A cartoon of cars and buildings

AI-generated content may be incorrect.

Table of Contents

[Scenario and Scope: Define the task (Task 1A) (**±** 200 words) 1](#_Toc159454338)

[1. Topic 1](#_Toc159454339)

[2. Purpose of program 1](#_Toc159454340)

[3. Possible solution 1](#_Toc159454341)

[4. Scope 1](#_Toc159454342)

[User Requirements (TASK 1B) 2](#_Toc159454343)

[Use a table OR a 'use case diagram' 2](#_Toc159454344)

[Design the Database (TASK 2) 3](#_Toc159454345)

[Database Tables 3](#_Toc159454346)

[Relationship between the tables 3](#_Toc159454347)

[Data Dictionary (TASK 3A) 4](#_Toc159454348)

[Classes and Objects 4](#_Toc159454349)

[Data Dictionary (TASK 3B) 5](#_Toc159454350)

[(Text Files and Arrays) or Advanced Concepts 5](#_Toc159454351)

[ Text File 5](#_Toc159454352)

[ Arrays 5](#_Toc159454353)

[ Advanced programming constructs 5](#_Toc159454354)

[Navigation/Description of Flow Diagram (TASK 4A) 6](#_Toc159454355)

[Option 1 6](#_Toc159454356)

[Option 2 7](#_Toc159454357)

[Design the Graphical User Interface (GUI) (TASK 4B) 8](#_Toc159454358)

[Screen 1: <screen name> 8](#_Toc159454359)

[Screen 2: <screen name> 8](#_Toc159454360)

[Screen 3… 8](#_Toc159454361)

[IPO – Software Design Tool (TASK 5) 9](#_Toc159454362)

[Data Input validation (TASK 5A) 9](#_Toc159454363)

[Screen 1: Screen name 10](#_Toc159454364)

[INPUT 10](#_Toc159454365)

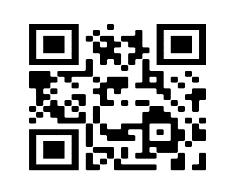
[Screen 2: Screen name 10](#_Toc159454366)

[INPUT 10](#_Toc159454367)

[Data Processing (TASK 5B) 11](#_Toc159454368)

[Data Output (TASK 5C) 12](#_Toc159454369)

Use the **headings** below in your PAT and complete it as described. The scenario and scope are often easier to do after you completed the other sections in Phase 1, but it has to be the first part of your Phase 1 document.



<https://youtu.be/dlMtjyK1m7o>

# Scenario and Scope: Define the task (Task 1A)

# (± 200 words)

Give a brief description (approximately 200 words) in your own words in terms of the problem/task and how the project will solve the problem. Use the subheadings in the numbered list (numbers 1 to 4) below for the description of the scenario and score.

# 1. Topic

# The topic I selected is “Vehicle License Management,” which is based on creating a desktop application that simplifies the handling of motor vehicle licenses. My project, *LicenseLink*, aims to assist both motorists and licensing departments by digitizing and streamlining tasks such as license generation, renewals, fine checking, and access to services like test applications and routing to nearest licensing stations.

# 2. Purpose of Program

# Many South Africans face challenges with manually managing vehicle licensing – including forgetting renewal dates, paying fines late, or not knowing how or where to apply for license-related services. This often leads to unnecessary penalties, expired licenses, and time-consuming queues. *LicenseLink* is designed to centralize and simplify these processes in one user-friendly program. It saves time, improves compliance, and reduces confusion for users.

# 3. Possible Solution

# *LicenseLink* will consist of three main tab screens:

# Tab 1 – License Management: For generating new licenses and renewing existing ones. This screen will capture vehicle and owner data and store it in a database.

# Tab 2 – Fines and Test Applications: Users can check for outstanding fines and be redirected to online learner’s or driving license test applications. Fine data will be retrieved from a database.

# Tab 3 – Station Locator: Uses location input to display nearby licensing stations using routing logic. A text file may be used to store static data about station addresses and operating hours.

# All data will be managed through an ADO-connected database that stores user profiles, licenses, and fines.

# 4. Scope

# The application will not process actual payments, provide real-time government updates, or integrate live geolocation services. It also will not interface directly with the Department of Transport systems but instead simulate their functionality with stored data.

# User Requirements (TASK 1B)

## Use a table OR a 'use case diagram'

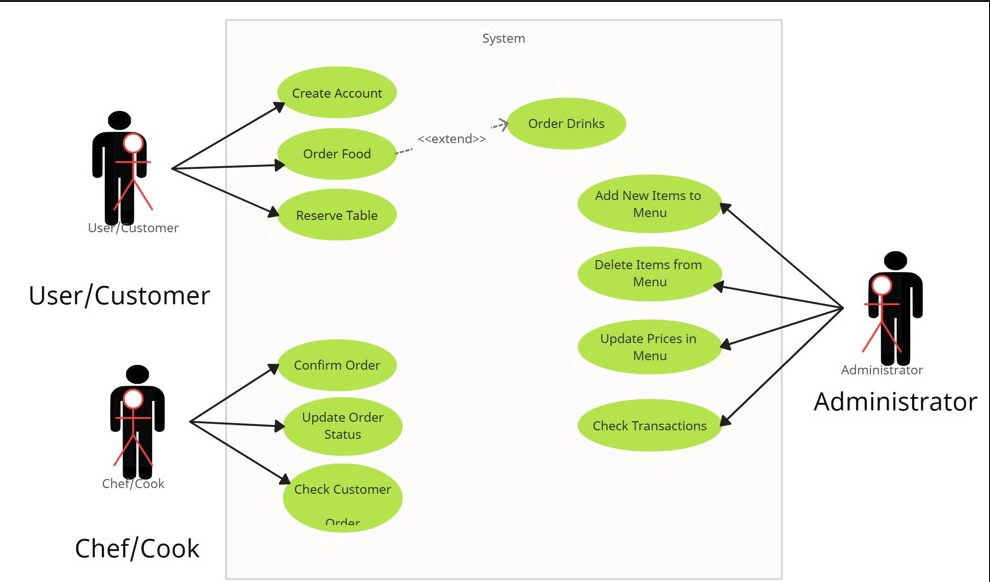
Complete the tables in detail to **clearly** describe the users (target audience) that will use your program. You need at least TWO different TYPES of users. Each of your users must use the program differently, although some of the tasks may be the same.

1. Vehicle Owner**:** (Give the user a "name" such as client, so it is clear who the user/users is/are)

|  |  |
| --- | --- |
| Role | Regular user / vehicle owner |
| Activity | -Create an account  - Delete their own account  - Generate a new vehicle license  - Renew an existing license  - Check outstanding fines  - View licensing station locations  - Book driving/learner’s license tests |
| Limitations | * Cannot manage other users * Cannot process or approve test applications * Cannot perform admin maintenance tasks |

1. Admin App Manager: (Give the user a "name" such as administrator, so it is clear who the user/users is/are)

|  |  |
| --- | --- |
| Role | Admin/official at the licensing departmentdocs |
| Activity | * Create and delete user accounts manually * Process and manage test applications * Access all user data for maintenance * Perform data cleanup or updates |
| Limitations | * Cannot generate or renew licenses for users * Cannot pay fines or book tests on behalf of users unless simulating the process for testing |



|  |  |  |
| --- | --- | --- |
| Task | Administrator | User |
| Log in to the system | Yes | Yes |
| Add a new User | Yes | Yes (self-registration) |
| Delete a User | Yes | Yes (delete own account) |
| Generate a new vehicle license | No | Yes |
| Renew an existing vehicle license | No | Yes |
| Check outstanding fines | Yes | Yes |
| View licensing station locations | Yes | Yes |
| Book learner’s or driving license test | Yes (manage) | Yes(request) |
| Approve or process test applications | Yes | No |
| Perform database or system maintenance | Yes | No |

# Design the Database (TASK 2)

Show the design of the database, including the tables, relationships, field names, field types and field sizes.

## Database Tables

## **Table Name: tblUsers**

|  |  |  |  |
| --- | --- | --- | --- |
| PK/FK | Field Name | Field Type | Field Size |
| PK | userID | Short Text | 20 |
|  | Username | Short Text | 25 |
|  | Password | Short Text | 25 |
|  | firstName | Short Text | 25 |
|  | lastName | Short Text | 25 |
|  | emailAddress | Short Text | 25 |
|  | contactNumber | Short Text | 15 |
|  | dateOfBirth | DateTime | 8 |
|  | accountStatus | Short Text | 20 |
|  | createdAt | Date/Time | 8 |
|  | updatedAt | Date/Time | 8 |
| FK | userRoleID | Integer | 2 |

## **Table Name: tblFines**

|  |  |  |  |
| --- | --- | --- | --- |
| PK/FK | Field Name | Field Type | Field Size |
| PK | fineID | Short Text | 20 |
|  | fineAmount | Currency | 8 |
|  | fineDate | Date/Time | 8 |
|  | createdAt | Date/Time | 8 |
|  | updatedAt | Date/Time | 8 |
| FK | ownerID | Short Text | 20 |

## **Table Name: tblLicenses**

|  |  |  |  |
| --- | --- | --- | --- |
| PK/FK | Field Name | Field Type | Field Size |
| PK | licenseID | Short Text | 255 |
|  | expirationDate | Date/Time | 8 |
|  | createdAt | Date/Time | 8 |
|  | updatedAt | Date/Time | 8 |
| FK | typeID | Integer | 2 |
| FK | ownerID | Short Text | 20 |
| FK | provinceID | Integer | 2 |

## **Table Name: tblLicenseTypes**

|  |  |  |  |
| --- | --- | --- | --- |
| PK/FK | Field Name | Field Type | Field Size |
| PK | typeID | Integer | 2 |
|  | typeName | Short Text | 255 |

## **Table Name: tblProvinces**

|  |  |  |  |
| --- | --- | --- | --- |
| PK/FK | Field Name | Field Type | Field Size |
| PK | provinceID | Integer | 2 |
|  | provinceName | Short Text | 255 |

## **Table Name: tblLicensingStations**

|  |  |  |  |
| --- | --- | --- | --- |
| PK/FK | Field Name | Field Type | Field Size |
| PK | stationID | Integer | 2 |
|  | stationName | Short Text | 100 |
|  | stationAddress | Long Text | - |
|  | stationContact | Short Text | 30 |
|  | createdAt | Date/Time | 8 |
|  | updatedAt | Date/Time | 8 |
|  | coordinates | Short Text | 255 |

## **Table Name: tblTestApplications**

|  |  |  |  |
| --- | --- | --- | --- |
| PK/FK | Field Name | Field Type | Field Size |
| PK | applicationID | Short Text | 20 |
|  | testDate | Date/Time | 8 |
|  | applicationStatus | Short Text | 20 |
|  | createdAt | Date/Time | 8 |
|  | updatedAt | Date/Time | 8 |
| FK | userID | Short Text | 20 |
| FK | stationID | Integer | 2 |

## **Table Name: tblUserRoles**

|  |  |  |  |
| --- | --- | --- | --- |
| PK/FK | Field Name | Field Type | Field Size |
|  | RoleName | Short Text | 255 |
| PK | RoleID | Integer | 2 |

## Relationship between the tables



<https://youtu.be/ZWEPU019Nmw>

Relationship Description: *<Explain the link between relational tables>*

tblUsers

tblFines

tblUsers

tblLicenses

has

has

tblUsers

tblTestApplications

tblUserRoles

tblUsers

has

are

tblLicenseTypes

tblLicenses

tblProvinces

tblLicenses

are

has

tblLicensingStations

tblTestApplications

has

Example of the tables:

|  |  |  |
| --- | --- | --- |
| **Parent table name** |  | **Child table name** |
| Primary Key Field (PK)**1** |  | Primary Key Field (PK) |
| Field |  | Field |
| Field |  | Field |
| Field |  | Field |
| Field |  | Field |
| Field |  | **∞**Foreign Key Field (FK) |

# Data Dictionary (TASK 3A)

## Classes and Objects

* Application must contain at least ONE object class.
* How is the class going to be used?
* ***Class description and class diagram*:**
* **Class diagram** Use UML here. NOTE: UMLs are sometimes asked in **theory exams**, this is your time to practice.
* **Class description**: **Clearly explain** the purpose of your class in your program and how it will be used to add value to your program.
* Explain where objects can be used in your program application so that it adds value to your application

|  |
| --- |
| **Class Name:** TUser |
| ***Attributes*** |
| * fFirstname : string * field\_name : data\_type * field\_name : data\_type |
| ***Methods*** |
| + Constructor Create ( parameters : data\_type )  + function FuncName : return\_data\_type //Explain function  + procedure ProcName ( paramters : data\_type ) //Explain procedure |

- 🡪 private

+ 🡪 public

# Data Dictionary (TASK 3B)

## Text Files and (Arrays or Advanced Concepts)

### Text File

**Text File: LoginTrail.txt**

**Purpose**

The purpose of the LoginTrail.txt text file is to **record each user login** attempt. It stores a history of logins including the user's unique ID, the date and time of login, and their user type (Admin or Car Owner). This file is useful for auditing and tracking access without placing unnecessary load on the database.

**Read/Write Usage**

* **Write to File:**  
  The program writes to the file during a successful login event (inside the imgLoginButtonClick procedure). It appends a new line each time someone logs in.
* **Read from File:**  
  In this specific implementation, the file is only written to, but it could be read from in the future for displaying login history to the admin or tracking misuse.

**When This Happens:**

* The file is written to **immediately after a successful login** (both for Admins and regular users).
* The Append() function is used so that existing entries are preserved, and each new login is added to the end of the file.

**Text File Format**

Each line in the file is formatted as follows:

<UserID>#<Login Date>#<Login Time>#<User Type>

**Example:**

USR123#2025/05/01#09:45:32#Car Owner

* UserID links the text file to the tblUsers table in the database (foreign key style).
* The rest of the line stores extra data not found in the database (login time & user type label).
  + - **Extract of data:** Place three lines as an example of your text file here.

### Arrays

Your application must use a one-/two-dimensional array.

If you have a Login File, you could make a LOG Object, and then read the text file into an array of these Log Objects. They can now be sorted and Filtered to produce stats of the people who log in, and the times etc

OR

### Advanced programming constructs

Must apply programming concepts such as inheritance, polymorphism, overloaded methods, method binding, array of object etc.

# Navigation/Description of Flow Diagram (TASK 4A)

Clearly indicate the logical program flow and/or navigation between screens. Use a UI flow diagram or any other form of illustration to present a global overview of the project/system navigation.

Do the following:

* + - Create a *flowchart* of your program.
    - Do **NOT** include the help and reset buttons in your flowchart.
    - Go to **draw.io** website to create your Flowchart.
    - Watch the video on Dandel10n Delphi YouTube channel in the PAT paylist named: **PAT: Flowcharts on draw.io**



<https://youtu.be/eNRe5Ofrbgo>

## Option 1

# Design the Graphical User Interface (GUI) (TASK 4B)

## Screen 1: Login Screen

A screenshot of a login screen

AI-generated content may be incorrect.



<https://youtu.be/16Qb8ZOPMIA>

## Screen 2: Sign up

*A screenshot of a login screen

AI-generated content may be incorrect.*

## Main Screen

*A screenshot of a computer

AI-generated content may be incorrect.*

# Profile

# A screenshot of a computer AI-generated content may be incorrect.

# Routing

# A screenshot of a computer screen AI-generated content may be incorrect.

# Generation

# A screenshot of a computer AI-generated content may be incorrect.

# IPO – Software Design Tool (TASK 5)

## Data Input validation (TASK 5A)

Input interfaces and validation.

You need to validate at least

* + - 4 different data types
    - At least FOUR inputs validated, including:

- Validate for NULL/empty field AND

- Test if value was selected in a selection component

* + - Associated error messages

For example:



<https://youtu.be/5fjW_AtGGXM>

* Test if an Edit has been left empty.
* Test if the user selected something from the ComboBox / RadioGroup / ListBox.
* Use Try .. catch StrToFloat for real input (real data type).
* Test the length of a string entered (string data type).
* Test if an integer number is in a certain range. (Integer data type).
* Test if the file exists (Text file data type).

For the *DateTimePicker*, you can test if they selected a date before or after a certain date depending on your scenario. This will give you another data type check should you need one.

Specify the format, data types, source of input, validation of input and error checking mechanisms of at least TWO INPUT interfaces.

Use the following format:

### Screen 1: Login Screen

### INPUT

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Input*** | ***Source*** | ***Data Type*** | ***Format*** | ***GUI Component*** | ***Validation*** |
| sUsername | Keyboard | String | General text | edtUsernameLogin | *Check if empty using*  *TValidation.notEmpty* |
| sPassword | Keyboard | String | General text | edtPasswordLogin | *Check if empty using*  *TValidation.notEmpty* |
| Text File | File | File | FileExists | LoginTrail.txt | Check if file exists using FileExists(…) |
|  |  |  |  |  |  |

### Screen 2: Screen name

### INPUT

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Input*** | ***Source*** | ***Data Type*** | ***Format*** | ***GUI Component*** | ***Validation*** |
| Example:  firstName | Keyboard  *Other options: text file, database, array* | string | General text | edtFirstName | *Check if not empty using Tvalidation class* |
| Contact Number | Keyboard | String | Numeric String | edtContact | Check if length =10 |
| dateOfBirth | User | Date | Mm/dd/yyyy | dtpDOB | Ensure selected date is not in the future |
|  |  |  |  |  |  |

## Data Processing (TASK 5B)

**WHAT** and **HOW** the processing will need to be done

Specify the processing that needs to be done and provide algorithm(s)/formulae to show how the processing will be done.

You need to **list** a total of **8** processes in your program (You need to write **algorithms** for **4** of these processes. Ensure that the FOUR algorithms you write have **significant** processing. Reading from the text files, string handling, your array manipulation planned earlier, or Database manipulation using Delphi code, should provide you with the proper algorithms to write.



<https://youtu.be/oakEThuYQ6Y>

**NOTE:** (SQLs are NOT considered as algorithms).

PROCESSING

|  |  |
| --- | --- |
| **What processing needs to be done** | **How processing will be done** |
| *<List AT LEAST EIGHT processes (****brief explanation) over 2 (or more) IPO tables>*** | *<Give Pseudo code or example code/algorithm of FOUR out of the EIGHT processes listed>* |
| Generate new license (creates a new record in tbl licenses) | procedure TfrmLicenseGen.btnGenClick(Sender: TObject);  {  simple license generation based on province and last entry in DB  }  var  sProv, sType, sLicense, sNewest, sNum: string;  i: Integer;  begin  sProv := inttostr(cmbProvince.Itemindex);  sType := inttostr(cmbLicenseType.Itemindex);  with DataModule1 do  begin  try  ADOQuery1.close;  ADOQuery1.SQL.text :=  'SELECT Top 1 \* FROM tblLicenses WHERE provinceID = '+sType+' ORDER BY createdAt DESC';  ADOQuery1.open;  if not ADOQuery1.IsEmpty then  begin  sNewest := ADOQuery1.FieldByName('licenseID').AsString;  end  else  begin  sNewest := '';  end;  except  on E: Exception do  sNewest := '';  end;  end;  if sNewest = '' then  begin  sNum := '1';  end  else  begin  for i := 1 to length(sNewest) do  begin  if sNewest[i] in ['0' .. '9'] then  begin  sNum := sNum + sNewest[i];  end;  end;  sNum := inttostr(strtoint(sNum) + 1);  end;  case cmbProvince.Itemindex of  0:  sLicense := 'B ' + sNum + ' GP';  1:  sLicense := 'CA ' + sNum;  2:  sLicense := 'ND ' + sNum;  3:  sLicense := 'EL ' + sNum + ' EC';  4:  sLicense := 'L ' + sNum + ' L';  5:  sLicense := sNum + ' MP';  6:  sLicense := sNum + ' NW';  7:  sLicense := sNum + ' NC';  8:  sLicense := sNum + ' FS';  end; |
| Login | procedure TfrmLogin.imgLoginButtonClick(Sender: TObject);  var  sLine: string;  {  Login using sql query  - check for matching records  }  begin  bAdmin := false;  if TValidation.notEmpty(edtUsernameLogin.text, 'Username') then  begin  if TValidation.notEmpty(edtPasswordLogin.text, 'Password') then  begin  sUsername := edtUsernameLogin.text;  sPassword := edtPasswordLogin.text;  if chbAdmin.checked then  begin  with Datamodule1 do  begin  ADOQuery1.sql.text := 'SELECT \* FROM tblUsers WHERE Username = "' +  sUsername + '" AND Password = "' + sPassword +  '" AND userRoleID = 0';  ADOQuery1.open;  if ADOQuery1.RecordCount > 0 then  begin  showmessage('Successful login');  sID := ADOQuery1.FieldByName('UserID').AsString;  bAdmin := True;  // Text file login trail  if FileExists('LoginTrail.txt') then // TO DO: USER TYPES  begin  Assignfile(fDatafile, 'LoginTrail.txt');  Append(fDatafile);  sLine := sID + '#' + Datetostr(Date) + '#' + timetostr(time) + '#'  + 'Admin';  writeln(fDatafile, sLine);  closefile(fDatafile);  end;  frmMain.show;  frmLogin.hide;  end  else  begin  showmessage('unsuccessful');  end  end;  end  else  begin  with Datamodule1 do  begin  ADOQuery1.sql.text := 'SELECT \* FROM tblUsers WHERE Username = "' +  sUsername + '" AND Password = "' + sPassword + '"';  ADOQuery1.open;  if ADOQuery1.RecordCount > 0 then  begin  showmessage('Successful login');  sID := ADOQuery1.FieldByName('UserID').AsString;  // Text file login trail  if FileExists('LoginTrail.txt') then // TO DO: USER TYPES  begin  Assignfile(fDatafile, 'LoginTrail.txt');  Append(fDatafile);  sLine := sID + '#' + Datetostr(Date) + '#' + timetostr(time) + '#'  + 'Car Owner';  writeln(fDatafile, sLine);  closefile(fDatafile);  end;  frmMain.show;  frmLogin.hide;  end  else  begin  showmessage('unsuccessful');  end  end  end;;  end;  end  else  begin  showmessage('Email cannot be blank');  end;  end; |
| Sign up | procedure TfrmSignup.imgLoginButtonClick(Sender: TObject);  {  Sign up of user  - extract from edit boxed  - check for unique vals  - update DB  }  var  sUser, sPass, sPassConfirmed, sFirstName, sLastName, sEmail, sContact,  sUserID, OwnerID, AdminID, sType: string;  DOB: tDatetime;  begin  sPass := edtPassword.text;  sPassConfirmed := edtConfirmPassword.text;  if sPass = sPassConfirmed then  begin  if (tdb.ExistingRecordCheck('tblUsers', 'Username', sUser))  { (tdb.ExistingRecordCheck('tblUsers', 'Email', sEmail)) } then  begin  if (TValidation.notEmpty(edtUsername.text, 'Username')) AND  (TValidation.notEmpty(edtFirstName.text, 'First Name')) AND  (TValidation.notEmpty(edtLastName.text, 'Last Name')) then  begin  sUser := edtUsername.text;  sFirstName := edtFirstName.text;  sLastName := edtLastName.text;  end;  sEmail := edtEmail.text;  if length(sContact) = 10 then  begin  sContact := edtContact.text;  end  else  begin  showmessage('Contact number must be ten digits long');  end;  if dtpDOB.Date < Date then  begin  DOB := dtpDOB.Date;  end;  sUserID := copy(sFirstName, 1, 1) + copy(sLastName, 1, 1) +  InttoStr(random(10000 - 1 + 1) + 1);  try  if chbAdmin.checked then  begin  sType := '0';  end // if  else  begin  sType := '1';  end; // else  with Datamodule1 do  begin  tblUsers.insert;  tdb.UpdateField('userID', sUserID, tblUsers);  tdb.UpdateField('Username', sUser, tblUsers);  tdb.UpdateField('firstName', sFirstName, tblUsers);  tdb.UpdateField('lastName', sFirstName, tblUsers);  tdb.UpdateField('Password', sPass, tblUsers);  tdb.UpdateField('contactNumber', sContact, tblUsers);  tdb.UpdateField('dateOfBirth', DateToStr(DOB), tblUsers);  tdb.UpdateField('accountStatus', 'Active', tblUsers);  tdb.UpdateField('userRoleID', sType, tblUsers);  tdb.UpdateField('createdAt', DateToStr(Date), tblUsers);  tdb.UpdateField('updatedAt', DateToStr(Date), tblUsers);  tdb.UpdateField('emailAddress', sEmail, tblUsers);  tblUsers.post;  tblUsers.refresh;  end; // if DM  showmessage('Account successfully added');  frmLogin.show;  frmSignup.hide;  except  on E: Exception do  showmessage('Problem with adding data to DB');  end;  end; // if exist  end // if passconfirm  else  begin  showmessage('Passwords are not the same');  end;  end; |
| Get directions to licensing station | procedure TfrmRouting.openMap(coords: string);  begin  ShellExecute(0, 'open', PChar('https://www.google.com/maps?q=' + coords), nil,  nil, SW\_SHOWNORMAL);  end; |
| Account deletion |  |



<https://youtu.be/oakEThuYQ6Y>

## Data Output (TASK 5C)

Provide a clear description to indicate the output requirements of the system for at least TWO of the main interfaces.

You need to describe **all** the output of **TWO** of your screens / Tab Sheetswith **SIGNIFICANT** output. Create a table as below to describe 2 of your screens / your Tab Sheets.

**NOTE**: when you describe the output in the DB grid, make sure you add the field names and format of the output of everything that will be displayed.

|  |  |  |
| --- | --- | --- |
| **Name of Screen 1 / tab sheet** | | |
| **Output** | **Format (type, size)** | **Output Component** |
| Cost | You owe: <Cost as a Currency, with two decimal places> | Label |
| Name, Surname and Total Cost | In neat columns with the headings Name, Surname and Cost. Cost will be displayed in Rand currency, with two decimal places. | RichEdit |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Name of Screen 2 / tab sheet** | | |
| **Output** | **Format (type, size)** | **Output Component** |
| Cost | The label shows "You owe the following" and the edit displays the value in Rand Currency | Labelled edit |
| Date of Sale | Date – dd/mm/yyyy | RichEdit |
|  |  |  |

**Resources**:

* Text to Speech by Mr Long: <https://youtu.be/TqCZUioQDeg>
* Add a video by Mr Long: <https://youtu.be/lbPIwgzhWCE>
* All QR codes extracted from: ***PAT 2023*** by Lilané le Grange & Karen Andersen
* <http://www.delphibasics.co.uk/>
* <http://festra.com/>