**Collaborative Sheets**

1. **How do you use ChatGPT or other LLM-based tools in developing, modifying, and debugging the program.**

* Not used LLM-based tool to construct the source code, but in the debugging state, we present the full source code to LLM to scan the potential issue quickly. Take for example, when we finished the program. We tested the commands that were proposed in the system. And We found that the command of **“Collaborate and uncollaborated with another user”.** When a user shares the sheet’s permission to another user, it works. But when the collaborate user wants to edit the sheet’s content, the system didn’t find the sheet. Through the LLM, We found the flaws quickly. The detail of this error that described in **Section G**.

1. **Overview of source code**
   1. **How does it work:** First of all, the systemprovides the following commands for users to execute the belonging functions of program.

|  |
| --- |
|  |

* + 1. **Create a user:** Input a name and create the user.
       - Mark
* Create a user named “Mark”.
  + 1. **Create a sheet:** Create a sheet under the user.
       - Mark SheetA
* Create a sheet named “SheetA” for “Mark”.
  + 1. **Check a sheet:** Check the present content of sheet.
       - Mark SheetA
* 0, 0, 0,

0, 0, 0,

0, 0, 0,

* + 1. **Change the value in a sheet:** Change the specified sheet value of specified row or column.
       - Mark SheetA
       - 1 2 3 (Format: row col value)
* 0, 3, 0,

0, 0, 0,

0, 0, 0,

* + 1. **Change a sheet’s access right:** Constrained by the program designed, only the owner can edit the access of the sheet.
       - Mark SheetA E
* Changed access rights of SheetA to editable.
* Kevin SheetA R
* Only the owner can edit the sheet’s permission. (Mark is the owner.)
  + 1. **Collaborate or uncollaborated with another user:** Shares or unshare the sheet’s permission (editable) to a designated user.
       - Mark SheetA Kevin Share
* Mark shared the sheet ‘SheetA’ with Kevin.
* Mark SheetA Kevin Unshare
* Access to sheet “SheetA” has been revoked from Kevin.
  + 1. **List all sheets with access rights:**  List all the sheets’ permissions under the user. With this command, you can easily see if the user is in the system or not and check the access rights to the sheets of the user.
       - Mark
* User: Mark

Sheets (Editable): SheetA

Sheets (ReadOnly): None

And in the program, there is a global manager called **Manager,** which is responsible for creating and managing the user and indirectly managing the sheet through the user. Each **User** has their own sheets list and can create their own sheets. It also has a **Sheet** class to encapsulate the sheet’s name, content, access rights, and permissions.

Manager uses the dictionary structure to manage multiple User instances. And there are some methods, like creating and getting the user, that are described in detail in the next sub-section.

Each User instance manages its sheet sets through a dictionary structure and provides an interface for searching sheets by name. In the User class, there are no direct methods to manipulate the sheet. The relevant methods are encapsulated in Sheet class.

* 1. **Data Structures:**  In the program, which defined three main classes: User, Sheet, and Manager. Here are the following definitions:
     1. **User:** Define how each user creates their own sheet in the system.
        + **Attributes:** 
          - username: Define the unique identifier of the user.
          - sheets: Utilizing dictionary structure to store the user’s sheets.
        + **Methods:** 
          - create\_sheet: Corresponds to the command 2, **“Create a sheet.”** To create a sheet that belongs to the user and add it to the set of sheets.
     2. **Sheet:** Define the sheet properties and the method of checking sheets, modifying contents, and permissions.
        + **Attributes:**
          - name: Define sheet’s name.
          - owner: Define who created the sheet.
          - access\_right: Define the sheet permission.
          - content:  Create a two-dimensional list to store the value of the sheet.
          - permissions: Store the user edit or read permissions to the sheet.
        + **Methods:** 
          - check\_sheet: Corresponds to the command 3, **“Check a sheet.”** Return the contents of the current state of the sheet. If not found, return the string of “Sheet or user not found.”
          - update\_cell: Corresponds to the command 4, “**Change the value in a sheet.**” Update the value ​​based on specified rows and columns. There’s also a access right check, if the current state is ReadOnly, then it cannot be edited.
          - change\_access\_right: Corresponds to the command 5, **“Change a sheet’s access right.”** Define the current sheet’s access right. And it also has the identity check; if the user is not the sheet’s owner, then he or she cannot change the sheet’s rights.
          - share\_sheet: Corresponds to the command 6, **“Collaborate or uncollaborated with another user.”** Share the permissions with another user.
          - unshare\_sheet: Corresponds to the command 6, “**Collaborate or uncollaborated with another user.**” Unshared the permissions with designated user.
     3. **Manager:** Define the global manager of the system, which is responsible for creating the user and the overall management of the system.
        + **Attributes:**
          - users: Utilize a dictionary structure to store the user and the corresponding User class instances.
        + **Methods:** 
          - create\_user: Create the new user and add it in the users list.
          - get\_user:Based on the username, search the corresponding User instance.
          - get\_sheet:Based on the username and sheet name, search the corresponding sheet’s content.
          - check\_user:Verify whether the user already exists or not.
  2. **Specify how you switch on/off some functionalities:**

There are some functionalities switching in the program. Like the command 6 **“Collaborate or uncollaborated with another user.”** For **static approach,** if you want to cancel the share permission functions, you can delete or comment out this conditional judgment.

As to **dynamic ways**, methods like change\_access\_right and share\_sheet in the Sheet class. When the system executes the **Sheet.change\_access\_right**. First, it will check whether the user is the sheet’s owner or not. If he is, the user can change the sheet’s right. When the sheet’s state is **ReadOnly**, then it cannot be edited.  On the other side, if the sheet’s state is set to **Editable**, then it can be edited, as depicted in Figure 1.

There’s another instance like execute **Sheet.share\_sheet.** The user can modify the collaborator’s sheet’s permission in the runtime through setting the operations **(Share,** **Unshare**), as depicted in Figure 2.

|  |  |
| --- | --- |
|  |  |
| Figure 1. | Figure 2. |

1. **Programming paradigm.**
   1. **Features:** In this program, we used Object-Oriented Programming (OOP) to build up the system. Here are the three main features in OOP: **encapsulation**, **inheritance**, and **polymorphism**. However, the system construct solely implements encapsulation in this version.

* **Encapsulation**: An object encapsulates the details, like hiding variables and methods from others. Restrict direct access to internal details.
  1. **Apply in the source code:** 
     + **Encapsulation:** In the program, the User and Sheet class encapsulates their own attributes and methods, which are introduced in the upon section.

|  |
| --- |
|  |
|  |

* 1. **Functional Programming in code in detail:** This program has three classes and their own methods to build up the whole system operating behaviors.
     + **User:** There’s a method called “create\_sheet”, which defines how each user instance can create their own sheet and store the sheets.

|  |
| --- |
|  |

* + - **Sheets: There’re five methods to execute the belonging behavior.** We just proposed some of them to demonstrate the core concept of functional programming. Like pure function and immutability.
      1. **check\_sheet**: Using the “**pure function**” concept to construct the function, which function is let the user examine the current sheet’s content.

|  |
| --- |
|  |

* + - 1. **update\_cell:** There’s another important concept in functional programming: “**Immutability”** , which means when an object is created, it cannot be modified directly. It must return a new object to update the data. But in our program, we just adopt ReadOnly to restrict the user from revising the content, not actually follow the concept to construct the new object.

|  |
| --- |
|  |

If you really want to follow the concept, maybe you can change the code to the following picture. But in the project, I think it would not be multiple users modifying the sheet content in the meantime. For the convenience of time and space-saving, we are not following the concept totally.

|  |
| --- |
|  |

* + - **Manager:** In this class, there’s also some methods to show the concept thefunctional programming. Like get\_user function. Only return the data but not modify the contents.

|  |
| --- |
|  |

There’s another core concept of “Function composition” in functional programming used in the method “get\_sheet”. As we method before, the User class has two attributes, that is, username and its own sheet list. So, when we execute the command 3 “Check the sheet”. The program would call the get\_sheet to get the sheet’s content. Following the code execution, we could find that the function will execute another function, “get\_user,” to find the sheet name under the designated user.

|  |
| --- |
|  |

1. **Programming paradigm that was not used in this code.**
   1. **Features:** Like we mentioned in the previous section, we adopt the OOP to construct this project. But there are two core concepts that we’re not actually using in the program, as listed below.
      * **Inheritance**: Subclass can reuse the code in the superclass and add its own unique methods.
      * **Polymorphism**: The objects of subclasses can behavior differently, according to the definition of their subclasses.
   2. **What is the difference between when choose it to the code.**
      * **Inheritance:** In this project, we clearly create User and Sheet classes and define sufficient attributes and methods to satisfy system operations. And the goal achieved by each user and sheet instance is consistent and simple. There’s no other special users or sheets that need to be inherited by super class. However, the extension may have more than one user, such as root and regular users. They can inherit from User and add their own new attributes or methods.As depicted in the below code, only root can delete a sheet, and a regular user cannot create a new sheet.

|  |
| --- |
|  |

* + - **Polymorphism:** As mentioned above, we think there’s not many different types of users or sheets that need to be created, in this case, it lacks scenarios where multiple subclasses need to override parent methods. If adapted, this concept would look like the program below.

|  |
| --- |
|  |

1. **Idioms or Design Patterns use in the code**
   1. **Features:**
   2. **Use it in the code**
2. **Idioms or Design Patterns that was not used in the code**
   1. **Features:**
      * **Observer pattern:** It’s a Behavioral design pattern. It allows users to create a notification system to update objects (observers) when any event occurs to the object (the subject) being observed.
   2. **Difference the use in the code:**
      * **Observer pattern:** For instance**,** in this version of system, there’s no notification when the behavior like share or unshared the sheet’s permission to another user. The following code is added to this design pattern.
        + **User:** In the User class, adding the notification operation.

|  |
| --- |
|  |

* + - * **Sheet:** In the Sheet class, adding the relative operation of store observers. And when the system executes the command 6 “**Collaborate or uncollaborated with another user.**” The system would notify all the users in the system, as shown in the output below.

|  |
| --- |
|  |
|  |

1. **Errors encountered during programming**
   1. **What happened**
      1. **The input and condition that triggered the errors:** 
         * Like we mentioned in part A, in the very beginning. When the system executed the command 6 “Collaborated or uncollaborated with another user.” It should share the sheet’s edit right with the designated user. And the terminal also shows the Sheet has been shared to the collaborator correctly. However, when the collaborator wants to edit the sheet, the system shows the collaborator doesn't have permission.
   2. **How to use Debugger to solve it?**

* In this case, we have two ways to debug the program. First, we can set the break point to see the result. Check whether the collaborator has been added to the sheet’s permission when the system executes command 6. Second, through the LLM-based tools, like ChatGPT. It can find the issue quickly and supply the better solutions. With its help, we found that when we executed command 6, in the method of share\_sheet, we were missing a line to add the collaborator in the sheet’s permission. The detail prompt with ChatGPT is at the following URL:
* <https://chatgpt.com/share/67758f79-3c64-8004-9f3b-e712ec701630>