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
```

```python
# ----------------------------
# 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"}
```