PET - Personal Expense Tracker

Source code

constants.py

```
from enum import Enum, auto

# File Paths
EXPENSE_FILE_NAME = "data/expenses.txt"
BUDGET_FILE_NAME = "data/budget.txt"
LOG_FILE_NAME = "logs/logs.txt"
PRINT_DATE_WIDTH = 10
PRINT_CATEGORY_WIDTH = 25
PRINT_AMOUNT_WIDTH = 15

# UI Settings
SCREEN_WIDTH = 100

# Enum for log types
class LogType(Enum):
    WARNING = auto()
    ERROR = auto()
    INFORMATION = auto()

# Enum for Menu choices
class MenuChoice(Enum):
    ADD_EXPENSE = "1"
    VIEW_EXPENSE = "2"
    TRACK_BUDGET = "3"
    SAVE_EXPENSE = "4"
    EXIT = "5"
```

utils.py

```
from constants import SCREEN_WIDTH

def print_border():
    """Prints a border line."""
    print("=" * SCREEN_WIDTH + "\n")

def is_valid_date(date_str: str) -> bool:
    """Checks if the input date follows the YYYY-MM-DD format."""
    from datetime import datetime
    try:
        datetime.strptime(date_str, "%Y-%m-%d")
        return True
    except ValueError:
        return False

def is_valid_amount(amount_str: str) -> bool:
    """Validates if the input amount is a positive float."""
    try:
        return float(amount_str) > 0
    except ValueError:
        return False
```

logger.py

```
from datetime import datetime
from constants import LOG_FILE_NAME, LogType

def log_data(message: str, log_type: LogType):
    """Writes a log message to the log file."""
    try:
        with open(LOG_FILE_NAME, "a") as file:
            file.write(f"{datetime.now()}: {log_type.name}: {message}\n")
    except Exception as e:
        print(f"Error writing to log file: {e}")
```

expense_manager.py

```
import os
from logger import log_data
from constants import LogType, PRINT_DATE_WIDTH, PRINT_AMOUNT_WIDTH,
PRINT_CATEGORY_WIDTH
from utils import print_border, is_valid_date, is_valid_amount

def load_expenses(file_name: str):
    """Loads expenses from a file and calculates the total amount."""
    expenses = []
    total = 0.0

    if not os.path.exists(file_name):
        log_data(f"File not found - {file_name}", LogType.WARNING)
        return expenses, total

    with open(file_name, "r") as file:
        for line_number, line in enumerate(file, start=1):
            data = [x.strip() for x in line.strip().split(",")]

            if len(data) == 4 and is_valid_date(data[0]) and data[1] and
is_valid_amount(data[2]) and data[3]:
                expenses.append({"Date": data[0], "Category": data[1], "Amount":
data[2], "Description": data[3]})
                total += float(data[2])
            else:
                log_data(f"Invalid data at line {line_number} in {file_name}:
{line.strip()}", LogType.ERROR)

    return expenses, total

def save_expenses(expenses: list, file_name: str):
    """Saves the expense list to a file."""
    try:
        with open(file_name, "w") as file:
            for expense in expenses:
                file.write(get_expense_string(expense))
                #file.write(f"{expense['Date'].ljust(PRINT_DATE_WIDTH)},
{expense['Category'].ljust(PRINT_CATEGORY_WIDTH)},
{expense['Amount'].ljust(PRINT_AMOUNT_WIDTH)},
{expense['Description'].ljust(PRINT_DESCRIPTION_WIDTH)}\n")
    except Exception as e:
        log_data(f"Error saving expenses: {e}", LogType.ERROR)

def add_expense() -> dict:
    """Prompts the user to input expense details and returns an expense
dictionary."""
    print_border()
    print("PET - Add Expense".center(100))
    print_border()
```

```
    while True:
        date = input("Enter date (YYYY-MM-DD): ").strip()
        if is_valid_date(date):
            break
        print("Invalid date format. Please try again.")

    while True:
        category = input(f"Enter category (e.g., Food, Travel - max
{PRINT_CATEGORY_WIDTH} characters) : ").strip()[:PRINT_CATEGORY_WIDTH]
        if category:
            break
        print("Category cannot be empty. Please try again.")

    while True:
        amount = input("Enter amount: ").strip()
        if is_valid_amount(amount):
            break
        print("Invalid amount. Please try again.")

    while True:
        description = input("Enter description: ").strip()
        if description:
            break
        print("Description cannot be empty. Please try again.")

    return {
        "Date": date,
        "Category": category,
        "Amount": amount,
        "Description": description
    }

def view_expenses(expenses: list):
    """Displays all stored expenses."""
    print_border()
    print("DATE".ljust(PRINT_DATE_WIDTH) + "," +
"CATEGORY".ljust(PRINT_CATEGORY_WIDTH) + "," + "AMOUNT".ljust(PRINT_AMOUNT_WIDTH)
+ "," + "DESCRIPTION")
    print_border()

    for expense in expenses:
        print(get_expense_string(expense).strip())

    print_border()
    print(f"Total Expenses: {len(expenses)} items")
    print_border()

def get_expense_string(expense: dict) -> str:
    """Formats an expense dictionary into a string for file storage or display."""
    return f"{expense['Date'].ljust(PRINT_DATE_WIDTH)},
{expense['Category'].ljust(PRINT_CATEGORY_WIDTH)},
{expense['Amount'].ljust(PRINT_AMOUNT_WIDTH)},{expense['Description']}\n"
```

budget_manager.py

```
import os
from logger import log_data
from constants import LogType
from utils import is_valid_amount

def track_budget(budget, total_expenses, file_name):
    print(f"\nCurrent budget amount is : {budget:.2f}")
    while True:
        budget_str = input("Enter new budget: ") or budget
        if is_valid_amount(budget_str):
            break
        print("\nInvalid Budget amount. Please try again")

    budget = float(budget_str)
    print(f"\nNew budget is : {budget:.2f}")
    compare_budget(budget, total_expenses)
    save_budget(budget, file_name)

def load_budget(file_name: str) -> float:
    """Loads the budget from a file."""
    budget = 0
    if not os.path.exists(file_name):
        log_data(f"Budget File not found - {file_name}", LogType.WARNING)
        return budget

    try:
        with open(file_name, "r") as file:
            budget = float(file.read().strip())
    except ValueError:
        log_data("Invalid budget data", LogType.ERROR)

    return budget

def save_budget(budget: float, file_name: str):
    """Saves the budget to a file."""
    try:
        with open(file_name, "w") as file:
            file.write(f"{budget:.2f}")
        log_data("Budget saved successfully", LogType.INFORMATION)
    except Exception as e:
        log_data(f"Error saving budget: {e}", LogType.ERROR)

def compare_budget(budget: float, total_expense: float):
    """Compares expenses with the budget."""
    difference = budget - total_expense
    if difference < 0:
        print(f"\nYou have exceeded your budget by {-difference:.2f}")
    else:
        print(f"\nYou have {difference:.2f} left in your budget")
```

PET.py

```
from constants import EXPENSE_FILE_NAME, BUDGET_FILE_NAME, SCREEN_WIDTH,
MenuChoice
from utils import print_border
from expense_manager import load_expenses, save_expenses, add_expense,
view_expenses
from budget_manager import load_budget, track_budget

def print_header(total_expenses, budget):
    """Prints the application header with budget details."""
    print_border()
    print("PET - Personal Expense Tracker".center(SCREEN_WIDTH) + "\n")
    print(f"Budget: {budget:.2f} | Total Expenses: {total_expenses:.2f} |
{"Remaining" if budget > total_expenses else "Overspend"}: {abs((budget -
total_expenses)):.2f}".center(SCREEN_WIDTH))
    print_border()

def get_menu_choice():
    """Displays menu and returns user choice."""
    print("\t 1 - Add Expense\n")
    print("\t 2 - View Expenses\n")
    print("\t 3 - Track Budget\n")
    print("\t 4 - Save Expenses\n")
    print("\t 5 - Exit\n")
    print_border()
    return input("Enter your choice: ").strip()

def main():
    """Main function to handle user interactions."""
    expenses, total_expenses = load_expenses(EXPENSE_FILE_NAME)
    budget = load_budget(BUDGET_FILE_NAME)
    expenses_to_save = False

    while True:
        print_header(total_expenses, budget)
        choice = get_menu_choice()

        if choice == MenuChoice.ADD_EXPENSE.value:
            expense = add_expense()
            expenses.append(expense)
            total_expenses += float(expense["Amount"])
            expenses_to_save = True
            print(f"\nExpense added successfully. {expense}")
            if total_expenses > budget:
                print(f"\nYou have overspent your budget by {(total_expenses -
budget):.2f}")
            input("Press Enter to continue.....")

        elif choice == MenuChoice.VIEW_EXPENSE.value:
            view_expenses(expenses)
            input("Press Enter to continue.....")
```

```
        elif choice == MenuChoice.TRACK_BUDGET.value:
            # print(f"Current budget amount is : {budget}")
            # budget = float(input("Enter new budget: ") or budget)
            # compare_budget(budget, total_expenses)
            # save_budget(budget, BUDGET_FILE_NAME)
            track_budget(budget, total_expenses, BUDGET_FILE_NAME)
            input("Press Enter to continue.....")

        elif choice == MenuChoice.SAVE_EXPENSE.value:
            save_expenses(expenses, EXPENSE_FILE_NAME)
            expenses_to_save = False
            print("Expenses saved.")
            input("Press Enter to continue.....")

        elif choice == MenuChoice.EXIT.value:
            if expenses_to_save:
                save_expenses(expenses, EXPENSE_FILE_NAME)
            print("Thank you for using PET!!!")
            break

        else:
            print("\nInvalid choice. Please try again.\n")

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