import sqlite3

import csv

from kivy\_garden.matplotlib.backend\_kivyagg import FigureCanvasKivyAgg

from kivy.app import App

from kivy.uix.boxlayout import BoxLayout

from kivy.uix.textinput import TextInput

from kivy.uix.button import Button

from kivy.uix.label import Label

from kivy.uix.recycleview import RecycleView

from kivy.uix.scrollview import ScrollView

from kivy.uix.gridlayout import GridLayout

from kivy.uix.spinner import Spinner

from kivy.core.window import Window

from kivy.graphics import Color, RoundedRectangle,Rectangle

from kivy.uix.screenmanager import ScreenManager, Screen

from datetime import datetime

from kivy.uix.popup import Popup

from collections import defaultdict

import datetime

from datetime import  timedelta

import matplotlib.pyplot as plt

from kivy.utils import get\_color\_from\_hex

from fpdf import FPDF

from kivy.uix.dropdown import DropDown

class ModernDropDown(DropDown):

    def \_\_init\_\_(self, \*\*kwargs):

        super().\_\_init\_\_(\*\*kwargs)

        self.auto\_width = True

        #self.width = 200  # Set a fixed width

    def add\_widget(self, widget, index=0, canvas=None):

        # Apply styling to the buttons in the dropdown

        if isinstance(widget, Button):

            widget.background\_normal = ''

            widget.background\_color = (0.1, 0.1, 0.1, 1)  # Dark background

            widget.color = (0.9, 0.9, 0.9, 1)  # Light text color

            widget.font\_size = 16

            widget.padding = [15, 15]

            with widget.canvas.before:

                Color(0.2, 0.2, 0.2, 1)  # Darker border color

                RoundedRectangle(pos=widget.pos, size=widget.size, radius=[10])

        return super().add\_widget(widget, index=index, canvas=canvas)

class ModernSpinner(Spinner):

    def \_\_init\_\_(self, \*\*kwargs):

        super().\_\_init\_\_(\*\*kwargs)

        self.background\_color = (0.1, 0.1, 0.1, 1)  # Dark background

        self.color = (0.9, 0.9, 0.9, 1)  # Light text color

        self.font\_size = 18

        self.padding = [15, 15]

        self.size\_hint\_y = None

        self.height = 50

        with self.canvas.before:

            Color(0.2, 0.2, 0.2, 1)  # Darker border color

            self.rect = RoundedRectangle(pos=self.pos, size=self.size, radius=[10])

        self.bind(pos=self.update\_rect, size=self.update\_rect)

        self.dropdown\_cls = ModernDropDown  # Assign the custom dropdown class

    def update\_rect(self, \*args):

        self.rect.pos = self.pos

        self.rect.size = self.size

# Set window background color

Window.clearcolor = (0.1, 0.1, 0.1, 1)  # Dark theme

# Database setup

conn = sqlite3.connect("expenses.db")

cursor = conn.cursor()

cursor.execute('''CREATE TABLE IF NOT EXISTS expenses (

                    id INTEGER PRIMARY KEY AUTOINCREMENT,

                    date TEXT,

                    time TEXT,

                    category TEXT,

                    amount REAL,

                    description TEXT,

                    expense\_type TEXT)''')

conn.commit()

class StylishLabel(Label):

    def \_\_init\_\_(self, \*\*kwargs):

        super().\_\_init\_\_(\*\*kwargs)

        self.font\_size = 42  # Slightly larger font size

        self.bold = True

        self.color = (0.1, 0.1, 0.1, 1)  # Darker gray text

        self.size\_hint\_y = None

        self.height = 80  # Slightly taller

        with self.canvas.before:

            Color(0.95, 0.95, 0.95, 1)  # Lighter gray background

            self.rect = RoundedRectangle(pos=self.pos, size=self.size, radius=[15])  # Slightly more rounded corners

            # Subtle shadow with a bit more offset

            Color(0, 0, 0, 0.15)  # Lighter shadow

            RoundedRectangle(pos=(self.x + 5, self.y - 5), size=(self.width - 10, self.height - 10), radius=[15])

        self.bind(pos=self.update\_graphics, size=self.update\_graphics)

    def update\_graphics(self, \*args):

        self.rect.pos = self.pos

        self.rect.size = self.size

class CustomLabel(Label):

    def \_\_init\_\_(self, \*\*kwargs):

        super().\_\_init\_\_(\*\*kwargs)

        self.size\_hint\_y = None

        self.height = 50  # Increased height for more space

        self.font\_size = 20  # Slightly larger font size

        self.color = (1, 1, 1, 1)  # White text

        self.bold = True

        self.padding = (25, 20)  # More padding for better spacing

        with self.canvas.before:

            # Solid background color

            Color(0.15, 0.15, 0.15, 1)  # Slightly darker gray

            self.rect = RoundedRectangle(pos=self.pos, size=self.size, radius=[20])  # Larger radius for smoother corners

            # Inner shadow

            Color(0, 0, 0, 0.3)  # Semi-transparent black

            self.shadow = Rectangle(pos=self.pos, size=self.size)

        self.bind(pos=self.update\_graphics, size=self.update\_graphics)

    def update\_graphics(self, \*args):

        self.rect.pos = self.pos

        self.rect.size = self.size

        self.shadow.pos = (self.x + 5, self.y - 5)  # Offset the shadow slightly

        self.shadow.size = (self.width, self.height)  # Make the shadow slightly smaller

class ExpenseTracker(BoxLayout):

    def \_\_init\_\_(self, screen\_manager, cursor, conn, \*\*kwargs): # Added cursor and conn

        super().\_\_init\_\_(orientation='vertical', padding=50, spacing=20, \*\*kwargs)

        self.background\_color = '#262626'

        self.screen\_manager = screen\_manager

        self.cursor = cursor # Added this line

        self.conn = conn # Added this line

        # Get current date and time

        now = datetime.datetime.now()

        current\_date = now.strftime("%Y-%m-%d")

        current\_day = now.strftime("%d")

        current\_month = now.strftime("%m")

        current\_year = now.strftime("%Y")

        current\_hour = now.strftime("%I")  # 12-hour format

        current\_minute = now.strftime("%M")

        current\_ampm = now.strftime("%p")

        # Header

        self.add\_widget(StylishLabel(text="EXPENSE TRACKER"))

        # Input Layout

        input\_layout = GridLayout(cols=2, spacing=15, size\_hint\_y=None, height=350)

        input\_style = {'size\_hint\_y': None, 'height': 50, 'font\_size': 18, 'foreground\_color': (0.9, 0.9, 0.9, 1),

                       'background\_color': (0, 0, 0, 1), 'padding': [15, 15], 'halign': 'center'}

        # Date Selection

        self.day\_spinner = ModernSpinner(text=current\_day, values=[str(i).zfill(2) for i in range(1, 32)])

        self.month\_spinner = ModernSpinner(text=current\_month, values=[str(i).zfill(2) for i in range(1, 13)])

        self.year\_spinner = ModernSpinner(text=current\_year, values=[str(i) for i in range(2000, 2051)])

        date\_layout = GridLayout(cols=3, spacing=5, size\_hint\_y=None, height=50)

        date\_layout.add\_widget(self.day\_spinner)

        date\_layout.add\_widget(self.month\_spinner)

        date\_layout.add\_widget(self.year\_spinner)

        input\_layout.add\_widget(CustomLabel(text='Date:'))

        input\_layout.add\_widget(date\_layout)

        # Time Selection

        self.hour\_spinner = ModernSpinner(text=current\_hour, values=[str(i).zfill(2) for i in range(1, 13)])

        self.minute\_spinner = ModernSpinner(text=current\_minute, values=[str(i).zfill(2) for i in range(0, 60)])

        self.ampm\_spinner = ModernSpinner(text=current\_ampm, values=['AM', 'PM'])

        time\_layout = GridLayout(cols=3, spacing=5, size\_hint\_y=None, height=50)

        time\_layout.add\_widget(self.hour\_spinner)

        time\_layout.add\_widget(self.minute\_spinner)

        time\_layout.add\_widget(self.ampm\_spinner)

        input\_layout.add\_widget(CustomLabel(text='Time:'))

        input\_layout.add\_widget(time\_layout)

        # Category Dropdown

        self.category\_spinner = ModernSpinner(text='Select Category', values=('Food', 'Transport', 'Shopping', 'Entertainment', 'Bills', 'Others'),

                                         size\_hint\_y=None, height=50)

        input\_layout.add\_widget(CustomLabel(text='Category:'))

        input\_layout.add\_widget(self.category\_spinner)

        # Amount and Description Fields

        self.amount\_input = TextInput(hint\_text='Amount',\*\*input\_style)

        self.desc\_input = TextInput(hint\_text='Description', \*\*input\_style)

        # Expense Type Dropdown

        self.expense\_type\_spinner = ModernSpinner(

            text="Expense",

            values=("Expense", "Income"),

        )

        input\_layout.add\_widget(CustomLabel(text='Amount:'))

        input\_layout.add\_widget(self.amount\_input)

        input\_layout.add\_widget(CustomLabel(text='Description:'))

        input\_layout.add\_widget(self.desc\_input)

        input\_layout.add\_widget(CustomLabel(text='Type:'))

        input\_layout.add\_widget(self.expense\_type\_spinner)

        self.add\_widget(input\_layout)

        # Buttons

        button\_layout = GridLayout(cols=4, spacing=15, size\_hint\_y=None, height=60)  # changed to 4 columns.

        self.add\_button = Button(text='Add',background\_normal='',background\_color=(0, 0, 0, 1), font\_size=18, bold=True, size\_hint\_y=None, height=50)

        self.add\_button.bind(on\_press=self.add\_expense)

        self.update\_button = Button(text='Update',background\_normal='', background\_color=(0, 0, 0, 1), font\_size=18, bold=True, size\_hint\_y=None, height=50)

        self.update\_button.bind(on\_press=self.open\_history\_page)

        self.delete\_button = Button(text='Delete',background\_normal='', background\_color=(0, 0, 0, 1), font\_size=18, bold=True, size\_hint\_y=None, height=50)

        self.delete\_button.bind(on\_press=self.open\_history\_page)  # Change function to open history

        self.history\_button = Button(text='View History', background\_normal='',background\_color=(0, 0, 0, 1), font\_size=18, bold=True, size\_hint\_y=None, height=50)

        self.history\_button.bind(on\_press=self.view\_history)

        self.budget\_button = Button(text='Set Budget',background\_normal='',background\_color=(0, 0, 0, 1), font\_size=18, bold=True, size\_hint\_y=None, height=50)

        self.budget\_button.bind(on\_press=self.open\_budget\_page)  # new button and binding.

        self.view\_budget\_button = Button(text='View Budgets',background\_normal='', background\_color=(0, 0, 0, 1), font\_size=18, bold=True, size\_hint\_y=None, height=50)

        self.view\_budget\_button.bind(on\_press=self.open\_view\_budgets\_page)  # new button and binding.

        self.reports\_button = Button(text='Generate Reports',background\_normal='', background\_color=(0, 0, 0, 1),font\_size=18, bold=True, size\_hint\_y=None, height=50)

        self.reports\_button.bind(on\_press=self.open\_reports\_page)  # Bind to the function

        button\_layout.add\_widget(self.add\_button)

        button\_layout.add\_widget(self.update\_button)

        button\_layout.add\_widget(self.delete\_button)

        button\_layout.add\_widget(self.history\_button)

        button\_layout.add\_widget(self.budget\_button)  # add the budget button.

        button\_layout.add\_widget(self.view\_budget\_button)  # add the view budget button.

        button\_layout.add\_widget(self.reports\_button)  # Add button to layout

        self.add\_widget(BoxLayout(size\_hint\_y=0.1))  # Spacer

        self.add\_widget(button\_layout)

        self.add\_widget(BoxLayout(size\_hint\_y=1))  # Spacer

    def check\_budget(self, date\_obj, category, amount):

        assert isinstance(amount, float), "Amount should be a float in check\_budget"

        cursor = self.cursor

        conn = self.conn

        try:

            # Daily Check

            self.check\_daily\_budgets(cursor, date\_obj, category, amount)

            # Weekly Check

            start\_week = date\_obj - timedelta(days=date\_obj.weekday())

            end\_week = start\_week + timedelta(days=6)

            self.check\_weekly\_budgets(cursor, start\_week, end\_week, category, amount)

            # Monthly Check

            start\_month = date\_obj.replace(day=1)

            end\_month = (start\_month.replace(month=start\_month.month + 1) - timedelta(days=1))

            self.check\_monthly\_budgets(cursor, start\_month, end\_month, category, amount)

            # Yearly Check

            start\_year = date\_obj.replace(month=1, day=1)

            end\_year = date\_obj.replace(month=12, day=31)

            self.check\_yearly\_budgets(cursor, start\_year, end\_year, category, amount)

        except sqlite3.Error as e:

            print(f"Database error during budget check: {e}")

            self.show\_popup("Database Error", f"Database error: {e}")

        except Exception as e:

            print(f"Error checking budget: {e}")

            self.show\_popup("Error", f"Error checking budget: {e}")

    def check\_daily\_budgets(self, cursor, date\_obj, category, amount):

        self.check\_budget\_type(cursor, date\_obj, date\_obj, "Daily", category, amount)

        if category != "All":

            self.check\_budget\_type(cursor, date\_obj, date\_obj, "Daily", "All", amount)

    def check\_weekly\_budgets(self, cursor, start\_week, end\_week, category, amount):

        self.check\_budget\_type(cursor, start\_week, end\_week, "Weekly", category, amount)

        if category != "All":

            self.check\_budget\_type(cursor, start\_week, end\_week, "Weekly", "All", amount)

    def check\_monthly\_budgets(self, cursor, start\_month, end\_month, category, amount):

        self.check\_budget\_type(cursor, start\_month, end\_month, "Monthly", category, amount)

        if category != "All":

            self.check\_budget\_type(cursor, start\_month, end\_month, "Monthly", "All", amount)

    def check\_yearly\_budgets(self, cursor, start\_year, end\_year, category, amount):

        self.check\_budget\_type(cursor, start\_year, end\_year, "Yearly", category, amount)

        if category != "All":

            self.check\_budget\_type(cursor, start\_year, end\_year, "Yearly", "All", amount)

    def check\_budget\_type(self, cursor, start\_date, end\_date, budget\_type, category, amount):

        assert isinstance(amount, float), "Amount should be a float in check\_budget\_type"

        cursor.execute("SELECT amount FROM budgets WHERE budget\_type = ? AND (category = ? OR category = 'All')", (budget\_type, category))

        budget\_result = cursor.fetchone()

        if budget\_result:

            budget\_amount = float(budget\_result[0])

            if category == "All":

                cursor.execute("SELECT SUM(amount) FROM expenses WHERE date BETWEEN ? AND ?", (str(start\_date), str(end\_date)))

            else:

                cursor.execute("SELECT SUM(amount) FROM expenses WHERE date BETWEEN ? AND ? AND category = ?", (str(start\_date), str(end\_date), category))

            total\_spent\_result = cursor.fetchone()

            total\_spent = total\_spent\_result[0] if total\_spent\_result and total\_spent\_result[0] else 0.0

            if total\_spent + amount > budget\_amount:

                exceeded\_by = (total\_spent + amount) - budget\_amount

                self.show\_popup("Overspending", f"You've exceeded your {budget\_type} budget for {category} by {exceeded\_by:.2f}!")

    def show\_popup(self, title, message):

        content = BoxLayout(orientation='vertical')

        content.add\_widget(Label(text=message))

        close\_button = Button(text='Close', size\_hint\_y=None, height=40)

        content.add\_widget(close\_button)

        popup = Popup(title=title, content=content, size\_hint=(None, None), size=(600, 300))

        close\_button.bind(on\_press=popup.dismiss)

        popup.open()

    def open\_view\_budgets\_page(self, instance):

        budgets\_screen = self.screen\_manager.get\_screen("view\_budgets")

        budgets\_screen.load\_budgets()

        self.screen\_manager.current = "view\_budgets"

    def open\_reports\_page(self, instance):

        self.screen\_manager.current = 'reports'  # Navigate to ReportsScreen

    def open\_budget\_page(self, instance):

        self.screen\_manager.current = "budgets"

    def add\_expense(self, instance):

        date = f"{self.year\_spinner.text}-{self.month\_spinner.text}-{self.day\_spinner.text}"

        time = f"{self.hour\_spinner.text}:{self.minute\_spinner.text} {self.ampm\_spinner.text}"

        category = self.category\_spinner.text.strip()

        amount = self.amount\_input.text.strip()

        description = self.desc\_input.text.strip()

        expense\_type = self.expense\_type\_spinner.text.strip()

        if category == "Select Category" or not amount or not description:

            self.show\_popup("Input Error", "Some fields are empty!")

            return

        try:

            amount = float(amount)  # Convert amount to float and assign back to amount

        except ValueError:

            self.show\_popup("Input Error", "Amount must be a numeric value.")

            return

        try:

            if hasattr(self, "selected\_expense\_id") and self.selected\_expense\_id:

                self.cursor.execute("UPDATE expenses SET date=?, time=?, category=?, amount=?, description=? WHERE id=?",

                                    (date, time, category, amount, description, self.selected\_expense\_id))

                self.conn.commit()

                self.show\_popup("Expense Updated", "Expense updated successfully!")

                self.selected\_expense\_id = None

            else:

                self.cursor.execute("INSERT INTO expenses (date, time, category, amount, description, expense\_type) VALUES (?, ?, ?, ?, ?, ?)",

                                    (date, time, category, amount, description, expense\_type))

                self.conn.commit()

                self.show\_popup(f"{expense\_type} Added", f"{expense\_type} added successfully!")

            self.amount\_input.text = ""

            self.desc\_input.text = ""

            history\_screen = self.screen\_manager.get\_screen("history")

            history\_screen.load\_history()

            # Check budget after adding/updating expense

            date\_obj = datetime.datetime.strptime(date, "%Y-%m-%d").date()

            self.check\_budget(date\_obj, category, amount)

        except sqlite3.Error as e:

            self.show\_popup("Database Error", f"Database error: {e}")

        except Exception as e:

            self.show\_popup("Database Error", f"Error: {e}")

    def open\_history\_page(self, instance):

        history\_screen = self.screen\_manager.get\_screen("history")

        history\_screen.load\_history()

        if instance == self.update\_button:

            history\_screen.delete\_button.text = "Update Selected"

            history\_screen.delete\_button.background\_color = self.update\_button.background\_color

            history\_screen.delete\_button.unbind(on\_press=history\_screen.delete\_selected\_expense)

            # Corrected line: Use lambda to capture expense\_id

            history\_screen.delete\_button.bind(on\_press=lambda instance: history\_screen.load\_expense\_for\_editing())

        else:

            history\_screen.delete\_button.text = "Delete Selected"

            history\_screen.delete\_button.background\_color = self.delete\_button.background\_color

            history\_screen.delete\_button.unbind(on\_press=history\_screen.load\_expense\_for\_editing)

            history\_screen.delete\_button.bind(on\_press=history\_screen.delete\_selected\_expense)

        history\_screen.delete\_button.opacity = 1

        history\_screen.delete\_button.disabled = False

        self.screen\_manager.current = "history"

    def update\_expense(self, instance):

        if not hasattr(self, "selected\_expense\_id") or not self.selected\_expense\_id:

            print("No expense selected for update.")

            return

        date = f"{self.year\_spinner.text}-{self.month\_spinner.text}-{self.day\_spinner.text}"

        time = f"{self.hour\_spinner.text}:{self.minute\_spinner.text} {self.ampm\_spinner.text}"

        category = self.category\_spinner.text.strip()

        amount = self.amount\_input.text.strip()

        description = self.desc\_input.text.strip()

        if category == "Select Category" or not amount:

            print("Error: Some fields are empty!")

            return

        try:

            amount = float(amount)  # Convert amount to float

            cursor.execute("""

                UPDATE expenses

                SET date=?, time=?, category=?, amount=?, description=?

                WHERE id=?

            """, (date, time, category, amount, description, self.selected\_expense\_id))

            conn.commit()

            print("Expense updated successfully!")

            self.selected\_expense\_id = None  # Clear selection

            self.amount\_input.text = ""

            self.desc\_input.text = ""

            # Refresh history screen

            history\_screen = self.screen\_manager.get\_screen("history")

            history\_screen.load\_history()

            # Navigate back to history page

            self.screen\_manager.current = "history"

        except ValueError:

            print("Invalid amount. Please enter a numeric value.")

        except Exception as e:

            print("Error while updating:", e)

    def delete\_expense(self, instance):

        if not self.selected\_expense\_id:

            return

        try:

            cursor.execute("DELETE FROM expenses WHERE id=?", (self.selected\_expense\_id,))

            conn.commit()

            print("Expense deleted successfully!")

            self.selected\_expense\_id = None

        except Exception as e:

            print("Error:", e)

    def view\_history(self, instance):

        history\_screen = self.screen\_manager.get\_screen("history")

        history\_screen.load\_history()

        history\_screen.delete\_button.opacity = 0  # Hide delete button

        history\_screen.delete\_button.disabled = True

        self.screen\_manager.current = "history"

    def open\_update\_page(self, instance):

        self.screen\_manager.current = "history"  # Open history screen

        history\_screen = self.screen\_manager.get\_screen("history")

        history\_screen.set\_update\_mode(True)  # Enable update mode

class ViewBudgetsScreen(Screen):

    def \_\_init\_\_(self, \*\*kwargs):

        super().\_\_init\_\_(\*\*kwargs)

        layout = BoxLayout(orientation='vertical', padding=20, spacing=10, size\_hint=(1, 1))

        layout.add\_widget(StylishLabel(text="Budget List"))

        scroll\_view = ScrollView(size\_hint\_y=1, pos\_hint={'top': 1})

        self.budgets\_list = GridLayout(cols=4, spacing=10, size\_hint\_y=None) #increased cols to 4

        self.budgets\_list.bind(minimum\_height=self.budgets\_list.setter('height'))

        scroll\_view.add\_widget(self.budgets\_list)

        layout.add\_widget(scroll\_view)

        back\_button = Button(text="Back", size\_hint\_y=None, height=50, background\_color=(0.2, 0.6, 1, 1))

        back\_button.bind(on\_press=self.go\_back)

        layout.add\_widget(back\_button)

        self.add\_widget(layout)

    def load\_budgets(self):

        try:

            self.budgets\_list.clear\_widgets()

            self.budgets\_list.cols = 4 #Increased to 4

            headers = ["Budget Type", "Category", "Amount", "Action"] # Added "Action" header

            for header in headers:

                label = Label(text=header, bold=True, size\_hint\_y=None, height=30, color=(0, 1, 1, 1))

                self.budgets\_list.add\_widget(label)

            conn = sqlite3.connect("expenses.db")

            cursor = conn.cursor()

            cursor.execute("SELECT id, budget\_type, category, amount FROM budgets") #added id to select

            records = cursor.fetchall()

            for row in records:

                budget\_id, budget\_type, category, amount = row #added budget\_id

                labels = [budget\_type, category, f"₹{amount}"]

                for text in labels:

                    label = Label(text=text, size\_hint\_y=None, height=30, color=(0.5, 0.5, 0.5, 1), font\_size = 14)

                    self.budgets\_list.add\_widget(label)

                delete\_button = Button(text="X", size\_hint\_y=None, height=30, background\_color=(1, 0, 0, 1))

                delete\_button.bind(on\_press=lambda instance, budget\_id=budget\_id: self.delete\_budget(budget\_id)) #lambda function to pass budget id

                self.budgets\_list.add\_widget(delete\_button)

            self.budgets\_list.height = self.budgets\_list.minimum\_height

            self.budgets\_list.parent.scroll\_y = 1

        except Exception as e:

            print(f"Error loading budgets: {e}")

    def delete\_budget(self, budget\_id):

        try:

            conn = sqlite3.connect("expenses.db")

            cursor = conn.cursor()

            cursor.execute("DELETE FROM budgets WHERE id = ?", (budget\_id,))

            conn.commit()

            self.load\_budgets() #reload budgets after deletion

        except Exception as e:

            print(f"Error deleting budget: {e}")

    def go\_back(self, instance):

        self.manager.current = "main"

class BudgetsScreen(Screen):

    def \_\_init\_\_(self, \*\*kwargs):

        super().\_\_init\_\_(\*\*kwargs)

        layout = BoxLayout(orientation='vertical', padding=20, spacing=10, size\_hint=(1, 1))

        layout.add\_widget(StylishLabel(text="Set Budgets"))

        input\_layout = GridLayout(cols=2, spacing=15, size\_hint\_y=None, height=200)

        self.budget\_type\_spinner = Spinner(text="Daily", values=("Daily", "Weekly", "Monthly", "Yearly"), size\_hint\_y=None, height=50)

        self.budget\_category\_spinner = Spinner(text="All", values=("All", "Food", "Transport", "Shopping", "Entertainment", "Bills", "Others"), size\_hint\_y=None, height=50)

        self.budget\_amount\_input = TextInput(hint\_text="Budget Amount", size\_hint\_y=None, height=50)

        input\_layout.add\_widget(CustomLabel(text="Budget Type:"))

        input\_layout.add\_widget(self.budget\_type\_spinner)

        input\_layout.add\_widget(CustomLabel(text="Category:"))

        input\_layout.add\_widget(self.budget\_category\_spinner)

        input\_layout.add\_widget(CustomLabel(text="Amount:"))

        input\_layout.add\_widget(self.budget\_amount\_input)

        layout.add\_widget(input\_layout)

        set\_budget\_button = Button(text="Set Budget", size\_hint\_y=None, height=50, background\_color=(0.8, 0.4, 0.2, 1))

        set\_budget\_button.bind(on\_press=self.set\_budget)

        layout.add\_widget(set\_budget\_button)

        back\_button = Button(text="Back", size\_hint\_y=None, height=50, background\_color=(0.2, 0.6, 1, 1))

        back\_button.bind(on\_press=self.go\_back)

        layout.add\_widget(back\_button)

        layout.add\_widget(BoxLayout(size\_hint\_y=1)) # Spacer to push content to top.

        self.add\_widget(layout)

    def set\_budget(self, instance):

        # Add your budget setting logic here (database insertion, etc.)

        budget\_type = self.budget\_type\_spinner.text

        category = self.budget\_category\_spinner.text

        amount = self.budget\_amount\_input.text

        try:

            amount = float(amount)

            cursor.execute("INSERT INTO budgets (budget\_type, category, amount) VALUES (?, ?, ?)", (budget\_type, category, amount))

            conn.commit()

            self.show\_popup("budget added","budget added successfully")

        except:

            self.show\_popup("error", "invalid amount")

    def go\_back(self, instance):

        self.manager.current = "main"

    def show\_popup(self, title, message):

        content = BoxLayout(orientation='vertical')

        content.add\_widget(Label(text=message))

        close\_button = Button(text='Close', size\_hint\_y=None, height=40)

        content.add\_widget(close\_button)

        popup = Popup(title=title, content=content, size\_hint=(None, None), size=(400, 200))

        close\_button.bind(on\_press=popup.dismiss) # Bind close button to dismiss popup

        popup.open()

class ReportsScreen(Screen):

    def \_\_init\_\_(self, \*\*kwargs):

        super().\_\_init\_\_(\*\*kwargs)

        # Main Vertical Layout

        main\_layout = BoxLayout(orientation='vertical', padding=10, spacing=10)

        # Title Label (Always on top)

        main\_layout.add\_widget(Label(text="EXPENSE REPORTS", font\_size=24, bold=True, size\_hint\_y=None, height=75))

        # Charts Box (STRICTLY HALF THE SCREEN)

        self.chart\_container = BoxLayout(orientation='horizontal', size\_hint\_y=0.4, spacing=10)

        main\_layout.add\_widget(self.chart\_container)

        # Summary Layout (Centered, fills remaining space)

        self.summary\_layout = BoxLayout(orientation='vertical', spacing=5, padding=10, size\_hint\_y=0.2)

        main\_layout.add\_widget(self.summary\_layout)

        # Lower Section (Filters + Buttons)

        lower\_section = BoxLayout(orientation='vertical', size\_hint\_y=0.4, spacing=10)

        # Filters Layout (Period Selection & Date Inputs)

        filter\_layout = BoxLayout(size\_hint\_y=None, height=40, spacing=10)

        # Report Type Dropdown (Half Width)

        self.report\_type = Spinner(

            text="Daily",

            values=("Daily", "Weekly", "Monthly", "Yearly", "Custom"),

            size\_hint=(0.5, None),

            size=(120, 50)

        )

        filter\_layout.add\_widget(self.report\_type)

        self.report\_type.bind(text=self.on\_report\_type\_change)

        # Custom Date Inputs (Evenly Split Other Half)

        date\_inputs\_layout = BoxLayout(size\_hint=(0.5, None), height=40, spacing=10)

        self.start\_date\_input = TextInput(hint\_text="Start Date (YYYY-MM-DD)", size\_hint=(0.5, None), size=(140, 50))

        self.end\_date\_input = TextInput(hint\_text="End Date (YYYY-MM-DD)", size\_hint=(0.5, None), size=(140, 50))

        self.start\_date\_input.disabled = True

        self.end\_date\_input.disabled = True

        date\_inputs\_layout.add\_widget(self.start\_date\_input)

        date\_inputs\_layout.add\_widget(self.end\_date\_input)

        filter\_layout.add\_widget(date\_inputs\_layout)

        lower\_section.add\_widget(filter\_layout)

        # Button Layout (More Evenly Spaced)

        button\_layout = GridLayout(cols=3, spacing=10, size\_hint\_y=None, height=50)

        # Generate Reports Button

        self.generate\_button = Button(text="Generate Report", background\_color=(0, 1, 0, 1))

        self.generate\_button.bind(on\_press=self.generate\_reports)

        button\_layout.add\_widget(self.generate\_button)

        # Export CSV Button

        csv\_button = Button(text="Export CSV", background\_color=(0.8, 0.5, 0, 1))

        csv\_button.bind(on\_press=self.export\_csv)

        button\_layout.add\_widget(csv\_button)

        # Export PDF Button

        pdf\_button = Button(text="Export PDF", background\_color=(0.5, 0.2, 0.8, 1))

        pdf\_button.bind(on\_press=self.export\_pdf)

        button\_layout.add\_widget(pdf\_button)

        lower\_section.add\_widget(button\_layout)

        # Back Button (Smaller Height)

        back\_button = Button(text="Back", size\_hint\_x=1, size\_hint\_y=None, height=50, background\_color=(0.2, 0.6, 1, 1))

        back\_button.bind(on\_press=self.go\_back)

        lower\_section.add\_widget(back\_button)

        main\_layout.add\_widget(lower\_section)

        self.add\_widget(main\_layout)

        # Generate default daily report

        self.generate\_reports(None)

    def on\_report\_type\_change(self, spinner, text):

        if text == "Custom":

            self.start\_date\_input.disabled = False

            self.end\_date\_input.disabled = False

        else:

            self.start\_date\_input.disabled = True

            self.end\_date\_input.disabled = True

    def generate\_reports(self, instance):

        self.chart\_container.clear\_widgets()

        self.summary\_layout.clear\_widgets()

        report\_type = self.report\_type.text

        start\_date = self.start\_date\_input.text

        end\_date = self.end\_date\_input.text

        data = self.fetch\_expense\_data(report\_type, start\_date, end\_date)

        if data:

            pie\_chart, bar\_chart = self.create\_charts(data)

            self.chart\_container.add\_widget(FigureCanvasKivyAgg(pie\_chart))

            self.chart\_container.add\_widget(FigureCanvasKivyAgg(bar\_chart))

            pie\_summary = self.generate\_pie\_summary(data)

            bar\_summary = self.generate\_bar\_summary(data)

            # Layout for Summary Boxes

            summaries\_box = BoxLayout(orientation='horizontal', spacing=00, padding=(10, 00, 10, 0))

            # Pie Chart Summary Grid

            pie\_grid = GridLayout(cols=1, spacing=0, size\_hint\_x=0.5, padding=(10, 0))

            # Bar Chart Summary Grid

            bar\_grid = GridLayout(cols=1, spacing=0, size\_hint\_x=0.5, padding=(10, 0))

            # Pie Chart Summary Title

            pie\_label = Label(text="[b][size=20]Pie Chart Summary[/size][/b]", markup=True, halign='center', size\_hint\_y=None, height=40)

            pie\_grid.add\_widget(pie\_label)

            # Horizontal Divider

            pie\_grid.add\_widget(Label(text="--------------------------------------", bold=True, size\_hint\_y=None, height=20))

            for line in pie\_summary.split('\n'):

                if line:

                    category, value\_percentage = line.split(': ', 1) if ': ' in line else ("", line)

                    label = Label(text=f"[b]{category}:[/b] {value\_percentage}", markup=True, halign='left', valign='middle', size\_hint\_y=None, height=30)

                    pie\_grid.add\_widget(label)

            # Bar Chart Summary Title

            bar\_label = Label(text="[b][size=20]Bar Chart Summary[/size][/b]", markup=True, halign='center', size\_hint\_y=None, height=40)

            bar\_grid.add\_widget(bar\_label)

            # Horizontal Divider

            bar\_grid.add\_widget(Label(text="--------------------------------------", bold=True, size\_hint\_y=None, height=20))

            for line in bar\_summary.split('\n'):

                if line.strip():  # Ensure line is not empty

                    if ': ' in line:

                        category, value\_percentage = line.split(': ', 1)

                        label\_text = f"[b]{category}:[/b] {value\_percentage}"

                    else:

                        label\_text = f"[b]{line}[/b]"  # Display without extra colon

                    label = Label(text=label\_text, markup=True, halign='left', valign='middle', size\_hint\_y=None, height=30)

                    bar\_grid.add\_widget(label)

            # Add Grids to the Summary Box

            summaries\_box.add\_widget(pie\_grid)

            summaries\_box.add\_widget(bar\_grid)

            # Add Spacing & Summaries to Layout

            spacer = Label(size\_hint\_y=None, height=40)

            self.summary\_layout.add\_widget(spacer)

            self.summary\_layout.add\_widget(summaries\_box)

            # Adjust height based on content

            self.summary\_layout.height = self.summary\_layout.minimum\_height

        else:

            popup = Popup(title="No Data", content=Label(text="No expenses found for the selected period."), size\_hint=(0.6, 0.3))

            popup.open()

    def generate\_pie\_summary(self, data):

        if not data:

            return "No pie chart data available."

        total\_expenses = sum(amount for \_, amount in data)

        category\_summaries = ""

        for category, amount in data:

            percentage = (amount / total\_expenses) \* 100

            category\_summaries += f"{category}: [b]{amount:.2f} ({percentage:.2f}%)[/b]\n"

        return f"{category\_summaries}"

    def generate\_bar\_summary(self, data):

        if not data:

            return "No bar chart data available."

        total\_expenses = sum(amount for \_, amount in data)

        income = total\_expenses \* 1.2

        expense = total\_expenses

        income\_percentage = (income / (income + expense)) \* 100

        expense\_percentage = (expense / (income + expense)) \* 100

        return f"[b]Total Income:[/b] {income:.2f} ({income\_percentage:.2f}%)\n[b]Total Expense:[/b] {expense:.2f} ({expense\_percentage:.2f}%)"

    def fetch\_expense\_data(self, report\_type, start\_date, end\_date):

        conn = sqlite3.connect("expenses.db")

        cursor = conn.cursor()

        query = """

        SELECT category, SUM(amount)

        FROM expenses

        WHERE date BETWEEN ? AND ?

        GROUP BY category

        ORDER BY category ASC

        """

        if report\_type == "Daily":

            start\_date = end\_date = datetime.date.today().strftime("%Y-%m-%d")

        elif report\_type == "Weekly":

            start\_date = (datetime.date.today() - datetime.timedelta(days=7)).strftime("%Y-%m-%d")

            end\_date = datetime.date.today().strftime("%Y-%m-%d")

        elif report\_type == "Monthly":

            start\_date = (datetime.date.today().replace(day=1)).strftime("%Y-%m-%d")

            end\_date = datetime.date.today().strftime("%Y-%m-%d")

        elif report\_type == "Yearly":

            start\_date = (datetime.date.today().replace(month=1, day=1)).strftime("%Y-%m-%d")

            end\_date = datetime.date.today().strftime("%Y-%m-%d")

        elif report\_type == "Custom":

            if not start\_date or not end\_date:

                return None

        cursor.execute(query, (start\_date, end\_date))

        data = cursor.fetchall()

        conn.close()

        return data

    def create\_charts(self, data):

        categories, values = zip(\*data) if data else ([], [])

        # Pie Chart with Black Background and White Text

        fig1, ax1 = plt.subplots(figsize=(4, 4))

        ax1.pie(values, labels=categories, autopct='%1.1f%%', startangle=140, textprops={'color': 'white'})  # White text

        ax1.set\_title("Category-wise Expense Breakdown", color='white')  # White title

        ax1.set\_facecolor('#262626')  # Black background

        fig1.patch.set\_facecolor('#262626')  # Black figure background

        # Bar Chart with Black Background and White Text

        fig2, ax2 = plt.subplots()

        ax2.bar(["Total Income", "Total Expense"], [sum(values) \* 1.2, sum(values)], color=["green", "red"])

        ax2.set\_title("Total Income vs Expense", color='white')  # White title

        ax2.set\_facecolor('#262626')  # Black background

        fig2.patch.set\_facecolor('#262626')  # Black figure background

        # Set tick and axis label colors to white

        ax2.tick\_params(axis='x', colors='white')

        ax2.tick\_params(axis='y', colors='white')

        ax2.xaxis.label.set\_color('white')

        ax2.yaxis.label.set\_color('white')

        return fig1, fig2

    def export\_csv(self, instance):

        report\_type = self.report\_type.text

        start\_date = self.start\_date\_input.text

        end\_date = self.end\_date\_input.text

        data = self.fetch\_expense\_data(report\_type, start\_date, end\_date)  # Fetch the data

        if data:

            # 1. Determine the filename

            filename = f"expense\_report\_{report\_type}\_{start\_date}\_{end\_date}.csv"  # Example filename

            # 2. Open the CSV file in write mode

            with open(filename, 'w', newline='') as csvfile:

                writer = csv.writer(csvfile)

                # 3. Write the header row (if needed)

                if report\_type == "category":  # Example: Add header for category report

                    writer.writerow(["Category", "Amount", "Percentage"])

                elif report\_type == "income\_vs\_expense":

                    writer.writerow(["Type", "Amount", "Percentage"])

                # 4. Write the data rows

                for row in data:

                    writer.writerow(row)  # Assuming 'data' is a list of lists or tuples

            # (Optional) Display a success message or open the file

            popup = Popup(title="Success", content=Label(text=f"Report exported to {filename}"), size\_hint=(0.6, 0.3))

            popup.open()

        else:

            # Show "No Data" popup (you already have this part)

            popup = Popup(title="No Data", content=Label(text="No expenses found for the selected period."), size\_hint=(0.6, 0.3))

            popup.open()

    def export\_pdf(self, instance):

        report\_type = self.report\_type.text

        start\_date = self.start\_date\_input.text

        end\_date = self.end\_date\_input.text

        data = self.fetch\_expense\_data(report\_type, start\_date, end\_date)

        if data:

            # 1. Create a PDF object

            pdf = FPDF()

            pdf.add\_page()

            pdf.set\_font("Arial", size=12)

            # 2. Add title

            pdf.cell(200, 10, txt=f"Expense Report ({report\_type})", ln=True, align="C")

            pdf.cell(200, 10, txt=f"From {start\_date} to {end\_date}", ln=True, align="C")

            # 3. Add summary data to the PDF

            pie\_summary = self.generate\_pie\_summary(data)

            bar\_summary = self.generate\_bar\_summary(data)

            pdf.ln(10)  # Add some space

            # Add Pie Chart Summary

            pdf.cell(200, 10, txt="Pie Chart Summary", ln=True, align="L")

            for line in pie\_summary.split('\n'):

                pdf.cell(200, 10, txt=line, ln=True, align="L")

            pdf.ln(10)

            # Add Bar Chart Summary

            pdf.cell(200, 10, txt="Bar Chart Summary", ln=True, align="L")

            for line in bar\_summary.split('\n'):

                pdf.cell(200, 10, txt=line, ln=True, align="L")

            # 4. Add charts to the PDF

            pie\_chart, bar\_chart = self.create\_charts(data)

            pie\_chart\_path = "pie\_chart.png"

            bar\_chart\_path = "bar\_chart.png"

            pie\_chart.savefig(pie\_chart\_path)

            bar\_chart.savefig(bar\_chart\_path)

            pdf.image(pie\_chart\_path, x=10, y=100, w=100)

            pdf.image(bar\_chart\_path, x=110, y=100, w=100)

            # 5. Save the PDF

            pdf\_filename = f"expense\_report\_{report\_type}\_{start\_date}\_{end\_date}.pdf"

            pdf.output(pdf\_filename)

            # (Optional) Display a success message or open the file

            popup = Popup(title="Success", content=Label(text=f"Report exported to {pdf\_filename}"), size\_hint=(0.6, 0.3))

            popup.open()

        else:

            # Show "No Data" popup (you already have this part)

            popup = Popup(title="No Data", content=Label(text="No expenses found for the selected period."), size\_hint=(0.6, 0.3))

            popup.open()

    def go\_back(self, instance):

        self.manager.current = 'main'

class HistoryScreen(Screen):

    def \_\_init\_\_(self, \*\*kwargs):

        super().\_\_init\_\_(\*\*kwargs)

        layout = BoxLayout(orientation='vertical', padding=20, spacing=10, size\_hint=(1, 1)) # Add size\_hint

        # Header

        layout.add\_widget(StylishLabel(text="EXPENSE HISTORY"))

        # Total Expense, Income, Net Labels in a Horizontal BoxLayout

        totals\_layout = BoxLayout(orientation='horizontal', size\_hint\_y=None, height=50) #Horizontal boxlayout for totals.

        # Total Expense Label

        self.total\_expense\_label = Label(text="Total Expense: ₹0.00", font\_size=20, bold=True, color=(1, 1, 0, 1),

                                         size\_hint\_y=None, height=50)

        totals\_layout.add\_widget(self.total\_expense\_label)

        #Total Income Label

        self.total\_income\_label = Label(text="Total Income: ₹0.00", font\_size=20, bold=True, color=(1, 1, 0, 1),

                                         size\_hint\_y=None, height=50)

        totals\_layout.add\_widget(self.total\_income\_label)

        #Net Label

        self.net\_label = Label(text="Net: ₹0.00", font\_size=20, bold=True, color=(1, 1, 0, 1),

                                         size\_hint\_y=None, height=50)

        totals\_layout.add\_widget(self.net\_label)

        layout.add\_widget(totals\_layout) #Add the horizontal layout containing totals.

        # Scrollable List

        scroll\_view = ScrollView(size\_hint\_y=1, pos\_hint={'top': 1}) # Add pos\_hint to align top

        self.history\_list = GridLayout(cols=7, spacing=10, size\_hint\_y=None)  # Create GridLayout

        self.history\_list.bind(minimum\_height=self.history\_list.setter('height'))

        scroll\_view.add\_widget(self.history\_list)

        layout.add\_widget(scroll\_view)

        # Delete Button

        self.delete\_button = Button(text="Delete Selected", size\_hint\_y=None, height=50, background\_color=(1, 0, 0, 1))

        self.delete\_button.bind(on\_press=self.delete\_selected\_expense)

        layout.add\_widget(self.delete\_button)

        # Back Button

        back\_button = Button(text="Back", size\_hint\_y=None, height=50, background\_color=(0.2, 0.6, 1, 1))

        back\_button.bind(on\_press=self.go\_back)

        layout.add\_widget(back\_button)

        self.add\_widget(layout)

        self.selected\_expense\_id = None  # Store selected expense ID

    def show\_popup(self, title, message):

        content = BoxLayout(orientation='vertical')

        content.add\_widget(Label(text=message))

        close\_button = Button(text='Close', size\_hint\_y=None, height=40)

        content.add\_widget(close\_button)

        popup = Popup(title=title, content=content, size\_hint=(None, None), size=(400, 200))

        close\_button.bind(on\_press=popup.dismiss) # Bind close button to dismiss popup

        popup.open()

    def load\_history(self):

        try:

            self.history\_list.clear\_widgets()

            self.history\_list.cols = 7

            headers = ["Date", "Time", "Category", "Amount", "Description", "Expense Type", "Select"]

            for header in headers:

                label = Label(text=header, bold=True, size\_hint\_y=None, height=30, color=(0, 1, 1, 1))

                self.history\_list.add\_widget(label)

            conn = sqlite3.connect("expenses.db")

            cursor = conn.cursor()

            cursor.execute("SELECT id, date, time, category, amount, description, expense\_type FROM expenses ORDER BY date DESC, time DESC")

            records = cursor.fetchall()

            records\_by\_date = defaultdict(list)

            for row in records:

                date\_obj = datetime.datetime.strptime(row[1], "%Y-%m-%d").date()

                records\_by\_date[date\_obj].append(row)

            for date\_obj, daily\_records in records\_by\_date.items():

                date\_str = date\_obj.strftime("%Y-%m-%d")

                # Calculate empty label count for each side.

                empty\_count = (self.history\_list.cols - 1) // 2

                # Add empty labels for the left side

                for \_ in range(empty\_count):

                    self.history\_list.add\_widget(Label(text="", size\_hint\_y=None, height=30))

                # Add separator label

                separator\_label = Label(

                    text=f" ────────────────── {date\_str} ────────────────── ",

                    bold=True,

                    size\_hint\_y=None,

                    height=30,

                    color=(0.5, 0.5, 0.5, 1),

                    font\_size=22,

                )

                self.history\_list.add\_widget(separator\_label)

                # Add empty labels for the right side

                for \_ in range(self.history\_list.cols - 1 - empty\_count):

                    self.history\_list.add\_widget(Label(text="", size\_hint\_y=None, height=30))

                for row in daily\_records:

                    expense\_id, date, time, category, amount, description, expense\_type = row

                    color = (1, 0, 0, 1) if expense\_type == "Expense" else (0, 1, 0, 1)

                    labels = [date, time, category, f"₹{amount}", description, expense\_type]

                    for text in labels:

                        label = Label(text=text, size\_hint\_y=None, height=30, color=color)

                        self.history\_list.add\_widget(label)

                    select\_button = Button(text="Select", size\_hint\_y=None, height=30, background\_color=(0.3, 0.3, 0.3, 1))

                    select\_button.bind(on\_press=lambda instance, eid=expense\_id: self.select\_expense\_for\_delete(eid, instance))

                    self.history\_list.add\_widget(select\_button)

            # Calculate total expense, total income, and net

            cursor.execute("SELECT SUM(amount) FROM expenses WHERE expense\_type = 'Expense'")

            total\_expense = cursor.fetchone()[0] or 0.0

            cursor.execute("SELECT SUM(amount) FROM expenses WHERE expense\_type = 'Income'")

            total\_income = cursor.fetchone()[0] or 0.0

            net = total\_income - total\_expense

            # Update labels

            self.total\_expense\_label.text = f"Total Expense: ₹{total\_expense:.2f}"

            self.total\_income\_label.text = f"Total Income: ₹{total\_income:.2f}"

            self.net\_label.text = f"Net: ₹{net:.2f}"

            conn.close()

            if not records:

                no\_data\_label = Label(text="No history found.", font\_size=18, color=(1, 1, 1, 1), size\_hint\_y=None, height=30)

                self.history\_list.add\_widget(no\_data\_label)

                for \_ in range(6):

                    self.history\_list.add\_widget(Label(text=""))

            self.history\_list.height = self.history\_list.minimum\_height

            self.history\_list.parent.scroll\_y = 1

        except Exception as e:

            print(f"Error loading history: {e}")

    def select\_expense(self, expense\_id, instance, update\_mode=False):

        if hasattr(self, 'selected\_button') and self.selected\_button:

            self.selected\_button.background\_color = (0.3, 0.3, 0.3, 1)  # Reset previous selection

        self.selected\_expense\_id = expense\_id

        self.selected\_button = instance

        self.selected\_button.background\_color = (1, 0, 0, 1)  # Highlight selected

        if update\_mode:

            # Retrieve expense details and populate main screen

            cursor.execute("SELECT date, time, category, amount, description FROM expenses WHERE id=?", (expense\_id,))

            record = cursor.fetchone()

            if record:

                date, time, category, amount, description = record

                expense\_tracker = self.manager.get\_screen("main").children[0]  # Get main screen's ExpenseTracker instance

                # Split date and time for spinners

                year, month, day = date.split("-")

                hour\_minute, am\_pm = time.split()

                hour, minute = hour\_minute.split(":")

                # Populate fields in main screen

                expense\_tracker.year\_spinner.text = year

                expense\_tracker.month\_spinner.text = month

                expense\_tracker.day\_spinner.text = day

                expense\_tracker.hour\_spinner.text = hour

                expense\_tracker.minute\_spinner.text = minute

                expense\_tracker.ampm\_spinner.text = am\_pm

                expense\_tracker.category\_spinner.text = category

                expense\_tracker.amount\_input.text = str(amount)

                expense\_tracker.desc\_input.text = description

                # Store expense ID for update

                expense\_tracker.selected\_expense\_id = expense\_id

                # Switch to main screen

                self.manager.current = "main"

        # If update\_mode is False (delete or view), nothing happens after highlighting

    def delete\_selected\_expense(self, instance):

        if not self.selected\_expense\_id:

            print("No expense selected!")

            return

        try:

            cursor.execute("DELETE FROM expenses WHERE id=?", (self.selected\_expense\_id,))

            conn.commit()

            print("Expense deleted successfully!")

            self.show\_popup("Expense Deleted", "Expense deleted successfully!")

            self.selected\_expense\_id = None  # Reset selection

            self.load\_history()  # Refresh history after deletion

        except Exception as e:

            print("Error:", e)

    def set\_update\_mode(self, update\_mode):

        self.update\_mode = update\_mode

    def go\_back(self, instance):

        self.manager.current = "main"

    def load\_expense\_for\_editing(self):

        if not self.selected\_expense\_id:

            print("No expense selected!")

            return

        cursor.execute("SELECT date, time, category, amount, description FROM expenses WHERE id=?", (self.selected\_expense\_id,))

        record = cursor.fetchone()

        if record:

            date, time, category, amount, description = record

            main\_screen = self.manager.get\_screen("main").children[0]  # Get the ExpenseTracker instance

            # Fill the input fields

            main\_screen.year\_spinner.text, main\_screen.month\_spinner.text, main\_screen.day\_spinner.text = date.split("-")

            time\_parts = time.split(" ")

            if len(time\_parts) == 2:

                hour\_minute = time\_parts[0].split(":")

                if len(hour\_minute) == 2:

                    main\_screen.hour\_spinner.text = hour\_minute[0]

                    main\_screen.minute\_spinner.text = hour\_minute[1]

                    main\_screen.ampm\_spinner.text = time\_parts[1] # AM/PM

                else:

                    print(f"Invalid time format: {time}")

                    return

            elif len(time\_parts) == 1: #case where there is no AM/PM

                hour\_minute = time\_parts[0].split(":")

                if len(hour\_minute) == 2:

                    main\_screen.hour\_spinner.text = hour\_minute[0]

                    main\_screen.minute\_spinner.text = hour\_minute[1]

                    main\_screen.ampm\_spinner.text = "AM" #Default to AM if none is given.

                else:

                    print(f"Invalid time format: {time}")

                    return

            else:

                print(f"Invalid time format: {time}")

                return

            main\_screen.category\_spinner.text = category

            main\_screen.amount\_input.text = str(amount)

            main\_screen.desc\_input.text = description

            main\_screen.selected\_expense\_id = self.selected\_expense\_id  # Store selected ID

            # Go back to main screen for editing

            self.manager.current = "main"

    def select\_expense\_for\_update(self, expense\_id, instance):

        self.select\_expense(expense\_id, instance, update\_mode=True) #called when update button is pressed

    def select\_expense\_for\_delete(self, expense\_id, instance):

        self.select\_expense(expense\_id, instance) #called when history item is pressed for deletion

    def select\_expense\_for\_view(self, expense\_id, instance):

        self.select\_expense(expense\_id, instance) #called when history item is pressed for view

class ExpenseApp(App):

    def build(self):

        try:

            conn = sqlite3.connect("expenses.db")

            cursor = conn.cursor()

            # Create the expenses table (if it doesn't exist)

            cursor.execute('''CREATE TABLE IF NOT EXISTS expenses (

                                id INTEGER PRIMARY KEY AUTOINCREMENT,

                                date TEXT,

                                time TEXT,

                                category TEXT,

                                amount REAL,

                                description TEXT,

                                expense\_type TEXT)''')

            # Create the budgets table (if it doesn't exist)

            cursor.execute('''CREATE TABLE IF NOT EXISTS budgets (

                                id INTEGER PRIMARY KEY AUTOINCREMENT,

                                budget\_type TEXT,

                                category TEXT,

                                amount REAL)''')

            conn.commit()

            sm = ScreenManager()

            sm.add\_widget(Screen(name="main"))

            sm.add\_widget(HistoryScreen(name="history"))

            sm.add\_widget(BudgetsScreen(name="budgets"))

            sm.add\_widget(ViewBudgetsScreen(name="view\_budgets"))

            sm.add\_widget(ReportsScreen(name='reports'))

            sm.get\_screen("main").add\_widget(ExpenseTracker(screen\_manager=sm, cursor=cursor, conn=conn)) #added cursor and conn

            self.conn = conn  # Store the connection for later use

            return sm

        except sqlite3.Error as e:

            print(f"Database error during initialization: {e}")

            # Consider showing a popup or exiting the app if database setup fails

            return None  # Return None to prevent app from running without database

    def on\_stop(self):

        if hasattr(self, 'conn'):

            self.conn.close()

if \_\_name\_\_ == '\_\_main\_\_':

    ExpenseApp().run()