Department of Computing
電子計算學系

THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

COMP3211 Software Engineering

Group 13

Personal Information Manager

Design Document

JIANG Guanlin (21093962d)

MENG Guanlin (20099185d)

HU Yuhang (21106395d)

YE Feng (21098249d)

1 The PIM's Architectural Diagram

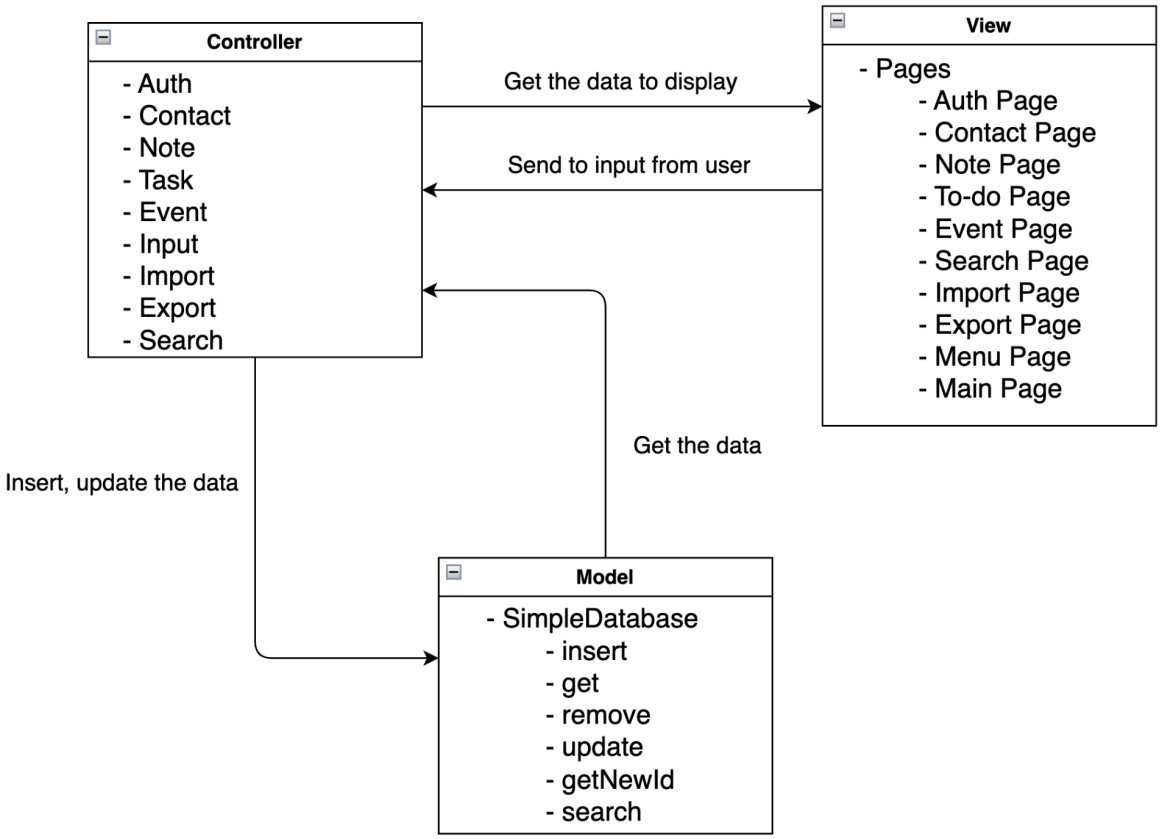

Figure 1: MVC Architectural Patterns

The MVC framework consists of three components: the model, view, and controller.

The model is a data-driven component that functions as a simple database, providing functions such as insert, get, remove, update, and getNewId.

The view component includes various pages such as the Auth page, Contact page, and Note page. These pages serve as the user interface for accessing and interacting with the system.

The controller component handles the logic and functionality of the system. It includes classes such as Auth, Contact, Event, Export, Input, Note, Search, and Todo.

The Main Function Page serves as the primary gateway for users to access different features and options. It provides a comprehensive range of choices and a user-friendly interface.

2 Structure and Relationship among Code Components

2.1 Data Type Design

User

userId	userName	password
Integer	String	String

Note

noteID	noteTitle	noteContent	lastModifyTime
Integer	String	String	String

Contact

contactID	firstName	lastName	phoneNumber	address
Integer	String	String	String	String

Task

taskID	taskTitle	taskDescription	taskDDL
Integer	String	String	String

Event

eventID	eventTitle	eventDescription	eventStartTime	eventAlarm
Integer	String	String	String	String

In these five types of data - user, note, contact, task, and event are the basic data model of own personal information manager.

2.2 Function Design

CLASS (TYPE)	METHOD	ARGUMENT NAME AND TYPE	RETURN TYPE	EXCEPTION DECLARATION	EXPLANATION
SimpleDatabase (Model)	isDatabaseExist	N/A	Boolean	If files already have, just use it. If not, create the new files	Check whether five data files exist or not. If one data file does not exist, call the "createDatabase" method to create the corresponding file.
	createDatabase	N/A	N/A	N/A	Create (Initial) Database, add the catalog (title) to the first row of each data file.
	getNewID	String fileName	Integer	If no data in the file, catch the error and return 1	If the user creates a new PIR, get the new class ID for this PIR based on the largest class ID of each data file. The new class ID is the largest class ID + 1 and then becomes the new largest class ID for the next calling "getNewID" method.
	get	String fileName	String[][]	If there is no data in the file, catch the error and return an empty string	Get all data from each data file.
	insert	FileWriter fileWriter,	N/A	If error, catch the data, and throw the	Insert the data of the user account into

SimpleDatabase (Model)		String[][] csvData		error, and remind user	user.csv.
	insertSorting	FileWriter fileWriter, String[][] newData, String[][] tempFile, String fileName	N/A	If error, catch the data, and throw the error, and remind user	Each time the user creates a PIR, the function inserts the created PIR into a specific position of the data file. First, for all four types of PIRs, we sort PIRs by user ID from largest to smallest. Second, considering the same user ID, for events and tasks, we sort PIRs by time from early to late; For notes, we sort PIRs by time from late to early.
	updateSorting	File file, int contactID, String[] newData, String fileName	N/A	If error, catch the data, and throw the error, and remind user	Each time the user modifies a PIR, the function updates the original data to the modified data and sorts the modified PIR again. The rule of sorting is the same as which is described in the method “insertSorting”.
	remove	File file, int userID, int classID	N/A	If error, catch the data, and throw the error, and remind user	Remove the specific data defined by the user in the data file.
	search	String keyword, String fileType	String[][]	If there is no data contain the searching content in the file, catch the error and return an empty string	Search the content by keyword in the specific data file
	appendTo	String[][]	String[][]	If there is no data	Append the search

	Results	results, String[] newRow		contain the searching content in the file, catch the error and return an empty string	results to the array and return it
	searchBy Time	String inputTime, String fileType	String[][]	If there is no data contain the searching content in the file, catch the error and return an empty string	Search the content by time in the specific data file and need the operators “=”, “>”, “<”
	matchTim e	char operator, String timeInFile, String timeString	Boolean	Return false if not right	For the search by time, if matches, return true. If not, return false
	searchWit hLogicalC onnectors	String expression, String type	String[][]	If there is no data contain the searching content in the file, catch the error and return an empty string	Search the content by words or time with the logic operator in the specific data file and need the operators “&&”, “||”, “!”
	performSe arch	String query, String fileType	String[][]	If there is no data contain the searching content in the file, catch the error and return an empty string	Perform search on the given query and file type
	determine FileType	String type	String	N/A	Determine the file type from the given type string
	negateRes ults	String[][] results, String fileType	String[][]	If there is no data contain the searching content in the file, catch the error and return an empty string	Find the negate of two results, and this function is used for search with logic connectors
	intersectR esults	String[][] results1, String[][]	String[][]	If there is no data contain the searching content in the file,	Find the intersection of two results, and this function is used

		results2		catch the error and return an empty string	for search with logic connectors
	unionResults	String[][] results1, String[][] results2	String[][]	If there is no data contain the searching content in the file, catch the error and return an empty string	Find the union of two results, and this function is used for search with logic connectors
Auth (Controller)	login	String userName, String password	Boolean	Return false if not right	The public method interface of the private method verifyAccount to connect with view
	verifyAccount	String userName, String password	Boolean	Return false if not right	The method which verifies the account use the username and password that user typed matches the user data file
	signup	String userName, String password	N/A	If error, catch the data, and throw the error, and remind user	The public method interface of the private method createAccount to connect with view
	createAccount	String userName, String password	N/A	If error, catch the data, and throw the error, and remind user	The method which creates the account use the username and password that user typed is used the insert API to create a new user
	getUserId	N/A	Integer	N/A	Get the user ID from the Auth Class
	getUserName	N/A	String	N/A	Get the user name from the Auth Class
Contact (Controller)	createContact	Contact contactInfo	N/A	N/A	The public method interface of the private method

					createContact to connect with view
	createContact	String firstName, String lastName, String phoneNumber, String address, int userId	N/A	If error, catch the data, and throw the error, and remind user	The method that creates the contact information that the user typed is uses the insert API to create a new contact
	getAllContacts	N/A	String[][]	If error, catch the data, and throw the error, and remind user	The public method interface of the get all the contacts to connect with view
	getOneContact	String contactId	String[]	If error, catch the data, and throw the error, and remind user	The public method interface of the private method get one of the contacts to connect with view
	modifyContact	Contact contactInfo, String contactId	N/A	N/A	The public method interface of the private method createContact to connect with view
	modifyContact	String firstName, String lastName, String phoneNumber, String address, int userId, String contactId	N/A	If error, catch the data, and throw the error, and remind user	The method that updates the contact information that the user typed is used the update API to update that choose the contact
	removeContact	String contactId	N/A	N/A	The public method interface of the private method removeContact to connect with view
	removeCo	int contactId	N/A	If error, catch the	The method that

	ntact			data, and throw the error, and remind user	removes the contact information that the user typed is used the removed API to remove a contact that user choose
Note (Controller)	createNote	Note noteInfo	N/A	N/A	The public method interface of the private method createNote to connect with view
	createNote	String noteTitle, String noteContent, String lastModifyTime, int userId	N/A	If error, catch the data, and throw the error, and remind user	The method that creates the note that the user typed uses the insert API to create a new note
	getAllNotes	N/A	String[][]	If error, catch the data, and throw the error, and remind user	The public method interface of the get all the notes to connect with view
	getOneNote	String noteId	String[]	If error, catch the data, and throw the error, and remind user	The public method interface of the private method get one of the notes to connect with view
	modifyNote	Note noteInfo, String noteId	N/A	N/A	The public method interface of the private method modifyNote to note with view
	modifyNote	String noteTitle, String noteContent, String lastModifyTime, int userId, String noteId	N/A	If error, catch the data, and throw the error, and remind user	The method that updates the note that the user typed is used the update API to update that choose the note

	removeNote	String noteId	N/A	N/A	The public method interface of the private method removeNote to connect with view
	removeNote	int noteId	N/A	If error, catch the data, and throw the error, and remind user	The method that removes the note that the user typed uses the removed API to remove a note that the user chose
Todo (Controller)	createTask	Todo taskInfo	N/A	N/A	The public method interface of the private method createTask to connect with view
	createTask	String taskName, String taskDDL, String taskDescription, int userId	N/A	If error, catch the data, and throw the error, and remind user	The method that creates the task that the user typed uses the insert API to create a new task
	getAllTasks	N/A	String[][]	If error, catch the data, and throw the error, and remind user	The public method interface of the get all the tasks to connect with view
	getOneTask	String taskId	String[]	If error, catch the data, and throw the error, and remind user	The public method interface of the private method get one of the tasks to connect with view
	modifyTask	Todo taskInfo, String taskId	N/A	N/A	The public method interface of the private method modifyTask to note with view
	modifyTask	String taskName, String	N/A	If error, catch the data, and throw the error, and remind	The method that updates the task that the user typed is used

		taskDDL, String taskDescription, int userId, String taskId		user	the update API to update that choose the task
	removeTask	String taskId	N/A	N/A	The public method interface of the private method removeTask to connect with view
	removeTask	int taskId	N/A	If error, catch the data, and throw the error, and remind user	The method that removes the task that the user typed uses the removed API to remove a task that the user chose
Event (Controller)	createEvent	Event eventInfo	N/A	N/A	The public method interface of the private method createEvent to connect with view
	createEvent	String eventName, String eventStartTime, String eventAlarm, String eventDescription, int userId	N/A	If error, catch the data, and throw the error, and remind user	The method that creates the event that the user typed uses the insert API to create a new event
	getAllEvents	N/A	String[][]	If error, catch the data, and throw the error, and remind user	The public method interface of the get all the events to connect with view
	getOneEvent	String eventId	String[]	If error, catch the data, and throw the error, and remind user	The public method interface of the private method get one of the events to connect with view
	modifyEv	Event	N/A	N/A	The public method

	ent	eventInfo, String eventId			interface of the private method modifyEvent to event with view
	modifyEvent	String eventName, String eventStartTime, String eventAlarm, String eventDescription, int userId, String eventId	N/A	If error, catch the data, and throw the error, and remind user	The method that updates the event that the user typed is used the update API to update that choose the event
	removeEvent	String eventId	N/A	N/A	The public method interface of the private method removeEvent to connect with view
	removeEvent	int eventId	N/A	If error, catch the data, and throw the error, and remind user	The method that removes the event that the user typed uses the removed API to remove a event that the user chose
Search (Controller)	search	String keyword, String type	String[][]	N/A	Use the Model API - search() to catch the user input, and return the result
	searchByTime	String inputTime, String type	String[][]	N/A	Use the Model API - searchByTime() to catch the user input, and return the result
	searchWithLogicalConnectors	String expression, String type	String[][]	N/A	Use the Model API - searchWithLogicalConnectors() to catch the user input, and return the result
Export (Controll	export	N/A	N/A	If error, catch the data, and throw the	Export the .pim file which include all the

er)				error, and remind user	information for that user, which also can read by our program
Import (Controller)	importPIMFile	String fileName	N/A	If error, catch the data, and throw the error, and remind user	Get the user input, and pass the input to load method
	getFilesInInputFolder	N/A	ArrayList <String>	If no file return empty	Get the .pim file which in that folder - pim_file
	load	File file	N/A	N/A	Get the file, and read the .pim file, use insert to load the data to the CSV files
Input (Controller)	getInput	N/A	String	N/A	Get the user input (Handle after the input, and get the things)
	setInput	N/A	N/A	N/A	Collect and set the user input (Handle all the input)
PIM (Controller)	main	String[] args	N/A	N/A	The method call all the other methods to start
	checkDateFormat	String date	Boolean	If time format is wrong will remind user, and return false	Check the date and time format is correct or not, and handle the input error
	loginSignupConnector	N/A	String[]	N/A	The connector method for handle the login and signup view and input
	NoteConnector	String[] loginSignupInput	N/A	If the number input wrong, will be let user input again	The connector that for handle all the note view and Note Controller
	ContactCo	String[]	N/A	If the number input	The connector that for

	nnector	loginSignupInput		wrong, will be let user input again	handle all the contact view and Contact Controller
	TodoConnector	String[] loginSignupInput	N/A	If the number input wrong, will be let user input again	The connector that for handle all the todo view and Todo Controller
	EventConnector	String[] loginSignupInput	N/A	If the number input wrong, will be let user input again	The connector that for handle all the event view and Event Controller
	importConnector	N/A	N/A	If the number input wrong, will be let user input again	The connector that for handle all the import view and Import Controller
	exportConnector	N/A	N/A	If the number input wrong, will be let user input again	The connector that for handle all the export view and Export Controller
	MenuConnector	String[] loginSignupInput	N/A	If the number input wrong, will be let user input again	The connector for handle the menu view and user select go to which connector
	mainConnector	N/A	N/A	If login failed, will be not go to next step and exit system	The connector for handle if the login success, go to MenuConnector, if not, failed.

2.3 Diagrams

This section will show both the structure of and the relationship among the major code components in PIM, and how the methods of the code components transport and deal with the data when the user inputs the commands in PIM.

The following class diagram mainly demonstrates both the structure and the relationship among the major code components. The classes Auth, Contact, Event, Export, Note, Search, and Todo need to use the methods of the class SimpleDatabase. The class PIM needs to use the methods of

all other classes. The class Import needs to use the methods of the classes Contact, Event, Note and Todo.

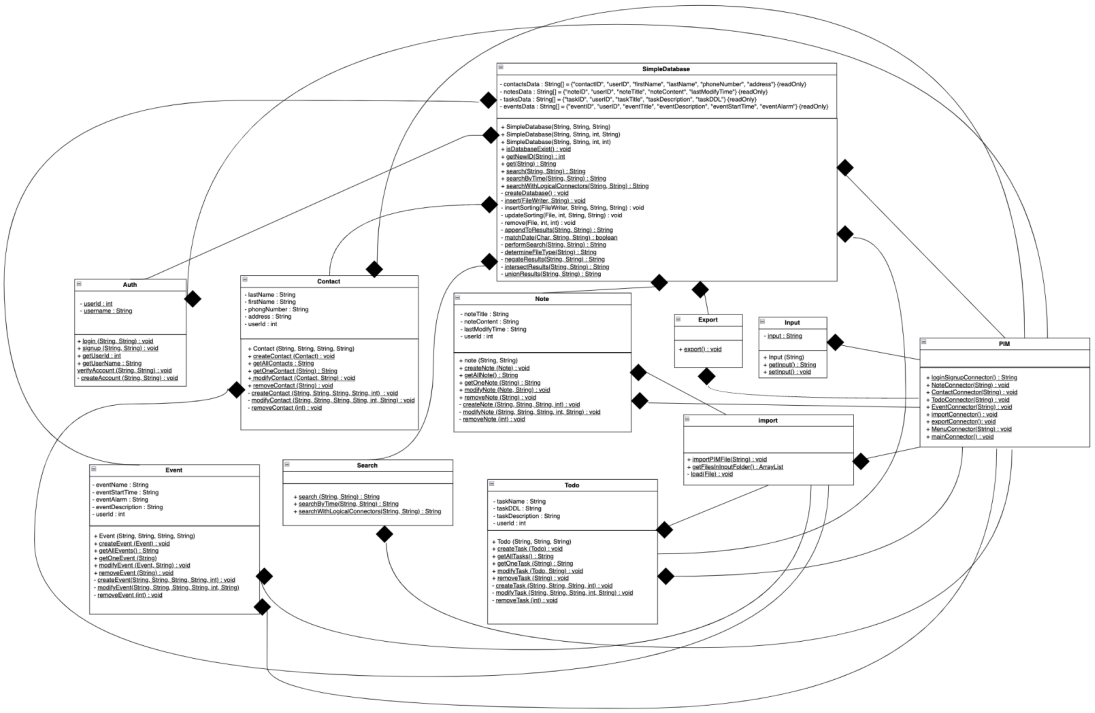

Figure 2: UML Class Diagram of the PIM

The following activity diagram mainly demonstrates how a user implements PIM based on the above major code components. When the user signs up a new account, loginSignupConnector() in PIM class calls createAccount() in Auth class to use insert() in SimpleDatabase class to insert the information of the user account into user.csv. When the user logs in the account, loginSignupConnector() in PIM class calls verifyAccount() in Auth class to judge whether the account exists. After logging in the account successfully, there is a main menu page and MenuConnector() in PIM class is called. If the user wants to enter into six subsystems from the main menu page, MenuConnector() calls NoteConnector(), ContactConnector(), TodoConnector(), EventConnector(), ImportConnector(), ExportConnector() separately. When the user wants to enter into four pages to operate four types of PIRs, the four connectors (NoteConnector(), ContactConnector(), TodoConnector() and EventConnector()) in PIM class calls the corresponding methods about getting, creating, modifying and removing four types of PIRs in Note, Contact, Todo and Event class separately further to use insertSorting(),

updateSorting(), get(), remove() and search() to operate the data sent by the user in four data files. Specially, for getting and searching, there is an additional step that requires the system to pass the return value of the corresponding functions back to the user interface. When the user wants to import a PIM file, ImportConnector() in PIM class calls load() in Import class to use insertSorting() in SimpleDatabase class to insert four types of PIRs in the PIM file into four data files in certain order. When the user wants to export a PIM file, ExportConnector() in PIM class calls export() in Export class to use get() in SimpleDatabase class to get four types of PIRs in four data files into a PIM file. When the user wants to exit the system, a corresponding command should be input.

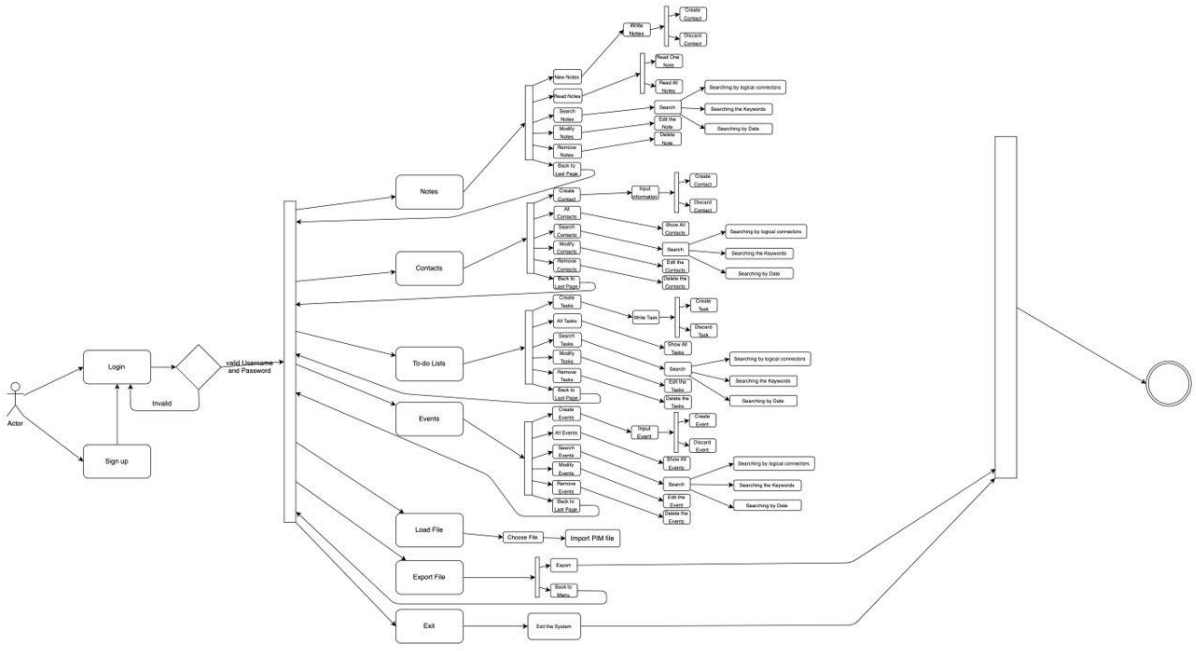

Figure 3: UML Activity Diagram of the PIM

3 Example Use

This section will outline the process of an example use of the PIM, how a user searches for some personal information records (PIRs) based on a criterion and prints out the matching PIRs.

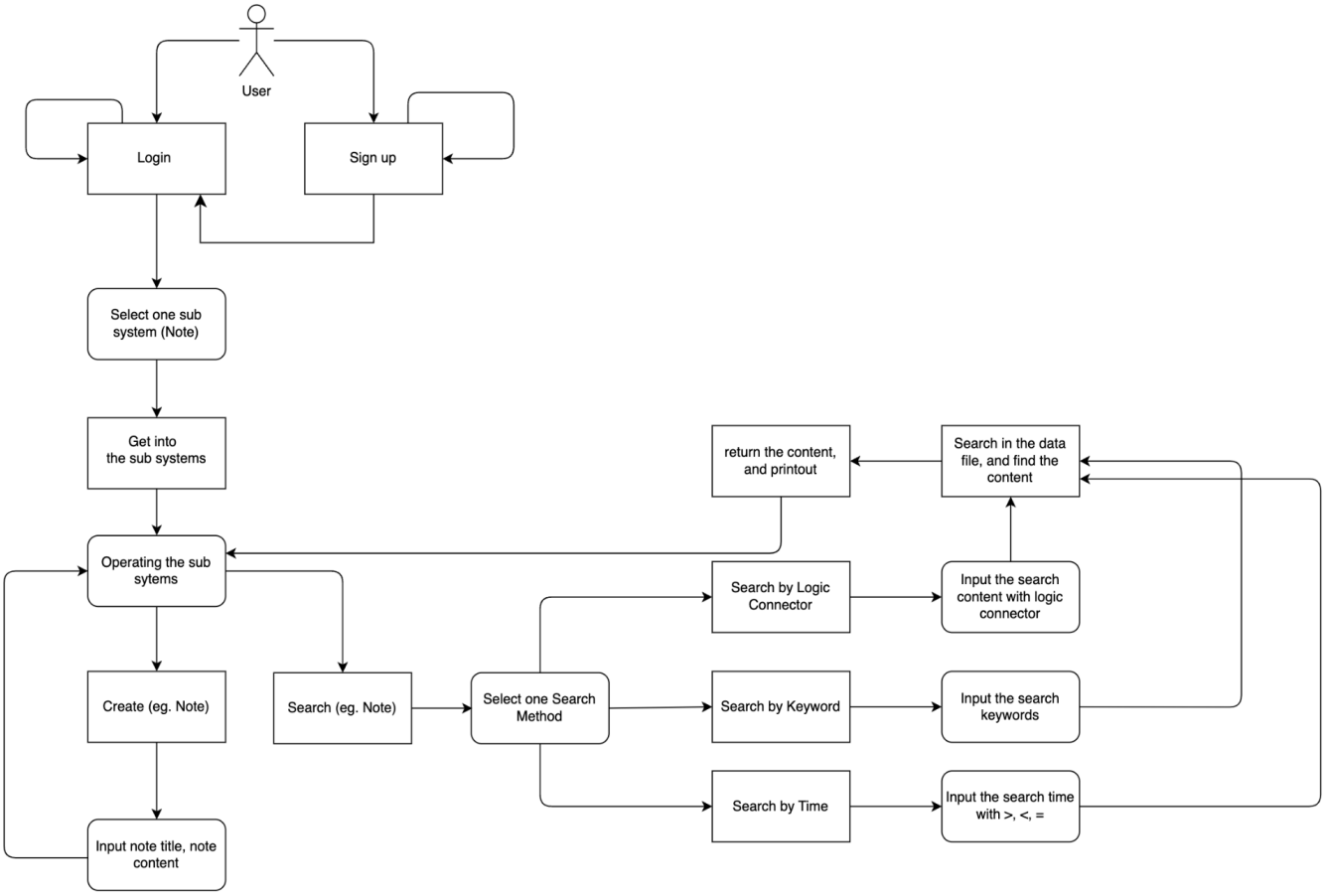

Figure 4: Example Use of Searching in PIM

The above activity diagram mainly demonstrates how a user searches for specific PIRs (notes as an example) based on a criterion and prints out the matching PIRs. First, the new user should sign up and then log in PIM. Second, according to the instructions on the main menu page, the user selects “Notes” to enter into the note page. Third, the user can input the command in the note page to create and then search for notes. For creating notes, the user should input the note title and note content. For searching notes, the user can select one mode among the three. Three modes separately are search by keyword, search by time and search with logical connector. When searching by keyword, the user should input a single keyword. When searching by time, the user should input an operator (>, <, =) with a single time. When searching with logical connector, the user should input multiple keywords and/or time with logical connectors (&&, ||, !). After inputting, the system starts to search in the data files. If the system finds the data successfully, it will be get from the model through the controller to the view and then be printed out on the interface.