# Grocery POS System

Vinay Reddy Nalla

reddyvinay977@gmail.com

Table of Contents:

Table of Contents

Main page4

code4

main.py4

My.kv9

Clock-in & Clock-out22

code24

TimePunch.py24

Add product and pricing update30

All products30

Add product31

Product update31

code32

Price.py32

Delivery and supplier44

Add delivery45

All suppliers45

Add new supplier46

Edit supplier46

Delete supplier47

code47

delivery.py47

Order reports70

Check customer orders71

Order for store71

Customer orders details72

code72

Orders\_for\_suppliers.py72

Employee Management82

All Employees83

Edit Employee83

Add Employee84

Delete Employee84

code85

EmployeeScreen.py85

employee.kv102

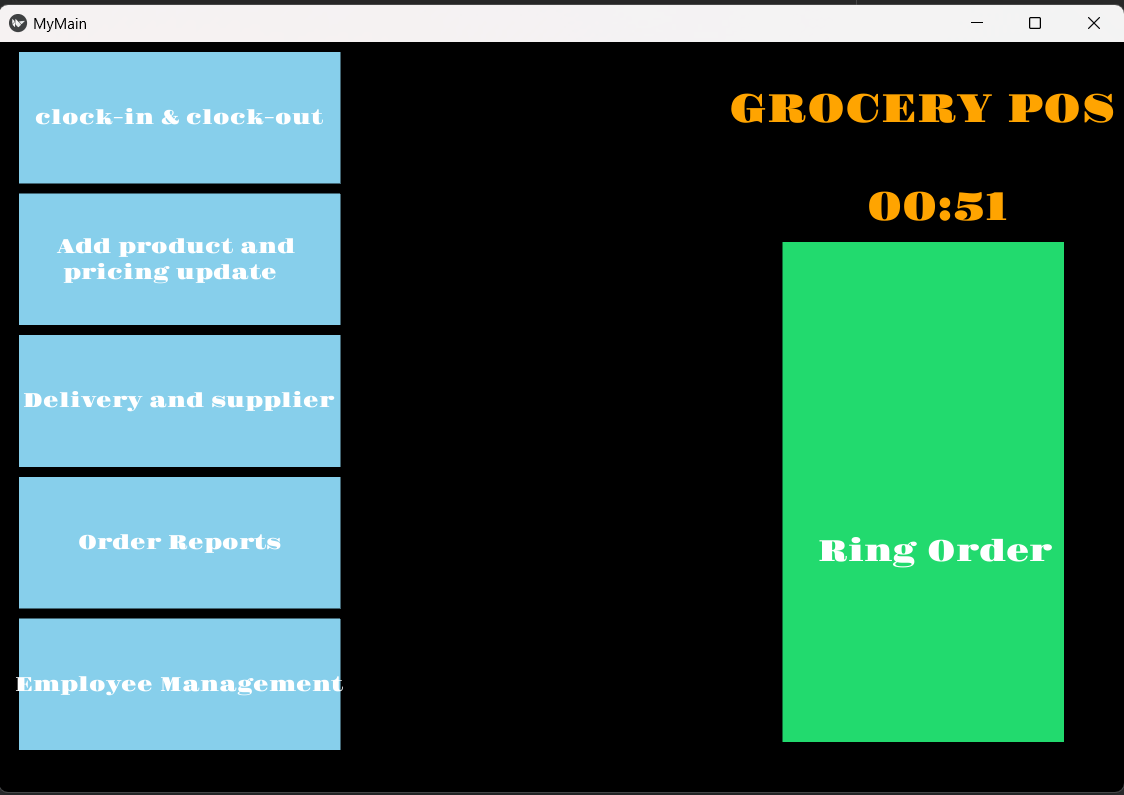
Ring Orders104

Show All Products104

Customer details105

Bill Generation105

Main Page



In the main page we have included several buttons such as clock-in & clock-out, add product and pricing update, delivery and supplier, order reports, Employee management and Ring Order where a user can perform several operations.

**Code:**

**Main.py:**

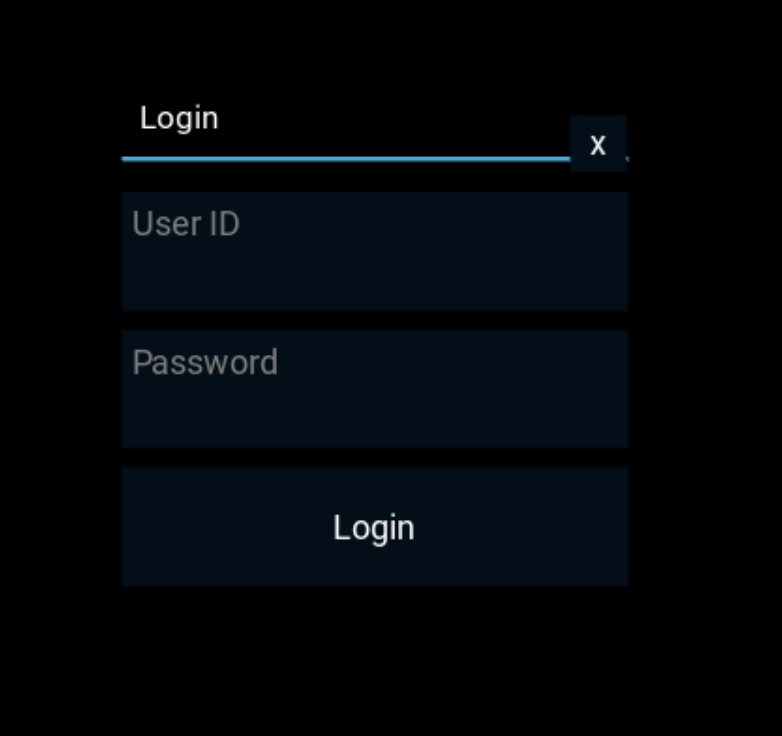
import datetime  
from EmployeeScreen import \*  
from TimePunch import \*  
from orders import \*  
from price import \*  
from delivery import \*  
from orders\_for\_suppliers import \*  
import time  
from kivy.app import App  
from kivy.clock import Clock  
from kivy.lang import Builder  
from kivy.uix.boxlayout import BoxLayout  
from kivy.uix.button import Button  
from kivy.uix.label import Label  
from kivy.uix.popup import Popup  
from kivy.uix.screenmanager import ScreenManager, Screen  
from kivy.core.window import Window

Window.clearcolor = 0.0, 0.0, 0.0, 0.0  
Window.size = 900, 600

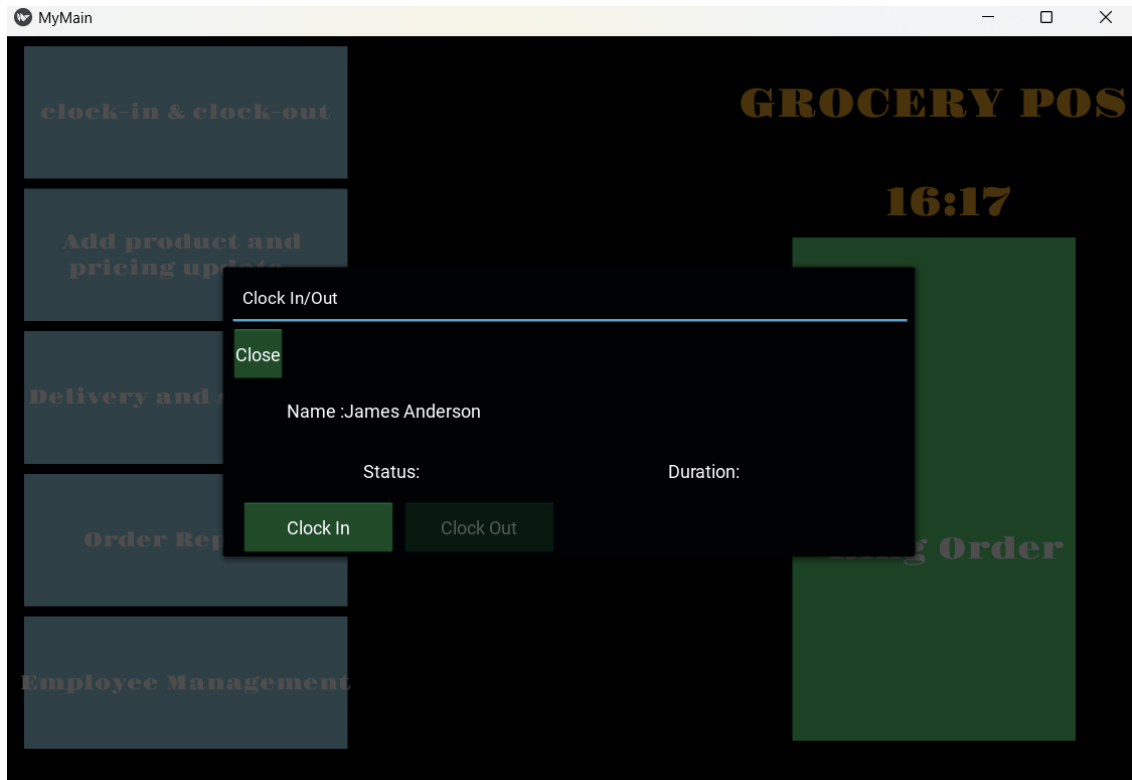
class MainWindow(Screen):  
 def \_\_init\_\_(self, \*\*kwargs):  
 super(MainWindow, self).\_\_init\_\_(\*\*kwargs)  
 Clock.schedule\_interval(self.update\_time, 1) # Update time every 1 second  
  
 def update\_time(self, dt):  
 current\_time = time.strftime("%H:%M")  
 self.ids.clock.text = f' {current\_time}'  
  
 def show\_login\_popup(self, text):  
 text1 = text  
 # Create a pop-up window for entering user ID and password  
 content = BoxLayout(orientation='vertical', spacing=10, background\_normal=''  
 , background\_color=(0.004, 0.055, 0.102, 1.0))  
  
 # Create a layout for the close button  
 close\_button\_layout = BoxLayout(size\_hint\_y=None, height=5, padding=1, spacing=1, background\_normal=''  
 , background\_color=(0.004, 0.055, 0.102, 1.0))  
  
 # Add a "Close" button to the top-right corner  
 close\_button = Button(text='x', size\_hint=(None, None), size=(30, 30), on\_press=self.dismiss\_popup,  
 background\_normal=''  
 , background\_color=(0.004, 0.055, 0.102, 1.0))  
 close\_button\_layout.add\_widget(Label()) # Add an empty label for spacing  
 close\_button\_layout.add\_widget(close\_button)  
  
 # Add the close button layout to the main content  
 content.add\_widget(close\_button\_layout)  
  
 user\_id\_input = TextInput(hint\_text='User ID', background\_normal=''  
 , background\_color=(0.004, 0.055, 0.102, 1.0), foreground\_color=(1, 1, 1, 1),  
 multiline=False)  
 password\_input = TextInput(hint\_text='Password', password=True, background\_normal=''  
 , background\_color=(0.004, 0.055, 0.102, 1.0), foreground\_color=(1, 1, 1, 1),  
 multiline=False)  
 login\_button = Button(text='Login',  
 on\_press=lambda btn: self.check\_credentials(user\_id\_input.text, password\_input.text,  
 text1),  
 background\_normal=''  
 , background\_color=(0.004, 0.055, 0.102, 1.0))  
  
 content.add\_widget(user\_id\_input)  
 content.add\_widget(password\_input)  
 content.add\_widget(login\_button)  
  
 self.popup = Popup(title='Login', content=content, size\_hint=(None, None), size=(300, 300),  
 background\_color=(0, 0, 0, 0))  
 self.popup.open()  
  
 def dismiss\_popup(self, instance):  
 if hasattr(self, 'popup') and self.popup:  
 self.popup.dismiss()  
  
 def check\_credentials(self, user\_id, password, text):  
  
 conn = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Sahith@12",  
 port='3306',  
 database='grocerystore'  
 )  
 cursor = conn.cursor()  
  
 # Fetch specific columns (first\_name, last\_name, emp\_id, phone) from the database  
 cursor.execute("SELECT employee\_login\_status,emp\_level FROM employee where emp\_id=%s and emp\_password=%s",  
 (user\_id, password))  
  
 employees\_data = cursor.fetchone()  
  
 # Close the database connection  
 cursor.close()  
 conn.close()  
 if employees\_data is not None:  
 # For simplicity, check if user ID and password are correct  
 if text == 'clock-in & clock-out':  
 self.dismiss\_popup(self)  
 TimePunch.show\_clock\_popup(self, user\_id)  
  
 elif employees\_data[0].upper() == 'ACTIVE' and text == "Ring Order":  
 self.dismiss\_popup(self)  
 s = self.manager  
 SecondWindow(id=user\_id)  
 s.current = "second"  
  
 s.transition.direction = "left"  
  
 elif employees\_data[0].upper() == 'ACTIVE' and int(employees\_data[1]) in [5] and text == "Employee Management":  
 self.dismiss\_popup(self)  
 s = self.manager  
 s.current = "Labor"  
 s.transition.direction = "right"  
  
 elif employees\_data[0].upper() == 'ACTIVE' and int(employees\_data[1]) in [4, 5] and text == "Order Reports":  
 self.dismiss\_popup(self)  
 s = self.manager  
 s.current = "order\_list"  
 s.transition.direction = "right"  
 elif employees\_data[0].upper() == 'ACTIVE' and int(employees\_data[1]) in [4, 5] and text == "Delivery and supplier":  
 self.dismiss\_popup(self)  
 s = self.manager  
 s.current = "Delivery\_list"  
 s.transition.direction = "right"  
 elif employees\_data[0].upper() == 'ACTIVE' and int(employees\_data[1]) in [4, 5] and text == "Add product and \n pricing update":  
 self.dismiss\_popup(self)  
 s = self.manager  
 s.current = "price\_update"  
 s.transition.direction = "right"  
 else:  
 # Show an error message in the pop-up  
 error\_label = Label(text='You dont have Access !')  
 self.popup.content.add\_widget(error\_label)  
  
 # Remove the error label after a short delay (adjust as needed)  
 Clock.schedule\_once(lambda dt: self.popup.content.remove\_widget(error\_label), 0.7)  
  
  
 else:  
 # Show an error message in the pop-up  
 error\_label = Label(text='Incorrect credentials. Please try again.')  
 self.popup.content.add\_widget(error\_label)  
  
 # Remove the error label after a short delay (adjust as needed)  
 Clock.schedule\_once(lambda dt: self.popup.content.remove\_widget(error\_label), 0.7)  
  
 def show\_main\_window(self):  
 # Close the pop-up and show the main window  
 self.dismiss\_popup()  
 self.layout.clear\_widgets()  
  
 pass  
  
  
class WindowManager(ScreenManager):  
 pass  
  
  
kv = Builder.load\_file("my.kv")  
  
  
class MyMainApp(App):  
 def build(self):  
 return kv  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 MyMainApp().run()

**My.kv:**

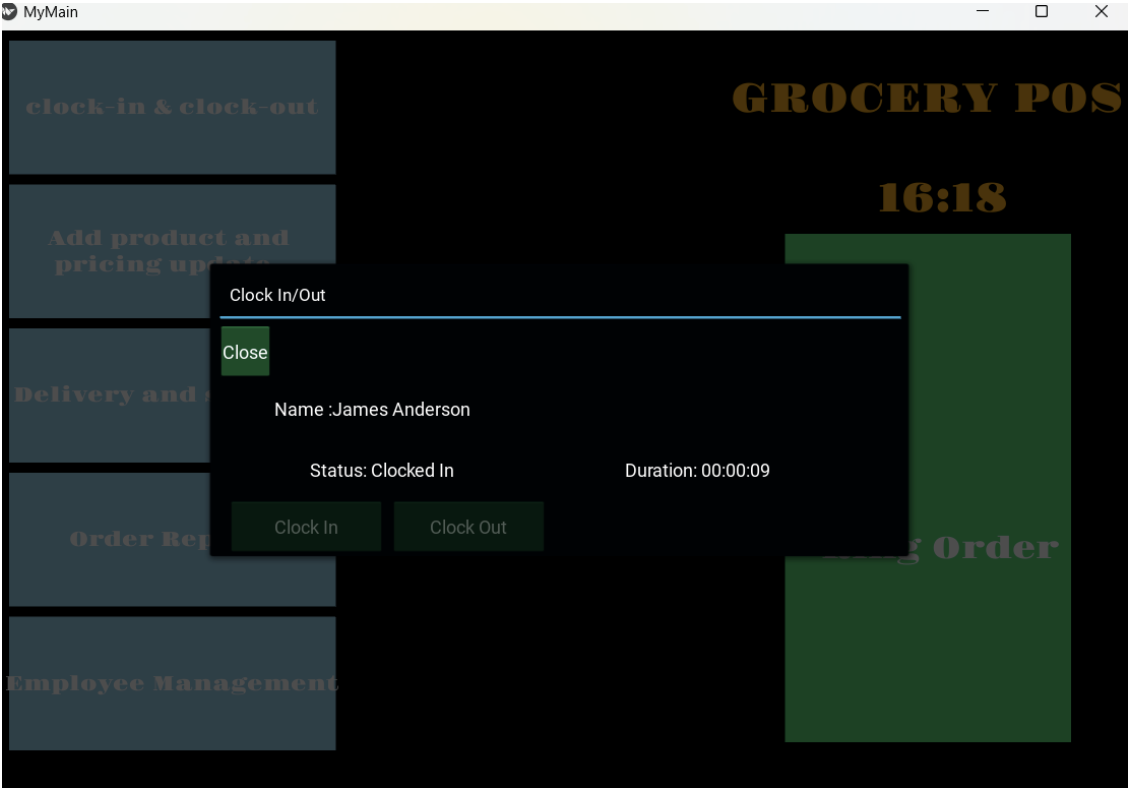
WindowManager:  
 id: screen\_manager  
 MainWindow:  
 SecondWindow:  
 TimePunch:  
 EmployeeScreen:  
 Orders:  
 Delivery:  
 Price:  
<MainWindow>:  
 name:"main"  
  
 BoxLayout:  
 orientation: 'horizontal'  
 spacing:10  
 padding:10,10,40,40  
 background\_normal:''  
 background\_color:0.004,0.055,0.102,1.0  
 BoxLayout:  
 col:1  
 orientation: 'vertical'  
 spacing: 10  
 padding:10,0,20,2  
 background\_normal:''  
 background\_color:0.53, 0.81, 0.92, 1  
 Button:  
 text: 'clock-in & clock-out'  
 font\_size:20  
 background\_normal:''  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_press: root.show\_login\_popup(self.text)  
 font\_name: 'GravitasOne-Regular.ttf'  
 Button:  
 text: 'Add product and \n pricing update'  
 font\_size:20  
 background\_normal:''  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_press: root.show\_login\_popup(self.text)  
 font\_name: 'GravitasOne-Regular.ttf'  
  
 Button:  
 font\_size:20  
 text: 'Delivery and supplier'  
 background\_normal:''  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_press: root.show\_login\_popup(self.text)  
 font\_name: 'GravitasOne-Regular.ttf'  
 Button:  
 font\_size:20  
 text: 'Order Reports'  
 background\_normal:''  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_press: root.show\_login\_popup(self.text)  
 font\_name: 'GravitasOne-Regular.ttf'  
 Button:  
 font\_size:20  
 text: 'Employee Management'  
 background\_normal:''  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_press: root.show\_login\_popup(self.text)  
 font\_name: 'GravitasOne-Regular.ttf'  
  
  
 GridLayout:  
 cols: 1  
 spacing: 20  
 padding:1,150,1,500  
 size\_hint\_y: None  
 height: self.minimum\_height  
 background\_normal:''  
 background\_color:0.004,0.055,0.102,1.0  
  
  
  
  
 BoxLayout:  
 orientation: 'vertical'  
 spacing: 10  
 padding: 50, 10, 20, 10  
 background\_normal: ''  
 background\_color: 0.004, 0.055, 0.102, 1.0  
  
 Label:  
 text: 'GROCERY POS'  
 size\_hint\_y: None  
 height: '100dp' # Adjust the height as needed  
 font\_name: 'GravitasOne-Regular.ttf'  
 font\_size: 40  
 color: 1, 0.647, 0  
  
 Label:  
 id: clock  
 text: ''  
 size\_hint\_y: None  
 height: '40dp' # Adjust the height as needed  
 font\_size: 40  
 font\_name: 'GravitasOne-Regular.ttf'  
 color: 1,0.647, 0  
  
  
  
 Button:  
  
 text: 'Ring Order'  
 padding:25,120,0,1  
 font\_size:30  
 size\_hint\_y:None  
 height:500  
 background\_normal:''  
 background\_color:0.133,0.855,0.431,1.0  
 font\_name: 'GravitasOne-Regular.ttf'  
 on\_press: root.show\_login\_popup(self.text)  
  
  
<SecondWindow>:  
 name: "second"  
  
 BoxLayout:  
 orientation: 'horizontal'  
 spacing: 10  
 padding: 10  
 # Selected Items Layout  
 BoxLayout:  
 orientation: 'vertical'  
 spacing: 10  
 # Back Button  
 BoxLayout:  
 orientation: 'horizontal'  
 size\_hint: (1, 0.1)  
  
 Button:  
 text: 'Back'  
 size\_hint: (None, None)  
 size: (100, 40)  
 font\_name: 'GravitasOne-Regular.ttf'  
 pos\_hint: {'left': 0, 'top': 1}  
 background\_normal:''  
 background\_color:0.53, 0.81, 0.92, 1  
  
 on\_release:  
 root.back(self)  
 app.root.current = "main"  
 root.manager.transition.direction = "right"  
  
 # Scroll View for Selected Items  
 ScrollView:  
  
 # Layout for Selected Items (using GridLayout)  
 GridLayout:  
 id: Selected\_items\_layout  
 cols: 2  
 spacing: 10  
 size\_hint\_y: None  
 height: self.minimum\_height  
 padding: 10, 10, 10, 100  
  
 # Dynamic content will be added programmatically  
  
 # Label for Total Amount with Tax  
 Label:  
 size\_hint\_y:0.10  
 id: total  
 text: 'Total Amount with Tax: $0.00'  
  
 # Table Layout for Discount Buttons  
 GridLayout:  
 cols: 3  
 spacing: 4  
 size\_hint\_y: None  
 height: 30  
  
 # Discount Buttons  
 Button:  
 id:dis1  
 font\_name: 'GravitasOne-Regular.ttf'  
 background\_normal:''  
 background\_color:0.004,0.055,0.102,1.0  
 text: '5% Discount'  
 size\_hint\_x: 1  
 disabled: True  
 on\_press: root.apply\_discount(0.05)  
 Button:  
 id:dis2  
 background\_normal:''  
 font\_name: 'GravitasOne-Regular.ttf'  
 background\_color:0.004,0.055,0.102,1.0  
 text: '6% Discount'  
 size\_hint\_x: 1  
 disabled: True  
 on\_press: root.apply\_discount(0.06)  
 Button:  
 id:dis3  
 background\_normal:''  
 font\_name: 'GravitasOne-Regular.ttf'  
 background\_color:0.004,0.055,0.102,1.0  
 text: '7% Discount'  
 size\_hint\_x: 1  
 disabled: True  
 on\_press: root.apply\_discount(0.07)  
 # "Remove All" Button  
 GridLayout:  
 cols: 2  
 spacing: 10  
 size\_hint\_y: None  
 height: 70  
 Button:  
 background\_normal:''  
 background\_color:0.004,0.055,0.102,1.0  
 text: 'Remove All'  
 size\_hint: (None, None)  
 size: (100, 40)  
  
 height: 40  
 on\_press: root.remove\_all\_items(self)  
 Button:  
 font\_name: 'GravitasOne-Regular.ttf'  
 background\_normal:''  
 background\_color:0.133,0.855,0.431,1.0  
 text:"pay\_bill"  
 size\_hint:(None,None)  
 size: (200,500)  
 size\_hint\_x: 1  
 height:60  
 on\_press: root.show\_customer\_details\_popup()  
 # Items Layout  
 BoxLayout:  
 orientation: 'vertical'  
  
 # Scroll View for Item Buttons  
 ScrollView:  
  
 # Layout for Item Buttons (using GridLayout)  
 GridLayout:  
 id:buttons  
 cols: 2  
 spacing: 1  
 size\_hint\_y: None  
 height: self.minimum\_height  
 Button:  
 background\_normal:''  
 background\_color:0.53, 0.81, 0.92, 1  
 text: 'Show All Products'  
 font\_name: 'GravitasOne-Regular.ttf'  
 size\_hint\_y:None  
 height:40  
 on\_press: root.build\_buttons()  
  
  
<EmployeeScreen>:  
 name: "Labor"  
 orientation: 'vertical'  
 padding: 10  
  
 BoxLayout:  
 background\_normal: ''  
 background\_color: 0.004, 0.055, 0.102, 1.0  
  
 orientation: 'vertical'  
 BoxLayout:  
 orientation: 'horizontal'  
 size\_hint: (1, 0.1)  
 background\_normal: ''  
 background\_color: 0.004, 0.055, 0.102, 1.0  
 padding:5  
 Button:  
 text: 'Back'  
 size\_hint: (None, None)  
 size: (100, 40)  
 font\_name: 'GravitasOne-Regular.ttf'  
  
 pos\_hint: {'left': 0, 'top': 1}  
 background\_normal: ''  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_release:  
 app.root.current = "main"  
 root.manager.transition.direction = "left"  
  
 GridLayout:  
 cols: 2  
 spacing: 30  
 background\_normal: ''  
 background\_color: 0.004, 0.055, 0.102, 1.0  
  
 size\_hint\_y: 0.4 # Adjust this value to control the height  
 pos\_hint: {'center\_x': 0.5}  
  
  
 Button:  
  
 text: 'All Employees'  
 background\_normal: ''  
 font\_name: 'GravitasOne-Regular.ttf'  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_press: root.show\_all\_employees()  
  
 Button:  
 text: 'Add Employee'  
 background\_normal: ''  
 font\_name: 'GravitasOne-Regular.ttf'  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_press: root.add\_employee()  
  
 Button:  
 text: 'Edit Employee'  
 background\_normal: ''  
 font\_name: 'GravitasOne-Regular.ttf'  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_press: root.edit\_employee()  
  
 Button:  
 text: 'Delete Employee'  
 font\_name: 'GravitasOne-Regular.ttf'  
 background\_normal: ''  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_press: root.delete\_employee()  
  
<Orders>:  
 name: "order\_list"  
 orientation: 'vertical'  
 padding: 10  
  
 BoxLayout:  
 background\_normal: ''  
 background\_color: 0.004, 0.055, 0.102, 1.0  
  
 orientation: 'vertical'  
 BoxLayout:  
 orientation: 'horizontal'  
 size\_hint: (1, 0.1)  
 background\_normal: ''  
 background\_color: 0.004, 0.055, 0.102, 1.0  
 padding:5  
 Button:  
 text: 'Back'  
 font\_name: 'GravitasOne-Regular.ttf'  
 size\_hint: (None, None)  
 size: (100, 40)  
 pos\_hint: {'left': 0, 'top': 1}  
 background\_normal: ''  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_release:  
 app.root.current = "main"  
 root.manager.transition.direction = "left"  
  
 GridLayout:  
 cols: 2  
 spacing: 30  
 padding:30  
 background\_normal: ''  
 background\_color: 0.004, 0.055, 0.102, 1.0  
  
 size\_hint\_y: 0.4 # Adjust this value to control the height  
 pos\_hint: {'center\_x': 0.5}  
  
  
 Button:  
  
 text: 'check customer orders'  
 font\_name: 'GravitasOne-Regular.ttf'  
 background\_normal: ''  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_press:root.order\_check()  
  
  
 Button:  
 text: 'order for store '  
 font\_name: 'GravitasOne-Regular.ttf'  
 background\_normal: ''  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_press:root.show\_all\_orders()  
  
 Button:  
 font\_name: 'GravitasOne-Regular.ttf'  
 text: 'customer orders id check '  
 background\_normal: ''  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_press:root.order\_id\_check()  
  
<Delivery>:  
 name: "Delivery\_list"  
 orientation: 'vertical'  
 padding: 10  
  
 BoxLayout:  
 background\_normal: ''  
 background\_color: 0.004,0.055,0.102,1.0  
  
 orientation: 'vertical'  
 BoxLayout:  
 orientation: 'horizontal'  
 size\_hint: (1, 0.1)  
 background\_normal: ''  
 background\_color: 0.004, 0.055, 0.102, 1.0  
 padding:5  
 Button:  
 text: 'Back'  
 font\_name: 'GravitasOne-Regular.ttf'  
 size\_hint: (None, None)  
 size: (100, 40)  
  
 pos\_hint: {'left': 0, 'top': 1}  
 background\_normal: ''  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_release:  
 app.root.current = "main"  
 root.manager.transition.direction = "left"  
  
 GridLayout:  
 cols: 2  
 spacing: 20  
 padding:20  
 background\_normal: ''  
 background\_color: 0.004, 0.055, 0.102, 1.0  
  
 size\_hint\_y: 0.4 # Adjust this value to control the height  
 pos\_hint: {'center\_x': 0.5}  
  
  
 Button:  
  
 text: 'Add delivery '  
 background\_normal: ''  
 font\_name: 'GravitasOne-Regular.ttf'  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_press: root.delivery\_details()  
  
 Button:  
 text: 'All supplier '  
 background\_normal: ''  
 font\_name: 'GravitasOne-Regular.ttf'  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_press:root.show\_all\_suppliers()  
 Button:  
 text: 'Add New supplier '  
 font\_name: 'GravitasOne-Regular.ttf'  
 background\_normal: ''  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_press:root.add\_supplier()  
 Button:  
 text: 'Edit supplier '  
 font\_name: 'GravitasOne-Regular.ttf'  
 background\_normal: ''  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_press:root.edit\_supplier()  
 Button:  
  
 text: 'Delete supplier'  
 font\_name: 'GravitasOne-Regular.ttf'  
 background\_normal: ''  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_press:root.delete\_supplier()  
  
  
  
<Price>:  
 name: "price\_update"  
 orientation: 'vertical'  
 padding: 10  
  
 BoxLayout:  
 background\_normal: ''  
 background\_color: 0.004, 0.055, 0.102, 1.0  
  
 orientation: 'vertical'  
 BoxLayout:  
 orientation: 'horizontal'  
 size\_hint: (1, 0.1)  
 background\_normal: ''  
 background\_color: 0.004, 0.055, 0.102, 1.0  
 padding:5  
 spacing:5  
  
 Button:  
 text: 'Back'  
 size\_hint: (None, None)  
 size: (100, 40)  
 font\_name: 'GravitasOne-Regular.ttf'  
 pos\_hint: {'left': 0, 'top': 1}  
 background\_normal:''  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_release:  
 app.root.current = "main"  
 root.manager.transition.direction = "left"  
  
 GridLayout:  
 cols: 2  
 padding:80  
 spacing: 30 # Adjust this value to control the spacing between buttons  
 background\_normal: ''  
 background\_color: 0.004, 0.055, 0.102, 1.0  
  
 size\_hint\_y: 0.4 # Adjust this value to control the height  
 pos\_hint: {'center\_x': 0.5}  
  
 Button:  
 text: 'All Products '  
 background\_normal: ''  
 font\_name: 'GravitasOne-Regular.ttf'  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_press: root.show\_all\_products()  
 Button:  
 text: 'Add Products '  
 background\_normal: ''  
 font\_name: 'GravitasOne-Regular.ttf'  
 background\_color:0.53, 0.81, 0.92, 1  
 on\_press:root.add\_product()  
 Button:  
 text: 'product update '  
 background\_normal: ''  
 background\_color:0.53, 0.81, 0.92, 1  
 font\_name: 'GravitasOne-Regular.ttf'  
 on\_press:root.edit\_prdouct()  
  
 **Clock-in & Clock-out page:**



When the user clicks on the button “clock-in & clock-out" a pop-up page would be appeared where the user needs to provide his userid and password to make a successful clock-in.



Once the user enters his credentials a screen with his full name, status, duration and clock-in button appears, where the user can hit the clock in button to complete the clock-in process.



When the user hits the clock in button, we can see the status updated to “Clocked In” and the duration of work starts counting. Then the user can hit the close button to crash the window and can proceed to work.

**Code:**

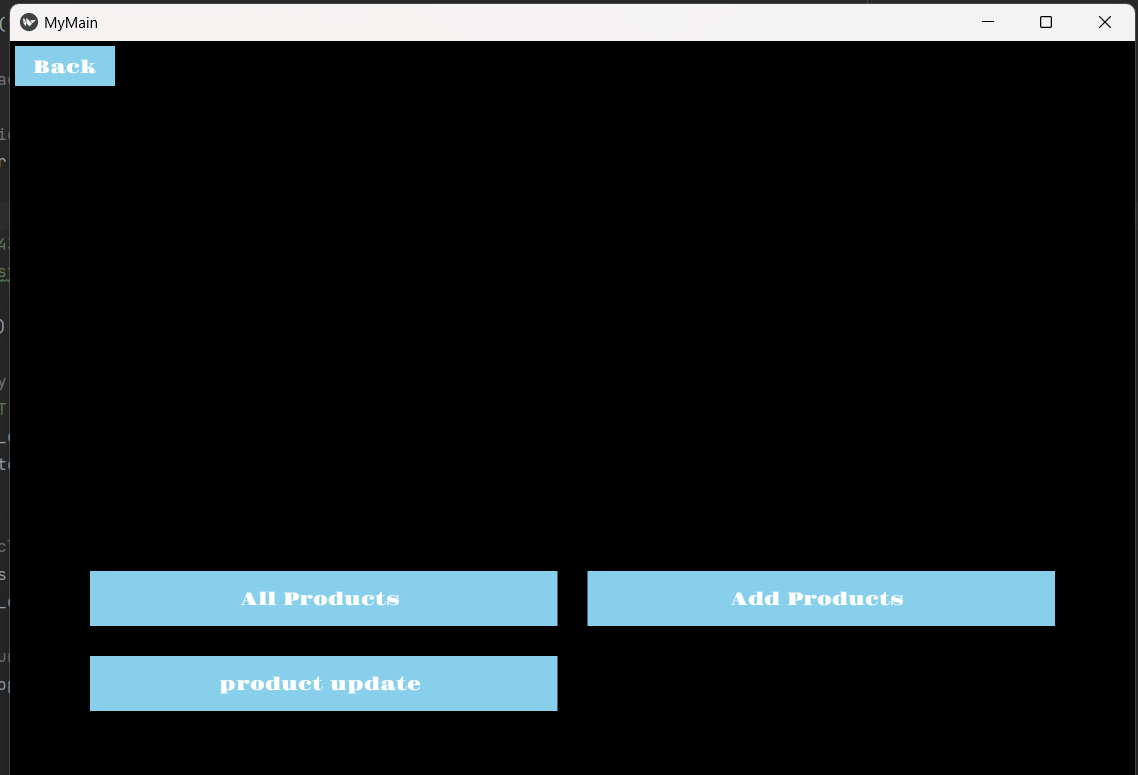
**Timepunch.py:**

from kivy.uix.gridlayout import GridLayout  
from kivy.uix.popup import Popup  
from kivy.uix.boxlayout import BoxLayout  
from kivy.uix.button import Button  
from kivy.uix.label import Label  
from kivy.clock import Clock  
from datetime import datetime  
  
# Import necessary database modules  
import mysql.connector  
from kivy.uix.screenmanager import Screen

class TimePunch(Screen):

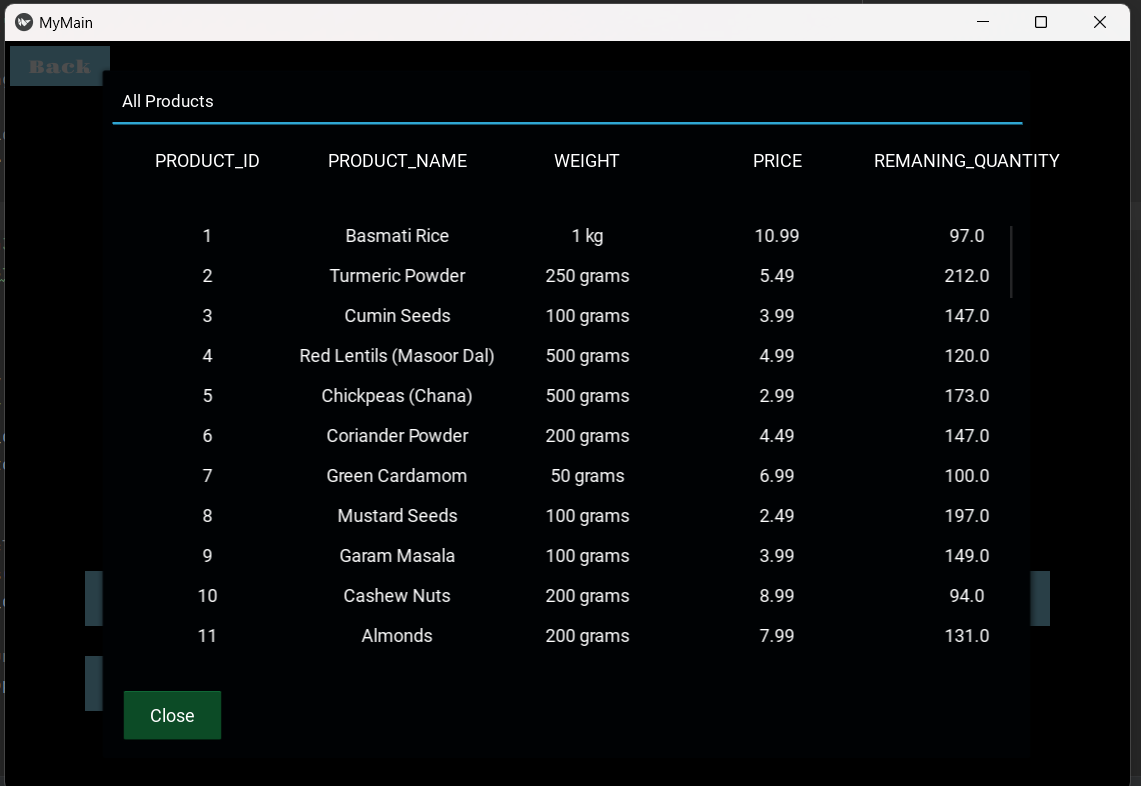
def show\_clock\_popup(self, user):  
  
 # Create an instance of the popup  
 popup = ClockPopup(user\_id=user)  
 # Open the popup  
 popup.open()  
  
  
class ClockPopup(Popup):  
 def \_\_init\_\_(self, user\_id, \*\*kwargs):  
 super().\_\_init\_\_(\*\*kwargs)  
 self.title = "Clock In/Out"  
 self.size\_hint = (None, None)  
 self.size = (700, 300) # Larger size  
 self.user = user\_id  
 self.background\_color = (0.004, 0.055, 0.102, 1.0)  
 self.auto\_dismiss=False  
 # Connect to the database  
 self.connection = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Sahith@12",  
 database="grocerystore"  
 )  
  
 # Create layouts  
 self.layout1 = BoxLayout(orientation="vertical",background\_color=(0.004, 0.055, 0.102, 1.0))  
 self.layout11 = BoxLayout(orientation="vertical",background\_color=(0.004, 0.055, 0.102, 1.0))  
 self.layout2 = GridLayout(cols=2, spacing=10, padding=10,background\_color=(0.004, 0.055, 0.102, 1.0) )  
 self.layout3 = GridLayout(cols=2, spacing=10, padding=10,background\_color=(0.004, 0.055, 0.102, 1.0) )  
  
 # First Layout: Close button at right corner  
 self.close\_button = Button(text="Close",background\_color=(0.133, 0.855, 0.431, 1.0), size\_hint=(None, None), size=(50, 50))  
 self.close\_button.bind(on\_press=self.dismiss)  
 self.layout1.add\_widget(Label()) # Empty widget for spacing  
 self.layout1.add\_widget(self.close\_button)  
 self.name\_label = Label(text="", size\_hint=(None, None), size=(300, 50))  
 self.layout11.add\_widget(self.name\_label)  
 # Second Layout: Status and duration labels  
 self.status\_label = Label(text="Status: ", size\_hint=(None, None), size=(300, 50))  
 self.duration\_label = Label(text="Duration: ", size\_hint=(None, None), size=(300, 50))  
 self.layout2.add\_widget(self.status\_label)  
 self.layout2.add\_widget(self.duration\_label)  
  
 # Third Layout: Clock in and clock out buttons  
 self.clock\_in\_button = Button(text="Clock In",background\_color=(0.133, 0.855, 0.431, 1.0), size\_hint=(None, None), size=(150, 50))  
 self.clock\_out\_button = Button(text="Clock Out",background\_color=(0.133, 0.855, 0.431, 1.0), size\_hint=(None, None), size=(150, 50))  
 self.layout3.add\_widget(self.clock\_in\_button)  
 self.layout3.add\_widget(self.clock\_out\_button)  
  
 # Bind the clock\_in and clock\_out methods to the buttons' on\_press event  
 self.clock\_in\_button.bind(on\_press=lambda instance: self.clock\_in(self.user, instance))  
 self.clock\_out\_button.bind(on\_press=lambda instance: self.clock\_out(self.user, instance))  
  
 # Add layouts to the main layout  
 self.layout = BoxLayout(orientation="vertical",background\_color=(0.004, 0.055, 0.102, 1.0))  
 self.layout.add\_widget(self.layout1)  
 self.layout.add\_widget(self.layout11)  
 self.layout.add\_widget(self.layout2)  
 self.layout.add\_widget(self.layout3)  
  
 self.add\_widget(self.layout)  
  
 self.clocked\_in = False  
 self.clock\_start\_time = None  
 self.timer\_event = None  
  
 # Fetch initial clock-in status from the database  
 self.fetch\_clock\_status(self.user)  
  
 def fetch\_clock\_status(self, user):  
 # Query to fetch the last clock-in status and time from the database  
 query = "SELECT clock\_type, emp\_date, emp\_time FROM timepunch WHERE emp\_id = %s ORDER BY emp\_date DESC, emp\_time DESC LIMIT 1"  
 user\_id = user  
  
 # Execute the query  
 cursor = self.connection.cursor()  
 cursor.execute(query, (user\_id,))  
 result = cursor.fetchone()  
 query1 = "SELECT concat(first\_name,' ',last\_name) from employee where emp\_id=%s"  
 user\_id = user # Assuming user ID is 1 for demonstration  
  
 # Execute the query  
 cursor = self.connection.cursor()  
 cursor.execute(query1, (user\_id,))  
 result1 = cursor.fetchone()  
 self.name\_label.text = "Name :"+str(result1[0])  
  
 # Update the status label and start the timer based on the fetched result  
 if result:  
 self.clocked\_in = result[0] == 'clockin'  
 if self.clocked\_in:  
 self.status\_label.text = "Status: Clocked In"  
 # Extract date and time components from the result  
 clock\_date = result[1]  
 clock\_time = result[2]  
 # Convert time component to datetime.time  
 clock\_time = datetime.strptime(str(clock\_time), '%H:%M:%S').time()  
 # Combine date and time components  
 clock\_datetime = datetime.combine(clock\_date, clock\_time)  
 self.start\_timer(clock\_datetime) # Start the timer with the last clocked-in time  
 else:  
 self.status\_label.text = "Status: Clocked Out"  
 # Disable clock-in button if already clocked in, disable clock-out button if already clocked out  
 if self.clocked\_in:  
 self.clock\_in\_button.disabled = True  
 self.clock\_out\_button.disabled = False  
 else:  
 self.clock\_in\_button.disabled = False  
 self.clock\_out\_button.disabled = True  
 cursor.close()  
  
 def start\_timer(self, start\_time):  
 # Start the timer with the given start time  
 self.clock\_start\_time = start\_time  
 self.update\_timer()  
  
 if self.timer\_event:  
 self.timer\_event.cancel()  
  
 self.timer\_event = Clock.schedule\_interval(self.update\_timer, 1)  
  
 def update\_timer(self, \*args):  
 # Update the duration label with the elapsed time since the start time  
 current\_time = datetime.now()  
 elapsed\_time = current\_time - self.clock\_start\_time  
 hours, remainder = divmod(elapsed\_time.total\_seconds(), 3600)  
 minutes, seconds = divmod(remainder, 60)  
 self.duration\_label.text = "Duration: {:02}:{:02}:{:02}".format(int(hours), int(minutes), int(seconds))  
  
 def clock\_in(self, user, instance):  
 # Update clock-in status in the database  
 self.update\_clock\_status('clockin', user)  
  
 def clock\_out(self, user, instance):  
 # Stop the timer  
 if self.timer\_event:  
 self.timer\_event.cancel()  
 # Disable clock-out button  
 self.clock\_out\_button.disabled = True  
 # Update clock-out status in the database  
 self.update\_clock\_status('clockout', user)  
  
 def update\_clock\_status(self, clock\_in\_status, user):  
 # Insert a new row into the timepunch table with clock\_type, emp\_id, emp\_date, and emp\_time  
 query = "INSERT INTO timepunch (clock\_type, emp\_id, emp\_date, emp\_time) VALUES (%s, %s, %s, %s)"  
 user\_id = user # Assuming user ID is 1 for demonstration  
 current\_datetime = datetime.now()  
 clock\_date = current\_datetime.date()  
 clock\_time = current\_datetime.time()  
  
 # Execute the insert query  
 cursor = self.connection.cursor()  
 cursor.execute(query, (clock\_in\_status, user\_id, clock\_date, clock\_time))  
 self.connection.commit()  
 cursor.close()  
  
 # Update the status label in the popup  
 self.clocked\_in = clock\_in\_status == 'clockin'  
 if self.clocked\_in:  
 query1='update employee set employee\_login\_status=%s where emp\_id=%s'  
 cursor=self.connection.cursor()  
 cursor.execute(query1,('ACTIVE',user\_id))  
 self.connection.commit()  
 cursor.close()  
 self.status\_label.text = "Status: Clocked In"  
 self.start\_timer(current\_datetime) # Start timer if clocked in  
 else:  
 query2 = 'update employee set employee\_login\_status=%s where emp\_id=%s'  
 cursor = self.connection.cursor()  
 cursor.execute(query2, ('INACTIVE', user\_id))  
 self.connection.commit()  
 cursor.close()  
 self.status\_label.text = "Status: Clocked Out"  
 current\_time = datetime.now()  
 elapsed\_time = current\_time - self.clock\_start\_time  
 hours, remainder = divmod(elapsed\_time.total\_seconds(), 3600)  
 minutes, seconds = divmod(remainder, 60)  
 self.duration\_label.text = "Duration: {:02}:{:02}:{:02}".format(int(hours), int(minutes), int(seconds))  
 # Reset duration label if clocked out  
 self.clock\_in\_button.disabled = True  
 self.clock\_out\_button.disabled = True

**Add product and pricing update:**



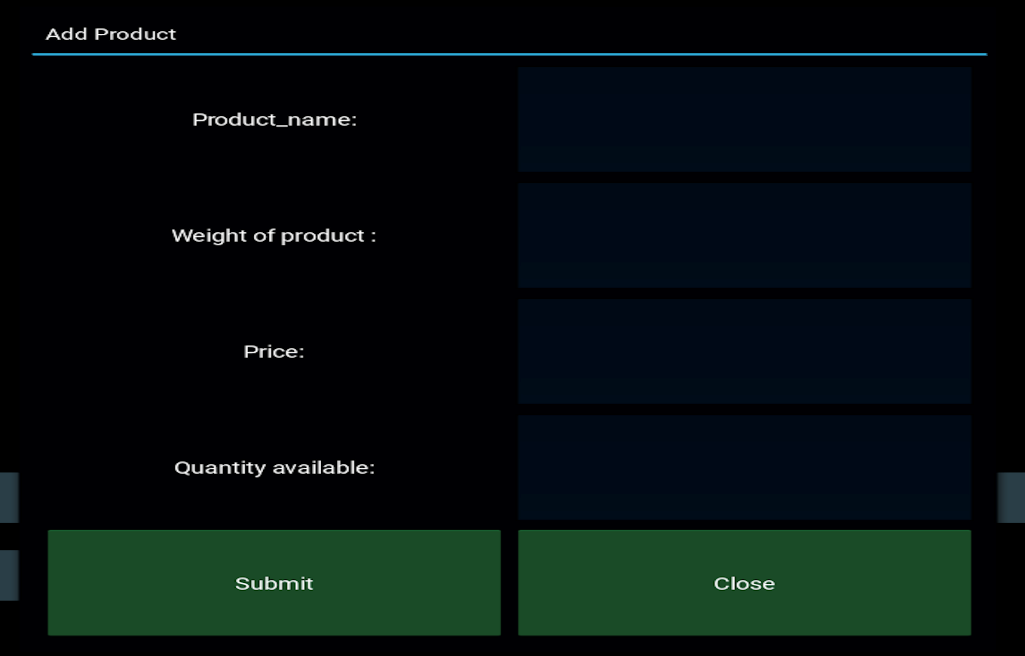
When the user hits “Add product and pricing update” button, the above page will be shown which consists of buttons “All Products”, “Add Products” and “product update”. User can also use a button named “back” where he will be redirected to home page.

**All Products:**



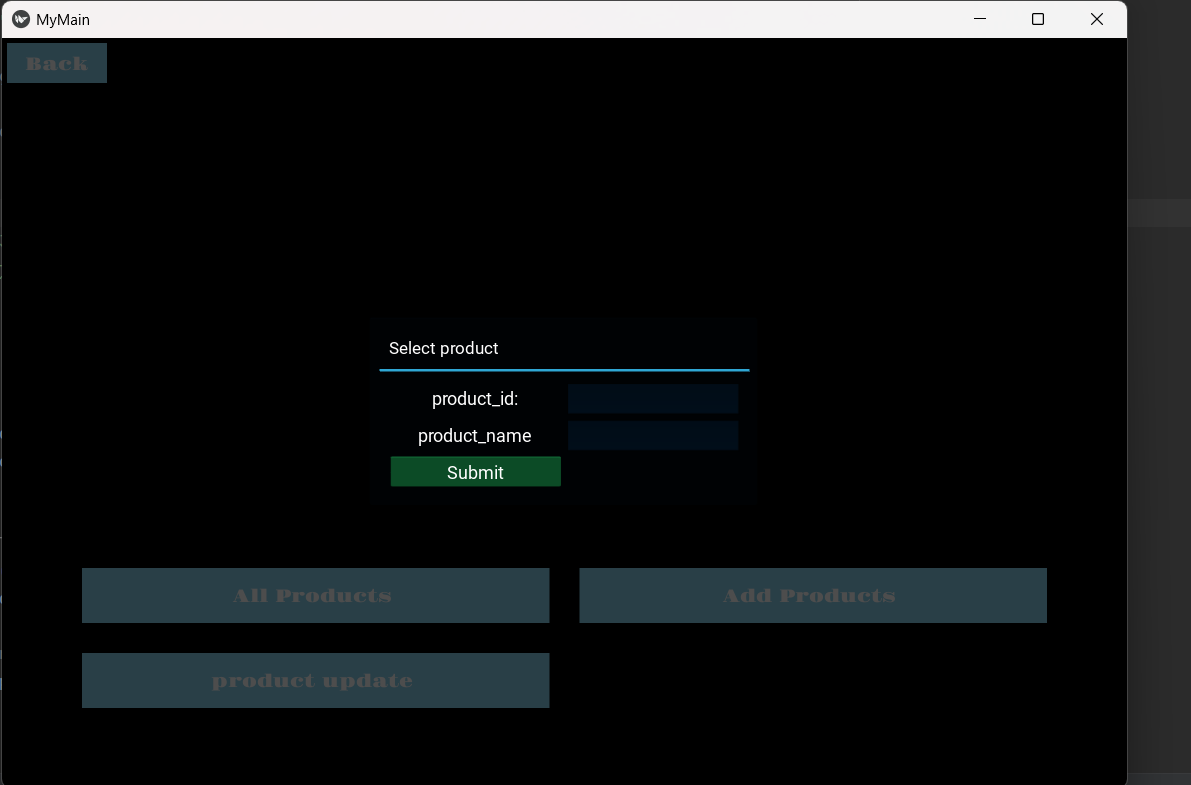
When the user hits “All products” button a pop is displayed with the quantity available in the inventory. A close button is included to crush the pop-up page.

**Add Products:**



When the user hits “Add Products” button a pop up is displayed which is required to fill all the information. A submit button is used to save the details of products to database and a close button is also added to close the pop-up page.

**Product update:**



When the user hits “product update” button a pop up is displayed showing values to be entered in product\_id and product\_name and a submit button to update the product.

**Code:**

**Price.py:**

import mysql.connector  
from kivy.uix.scrollview import ScrollView  
from kivy.uix.boxlayout import BoxLayout  
from kivy.uix.screenmanager import Screen  
from kivy.uix.popup import Popup  
from kivy.uix.gridlayout import GridLayout  
from kivy.uix.label import Label

from kivy.uix.textinput import TextInput  
from kivy.uix.button import Button  
from kivy.uix.spinner import Spinner  
import re  
class Price(Screen):  
  
 def show\_all\_products(self):  
 # Establish a connection to your MySQL database  
 conn = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Sahith@12",  
 database='grocerystore'  
 )  
 cursor = conn.cursor()  
  
 # Fetch specific columns (first\_name, last\_name, emp\_id, phone) from the database  
 cursor.execute(  
 "SELECT PRODUCT\_ID,PRODUCT\_NAME,WEIGHT\_OF\_PRODUCT,PRODUCT\_PRICE,PRODUCT\_AVAILABLE\_QUANTITY FROM PRODUCTS")  
 PRODUCTS\_data = cursor.fetchall()  
 # Close the database connection  
 cursor.close()  
 conn.close()  
  
 # Create a BoxLayout to organize the labels and the scroll view  
 content\_layout = BoxLayout(orientation='vertical', padding=10, spacing=40,  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
  
 # Create a GridLayout for the labels  
 labels\_layout = GridLayout(cols=5, size\_hint\_y=None, height='40dp', padding=10, spacing=40)  
  
 # Set fixed widths for columns  
 column\_widths = [150, 150, 150, 150]  
  
 # Add column labels  
 labels\_layout.add\_widget(Label(text='PRODUCT\_ID', size\_hint\_x=None, width=column\_widths[0]))  
 labels\_layout.add\_widget(Label(text='PRODUCT\_NAME', size\_hint\_x=None, width=column\_widths[1]))  
 labels\_layout.add\_widget(Label(text='WEIGHT', size\_hint\_x=None, width=column\_widths[2]))  
 labels\_layout.add\_widget(Label(text='PRICE', size\_hint\_x=None, width=column\_widths[3]))  
 labels\_layout.add\_widget(Label(text='REMANING\_QUANTITY', size\_hint\_x=None, width=column\_widths[3]))  
  
 # Add labels layout to content layout  
 content\_layout.add\_widget(labels\_layout)  
  
 # Create a ScrollView with GridLayout inside to allow scrolling  
 layout = GridLayout(cols=5, size\_hint\_y=None, padding=10, spacing=40)  
 layout.bind(minimum\_height=layout.setter('height'))  
  
 # Add employee data  
 for product in PRODUCTS\_data:  
 id = Label(text=str(product[0]), size\_hint\_x=None, width=column\_widths[0], height='40dp',  
 )  
 name = Label(text=product[1], size\_hint\_x=None, width=column\_widths[1], height='40dp',  
 )  
 weight = Label(text=product[2], size\_hint\_x=None, width=column\_widths[2], height='40dp',  
 )  
 price = Label(text=str(product[3]), size\_hint\_x=None, width=column\_widths[3], height='40dp',  
 )  
 quantity = Label(text=str(product[4]), size\_hint\_x=None, width=column\_widths[3], height='40dp',  
 )  
  
 layout.add\_widget(id)  
 layout.add\_widget(name)  
 layout.add\_widget(weight)  
 layout.add\_widget(price)  
 layout.add\_widget(quantity)  
  
 # Create a ScrollView with GridLayout inside to allow scrolling  
 scroll\_view = ScrollView()  
 scroll\_view.add\_widget(layout)  
  
 # Add scroll view to content layout  
 content\_layout.add\_widget(scroll\_view)  
  
 # Create a close button  
 close\_button = Button(text='Close', size\_hint=(None, None), size=(100, 50),  
 background\_color=(0.133, 0.855, 0.431, 1.0))  
 close\_button.bind(on\_press=self.dismiss\_popup)  
  
 # Add close button to content layout  
 content\_layout.add\_widget(close\_button)  
  
 # Create a Popup with the content layout  
 self.popup = Popup(title='All Products', content=content\_layout, size\_hint=(None, None), size=(940, 700),  
 background\_color=(0.004, 0.055, 0.102, 1.0), auto\_dismiss=False)  
 self.popup.open()  
  
 def dismiss\_popup(self, instance):  
 if hasattr(self, 'popup') and self.popup:  
 self.popup.dismiss()  
  
 def edit\_prdouct(self):  
 # Create a GridLayout to organize input fields  
 layout = GridLayout(cols=2, spacing=5, padding=10)  
  
 # Add labels and input fields for user ID and password  
 layout.add\_widget(Label(text='product\_id:', ))  
 self.product\_id = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.product\_id)  
  
 layout.add\_widget(Label(text='product\_name', ))  
 self.product\_name = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.product\_name)  
 # Create a submit button  
 submit\_button1 = Button(text='Submit', background\_color=(0.133, 0.855, 0.431, 1.0))  
 submit\_button1.bind(on\_press=self.authenticate\_product)  
 layout.add\_widget(submit\_button1)  
  
 # Create a Popup with the layout  
 self.popup = Popup(title='Select product', content=layout, size\_hint=(None, None),  
 background\_color=(0.004, 0.055, 0.102, 1.0), size=(400, 200))  
 self.popup.open()  
  
 def authenticate\_product(self, instance):  
 # Retrieve user ID and password from input fields  
 proid = self.product\_id.text.strip()  
 proname = self.product\_name.text.strip()  
 if not all([proname, proid]):  
 self.show\_error\_popup("All fields are required.")  
 return  
 if not proid.isdigit():  
 self.show\_error\_popup("Invalid product id number.\n product id number must be digits.")  
 return  
 # Perform authentication against the database  
 try:  
 # Establish a connection to MySQL database  
 conn = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Sahith@12",  
 database="grocerystore"  
 )  
 cursor = conn.cursor()  
  
 # Execute SELECT query to retrieve employee details based on user ID and password  
 select\_query = "SELECT \* FROM products WHERE product\_name = %s AND product\_id = %s"  
 cursor.execute(select\_query, (proname, proid))  
 product\_e = cursor.fetchone()  
  
 if product\_e:  
 # product found, close current popup and display details in another popup  
 self.popup.dismiss()  
 self.show\_product\_details(product\_e)  
 else:  
 # Employee not found, show error message  
 self.show\_error\_popup("Invalid product\_id or product\_name.")  
  
 cursor.close()  
 conn.close()  
  
 except mysql.connector.Error as e:  
 p = str(e)  
 self.show\_error\_popupp("Failed to authenticate \n{}".format(  
 p[13:].replace('Duplicate entry', 'Already Exist ').replace('employee.', 'in ').replace('for key',  
 ' ')))  
  
 def show\_product\_details(self, products):  
 # Convert the tuple to a dictionary  
 employee\_dict = {  
  
 'product\_id': products[0],  
 'product\_name': products[1],  
 'weight': products[2],  
 'pprice': products[3],  
 'available\_quantity': products[4],  
 }  
  
 # Create a Popup to display employee details  
 self.selected\_employee\_popup = Popup(title='Selected product', size\_hint=(None, None), auto\_dismiss=False,  
 background\_color=(0.004, 0.055, 0.102, 1.0), size=(500, 500))  
  
 # Create a GridLayout to organize employee details  
 layout = GridLayout(cols=2, spacing=5, padding=10)  
  
 # Add labels and employee details to the layout  
 for key, value in employee\_dict.items():  
 layout.add\_widget(Label(text=str(key), ))  
 layout.add\_widget(Label(text=str(value), ))  
  
 # Add an "Edit" button to allow editing employee details  
 edit\_button = Button(text='Edit', background\_color=(0.133, 0.855, 0.431, 1.0))  
 edit\_button.bind(on\_press=lambda instance: self.edit\_product1(employee\_dict))  
 layout.add\_widget(edit\_button)  
  
 # Add the layout to the popup  
 self.selected\_employee\_popup.content = layout  
  
 # Open the popup with employee details  
 self.selected\_employee\_popup.open()  
  
 def edit\_product1(self, product):  
 self.selected\_employee\_popup.dismiss()  
 values = list(product.values())  
 # Create a GridLayout to organize input fields  
 layout = GridLayout(cols=2, spacing=10, padding=10, background\_color=(0.004, 0.055, 0.102, 1.0))  
 # Add labels and input fields for each attribute  
 layout.add\_widget(Label(text='Product\_id:', ))  
 self.productid = TextInput(multiline=False, text=str(values[0]),  
 background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.productid)  
  
 layout.add\_widget(Label(text='Product\_name:', ))  
 self.productname = TextInput(multiline=False, text=str(values[1]),  
 background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.productname)  
  
 layout.add\_widget(Label(text='Weight of product :', ))  
 self.Weightofproduct = TextInput(multiline=False, text=str(values[2]),  
 background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.Weightofproduct)  
  
 layout.add\_widget(Label(text='Price:', ))  
 self.priceofproduct = TextInput(multiline=False, text=str(values[3]),  
 background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.priceofproduct)  
  
 layout.add\_widget(Label(text='Quantity available:', ))  
 self.quantityavailable = TextInput(multiline=False, text=str(values[4]),  
 background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.quantityavailable)  
 # Create a submit button  
 submit\_buttonp = Button(text='Submit', background\_color=(0.133, 0.855, 0.431, 1.0))  
 submit\_buttonp.bind(on\_press=lambda instance: self.submit\_product1(values[0]))  
  
 layout.add\_widget(submit\_buttonp)  
  
 # Create a cancel button  
 cancel\_buttonp = Button(text='Close', background\_color=(0.133, 0.855, 0.431, 1.0))  
 cancel\_buttonp.bind(on\_press=self.dismiss\_popupp)  
 layout.add\_widget(cancel\_buttonp)  
  
 # Create a Popup with the layout and background color  
 self.popup11 = Popup(title='Edit product', content=layout, size\_hint=(None, None), size=(720, 720),  
 background\_color=(0.004, 0.055, 0.102, 1.0), auto\_dismiss=False)  
 self.popup11.open()  
  
 def submit\_product1(self, value):  
 # Retrieve employee data from input fields  
 p\_id = self.productid.text  
 p\_name = self.productname.text  
 weight\_p = self.Weightofproduct.text  
 price\_p = self.priceofproduct.text  
 quantity\_p = self.quantityavailable.text  
 # replacing the kg,grams  
 new\_weight = self.Weightofproduct.text.replace('kg', '').replace('grams', '')  
 # Perform validation checks  
 if not all(  
 [p\_id, p\_name, weight\_p, price\_p, quantity\_p]):  
 self.show\_error\_popup("All fields are required.")  
 return  
 if not p\_id.isdigit():  
 self.show\_error\_popup("Invalid id.\n id number must be number.")  
 return  
 if not isinstance(float(price\_p), float):  
 self.show\_error\_popup("Invalid price number.\n price number must be digits.")  
 return  
  
 weight\_parts = weight\_p.split(' ')  
 if len(weight\_parts) != 2:  
 self.show\_error\_popup("Invalid weight format.\n Please use format like '250 grams'.")  
 return  
  
 # Check if the first part is a number  
 if not weight\_parts[0].isdigit():  
 self.show\_error\_popup("Invalid weight format.\n First part of weight must be a number.")  
 return  
  
 # Check if the second part is a valid unit  
 if weight\_parts[1] not in ('kg', 'grams', 'liter', 'ml'):  
 self.show\_error\_popup("Invalid weight unit.\n Valid units are: kg, grams, liter, ml.")  
 return  
 if not quantity\_p.isdigit():  
 self.show\_error\_popup("Invalid quantity .\nquantity must be digit")  
 return  
 # If all validation checks pass, save employee information to the database  
 try:  
 # Establish a connection to MySQL database  
 conn = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Sahith@12",  
 database="grocerystore"  
 )  
 cursor = conn.cursor()  
  
 id = value  
  
 # Construct the UPDATE query  
 update\_query = """  
 UPDATE products  
 SET product\_id= %s, product\_name = %s, weight\_of\_product = %s, product\_price = %s,product\_available\_quantity = %s  
  
 WHERE product\_id = %s  
 """  
 cursor.execute(update\_query, (p\_id, p\_name, weight\_p, float(price\_p), float(quantity\_p), id  
 ))  
 conn.commit() # Commit the transaction  
  
 # Close the cursor and connection  
 cursor.close()  
 conn.close()  
  
 # Close the popup after update  
  
 # Show success popup  
 self.show\_success\_popup("product details \nupdated to database successfully.")  
 self.popup11.dismiss()  
  
 except mysql.connector.Error as e:  
 p = str(e)  
 self.show\_error\_popup("Failed to update product \n{}".format(  
 p[13:].replace('Duplicate entry', 'Already Exist ').replace('employee.', 'in ').replace('for key',  
 ' ')))  
  
 def add\_product(self):  
 # Create a GridLayout to organize input fields  
 layout = GridLayout(cols=2, spacing=10, padding=10, background\_color=(0.004, 0.055, 0.102, 1.0))  
  
 # Add labels and input fields for each attribute  
  
 layout.add\_widget(Label(text='Product\_name:', ))  
 productname = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(productname)  
  
 layout.add\_widget(Label(text='Weight of product :', ))  
 weightofproduct = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(weightofproduct)  
  
 layout.add\_widget(Label(text='Price:', ))  
 priceofproduct = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(priceofproduct)  
  
 layout.add\_widget(Label(text='Quantity available:', ))  
 quantityavailable = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(quantityavailable)  
  
 # Create a submit button  
 submit\_buttonp = Button(text='Submit', background\_color=(0.133, 0.855, 0.431, 1.0))  
 submit\_buttonp.bind(  
 on\_press=lambda instance: self.submit\_product2(productname.text, weightofproduct.text,  
 priceofproduct.text, quantityavailable.text))  
  
 layout.add\_widget(submit\_buttonp)  
  
 # Create a cancel button  
 cancel\_buttonp = Button(text='Close', background\_color=(0.133, 0.855, 0.431, 1.0))  
 cancel\_buttonp.bind(on\_press=self.dismiss\_popupp1)  
 layout.add\_widget(cancel\_buttonp)  
  
 # Create a Popup with the layout and background color  
 self.popup = Popup(title='Add Product', content=layout, size\_hint=(None, None), size=(720, 720),  
 background\_color=(0.004, 0.055, 0.102, 1.0), auto\_dismiss=False)  
 self.popup.open()  
  
 def submit\_product2(self, p\_name, weight\_p, price\_p, quantity\_p):  
 # Perform validation checks  
 if not all([p\_name, weight\_p, price\_p, quantity\_p]):  
 self.show\_error\_popup("All fields are required.")  
 return  
 if not isinstance(float(price\_p), float):  
 self.show\_error\_popup("Invalid price.\n price must be digits.")  
 return  
 weight\_parts = weight\_p.split(' ')  
 if len(weight\_parts) != 2:  
 self.show\_error\_popup("Invalid weight format.\n Please use format like '250 grams'.")  
 return  
  
 # Check if the first part is a number  
 if not weight\_parts[0].isdigit():  
 self.show\_error\_popup("Invalid weight format.\n First part of weight must be a number.")  
 return  
  
 # Check if the second part is a valid unit  
 if weight\_parts[1] not in ('kg', 'grams', 'liter', 'ml'):  
 self.show\_error\_popup("Invalid weight unit.\n Valid units are: kg, grams, liter, ml.")  
 return  
 if not quantity\_p.isdigit():  
 self.show\_error\_popup("Invalid quantity.\n quantity must be digit.")  
 return  
  
 try:  
 # Establish a connection to MySQL database  
 conn = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Sahith@12",  
 database="grocerystore"  
 )  
 cursor = conn.cursor()  
  
 # Check if the product ID already exists  
 cursor.execute("SELECT \* FROM products WHERE product\_name = %s", (p\_name,))  
 existing\_product = cursor.fetchone()  
  
 if existing\_product:  
 self.show\_error\_popup("Product with ID {} already exists.".format(p\_name))  
 return  
  
 # Construct the INSERT query  
 insert\_query = """  
 INSERT INTO products (product\_name, weight\_of\_product, product\_price, product\_available\_quantity)  
 VALUES (%s, %s, %s, %s)  
 """  
 cursor.execute(insert\_query, (p\_name, weight\_p, price\_p, quantity\_p))  
 conn.commit() # Commit the transaction  
  
 # Close the cursor and connection  
 cursor.close()  
 conn.close()

# Close the popup after successful addition  
 self.dismiss\_popupp1()  
  
 # Show success popup  
 self.show\_success\_popup("Product added to database successfully.")  
  
 except mysql.connector.Error as e:  
 p = str(e)  
 self.show\_error\_popup("Failed to add product.\n{}".format(  
 p[13:].replace('Duplicate entry', 'Already Exist ').replace('employee.', 'in ').replace('for key',  
 ' ')))  
  
 def show\_error\_popup(self, message):  
 # Display an error popup with the given message  
 error\_popup = Popup(title='Error', content=Label(text=message), size\_hint=(None, None), size=(300, 200))  
 error\_popup.open()  
  
 def dismiss\_popupp(self, instance=None):  
 self.popup11.dismiss()

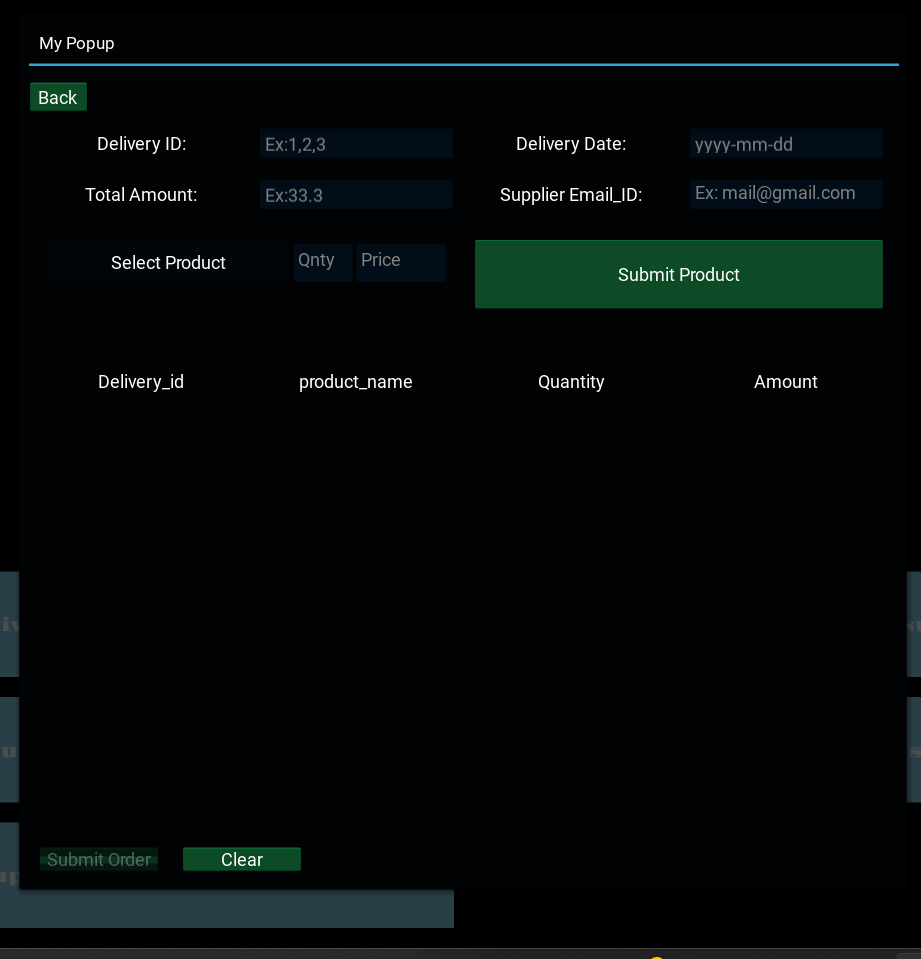
def dismiss\_popupp1(self, instance=None):  
 self.popup.dismiss()  
  
 def show\_success\_popup(self, message):  
 # Display a success popup with the given message  
 success\_popup = Popup(title='Success', content=Label(text=message), size\_hint=(None, None), size=(300, 200))  
 success\_popup.open()

**Delivery and supplier:**



Here the user can add deliveries, suppliers, edit the details of suppliers and delete the suppliers.

**Add delivery:**



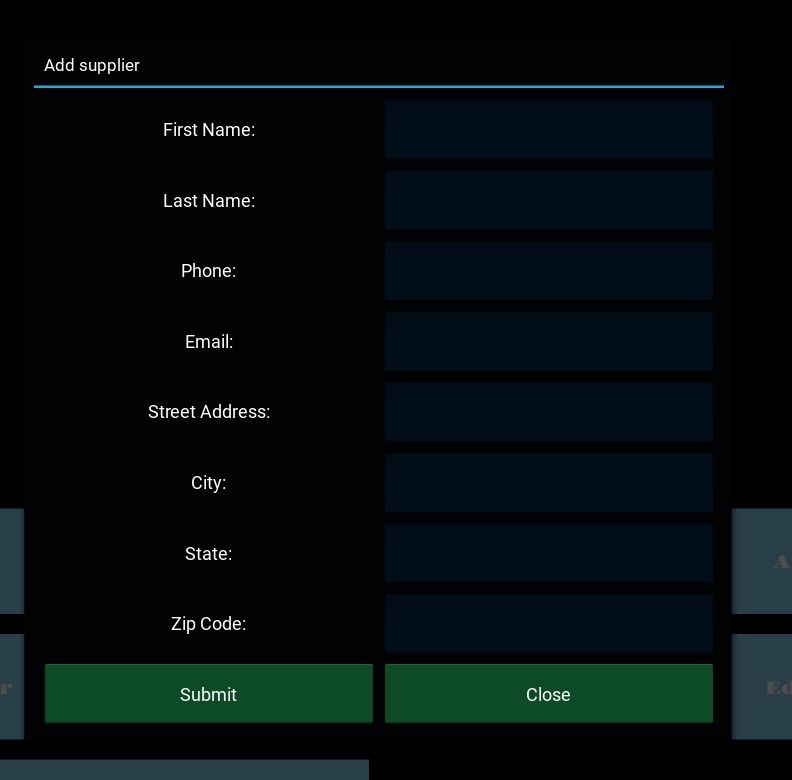
When the user hits “Add delivery” button, a pop up is displayed where the details of product and supplier email id are required to be entered. Once the required details are entered, the user can click on the submit product button to add the product for delivery. After adding all the necessary products, the user can submit the orders by clicking on the submit order button. A clear button is provided to clear all the fields that have entered and a back button to close the pop-up window.

**All suppliers:**



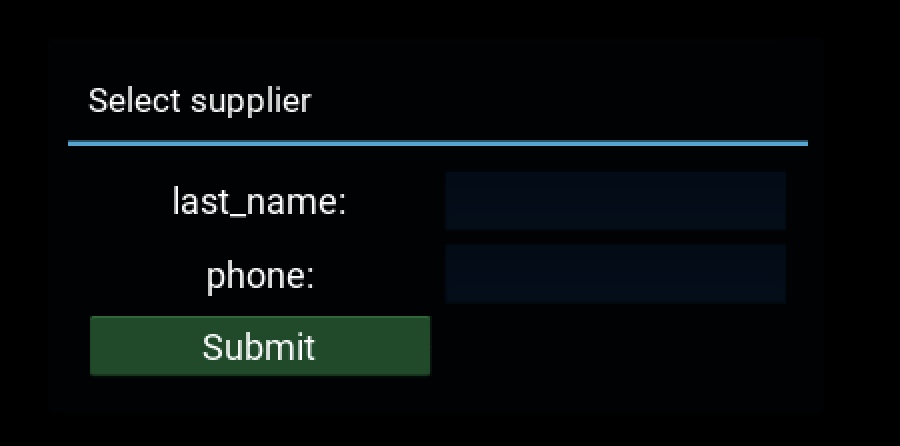
The user can find all the details of suppliers by clicking on All suppliers button. A close button is used to close the pop-up window.

**Add New supplier:**



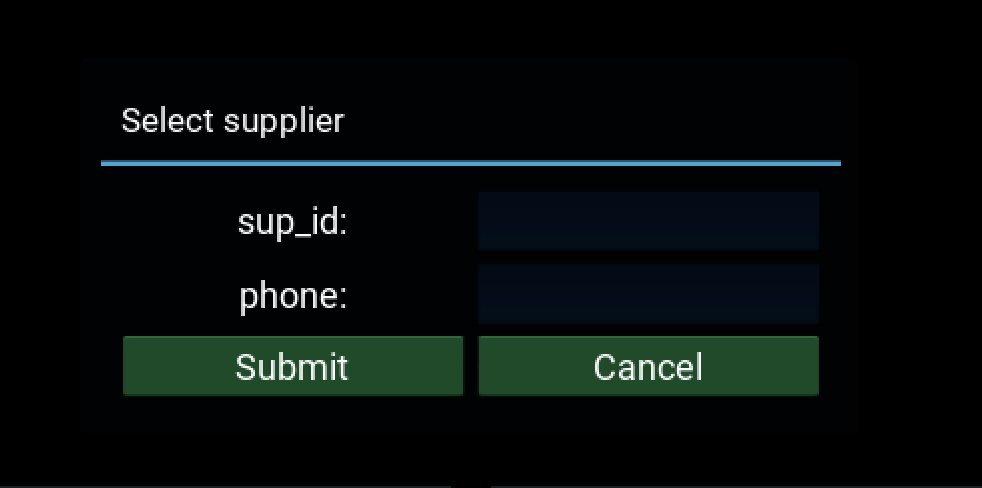
Here we need to provide all the necessary information about the supplier and once the user clicks on the submit button the new supplier details will be added to the database.

**Edit supplier:**



For editing the details of any supplier, the user needs to provide the last name and phone number of that supplier. Once the details are provided the user can edit the supplier.

**Delete supplier:**



To delete a supplier from database, the user needs to provide the supplier id and phone number.

**Code:**

**Delivery.py:**

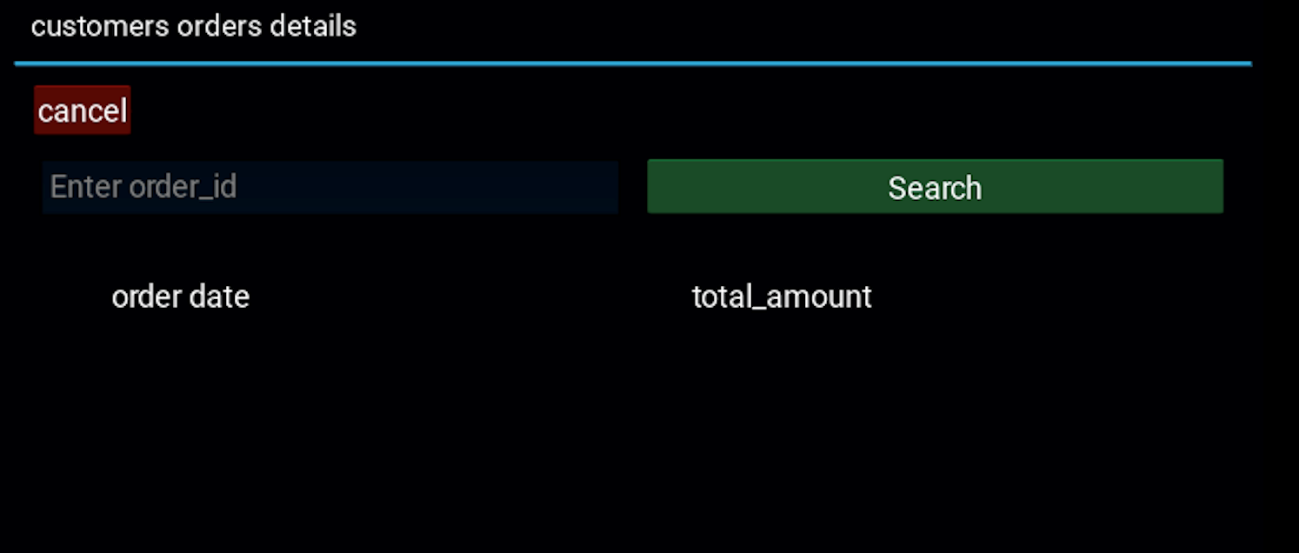
import mysql.connector  
from kivy.uix.screenmanager import Screen  
from datetime import datetime  
import re  
from kivy.uix.gridlayout import GridLayout  
from kivy.uix.boxlayout import BoxLayout  
from kivy.uix.label import Label  
from kivy.uix.button import Button  
from kivy.uix.popup import Popup  
from kivy.uix.textinput import TextInput  
from kivy.uix.dropdown import DropDown  
from kivy.uix.scrollview import ScrollView  
import mysql.connector  
class Delivery(Screen):  
 def show\_all\_suppliers(self):  
 # Establish a connection to your MySQL database  
 conn = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Sahith@12",  
 port='3306',  
 database='grocerystore'  
 )  
 cursor = conn.cursor()  
  
 # Fetch specific columns (first\_name, last\_name, sup\_id, phone) from the database  
 cursor.execute("SELECT first\_name, last\_name, sup\_id, phone FROM supplier")  
 supplier\_data = cursor.fetchall()  
  
 # Close the database connection  
 cursor.close()  
 conn.close()  
  
 # Create a BoxLayout to organize the labels and the scroll view  
 content\_layout = BoxLayout(orientation='vertical', padding=10, spacing=40,  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
  
 # Create a GridLayout for the labels  
 labels\_layout = GridLayout(cols=4, size\_hint\_y=None, height='40dp', padding=10, spacing=40)  
  
 # Set fixed widths for columns  
 column\_widths = [150, 150, 150, 150]  
  
 # Add column labels  
 labels\_layout.add\_widget(Label(text='First Name', size\_hint\_x=None, width=column\_widths[0]))  
 labels\_layout.add\_widget(Label(text='Last Name', size\_hint\_x=None, width=column\_widths[1]))  
 labels\_layout.add\_widget(Label(text='supplier id', size\_hint\_x=None, width=column\_widths[2]))  
 labels\_layout.add\_widget(Label(text='Phone no.', size\_hint\_x=None, width=column\_widths[3]))  
  
 # Add labels layout to content layout  
 content\_layout.add\_widget(labels\_layout)  
  
 # Create a ScrollView with GridLayout inside to allow scrolling  
 layout = GridLayout(cols=4, size\_hint\_y=None, padding=10, spacing=40)  
 layout.bind(minimum\_height=layout.setter('height'))  
 # Add data  
 for supplier in supplier\_data:  
 first\_name\_label = Label(text=supplier[0], size\_hint\_x=None, width=column\_widths[0], height='40dp',  
 )  
 last\_name\_label = Label(text=supplier[1], size\_hint\_x=None, width=column\_widths[1], height='40dp',  
 )  
 emp\_id\_label = Label(text=str(supplier[2]), size\_hint\_x=None, width=column\_widths[2], height='40dp',  
 )  
 phone\_label = Label(text=supplier[3], size\_hint\_x=None, width=column\_widths[3], height='40dp',  
 )  
  
 layout.add\_widget(first\_name\_label)  
 layout.add\_widget(last\_name\_label)  
 layout.add\_widget(emp\_id\_label)  
 layout.add\_widget(phone\_label)  
  
 # Create a ScrollView with GridLayout inside to allow scrolling  
 scroll\_view = ScrollView()  
 scroll\_view.add\_widget(layout)  
  
 # Add scroll view to content layout  
 content\_layout.add\_widget(scroll\_view)  
  
 # Create a close button  
 close\_button = Button(text='Close', size\_hint=(None, None), size=(100, 50),  
 background\_color=(0.133, 0.855, 0.431, 1.0))  
 close\_button.bind(on\_press=self.dismiss\_popup)  
  
 # Add close button to content layout  
 content\_layout.add\_widget(close\_button)  
  
 # Create a Popup with the content layout  
 self.popup = Popup(title='All suppliers', content=content\_layout, size\_hint=(None, None), size=(800, 600),  
 background\_color=(0.004, 0.055, 0.102, 1.0), auto\_dismiss=False)  
 self.popup.open()  
  
 def dismiss\_popup(self, instance):  
 if hasattr(self, 'popup') and self.popup:  
 self.popup.dismiss()  
  
 def add\_supplier(self):  
 # Create a GridLayout to organize input fields  
 layout = GridLayout(cols=2, spacing=10, padding=10, background\_color=(0.004, 0.055, 0.102, 1.0))  
  
 # Add labels and input fields for each attribute  
 layout.add\_widget(Label(text='First Name:', ))  
 self.first\_name\_input = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.first\_name\_input)  
  
 layout.add\_widget(Label(text='Last Name:', ))  
 self.last\_name\_input = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.last\_name\_input)  
  
 layout.add\_widget(Label(text='Phone:', ))  
 self.phone\_input = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.phone\_input)  
  
 layout.add\_widget(Label(text='Email:', ))  
 self.email\_input = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.email\_input)  
  
 layout.add\_widget(Label(text='Street Address:', ))  
 self.street\_address\_input = TextInput(multiline=True, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.street\_address\_input)  
  
 layout.add\_widget(Label(text='City:', ))  
 self.city\_input = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.city\_input)  
  
 layout.add\_widget(Label(text='State:', ))  
 self.state\_input = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.state\_input)  
  
 layout.add\_widget(Label(text='Zip Code:', ))  
 self.Zip\_input = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.Zip\_input)  
  
 # Create a submit button  
 submit\_button = Button(text='Submit', background\_color=(0.133, 0.855, 0.431, 1.0))  
 submit\_button.bind(on\_press=self.submit\_supplier)  
 layout.add\_widget(submit\_button)  
  
 # Create a cancel button  
 self.cancel\_button = Button(text='Close', background\_color=(0.133, 0.855, 0.431, 1.0))  
 self.cancel\_button.bind(on\_press=self.dismiss\_popup1)  
 layout.add\_widget(self.cancel\_button)  
  
 # Create a Popup with the layout and background color  
 self.popup1 = Popup(title='Add supplier', content=layout, size\_hint=(None, None), size=(720, 720),  
 background\_color=(0.004, 0.055, 0.102, 1.0), auto\_dismiss=False)  
 self.popup1.open()  
  
 def submit\_supplier(self, instance):  
 # Retrieve employee data from input fields  
 first\_name = self.first\_name\_input.text  
 last\_name = self.last\_name\_input.text  
 phone = self.phone\_input.text  
 email = self.email\_input.text  
 street\_address = self.street\_address\_input.text  
 zip = self.Zip\_input.text  
 city = self.city\_input.text  
 state = self.state\_input.text  
  
 # Perform validation checks  
 if not all([first\_name, last\_name, phone, email, street\_address, city, state, zip]):  
 self.show\_error\_popup("All fields are required.")  
 return  
 if len(phone) != 10 or not phone.isdigit():  
 self.show\_error\_popup("Invalid phone number.\n Phone number must be 10 digits.")  
 return  
  
 if not [email.endswith('@gmail.com](mailto:email.endswith('@gmail.com)') or not email[0].isalpha():  
 self.show\_error\_popup("Invalid email address. \nEmail must end with @gmail.com")  
 return  
 if len(zip) != 6 or not zip.isdigit():  
 self.show\_error\_popup("Invalid zip number.\n zip number must be 6 digits.")  
 return  
 # If all validation checks pass, save employee information to the database  
 try:  
 # Establish a connection to MySQL database  
 conn = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Sahith@12",  
 database="grocerystore"  
 )  
 cursor = conn.cursor()  
 number = self.add\_dashes\_to\_number\_with\_existing\_dashes(phone)  
 # Execute INSERT query to insert employee details into the database  
 insert\_query = "INSERT INTO supplier (first\_name, last\_name, phone,zip, emial, street, city, state) VALUES (%s,%s, %s, %s, %s, %s, %s, %s)"  
 cursor.execute(insert\_query, (  
 first\_name, last\_name, number, zip, email, street\_address, city, state))  
  
 # Commit changes and close connection  
 conn.commit()  
 cursor.close()  
 conn.close()  
  
 # Show success popup  
 self.show\_success\_popup("supplier details saved to database successfully.")  
  
 except mysql.connector.Error as e:  
 p = str(e)  
 self.show\_error\_popup("Failed to add supplier \n{}".format(  
 p[13:].replace('Duplicate entry', 'Already Exist ').replace('employee.', 'in ').replace('for key',  
 ' ')))  
  
 def add\_dashes\_to\_number\_with\_existing\_dashes(self, number):  
 # Convert number to string  
 number\_str = str(number)  
  
 # Use regular expression to add dashes after every three digits for the first two groups  
 # and after every four digits for the last group  
 formatted\_number = re.sub(r'(\d{3})(\d{3})(\d{4})', r'\1-\2-\3', number\_str)  
  
 return formatted\_number  
  
 def show\_success\_popup(self, message):  
 # Display a success popup with the given message  
 success\_popup = Popup(title='Success', content=Label(text=message), size\_hint=(None, None), size=(300, 200))  
 success\_popup.open()  
  
 def show\_error\_popup(self, message):  
 # Display an error popup with the given message  
 popup\_width = len(message) \* 10 # Adjust the multiplier based on your preference  
 popup\_height = max(len(message) // 15,  
 1) \* 40 # Adjust the divisor and multiplier based on your preference  
  
 error\_popup = Popup(title='Error', content=Label(text=message), size\_hint=(None, None),  
 size=(popup\_width, popup\_height))  
 error\_popup.open()  
  
 def dismiss\_popup\_1(self, instance=None):  
 self.popup1.dismiss()  
  
 def edit\_supplier(self):  
 # Create a GridLayout to organize input fields  
 layout = GridLayout(cols=2, spacing=5, padding=10)  
  
 # Add labels and input fields for user ID and password  
 layout.add\_widget(Label(text='last\_name:', ))  
 self.lastname\_input = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.lastname\_input)  
  
 layout.add\_widget(Label(text='phone:', ))  
 self.phonenumber\_input = TextInput(multiline=False, password=True,  
 background\_color=(0.004, 0.055, 0.102, 1.0), cursor\_color=(1, 1, 1, 1),  
 foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.phonenumber\_input)  
 # Create a submit button  
 submit\_button = Button(text='Submit', background\_color=(0.133, 0.855, 0.431, 1.0))  
 submit\_button.bind(on\_press=self.authenticate\_supplier)  
 layout.add\_widget(submit\_button)  
  
 # Create a Popup with the layout  
 self.popup = Popup(title='Select supplier', content=layout, size\_hint=(None, None),  
 background\_color=(0.004, 0.055, 0.102, 1.0), size=(400, 200))  
 self.popup.open()  
  
 def authenticate\_supplier(self, instance):  
 # Retrieve user ID and password from input fields  
 lastname = self.lastname\_input.text.strip()  
 phone = self.phonenumber\_input.text.strip()  
 if not all([lastname, phone]):  
 self.show\_error\_popup1("All fields are required.")  
 return  
 if len(phone) != 10 or not phone.isdigit():  
 self.show\_error\_popup1("Invalid phone number.\n Phone number must be 10 digits.")  
 return  
 # Perform authentication against the database  
 try:  
 # Establish a connection to MySQL database  
 conn = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Sahith@12",  
 database="grocerystore"  
 )  
 cursor = conn.cursor()  
  
 # Execute SELECT query to retrieve employee details based on user ID and password  
 select\_query = "SELECT \* FROM supplier WHERE last\_name = %s AND phone = %s"  
 cursor.execute(select\_query, (lastname, self.add\_dashes\_to\_number\_with\_existing\_dashes(phone)))  
 sup = cursor.fetchone()  
  
 if sup:  
 # Employee found, close current popup and display details in another popup  
 self.popup.dismiss()  
 self.show\_supplier\_details(sup)  
 else:  
 # Employee not found, show error message  
 self.show\_error\_popup1("Invalid lastname or phone number.")  
  
 cursor.close()  
 conn.close()  
  
 except mysql.connector.Error as e:  
 p = str(e)  
 self.show\_error\_popup("Failed to authenticate \n{}".format(  
 p[13:].replace('Duplicate entry', 'Already Exist ').replace('supplier.', 'in ').replace('for key',  
 ' ')))  
  
 def show\_supplier\_details(self, employee):  
 # Convert the tuple to a dictionary  
 employee\_dict = {  
  
 'first\_name': employee[1],  
 'last\_name': employee[2],  
 'phone': employee[3],  
 'email': employee[4],  
 'street\_address': employee[5],  
 'city': employee[6],  
 'state': employee[7],  
 'zip': employee[8],  
 'id': employee[0]  
 }  
  
 # Create a Popup to display employee details  
 self.selected\_employee\_popup = Popup(title='Selected supplier', size\_hint=(None, None), auto\_dismiss=False,  
 background\_color=(0.004, 0.055, 0.102, 1.0), size=(500, 500))  
  
 # Create a GridLayout to organize employee details  
 layout = GridLayout(cols=2, spacing=5, padding=10)  
  
 # Add labels and employee details to the layout  
 for key, value in employee\_dict.items():  
 layout.add\_widget(Label(text=str(key), ))  
 layout.add\_widget(Label(text=str(value), ))  
  
 # Add an "Edit" button to allow editing employee details  
 edit\_button = Button(text='Edit', background\_color=(0.133, 0.855, 0.431, 1.0))  
 edit\_button.bind(on\_press=lambda instance: self.edit\_supplier1(employee\_dict))  
 layout.add\_widget(edit\_button)  
  
 # Add the layout to the popup  
 self.selected\_employee\_popup.content = layout  
  
 # Open the popup with employee details  
 self.selected\_employee\_popup.open()  
  
 def edit\_supplier1(self, employee):  
 self.selected\_employee\_popup.dismiss()  
 values = list(employee.values())  
 # Create a GridLayout to organize input fields  
 layout = GridLayout(cols=2, spacing=10, padding=10, background\_color=(0.004, 0.055, 0.102, 1.0))  
 print(values)  
 # Add labels and input fields for each attribute  
 layout.add\_widget(Label(text='First Name:', ))  
 self.first\_name\_input1 = TextInput(multiline=False, text=str(values[0]),  
 background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.first\_name\_input1)  
  
 layout.add\_widget(Label(text='Last Name:', ))  
 self.last\_name\_input1 = TextInput(multiline=False, text=str(values[1]),  
 background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.last\_name\_input1)  
  
 layout.add\_widget(Label(text='Phone:', ))  
 self.phone\_input1 = TextInput(multiline=False, text=str(values[2].replace('-', '')),  
 background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.phone\_input1)  
  
 layout.add\_widget(Label(text='Email:', ))  
 self.email\_input1 = TextInput(multiline=False, text=str(values[3]),  
 background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.email\_input1)  
  
 layout.add\_widget(Label(text='Street Address:', ))  
 self.street\_address\_input1 = TextInput(multiline=True, text=str(values[4]),  
 background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.street\_address\_input1)  
  
 layout.add\_widget(Label(text='City:', ))  
 self.city\_input1 = TextInput(multiline=False, text=str(values[5]),  
 background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.city\_input1)  
  
 layout.add\_widget(Label(text='State:', ))  
 self.state\_input1 = TextInput(multiline=False, text=str(values[6]),  
 background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.state\_input1)  
  
 layout.add\_widget(Label(text='Zip Code:', ))  
 self.Zip\_input1 = TextInput(multiline=False, text=str(values[7]),  
 background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.Zip\_input1)  
  
 # Create a submit button  
 submit\_button1 = Button(text='Submit', background\_color=(0.133, 0.855, 0.431, 1.0))  
 submit\_button1.bind(on\_press=lambda instance: self.submit\_supplier1(values[8]))  
  
 layout.add\_widget(submit\_button1)  
  
 # Create a cancel button  
 cancel\_button1 = Button(text='Close', background\_color=(0.133, 0.855, 0.431, 1.0))  
 cancel\_button1.bind(on\_press=self.dismiss\_popup11)  
 layout.add\_widget(cancel\_button1)  
  
 # Create a Popup with the layout and background color  
 self.popup11 = Popup(title='Edit Employee', content=layout, size\_hint=(None, None), size=(720, 720),  
 background\_color=(0.004, 0.055, 0.102, 1.0), auto\_dismiss=False)  
 self.popup11.open()  
  
 def submit\_supplier1(self, value):  
 # Retrieve employee data from input fields  
 first\_name = self.first\_name\_input1.text  
 last\_name = self.last\_name\_input1.text  
 phone = self.phone\_input1.text  
 email = self.email\_input1.text  
 street\_address = self.street\_address\_input1.text  
 zip = self.Zip\_input1.text  
 city = self.city\_input1.text  
 state = self.state\_input1.text  
  
 # Perform validation checks  
 if not all(  
 [first\_name, last\_name, phone, email, street\_address, city, state, zip]):  
 self.show\_error\_popup("All fields are required.")  
 return  
 if len(phone) != 10 or not phone.isdigit():  
 self.show\_error\_popup("Invalid phone number.\n Phone number must be 10 digits.")  
 return  
  
 if not [email.endswith('@gmail.com](mailto:email.endswith('@gmail.com)') or not email[0].isalpha():  
 self.show\_error\_popup("Invalid email address. \nEmail must end with @gmail.com")  
 return  
 if len(zip) != 6 or not zip.isdigit():  
 self.show\_error\_popup("Invalid zip number.\n zip number must be 6 digits.")  
 return  
 # If all validation checks pass, save employee information to the database  
 try:  
 # Establish a connection to MySQL database  
 conn = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Sahith@12",  
 database="grocerystore"  
 )  
 id = value  
 phone = self.add\_dashes\_to\_number\_with\_existing\_dashes(phone)  
 cursor = conn.cursor()  
 # Construct the UPDATE query  
 update\_query = """  
 UPDATE supplier  
 SET first\_name = %s, last\_name = %s, phone = %s, emial = %s,  
 street = %s, zip = %s, city = %s, state = %s  
  
 WHERE sup\_id = %s  
 """  
 cursor.execute(update\_query, (  
 first\_name, last\_name, phone, email, street\_address, zip,  
 city, state, id  
 ))  
 conn.commit() # Commit the transaction  
  
 # Close the cursor and connection  
 cursor.close()  
 conn.close()  
  
 # Close the popup after update  
  
 # Show success popup  
 self.show\_success\_popup("supplier details \nupdated to database successfully.")  
 self.popup11.dismiss()  
  
 except mysql.connector.Error as e:  
 p = str(e)  
 self.show\_error\_popup("Failed to add supplier\n{}".format(  
 p[13:].replace('Duplicate entry', 'Already Exist ').replace('employee.', 'in ').replace('for key',  
 ' ')))  
  
 def show\_error\_popup1(self, message):  
 # Display an error popup with the given message  
 error\_popup = Popup(title='Error', content=Label(text=message), size\_hint=(None, None), size=(300, 200))  
 error\_popup.open()  
  
 def delete\_supplier(self):  
  
 # Create a GridLayout to organize input fields  
 layout = GridLayout(cols=2, spacing=5, padding=10)  
  
 # Add labels and input fields for user ID and password  
 layout.add\_widget(Label(text='sup\_id:', ))  
 self.emp\_id\_d = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),

cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.emp\_id\_d)  
  
 layout.add\_widget(Label(text='phone:', ))  
 self.phone\_d = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.phone\_d)  
 # Create a submit button  
 submit\_button = Button(text='Submit', background\_color=(0.133, 0.855, 0.431, 1.0))  
 submit\_button.bind(on\_press=self.dele\_supplier)  
 layout.add\_widget(submit\_button)  
 cancel = Button(text='Cancel', background\_color=(0.133, 0.855, 0.431, 1.0))  
 cancel.bind(on\_press=self.dismis)  
 layout.add\_widget(cancel)  
  
 # Create a Popup with the layout  
 self.popup111 = Popup(title='Select supplier', content=layout, auto\_dismiss=False, size\_hint=(None, None),  
 background\_color=(0.004, 0.055, 0.102, 1.0), size=(400, 200))  
 self.popup111.open()  
  
 def dismiss\_popup11(self, instance=None):  
 self.popup11.dismiss()  
  
 def dele\_supplier(self, instance):  
 # Retrieve user ID and password from input fields  
 emp\_id1 = self.emp\_id\_d.text.strip()  
 passs1 = self.phone\_d.text.strip()  
 if not all([emp\_id1, passs1]):  
 self.show\_error\_popup1("All fields are required.")  
 return  
 if len(passs1) != 10 or not emp\_id1.isdigit():  
 self.show\_error\_popup1("Invalid suplier id number.\n supplier id must be digit")  
 return  
 # Perform authentication against the database  
 try:  
 # Establish a connection to MySQL database  
 conn = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Sahith@12",  
 database="grocerystore"  
 )  
 cursor = conn.cursor()  
 phone1 = self.add\_dashes\_to\_number\_with\_existing\_dashes(passs1)  
 # Execute SELECT query to retrieve employee details based on user ID and password  
 select\_query = "DELETE FROM supplier WHERE sup\_id = %s AND phone = %s"  
 cursor.execute(select\_query, (emp\_id1, phone1))  
 conn.commit()  
 cursor.close()  
 conn.close()  
  
 # Show success popup  
 if cursor.rowcount != 0:  
 self.show\_success\_popup("Employee details Deleted to database successfully.")  
 self.popup111.dismiss()  
 else:  
 self.show\_error\_popup1('employee details not exist.')  
  
 except mysql.connector.Error as e:  
 p = str(e)  
 self.show\_error\_popup("Failed to authenticate \n{}".format(  
 p[13:].replace('Duplicate entry', 'Already Exist ').replace('employee.', 'in ').replace('for key',  
 ' ')))  
  
 def dismis(self, instance=None):  
 self.popup111.dismiss()  
  
 def fetch\_products\_from\_database(self):  
 # Connect to the MySQL database  
 connection = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Sahith@12",  
 database="grocerystore"  
 )  
  
 # Create a cursor object to execute queries  
 cursor = connection.cursor()  
  
 # Execute the query to fetch products  
 cursor.execute("SELECT product\_name FROM products order by product\_name")  
  
 # Fetch all the products  
 products = cursor.fetchall()  
  
 # Close the cursor and connection  
 cursor.close()  
 connection.close()  
  
 # Extract the product names from the fetched data  
 product\_names = [product[0] for product in products]  
  
 return product\_names  
  
 def delivery\_details(self):  
 # Fetch products from the database  
 products = self.fetch\_products\_from\_database()  
  
 # First layout with back button  
 first\_layout = BoxLayout(orientation='vertical', size\_hint\_y=0.10, background\_color=(0.004, 0.055, 0.102, 1.0))  
 back\_button = Button(text="Back", background\_color=(0.133, 0.855, 0.431, 1.0), size\_hint=(None, None),  
 size=(59, 30))  
 back\_button.bind(on\_press=self.dismiss\_popup\_1)  
 first\_layout.add\_widget(back\_button)  
  
 # Second layout with inputs  
 second\_layout = GridLayout(cols=4, padding=15, spacing=20, size\_hint\_y=0.25)  
 self.delivery\_id\_input = TextInput(multiline=False, hint\_text="Ex:1,2,3",  
 background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 self.delivery\_date\_input = TextInput(multiline=False, hint\_text="yyyy-mm-dd",  
 background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 self.total\_amount\_input = TextInput(multiline=False, hint\_text="Ex:33.3",  
 background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 self.supplier\_id\_input = TextInput(multiline=False, hint\_text="Ex: [mail@gmail.com](mailto:mail@gmail.com)",  
 background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
  
 second\_layout.add\_widget(Label(text="Delivery ID:"))  
 second\_layout.add\_widget(self.delivery\_id\_input)  
 second\_layout.add\_widget(Label(text="Delivery Date:"))  
 second\_layout.add\_widget(self.delivery\_date\_input)  
 second\_layout.add\_widget(Label(text="Total Amount:"))  
 second\_layout.add\_widget(self.total\_amount\_input)  
 second\_layout.add\_widget(Label(text="Supplier Email\_ID:"))  
 second\_layout.add\_widget(self.supplier\_id\_input)  
  
 # Third layout with dropdown and input fields  
 third\_layout = GridLayout(cols=5, padding=3, spacing=2, size\_hint\_y=0.60,  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
 dropdown\_button = Button(text="Select Product", size\_hint\_y=None, size\_hint\_x=0.8, height=40,  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
 dropdown = DropDown()  
 for product in products:  
 btn = Button(text=product, size\_hint\_x=0.8, size\_hint\_y=None, height=40)  
 btn.bind(on\_release=lambda btn: dropdown.select(btn.text))  
 dropdown.add\_widget(btn)  
 dropdown\_button.bind(on\_release=dropdown.open)  
 dropdown.bind(on\_select=lambda instance, x: setattr(dropdown\_button, 'text', x))  
 third\_layout.add\_widget(dropdown\_button)  
 self.quantity\_input = TextInput(hint\_text="Qnty", multiline=False, size\_hint\_x=0.2, size\_hint\_y=None,  
 height=40, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 self.amount\_of\_product\_input = TextInput(hint\_text="Price", multiline=False, size\_hint\_x=0.3, size\_hint\_y=None,  
 height=40, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 third\_layout.add\_widget(self.quantity\_input)  
 third\_layout.add\_widget(self.amount\_of\_product\_input)  
 second\_layout2 = GridLayout(cols=4, padding=15, spacing=20, size\_hint\_y=0.25,  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
 second\_layout2.add\_widget(Label(text='Delivery\_id'))  
 second\_layout2.add\_widget(Label(text='product\_name'))  
 second\_layout2.add\_widget(Label(text='Quantity'))  
 second\_layout2.add\_widget(Label(text='Amount'))  
 # Fourth layout with ScrollView  
 fourth\_layout = GridLayout(cols=4, padding=15, spacing=20, size\_hint\_y=None,  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
 fourth\_layout.bind(minimum\_height=fourth\_layout.setter('height'))  
  
 # Fifth layout with "Submit Order" and "Clear" buttons  
 fifth\_layout = BoxLayout(orientation='horizontal', padding=10, spacing=23, size\_hint\_y=0.10,  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
 submit\_order\_button = Button(text="Submit Order", padding=15, size\_hint\_x=None, width=120,  
 background\_color=(0.133, 0.855, 0.431, 1.0))  
  
 # List to store row-by-row values  
 row\_values = []  
  
 def on\_submit\_order(instance):  
 # Clear the list before adding new values  
 row\_values.clear()  
 # Iterate over the widgets in fourth\_layout and add their text values to row\_values list  
 for i in range(len(fourth\_layout.children) // 4):  
 row = []  
 for j in range(4):  
 widget = fourth\_layout.children[i \* 4 + j]  
 row.append(widget.text)  
 row\_values.append(row)  
 try:  
 # Establish a connection to MySQL database  
 conn = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Sahith@12",  
 database="grocerystore"  
 )  
 cursor = conn.cursor()  
 query = "SELECT sup\_id FROM supplier WHERE emial = %s"  
 cursor.execute(query, [self.supplier\_id\_input.text, ])  
 id = cursor.fetchall()  
 if len(id) == 0:  
 self.show\_error\_popup\_1(  
 "Failed to add delivery. Error: {}".format(str('email entered is wrong')))  
 return  
 cursor.execute('Select \* from delivery where delivery\_id=%s', (self.delivery\_id\_input.text,))  
 e = cursor.fetchall()  
  
 if len(e) != 0:  
 self.show\_error\_popup\_1(  
 "Failed to add delivery. Error: {}".format(str('this delivery already exist')))  
 return  
 # Execute INSERT query to insert delivery details into the database  
 delivery\_query = "INSERT INTO delivery (delivery\_id, delivery\_date, total\_amount, sup\_id) VALUES (%s, %s, %s,%s)"  
 cursor.execute(delivery\_query, (  
 self.delivery\_id\_input.text, self.delivery\_date\_input.text, self.total\_amount\_input.text, id[0][0]))  
 # Commit changes  
 conn.commit()  
  
 # Execute INSERT queries to insert detail delivery details into the database  
 for r in row\_values:  
 p\_id = "SELECT product\_id,product\_available\_quantity FROM products WHERE product\_name = %s"  
 cursor.execute(p\_id, (r[2],))  
 r\_data = cursor.fetchall()  
 conn.commit()  
 sum = float(r\_data[0][1]) + float(r[1])  
  
 u\_q = 'UPDATE PRODUCTS SET product\_available\_quantity=%s where product\_id=%s'  
 cursor.execute(u\_q, (sum, r\_data[0][0]))  
 detail\_delivery\_query = "INSERT INTO detail\_delivery (delivery\_id, product\_id, quantity, amount\_of\_product) VALUES (%s,%s, %s, %s)"  
 detail\_delivery\_data = (r[3], r\_data[0][0], r[1], r[0])  
 cursor.execute(detail\_delivery\_query, detail\_delivery\_data)  
 # Commit changes for each row  
 conn.commit()  
  
 # Close cursor and connection  
 cursor.close()  
 conn.close()  
  
 # Show success popup  
 self.show\_success\_popup\_1("Delivery details saved to database successfully.")  
  
 except mysql.connector.Error as e:  
 # Show error popup  
 print(e)  
 self.show\_error\_popup\_1("Failed to add delivery. Error: {}".format(str(e)))  
  
 submit\_order\_button.bind(on\_release=on\_submit\_order)  
 submit\_order\_button.disabled = True  
 fifth\_layout.add\_widget(submit\_order\_button)  
  
 def on\_clear(instance):  
 # Clear the content of fourth layout  
 fourth\_layout.clear\_widgets()  
 # Enable input fields in layout 2  
  
 self.delivery\_id\_input.disabled = False  
 self.delivery\_date\_input.disabled = False  
 self.total\_amount\_input.disabled = False  
 self.supplier\_id\_input.disabled = False  
 self.delivery\_id\_input.clear\_widgets()  
 self.delivery\_date\_input.clear\_widgets()  
 self.total\_amount\_input.clear\_widgets()  
 self.supplier\_id\_input.clear\_widgets()  
 self.quantity\_input.clear\_widgets()  
 self.amount\_of\_product\_input.clear\_widgets()  
 clear\_button = Button(text="Clear", background\_color=(0.133, 0.855, 0.431, 1.0), padding=15, size\_hint\_x=None,  
 width=120)  
 clear\_button.bind(on\_release=on\_clear)  
 fifth\_layout.add\_widget(clear\_button)  
 submit\_order\_button.disabled = True  
  
 # Disable input fields in layout 2 when Submit Product button is clicked  
 def on\_submit(instance):  
 # Check if all fields in layout 2 are filled  
 if not all([self.delivery\_id\_input.text, self.delivery\_date\_input.text, self.total\_amount\_input.text,  
 self.supplier\_id\_input.text]):  
 show\_error\_popup("Please fill in all fields .")  
 return  
  
 # Check if the dropdown, quantity, and amount fields in layout 3 are filled  
 if dropdown\_button.text == "Select Product" or not self.quantity\_input.text or not self.amount\_of\_product\_input.text:  
 show\_error\_popup("Please select a product and fill in the quantity and amount fields ")  
 return  
  
 # Perform additional validations  
 delivery\_id = self.delivery\_id\_input.text  
 delivery\_date = self.delivery\_date\_input.text  
 total\_amount = self.total\_amount\_input.text  
 supplier\_id = self.supplier\_id\_input.text  
 quantity = self.quantity\_input.text  
 amount\_of\_product = self.amount\_of\_product\_input.text  
  
 if not delivery\_id.isdigit():  
 show\_error\_popup("Delivery ID must be an integer.")  
 return  
  
 try:  
 datetime.strptime(delivery\_date, "%Y-%m-%d")  
 except ValueError:  
 show\_error\_popup("Delivery date must be in the format YYYY-MM-DD.")  
 return  
  
 try:  
 float(total\_amount)  
 except ValueError:  
 show\_error\_popup("Total amount must be a float.")  
 return  
  
 if not re.match(r".+@gmail\.com$", supplier\_id):  
 show\_error\_popup("Supplier ID must end with @gmail.com.")  
 return  
  
 if not quantity.isdigit():  
 show\_error\_popup("Quantity must be an integer.")  
 return  
  
 try:  
 float(amount\_of\_product)  
 except ValueError:  
 show\_error\_popup("Amount must be a float.")  
 return  
  
 # Disable input fields in layout 2  
 self.delivery\_id\_input.disabled = True  
 self.delivery\_date\_input.disabled = True  
 self.total\_amount\_input.disabled = True  
 self.supplier\_id\_input.disabled = True  
  
 # Add data to fourth layout  
 fourth\_layout.add\_widget(Label(text=str(delivery\_id)))  
 fourth\_layout.add\_widget(Label(text=dropdown\_button.text))  
 fourth\_layout.add\_widget(Label(text=str(quantity)))  
 fourth\_layout.add\_widget(Label(text=str(amount\_of\_product)))  
 submit\_order\_button.disabled = False  
  
 def show\_error\_popup(message):  
 popup = Popup(title='Error', background\_color=(0.004, 0.055, 0.102, 1.0), content=Label(text=message),  
 size\_hint=(None, None), size=(400, 200))  
 popup.open()  
  
 submit\_button = Button(text="Submit Product", size=(30, 30), background\_color=(0.133, 0.855, 0.431, 1.0))  
 submit\_button.bind(on\_release=on\_submit)  
  
 # ScrollView for fourth layout  
 scroll\_view = ScrollView(size\_hint=(1, None), size=(400, 400))  
 scroll\_view.add\_widget(fourth\_layout)  
  
 # Create the popup  
 popup\_content = GridLayout(cols=1, background\_color=(0.004, 0.055, 0.102, 1.0))  
 popup\_content.add\_widget(first\_layout)  
 popup\_content.add\_widget(second\_layout)  
  
 # Add submit button beside third layout  
 submit\_container = GridLayout(cols=2, padding=15, spacing=20, size\_hint\_y=None,  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
 submit\_container.add\_widget(third\_layout)  
 submit\_container.add\_widget(submit\_button)  
 popup\_content.add\_widget(submit\_container)  
 popup\_content.add\_widget(second\_layout2)  
 popup\_content.add\_widget(scroll\_view)  
 popup\_content.add\_widget(fifth\_layout)  
  
 self.popup = Popup(title='My Popup', content=popup\_content, size\_hint=(None, None), auto\_dismiss=False,  
 size=(900, 890), background\_color=(0.004, 0.055, 0.102, 1.0))  
 self.popup.open()  
  
 def show\_success\_popup\_1(self, message):  
 # Display a success popup with the given message  
 success\_popup = Popup(title='Success', content=Label(text=message), size\_hint=(None, None), size=(300, 200))  
 success\_popup.open()  
  
 def show\_error\_popup\_1(self, message):  
 # Display an error popup with the given message  
 popup\_width = len(message) \* 10 # Adjust the multiplier based on your preference  
 popup\_height = max(len(message) // 15,  
 1) \* 40 # Adjust the divisor and multiplier based on your preference  
  
 error\_popup = Popup(title='Error', content=Label(text=message), size\_hint=(None, None),  
 size=(popup\_width, popup\_height,))  
 error\_popup.open()  
 def dismiss\_popup1(self, instance):  
 if self.popup1:  
 self.popup1.dismiss()  
  
 def dismiss\_popup\_1(self,instance=None):  
 self.popup.dismiss()

**Order Reports:**

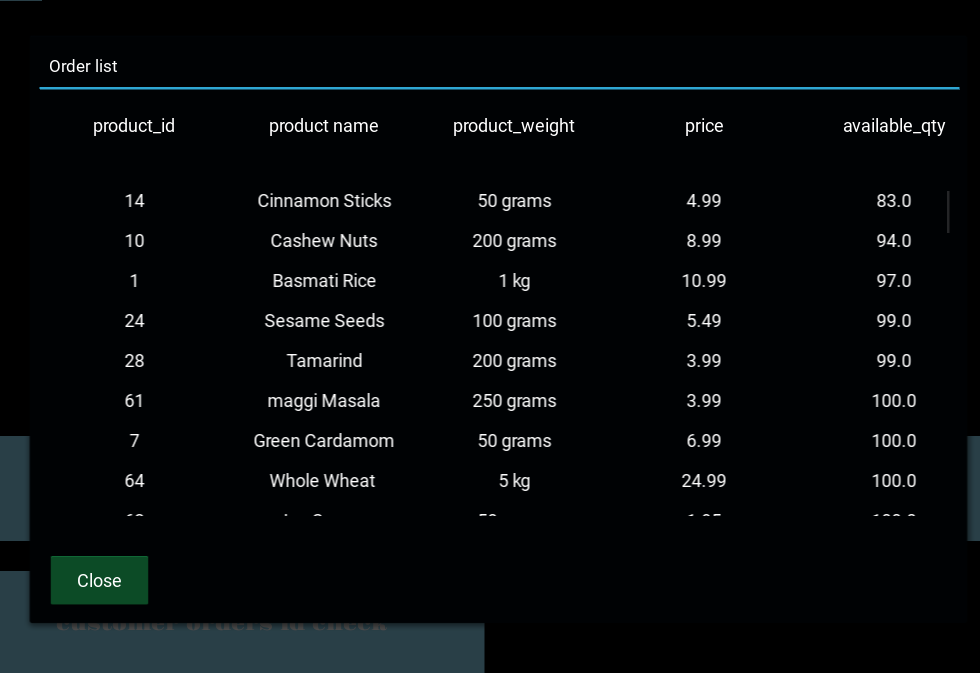
In the order reports page, the user can check the order reports of customers and store. We have also provided a button named ”customer orders id check” which is used to track the order receipt number of the customer if they can provide a bill receipt.

**Check customer orders:**



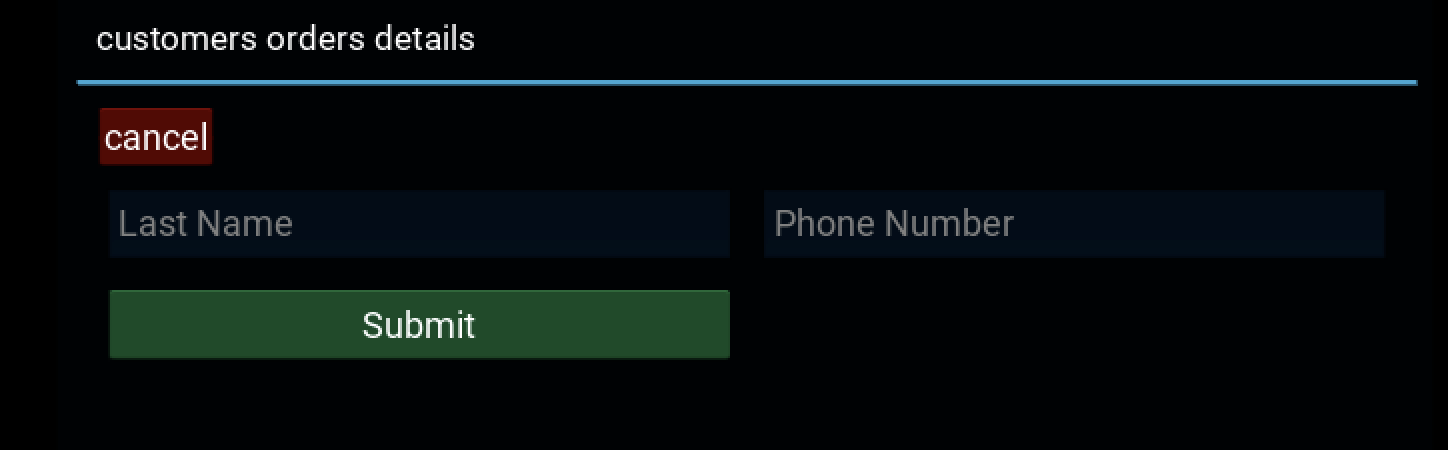
Here, the user needs to provide the order id number to get the purchase details of the customer which can be used for returns. A cancel button is provided to close the pop-up window.

**Order for store:**



Here, the user can find the details and inventory of the available products, we have made the available\_qty to show in ascending order so that the user can easily find out the low inventory product and make an order of it from the supplier quickly.

**Customer orders id check:**



The customer orders id check is used to find the purchase details of the customer if they don’t provide the billing receipt. The user needs to provide the last name and phone number of the customer to retrieve the purchase details of customer.

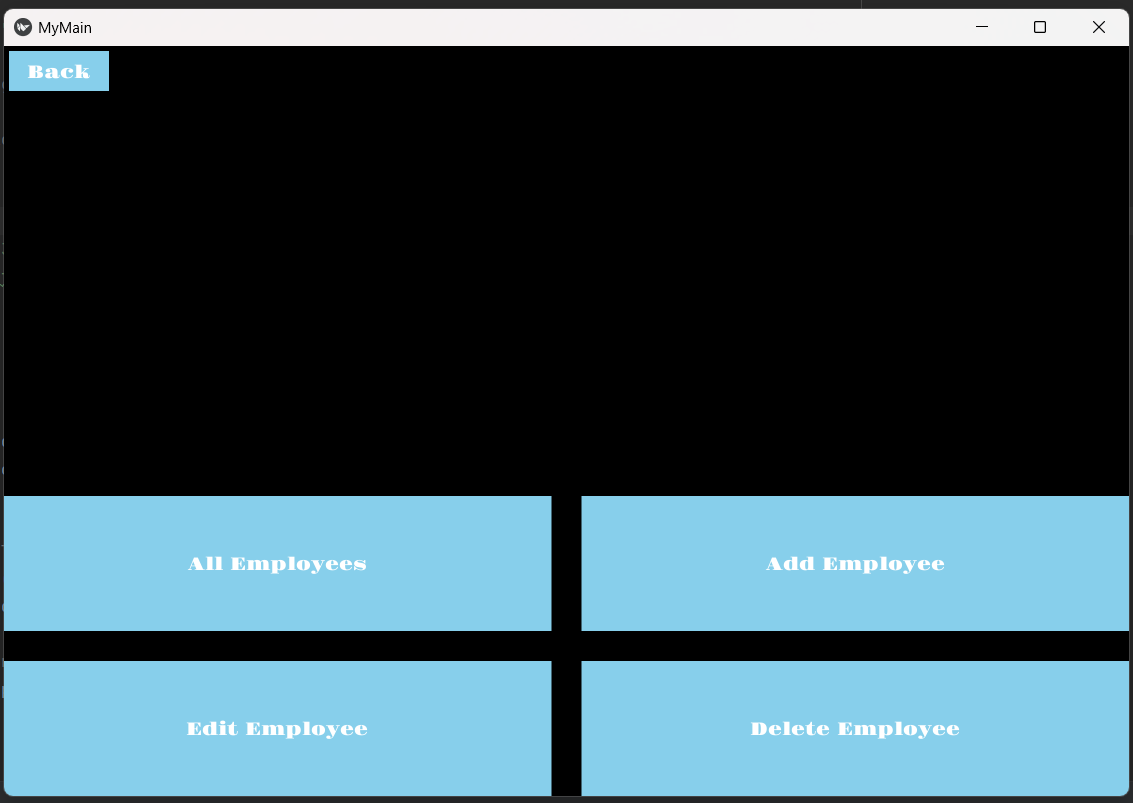
**Code:**

**Orders\_for\_suppliers.py:**

import mysql.connector  
from kivy.uix.scrollview import ScrollView  
from kivy.uix.boxlayout import BoxLayout  
from kivy.uix.screenmanager import Screen  
from kivy.uix.popup import Popup  
from kivy.uix.gridlayout import GridLayout  
from kivy.uix.label import Label  
from kivy.uix.textinput import TextInput  
from kivy.uix.button import Button  
from kivy.uix.spinner import Spinner  
import re  
class Orders(Screen):  
 def show\_all\_orders(self):  
 # Establish a connection to your MySQL database  
 conn = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Sahith@12",  
 port='3306',  
 database='grocerystore'  
 )  
 cursor = conn.cursor()  
  
 # Fetch specific columns (first\_name, last\_name, emp\_id, phone) from the database  
 cursor.execute(  
 "select product\_id, product\_name, weight\_of\_product,product\_price,product\_available\_quantity from products order by product\_available\_quantity ")  
 order = cursor.fetchall()  
  
 # Close the database connection  
 cursor.close()  
 conn.close()  
  
 # Create a BoxLayout to organize the labels and the scroll view  
 content\_layout = BoxLayout(orientation='vertical', padding=10, spacing=40,  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
  
 # Create a GridLayout for the labels  
 labels\_layout = GridLayout(cols=5, size\_hint\_y=None, height='40dp', padding=10, spacing=40)  
  
 # Set fixed widths for columns  
 column\_widths = [150, 150, 150, 150]  
  
 # Add column labels  
 labels\_layout.add\_widget(Label(text='product\_id', size\_hint\_x=None, width=column\_widths[0]))  
 labels\_layout.add\_widget(Label(text='product name', size\_hint\_x=None, width=column\_widths[1]))  
 labels\_layout.add\_widget(Label(text='product\_weight', size\_hint\_x=None, width=column\_widths[2]))  
 labels\_layout.add\_widget(Label(text='price', size\_hint\_x=None, width=column\_widths[3]))  
 labels\_layout.add\_widget(Label(text='available\_qty', size\_hint\_x=None, width=column\_widths[3]))  
 # Add labels layout to content layout  
 content\_layout.add\_widget(labels\_layout)  
  
 # Create a ScrollView with GridLayout inside to allow scrolling  
 layout = GridLayout(cols=5, size\_hint\_y=None, padding=10, spacing=40)  
 layout.bind(minimum\_height=layout.setter('height'))  
  
 # Add employee data  
 for ord in order:  
 p\_id = Label(text=str(ord[0]), size\_hint\_x=None, width=column\_widths[0], height='40dp',  
 )  
 p\_name = Label(text=str(ord[1]), size\_hint\_x=None, width=column\_widths[1], height='40dp',  
 )  
 p\_weight = Label(text=str(ord[2]), size\_hint\_x=None, width=column\_widths[2], height='40dp',  
 )  
 p\_price = Label(text=str(ord[3]), size\_hint\_x=None, width=column\_widths[3], height='40dp',  
 )  
 p\_q\_a = Label(text=str(ord[4]), size\_hint\_x=None, width=column\_widths[3], height='40dp',  
 )  
  
 layout.add\_widget(p\_id)  
 layout.add\_widget(p\_name)  
 layout.add\_widget(p\_weight)  
 layout.add\_widget(p\_price)  
 layout.add\_widget(p\_q\_a)  
  
 # Create a ScrollView with GridLayout inside to allow scrolling  
 scroll\_view = ScrollView()  
 scroll\_view.add\_widget(layout)  
  
 # Add scroll view to content layout  
 content\_layout.add\_widget(scroll\_view)  
  
 # Create a close button  
 close\_button = Button(text='Close', size\_hint=(None, None), size=(100, 50),  
 background\_color=(0.133, 0.855, 0.431, 1.0))  
 close\_button.bind(on\_press=self.dismiss\_popup)  
  
 # Add close button to content layout  
 content\_layout.add\_widget(close\_button)  
  
 # Create a Popup with the content layout  
 self.popup = Popup(title='Order list', content=content\_layout, size\_hint=(None, None), size=(950, 600),  
 background\_color=(0.004, 0.055, 0.102, 1.0), auto\_dismiss=False)  
 self.popup.open()  
  
 def dismiss\_popup(self, instance):  
 if hasattr(self, 'popup') and self.popup:  
 self.popup.dismiss()  
  
 def order\_id\_check(self):  
 # Create the first layout with the back button  
 first\_layout = GridLayout(cols=1, padding=10, size\_hint\_y=0.06, background\_color=(0.004, 0.055, 0.102, 1.0))  
 back\_button = Button(text='cancel', size\_hint=(None, None), size=(59, 30), background\_color=(1, 0, 0, 1))  
  
 # Define callback function for the back button  
 def back\_callback(instance):  
 popup.dismiss()  
  
 # Bind the callback function to the back button  
 back\_button.bind(on\_release=back\_callback)  
  
 # Add the back button to the first layout  
 first\_layout.add\_widget(back\_button)  
  
 # Create the second layout with input fields and submit button  
 second\_layout = GridLayout(cols=2, padding=15, spacing=15, size\_hint\_y=0.20,  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
 last\_name\_input = TextInput(hint\_text='Last Name', background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 phone\_input = TextInput(hint\_text='Phone Number', background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 submit\_button = Button(text='Submit', background\_color=(0.133, 0.855, 0.431, 1.0))  
  
 # Define callback function for the submit button  
 def submit\_callback(instance):  
 last\_name = last\_name\_input.text  
 phone\_number = phone\_input.text  
 if not all([phone\_number, last\_name]):  
 self.show\_error\_popup("All fields are required.")  
 return  
 if len(phone\_number) != 10 or not phone\_number.isdigit():  
 self.show\_error\_popup("Invalid phone number.\n Phone number must be 10 digits.")  
 return  
 try:  
 connection = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Sahith@12",  
 database="grocerystore"  
 )  
 phone = self.add\_dashes\_to\_number\_with\_existing\_dashes(phone\_number)  
 cursor = connection.cursor()  
 cursor.execute(  
 "SELECT order\_id, order\_date, total\_amount,case when payment\_id=1 then 'cash' else 'QR code' end as payment\_id"  
 " FROM orders where customer\_id=(select customer\_id from customer where last\_name=%s and cus\_number=%s) order by order\_date desc",  
 (last\_name, phone))  
 orders = cursor.fetchall()  
  
 # Add order details to ScrollView  
 scroll\_layout = GridLayout(cols=4, padding=20, spacing=20, size\_hint\_y=None,  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
 scroll\_layout.bind(  
 minimum\_height=scroll\_layout.setter('height')) # Allow ScrollView to scroll vertically  
 scroll\_layout.clear\_widgets()  
 for order in orders:  
 for detail in order:  
 scroll\_layout.add\_widget(Label(text=str(detail)))  
  
 cursor.close()  
 connection.close()  
  
 # Create a ScrollView and add the scrollable layout to it  
 scroll\_view = ScrollView(size\_hint=(1, None), size=(400, 400))  
 scroll\_view.add\_widget(scroll\_layout)  
  
 # Add labels layout and ScrollView to the third layout  
 third\_layout.clear\_widgets() # Clear previous content  
 third\_layout.add\_widget(labels\_layout)  
 third\_layout.add\_widget(scroll\_view)  
  
  
 except mysql.connector.Error as e:  
  
 p = str(e)  
  
 self.show\_error\_popup("Failed to get orders \n{}".format(  
  
 p[13:].replace('Duplicate entry', 'Already Exist ').replace('employee.', 'in ').replace('for key',  
  
 ' ')))  
  
 # Bind the callback function to the submit button  
 submit\_button.bind(on\_release=submit\_callback)  
  
 # Add input fields and submit button to the second layout  
 second\_layout.add\_widget(last\_name\_input)  
 second\_layout.add\_widget(phone\_input)  
 second\_layout.add\_widget(submit\_button)  
  
 # Create the third layout with labels and ScrollView  
 third\_layout = GridLayout(cols=1, padding=2, size\_hint\_y=0.81, background\_color=(0.004, 0.055, 0.102, 1.0))  
 labels\_layout = GridLayout(cols=4, padding=5, spacing=5, size\_hint\_y=0.05,  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
 labels\_layout.add\_widget(Label(text='Order ID', ))  
 labels\_layout.add\_widget(Label(text='Order Date', ))  
 labels\_layout.add\_widget(Label(text='Total Amount', ))  
 labels\_layout.add\_widget(Label(text='Payment Method', ))  
  
 # Create the popup window  
 popup = Popup(title='customers orders details', auto\_dismiss=False, size\_hint=(None, None), size=(700, 700),  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
  
 # Combine the layouts in a parent layout  
 parent\_layout = GridLayout(cols=1)  
 parent\_layout.add\_widget(first\_layout)  
 parent\_layout.add\_widget(second\_layout)  
 parent\_layout.add\_widget(third\_layout)  
  
 # Set the parent layout as the content of the popup  
 popup.content = parent\_layout  
  
 # Set the position of the back button to the top right corner  
 back\_button.pos\_hint = {'right': 1, 'top': 1}  
  
 # Open the popup  
 popup.open()  
  
 def add\_dashes\_to\_number\_with\_existing\_dashes(self, number):  
 # Convert number to string  
 number\_str = str(number)  
  
 # Use regular expression to add dashes after every three digits for the first two groups  
 # and after every four digits for the last group  
 formatted\_number = re.sub(r'(\d{3})(\d{3})(\d{4})', r'\1-\2-\3', number\_str)  
  
 return formatted\_number  
  
 def show\_error\_popup(self, message):  
 # Display an error popup with the given message  
 popup\_width = len(message) \* 10 # Adjust the multiplier based on your preference  
 popup\_height = max(len(message) // 15,  
 1) \* 40 # Adjust the divisor and multiplier based on your preference  
  
 error\_popup = Popup(title='Error', content=Label(text=message), size\_hint=(None, None),  
 size=(popup\_width, popup\_height))  
 error\_popup.open()  
  
 def order\_check(self):  
 # Create the first layout with the back button  
 first\_layout = GridLayout(cols=1, padding=10, size\_hint\_y=0.06, background\_color=(0.004, 0.055, 0.102, 1.0))  
 back\_button = Button(text='cancel', size\_hint=(None, None), size=(59, 30), background\_color=(1, 0, 0, 1))  
  
 # Define callback function for the back button  
 def back\_callback(instance):  
 popup.dismiss()  
  
 # Bind the callback function to the back button  
 back\_button.bind(on\_release=back\_callback)  
  
 # Add the back button to the first layout  
 first\_layout.add\_widget(back\_button)  
  
 # Create the second layout with input fields and submit button  
 second\_layout = GridLayout(cols=2, padding=15, spacing=15, size\_hint\_y=0.10,  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
 last\_name\_input = TextInput(hint\_text='Enter order\_id', background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 submit\_button = Button(text='Search', background\_color=(0.133, 0.855, 0.431, 1.0))  
  
 # Define callback function for the submit button  
 def submit\_callback(instance):  
 last\_name = last\_name\_input.text  
 if not all(last\_name):  
 self.show\_error\_popup("All fields are required.")  
 return  
 if not last\_name.isdigit():  
 self.show\_error\_popup("Invalid order number.\n order number must be a digits.")  
 return  
 try:  
 connection = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Sahith@12",  
 database="grocerystore"  
 )  
  
 cursor = connection.cursor()  
 cursor.execute(  
 " select product\_name,weight\_of\_product,product\_price ,quantity\_of\_product,amount\_of\_product from orders o join order\_items\_summary os on o.order\_id=os.order\_id join products p on p.product\_id=os.product\_id where o.order\_id=%s",  
 [last\_name, ])  
 orders = cursor.fetchall()  
 cursor.execute(  
 " select order\_date,total\_amount from orders where order\_id=%s",  
 [last\_name, ])  
 orderd = cursor.fetchone()  
 if orderd!= None:  
 l2.text = str(orderd[0])  
 l4.text = str(orderd[1])  
  
 # Add order details to ScrollView  
 scroll\_layout = GridLayout(cols=5, padding=23, spacing=23, size\_hint\_y=None,  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
 scroll\_layout.bind(  
 minimum\_height=scroll\_layout.setter('height')) # Allow ScrollView to scroll vertically  
 scroll\_layout.clear\_widgets()  
 for order in orders:  
 for detail in order:  
 scroll\_layout.add\_widget(Label(text=str(detail)))  
  
 cursor.close()  
 connection.close()  
  
 # Create a ScrollView and add the scrollable layout to it  
 scroll\_view = ScrollView(size\_hint=(1, None), size=(400, 400))  
 scroll\_view.add\_widget(scroll\_layout)  
  
 # Add labels layout and ScrollView to the third layout  
 third\_layout.clear\_widgets() # Clear previous content  
 third\_layout.add\_widget(labels\_layout)  
 third\_layout.add\_widget(scroll\_view)  
  
  
 except mysql.connector.Error as e:  
 print(e)  
  
 p = str(e)  
  
 self.show\_error\_popup("Failed to get orders \n{}".format(  
  
 p[13:].replace('Duplicate entry', 'Already Exist ').replace('employee.', 'in ').replace('for key',  
  
 ' ')))  
  
 # Bind the callback function to the submit button  
 submit\_button.bind(on\_release=submit\_callback)  
  
 # Add input fields and submit button to the second layout  
 second\_layout.add\_widget(last\_name\_input)  
 second\_layout.add\_widget(submit\_button)  
 second\_layout2 = GridLayout(cols=4, padding=20, spacing=20, size\_hint\_y=0.10,  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
 l1 = Label(text='order date')  
 l2 = Label(text=' ', )  
 l3 = Label(text='total\_amount', )  
 l4 = Label(text='', )  
 second\_layout2.add\_widget(l1)  
 second\_layout2.add\_widget(l2)  
 second\_layout2.add\_widget(l3)  
 second\_layout2.add\_widget(l4)  
  
 # Create the third layout with labels and ScrollView  
 third\_layout = GridLayout(cols=1, padding=2, size\_hint\_y=0.81, background\_color=(0.004, 0.055, 0.102, 1.0))  
 labels\_layout = GridLayout(cols=5, padding=5, spacing=5, size\_hint\_y=0.05,  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
 labels\_layout.add\_widget(Label(text='product\_name',))  
 labels\_layout.add\_widget(Label(text='prdouct\_weight', ))  
 labels\_layout.add\_widget(Label(text='product\_price', ))  
 labels\_layout.add\_widget(Label(text='product\_quantity', ))  
 labels\_layout.add\_widget(Label(text='total\_of\_product', ))  
  
 # Create the popup window  
 popup = Popup(title='customers orders details', auto\_dismiss=False, size\_hint=(None, None), size=(750, 750),  
 background\_color=(0.004, 0.055, 0.102, 1.0))

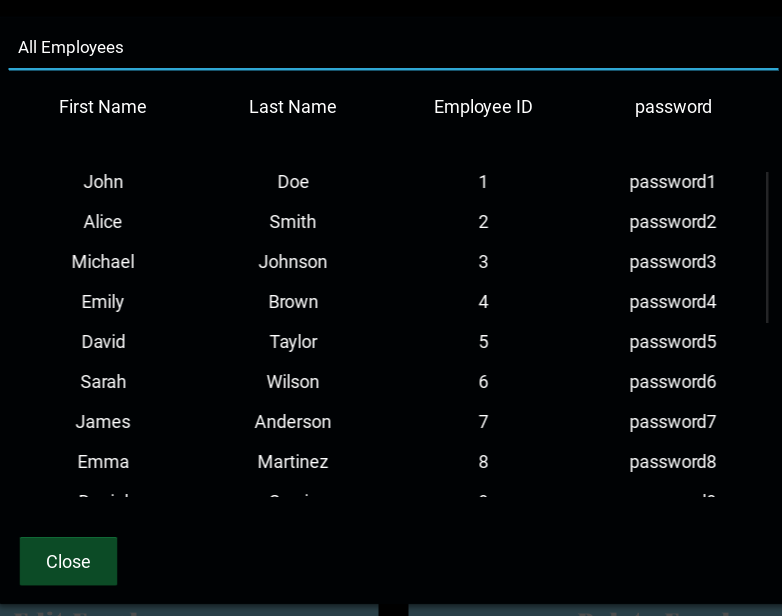
# Combine the layouts in a parent layout  
 parent\_layout = GridLayout(cols=1)  
 parent\_layout.add\_widget(first\_layout)  
 parent\_layout.add\_widget(second\_layout)  
 parent\_layout.add\_widget(second\_layout2)  
 parent\_layout.add\_widget(third\_layout)  
  
 # Set the parent layout as the content of the popup  
 popup.content = parent\_layout  
  
 # Set the position of the back button to the top right corner  
 back\_button.pos\_hint = {'right': 1, 'top': 1}  
  
 # Open the popup  
 popup.open()

**Employee Management:**



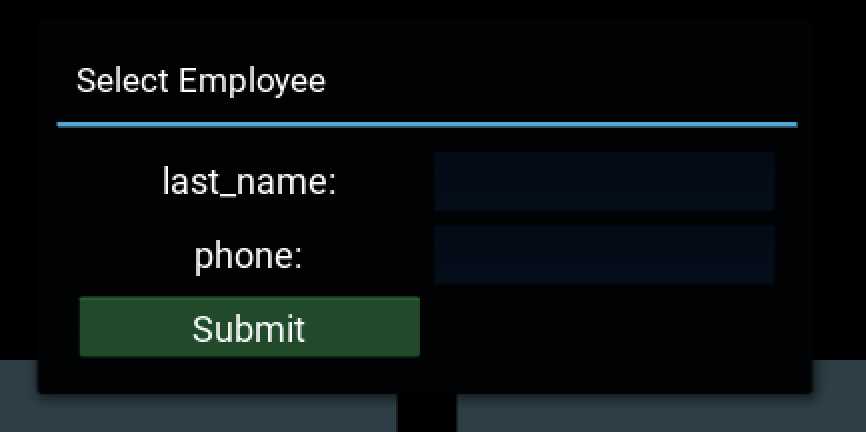
Here the user can manage all employees such as track details of employees, add a new employee to database, edit the existing employee details and remove the employee details from the database.

**All Employees:**



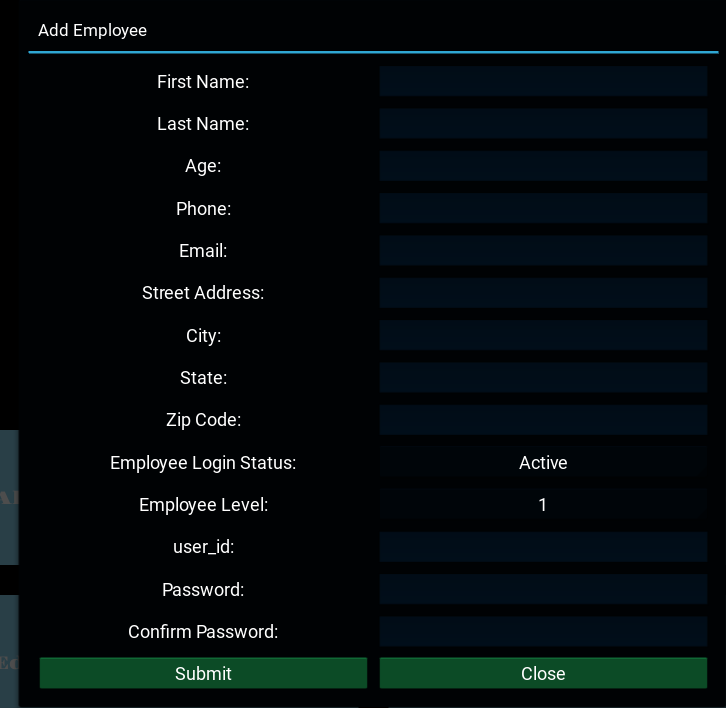
Here the user can find all the details of employees such as his name, id number and password. A close button is provided to exit the pop-up window.

**Edit Employee:**



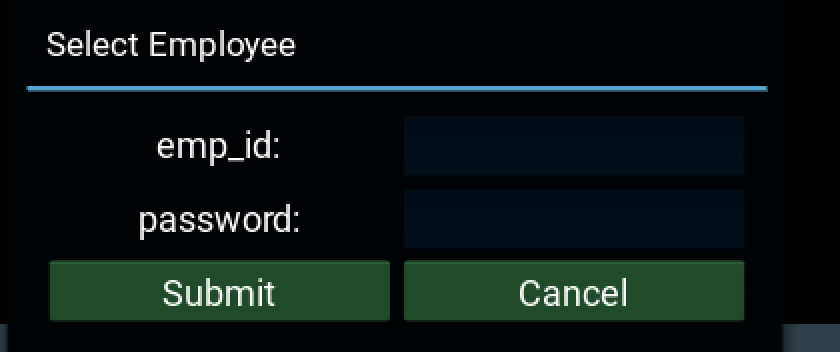
The user can edit the existing employee details by providing his last name and phone number.

**Add Employee:**



Here the user can add a new employee to the database by entering all details provided in the form.

**Delete Employee:**



Here the user can delete an employee from database by providing the employee id and password.

**Code:**

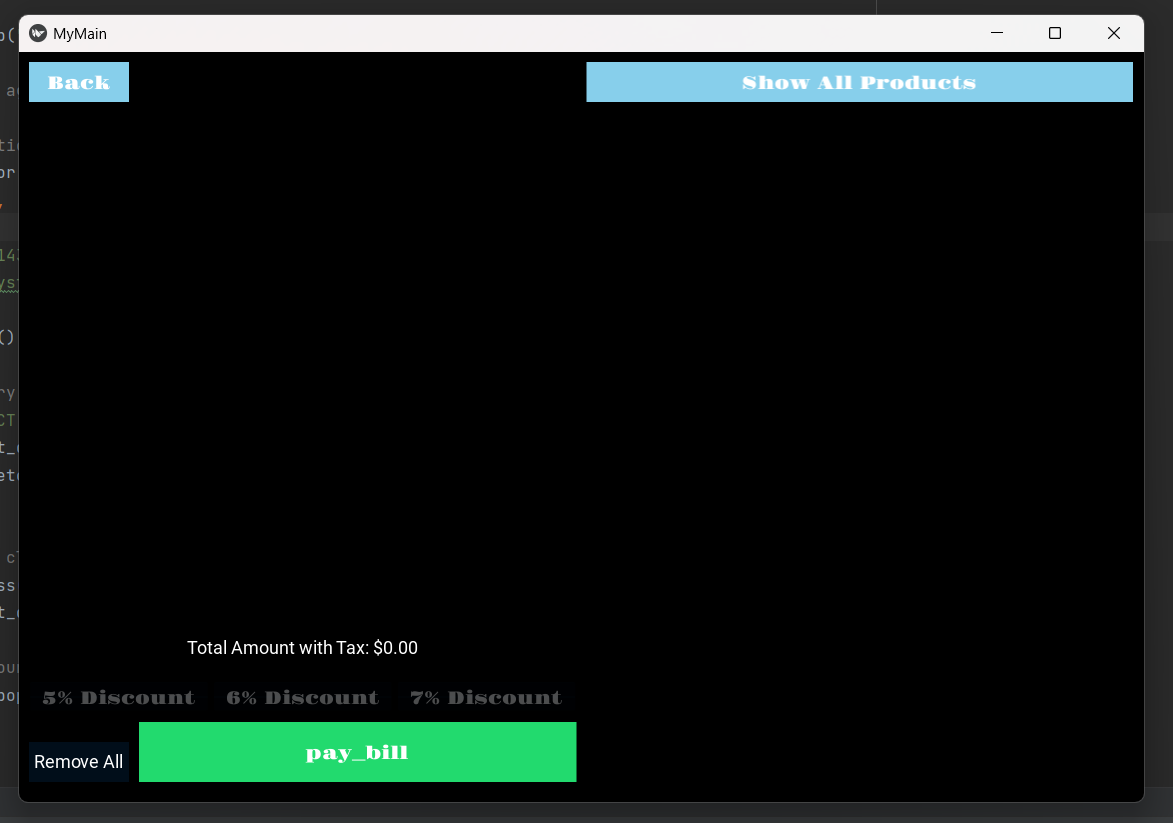
**EmployeeScreen.py:**

import mysql.connector  
from kivy.uix.scrollview import ScrollView  
from kivy.uix.boxlayout import BoxLayout  
from kivy.uix.screenmanager import Screen  
from kivy.uix.popup import Popup  
from kivy.uix.gridlayout import GridLayout  
from kivy.uix.label import Label  
from kivy.uix.textinput import TextInput  
from kivy.uix.button import Button  
from kivy.uix.spinner import Spinner  
import re  
class EmployeeScreen(Screen):  
 def show\_all\_employees(self):  
 # Establish a connection to your MySQL database  
 conn = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Sahith@12",  
 port='3306',  
 database='grocerystore'  
 )  
 cursor = conn.cursor()  
  
 # Fetch specific columns (first\_name, last\_name, emp\_id, phone) from the database  
 cursor.execute("SELECT first\_name, last\_name, emp\_id, emp\_password FROM employee")  
 employees\_data = cursor.fetchall()  
  
 # Close the database connection  
 cursor.close()  
 conn.close()  
  
 # Create a BoxLayout to organize the labels and the scroll view  
 content\_layout = BoxLayout(orientation='vertical', padding=10, spacing=40,  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
  
 # Create a GridLayout for the labels  
 labels\_layout = GridLayout(cols=4, size\_hint\_y=None, height='40dp', padding=10, spacing=40)  
  
 # Set fixed widths for columns  
 column\_widths = [150, 150, 150, 150]  
  
 # Add column labels  
 labels\_layout.add\_widget(Label(text='First Name', size\_hint\_x=None, width=column\_widths[0]))  
 labels\_layout.add\_widget(Label(text='Last Name', size\_hint\_x=None, width=column\_widths[1]))  
 labels\_layout.add\_widget(Label(text='Employee ID', size\_hint\_x=None, width=column\_widths[2]))  
 labels\_layout.add\_widget(Label(text='password', size\_hint\_x=None, width=column\_widths[3]))  
  
 # Add labels layout to content layout  
 content\_layout.add\_widget(labels\_layout)  
  
 # Create a ScrollView with GridLayout inside to allow scrolling  
 layout = GridLayout(cols=4, size\_hint\_y=None, padding=10, spacing=40)  
 layout.bind(minimum\_height=layout.setter('height'))  
  
 # Add employee data  
 for employee in employees\_data:  
 first\_name\_label = Label(text=employee[0], size\_hint\_x=None, width=column\_widths[0], height='40dp',  
 )  
 last\_name\_label = Label(text=employee[1], size\_hint\_x=None, width=column\_widths[1], height='40dp',  
 )  
 emp\_id\_label = Label(text=str(employee[2]), size\_hint\_x=None, width=column\_widths[2], height='40dp',  
 )  
 phone\_label = Label(text=employee[3], size\_hint\_x=None, width=column\_widths[3], height='40dp',  
 )  
  
 layout.add\_widget(first\_name\_label)  
 layout.add\_widget(last\_name\_label)  
 layout.add\_widget(emp\_id\_label)  
 layout.add\_widget(phone\_label)  
  
 # Create a ScrollView with GridLayout inside to allow scrolling  
 scroll\_view = ScrollView()  
 scroll\_view.add\_widget(layout)  
  
 # Add scroll view to content layout  
 content\_layout.add\_widget(scroll\_view)  
  
 # Create a close button  
 close\_button = Button(text='Close', size\_hint=(None, None), size=(100, 50),  
 background\_color=(0.133, 0.855, 0.431, 1.0))  
 close\_button.bind(on\_press=self.dismiss\_popup)  
  
 # Add close button to content layout  
 content\_layout.add\_widget(close\_button)  
  
 # Create a Popup with the content layout  
 self.popup = Popup(title='All Employees', content=content\_layout, size\_hint=(None, None), size=(800, 600),  
 background\_color=(0.004, 0.055, 0.102, 1.0), auto\_dismiss=False)  
 self.popup.open()  
  
 def dismiss\_popup(self, instance):  
 if hasattr(self, 'popup') and self.popup:  
 self.popup.dismiss()  
  
 def add\_employee(self):  
 # Create a GridLayout to organize input fields  
 layout = GridLayout(cols=2, spacing=10, padding=10, background\_color=(0.004, 0.055, 0.102, 1.0))  
  
 # Add labels and input fields for each attribute  
 layout.add\_widget(Label(text='First Name:', ))  
 self.first\_name\_input = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.first\_name\_input)  
  
 layout.add\_widget(Label(text='Last Name:', ))  
 self.last\_name\_input = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.last\_name\_input)  
  
 layout.add\_widget(Label(text='Age:', ))  
 self.age = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.age)  
  
 layout.add\_widget(Label(text='Phone:', ))  
 self.phone\_input = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.phone\_input)  
  
 layout.add\_widget(Label(text='Email:', ))  
 self.email\_input = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.email\_input)  
  
 layout.add\_widget(Label(text='Street Address:', ))  
 self.street\_address\_input = TextInput(multiline=True, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.street\_address\_input)  
  
 layout.add\_widget(Label(text='City:', ))  
 self.city\_input = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.city\_input)  
  
 layout.add\_widget(Label(text='State:', ))  
 self.state\_input = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.state\_input)  
  
 layout.add\_widget(Label(text='Zip Code:', ))  
 self.Zip\_input = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.Zip\_input)  
  
 layout.add\_widget(Label(text='Employee Login Status:', ))  
 self.login\_status\_spinner = Spinner(text='Active', values=['Active', 'Inactive'],  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
 layout.add\_widget(self.login\_status\_spinner)  
  
 layout.add\_widget(Label(text='Employee Level:', ))  
 self.emp\_level\_spinner = Spinner(text='1', values=['1', '2', '3', '4', '5'],  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
 layout.add\_widget(self.emp\_level\_spinner)  
 layout.add\_widget(Label(text='user\_id:', ))  
 self.user\_id = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.user\_id)  
 layout.add\_widget(Label(text='Password:', ))  
 self.password = TextInput(multiline=False, password=True, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.password)  
 layout.add\_widget(Label(text='Confirm Password:', ))  
 self.con\_password = TextInput(multiline=False, password=True, background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.con\_password)  
  
 # Create a submit button  
 submit\_button = Button(text='Submit', background\_color=(0.133, 0.855, 0.431, 1.0))  
 submit\_button.bind(on\_press=self.submit\_employee)  
 layout.add\_widget(submit\_button)  
  
 # Create a cancel button  
 cancel\_button = Button(text='Close', background\_color=(0.133, 0.855, 0.431, 1.0))  
 cancel\_button.bind(on\_press=self.dismiss\_popup1)  
 layout.add\_widget(cancel\_button)  
  
 # Create a Popup with the layout and background color  
 self.popup1 = Popup(title='Add Employee', content=layout, size\_hint=(None, None), size=(720, 720),  
 background\_color=(0.004, 0.055, 0.102, 1.0), auto\_dismiss=False)  
 self.popup1.open()  
  
 def submit\_employee(self, instance):  
 # Retrieve employee data from input fields  
 first\_name = self.first\_name\_input.text  
 last\_name = self.last\_name\_input.text  
 age = self.age.text  
 phone = self.phone\_input.text  
 email = self.email\_input.text  
 street\_address = self.street\_address\_input.text  
 zip = self.Zip\_input.text  
 city = self.city\_input.text  
 state = self.state\_input.text  
 login\_status = self.login\_status\_spinner.text  
 emp\_level = self.emp\_level\_spinner.text  
 pass1 = self.password.text  
 pass2 = self.con\_password.text  
 user = self.user\_id.text  
  
 # Perform validation checks  
 if not all([first\_name, last\_name, phone, email, street\_address, city, state, zip,pass1,pass2]):  
 self.show\_error\_popup("All fields are required.")  
 return  
 if not age.isdigit():  
 self.show\_error\_popup("Invalid Age.\n Age number must be between 18-45.")  
 return  
 if len(phone) != 10 or not phone.isdigit():  
 self.show\_error\_popup("Invalid phone number.\n Phone number must be 10 digits.")  
 return  
  
 if not [email.endswith('@gmail.com](mailto:email.endswith('@gmail.com)') or not email[0].isalpha():  
 self.show\_error\_popup("Invalid email address. \nEmail must end with @gmail.com")  
 return  
 if len(zip) != 6 or not zip.isdigit():  
 self.show\_error\_popup("Invalid zip number.\n zip number must be 6 digits.")  
 return  
 if not user.isdigit():  
 self.show\_error\_popup("Invalid user .\n user must be digit")  
 return  
 if pass1 != pass2:  
 self.show\_error\_popup("Invalid password.\n password must be same ")  
 return  
 # If all validation checks pass, save employee information to the database  
 try:  
 # Establish a connection to MySQL database  
 conn = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Sahith@12",  
 database="grocerystore"  
 )  
 cursor = conn.cursor()  
 number = self.add\_dashes\_to\_number\_with\_existing\_dashes(phone)  
 # Execute INSERT query to insert employee details into the database  
 insert\_query = "INSERT INTO employee (emp\_id,first\_name, last\_name, phone,zip, mail, street\_address, city, state, employee\_login\_status, emp\_level,emp\_password,age) VALUES (%s,%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"  
 cursor.execute(insert\_query, (user,  
 first\_name, last\_name, number, zip, email, street\_address, city, state,  
 login\_status, emp\_level, pass1, age))  
  
 # Commit changes and close connection  
 conn.commit()  
 cursor.close()  
 conn.close()  
  
 # Show success popup  
 self.show\_success\_popup("Employee details saved to database successfully.")  
  
 except mysql.connector.Error as e:  
 p = str(e)  
 self.show\_error\_popup("Failed to add employee \n{}".format(  
 p[13:].replace('Duplicate entry', 'Already Exist ').replace('employee.', 'in ').replace('for key',  
 ' ')))  
  
 def add\_dashes\_to\_number\_with\_existing\_dashes(self, number):  
 # Convert number to string  
 number\_str = str(number)  
  
 # Use regular expression to add dashes after every three digits for the first two groups  
 # and after every four digits for the last group  
 formatted\_number = re.sub(r'(\d{3})(\d{3})(\d{4})', r'\1-\2-\3', number\_str)  
  
 return formatted\_number  
  
 def show\_success\_popup(self, message):  
 # Display a success popup with the given message  
 success\_popup = Popup(title='Success', content=Label(text=message), size\_hint=(None, None), size=(300, 200))  
 success\_popup.open()  
  
 def show\_error\_popup(self, message):  
 # Display an error popup with the given message  
 popup\_width = len(message) \* 10 # Adjust the multiplier based on your preference  
 popup\_height = max(len(message) // 15,  
 1) \* 40 # Adjust the divisor and multiplier based on your preference  
  
 error\_popup = Popup(title='Error', content=Label(text=message), size\_hint=(None, None),  
 size=(popup\_width, popup\_height))  
 error\_popup.open()  
  
 def dismiss\_popup1(self, instance=None):  
 self.popup1.dismiss()  
  
 def edit\_employee(self):  
 # Create a GridLayout to organize input fields  
 layout = GridLayout(cols=2, spacing=5, padding=10)  
  
 # Add labels and input fields for user ID and password  
 layout.add\_widget(Label(text='last\_name:',))  
 self.lastname\_input = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.lastname\_input)  
  
 layout.add\_widget(Label(text='phone:',))  
 self.phonenumber\_input = TextInput(multiline=False, password=True, background\_color=(0.004, 0.055, 0.102, 1.0),cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.phonenumber\_input)  
 # Create a submit button  
 submit\_button = Button(text='Submit', background\_color=(0.133, 0.855, 0.431, 1.0))  
 submit\_button.bind(on\_press=self.authenticate\_employee)  
 layout.add\_widget(submit\_button)  
  
 # Create a Popup with the layout  
 self.popup = Popup(title='Select Employee', content=layout, size\_hint=(None, None),  
 background\_color=(0.004, 0.055, 0.102, 1.0), size=(400, 200))  
 self.popup.open()  
  
 def authenticate\_employee(self, instance):  
 # Retrieve user ID and password from input fields  
 lastname = self.lastname\_input.text.strip()  
 phone = self.phonenumber\_input.text.strip()  
 if not all([lastname, phone]):  
 self.show\_error\_popup1("All fields are required.")  
 return  
 if len(phone) != 10 or not phone.isdigit():  
 self.show\_error\_popup1("Invalid phone number.\n Phone number must be 10 digits.")  
 return  
 # Perform authentication against the database  
 try:  
 # Establish a connection to MySQL database  
 conn = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Sahith@12",  
 database="grocerystore"  
 )  
 cursor = conn.cursor()  
  
 # Execute SELECT query to retrieve employee details based on user ID and password  
 select\_query = "SELECT \* FROM employee WHERE last\_name = %s AND phone = %s"  
 cursor.execute(select\_query, (lastname, self.add\_dashes\_to\_number\_with\_existing\_dashes(phone)))  
 employee = cursor.fetchone()  
  
 if employee:  
 # Employee found, close current popup and display details in another popup  
 self.popup.dismiss()  
 self.show\_employee\_details(employee)  
 else:  
 # Employee not found, show error message  
 self.show\_error\_popup1("Invalid lastname or phone number.")  
  
 cursor.close()  
 conn.close()  
  
 except mysql.connector.Error as e:  
 p = str(e)  
 self.show\_error\_popup("Failed to authenticate \n{}".format(  
 p[13:].replace('Duplicate entry', 'Already Exist ').replace('employee.', 'in ').replace('for key',  
 ' ')))  
  
 def show\_employee\_details(self, employee):  
 # Convert the tuple to a dictionary  
 employee\_dict = {  
  
 'first\_name': employee[2],  
 'last\_name': employee[3],  
 'Age': employee[4],  
 'phone': employee[9],  
 'email': employee[10],  
 'street\_address': employee[5],  
 'city': employee[6],  
 'state': employee[7],  
 'zip': employee[8],  
 'login\_status': employee[11],  
 'emp\_level': employee[12],  
 'id': employee[0],  
 "password": employee[1]  
 }  
  
 # Create a Popup to display employee details  
 self.selected\_employee\_popup = Popup(title='Selected Employee', size\_hint=(None, None), auto\_dismiss=False,  
 background\_color=(0.004, 0.055, 0.102, 1.0), size=(500, 500))  
  
 # Create a GridLayout to organize employee details  
 layout = GridLayout(cols=2, spacing=5, padding=10)  
  
 # Add labels and employee details to the layout  
 for key, value in employee\_dict.items():  
 layout.add\_widget(Label(text=str(key), ))  
 layout.add\_widget(Label(text=str(value),))  
  
 # Add an "Edit" button to allow editing employee details  
 edit\_button = Button(text='Edit', background\_color=(0.133, 0.855, 0.431, 1.0))  
 edit\_button.bind(on\_press=lambda instance: self.edit\_employee1(employee\_dict))  
 layout.add\_widget(edit\_button)  
  
 # Add the layout to the popup  
 self.selected\_employee\_popup.content = layout  
  
 # Open the popup with employee details  
 self.selected\_employee\_popup.open()  
  
 def edit\_employee1(self, employee):  
 self.selected\_employee\_popup.dismiss()  
 values = list(employee.values())  
 # Create a GridLayout to organize input fields  
 layout = GridLayout(cols=2, spacing=10, padding=10, background\_color=(0.004, 0.055, 0.102, 1.0))  
 print(values)  
 # Add labels and input fields for each attribute  
 layout.add\_widget(Label(text='First Name:', ))  
 self.first\_name\_input1 = TextInput(multiline=False, text=str(values[0]),  
 background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.first\_name\_input1)  
  
 layout.add\_widget(Label(text='Last Name:', ))  
 self.last\_name\_input1 = TextInput(multiline=False, text=str(values[1]),  
 background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.last\_name\_input1)  
  
 layout.add\_widget(Label(text='Age:', ))  
 self.age1 = TextInput(multiline=False, text=str(values[2]), background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.age1)  
  
 layout.add\_widget(Label(text='Phone:', ))  
 self.phone\_input1 = TextInput(multiline=False, text=str(values[3].replace('-', '')),  
 background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.phone\_input1)  
  
 layout.add\_widget(Label(text='Email:', ))  
 self.email\_input1 = TextInput(multiline=False, text=str(values[4]), background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.email\_input1)  
  
 layout.add\_widget(Label(text='Street Address:', ))  
 self.street\_address\_input1 = TextInput(multiline=True, text=str(values[5]),  
 background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.street\_address\_input1)  
  
 layout.add\_widget(Label(text='City:',))  
 self.city\_input1 = TextInput(multiline=False, text=str(values[6]), background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.city\_input1)  
  
 layout.add\_widget(Label(text='State:', ))  
 self.state\_input1 = TextInput(multiline=False, text=str(values[7]), background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.state\_input1)  
  
 layout.add\_widget(Label(text='Zip Code:', ))  
 self.Zip\_input1 = TextInput(multiline=False, text=str(values[8]), background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.Zip\_input1)  
  
 layout.add\_widget(Label(text='Employee Login Status:', ))  
 self.login\_status\_spinner1 = Spinner(text=str(values[9]), values=['Active', 'Inactive'],  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
 layout.add\_widget(self.login\_status\_spinner1)  
  
 layout.add\_widget(Label(text='Employee Level:', ))  
 self.emp\_level\_spinner1 = Spinner(text=str(values[10]), values=['1', '2', '3', '4', '5'],  
 background\_color=(0.004, 0.055, 0.102, 1.0))  
 layout.add\_widget(self.emp\_level\_spinner1)  
 layout.add\_widget(Label(text='user\_id:', ))  
 self.user\_id1 = TextInput(multiline=False, text=str(values[11]), background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.user\_id1)  
 layout.add\_widget(Label(text='Password:', ))  
 self.password1 = TextInput(multiline=False, text=str(values[12]), background\_color=(0.004, 0.055, 0.102, 1.0),  
 cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 layout.add\_widget(self.password1)  
  
 # Create a submit button  
 submit\_button1 = Button(text='Submit', background\_color=(0.133, 0.855, 0.431, 1.0))  
 submit\_button1.bind(on\_press=lambda instance: self.submit\_employee1(values[11]))  
  
 layout.add\_widget(submit\_button1)  
  
 # Create a cancel button  
 cancel\_button1 = Button(text='Close', background\_color=(0.133, 0.855, 0.431, 1.0))  
 cancel\_button1.bind(on\_press=self.dismiss\_popup11)  
 layout.add\_widget(cancel\_button1)  
  
 # Create a Popup with the layout and background color  
 self.popup11 = Popup(title='Edit Employee', content=layout, size\_hint=(None, None), size=(720, 720),  
 background\_color=(0.004, 0.055, 0.102, 1.0), auto\_dismiss=False)  
 self.popup11.open()  
  
 def submit\_employee1(self, value):  
 # Retrieve employee data from input fields  
 first\_name = self.first\_name\_input1.text  
 last\_name = self.last\_name\_input1.text  
 age = self.age1.text  
 phone = self.phone\_input1.text  
 email = self.email\_input1.text  
 street\_address = self.street\_address\_input1.text  
 zip = self.Zip\_input1.text  
 city = self.city\_input1.text  
 state = self.state\_input1.text  
 login\_status = self.login\_status\_spinner1.text  
 emp\_level = self.emp\_level\_spinner1.text  
 pass1 = self.password1.text  
 user = self.user\_id1.text  
  
 # Perform validation checks  
 if not all(  
 [first\_name, last\_name, phone, email, street\_address, city, state, zip, user, pass1, age, login\_status,  
 emp\_level]):  
 self.show\_error\_popup("All fields are required.")  
 return  
 if not age.isdigit():  
 self.show\_error\_popup("Invalid Age.\n Age number must be between 18-45.")  
 return  
 if len(phone) != 10 or not phone.isdigit():  
 self.show\_error\_popup("Invalid phone number.\n Phone number must be 10 digits.")  
 return  
  
 if not [email.endswith('@gmail.com](mailto:email.endswith('@gmail.com)') or not email[0].isalpha():  
 self.show\_error\_popup("Invalid email address. \nEmail must end with @gmail.com")  
 return  
 if len(zip) != 6 or not zip.isdigit():  
 self.show\_error\_popup("Invalid zip number.\n zip number must be 6 digits.")  
 return  
 if not user.isdigit():  
 self.show\_error\_popup("Invalid user .\n user must be digit")  
 return  
 # If all validation checks pass, save employee information to the database  
 try:  
 # Establish a connection to MySQL database  
 conn = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Sahith@12",  
 database="grocerystore"  
 )  
 id = value  
 phone = self.add\_dashes\_to\_number\_with\_existing\_dashes(phone)  
 cursor = conn.cursor()  
 # Construct the UPDATE query  
 update\_query = """  
 UPDATE employee  
 SET first\_name = %s, last\_name = %s, age = %s, phone = %s, mail = %s,  
 street\_address = %s, zip = %s, city = %s, state = %s,  
 employee\_login\_status = %s, emp\_level = %s, emp\_password = %s,  
 emp\_id = %s  
 WHERE emp\_id = %s  
 """  
 cursor.execute(update\_query, (  
 first\_name, last\_name, age, phone, email, street\_address, zip,  
 city, state, login\_status, emp\_level, pass1, user, id  
 ))  
 conn.commit() # Commit the transaction  
  
 # Close the cursor and connection  
 cursor.close()  
 conn.close()  
  
 # Close the popup after update  
  
 # Show success popup  
 self.show\_success\_popup("Employee details \nupdated to database successfully.")  
 self.popup11.dismiss()  
  
 except mysql.connector.Error as e:  
 p = str(e)  
 self.show\_error\_popup("Failed to add employee \n{}".format(  
 p[13:].replace('Duplicate entry', 'Already Exist ').replace('employee.', 'in ').replace('for key',  
 ' ')))  
  
 def show\_error\_popup1(self, message):  
 # Display an error popup with the given message  
 error\_popup = Popup(title='Error', content=Label(text=message), size\_hint=(None, None), size=(300, 200))  
 error\_popup.open()  
  
 def delete\_employee(self):  
  
 # Create a GridLayout to organize input fields  
 self.layout = GridLayout(cols=2, spacing=5, padding=10)  
  
 # Add labels and input fields for user ID and password  
 self.layout.add\_widget(Label(text='emp\_id:',))  
 self.emp\_id\_d = TextInput(multiline=False, background\_color=(0.004, 0.055, 0.102, 1.0),cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 self.layout.add\_widget(self.emp\_id\_d)  
  
 self.layout.add\_widget(Label(text='password:', ))  
 self.password\_d = TextInput(multiline=False, password=True, background\_color=(0.004, 0.055, 0.102, 1.0),cursor\_color=(1, 1, 1, 1), foreground\_color=(1, 1, 1, 1))  
 self.layout.add\_widget(self.password\_d)  
 # Create a submit button  
 self.submit\_button = Button(text='Submit', background\_color=(0.133, 0.855, 0.431, 1.0))  
 self.submit\_button.bind(on\_press=self.dele\_employee)  
 self.layout.add\_widget(self.submit\_button)  
 self.cancel = Button(text='Cancel', background\_color=(0.133, 0.855, 0.431, 1.0))  
 self.cancel.bind(on\_press=self.dismiss\_popup111)  
 self.layout.add\_widget(self.cancel)  
  
 # Create a Popup with the layout  
 self.popup111 = Popup(title='Select Employee', content=self.layout, size\_hint=(None, None),  
 background\_color=(0.004, 0.055, 0.102, 1.0), size=(400, 200))  
 self.popup111.open()  
 def dismiss\_popup11(self, instance=None):  
 self.popup11.dismiss()  
 def dismiss\_popup111(self, instance=None):  
 self.popup111.dismiss()  
 def dele\_employee(self, instance):  
 # Retrieve user ID and password from input fields  
 emp\_id1 = self.emp\_id\_d.text.strip()  
 passs1 = self.password\_d.text.strip()  
 if not all([emp\_id1,passs1]):  
 self.show\_error\_popup1("All fields are required.")  
 return  
 if not emp\_id1.isdigit():  
 self.show\_error\_popup1("Invalid emp\_id number.\n emp\_id must be digit")  
 return  
 # Perform authentication against the database  
 try:  
 # Establish a connection to MySQL database  
 conn = mysql.connector.connect(  
 host="localhost",  
 user="root",  
 password="Sahith@12",  
 database="grocerystore"  
 )  
 cursor = conn.cursor()  
  
 # Execute SELECT query to retrieve employee details based on user ID and password  
 select\_query = "DELETE FROM employee WHERE emp\_id = %s AND emp\_password = %s"  
 cursor.execute(select\_query, (emp\_id1, passs1))  
 conn.commit()  
 cursor.close()  
 conn.close()  
  
 # Show success popup  
 if cursor.rowcount!=0:  
 self.show\_success\_popup("Employee details Deleted to database successfully.")  
 self.popup111.dismiss()  
 else:  
 self.show\_error\_popup1('employee details not exist.')  
  
 except mysql.connector.Error as e:  
 p = str(e)  
 self.show\_error\_popup("Failed to authenticate \n{}".format(  
 p[13:].replace('Duplicate entry', 'Already Exist ').replace('employee.', 'in ').replace('for key',  
 ' ')))

**Employee.kv:**

<EmployeeScreen>:  
 name: "Labor"  
 orientation: 'vertical'  
 padding: 10  
  
 BoxLayout:  
 background\_normal: ''  
 background\_color: 0.004, 0.055, 0.102, 1.0  
  
 orientation: 'vertical'  
 BoxLayout:  
 orientation: 'horizontal'  
 size\_hint: (1, 0.1)  
 background\_normal: ''  
 background\_color: 0.004, 0.055, 0.102, 1.0  
  
  
 Button:  
 text: 'Back'  
 size\_hint: (None, None)  
 size: (100, 40)  
 height: 100  
 pos\_hint: {'left': 0, 'top': 1}  
 background\_normal: ''  
 background\_color: 0.004, 0.055, 0.102, 1.0  
 on\_release:  
 app.root.current = "main"  
 root.manager.transition.direction = "left"  
  
 GridLayout:  
 cols: 2  
 spacing: 30  
 background\_normal: ''  
 background\_color: 0.004, 0.055, 0.102, 1.0  
  
 size\_hint\_y: 0.4 # Adjust this value to control the height  
 pos\_hint: {'center\_x': 0.5}  
  
  
 Button:  
  
 text: 'All Employees'  
 background\_normal: ''  
 background\_color: 0.004, 0.055, 0.102, 1.0  
 on\_press: root.show\_all\_employees()  
  
 Button:  
 text: 'Add Employee'  
 background\_normal: ''  
 background\_color: 0.004, 0.055, 0.102, 1.0  
 on\_press: root.add\_employee()  
  
 Button:  
 text: 'Edit Employee'  
 background\_normal: ''  
 background\_color: 0.004, 0.055, 0.102, 1.0  
 on\_press: root.edit\_employee()  
  
 Button:  
 text: 'Delete Employee'  
 background\_normal: ''  
 background\_color: 0.004, 0.055, 0.102, 1.0  
 on\_press: root.delete\_employee()

**Ring Orders:**



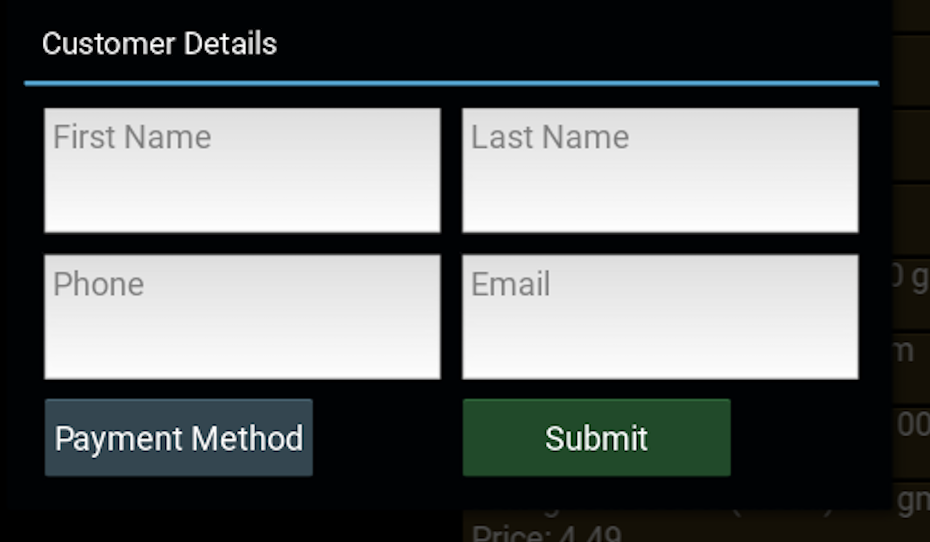
The user can find the list of available products once he clicks on “show all products” button and he can select the products of customer for the list to generate the bill. Discount buttons are provided to add the discount for the customers. A “Remove All” button is provided to delete the entire selected product list.

**Show All Products:**



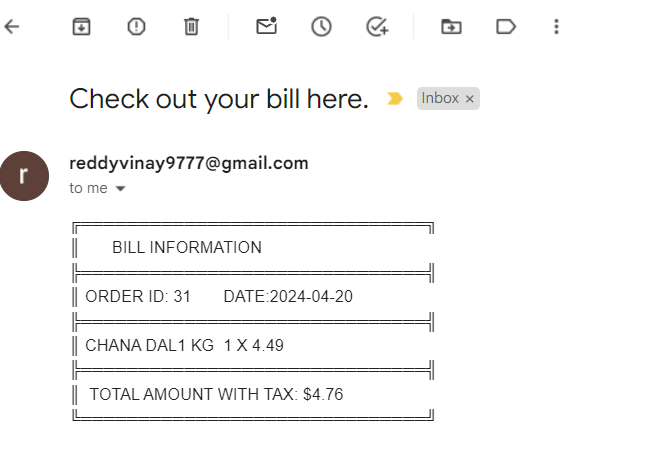
This is how the list of products is displayed.

**Customer details:**



Once the user clicks on the pay bill button a pop-up window appears where user needs to provide the details of customer to generate the bill. A payment method button is provided where the user can select according to the customer's desire such as cash or card.

**Bill Generation:**



A copy of the generated bill is sent to the customer's mail address.