2022

Cool Bikes ERP System

FINAL PROJECT REPORT

Jose Maria Rico Leal

Dublin Business School
Higher Diploma in Science in Computing Software Development
Student Number: 10539218
Email: 10539218@mydbs.ie
Under Supervision of Clive Gargan
09/09/2022

Abstract

The objective of this project is purely to explore other capabilities of python, as I use it every day in my job for tasks such as, importing/exporting data from SQL, automating reports and other data related tasks. Two additions for me thanks to this project, I have learnt how to build a graphic user interface thanks to *"tkinter"* library and how to manage records in a small database using the build-in function *"Sqlite3"*. For the completion of this project, I have created an imaginary company *"Cool Bikes Dublin, L.T.D."* and I will walk you through from specification, implementation and testing, to see all steps involved to get the final outcome.

Acknowledgements

Big thank you to Dublin Business School for giving me the chance to take part of this superb program and for allowing me to complete my studies in HigherDiploma in Science in Computing.
I would also like to thank my project supervisor Clive Gargan for his support, and I do not want to forget all lecturers that have guided me throughout the completion of this diploma.

Table of Contents

1. Introduction

The purpose of this project is to create a simple ERP system for a bike shop. We will be able to manage sales and stocks records. This artefact will be coded in python and two main modules will be used here “Tkinter” and “Sqlite3”. I wanted to use python to design a graphic user interface (GUI), and with this project I will be fulfilling this aim.
Besides this the application uses basic Python functions to generate menu options, title boxes, input message boxes, list and print text on the screen.
To retrieve information about bikes, users and sales I have created several functions to accomplish this.
The artefact uses tuples and lists to structure all the operations performed.
We can log into the app using two types of accounts manager or mechanic. By login in as a manager we will have the following functionalities:

- Add bikes to the inventory by specifying, bike ID, type, specs, group set, price, stock and customer.
- Modify a bike already inserted into the inventory except bike ID, we will be able to amend any of the fields stated above.
- Visualise all invoices.
- Create and modify users.

In contrast if we log in as mechanic functionalities are as it follows:

- Have access to the full bike inventory.
- Have the ability of executing a sale by creating an invoice.

The database has three tables, products, sales and users. Tables are not connected between them, however we will see visuals in the app that contain records from more than one table, I will deep dive on this in implementation section.
To wrap up the introduction, I would like to mention which method I have selected for development. Using Agile I could divide the project into sprints, first sprints I was trying to design the layout and functionality and last sprints was all about coding and trying to not crash the app. I can conclude that the simple app works well without any bug or crash.

1.1 High level project map

In this part we are going to see at a high level how the program works, when firing up the app a pop-up screen asks for credentials, there are only two types of users for that can log in. Depending on which privileges you have *"Manager"* or *"Mechanic"* that will determine features available for you. *"Manager"* is the highest level within the app.

Imagine that we log in as *"Mechanic"*, first thing we see is the bike inventory, in that way the employee knows if an order can be processed or there is a need to ask to the *"Manager"*, for a new item or stock addition. The other feature available at this level is to process an order, we can search for a bike based on specs and determine quantity and finally create the invoice, these invoices are visible at *"Manager"* level. Two more options available here, log out and change user.

Once we know the abilities of a *"Mechanic"* is time for the *"Manager"*, first screen that will appear is for bike addition, seven fields to fill up are required. As soon as the bike is inserted, we will be able to see it in the second screen, stocks, here we are able to amend certain fields. Third screen is for invoices, this is basically an archive. Fourth screen is for users, here we can create a user, or amend an existing one as well as deleting it. Same as the *"Mechanic"* we can log out or change user.

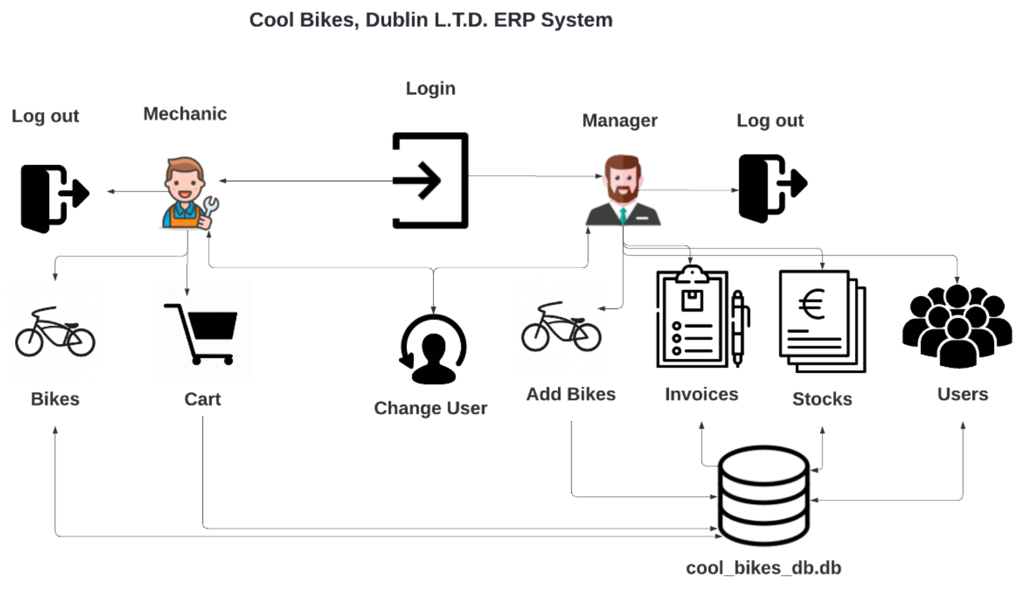


Fig.1: High level project map.

1.2 Sprints

The project has been completed in two months, and I have used the Agile methodology to identify the project crucial tasks to draw the plan. We can see in figure 2 how the process was carried out, this chart helped me to keep track of progress and assist with the key hand out dates. *"Agile project management works off the basis that a project can be continuously improved upon throughout its life cycle, with changes being made quickly and responsively. Agile is one of the most popular approaches to project management due to its flexibility, adaptability to change, and high level of customer input"* (https://www.wrike.com/project-management-guide/faq/what-is-agile-methodology-in-project-management/).

There are twelve phases divided into two sprints, development and implantation.

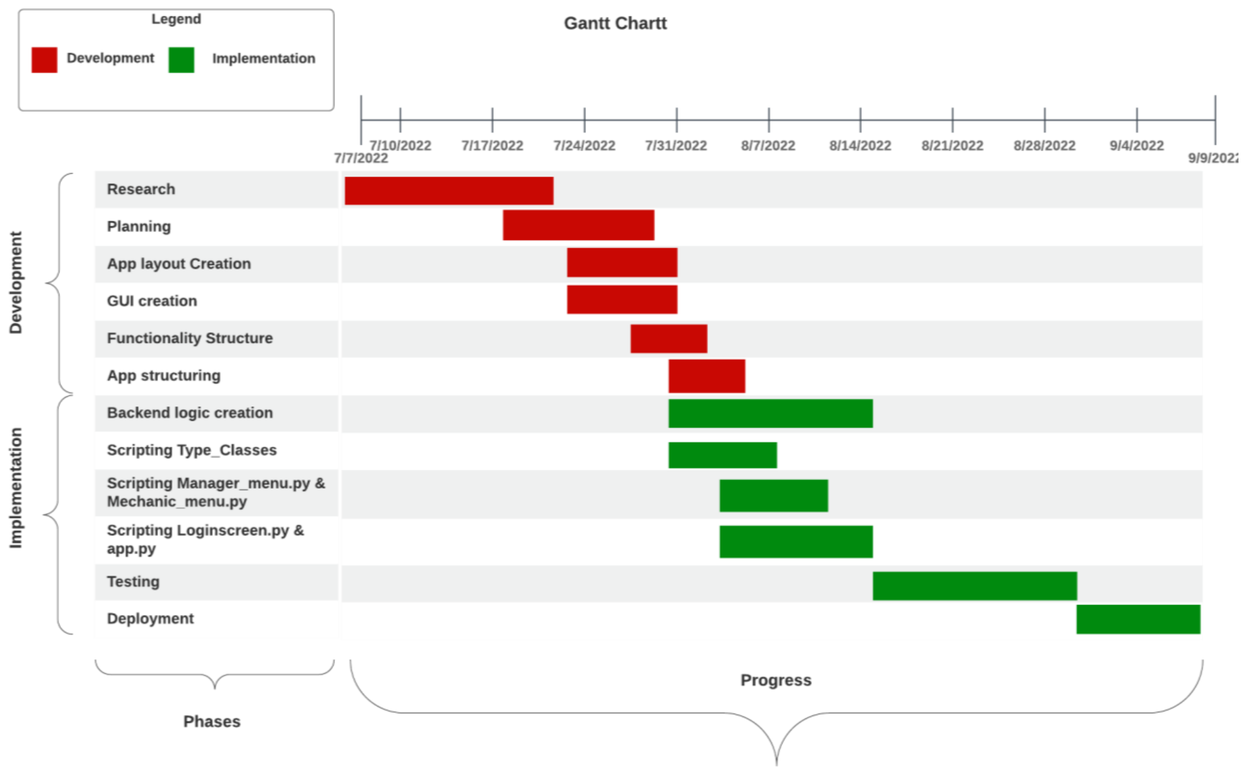


Fig.2: Gantt Chart.

Aside note about Agile method: *"Agile project management works off the basis that a project can be continuously improved upon throughout its life cycle, with changes being made quickly and responsively. Agile is one of the most popular approaches to project management due to its flexibility, adaptability to change, and high level of customer input"* (https://www.wrike.com/project-management-guide/faq/what-is-agile-methodology-in-project-management/).

1.3 Assumptions

Only one assumption has been made. This is an imaginary business for the purpose of this project.

2. Background/Literature Review

I wanted to develop an app with python where a user could log into it, and it could also perform actions with features designed. That is how the background of this project started. At that point my limitations were tied to creating a GUI and using a database in python, I did not know about python modules *"tkinter"* and *"sqlite3"*. After some research I have found interesting sources that helped me out to develop this app, I will name chronologically one by one:

Source	Link
Using this tutorial is how I get started with the module *"Tkinter".* It shows you how to create buttons, texts boxes, manage application layout and associating buttons with python functions.	https://docs.python.org/3/library/tkinter.html
Here I have learnt how to create the database and how to use important methods such as connect(), cursor(), execute(), fetchall(), close cursor and connection objects.	https://www.sqlitetutorial.net/sqlite-python/
Thanks to these instructions I was able to put the GUI and the database all together. I will use *"tkinter"* in the frontend and *"sqlite3"* in the backend. Actions as search, insert,add, update an delete from a database are well explained here.	https://www.python4networkengineers.com/posts/python-intermediate/create_a_tkinter_gui_with_sqlite_backend/
Agile method information source.	https://www.atlassian.com/agile
Testing and results, I need it a method to perform testing, behavioural testing was carried out to perform the tests.	https://www.educative.io/answers/what-is-behavior-testing-in-software-testing

3. Specification and Design

3.1 Overview

The simple ERP system for "Cool bikes, L.T.D." does exactly what his owner wanted. When we meet him, he wanted an app that, could keep record of bikes, invoices and could manage users. For users you can log in as a *"manager"* or *"mechanic".* Once I knew the details, I started coding a python app that uses *"tkinter"* for GUI and *"sqlite3"* for creating a database.

Five python scripts compose the app, these are app.py, Loginscreen.py, Manager_menu.py, Mechanic_menu.py and Tkinter_classes.py, in section 3.3 scripts I will go one by one explaining their use.

3.2 Functional and non-functional requirements

A comprehensive list of requirements is listed below:

	Functional Requirements
Number	**Requirement**
1	Two levels of employee must be in place, we will be able to log in as *"Manager"* or *"Mechanic".*
2	As a *"Manager"* we will have the ability of adding bikes.
3	As a *"Manager"* we will have the ability of controlling stocks.
4	As a *"Manager"* we will have the ability of keeping track of invoices.
5	As a "Manager" we will have the ability of managing users.
6	As a *"Mechanic"* we can see the inventory.
7	As a *"Mechanic"* we perform the sale and create invoices.
8	Both levels can log off and change user at any time.
	Non-Functional Requirements
Number	**Requirement**
1	This artefact has to be user friendly.

3.3 Scripts

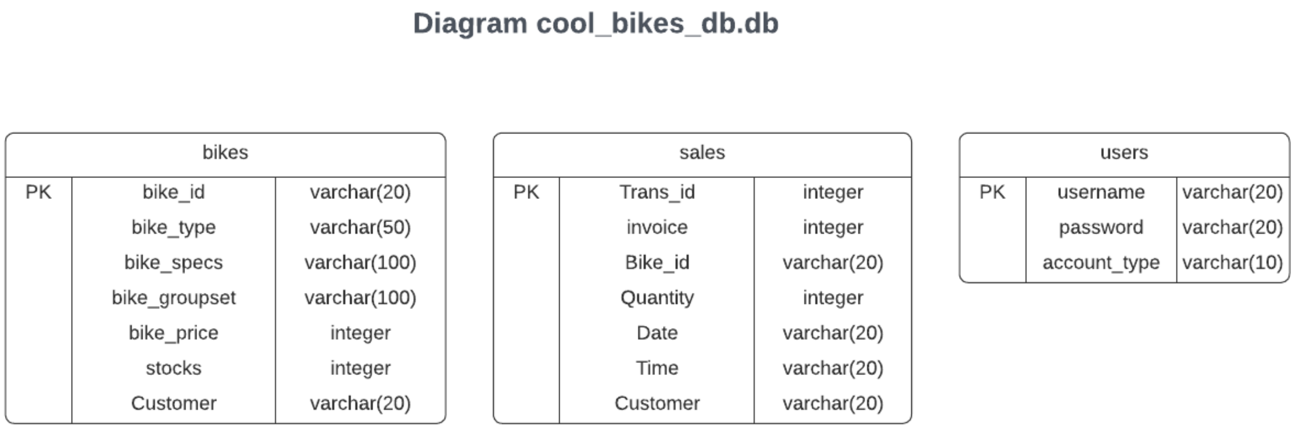
In this section I will be dissecting each of the app scripts:

1. Tkinter_classes.py, here we have two classes *"class typeauto(tkinter.Entry):"* and *"class typebox(tkinter.ttk.Combobox):"*.The first class allows you to accept values from user. It helps for term search and autocompletion. The second one does the same, but instead for type entry text, it gives you a drop down list.
2. Mechanic_menu.py, this is the employee screen where we can see the inventory, log an invoice, switch user and log off. There is one class called *"Mechanic"* and inside it 10 methods to action all features.
3. Manager_menu.py, the largest script for this project, it has a class called *"Manager"* and twenty-three methods belonging to it. This is the highest level for a user, we can add bikes, modify stocks, keep track of invoices, manage users, log off and change user.

4. Loginscreen.py, is it name says this is the welcome screen to log into the app. There is one class formed by twelve methods.
5. app.py, is the main script. It has a class called *"Main"* formed by five methods, this class calls previous classes as in *"Screen", "Manager"* and *"Mechanic".* We run the app by selecting this scrip as startup or just clicking on it.

3.4 Database

For our database we have three tables, bikes, sales and users, we can see how the look like in a diagram:



bikes		
PK	bike_id	varchar(20)
	bike_type	varchar(50)
	bike_specs	varchar(100)
	bike_groupset	varchar(100)
	bike_price	integer
	stocks	integer
	Customer	varchar(20)

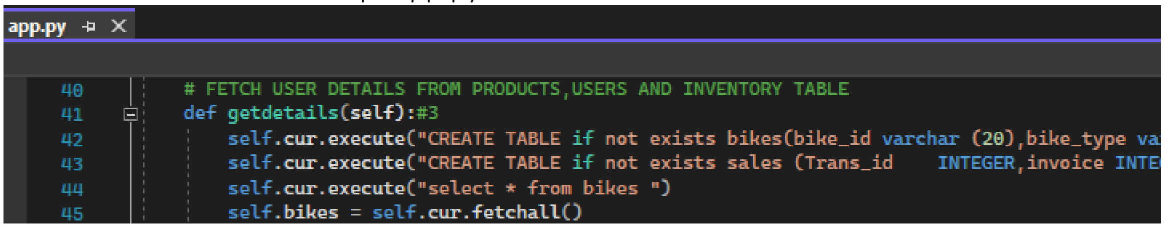
sales		
PK	Trans_id	integer
	invoice	integer
	Bike_id	varchar(20)
	Quantity	integer
	Date	varchar(20)
	Time	varchar(20)
	Customer	varchar(20)

users		
PK	username	varchar(20)
	password	varchar(20)
	account_type	varchar(10)

Fig.3: DBMS ER Diagram.

Important thing to note here is that entities are not linked, this is due to the nature of this app, when we need to combine data from different tables, we use data stored in tuples to build up tables.

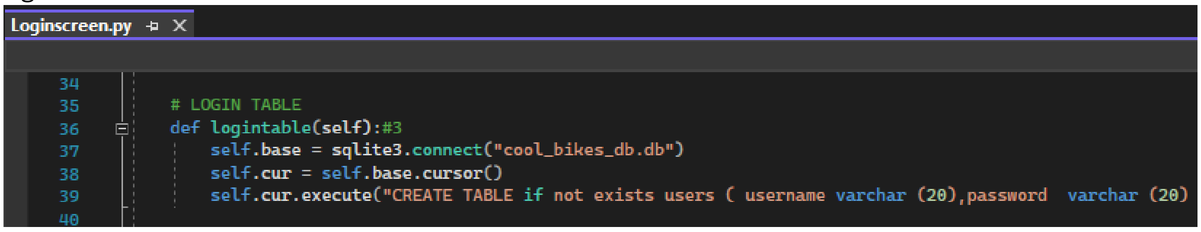
Next, I would like to point out how and where tables are created. Tables bikes and sales are created in script app.py inside method number three:



```
# FETCH USER DETAILS FROM PRODUCTS,USERS AND INVENTORY TABLE
def getdetails(self):#3
    self.cur.execute("CREATE TABLE if not exists bikes(bike_id varchar (20),bike_type va
    self.cur.execute("CREATE TABLE if not exists sales (Trans_id    INTEGER,invoice INTE
    self.cur.execute("select * from bikes ")
    self.bikes = self.cur.fetchall()
```

Fig.4: bike and sales table creation.

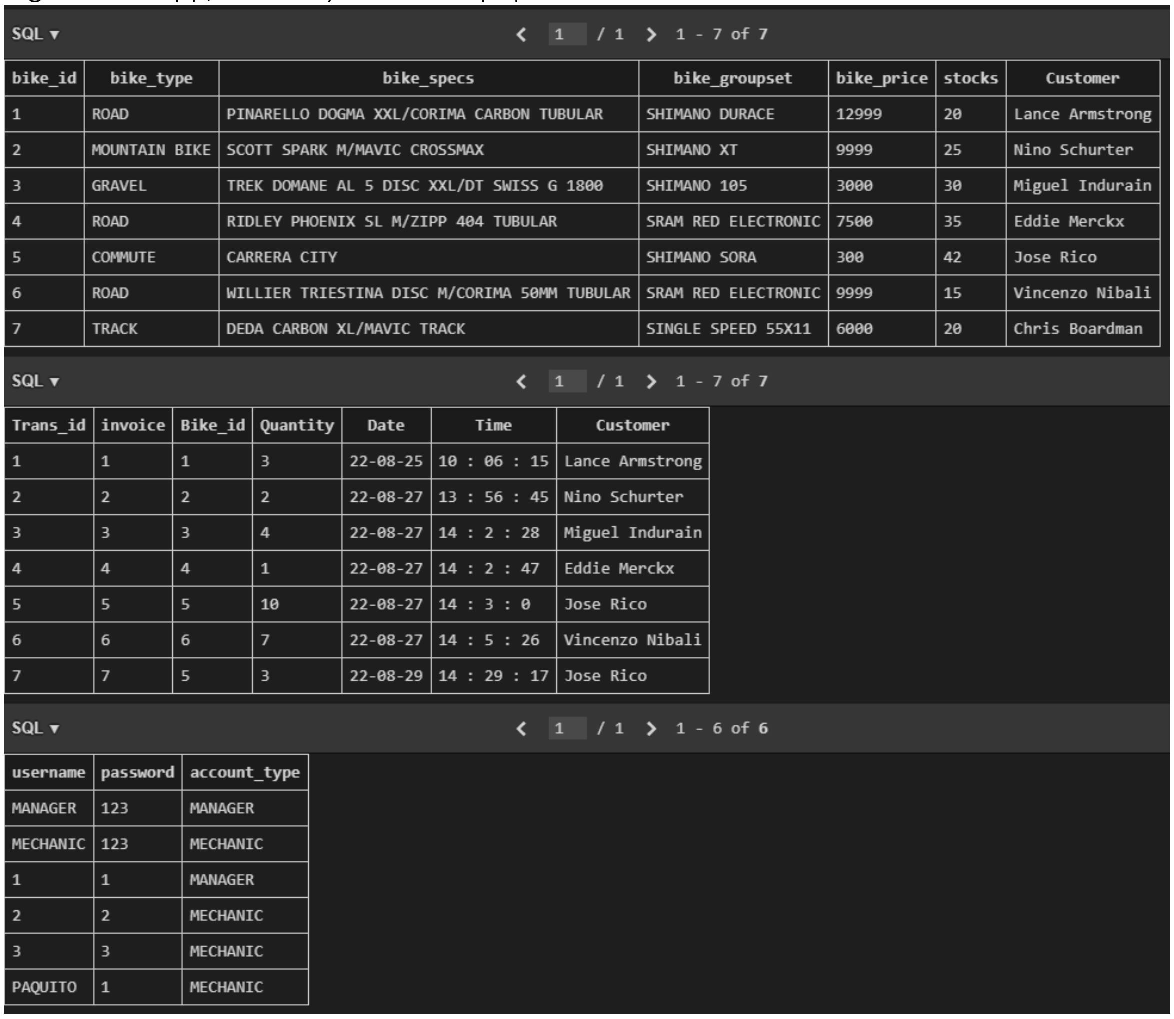
Remaining table users is created in script Loginscreen.py method number three, this method is crucial because is where we connect the database, as seen in figure below:



```
    # LOGIN TABLE
    def logintable(self):#3
        self.base = sqlite3.connect("cool_bikes_db.db")
        self.cur = self.base.cursor()
        self.cur.execute("CREATE TABLE if not exists users ( username varchar (20),password  varchar (20)
```

Fig.5: users table creation and database connection.

There is an important aspect to consider, after we create these three tables just by running app.py, there are no records in them therefore we will not be able to log into the app, currently tables are populated as it follows:



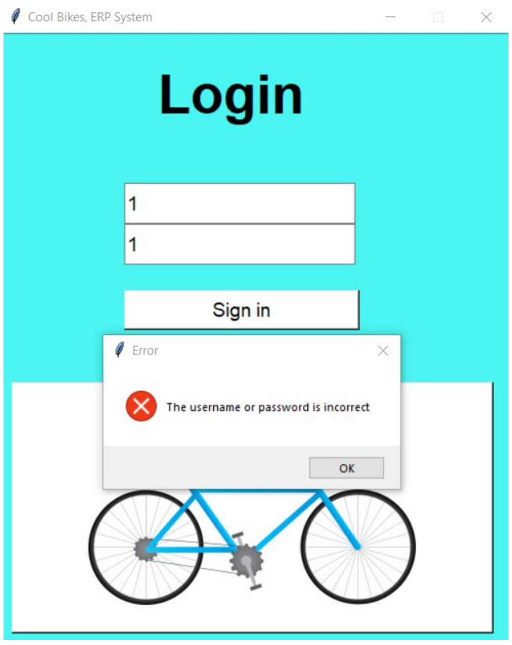
bike_id	bike_type	bike_specs	bike_groupset	bike_price	stocks	Customer
1	ROAD	PINARELLO DOGMA XXL/CORIMA CARBON TUBULAR	SHIMANO DURACE	12999	20	Lance Armstrong
2	MOUNTAIN BIKE	SCOTT SPARK M/MAVIC CROSSMAX	SHIMANO XT	9999	25	Nino Schurter
3	GRAVEL	TREK DOMANE AL 5 DISC XXL/DT SWISS G 1800	SHIMANO 105	3000	30	Miguel Indurain
4	ROAD	RIDLEY PHOENIX SL M/ZIPP 404 TUBULAR	SRAM RED ELECTRONIC	7500	35	Eddie Merckx
5	COMMUTE	CARRERA CITY	SHIMANO SORA	300	42	Jose Rico
6	ROAD	WILLIER TRIESTINA DISC M/CORIMA 50MM TUBULAR	SRAM RED ELECTRONIC	9999	15	Vincenzo Nibali
7	TRACK	DEDA CARBON XL/MAVIC TRACK	SINGLE SPEED 55X11	6000	20	Chris Boardman

Trans_id	invoice	Bike_id	Quantity	Date	Time	Customer
1	1	1	3	22-08-25	10 : 06 : 15	Lance Armstrong
2	2	2	2	22-08-27	13 : 56 : 45	Nino Schurter
3	3	3	4	22-08-27	14 : 2 : 28	Miguel Indurain
4	4	4	1	22-08-27	14 : 2 : 47	Eddie Merckx
5	5	5	10	22-08-27	14 : 3 : 0	Jose Rico
6	6	6	7	22-08-27	14 : 5 : 26	Vincenzo Nibali
7	7	5	3	22-08-29	14 : 29 : 17	Jose Rico

username	password	account_type
MANAGER	123	MANAGER
MECHANIC	123	MECHANIC
1	1	MANAGER
2	2	MECHANIC
3	3	MECHANIC
PAQUITO	1	MECHANIC

Fig.6: records in tables bikes, sales and users.

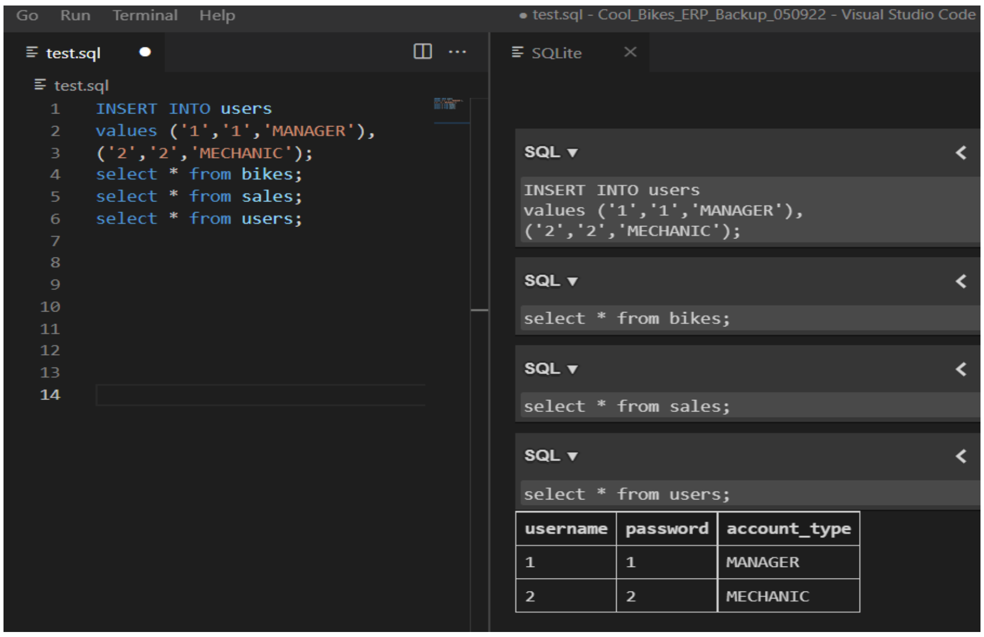
Now let us get at the primary point when there is no data inserted in any of the tables, I will delete all records from the three tables.
First table we must tackle is users, after login screen is prompted it will be impossible to log in, since user table is empty:



The username or password is incorrect

Fig.7: Impossible to log in, users table empty.

To amend this, we must insert a record in user table. I have benefited from extension called *"SQLite"* in Visual Studio Code, to manipulate the python database, otherwise it would have been impossible to complete this project.

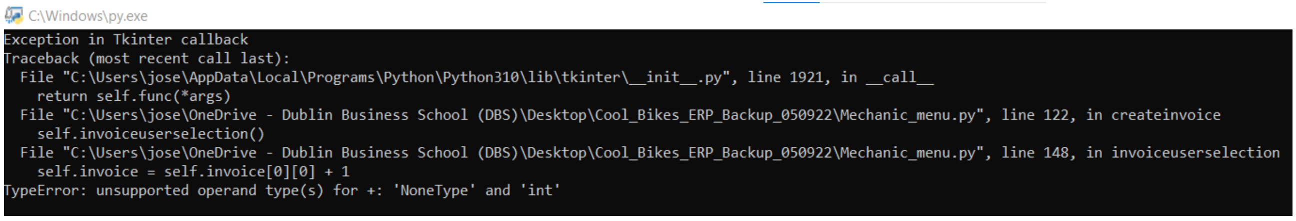

Go
Run
Terminal
Help
● test.sql - Cool_Bikes_ERP_Backup_050922 - Visual Studio Code
test.sql
SQLite
SQL ▾

```
INSERT INTO users
values ('1','1','MANAGER'),
('2','2','MECHANIC');
select * from bikes;
select * from sales;
select * from users;
```

username	password	account_type
1	1	MANAGER
2	2	MECHANIC

Fig.8: Inserting users.

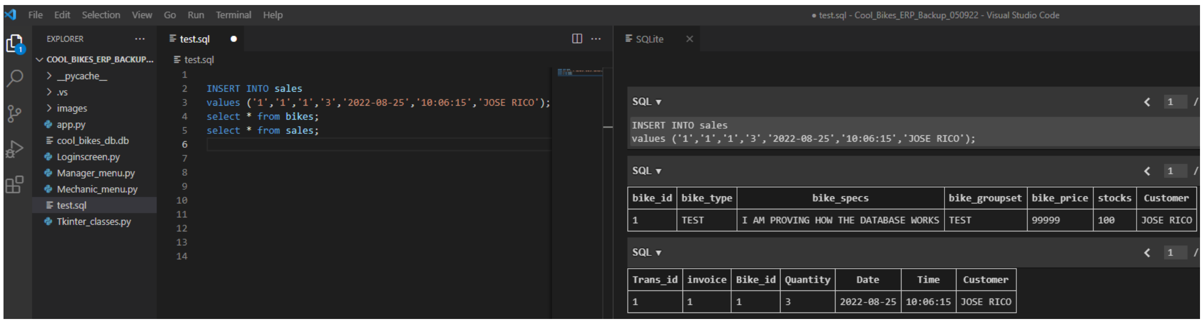
Once we log in, I can start populating bikes table. In contrast in mechanic menu, it is impossible to perform a sale, as soon as we hit cart button, the app throws the following error:



```
C:\Windows\py.exe
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\jose\AppData\Local\Programs\Python\Python310\lib\tkinter\__init__.py", line 1921, in __call__
    return self.func(*args)
  File "C:\Users\jose\OneDrive - Dublin Business School (DBS)\Desktop\Cool_Bikes_ERP_Backup_050922\Mechanic_menu.py", line 122, in createinvoice
    self.invoiceuserselection()
  File "C:\Users\jose\OneDrive - Dublin Business School (DBS)\Desktop\Cool_Bikes_ERP_Backup_050922\Mechanic_menu.py", line 148, in invoiceuserselection
    self.invoice = self.invoice[0][0] + 1
TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
```

Fig.9: Error Empty sales table.

This happens because method six in script Mechanic_menu.py is trying to retrieve an invoice and there is none, let us say that it needs to retrieve an *"int"* and it is fetching a *"null"* value that makes the app crash. Starts in line 148 and escalates to method five *"createinvoce".* Thankfully this has an easy fix, same as we did with users an insertion needs to be done in table sales using Visual Studio Code:



```
INSERT INTO sales
values ('1','1','1','3','2022-08-25','10:06:15','JOSE RICO');
select * from bikes;
select * from sales;
```

```
INSERT INTO sales
values ('1','1','1','3','2022-08-25','10:06:15','JOSE RICO');
```

bike_id	bike_type	bike_specs	bike_groupset	bike_price	stocks	Customer
1	TEST	I AM PROVING HOW THE DATABASE WORKS	TEST	99999	100	JOSE RICO

Trans_id	invoice	Bike_id	Quantity	Date	Time	Customer
1	1	1	3	2022-08-25	10:06:15	JOSE RICO

Fig.10: Inserting first invoice.

After considerations stated in this section, we are good to fully use the database interacting just with the app GUI.

3.5 Tools

To complete this project, I used the following tools:

- Microsoft Visual Studio.
- Visual Studio Code.
- Lucidchard.

4. Implementation

All features of this artefact have been developed and implemented using python version 3.10.5. From what I know, and I have been using python, this app will have a long life, due to the stability of this programming language.
Next let us focus on how the app works, how have been deployed and what exactly each script does:

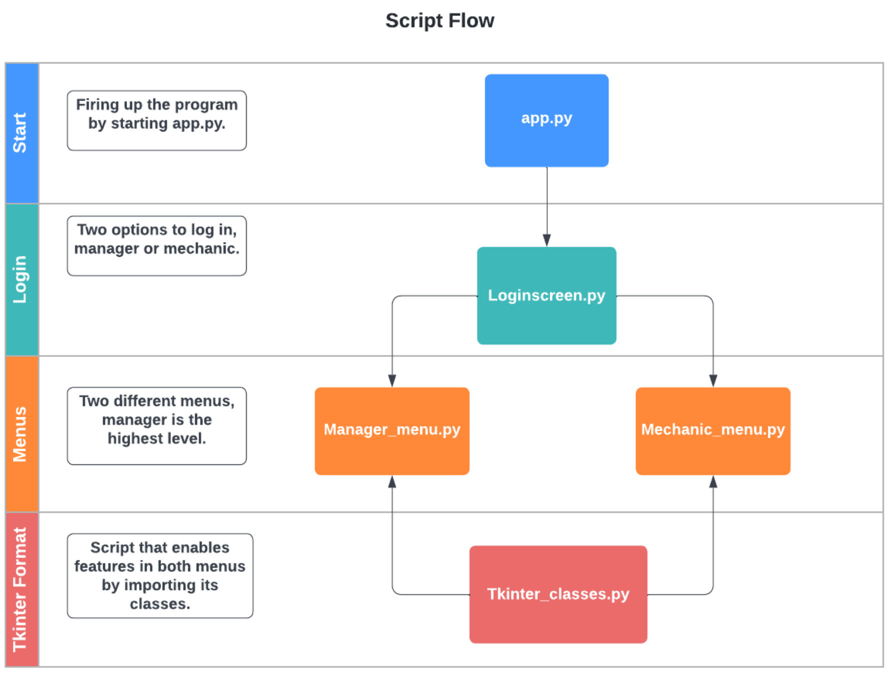

Script Flow
Start
Login
Menus
Tkinter Format
Firing up the program by starting app.py.
Two options to log in, manager or mechanic.
Two different menus, manager is the highest level.
Script that enables features in both menus by importing its classes.
app.py
Loginscreen.py
Manager_menu.py
Mechanic_menu.py
Tkinter_classes.py

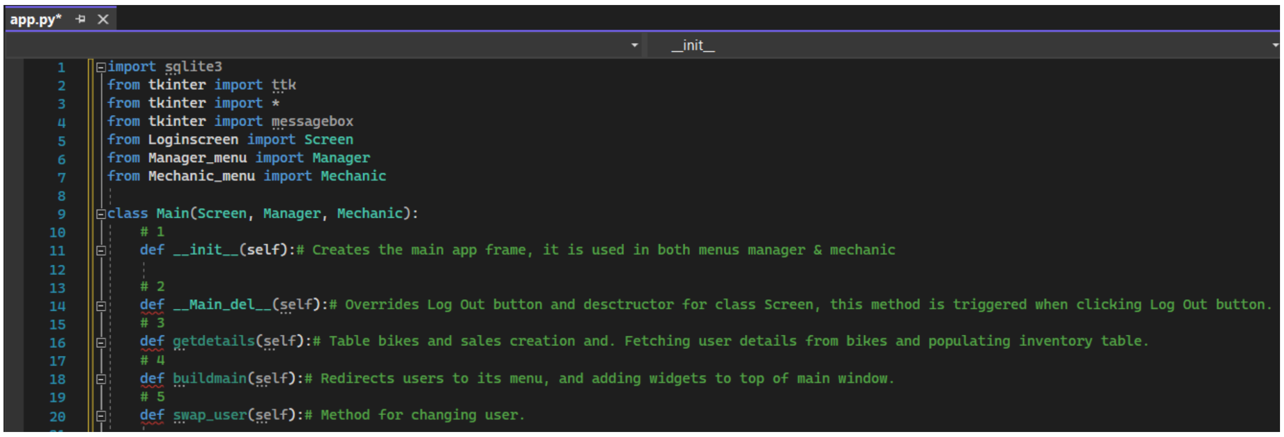
Fig.11: Script flow process.

Figure shown above may be trivial, but I think is necessary to understand the app logic, in the following section I am going to break down each script, to see what each class method script does.

4.1. app.py

The core script that fires up the process, here we have a class and inside it five methods. Note that features build up through this script are shared in both menus:

1. First method creates the main app frame, that it is used in both menus.
2. Second method is used to log out from the app, every time we will get a pop-up box and this message *"Do you want to Exit Cool Bikes Management System?"*.
3. Third, table creation for bikes and sales as well as fetching bikes details and populating inventory table.
4. Fourth, redirects user to its menu and widget addition to top of main window.
5. Method to change a user, when actioned a pop-up box will appear asking *"Do you want to change user?"*.



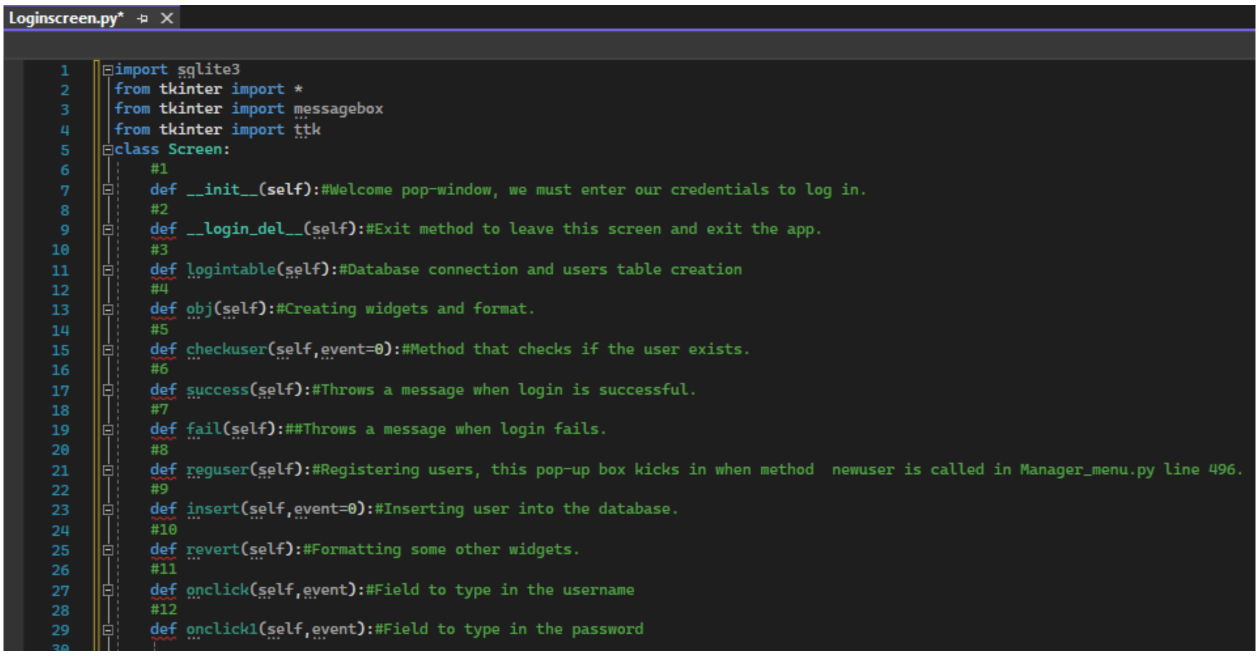
```python
import sqlite3
from tkinter import ttk
from tkinter import *
from tkinter import messagebox
from Loginscreen import Screen
from Manager_menu import Manager
from Mechanic_menu import Mechanic

class Main(Screen, Manager, Mechanic):
    # 1
    def __init__(self):# Creates the main app frame, it is used in both menus manager & mechanic

    # 2
    def __Main_del__(self):# Overrides Log Out button and desctructor for class Screen, this method is triggered when clicking Log Out button.
    # 3
    def getdetails(self):# Table bikes and sales creation and. Fetching user details from bikes and populating inventory table.
    # 4
    def buildmain(self):# Redirects users to its menu, and adding widgets to top of main window.
    # 5
    def swap_user(self):# Method for changing user.
```

Fig.12: Simplified code showing the skeleton of app.py.

4.2. Loginscreen.py

Main door to access to one of the menus depending on our credentials, class Screen is created all together with twelve methods:

1. First method creates a pop-up welcome window, we must enter our credentials to log in.
2. Exit method to leave this screen and exit the app.
3. Database connection and user table creation.
4. Creating widgets and formatting.
5. Method that checks if the user exists.
6. Throws a message when login is successful.
7. Throws a message when login fails.
8. Registering users, this pop-up box kicks in when this method "*newuser*" is called from Manager_menu.py line 496. We only create users in that level.
9. Inserting users into users table.
10. Formatting some other widgets.
11. Field to type in the username.
12. Field to type in the password.



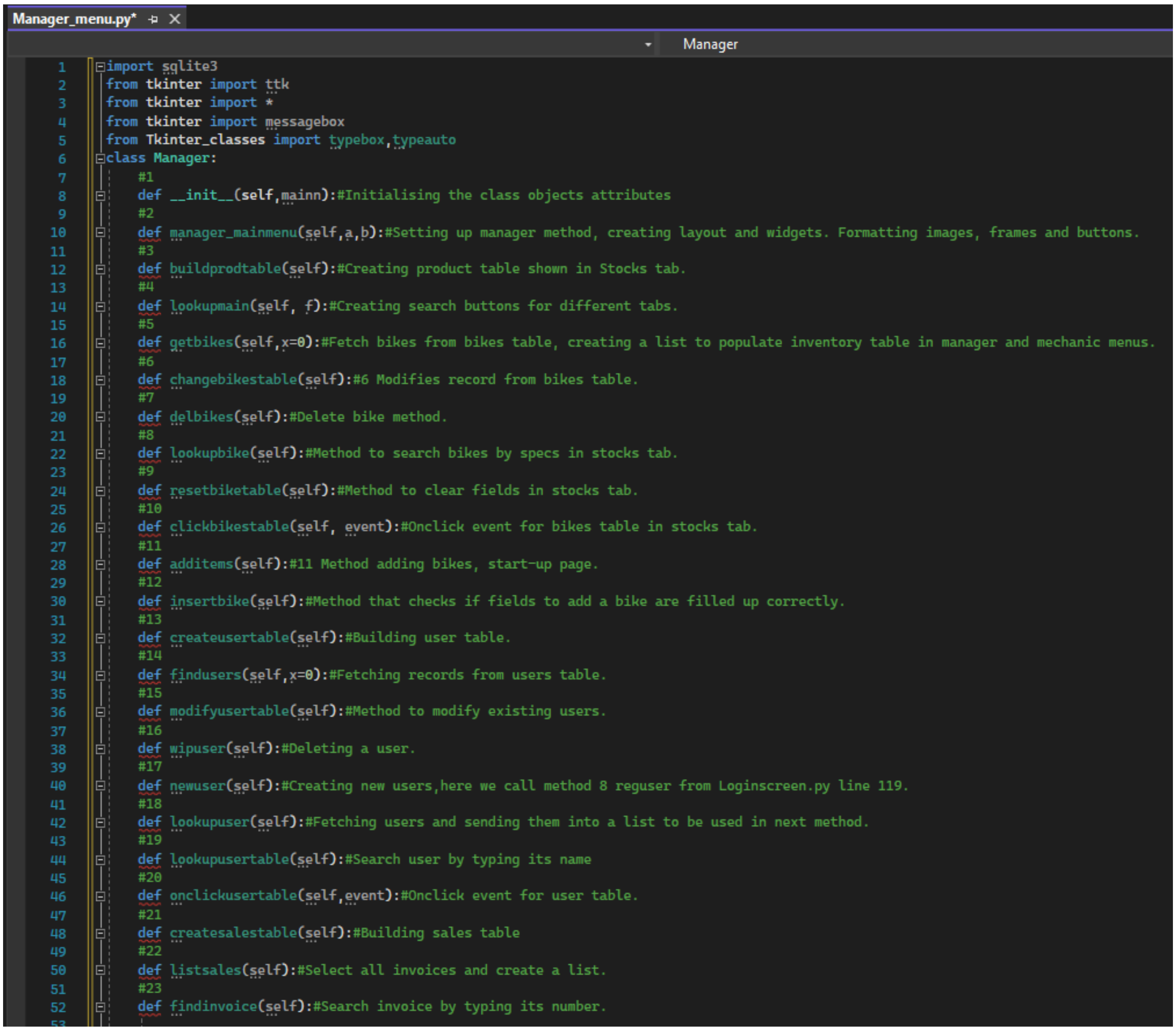
```python
import sqlite3
from tkinter import *
from tkinter import messagebox
from tkinter import ttk
class Screen:
    #1
    def __init__(self):#Welcome pop-window, we must enter our credentials to log in.
    #2
    def __login_del__(self):#Exit method to leave this screen and exit the app.
    #3
    def logintable(self):#Database connection and users table creation
    #4
    def obj(self):#Creating widgets and format.
    #5
    def checkuser(self,event=0):#Method that checks if the user exists.
    #6
    def success(self):#Throws a message when login is successful.
    #7
    def fail(self):##Throws a message when login fails.
    #8
    def reguser(self):#Registering users, this pop-up box kicks in when method  newuser is called in Manager_menu.py line 496.
    #9
    def insert(self,event=0):#Inserting user into the database.
    #10
    def revert(self):#Formatting some other widgets.
    #11
    def onclick(self,event):#Field to type in the username
    #12
    def onclick1(self,event):#Field to type in the password
```

Fig.13: Simplified code for Loginscreen.py.

4.3. Manager_menu.py

Higher menu for a user, it is composed of one class *"Manager"* and belonging to it twenty-three methods:

1. First method initiliases the class objects and attributes.
2. Setting up manager method, creating layout and widgets. Formatting images, frames and buttons.
3. Creating product table shown in stocks tab.
4. Creating search buttons for different tabs.
5. Fetch bikes from bikes table, creating a list to populate table in manager and mechanic menus.
6. Modifying records from bikes table.
7. Delete bike method.
8. Method to search bikes by specs in stocks tab.
9. Method to clear fields in stocks tab.
10. Onclick event for bikes table in stocks tab.
11. Method adding bikes, start-up page.
12. Checking if fields to add a bike are correctly filled up.
13. Building user table.
14. Fetching records from users table.
15. Modifying existing users.
16. Deleting a user.
17. Creating new users, here we call method 8 *"reguser"* from Loginscreen.py line 119.
18. Fetching users and sending them into a list to be used in the next method.
19. Search user by typing its name.
20. Onclick event for user table.
21. Building up sales table.
22. Select all invoices and a create a list.
23. Search invoice by typing its number



```python
import sqlite3
from tkinter import ttk
from tkinter import *
from tkinter import messagebox
from Tkinter_classes import typebox,typeauto
class Manager:
    #1
    def __init__(self,mainn):#Initialising the class objects attributes
    #2
    def manager_mainmenu(self,a,b):#Setting up manager method, creating layout and widgets. Formatting images, frames and buttons.
    #3
    def buildprodtable(self):#Creating product table shown in Stocks tab.
    #4
    def lookupmain(self, f):#Creating search buttons for different tabs.
    #5
    def getbikes(self,x=0):#Fetch bikes from bikes table, creating a list to populate inventory table in manager and mechanic menus.
    #6
    def changebikestable(self):#6 Modifies record from bikes table.
    #7
    def delbikes(self):#Delete bike method.
    #8
    def lookupbike(self):#Method to search bikes by specs in stocks tab.
    #9
    def resetbiketable(self):#Method to clear fields in stocks tab.
    #10
    def clickbikestable(self, event):#Onclick event for bikes table in stocks tab.
    #11
    def additems(self):#11 Method adding bikes, start-up page.
    #12
    def insertbike(self):#Method that checks if fields to add a bike are filled up correctly.
    #13
    def createusertable(self):#Building user table.
    #14
    def findusers(self,x=0):#Fetching records from users table.
    #15
    def modifyusertable(self):#Method to modify existing users.
    #16
    def wipuser(self):#Deleting a user.
    #17
    def newuser(self):#Creating new users,here we call method 8 reguser from Loginscreen.py line 119.
    #18
    def lookupuser(self):#Fetching users and sending them into a list to be used in next method.
    #19
    def lookupusertable(self):#Search user by typing its name
    #20
    def onclickusertable(self,event):#Onclick event for user table.
    #21
    def createsalestable(self):#Building sales table
    #22
    def listsales(self):#Select all invoices and create a list.
    #23
    def findinvoice(self):#Search invoice by typing its number.
```

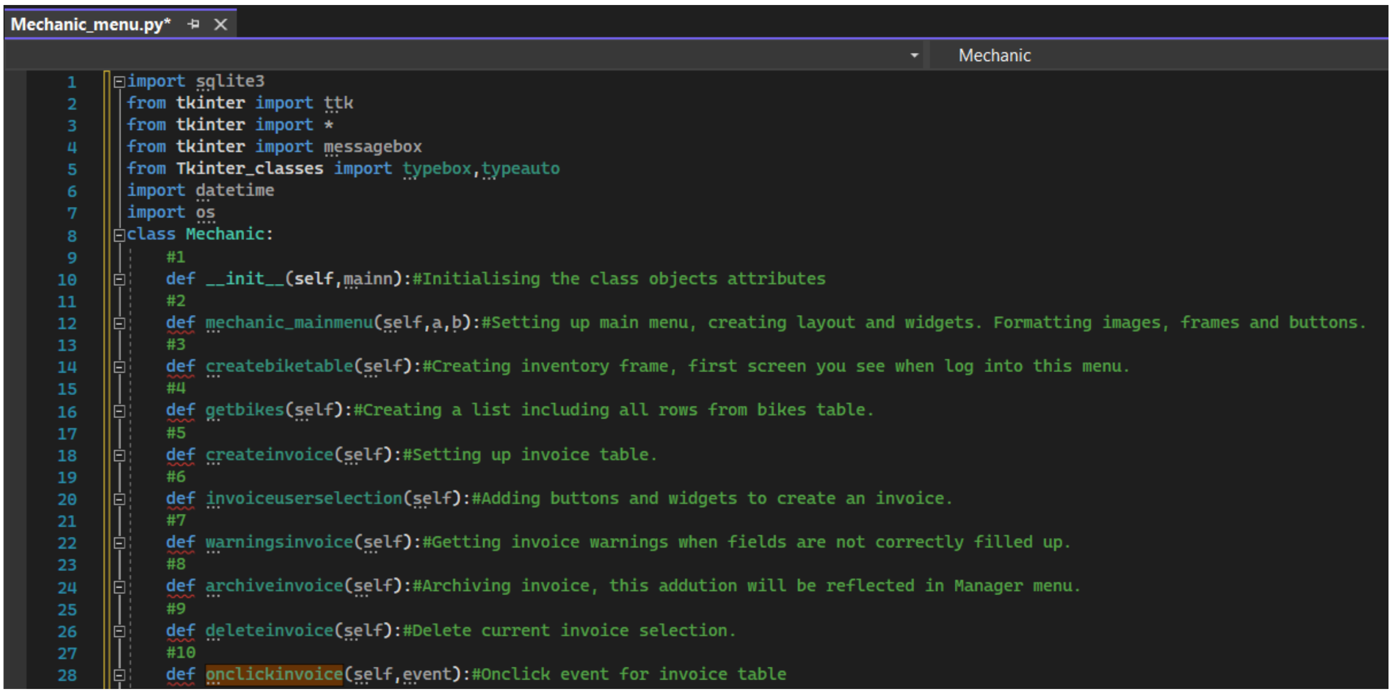
Fig.14: Simplified code for Manager_menu.py.

4.4. Mechanic_menu.py

Second tier menu for a user, it is composed of one class *"Mechanic"* and belonging to it ten methods:

1. First method initiliases the class objects and attributes.
2. Second, setting up manin menu, creating layout and widgets. Formatting images, frames and buttons.
3. Creating inventory frame, first screen you see when log into this menu.
4. Creating a list including all rows from bikes table.
5. Setting up invoice table.
6. Adding buttons and widgets to create an invoice.
7. Getting invoice warnings when fields are not correctly filled up.
8. Archiving invoice, this addition will be reflected in Manager menu.
9. Delete current invoice selection.

10. Onclick event for invoice table.



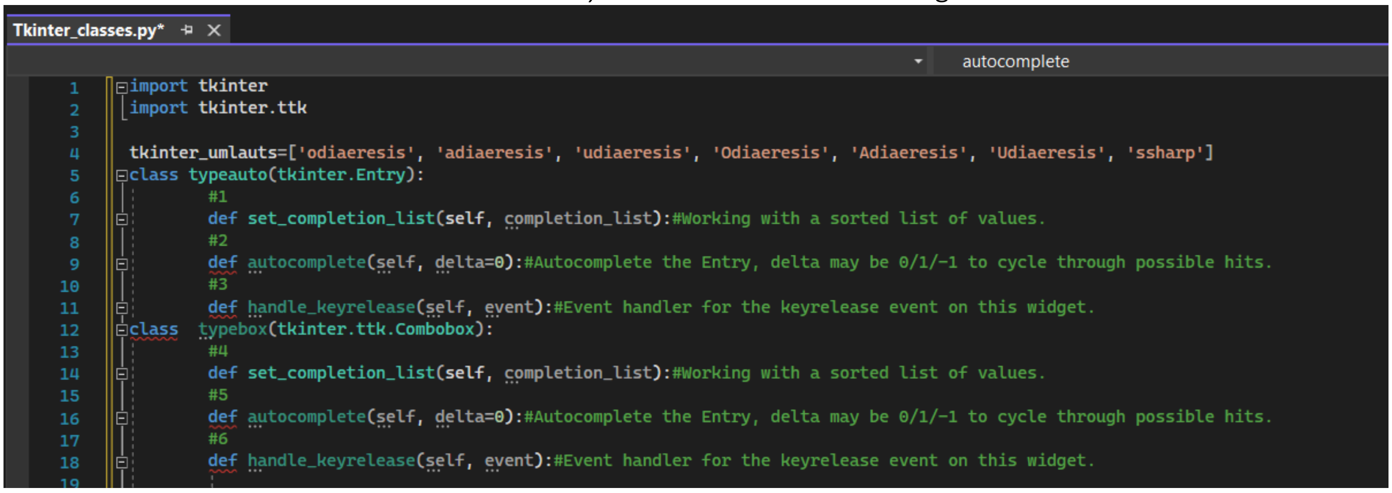
```python
import sqlite3
from tkinter import ttk
from tkinter import *
from tkinter import messagebox
from Tkinter_classes import typebox,typeauto
import datetime
import os
class Mechanic:
    #1
    def __init__(self,mainn):#Initialising the class objects attributes
    #2
    def mechanic_mainmenu(self,a,b):#Setting up main menu, creating layout and widgets. Formatting images, frames and buttons.
    #3
    def createbiketable(self):#Creating inventory frame, first screen you see when log into this menu.
    #4
    def getbikes(self):#Creating a list including all rows from bikes table.
    #5
    def createinvoice(self):#Setting up invoice table.
    #6
    def invoiceuserselection(self):#Adding buttons and widgets to create an invoice.
    #7
    def warningsinvoice(self):#Getting invoice warnings when fields are not correctly filled up.
    #8
    def archiveinvoice(self):#Archiving invoice, this addution will be reflected in Manager menu.
    #9
    def deleteinvoice(self):#Delete current invoice selection.
    #10
    def onclickinvoice(self,event):#Onclick event for invoice table
```

Fig.15: Abbreviated code for Mechanic_menu.py.

4.5. Tkinter_classes.py

Last script allows autocompletion when typing text in or shows a drop-down list when clicking a predefined arrow. Two classes and three methods belonging to each class. Methods are nearly identical from one class to another:

1. Fist method, working with a sorted list of values.
2. Autocomplete the entry, delta may be 0/1/-1 to cycle through possible hits.
3. Event handler for the *"keyrelease"* event on this widget.



```python
import tkinter
import tkinter.ttk

tkinter_umlauts=['odiaeresis', 'adiaeresis', 'udiaeresis', 'Odiaeresis', 'Adiaeresis', 'Udiaeresis', 'ssharp']
class typeauto(tkinter.Entry):
        #1
        def set_completion_list(self, completion_list):#Working with a sorted list of values.
        #2
        def autocomplete(self, delta=0):#Autocomplete the Entry, delta may be 0/1/-1 to cycle through possible hits.
        #3
        def handle_keyrelease(self, event):#Event handler for the keyrelease event on this widget.
class  typebox(tkinter.ttk.Combobox):
        #4
        def set_completion_list(self, completion_list):#Working with a sorted list of values.
        #5
        def autocomplete(self, delta=0):#Autocomplete the Entry, delta may be 0/1/-1 to cycle through possible hits.
        #6
        def handle_keyrelease(self, event):#Event handler for the keyrelease event on this widget.
```

Fig.16: Abbreviated code for Tkinter_classes.py.

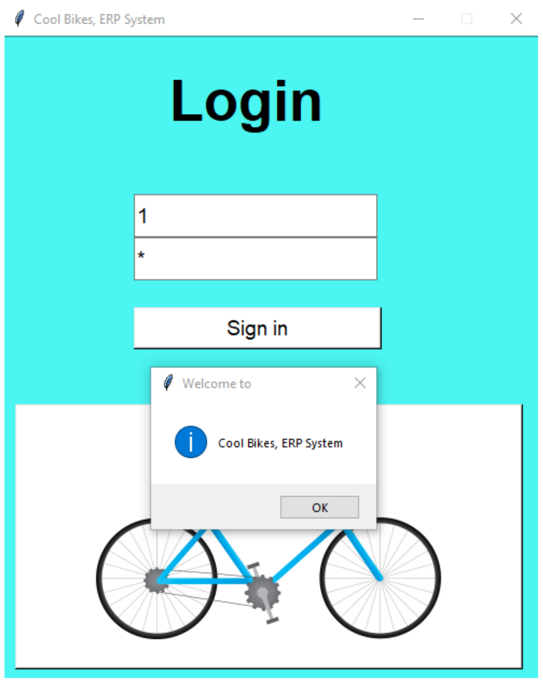
5.Testing and Results

Behavioral testing has been used to test app features. The several tests performed cascade down from the requirements that were stated in section 3.2. These requirements showcase the need of the app performing in a certain way. I have built up a table divided in three sections, *"Login"*, *"Manager menu"* and *"Mechanic menu"*, each section contains a series of tests.

	Number	Test	Successful (Yes/No)
Login	1	Login for both users.	YES
Manager	2	Add a bike.	YES
	3	Modify bike and delete.	
	4	Visualise and search for an invoice.	YES
	5	Add, modify, delete and search for a user.	YES
	6	Change user.	YES
	7	Log out.	YES
Mechanic	8	Visualise inventory	YES
	9	Perform a sale	YES
	10	Change user.	YES
	11	Log out.	YES

Test 1:

Action: Login in as a Manager and Employee, run app.py and a login screen will appear. Credentials for manager are 1 for username and password and same logic for mechanic but using 2.

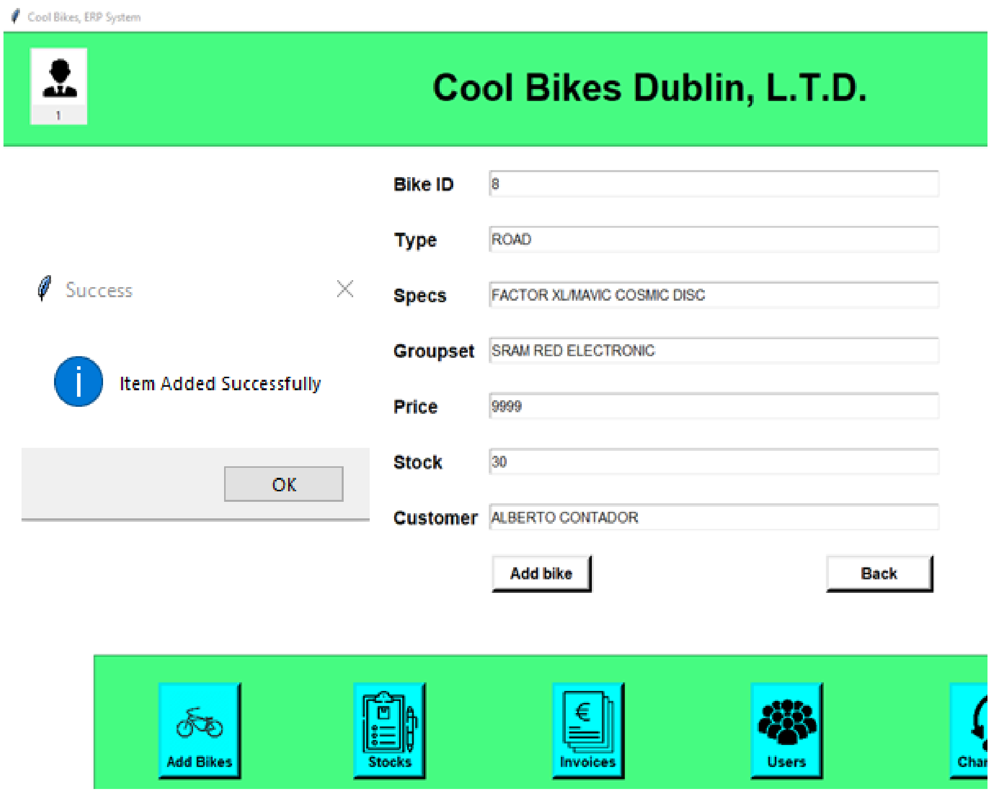


Cool Bikes, ERP System

Fig.17: Login Screen

<u>Test 2:</u>

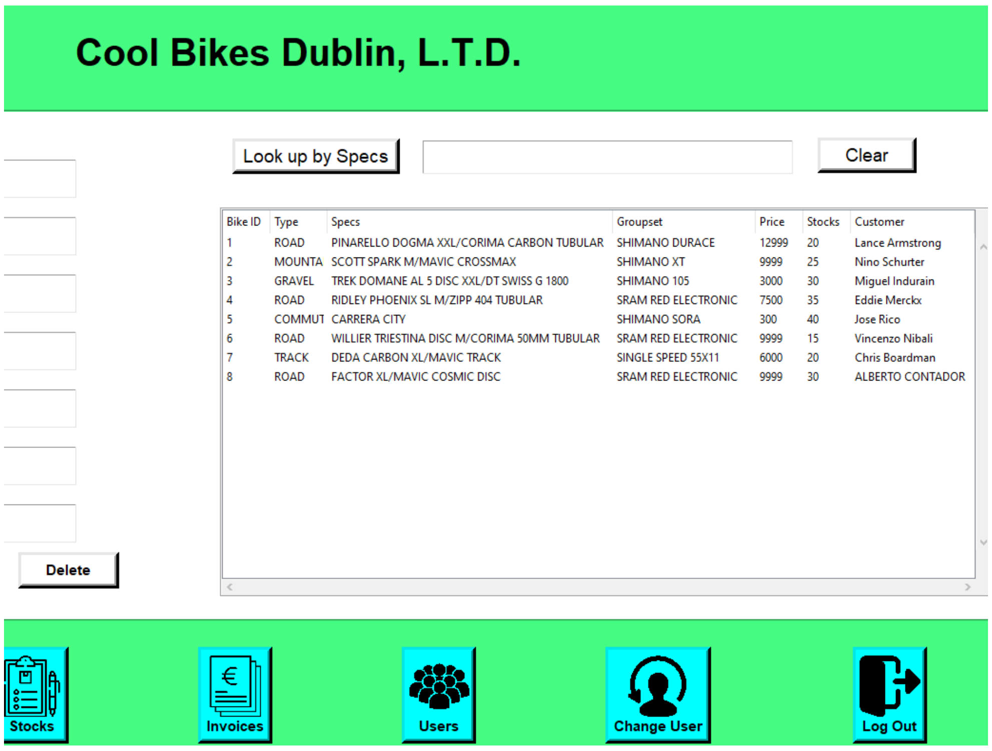
Action: Adding a bike to the inventory:



Bike ID	8
Type	ROAD
Specs	FACTOR XL/MAVIC COSMIC DISC
Groupset	SRAM RED ELECTRONIC
Price	9999
Stock	30
Customer	ALBERTO CONTADOR

Fig.18: Adding a Bike.

Once the bike is inserted it will appear in inventory:

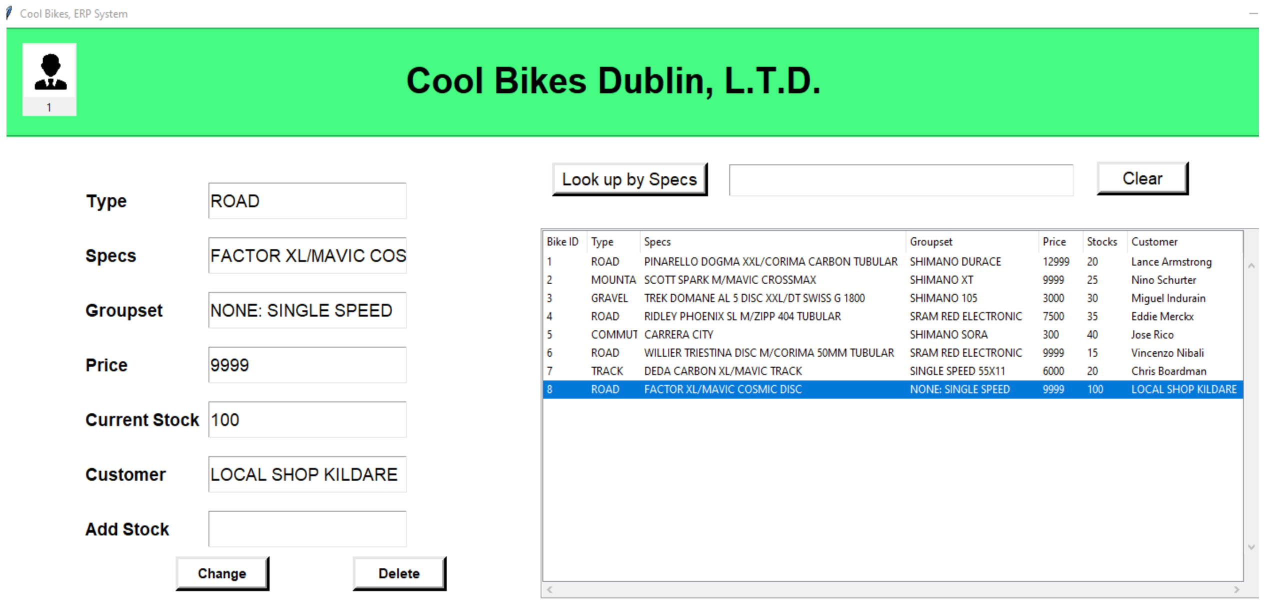


Bike ID	Type	Specs	Groupset	Price	Stocks	Customer
1	ROAD	PINARELLO DOGMA XXL/CORIMA CARBON TUBULAR	SHIMANO DURACE	12999	20	Lance Armstrong
2	MOUNTA	SCOTT SPARK M/MAVIC CROSSMAX	SHIMANO XT	9999	25	Nino Schurter
3	GRAVEL	TREK DOMANE AL 5 DISC XXL/DT SWISS G 1800	SHIMANO 105	3000	30	Miguel Indurain
4	ROAD	RIDLEY PHOENIX SL M/ZIPP 404 TUBULAR	SRAM RED ELECTRONIC	7500	35	Eddie Merckx
5	COMMUT	CARRERA CITY	SHIMANO SORA	300	40	Jose Rico
6	ROAD	WILLIER TRIESTINA DISC M/CORIMA 50MM TUBULAR	SRAM RED ELECTRONIC	9999	15	Vincenzo Nibali
7	TRACK	DEDA CARBON XL/MAVIC TRACK	SINGLE SPEED 55X11	6000	20	Chris Boardman
8	ROAD	FACTOR XL/MAVIC COSMIC DISC	SRAM RED ELECTRONIC	9999	30	ALBERTO CONTADOR

Fig.19: Bike ID 8 recently inserted.

Test 3:

Action: Modifying bike Id 8, we are going to add stock, change customer number and amend the *"groupset"*.


Cool Bikes, ERP System

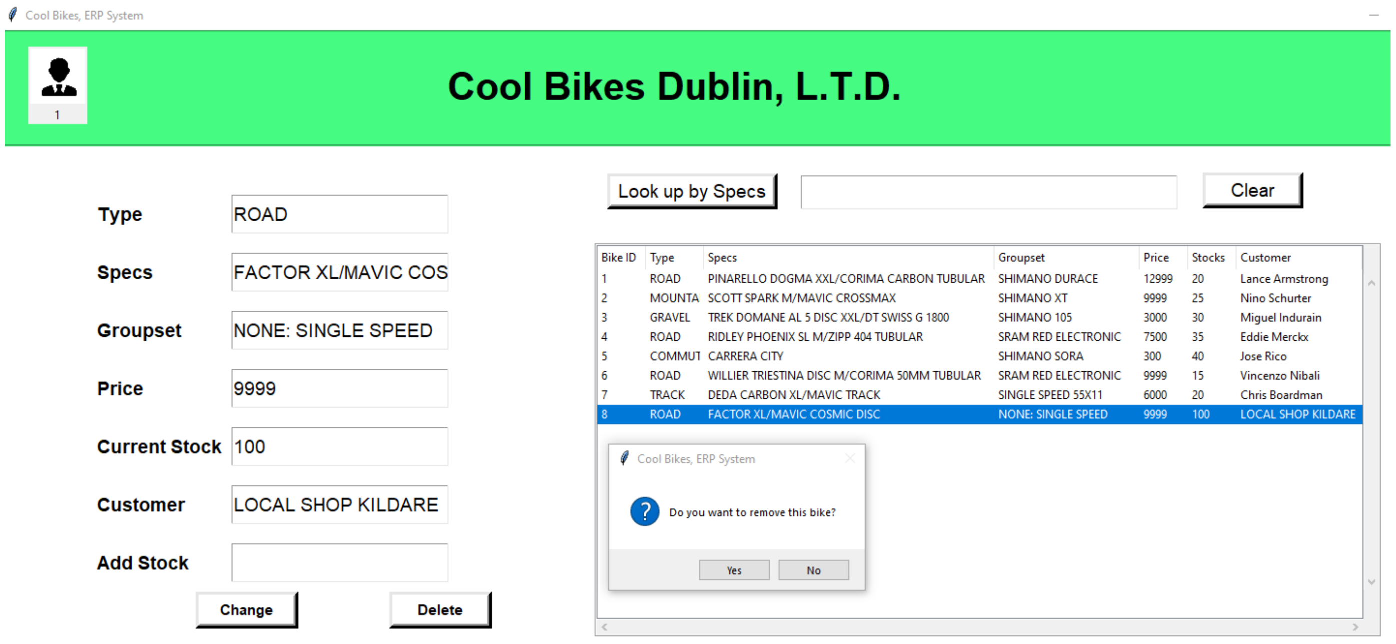
Cool Bikes Dublin, L.T.D.

Type	ROAD
Specs	FACTOR XL/MAVIC COS
Groupset	NONE: SINGLE SPEED
Price	9999
Current Stock	100
Customer	LOCAL SHOP KILDARE
Add Stock	

Bike ID	Type	Specs	Groupset	Price	Stocks	Customer
1	ROAD	PINARELLO DOGMA XXL/CORIMA CARBON TUBULAR	SHIMANO DURACE	12999	20	Lance Armstrong
2	MOUNTA	SCOTT SPARK M/MAVIC CROSSMAX	SHIMANO XT	9999	25	Nino Schurter
3	GRAVEL	TREK DOMANE AL 5 DISC XXL/DT SWISS G 1800	SHIMANO 105	3000	30	Miguel Indurain
4	ROAD	RIDLEY PHOENIX SL M/ZIPP 404 TUBULAR	SRAM RED ELECTRONIC	7500	35	Eddie Merckx
5	COMMUT	CARRERA CITY	SHIMANO SORA	300	40	Jose Rico
6	ROAD	WILLIER TRIESTINA DISC M/CORIMA 50MM TUBULAR	SRAM RED ELECTRONIC	9999	15	Vincenzo Nibali
7	TRACK	DEDA CARBON XL/MAVIC TRACK	SINGLE SPEED 55X11	6000	20	Chris Boardman
8	ROAD	FACTOR XL/MAVIC COSMIC DISC	NONE: SINGLE SPEED	9999	100	LOCAL SHOP KILDARE

Fig.20: Making changes to Bike Id 8.

Action: Deleting Bike id 8, when pop-up box appears by hitting yes it will delete a record.


Cool Bikes, ERP System

Cool Bikes Dublin, L.T.D.

Type	ROAD
Specs	FACTOR XL/MAVIC COS
Groupset	NONE: SINGLE SPEED
Price	9999
Current Stock	100
Customer	LOCAL SHOP KILDARE
Add Stock	

Bike ID	Type	Specs	Groupset	Price	Stocks	Customer
1	ROAD	PINARELLO DOGMA XXL/CORIMA CARBON TUBULAR	SHIMANO DURACE	12999	20	Lance Armstrong
2	MOUNTA	SCOTT SPARK M/MAVIC CROSSMAX	SHIMANO XT	9999	25	Nino Schurter
3	GRAVEL	TREK DOMANE AL 5 DISC XXL/DT SWISS G 1800	SHIMANO 105	3000	30	Miguel Indurain
4	ROAD	RIDLEY PHOENIX SL M/ZIPP 404 TUBULAR	SRAM RED ELECTRONIC	7500	35	Eddie Merckx
5	COMMUT	CARRERA CITY	SHIMANO SORA	300	40	Jose Rico
6	ROAD	WILLIER TRIESTINA DISC M/CORIMA 50MM TUBULAR	SRAM RED ELECTRONIC	9999	15	Vincenzo Nibali
7	TRACK	DEDA CARBON XL/MAVIC TRACK	SINGLE SPEED 55X11	6000	20	Chris Boardman
8	ROAD	FACTOR XL/MAVIC COSMIC DISC	NONE: SINGLE SPEED	9999	100	LOCAL SHOP KILDARE

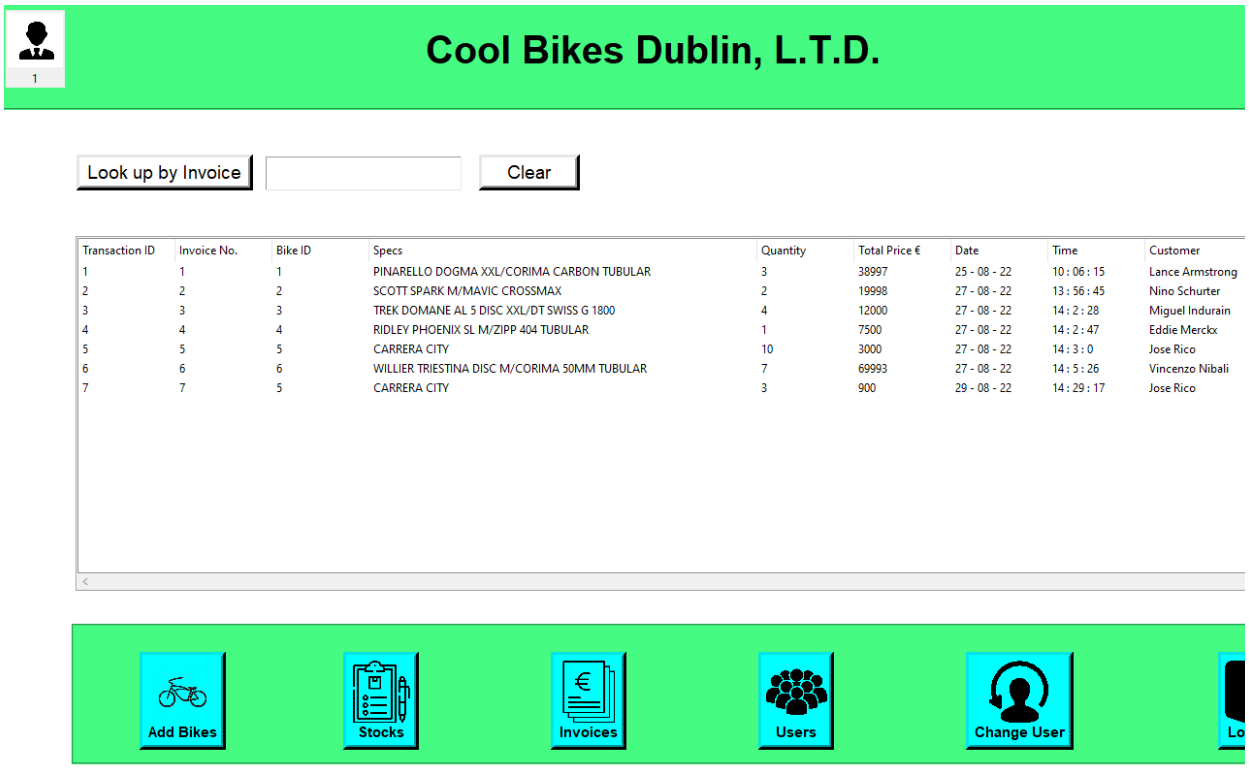
Cool Bikes, ERP System

Do you want to remove this bike?

Fig.21: Deleting Bike Id 8.

Test 4:

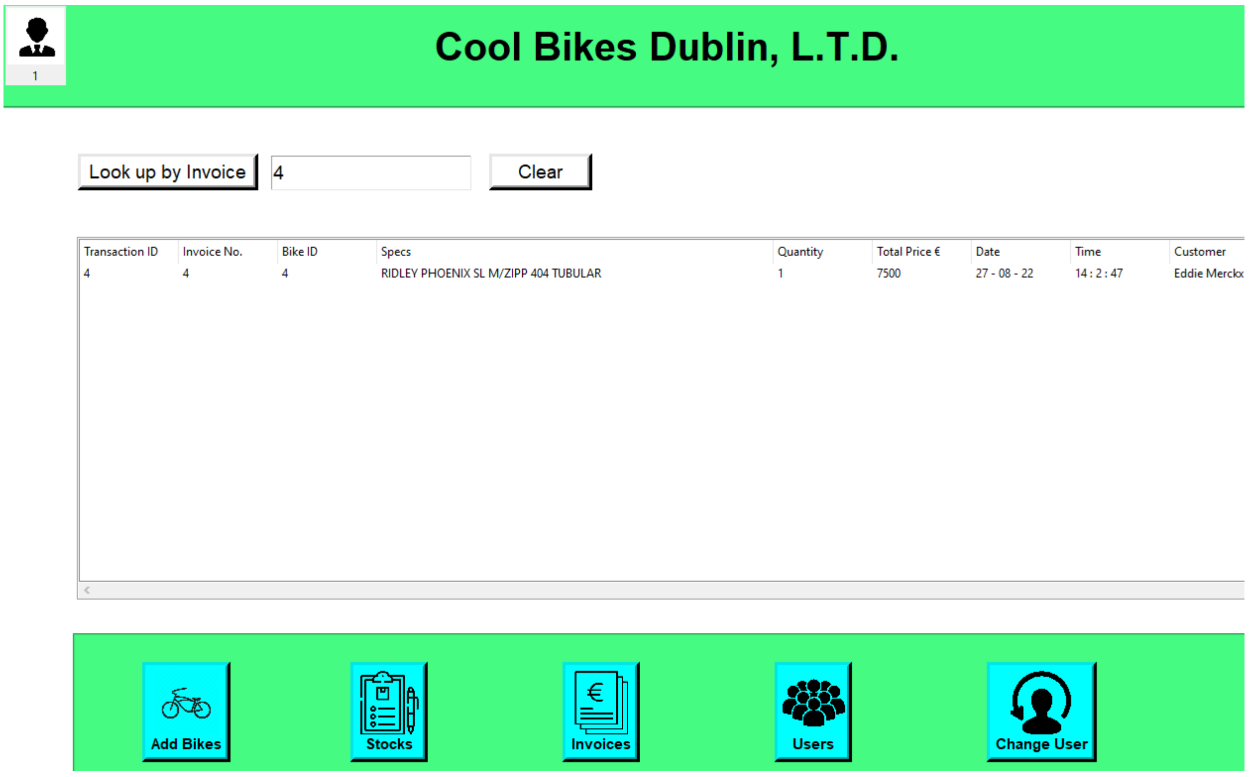
Action: Visualising all invoices.



Transaction ID	Invoice No.	Bike ID	Specs	Quantity	Total Price €	Date	Time	Customer
1	1	1	PINARELLO DOGMA XXL/CORIMA CARBON TUBULAR	3	38997	25 - 08 - 22	10 : 06 : 15	Lance Armstrong
2	2	2	SCOTT SPARK M/MAVIC CROSSMAX	2	19998	27 - 08 - 22	13 : 56 : 45	Nino Schurter
3	3	3	TREK DOMANE AL 5 DISC XXL/DT SWISS G 1800	4	12000	27 - 08 - 22	14 : 2 : 28	Miguel Indurain
4	4	4	RIDLEY PHOENIX SL M/ZIPP 404 TUBULAR	1	7500	27 - 08 - 22	14 : 2 : 47	Eddie Merckx
5	5	5	CARRERA CITY	10	3000	27 - 08 - 22	14 : 3 : 0	Jose Rico
6	6	6	WILLIER TRIESTINA DISC M/CORIMA 50MM TUBULAR	7	69993	27 - 08 - 22	14 : 5 : 26	Vincenzo Nibali
7	7	5	CARRERA CITY	3	900	29 - 08 - 22	14 : 29 : 17	Jose Rico

Fig.22: Sales visual.

Action: Searching for invoice 4.



Transaction ID	Invoice No.	Bike ID	Specs	Quantity	Total Price €	Date	Time	Customer
4	4	4	RIDLEY PHOENIX SL M/ZIPP 404 TUBULAR	1	7500	27 - 08 - 22	14 : 2 : 47	Eddie Merckx

Fig.23: Searching for invoice 4.

Test 5:

Action: Adding a user, we head to tab users and hit *"Add an Employee"*, a *"New User"* pop-up screen will appear.

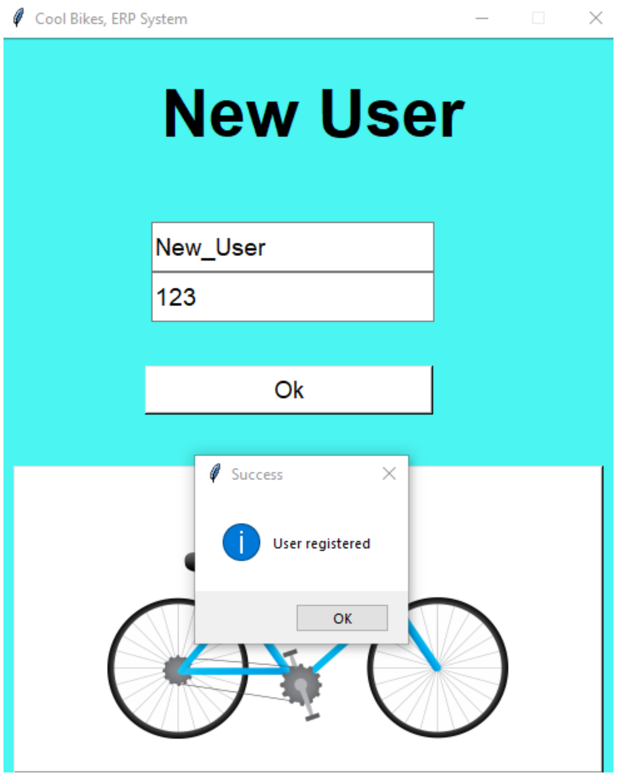
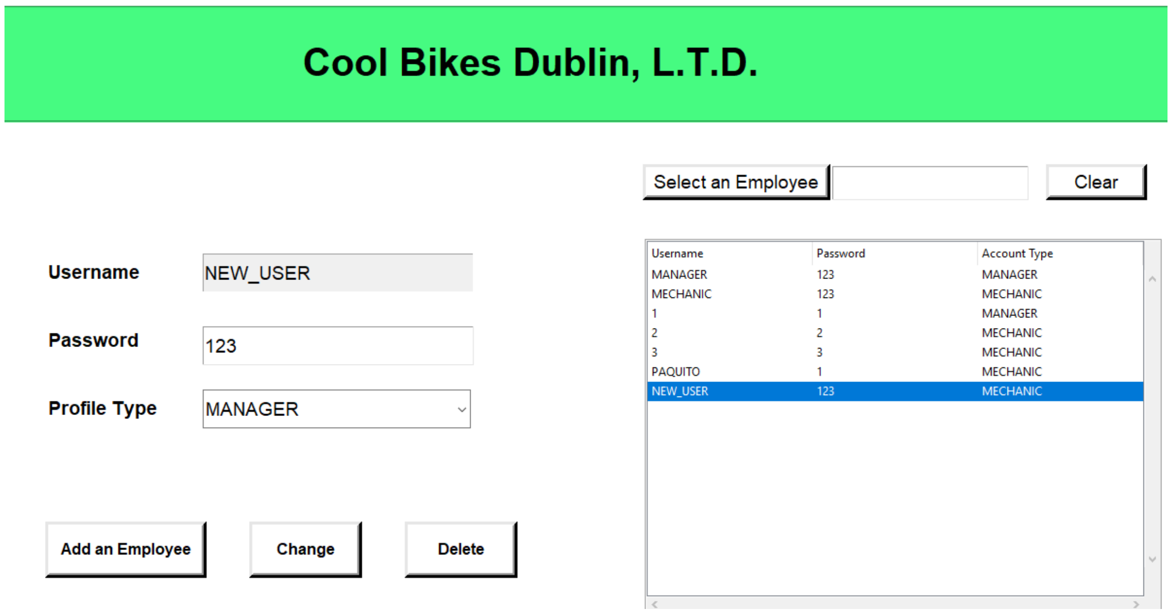


Fig.24: Adding a new user.

Action: Modifying user profile type, by default profile type is set to mechanic, we are elevating recently added *"New_User"*.

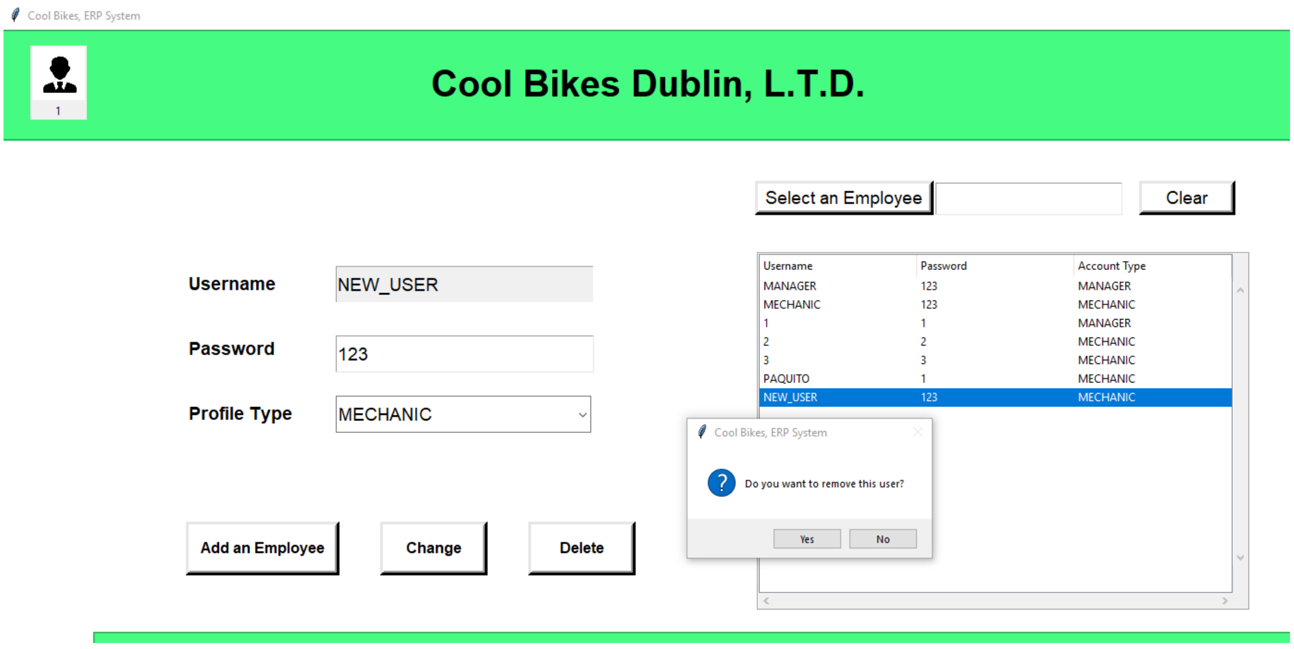


Username	NEW_USER
Password	123
Profile Type	MANAGER

Username	Password	Account Type
MANAGER	123	MANAGER
MECHANIC	123	MECHANIC
1	1	MANAGER
2	2	MECHANIC
3	3	MECHANIC
PAQUITO	1	MECHANIC
NEW_USER	123	MECHANIC

Fig.25: Modifying user.

Action: Deleting las user added.



Username NEW_USER

Password 123

Profile Type MECHANIC

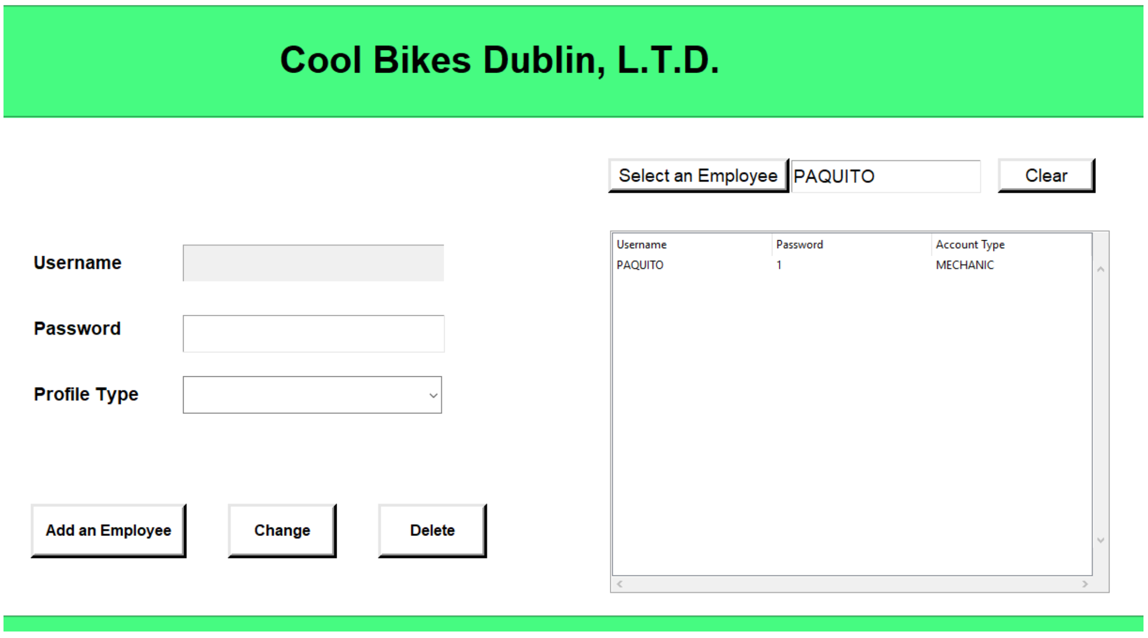
Username	Password	Account Type
MANAGER	123	MANAGER
MECHANIC	123	MECHANIC
1	1	MANAGER
2	2	MECHANIC
3	3	MECHANIC
PAQUITO	1	MECHANIC
NEW_USER	123	MECHANIC

Cool Bikes, ERP System

Do you want to remove this user?

Fig.26: Deleting user.

Action: Searching for employee *“Username” “Paquito”.*



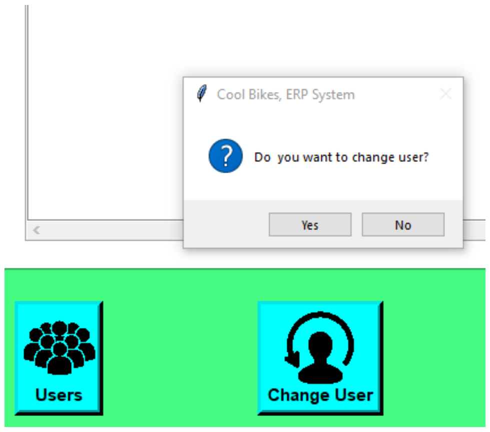
PAQUITO

Username	Password	Account Type
PAQUITO	1	MECHANIC

Fig.27: Searching user by “username”.

<u>Test 6:</u>

Action: We can simply change user by clicking on button *"Change User"*.

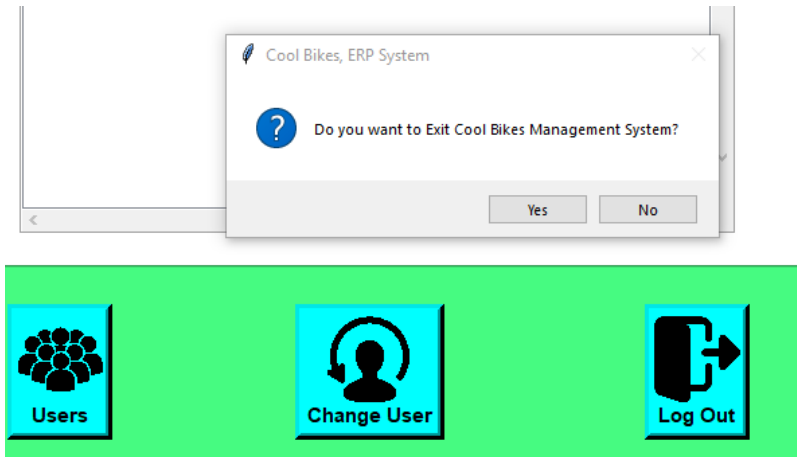

Cool Bikes, ERP System

Do you want to change user?

Fig.27: Changing user at Manager level.

<u>Test 7:</u>

Action: Log off from *"Manager"* level, by clicking on *"Log out"*.

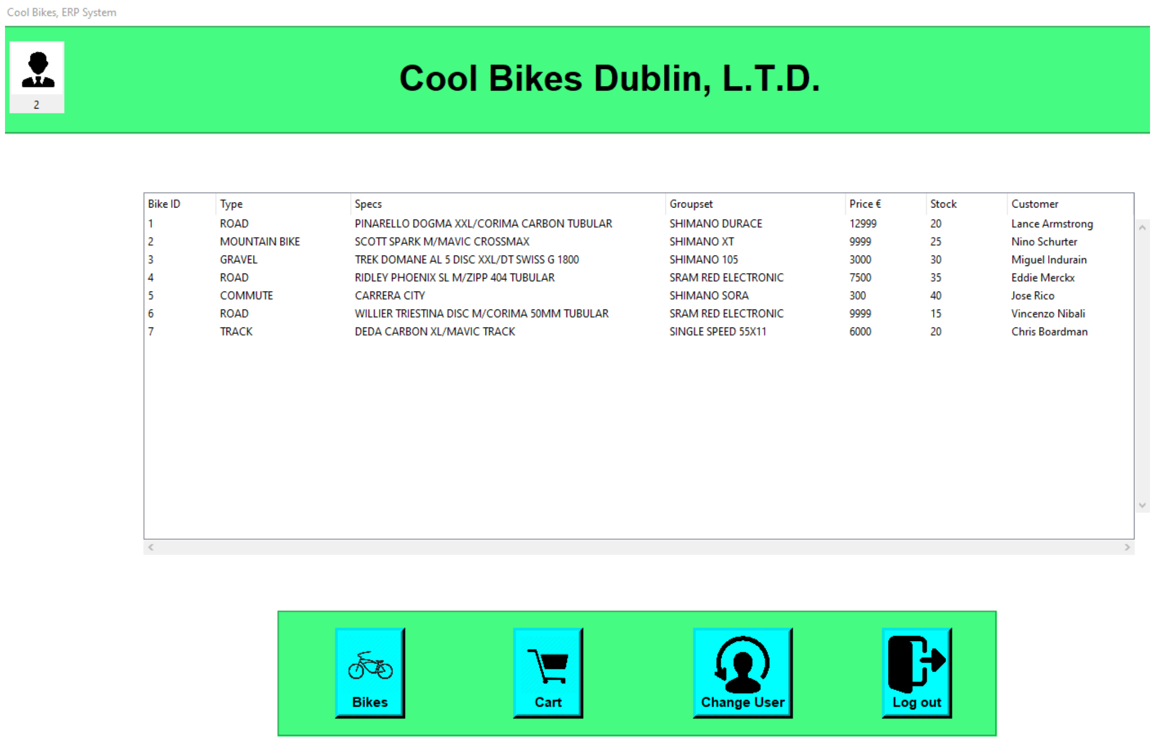

Cool Bikes, ERP System

Do you want to Exit Cool Bikes Management System?

Fig.28: Logging off from Manager level.

Test 8:

Action: Visualising bikes inventory as soon as we log in at *"Mechanic"* level.

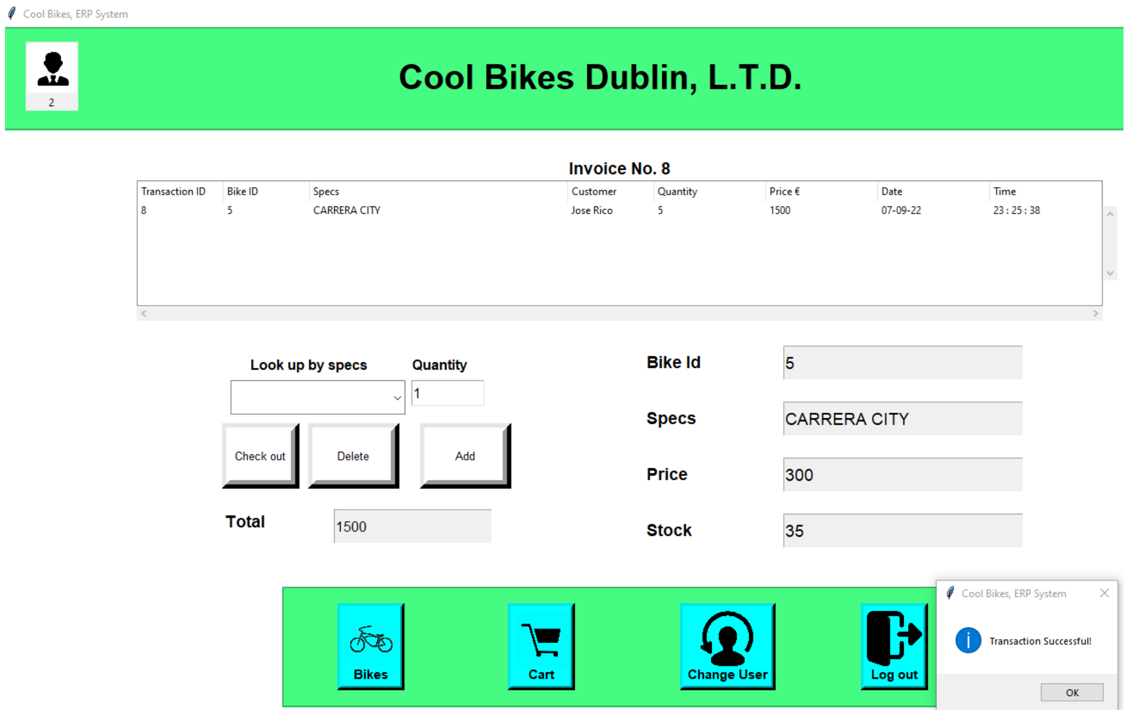


Bike ID	Type	Specs	Groupset	Price €	Stock	Customer
1	ROAD	PINARELLO DOGMA XXL/CORIMA CARBON TUBULAR	SHIMANO DURACE	12999	20	Lance Armstrong
2	MOUNTAIN BIKE	SCOTT SPARK M/MAVIC CROSSMAX	SHIMANO XT	9999	25	Nino Schurter
3	GRAVEL	TREK DOMANE AL 5 DISC XXL/DT SWISS G 1800	SHIMANO 105	3000	30	Miguel Indurain
4	ROAD	RIDLEY PHOENIX SL M/ZIPP 404 TUBULAR	SRAM RED ELECTRONIC	7500	35	Eddie Merckx
5	COMMUTE	CARRERA CITY	SHIMANO SORA	300	40	Jose Rico
6	ROAD	WILLIER TRIESTINA DISC M/CORIMA 50MM TUBULAR	SRAM RED ELECTRONIC	9999	15	Vincenzo Nibali
7	TRACK	DEDA CARBON XL/MAVIC TRACK	SINGLE SPEED 55X11	6000	20	Chris Boardman

Fig.29: Visualising bikes inventory.

Test 9:

Action: Performing a sale and checking recently added invoice at manager level.



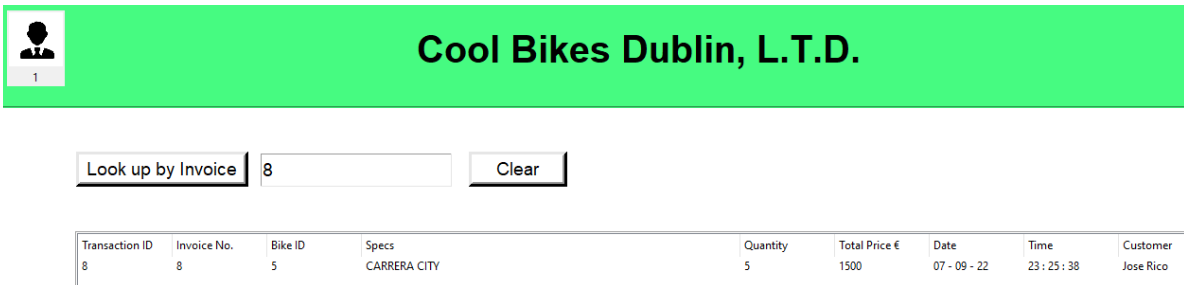
Invoice No. 8

Transaction ID	Bike ID	Specs	Customer	Quantity	Price €	Date	Time
8	5	CARRERA CITY	Jose Rico	5	1500	07-09-22	23 : 25 : 38

Look up by specs	
Quantity	1
Total	1500
Bike Id	5
Specs	CARRERA CITY
Price	300
Stock	35

Transaction Successful!

Fig.30: Creating an invoice.

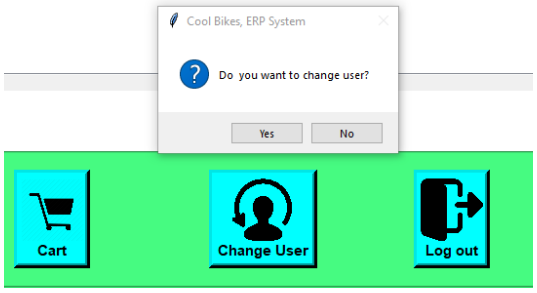

Cool Bikes Dublin, L.T.D.

Transaction ID	Invoice No.	Bike ID	Specs	Quantity	Total Price €	Date	Time	Customer
8	8	5	CARRERA CITY	5	1500	07 - 09 - 22	23 : 25 : 38	Jose Rico

Fig.31: Searching for invoice recently added number 8.

Test 10:

Action: Changing user.

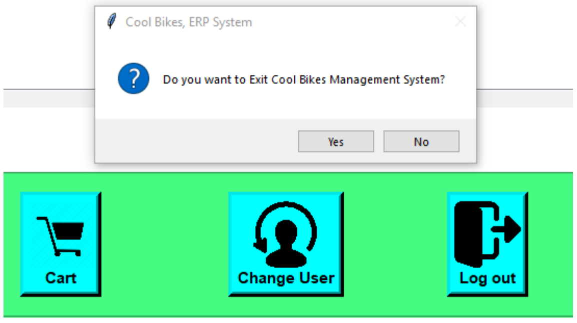

Cool Bikes, ERP System

Do you want to change user?

Fig.32: Changing user at Mechanic level.

Test 11:

Action: Logging off.


Cool Bikes, ERP System

Do you want to Exit Cool Bikes Management System?

Fig.33: Logging off from Mechanic level.

6. Conclusion and future work.

The project fulfilled the initial objective of learning about new python libraries. At the beginning I had a lot of setbacks implementing methods and finding reliable sources of information, however after some time everything started working out.
The program could be improved in many ways, starting at *"manager"* level I can enumerate the following:

1. Creating a separate table for customers, I did not consider this option when starting the project due to the complexity of adding another table. I just decided to add a customer column at *"bikes"* table.
2. Feature development of *"Invoices"* by adding an option to delete records. Once an invoice gets into the archive there is no way to delete it.

At mechanic level there is not much room for further expansion, except:

1. At *"Cart"* tab a great addition would be to add the employee who performs the sale. With current app design it is impossible to know which employee is the best seller in the company.

Wrapping up this section, I want to mention that I have enjoyed throughout the development of this project, and my python skills have been reinforced by ingesting all work being done.

Appendix

Bibliography

Wrike (no date) What Is Agile Methodology in Project Management? Available at: https://www.wrike.com/project-management-guide/faq/what-is-agile-methodology-in-project-management/ (Accessed: 10 July 2022).

Atlassian (no date) What Is Agile? Available at: https://www.atlassian.com/agile (Accessed: 10 July 2022).

Python (no date) tkinter — Python interface to Tcl/TkAvailable at: https://docs.python.org/3/library/tkinter.html (Accessed: 24 July 2022).

SQLite Tutorial (no date) SQLite Python at: https://www.sqlitetutorial.net/sqlite-python/ (Accessed: 24 July 2022).

George, EI. (2020) *'Create a Tkinter Gui With Sqlite Backend'* at: https://www.python4networkengineers.com/posts/python-intermediate/create_a_tkinter_gui_with_sqlite_backend/ (Accessed: 25 July 2022).

Us Suqlain, D. (2022) *'What is behaviour testing in software testing/'* at: https://www.educative.io/answers/what-is-behavior-testing-in-software-testing (Accessed: 15 August 2022).