Python Project 08: Expense Tracker

1. Project Overview

An **Expense Tracker** helps users record and manage their daily expenses.

The application allows users to:

- Add expenses
- View expenses
- Delete expenses
- Filter expenses by category
- View total spending
- View spending by category
- Store expenses permanently

The project uses **Tkinter** for the desktop interface and Python's built-in **SQLite** database for persistent storage. SQLite is a lightweight disk-based database that does not require a separate database server.

---

2. Features

- Add expense
- Expense description
- Amount
- Category
- Date
- Expense table
- Delete selected expense
- Total expenses

- Category summary
- SQLite database
- Input validation
- Clean dark UI
- No external packages

---

3. Technologies

- Python 3
- Tkinter
- SQLite
- `sqlite3`
- `datetime`
- Object Oriented Programming

Python's `sqlite3` module provides a DB-API interface for working with SQLite databases directly from Python.

---

4. Folder Structure

```
expense_tracker/
□□□ main.py
□□□ ui.py
□□□ database.py
□□□ expenses.db
```

File Responsibilities

File	Responsibility
main.py	Starts the application
ui.py	Handles the graphical interface
database.py	Handles SQLite operations
expenses.db	Stores expense data

The expenses.db file is created automatically when the application runs.

---

5. How the Project Works

User enters expense
↓
Tkinter Frontend
↓
Expense validation
↓
Database Manager
↓
SQLite Database
↓
Expense saved
↓
Expense table updated

For example:

```
Description: Groceries
Amount: 2500
Category: Food
Date: 2026-09-27
```

is stored as an expense record in SQLite.

---

6. Complete Backend Code

database.py

```
import sqlite3


class ExpenseDatabase:

    def __init__(self, database_name="expenses.db"):
        self.database_name = database_name
        self.create_table()

    def connect(self):
        return sqlite3.connect(self.database_name)

    def create_table(self):
        connection = self.connect()

        cursor = connection.cursor()

        cursor.execute("""
            CREATE TABLE IF NOT EXISTS expenses (
```

```
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            description TEXT NOT NULL,
            amount REAL NOT NULL,
            category TEXT NOT NULL,
            date TEXT NOT NULL
        )
    """)

    connection.commit()
    connection.close()

def add_expense(
    self,
    description,
    amount,
    category,
    date
):
    connection = self.connect()

    cursor = connection.cursor()

    cursor.execute("""
        INSERT INTO expenses
        (description, amount, category, date)
        VALUES (?, ?, ?, ?)
    """, (
        description,
        amount,
        category,
        date
    ))

    connection.commit()
```

```
        connection.close()

    def get_expenses(self):
        connection = self.connect()

        cursor = connection.cursor()

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

        expenses = cursor.fetchall()

        connection.close()

        return expenses

    def delete_expense(self, expense_id):
        connection = self.connect()

        cursor = connection.cursor()

        cursor.execute("""
            DELETE FROM expenses
            WHERE id = ?
        """, (expense_id,))

        connection.commit()
        connection.close()

    def get_total(self):
        connection = self.connect()
```

```
        cursor = connection.cursor()

        cursor.execute("""
            SELECT COALESCE(SUM(amount), 0)
            FROM expenses
        """)

        total = cursor.fetchone()[0]

        connection.close()

        return total

    def get_category_totals(self):
        connection = self.connect()

        cursor = connection.cursor()

        cursor.execute("""
            SELECT category, SUM(amount)
            FROM expenses
            GROUP BY category
            ORDER BY SUM(amount) DESC
        """)

        totals = cursor.fetchall()

        connection.close()

        return totals
```

Backend Responsibilities

The database layer handles:

1. Creating the database table.
2. Adding expenses.
3. Reading expenses.
4. Deleting expenses.
5. Calculating total spending.
6. Calculating spending by category.

The SQL queries use parameter placeholders such as `?` instead of directly inserting user input into SQL strings. Python's SQLite documentation recommends parameter substitution for values rather than assembling SQL statements with string operations.

---

7. Complete Frontend Code

ui.py

```
import tkinter as tk
from tkinter import ttk, messagebox
from datetime import date

from database import ExpenseDatabase


class ExpenseTrackerUI:

    BG = "#0F172A"
    CARD = "#1E293B"
    INPUT = "#334155"
    TEXT = "#F8FAFC"
    MUTED = "#94A3B8"
    ACCENT = "#38BDF8"
    DANGER = "#EF4444"
    SUCCESS = "#22C55E"
```

```
    def __init__(self, root):

        self.root = root

        self.root.title("Expense Tracker")
        self.root.geometry("900x650")
        self.root.minsize(800, 600)
        self.root.configure(bg=self.BG)

        self.database = ExpenseDatabase()

        self.description_var = tk.StringVar()
        self.amount_var = tk.StringVar()
        self.category_var = tk.StringVar(value="Food")
        self.date_var = tk.StringVar(
            value=date.today().isoformat()
        )

        self.total_var = tk.StringVar(value="0.00")

        self.build_ui()
        self.load_expenses()
        self.update_summary()

    def build_ui(self):

        # Main container

        main = tk.Frame(
            self.root,
            bg=self.BG,
            padx=25,
            pady=20
        )
```

```
        main.pack(
            fill="both",
            expand=True
        )

        # Header

        tk.Label(
            main,
            text="Expense Tracker",
            font=("Segoe UI", 24, "bold"),
            bg=self.BG,
            fg=self.TEXT
        ).pack(anchor="w")

        tk.Label(
            main,
            text="Track your spending and manage your expenses",
            font=("Segoe UI", 10),
            bg=self.BG,
            fg=self.MUTED
        ).pack(anchor="w", pady=(0, 20))

        # Summary card

        summary = tk.Frame(
            main,
            bg=self.CARD,
            padx=20,
            pady=15
        )

        summary.pack(
```

```
            fill="x",
            pady=(0, 20)
        )

        tk.Label(
            summary,
            text="Total Spending",
            font=("Segoe UI", 10),
            bg=self.CARD,
            fg=self.MUTED
        ).pack(anchor="w")

        tk.Label(
            summary,
            textvariable=self.total_var,
            font=("Segoe UI", 22, "bold"),
            bg=self.CARD,
            fg=self.ACCENT
        ).pack(anchor="w")

        # Input card

        input_card = tk.Frame(
            main,
            bg=self.CARD,
            padx=15,
            pady=15
        )

        input_card.pack(
            fill="x",
            pady=(0, 20)
        )
```

```
        # Description

        tk.Label(
            input_card,
            text="Description",
            bg=self.CARD,
            fg=self.TEXT,
            font=("Segoe UI", 9, "bold")
        ).grid(
            row=0,
            column=0,
            sticky="w",
            padx=5
        )

        self.description_entry = tk.Entry(
            input_card,
            textvariable=self.description_var,
            bg=self.INPUT,
            fg=self.TEXT,
            insertbackground=self.TEXT,
            relief="flat"
        )

        self.description_entry.grid(
            row=1,
            column=0,
            sticky="ew",
            padx=5,
            pady=(5, 0),
            ipady=7
        )

        # Amount
```

```
        tk.Label(
            input_card,
            text="Amount",
            bg=self.CARD,
            fg=self.TEXT,
            font=("Segoe UI", 9, "bold")
        ).grid(
            row=0,
            column=1,
            sticky="w",
            padx=5
        )

        self.amount_entry = tk.Entry(
            input_card,
            textvariable=self.amount_var,
            bg=self.INPUT,
            fg=self.TEXT,
            insertbackground=self.TEXT,
            relief="flat"
        )

        self.amount_entry.grid(
            row=1,
            column=1,
            sticky="ew",
            padx=5,
            pady=(5, 0),
            ipady=7
        )

        # Category
```

```
        tk.Label(
            input_card,
            text="Category",
            bg=self.CARD,
            fg=self.TEXT,
            font=("Segoe UI", 9, "bold")
        ).grid(
            row=0,
            column=2,
            sticky="w",
            padx=5
        )

        categories = [
            "Food",
            "Transport",
            "Shopping",
            "Bills",
            "Health",
            "Education",
            "Entertainment",
            "Other"
        ]

        self.category_box = ttk.Combobox(
            input_card,
            textvariable=self.category_var,
            values=categories,
            state="readonly"
        )

        self.category_box.grid(
            row=1,
            column=2,
```

```
            sticky="ew",
            padx=5,
            pady=(5, 0),
            ipady=5
        )

        # Date

        tk.Label(
            input_card,
            text="Date",
            bg=self.CARD,
            fg=self.TEXT,
            font=("Segoe UI", 9, "bold")
        ).grid(
            row=0,
            column=3,
            sticky="w",
            padx=5
        )

        self.date_entry = tk.Entry(
            input_card,
            textvariable=self.date_var,
            bg=self.INPUT,
            fg=self.TEXT,
            insertbackground=self.TEXT,
            relief="flat"
        )

        self.date_entry.grid(
            row=1,
            column=3,
            sticky="ew",
```

```
            padx=5,
            pady=(5, 0),
            ipady=7
        )

        # Add button

        tk.Button(
            input_card,
            text="Add Expense",
            command=self.add_expense,
            bg=self.ACCENT,
            fg=self.BG,
            activebackground="#7DD3FC",
            activeforeground=self.BG,
            font=("Segoe UI", 10, "bold"),
            relief="flat",
            cursor="hand2"
        ).grid(
            row=1,
            column=4,
            padx=(10, 5),
            ipady=6
        )

        input_card.columnconfigure(0, weight=2)
        input_card.columnconfigure(1, weight=1)
        input_card.columnconfigure(2, weight=1)
        input_card.columnconfigure(3, weight=1)

        # Table

        table_frame = tk.Frame(
            main,
```

```
            bg=self.CARD
        )

        table_frame.pack(
            fill="both",
            expand=True
        )

        columns = (
            "id",
            "description",
            "amount",
            "category",
            "date"
        )

        self.tree = ttk.Treeview(
            table_frame,
            columns=columns,
            show="headings"
        )

        self.tree.heading(
            "id",
            text="ID"
        )

        self.tree.heading(
            "description",
            text="Description"
        )

        self.tree.heading(
            "amount",
```

```
            text="Amount"
        )

        self.tree.heading(
            "category",
            text="Category"
        )

        self.tree.heading(
            "date",
            text="Date"
        )

        self.tree.column(
            "id",
            width=50,
            anchor="center"
        )

        self.tree.column(
            "description",
            width=250
        )

        self.tree.column(
            "amount",
            width=120,
            anchor="center"
        )

        self.tree.column(
            "category",
            width=130,
            anchor="center"
```

```
        )

        self.tree.column(
            "date",
            width=120,
            anchor="center"
        )

        scrollbar = ttk.Scrollbar(
            table_frame,
            orient="vertical",
            command=self.tree.yview
        )

        self.tree.configure(
            yscrollcommand=scrollbar.set
        )

        self.tree.pack(
            side="left",
            fill="both",
            expand=True
        )

        scrollbar.pack(
            side="right",
            fill="y"
        )

        # Bottom buttons

        button_frame = tk.Frame(
            main,
            bg=self.BG
```

```
        )

        button_frame.pack(
            fill="x",
            pady=(15, 0)
        )

        tk.Button(
            button_frame,
            text="Delete Selected",
            command=self.delete_expense,
            bg=self.DANGER,
            fg=self.TEXT,
            activebackground="#F87171",
            activeforeground=self.TEXT,
            font=("Segoe UI", 10, "bold"),
            relief="flat",
            cursor="hand2"
        ).pack(
            side="left",
            ipadx=10,
            ipady=7
        )

        tk.Button(
            button_frame,
            text="Category Summary",
            command=self.show_category_summary,
            bg=self.INPUT,
            fg=self.TEXT,
            activebackground="#475569",
            activeforeground=self.TEXT,
            font=("Segoe UI", 10, "bold"),
            relief="flat",
```

```
            cursor="hand2"
        ).pack(
            side="right",
            ipadx=10,
            ipady=7
        )

    def add_expense(self):

        description = self.description_var.get().strip()
        amount_text = self.amount_var.get().strip()
        category = self.category_var.get().strip()
        expense_date = self.date_var.get().strip()

        if not description:
            messagebox.showerror(
                "Invalid Input",
                "Please enter an expense description."
            )
            return

        try:
            amount = float(amount_text)

            if amount <= 0:
                raise ValueError

        except ValueError:
            messagebox.showerror(
                "Invalid Amount",
                "Please enter a valid amount greater than 0."
            )
            return
```

```
        if not category:
            messagebox.showerror(
                "Invalid Category",
                "Please select a category."
            )
            return

        try:
            self.database.add_expense(
                description,
                amount,
                category,
                expense_date
            )

        except Exception as error:
            messagebox.showerror(
                "Database Error",
                str(error)
            )
            return

        self.clear_inputs()
        self.load_expenses()
        self.update_summary()

    def load_expenses(self):

        for item in self.tree.get_children():
            self.tree.delete(item)

        expenses = self.database.get_expenses()

        for expense in expenses:
```

```
            expense_id = expense[0]
            description = expense[1]
            amount = expense[2]
            category = expense[3]
            expense_date = expense[4]

            self.tree.insert(
                "",
                "end",
                values=(
                    expense_id,
                    description,
                    f"{amount:.2f}",
                    category,
                    expense_date
                )
            )

    def delete_expense(self):

        selected = self.tree.selection()

        if not selected:
            messagebox.showwarning(
                "No Selection",
                "Please select an expense to delete."
            )
            return

        item = self.tree.item(selected[0])

        expense_id = item["values"][0]
```

```
        confirm = messagebox.askyesno(
            "Delete Expense",
            "Are you sure you want to delete this expense?"
        )

        if not confirm:
            return

        self.database.delete_expense(
            expense_id
        )

        self.load_expenses()
        self.update_summary()

    def update_summary(self):

        total = self.database.get_total()

        self.total_var.set(
            f"{total:,.2f}"
        )

    def show_category_summary(self):

        totals = self.database.get_category_totals()

        if not totals:
            messagebox.showinfo(
                "Category Summary",
                "No expenses recorded yet."
            )
            return
```

```
        summary = "Spending by Category\n\n"

        for category, total in totals:

            summary += (
                f"{category}: {total:,.2f}\n"
            )

        messagebox.showinfo(
            "Category Summary",
            summary
        )

    def clear_inputs(self):

        self.description_var.set("")
        self.amount_var.set("")
        self.category_var.set("Food")
        self.date_var.set(
            date.today().isoformat()
        )

        self.description_entry.focus()
```

Tkinter provides the GUI widgets used here, while `messagebox` supplies modal information, warning, error, and confirmation dialogs.

---

8. Main Application Code

main.py

```
import tkinter as tk

from ui import ExpenseTrackerUI
```

```
def main():
    root = tk.Tk()
    ExpenseTrackerUI(root)
    root.mainloop()

if __name__ == "__main__":
    main()
```

---

9. How Frontend and Backend Connect

The frontend creates the database manager:

```
self.database = ExpenseDatabase()
```

When the user clicks **Add Expense**, the frontend sends the data to:

```
self.database.add_expense(
    description,
    amount,
    category,
    expense_date
)
```

The backend then executes the SQL `INSERT`.

The complete flow is:

Tkinter Form

User Input

Validation

ExpenseDatabase

SQLite

expenses.db

When expenses need to be displayed:

expenses.db

SQLite SELECT

ExpenseDatabase

load_expenses()

Treeview

---

10. How to Run

Open the terminal inside the project folder:

```
cd expense_tracker
```

Run:

```
python main.py
```

The application will automatically create:

```
expenses.db
```

on first run.

No external packages are required.

---

11. Using the Application

Add an expense

Enter:

```
Description: Groceries
Amount: 2500
Category: Food
Date: 2026-09-27
```

Click:

```
Add Expense
```

The expense will appear in the table.

Delete an expense

Select an expense from the table and click:

```
Delete Selected
```

View total spending

The total is automatically displayed at the top.

View category spending

Click:

```
Category Summary
```

Example:

```
Spending by Category

Food: 5000.00
Transport: 2500.00
Shopping: 3000.00
```

---

12. Basic Testing

Test 1: Add Expense

```
Description: Lunch
Amount: 500
Category: Food
```

Expected:

```
Lunch | 500.00 | Food
```

Test 2: Invalid Amount

Enter:

`Amount: abc`

Expected:

`Please enter a valid amount greater than 0.`

Test 3: Negative Amount

Enter:

`Amount: -500`

Expected:

`Please enter a valid amount greater than 0.`

Test 4: Delete Expense

Select an expense and click:

`Delete Selected`

Expected result:

The expense disappears from the table and the total is updated.

Test 5: Persistence

1. Add an expense.
2. Close the application.
3. Run it again.

Expected result:

The previous expense is still available because it was stored in SQLite.

---

13. Small Improvements

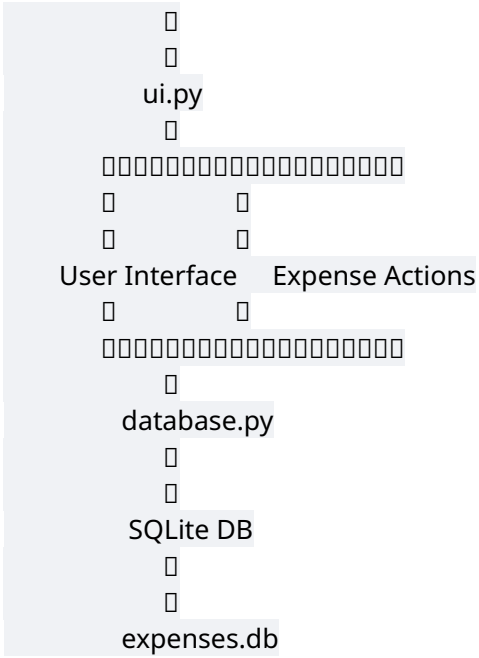
The project can later be extended with:

- Monthly expense reports
- Income tracking
- Budget limits
- Remaining balance
- Expense editing
- Date filtering
- Category filtering
- Search
- Pie charts
- Bar charts
- CSV export
- PDF reports
- Recurring expenses
- SQLite database backup
- Login and multiple users

---

14. Project Architecture

```
main.py
```



The project demonstrates:

User Input → Validation → Business Logic → Database → Result

This structure is useful because the UI and database logic remain separated, making the application easier to modify later.

Visit haas.dev for more resources and guides.