Design a Python program that tracks user login and logout activity.
The program must include a main class, a logging system, and Git/GitHub version control.

Part 1 — User Class

Create a User class with the following attributes:

- username
- email
- is_logged_in (current login status: True/False)

And the following methods:

- login()
- logout()

Part 2 — Logging System

Create a central logger that includes:

- A StreamHandler with level WARNING
- A FileHandler with level INFO that writes logs into activity.log

The system must log:

- User login → level INFO
- User logout → level INFO
- Attempting to log in while already logged in → WARNING
- Attempting to log out while not logged in → WARNING
- Invalid menu input → WARNING

Part 3 — CLI Menu

Implement the following simple menu:

1. Log in

2. Log out

3. Show current status (Logged In / Logged Out)

4. Exit program

Part 4 — Git and GitHub Requirements

The student must:

1. Initialize a Git repository for the project
2. Make at least four meaningful commits
3. Create a .gitignore file that excludes log files
4. Create a GitHub repository and push the project to it
5. Add their teammates as collaborators on GitHub

Qustion2:

Design a Python program that manages financial accounts and their transactions.
The system must use OOP, a logging system with four handlers, and a **generic DBManager** that stores all data across **multiple JSON files** (e.g., accounts.json, transactions.json).
All operations must read/write directly from these JSON files.

Part 1 — Account Class

Create an Account class with:

- owner
- balance
- transactions (a list of transaction IDs or objects)

Methods:

- deposit(amount)
- withdraw(amount)
- transfer(target_account, amount)

Important Requirements:

- When a new account is created, it **must be immediately saved** into the JSON file (e.g., accounts.json) using the DBManager.
- During **deposit**, **withdraw**, and **transfer**:
    - The account data must be read from the JSON file
    - The balance must be updated
    - The modified data must be saved back to the file
- For **displaying balance** or **showing transactions**, data must always be read directly from the JSON file, not from memory.

Part 2 — Transaction Class

Create a Transaction class with:

- type (DEPOSIT / WITHDRAW / TRANSFER)
- amount
- timestamp

And a method to convert the transaction into a nicely formatted string for printing.

Part 3 — Logging System (4 Required Handlers)

Create a central logger with the following handlers:

Handler 1 — StreamHandler (level WARNING)

Format:

[WARNING] %(message)s

Handler 2 — FileHandler for Transactions (level INFO)

File: transactions.log
Format:

%(asctime)s | TRANSACTION | %(levelname)s | %(message)s

Handler 3 — FileHandler for Errors (level ERROR)

File: errors.log
Format:

ERROR at line %(lineno)d: %(message)s

Handler 4 — FileHandler for Debug Logs (level DEBUG)

File: debug.log
Format:

%(asctime)s - %(filename)s:%(lineno)d - %(levelname)s - %(message)s

---

Part 4 — Logging Rules

INFO

- Successful deposit
- Successful withdrawal
- Successful transfer
- Transaction creation

WARNING

- Insufficient balance
- Invalid user input
- Trying to access a non-existing account

ERROR

- Negative amount
- Corrupted JSON file
- File read/write/delete failures

DEBUG

- Account initialization
- Transaction initialization
- Execution of major methods

Part 5 — DBManager (Generic, Multi-File JSON Storage)

Create a DBManager class .

It must interact only with JSON files and support exactly **four core operations**:

---

1. list(file_name)

- Return all data from the JSON file.
- If the file does not exist:
    - create a new empty JSON file ({} or [])
    - log a WARNING.

---

2. read(file_name, key)

- Return the value associated with key.
- If the file does not exist → create it and log a WARNING.
- If the key does not exist → log a WARNING.
- If the JSON content is corrupted → log an ERROR and return None.

---

3. write(file_name, key, data)

- Save data under the given key.
- If the file does not exist → create it.
- If writing fails → log an ERROR.

---

4. delete(file_name, key)

- Remove the entry associated with key.
- If the file does not exist → create it and log a WARNING.
- If the key does not exist → log a WARNING.
- If deletion fails → log an ERROR.

Example JSON files:

accounts.json

transactions.json

All account edits, balance updates, and transaction logs **must be done using DBManager**.

Part 6 — Two-Level CLI Menu

Your program must implement two interactive menus:

🔹 **Level 1 Menu (Account Management)**

This menu appears when the program starts:

1. Create new account

2. Select account

3. Delete account

4. List all accounts

5. Exit

- "Create account" → immediately save the account into accounts.json
- "Delete account" → remove it from accounts.json using DBManager.delete
- "List all accounts" → read from accounts.json using DBManager.list

---

🔹 **Level 2 Menu (Operations on the Selected Account)**

Once a user selects an account, show this second menu:

1. Deposit into this account

2. Withdraw from this account

3. Transfer from this account to another account

4. Show balance

5. Show transaction history

6. Back to main menu

All operations must:

- Read account data from the JSON file
- Update it
- Save it back immediately
  — using DBManager

Part 7 — Git / GitHub Requirements

1. Initialize a Git repository for the project
2. Make at least **7 meaningful commits** with clear commit messages
3. Create a .gitignore file that excludes all log files
4. Create a GitHub repository and push the entire project to it
5. **Add your teammates as collaborators to the GitHub repository**