Department of Computing
電子計算學系

85th THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

COMP3211 Software Engineering

Group 13

Personal Information Manager

Software Requirements Specification

JIANG Guanlin (21093962d)

MENG Guanlin (20099185d)

HU Yuhang (21106395d)

YE Feng (21098249d)

Table of Contents

1 Preface

1.1 Expected Readership: Project Stakeholders, Future Development and Maintenance Teams, User

1.2 version history: 1.0

1.3 Updates: N/A

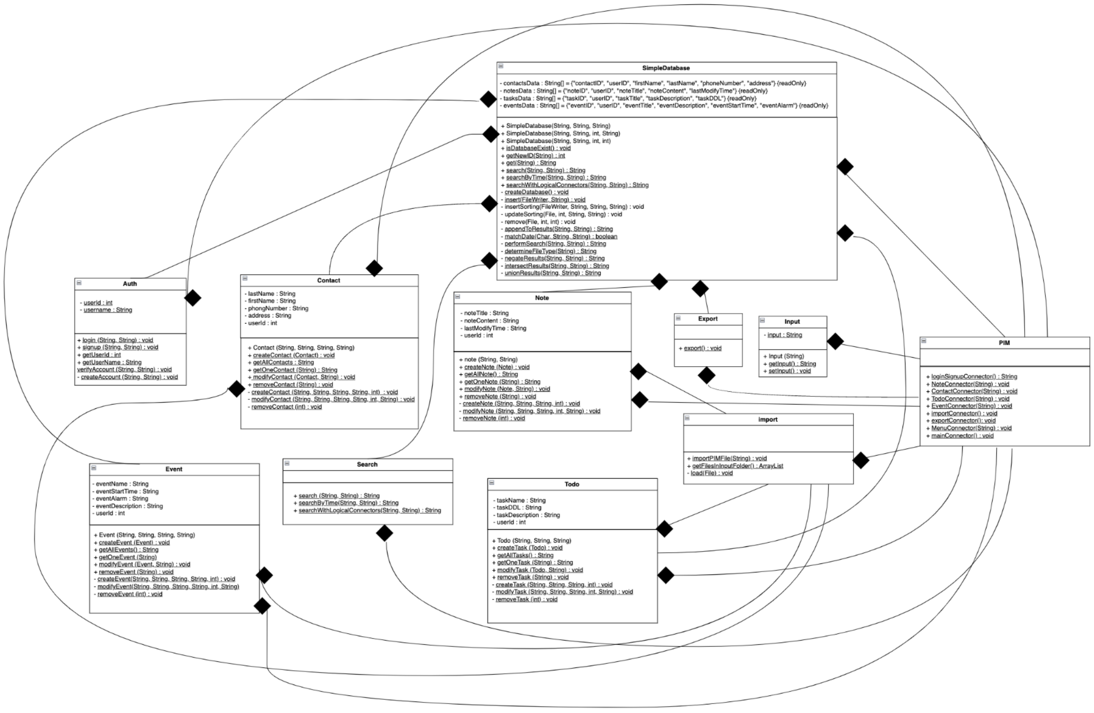
1.4 rationale: N/A

2 Introduction

2.1 Purpose

A **command-line-based personal information manager (PIM)** is designed to provide users with a centralized system for managing various types of personal information records (PIRs). It aims to enable users to efficiently organize and access their notes, tasks, events, contacts, and other important information in a single location, facilitating productivity, organization, and effective personal information management.

2.2 System Functions


SimpleDatabase
- contactsData : String[] = {"contactID", "userID", "firstName", "lastName", "phoneNumber", "address"} {readOnly}
- notesData : String[] = {"noteID", "userID", "noteTitle", "noteContent", "lastModifyTime"} {readOnly}
- tasksData : String[] = {"taskID", "userID", "taskTitle", "taskDescription", "taskDDL"} {readOnly}
- eventsData : String[] = {"eventID", "userID", "eventTitle", "eventDescription", "eventStartTime", "eventAlarm"} {readOnly}
+ SimpleDatabase(String, String, String)
+ SimpleDatabase(String, String, int, String)
+ SimpleDatabase(String, String, int, int)
+ isDatabaseExist() : void
+ getNewID(String) : int
+ get(String) : String
+ search(String, String) : String
+ searchByTime(String, String) : String
+ searchWithLogicalConnectors(String, String) : String
- createDatabase() : void
- insert(FileWriter, String) : void
- insertSorting(FileWriter, String, String, String) : void
- updateSorting(File, int, String, String) : void
- remove(File, int, int) : void
- appendToResults(String, String) : String
- matchDate(Char, String, String) : boolean
- performSearch(String, String) : String
- determineFileType(String) : String
- negateResults(String, String) : String
- intersectResults(String, String) : String
- unionResults(String, String) : String
Auth
- userId : int
- username : String
+ login (String, String) : void
+ signup (String, String) : void
+ getUserId : int
+ getUserName : String
verifyAccount (String, String) : void
- createAccount (String, String) : void
Contact
- lastName : String
- firstName : String
- phongNumber : String
- address : String
- userId : int
+ Contact (String, String, String, String)
+ createContact (Contact) : void
+ getAllContacts : String
+ getOneContact (String) : String
+ modifyContact (Contact, String) : void
+ removeContact (String) : void
- createContact (String, String, String, String, int) : void
- modifyContact (String, String, String, Sting, int, String) : void
- removeContact (int) : void
Note
- noteTitle : String
- noteContent : String
- lastModifyTime : String
- userId : int
+ note (String, String)
+ createNote (Note) : void
+ getAllNote() : String
+ getOneNote (String) : String
+ modifyNote (Note, String) : void
+ removeNote (String) : void
- createNote (String, String, String, int) : void
- modifyNote (String, String, String, int, String) : void
- removeNote (int) : void
Export
+ export() : void
Input
- input : String
+ Input (String)
+ getInput() : String
+ setInput() : void
PIM
+ loginSignupConnector() : String
+ NoteConnector(String) : void
+ ContactConnector(String) : void
+ TodoConnector(String) : void
+ EventConnector(String) : void
+ importConnector() : void
+ exportConnector(): void
+ MenuConnector(String) : void
+ mainConnector() : void
import
+ importPIMFile(String) : void
+ getFilesInInputFolder() : ArrayList
- load(File) : void
Event
- eventName : String
- eventStartTime : String
- eventAlarm : String
- eventDescription : String
- userId : int
+ Event (String, String, String, String)
+ createEvent (Event) : void
+ getAllEvents() : String
+ getOneEvent (String)
+ modifyEvent (Event, String) : void
+ removeEvent (String) : void
- createEvent(String, String, String, String, int) : void
- modifyEvent(String, String, String, String, int, String)
- removeEvent (int) : void
Search
+ search (String, String) : String
+ searchByTime(String, String) : String
+ searchWithLogicalConnectors(String, String) : String
Todo
- taskName : String
- taskDDL : String
- taskDescription : String
- userId : int
+ Todo (String, String, String)
+ createTask (Todo) : void
+ getAllTasks() : String
+ getOneTask (String) : String
+ modifyTask (Todo, String) : void
+ removeTask (String) : void
- createTask (String, String, String, int) : void
- modifyTask (String, String, String, int, String) : void
- removeTask (int) : void

The system functions of the program include creating different types of personal information records (PIRs) such as plain texts, tasks, events, and contacts. Users can also modify existing records, search for records based on criteria, print out detailed information, delete specified records, store records in a file, and upload records from a file.r

3 Glossary

To aid users in understanding the document, this part mostly specifies the technical vocabulary used in it and links its description to its acronym.

Abbreviation	Meaning

CLI	**Command-line Interface**
PIM	**Personal Information Manager**
PIR	**Personal Information Record**
MVC	**Model-View-Controller**
SFR	**System Functional Requirement**
OOP	**Object-Oriented Programming**
NFR	**Non-functional Requirement**
SRS	**Software Requirements Specification**
UR	**User Requirement**

4 User requirements definition

4.1 User Requirements

UR-1. The system shall allow users to create different types of PIRs including plain texts for quick notes, tasks with descriptions and deadlines, events with details and alarms, and contacts with names, addresses, and phone numbers.(in one page).

UR-2. The system shall provide the ability for users to create plain text PIRs, including titles, and contents, for taking quick notes.

UR-3. The system shall allow users to create tasks as PIRs, including titles, descriptions and deadlines, to effectively manage their to-do lists.

UR-4. The system shall support the creation of event PIRs, including titles, descriptions, starting times, and alarms, to assist users in organizing their schedules.

UR-5. The system shall allow users to create contact PIRs, including first names, last names, addresses, and mobile numbers, to efficiently manage their contacts.

UR-6. The system shall allow users to edit and update the data in existing PIRs to ensure information remains current and accurate.

UR-7. The system shall provide search functionality for users to find PIRs based on criteria such as type and data stored in their fields. The criteria should include text matching, time comparisons, logical connectors, and negation.

UR-8. The system shall allow users to print detailed information about specific PIRs or all PIRs.

UR-9. The system shall allow users to delete specified PIRs.

UR-10. The system shall allow users to store PIRs in a ".pim" file format for future access and retrieval.

UR-11. The system shall provide the ability to load PIRs from a ".pim" file, allowing users to continue working with previously stored PIRs.

4.2 User Non-Functional Requirements

Usability: The system shall feature an intuitive interface that allows users to easily navigate and manage their PIRs with minimal learning curve.

Performance: The system shall perform actions such as search, save, and load quickly and efficiently, even as the number of PIRs grows.

Reliability: The system shall be reliable, ensuring that PIRs are not lost or corrupted, and that the system is available whenever the user needs it.

Security: The system shall provide that the sensitive information within PIRs must be protected against unauthorized access or breaches and shall employ robust security measures.

Scalability: The system shall be able to scale with the user's needs, accommodating an increasing number of PIRs without degradation in performance.

Interoperability: The system should be capable of interacting with other applications if necessary, allowing for import/export of data in different formats.

Maintainability: The system shall be designed in a way that facilitates easy updates and maintenance without significant downtime.

Compliance: The system should adhere to relevant data protection regulations and standards to ensure compliance with legal requirements.

Availability: The system shall manage PIRs in an orderly way so that users can arrange their affairs better when they want to check one type of all PIRs.

5 System architecture

5.1 Architectural Patterns

The MVC (Model-View-Controller) architectural pattern is used in the anticipated system architecture. It consists of three main components: the model, view, and controller.

1. Model: This component serves as the data-driven module of the system and functions as a simple database. It provides functions such as insert, get, remove, update, and getNewId. The model is reused across different modules to handle data storage and retrieval.

2. View: 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 view is responsible for displaying data to the user and receiving user input. It is reused across different modules to provide a consistent and user-friendly interface.

3. Controller: The controller component handles the logic and functionality of the system. It includes functions such as Auth, Contact, Event, Export, Input, Note, Search, and Todo. The controller acts as an intermediary between the model and view, processing user input, updating the model, and updating the view accordingly. It is reused across different modules to handle the system's functionality and ensure proper data flow.

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. The available options on the Main Function Page are highlighted as reusable components:

1. Notes: Reuses the model, view, and controller components to manage notes, including creating, reading, searching, modifying, and removing notes.

2. Contacts: Reuses the model, view, and controller components to manage contacts, including viewing, searching, modifying, and removing contacts.

3. Events: Reuses the model, view, and controller components to manage events, including viewing, searching, modifying, and removing events.

4. To-Do List: Reuses the model, view, and controller components to create and manage tasks and to-do lists.

5. Export .pim file: Reuses the model and controller components to export data or settings into a .pim file.

6. Load .pim file: Reuses the model and controller components to import .pim files into the system's data storage.

7. Exit System: Provides an option for users to exit or close the management system.

5.2 Overall Structure

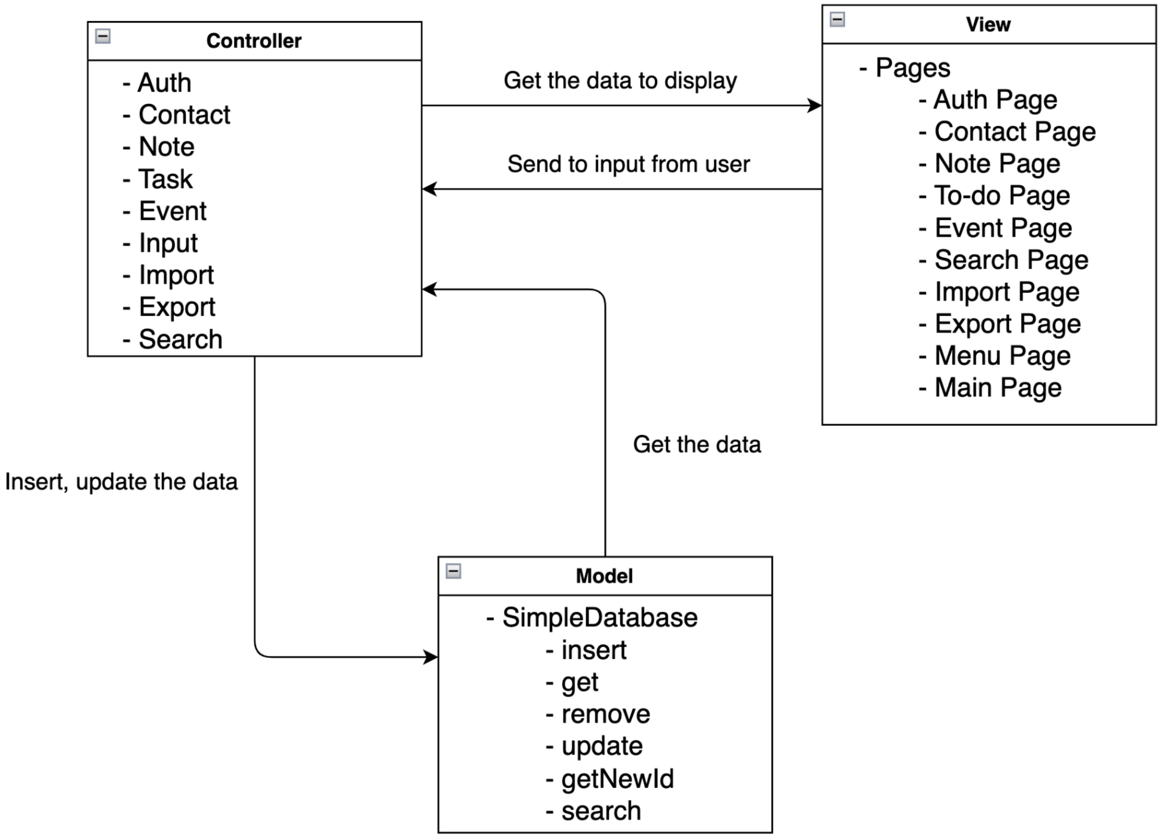


Figure: MVC Architectural Patterns

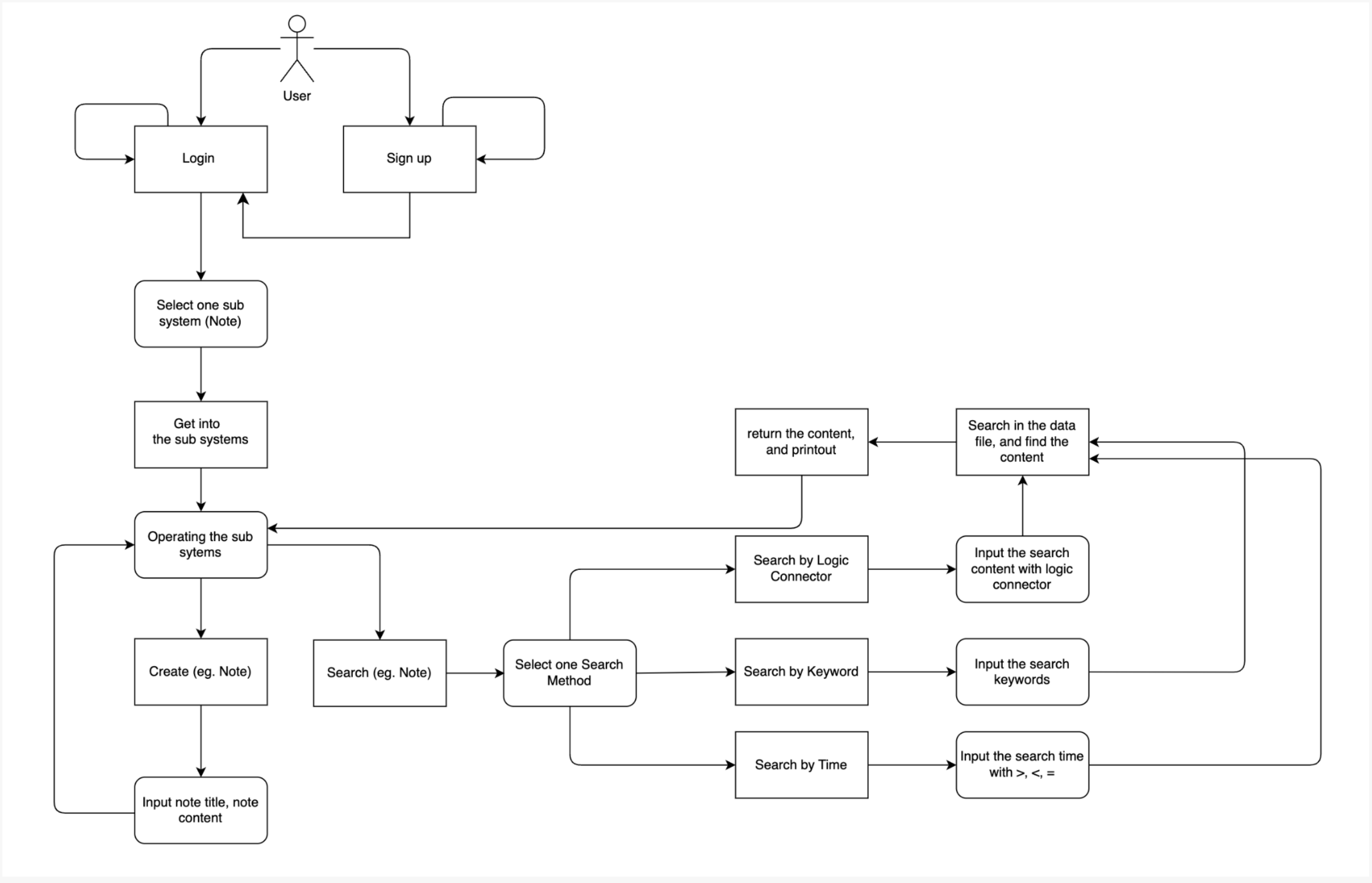
The MVC framework consists of three components: the model, view, and controller. The model is a data-driven component that functions as a simple database, providing functions such as insert, get, remove, update, and getNewId.

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.

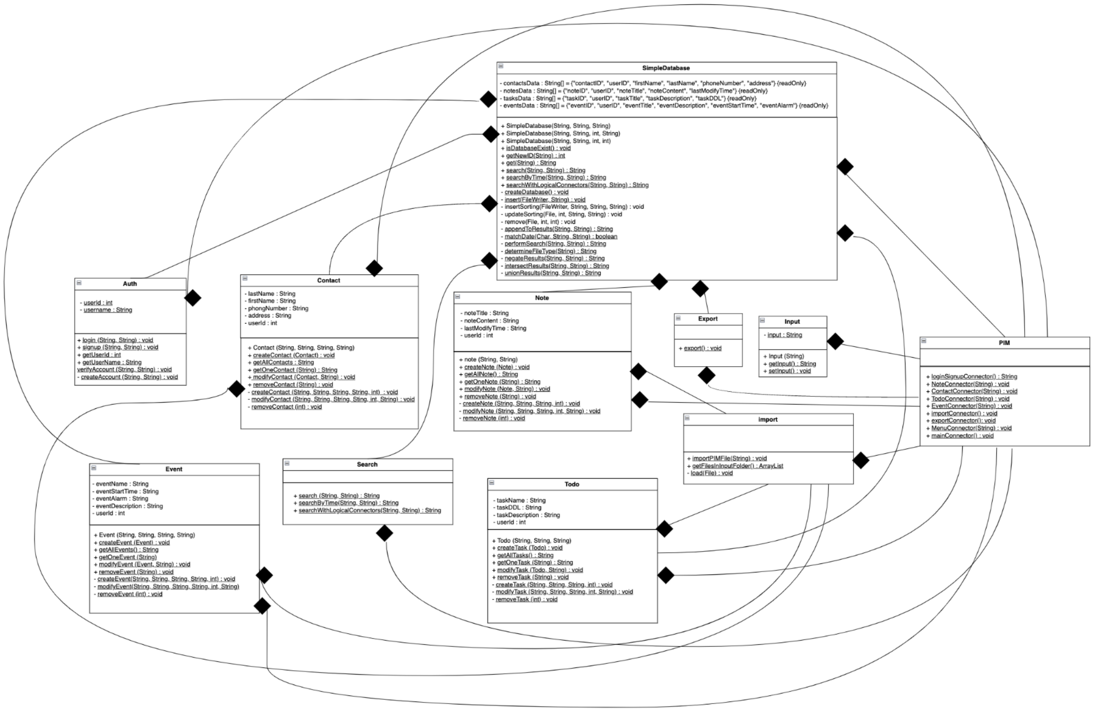
The following activity diagram mainly demonstrates how a user searches for specific PIRs (notes as an example) based on a criterion and prints out the matching PIRs. First, the new user should sign up and then log in PIM. Second, according to the instructions on the main menu page, the user selects "Notes" 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.


User
Login
Sign up
Select one sub system (Note)
Get into the sub systems
Operating the sub sytems
Create (eg. Note)
Input note title, note content
Search (eg. Note)
Select one Search Method
Search by Logic Connector
Search by Keyword
Search by Time
Input the search content with logic connector
Input the search keywords
Input the search time with >, <, =
Search in the data file, and find the content
return the content, and printout

5.3 Interaction Relationship

This section would show the architecture of the connections and interactions of each component.

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.


SimpleDatabase
- contactsData : String[] = {"contactID", "userID", "firstName", "lastName", "phoneNumber", "address"} {readOnly}
- notesData : String[] = {"noteID", "userID", "noteTitle", "noteContent", "lastModifyTime"} {readOnly}
- tasksData : String[] = {"taskID", "userID", "taskTitle", "taskDescription", "taskDDL"} {readOnly}
- eventsData : String[] = {"eventID", "userID", "eventTitle", "eventDescription", "eventStartTime", "eventAlarm"} {readOnly}
+ SimpleDatabase(String, String, String)
+ SimpleDatabase(String, String, int, String)
+ SimpleDatabase(String, String, int, int)
+ isDatabaseExist() : void
+ getNewID(String) : int
+ get(String) : String
+ search(String, String) : String
+ searchByTime(String, String) : String
+ searchWithLogicalConnectors(String, String) : String
- createDatabase() : void
- insert(FileWriter, String) : void
- insertSorting(FileWriter, String, String, String) : void
- updateSorting(File, int, String, String) : void
- remove(File, int, int) : void
- appendToResults(String, String) : String
- matchDate(Char, String, String) : boolean
- performSearch(String, String) : String
- determineFileType(String) : String
- negateResults(String, String) : String
- intersectResults(String, String) : String
- unionResults(String, String) : String
Auth
- userId : int
- username : String
+ login (String, String) : void
+ signup (String, String) : void
+ getUserId : int
+ getUserName : String
verifyAccount (String, String) : void
- createAccount (String, String) : void
Contact
- lastName : String
- firstName : String
- phongNumber : String
- address : String
- userId : int
+ Contact (String, String, String, String)
+ createContact (Contact) : void
+ getAllContacts : String
+ getOneContact (String) : String
+ modifyContact (Contact, String) : void
+ removeContact (String) : void
- createContact (String, String, String, String, int) : void
- modifyContact (String, String, String, Sting, int, String) : void
- removeContact (int) : void
Note
- noteTitle : String
- noteContent : String
- lastModifyTime : String
- userId : int
+ note (String, String)
+ createNote (Note) : void
+ getAllNote() : String
+ getOneNote (String) : String
+ modifyNote (Note, String) : void
+ removeNote (String) : void
- createNote (String, String, String, int) : void
- modifyNote (String, String, String, int, String) : void
- removeNote (int) : void
Export
+ export() : void
Input
- input : String
+ Input (String)
+ getInput() : String
+ setInput() : void
PIM
+ loginSignupConnector() : String
+ NoteConnector(String) : void
+ ContactConnector(String) : void
+ TodoConnector(String) : void
+ EventConnector(String) : void
+ importConnector() : void
+ exportConnector(): void
+ MenuConnector(String) : void
+ mainConnector() : void
import
+ importPIMFile(String) : void
+ getFilesInInputFolder() : ArrayList
- load(File) : void
Event
- eventName : String
- eventStartTime : String
- eventAlarm : String
- eventDescription : String
- userId : int
+ Event (String, String, String, String)
+ createEvent (Event) : void
+ getAllEvents() : String
+ getOneEvent (String)
+ modifyEvent (Event, String) : void
+ removeEvent (String) : void
- createEvent(String, String, String, String, int) : void
- modifyEvent(String, String, String, String, int, String)
- removeEvent (int) : void
Search
+ search (String, String) : String
+ searchByTime(String, String) : String
+ searchWithLogicalConnectors(String, String) : String
Todo
- taskName : String
- taskDDL : String
- taskDescription : String
- userId : int
+ Todo (String, String, String)
+ createTask (Todo) : void
+ getAllTasks() : String
+ getOneTask (String) : String
+ modifyTask (Todo, String) : void
+ removeTask (String) : void
- createTask (String, String, String, int) : void
- modifyTask (String, String, String, int, String) : void
- removeTask (int) : void

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.

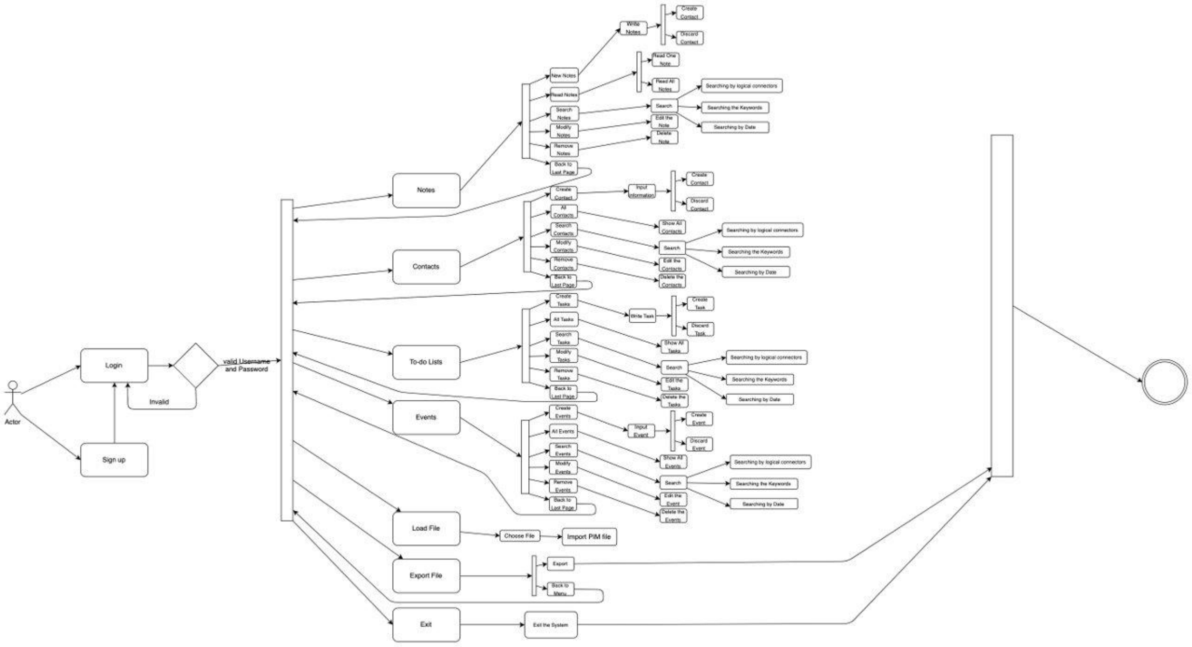


Figure 4: UML Activity Diagram of the PIM

6 System requirements specification

This section describes functional and non-functional requirements in more details and futher details will added to the non-functional requirements.

6.1 System Functional Requirement

ID	SFR-01
Title	Unified PIR Management
Requirement	System shall let users create and manage different types of personal information records (PIRs) within a unified location.
Rationale	To provide a centralized platform for users to store and access various types of personal information.
Reference	UR1
Priority	High

ID	SFR-02
Title	Plain Text PIRs
Requirement	The system shall provide the ability for users to create plain text PIRs for taking quick notes.
Rationale	To allow users to capture and store quick notes easily
Reference	UR-2
Priority	High

ID	SFR-03
Title	Task Management
Requirement	Users shall have the ability to create tasks as PIRs, including descriptions and deadlines, to effectively manage their to-do lists.

Rationale	To enable users to track and organize their tasks efficiently.
Reference	UR-3
Priority	High

ID	SFR-04
Title	Event Management
Requirement	The system shall support the creation of event PIRs, including descriptions, starting times, and alarms, to assist users in organizing their schedules.
Rationale	To help users manage their events and schedule effectively.
Reference	UR-4
Priority	High

ID	SFR-05
Title	Contact Management
Requirement	The system shall allow users to create contact PIRs, including names, addresses, and mobile numbers, to efficiently manage their contacts.
Rationale	To provide users with a convenient way to store and access their contact information.
Reference	UR-5
Priority	High

ID	SFR-06
Title	PIR Modification

Requirement	The system shall permit users to modify the data within existing PIRs to keep the information up to date.
Rationale	To allow users to edit and update their personal information records as needed.
Reference	UR-6
Priority	Medium

ID	SFR-07
Title	PIR Search Functionality
Requirement	The system shall provide search functionality for users to find PIRs based on criteria such as type and data stored in their fields. The criteria should include text matching, time comparisons, logical connectors, and negation
Rationale	To enable users to locate specific PIRs quickly and efficiently.
Reference	UR-7
Priority	High

ID	SFR-08
Title	Printing PIRs
Requirement	The system shall allow users to print detailed information about specific PIRs or all PIRs.
Rationale	To provide users with the option to generate hard copies of their personal information records.
Reference	UR-7
Priority	Medium

ID	SFR-09

Title	PIR Deletion
Requirement	The system shall allow users to delete specified PIRs.
Rationale	To provide users with the capability to remove unwanted or outdated personal information records.
Reference	UR-9
Priority	High

ID	SFR-10
Title	PIR Export
Requirement	Users shall have the capability to store PIRs in a ".pim" file format for future access and retrieval.
Rationale	To enable users to backup and transfer their personal information records in a standardized file format.
Reference	UR-10
Priority	High

ID	SFR-11
Title	PIR Import
Requirement	The system shall provide the ability to load PIRs from a ".pim" file, allowing users to continue working with previously stored PIRs.
Rationale	To allow users to restore their personal information records from backup files or other sources.
Reference	UR-11
Priority	High

6.2 System Non-Functional Requirement

ID	NFR-01
Title	Security
Requirement	The system shall implement robust security measures to protect the privacy and confidentiality of users' personal information. This includes secure authentication and access controls. User roles and permissions should be defined to control access to different types of PIRs based on user privileges.
Rationale	To ensure the privacy and security of user data.
Reference	-
Priority	High

ID	NFR-02
Title	Reliability
Requirement	The system shall be highly reliable, with minimal downtime and data loss, to ensure that users can access and manage their personal information records without interruptions.
Rationale	To avoid inconvenience and data loss for users.
Reference	-
Priority	High

ID	NFR-03
Title	Usability

Requirement	The system shall have a user-friendly interface with intuitive navigation and clear instructions to ensure ease of use for all users.
Rationale	To enhance user satisfaction and minimize the learning curve.
Reference	-
Priority	High

ID	NFR-04
Title	Scalability
Requirement	The system shall be able to handle a growing number of users and an increasing amount of personal information records without compromising performance.
Rationale	To accommodate future growth and ensure system availability for all users.
Reference	-
Priority	Medium

ID	NFR-05
Title	Performance
Requirement	The system shall provide fast response times for user interactions, ensuring quick loading of pages and efficient execution of operations.
Rationale	To enhance user experience and productivity.
Reference	-
Priority	High

ID	NFR-06

Title	Maintainability
Requirement	The system shall be designed and implemented using modular and well-documented code, making it easier to maintain, update, and enhance in the future.
Rationale	To reduce maintenance efforts and facilitate future system improvements.
Reference	-
Priority	Medium

ID	NFR-07
Title	Availability
Requirement	The system shall manage PIRs in an orderly way so that users can arrange their affairs better when they want to check one type of all PIRs.
Rationale	To enhance user experience and improve the performance of the system.
Reference	-
Priority	Medium