |  |
| Store trip details | ✓ |  |  | ✓ |  |
| Reset database | ✓ |  |  | ✓ |  |
| Add expense | ✓ |  |  | ✓ |  |
| Search | ✓ |  |  | ✓ |  |
| Upload details to a cloud-based web service | ✓ |  |  | ✓ |  |

Three of the additional feature that was implemented to Cordova and was run successfully

+ Employee can select which vehicle ( specifically car) they would like to travel with

+ Employee can ask for an adviser during their job, this is use fo if there is any accommodation happens

+ Employee can write notice, this ensure they had a good work-trip

# **III/ Reflection**

The application was being developed when I take the Mobile Application Design and Development course. The application was design using the Android Studio platform at first. Android Studio provides environment that I can build applications or Android, combining with the Java code, I was able to create the Native application on it , which some of the requirements. There also an issue where the application must be implemented to Cordova. I used Visual Studio Code as an environment for it, using HTML,CSS and JS as the backbone. In order to design this application, I was divided the process into four phases each phase focus on a specific task.This project is a 3-month project

For the first 15 days, it is a researching, I would collect data from this phase through user, I would like to know what they want, what they need. I would like to understand their behaviors too. This process is done through I hanged out survey. The second phase and third phase is around I started implemented all the functions that need to be done, also changing the layout for the application. More over I will implement the the Cordova in the third phase which most the same function of the Nativa application. The fourth phase will be hanged out to users and getting feedback in order to improve

Throughout this project, I was able to have an understanding on how an application was built. I am also getting the idea on how Java and Android Studio were contributed to the Design and Development. I am also understanding the interface and the definition of Cordova. Because the application was built around Java,HTML,CSS,JS, this also give me a change to review all of these code as well as looking for more complex functions to meet the requirements. Throughout this project, Im also faced many hardship. I am not that experience in mobile application design and development so it took a lot of time for me to have a basic understanding, the requirements for the application this makes the project slows down. After this project, in order to improve, I view more android application, also start to code more, this takes steps by steps for my speed and my understanding to be improved.

# **III/ Evaluation**

The application that I was produced is followed by some following statement which are human computing interaction, security, the ability to adjust base on screen-size. The application also need changes that would make for live user

## **Human Computing Interaction**

The basic definition of human computing interaction is how well the interaction between human and computer/application throughout the designing.My application is strictly follow the standard of human computing interaction. First into discuss is how I display icon though the interface. I selected the familiar icons for each function, eg the house for home function. I also highlight the main icon ( subsequent for the main functions) at the position that employee can easily view it. The logo also shows in the center of the application, which will give employee a brief definition on what the app is doing, short lines of explanation is also implemented.I highly put the reset database and back up function under the logo, this highlighted the two functions that employee may forgot when experience the application while these two also helpful at sometime.I tried to make the function as simple as possible, so employee doesn’t require to read a guidebook in order to use the application. Each task after done their function also place in well-organize following with the other, the font also similar every time, the size for the title is bigger than the text plus all the icon at footer is the same for every page, this will avoid misunderstanding.

## **Security**

Due to the time-limit when I implemented this project. I was not able to go deeply into how security will be implemented to the application. Despite not having the security implemented to the system, I was able to sketch out some idea that would make the application more secure. Login also a priority, basically I will implemented a login function where ID and password are required in order to login into the system, also the database need to be secure so not everyone can access to it. In order to enhance employee’s understanding on security, I am also aiming for create small guidebook to guide them on how back-up data is important-backing up data can prevent malware, also giving them contact to the security service when they had any problems connects to security

## **Screen Adjusting**

Android runs on a mass diversity of devices with different size and different pixel. The system will adjust through scaling and re-sizing to adapt the interface that been used by the employee This will enhance the UI on each type of screen.

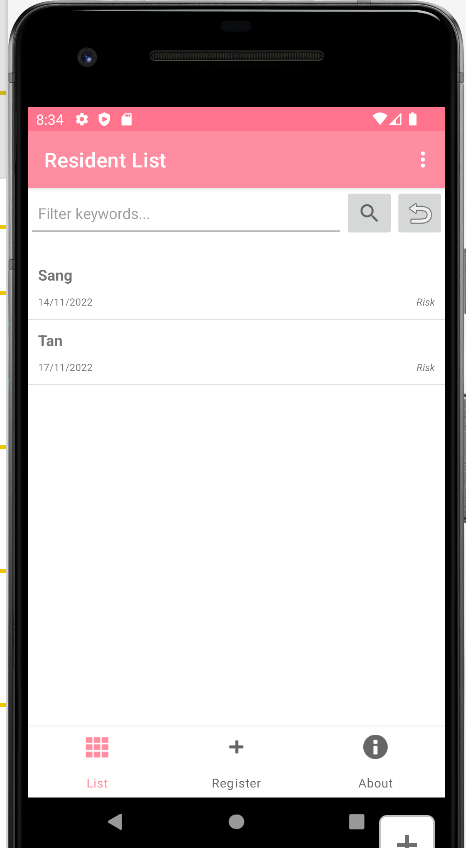
The screen adjusting covers alternative layouts,stretchable images,density independence,vector graphic. During the development of the application, due to the time limit, I was not put too much time on screen adjusting but I did look through it during the process. I tried to put the layout in separate container and adjust them to the point it can fit any interface. The problem start with the application logo, when it is too big that it can not fit big screen, but I was able to make it fit to a smaller screen. For other functions from add expense, add trip, view details; the adjustment had been made so on different devices those main task can perform perfectly, also some popup bars also place in the same position despite the interface.Mostly all the tabs, the icons shall display the same on different interface.

## **Changes**

In order to apply for live use, the application still needs a lot of changes. Most notable that can be mentioned is the security of the application. Lacking security is a block for the application because no one would prefer to use a product that can easy get attacked from malware. The application also missing some crucial functions for example in order to edit expense, more search queries. Personally I still feel the application is lacking some interaction between employee and developers, a function should be create so the developers can immediately fix problem for the employee. Moreover, it would be wisely to enhance the human computing interaction for this application, the application only meet the basic standards of human computing interaction, I think I should focus the layout more and how can the task be done easier without too much of switching tabs ( currently one page can only one task). The screen adjusting also needed to be focused, currently the application faced a problem when the layout will break when a huge image is added. Before deployed for live use, it is also wise to send out test sample, betas to some experience developers or other users for feedback and improvements

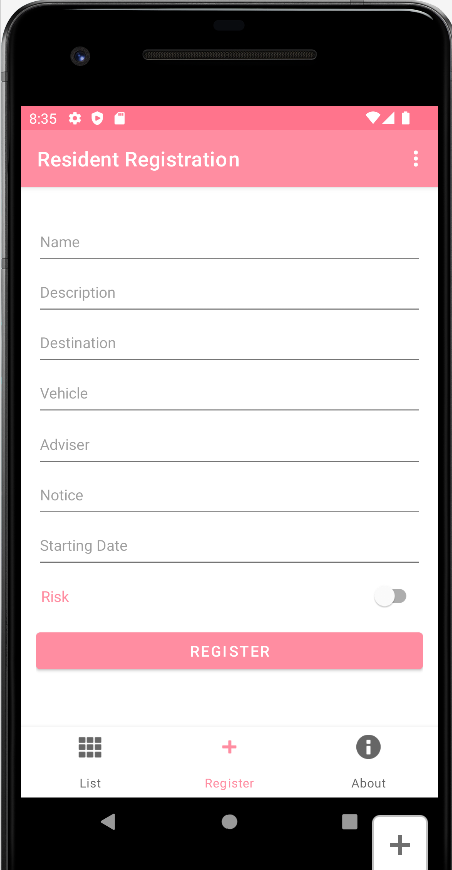
# **IV/ Screenshot and demonstration**

## **Native app**



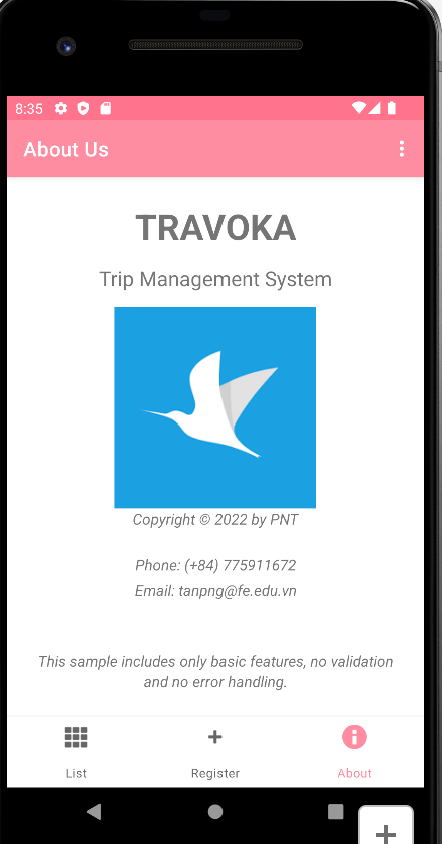
**Picture 1 Residential List**

The registered employee will be placed here and employee can enter it for adding request( showing in below0



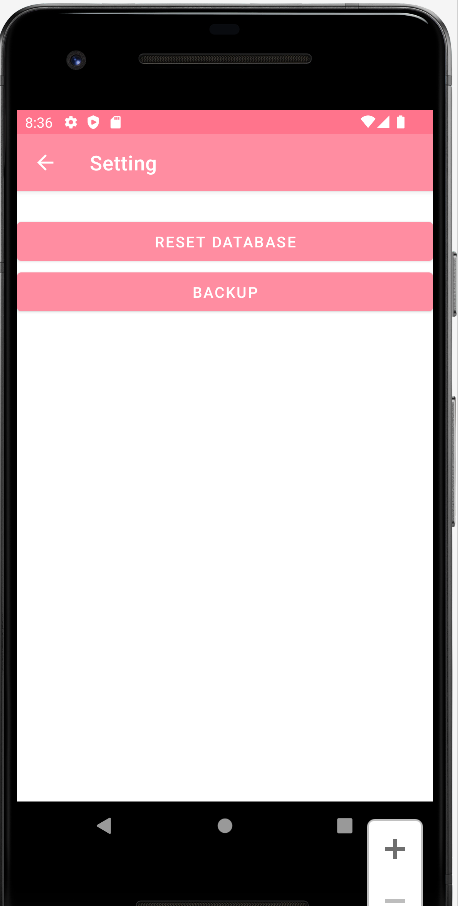
**Picture 2 Register function**

The page has 3 fields, by enter all those fields and click to the register button, the registered employee will be shown in page 1

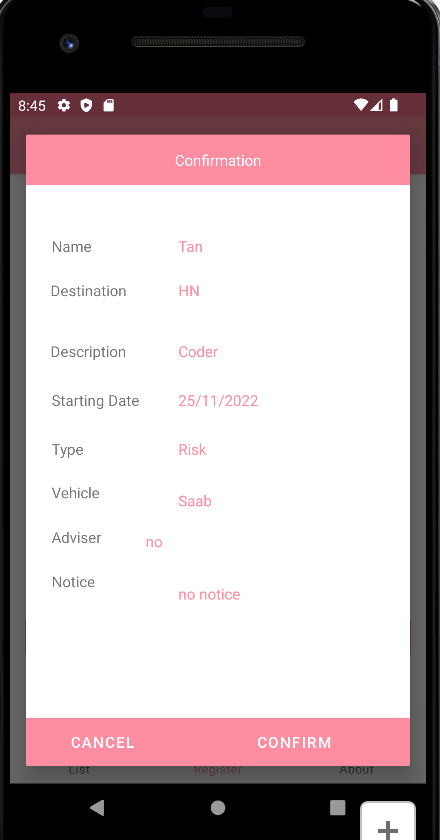


**Picture 3: About page-show details information about the application**

This page mainly focus on writing the credits for the developers

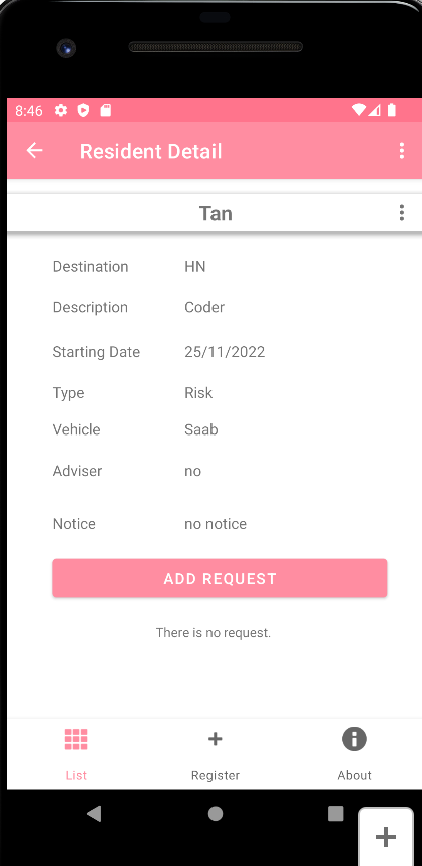


**Picture 4 Setting page where reset database and backup are available**



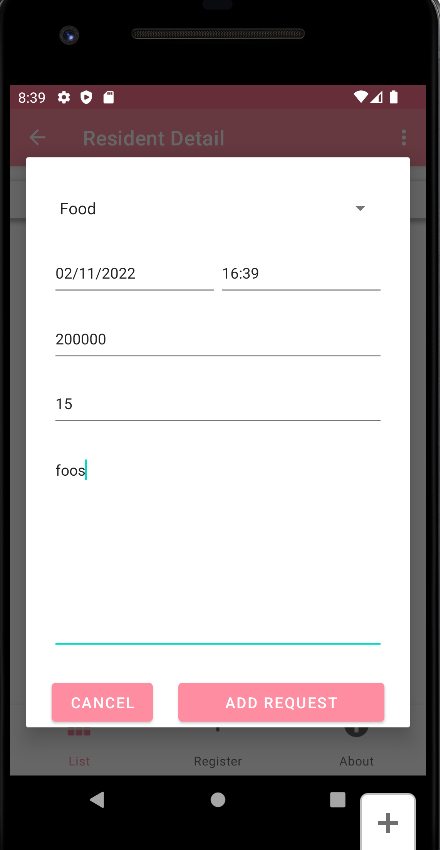
**Picture 5: Confirmation for register**

After enter all the field, this pop up will be shown up, by clicking confirm , all the information will pass to page 1



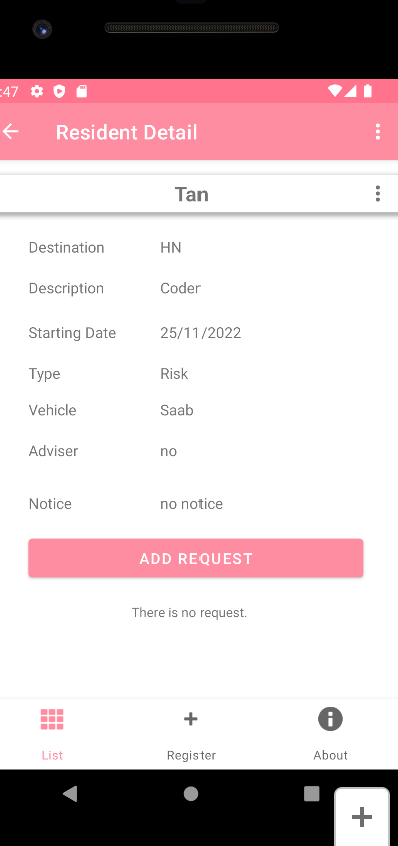
**Picture 6: Employee Details**

After the information pass to page 1, by clicking into it, it will bring to an extra page that show all the information and a tab for adding request



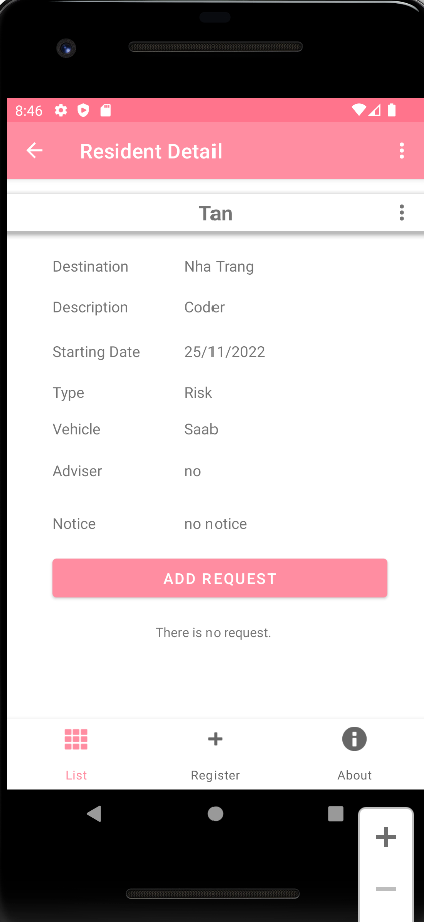
**Picture 7: Adding request**

Enter the available field for adding a request, after click the add request button the information will be show up to the page

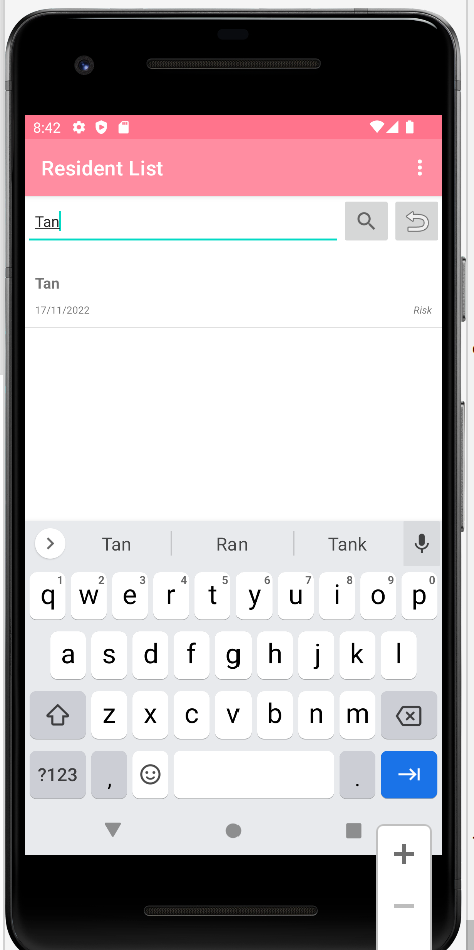
****

**Picture 8: By clicking update it will update**

**By clicking the delete button, the employee information will be removed**

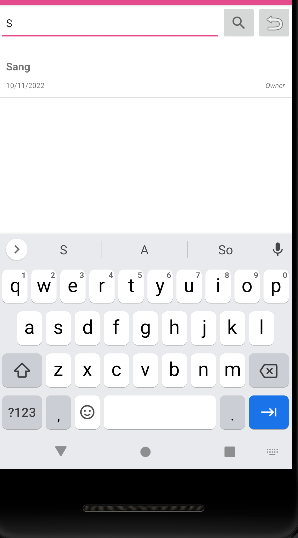
****

**Tan’s trip is updated**



**Picture 9: Searching function**

If there are too many employees are available, searching function is required so they can find their request more easily, here is the test where there are four given employee



**Picture 10: Searching fuction**

Only Sang is shown up because only the S was typed into the taskbar

**Generated code**

package vn.edu.greenwich.cw\_1\_sample.database;  
  
public class RequestEntry {  
 public static final String *TABLE\_NAME* = "request";  
 public static final String *COL\_ID* = "id";  
 public static final String *COL\_CONTENT* = "content";  
 public static final String *COL\_DATE* = "date";  
 public static final String *COL\_TIME* = "time";  
 public static final String *COL\_TYPE* = "type";  
 public static final String *COL\_PRICE* = "price";  
 public static final String *COL\_AMOUNT* = "amount";  
 public static final String *COL\_RESIDENT\_ID* = "resident\_id";  
  
 public static final String *SQL\_CREATE\_TABLE* =  
 "CREATE TABLE IF NOT EXISTS " + *TABLE\_NAME* + " (" +  
 *COL\_ID* + " INTEGER PRIMARY KEY," +  
 *COL\_CONTENT* + " TEXT NOT NULL," +  
 *COL\_DATE* + " TEXT NOT NULL," +  
 *COL\_TIME* + " TEXT NOT NULL," +  
 *COL\_TYPE* + " INTEGER NOT NULL," +  
 *COL\_PRICE* + " INTEGER NOT NULL," +  
 *COL\_AMOUNT* + " INTEGER NOT NULL," +  
 *COL\_RESIDENT\_ID* + " INTEGER NOT NULL," +  
 "FOREIGN KEY(" + *COL\_RESIDENT\_ID* + ") " +  
 "REFERENCES " + ResidentEntry.*TABLE\_NAME* + "(" + ResidentEntry.*COL\_ID* + "))";  
  
 public static final String *SQL\_DELETE\_TABLE* = "DROP TABLE IF EXISTS " + *TABLE\_NAME*;  
}

**First we had to generate the request entry which is all the column is required for the register page**

package vn.edu.greenwich.cw\_1\_sample.models;  
  
import java.io.Serializable;  
  
public class Resident implements Serializable {  
 protected long \_id;  
 protected String \_name;  
 protected String \_startDate;  
 protected int \_owner;  
 protected String \_destination;  
 protected String \_description;  
 protected String \_vehicle;  
 protected String \_adviser;  
 protected String \_notice;  
  
 public Resident() {  
 \_id = -1;  
 \_name = null;  
 \_startDate = null;  
 \_owner = -1;  
 \_destination = null;  
 \_description = null;  
 \_vehicle = null;  
 \_adviser = null;  
 \_notice = null;  
 }  
  
 public Resident(long id, String name,String destination, String description, String vehicle, String adviser, String notice, String startDate, int owner) {  
 \_id = id;  
 \_name = name;  
 \_startDate = startDate;  
 \_owner = owner;  
 \_description = description;  
 \_destination = destination;  
 \_vehicle = vehicle;  
 \_adviser = adviser;  
 \_notice = notice;  
 }  
  
 public long getId() { return \_id; }  
 public void setId(long id) {  
 \_id = id;  
 }  
  
 public String getName() {  
 return \_name;  
 }  
 public void setName(String name) {  
 \_name = name;  
 }  
  
 public String getStartDate() {  
 return \_startDate;  
 }  
 public void setStartDate(String startDate) {  
 \_startDate = startDate;  
 }  
  
 public int getOwner() {  
 return \_owner;  
 }  
 public void setOwner(int owner) {  
 \_owner = owner;  
 }  
  
 public String getDestination() {  
 return \_destination;  
 }  
 public void setDestination(String destination) {  
 \_destination = destination;  
 }  
  
 public String getDescription() {  
 return \_description;  
 }  
 public void setDescrtiption(String description) {  
 \_description = description;  
 }  
  
 public String getVehicle() {  
 return \_vehicle;  
 }  
 public void setVehicle(String vehicle) {  
 \_vehicle = vehicle;  
 }  
  
 public String getAdviser() {  
 return \_adviser;  
 }  
 public void setAdviser(String adviser) {  
 \_adviser = adviser;  
 }  
  
 public String getNotice() {  
 return \_notice;  
 }  
 public void setNotice(String notice) {  
 \_notice = notice;  
 }  
  
 public boolean isEmpty() {  
 if (-1 == \_id && null == \_name && null == \_startDate && -1 == \_owner && null == \_description && null == \_destination && null == \_vehicle && null == \_adviser&& null == \_notice)  
 return true;  
  
 return false;  
 }  
  
 @Override  
 public String toString() {  
 return "[" + \_startDate + "] " + \_name;  
 }  
}

**I set all the variables for the field that will enter to the request list**

public class Request implements Serializable {  
 protected long \_id;  
 protected String \_content;  
 protected String \_date;  
 protected String \_time;  
 protected String \_type;  
 protected long \_residentId;  
 protected int \_price;  
 protected int \_amount;  
  
 public Request() {  
 \_id = -1;  
 \_content = null;  
 \_date = null;  
 \_time = null;  
 \_type = null;  
 \_residentId = -1;  
 \_price = 0;  
 \_amount = 0;  
 }  
  
 public Request(long id, String content, String date, String time, String type, long residentId,int price,int amount) {  
 \_id = id;  
 \_content = content;  
 \_date = date;  
 \_time = time;  
 \_type = type;  
 \_residentId = residentId;  
 \_price = price;  
 \_amount = amount;  
 }  
  
 public long getId() {  
 return \_id;  
 }  
  
 public void setId(long id) {  
 \_id = id;  
 }  
  
 public String getContent() {  
 return \_content;  
 }  
  
 public void setContent(String content) {  
 \_content = content;  
 }  
  
 public String getDate() {  
 return \_date;  
 }  
  
 public void setDate(String date) {  
 \_date = date;  
 }  
  
 public String getTime() {  
 return \_time;  
 }  
  
 public void setTime(String time) {  
 \_time = time;  
 }  
  
 public String getType() {  
 return \_type;  
 }  
  
 public void setType(String type) {  
 \_type = type;  
 }  
  
 public long getResidentId() {  
 return \_residentId;  
 }  
  
 public void setResidentId(long residentId) {  
 \_residentId = residentId;  
 }  
  
 public int getPrice() {  
 return \_price;  
 }  
  
 public void setPrice(int \_price) {  
 this.\_price = \_price;  
 }  
  
 public int getAmount() {  
 return \_amount;  
 }  
  
 public void setAmount(int \_amount) {  
 this.\_amount = \_amount;  
 }  
  
 public boolean isEmpty() {  
 if (-1 == \_id && null == \_content && null == \_date && null == \_time && null == \_type && -1 == \_residentId && 0 == \_price && 0 == \_amount)  
 return true;  
  
 return false;  
 }  
  
 @Override  
 public String toString() {  
 return "[" + \_type + "][" + \_date + "] " + \_content;  
 }  
}

**For the same in here I set all the variables for the field that will enter to the request list**

public View onCreateView(LayoutInflater inflater, ViewGroup container,  
 Bundle savedInstanceState) {  
 View view = inflater.inflate(R.layout.*fragment\_resident\_register\_confirm*, container, false);  
  
 String name = getString(R.string.*error\_no\_info*);  
 String description = getString(R.string.*error\_no\_info*);  
 String destination = getString(R.string.*error\_no\_info*);  
 String vehicle = getString(R.string.*error\_no\_info*);  
 String adviser = getString(R.string.*error\_no\_info*);  
 String notice = getString(R.string.*error\_no\_info*);  
 String startDate = getString(R.string.*error\_no\_info*);  
 String ownerType = getString(R.string.*error\_no\_info*);  
  
 fmResidentRegisterConfirmName = view.findViewById(R.id.*fmResidentRegisterConfirmName*);  
 fmResidentRegisterConfirmStartDate = view.findViewById(R.id.*fmResidentRegisterConfirmStartDate*);  
 fmResidentRegisterConfirmOwner = view.findViewById(R.id.*fmResidentRegisterConfirmOwner*);  
 fmResidentRegisterConfirmDescription = view.findViewById(R.id.*fmResidentRegisterConfirmDescription*);  
 fmResidentRegisterConfirmDestination = view.findViewById(R.id.*fmResidentRegisterConfirmDestination*);  
 fmResidentRegisterConfirmVehicle = view.findViewById(R.id.*fmResidentRegisterConfirmVehicle*);  
 fmResidentRegisterConfirmAdviser = view.findViewById(R.id.*fmResidentRegisterConfirmAdviser*);  
 fmResidentRegisterConfirmNotice = view.findViewById(R.id.*fmResidentRegisterConfirmNotice*);  
 fmResidentRegisterConfirmButtonCancel = view.findViewById(R.id.*fmResidentRegisterConfirmButtonCancel*);  
 fmResidentRegisterConfirmButtonConfirm = view.findViewById(R.id.*fmResidentRegisterConfirmButtonConfirm*);  
  
 if (\_resident.getOwner() != -1) {  
 ownerType = \_resident.getOwner() == 1 ? getString(R.string.*label\_owner*) : getString(R.string.*label\_tenant*);  
 }  
  
 if (\_resident.getName() != null && !\_resident.getName().trim().isEmpty()) {  
 name = \_resident.getName();  
 }  
  
 if (\_resident.getStartDate() != null && !\_resident.getStartDate().trim().isEmpty()) {  
 startDate = \_resident.getStartDate();  
 }  
  
 if (\_resident.getDescription() != null && !\_resident.getDescription().trim().isEmpty()) {  
 description = \_resident.getDescription();  
 }  
  
 if (\_resident.getDestination() != null && !\_resident.getDestination().trim().isEmpty()) {  
 destination = \_resident.getDestination();  
 }  
 if (\_resident.getVehicle() != null && !\_resident.getVehicle().trim().isEmpty()) {  
 vehicle = \_resident.getVehicle();  
 }  
 if (\_resident.getAdviser() != null && !\_resident.getAdviser().trim().isEmpty()) {  
 adviser = \_resident.getAdviser();  
 }  
 if (\_resident.getNotice() != null && !\_resident.getNotice().trim().isEmpty()) {  
 notice = \_resident.getNotice();  
 }  
  
  
 fmResidentRegisterConfirmName.setText(name);  
 fmResidentRegisterConfirmStartDate.setText(startDate);  
 fmResidentRegisterConfirmOwner.setText(ownerType);  
 fmResidentRegisterConfirmDestination.setText(destination);  
 fmResidentRegisterConfirmDescription.setText(description);  
 fmResidentRegisterConfirmVehicle.setText(vehicle);  
 fmResidentRegisterConfirmAdviser.setText(adviser);  
 fmResidentRegisterConfirmNotice.setText(notice);  
  
 fmResidentRegisterConfirmButtonCancel.setOnClickListener(v -> dismiss());  
 fmResidentRegisterConfirmButtonConfirm.setOnClickListener(v -> confirm());  
  
 return view;  
 }  
  
 protected void confirm() {  
 long status = \_db.insertResident(\_resident);  
  
 FragmentListener listener = (FragmentListener) getParentFragment();  
 listener.sendFromResidentRegisterConfirmFragment(status);  
  
 dismiss();  
 }  
  
 public interface FragmentListener {  
 void sendFromResidentRegisterConfirmFragment(long status);  
 }  
}

protected Resident getResidentFromInput(long id) {  
 String name = fmResidentRegisterName.getText().toString();  
 String startDate = fmResidentRegisterStartDate.getText().toString();  
 int owner = fmResidentRegisterOwner.isChecked() ? 1 : 0;  
 String destination = fmResidentRegisterDestination.getText().toString();  
 String description = fmResidentRegisterDescription.getText().toString();  
 String vehicle = fmResidentRegisterVehicle.getText().toString();  
 String adviser = fmResidentRegisterAdviser.getText().toString();  
 String notice = fmResidentRegisterNotice.getText().toString();  
  
 return new Resident(id, name,destination,description, vehicle, adviser, notice, startDate, owner);  
}

**This function shows how the create form got pop up and how the system will store data from it**

public long updateResident(Resident resident) {  
 ContentValues values = getResidentValues(resident);  
  
 String selection = ResidentEntry.*COL\_ID* + " = ?";  
 String[] selectionArgs = {String.*valueOf*(resident.getId())};  
  
 return dbWrite.update(ResidentEntry.*TABLE\_NAME*, values, selection, selectionArgs);  
}

**This function shows how to update the resident information take the Id from column**

public long deleteResident(long id) {  
 String selection = ResidentEntry.*COL\_ID* + " = ?";  
 String[] selectionArgs = {String.*valueOf*(id)};  
  
 return dbWrite.delete(ResidentEntry.*TABLE\_NAME*, selection, selectionArgs);  
}

**This function shows how to delete the resident information take the Id from column**

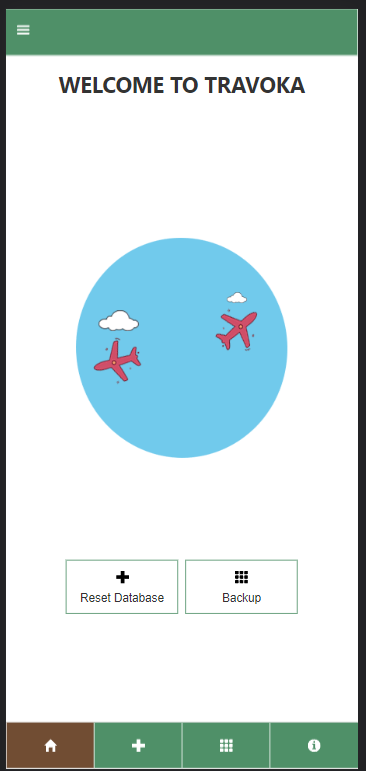
ublic ResidentSearchFragment() {}  
  
 @Override  
 public void onResume() {  
 super.onResume();  
  
 ViewGroup.LayoutParams params = getDialog().getWindow().getAttributes();  
 params.width = ViewGroup.LayoutParams.*MATCH\_PARENT*;  
 getDialog().getWindow().setAttributes((android.view.WindowManager.LayoutParams) params);  
 }  
  
 @Override  
 public View onCreateView(LayoutInflater inflater, ViewGroup container,  
 Bundle savedInstanceState) {  
 View view = inflater.inflate(R.layout.*fragment\_resident\_search*, container, false);  
  
 fmResidentSearchDate = view.findViewById(R.id.*fmResidentSearchDate*);  
 fmResidentSearchName = view.findViewById(R.id.*fmResidentSearchName*);  
 fmResidentSearchButtonCancel = view.findViewById(R.id.*fmResidentSearchButtonCancel*);  
 fmResidentSearchButtonSearch = view.findViewById(R.id.*fmResidentSearchButtonSearch*);  
  
 fmResidentSearchButtonSearch.setOnClickListener(v -> search());  
 fmResidentSearchButtonCancel.setOnClickListener(v -> dismiss());  
 fmResidentSearchDate.setOnTouchListener((v, motionEvent) -> showCalendar(motionEvent));  
  
 return view;  
 }  
  
 protected void search() {  
 Resident \_resident = new Resident();  
  
 String date = fmResidentSearchDate.getText().toString();  
 String name = fmResidentSearchName.getText().toString();  
  
 if (date != null && !date.trim().isEmpty())  
 \_resident.setStartDate(date);  
  
 if (name != null && !name.trim().isEmpty())  
 \_resident.setName(name);  
  
 FragmentListener listener = (FragmentListener) getParentFragment();  
 listener.sendFromResidentSearchFragment(\_resident);  
  
 dismiss();  
 }  
  
 protected boolean showCalendar(MotionEvent motionEvent) {  
 if (motionEvent.getAction() == MotionEvent.*ACTION\_DOWN*) {  
 new CalendarFragment().show(getChildFragmentManager(), null);  
 }  
  
 return false;  
 }  
  
 @Override  
 public void sendFromCalendarFragment(String date) {  
 fmResidentSearchDate.setText(date);  
 }  
  
 public interface FragmentListener {  
 void sendFromResidentSearchFragment(Resident resident);  
 }  
}

**This how the search function work, search based on the queries**

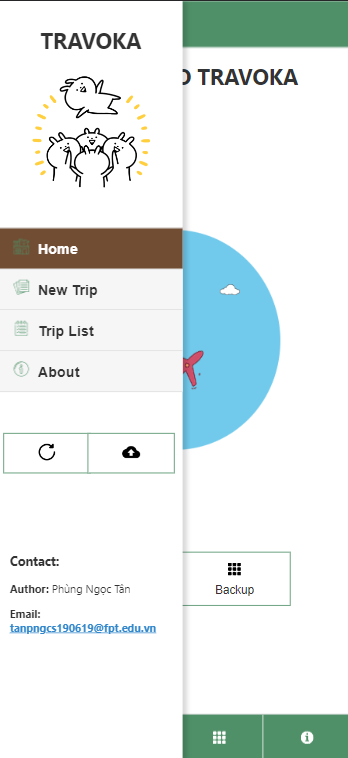
public ResidentUpdateFragment() {}  
  
 @Override  
 public View onCreateView(LayoutInflater inflater, ViewGroup container,  
 Bundle savedInstanceState) {  
 View view = inflater.inflate(R.layout.*fragment\_resident\_update*, container, false);  
  
 if (getArguments() != null) {  
 Resident resident = (Resident) getArguments().getSerializable(*ARG\_PARAM\_RESIDENT*);  
  
 Bundle bundle = new Bundle();  
 bundle.putSerializable(ResidentRegisterFragment.*ARG\_PARAM\_RESIDENT*, resident);  
  
 // Send arguments (resident info) to ResidentRegisterFragment.  
 getChildFragmentManager().getFragments().get(0).setArguments(bundle);  
 }  
  
 return view;  
 }  
  
 @Override  
 public void sendFromResidentRegisterFragment(long status) {  
 switch ((int) status) {  
 case 0:  
 Toast.*makeText*(getContext(), R.string.*notification\_update\_fail*, Toast.*LENGTH\_SHORT*).show();  
 return;  
  
 default:  
 Toast.*makeText*(getContext(), R.string.*notification\_update\_success*, Toast.*LENGTH\_SHORT*).show();  
 Navigation.*findNavController*(getView()).navigateUp();  
 }  
 }  
}

**This fuction shows how the function got update by obtain the new ID and pop up the message when it update success**

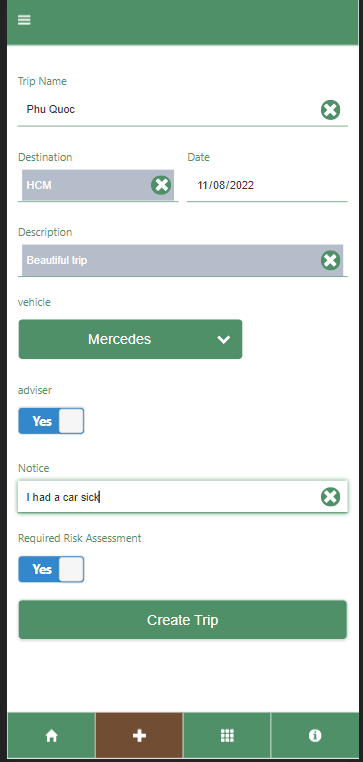
## **2. Cordova**



**Picture 11: Home Screen**

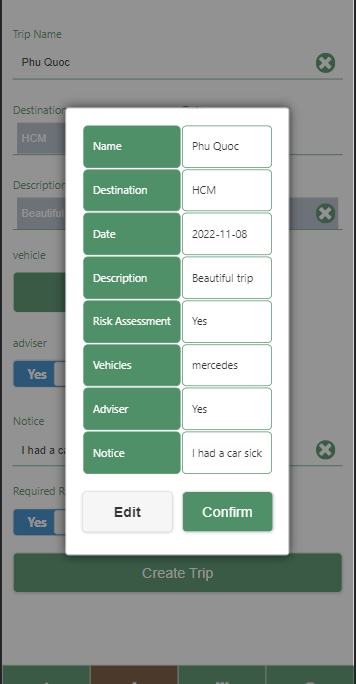


**Picture 12 : Tab bar design**



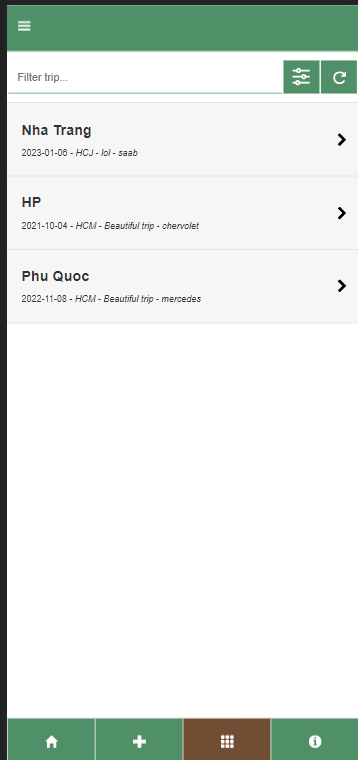
**Picture 13 : Trip Register**

Enter all the available field and all the information will pass to the trip list ( the 3rd tab)



**Picture 14 Confirmation**

The application will once again ask the user to confirm all their informtion. Is they input them all correctly

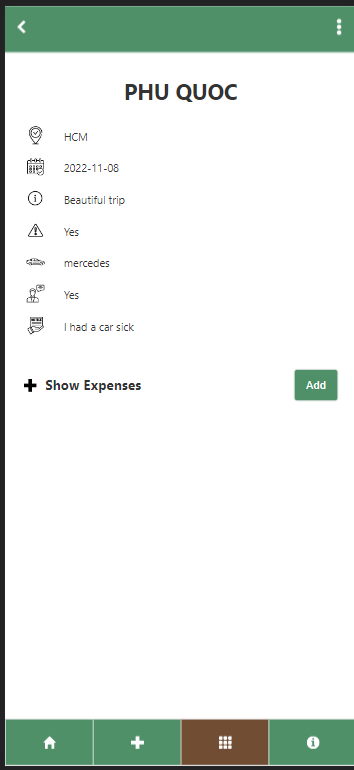


**Picture 15: Phu Quoc was added to the list**



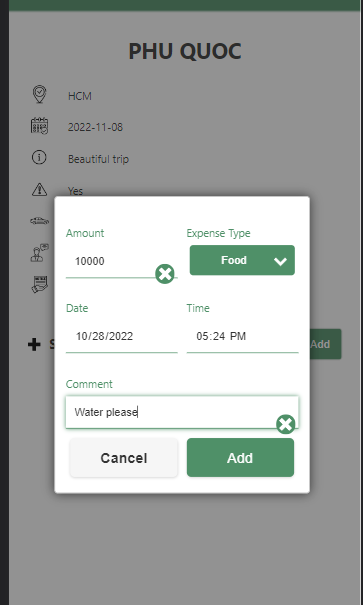
**Picture 16 Searching**

I also put a search query for this application, in the picture when typing “PH”, only Phu Quoc is shown

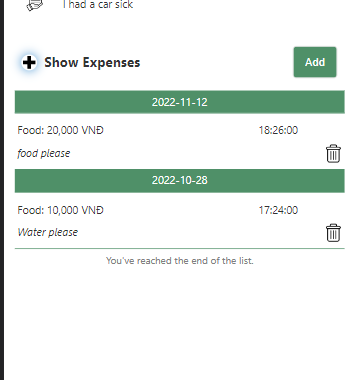


**Picture 17: Trip details**

All the information that was input in the form will get pass to here, in here there are also show expense function available, by clicking the word “ add” the tab below will show and click to the plus button to view all expenses

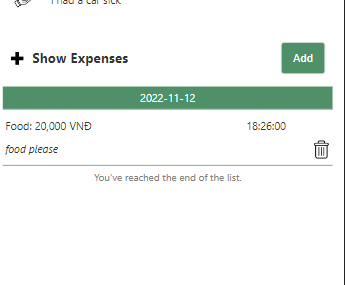


**Picture 18: Add Expense**

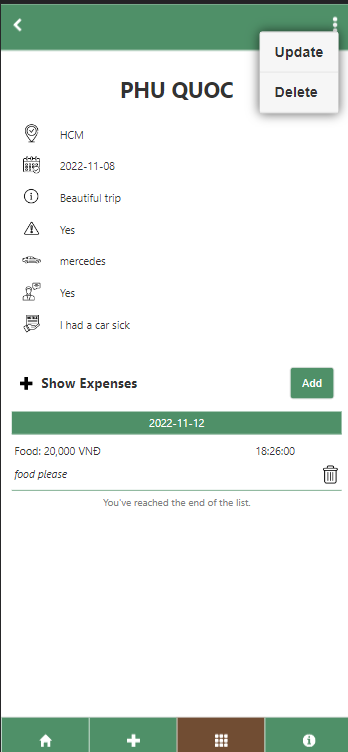


**Picture 19: Multiple added expenses**

There also a bin icon near the right-edge, by clicking to this icon, that expense will get remove

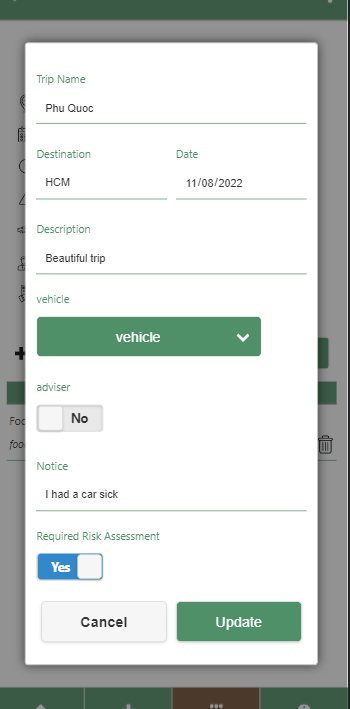


**Picture 20: Food 10000VND was removed**

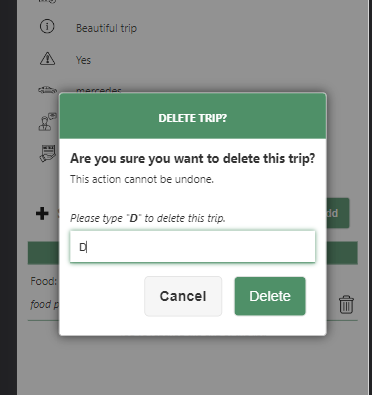


**Picture 21 Update function**

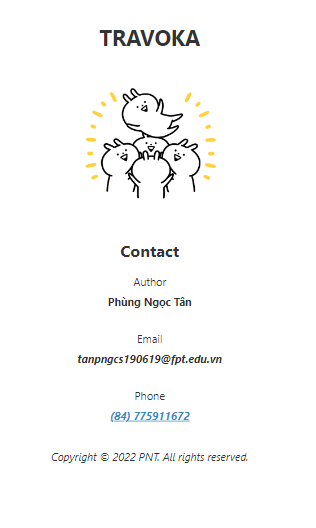
By clicking to the 3-dot button, employee can either update their form or delete it. The delete function only works when employee type the letter “D” in to the text box. This prevent unwillingly delete a form



**Picture 22: Update**

****

**Picture 23: Delete function**



**Picture 24: About Page**

**V/ Code demonstration**

1. **Native application**
2. **Cordova application**

function confirmTrip(e) {

    e.preventDefault();

    log('Open the confirmation popup.');

    $('#page-create #frm-confirm #name').text($('#page-create #frm-register #name').val());

    $('#page-create #frm-confirm #destination').text($('#page-create #frm-register #destination').val());

    $('#page-create #frm-confirm #date').text($('#page-create #frm-register #date').val());

    $('#page-create #frm-confirm #description').text($('#page-create #frm-register #description').val());

    $('#page-create #frm-confirm #risk').text($('#page-create #frm-register #risk').val());

    $('#page-create #frm-confirm #vehicle').text($('#page-create #frm-register #vehicle').val());

    $('#page-create #frm-confirm #adviser').text($('#page-create #frm-register #adviser').val());

    $('#page-create #frm-confirm #notice').text($('#page-create #frm-register #notice').val());

    $('#page-create #frm-confirm').popup('open');

}

**This function pops the confirmation box before confirm to make sure the employee want any editing or not**

function registerTrip(e) {

    e.preventDefault();

    let name = $('#page-create #frm-register #name').val();

    let destination = $('#page-create #frm-register #destination').val();

    let date = $('#page-create #frm-register #date').val();

    let description = $('#page-create #frm-register #description').val();

    let risk = $('#page-create #frm-register #risk').val();

    let vehicle = $('#page-create #frm-register #vehicle').val();

    let adviser = $('#page-create #frm-register #adviser').val();

    let notice= $('#page-create #frm-register #notice').val();

    db.transaction(function (tx) {

        let query = `INSERT INTO Trip (Name, Destination, Description, Risk, Vehicle, Adviser, Notice, Date) VALUES (?, ?, ?, ?, ?, ?, ?, julianday('${date}'))`;

        tx.executeSql(query, [name, destination, description, risk, vehicle, adviser, notice], transactionSuccess, transactionError);

        function transactionSuccess(tx, result) {

            let message = `Added trip '${name}'.`;

            log(message);

            toast(message);

            $('#page-create #frm-register').trigger('reset');

            $('#page-create #frm-register #name').focus();

            $('#page-create #frm-confirm').popup('close');

        }

    });

**With this function, employee will able to enter all the information to the register page to the trip list page. Each let statement is a variable with it ( name, description,date,etc)**

function showDetail() {

    let id = localStorage.getItem(currentTripId);

    db.transaction(function (tx) {

        let query = `SELECT \*, date(Date) AS DateConverted FROM Trip WHERE Id = ?`;

        tx.executeSql(query, [id], transactionSuccess, transactionError);

        function transactionSuccess(tx, result) {

            if (result.rows[0] != null) {

                log(`Get details of trip '${result.rows[0].name}' successfully.`);

                $('#page-detail #detail #name').text(result.rows[0].Name);

                $('#page-detail #detail #destination').text(result.rows[0].Destination);

                $('#page-detail #detail #description').text(result.rows[0].Description);

                $('#page-detail #detail #risk').text(result.rows[0].Risk);

                $('#page-detail #detail #date').text(result.rows[0].DateConverted);

                $('#page-detail #detail #vehicle').text(result.rows[0].Vehicle);

                $('#page-detail #detail #adviser').text(result.rows[0].Adviser);

                $('#page-detail #detail #notice').text(result.rows[0].Notice);

                showExpense();

            }

            else {

                let errorMessage = 'Trip not found.';

                log(errorMessage, ERROR);

                $('#page-detail #detail #name').text(errorMessage);

                $('#page-detail #btn-update').addClass('ui-disabled');

                $('#page-detail #btn-delete-confirm').addClass('ui-disabled');

            }

        }

    });

}

**This function shows the information on the trip list page, the function that show the trip details**

function deleteTrip(e) {

    e.preventDefault();

    let tripId = localStorage.getItem(currentTripId);

    db.transaction(function (tx) {

        let name = '';

        let query = 'SELECT \* FROM Trip WHERE Id = ?';

        tx.executeSql(query, [tripId], function (tx, result) {

            name = result.rows[0].Name;

        }, transactionError);

        query = 'DELETE FROM Expense WHERE TripId = ?';

        tx.executeSql(query, [tripId], function (tx, result) {

            log(`Delete expenses of trip '${tripId}' successfully.`);

        }, transactionError);

        query = 'DELETE FROM Trip WHERE Id = ?';

        tx.executeSql(query, [tripId], function (tx, result) {

            let message = `Deleted trip '${name}'.`;

            log(message);

            toast(message);

            $('#page-detail #frm-delete').trigger('reset');

            $.mobile.navigate('#page-list', { transition: 'none' });

        }, transactionError);

    });

}

unction confirmDeleteTrip() {

    let text = $('#page-detail #frm-delete #txt-confirm').val();

    if (text == 'D') {

        $('#page-detail #frm-delete #btn-delete').removeClass('ui-disabled');

    }

    else {

        $('#page-detail #frm-delete #btn-delete').addClass('ui-disabled');

    }

}

**The first function is executed when user want to delete trip from the trip list page**

**This second fuction is for deleting the trip, only trip be deleted when type “ D”**

function addExpense(e) {

    e.preventDefault();

    let tripId = localStorage.getItem(currentTripId);

    let type = $('#page-detail #frm-add-expense #type').val();

    let amount = parseInt($('#page-detail #frm-add-expense #amount').val());

    let date = $('#page-detail #frm-add-expense #date').val();

    let time = $('#page-detail #frm-add-expense #time').val();

    let comment = $('#page-detail #frm-add-expense #comment').val();

    db.transaction(function (tx) {

        let query = `INSERT INTO Expense (Type, Amount, Comment, TripId, Date, Time) VALUES (?, ?, ?, ?, julianday('${date}'), julianday('${time}'))`;

        tx.executeSql(query, [type, amount, comment, tripId], transactionSuccess, transactionError);

        function transactionSuccess(tx, result) {

            let message = `Added expense '${amount.toLocaleString("en-US")} VNĐ (${type})'.`;

            log(message);

            toast(message);

            $('#page-detail #frm-add-expense').trigger('reset');

            $('#page-detail #frm-add-expense').popup('close');

            showExpense();

        }

    });

}

**There an ability for user to add expense, this function will run when user want to use that. Once again each let statements go with one variable ( tripID,type,amount,etc)**

function showExpense() {

    let id = localStorage.getItem(currentTripId);

    db.transaction(function (tx) {

        let query = `SELECT \*, date(Date) AS DateConverted, time(Time) AS TimeConverted FROM Expense WHERE TripId = ? ORDER BY Date DESC, Time DESC`;

        tx.executeSql(query, [id], transactionSuccess, transactionError);

        function transactionSuccess(tx, result) {

            log(`Get list of expenses successfully.`);

            let expenseList = '';

            let currentDate = '';

            for (let expense of result.rows) {

                if (currentDate != expense.DateConverted) {

                    expenseList += `<div class='list-date'>${expense.DateConverted}</div>`;

                    currentDate = expense.DateConverted;

                }

                expenseList += `

                    <div class='list'>

                        <table style='white-space: nowrap; width: 100%;'>

                            <tr style='height: 25px;'>

                                <td>${expense.Type}: ${expense.Amount.toLocaleString('en-US')} VNĐ</td>

                                <td style='text-align: right;'>${expense.TimeConverted}</td>

                            </tr>

                            <tr>

                                <td colspan='2'><em>${expense.Comment}</em></td>

                                <td><a data-id="${expense.Id}" id="btn-delete-expense-popup">

                                <img style="width: 20px;height: 20px;float:right" src="img/delete.png" />

                                </a></td>

                            </tr>

                        </table>

                    </div>`;

            }

            expenseList += `<div class='list end-list'>You've reached the end of the list.</div>`;

            $('#page-detail #expense #list').empty().append(expenseList);

            log(`Show list of expenses successfully.`);

        }

    });

}

**This functions show how can the expense above will got show up to the interface**

function showUpdate() {

    let id = localStorage.getItem(currentTripId);

    db.transaction(function (tx) {

        let query = `SELECT \*, date(Date) as DateConverted FROM Trip WHERE Id = ?`;

        tx.executeSql(query, [id], transactionSuccess, transactionError);

        function transactionSuccess(tx, result) {

            if (result.rows[0] != null) {

                log(`Get details of trip '${result.rows[0].Name}' successfully.`);

                $(`#page-detail #frm-update #name`).val(result.rows[0].Name);

                $(`#page-detail #frm-update #destination`).val(result.rows[0].Destination);

                $(`#page-detail #frm-update #risk`).val(result.rows[0].Risk).slider("refresh");

                $(`#page-detail #frm-update #date`).val(result.rows[0].DateConverted);

                $(`#page-detail #frm-update #description`).val(result.rows[0].Description);

                $(`#page-detail #frm-update #vehicle`).val(result.rows[0].Vehicle);

                $(`#page-detail #frm-update #adviser`).val(result.rows[0].Adviser);

                $(`#page-detail #frm-update #notice`).val(result.rows[0].Notice);

                changePopup($('#page-detail #option'), $('#page-detail #frm-update'));

            }

        }

    });

}

**When user want to update their form, this function will be execute.**

function filterTrip() {

    let filter = $('#page-list #txt-filter').val().toLowerCase();

    let li = $('#page-list #list-trip ul li');

    for (let i = 0; i < li.length; i++) {

        let a = li[i].getElementsByTagName('a')[0];

        let text = a.textContent || a.innerText;

        li[i].style.display = text.toLowerCase().indexOf(filter) > -1 ? '' : 'none';

    }

}

**Filter trip code, user can filter their entered trip**

function search(e) {

    e.preventDefault();

    let name = $('#page-list #frm-search #name').val();

    let destination = $('#page-list #frm-search #destination').val();

    let date = $('#page-list #frm-search #date').val();

    db.transaction(function (tx) {

        let query = `SELECT \*, date(Date) as DateConverted FROM Trip WHERE`;

        query += name ? ` Name LIKE "%${name}%"   AND` : '';

        query += destination ? ` Destination LIKE "%${destination}%"   AND` : '';

        query += date ? ` Date = julianday('${date}')   AND` : '';

        query = query.substring(0, query.length - 6);

        tx.executeSql(query, [], transactionSuccess, transactionError);

        function transactionSuccess(tx, result) {

            log(`Search trips successfully.`);

            displayList(result.rows);

            $('#page-list #frm-search').trigger('reset');

            $('#page-list #frm-search').popup('close');

        }

    });

}

**The search function , user can search fron name, destination,date**