**Personal Finance Tracker Backend (FastAPI Project)**

Features:

1. Add Income or Expense

2. View All Transactions

3. Get Summary (income, expenses, balance, % breakdown)

4. Show Graph (Pie chart of category-wise expenses)

5. Search Transactions by Category

6. Filter Transactions by Date

7. Clear All Transactions (reset)

8. Delete Single Transaction by ID

9. Update a Transaction

10. View Income Only or Expense Only

11. Export and Import Transactions (File I/O)

12. Decorators for logging, validation, timing

"""

# ----------------------------

# Imports

# ----------------------------

from fastapi import FastAPI

from pydantic import BaseModel

import matplotlib.pyplot as plt

from datetime import datetime

import time

import json

# ----------------------------

# Decorators

# ----------------------------

def log\_action(func):

"""Decorator to log when a function starts and ends"""

def wrapper(\*args, \*\*kwargs):

print(f"[LOG] Function {func.\_\_name\_\_} started")

result = func(\*args, \*\*kwargs)

print(f"[LOG] Function {func.\_\_name\_\_} finished successfully")

return result

return wrapper

def validate\_amount(func):

"""Decorator to check if amount > 0"""

def wrapper(\*args, \*\*kwargs):

transaction = args[1] if len(args) > 1 else kwargs.get("transaction")

amount = transaction.amount

if amount <= 0:

raise ValueError("Amount must be greater than zero")

return func(\*args, \*\*kwargs)

return wrapper

def track\_time(func):

"""Decorator to measure function execution time"""

def wrapper(\*args, \*\*kwargs):

start = time.time()

result = func(\*args, \*\*kwargs)

end = time.time()

print(f"[TIME] {func.\_\_name\_\_} executed in {end - start:.4f} seconds")

return result

return wrapper

# ----------------------------

# Pydantic Model for Transactions

# ----------------------------

class Transaction(BaseModel):

t\_type: str # "income" or "expense"

amount: float # must be > 0

category: str # e.g., Food, Rent, Salary

description: str # user notes

date: str = datetime.now().strftime("%Y-%m-%d")

# ----------------------------

# Main Finance Tracker Class

# ----------------------------

class FinanceTracker:

def \_init\_(self):

self.transactions = [] # list to hold all income/expense records

self.counter = 1 # unique transaction ID

@log\_action

@validate\_amount

def add\_transaction(self, transaction: Transaction):

"""Add income or expense"""

record = transaction.dict()

record["id"] = self.counter

self.transactions.append(record)

self.counter += 1

@log\_action

def get\_transactions(self):

"""Return all transactions"""

return self.transactions

@log\_action

def search\_by\_category(self, category: str):

"""Search all transactions by category"""

return [t for t in self.transactions if t["category"].lower() == category.lower()]

@log\_action

def filter\_by\_date(self, date: str):

"""Filter transactions by date (YYYY-MM-DD)"""

return [t for t in self.transactions if t["date"] == date]

@log\_action

def clear\_transactions(self):

"""Remove all transactions"""

self.transactions.clear()

self.counter = 1

return {"message": "All transactions cleared"}

@log\_action

def delete\_transaction\_by\_id(self, t\_id: int):

"""Delete a single transaction by its ID"""

for t in self.transactions:

if t["id"] == t\_id:

self.transactions.remove(t)

return {"message": f"Transaction {t\_id} deleted"}

return {"error": "Transaction not found"}

@log\_action

def update\_transaction(self, t\_id: int, new\_data: Transaction):

"""Update an existing transaction"""

for t in self.transactions:

if t["id"] == t\_id:

t.update(new\_data.dict())

return {"message": f"Transaction {t\_id} updated"}

return {"error": "Transaction not found"}

@log\_action

def get\_income\_only(self):

"""Return only income transactions"""

return [t for t in self.transactions if t["t\_type"].lower() == "income"]

@log\_action

def get\_expense\_only(self):

"""Return only expense transactions"""

return [t for t in self.transactions if t["t\_type"].lower() == "expense"]

@log\_action

def export\_to\_file(self, filename="transactions.json"):

"""Export transactions to JSON file"""

with open(filename, "w") as f:

json.dump(self.transactions, f, indent=4)

return {"message": f"Data exported to {filename}"}

@log\_action

def import\_from\_file(self, filename="transactions.json"):

"""Import transactions from JSON file"""

try:

with open(filename, "r") as f:

self.transactions = json.load(f)

if self.transactions:

self.counter = max(t["id"] for t in self.transactions) + 1

return {"message": f"Data imported from {filename}"}

except FileNotFoundError:

return {"error": f"File {filename} not found"}

@log\_action

@track\_time

def get\_summary(self):

"""Generate summary of income, expenses, balance, and category breakdown"""

total\_income = sum(t["amount"] for t in self.transactions if t["t\_type"].lower() == "income")

total\_expense = sum(t["amount"] for t in self.transactions if t["t\_type"].lower() == "expense")

balance = total\_income - total\_expense

# Category-wise expense breakdown

category\_expenses = {}

for t in self.transactions:

if t["t\_type"].lower() == "expense":

category\_expenses[t["category"]] = category\_expenses.get(t["category"], 0) + t["amount"]

# Convert to percentages

category\_percentages = {

cat: f"{(amt / total\_expense) \* 100:.2f}%"

for cat, amt in category\_expenses.items()

} if total\_expense > 0 else {}

return {

"Total Income": total\_income,

"Total Expense": total\_expense,

"Balance Left": balance,

"Expense Breakdown": category\_percentages,

}

@log\_action

@track\_time

def plot\_expenses(self):

"""Display pie chart of expenses by category"""

expenses = [t for t in self.transactions if t["t\_type"].lower() == "expense"]

if not expenses:

return "No expenses to plot"

categories = [t["category"] for t in expenses]

amounts = [t["amount"] for t in expenses]

plt.figure(figsize=(6, 6))

plt.pie(amounts, labels=categories, autopct="%1.1f%%", startangle=140)

plt.title("Expense Breakdown by Category")

plt.show()

return "Graph displayed successfully"

# ----------------------------

# FastAPI Setup

# ----------------------------

app = FastAPI()

tracker = FinanceTracker()

# ----------------------------

# API Endpoints

# ----------------------------

@app.post("/add")

def add\_transaction(transaction: Transaction):

tracker.add\_transaction(transaction)

return {"message": "Transaction added successfully"}

@app.get("/transactions")

def get\_transactions():

return tracker.get\_transactions()

@app.get("/summary")

def get\_summary():

return tracker.get\_summary()

@app.get("/plot")

def plot\_graph():

return {"message": tracker.plot\_expenses()}

@app.get("/search/{category}")

def search\_category(category: str):

return tracker.search\_by\_category(category)

@app.get("/filter/{date}")

def filter\_date(date: str):

return tracker.filter\_by\_date(date)

@app.delete("/clear")

def clear\_data():

return tracker.clear\_transactions()

@app.delete("/delete/{t\_id}")

def delete\_transaction(t\_id: int):

return tracker.delete\_transaction\_by\_id(t\_id)

@app.put("/update/{t\_id}")

def update\_transaction(t\_id: int, transaction: Transaction):

return tracker.update\_transaction(t\_id, transaction)

@app.get("/income")

def get\_income():

return tracker.get\_income\_only()

@app.get("/expenses")

def get\_expenses():

return tracker.get\_expense\_only()

@app.get("/export")

def export\_data():

return tracker.export\_to\_file()

@app.get("/import")

def import\_data():

return tracker.import\_from\_file()

@app.get("/")

def home():

return {"message": "Welcome to Personal Finance Tracker API"}