Software Engineering Design I

Winter 2015

Assignment 3 – Software Design Document

Group #1

PRMS

PROFESSIONAL RESOURCE MANAGEMENT SYSTEM

Contributing Members:

Abad H., Brandon C. and Rafael N.

Date Submitted: 3/3/2015

Approval

This document has been read and approved by the following team members responsible for its implementation:

Print Name	Signature	Comments

Print Name	Signature	Comments

Print Name	Signature	Comments

Document Status Sheet

Document Title	Software Design Document
Document ID	PRMS/Documents/Product/SDD/1.1.0
Author(s)	A. Hameed, B. Crispino, R. Nazario
Version	1.1.0
Document Status	draft / internally approved / [✓ conditionally approved] / approved

Version	**Date**	**Author(s)**	**Comments**
0.0.1	26-02-2015	A. Hameed	Document Creation
0.0.2	26-02-2015	A. Hameed	Lay out the full document template with headings and formatting
0.0.3	26-02-2015	A. Hameed	Added text to ch. 1.3, 2, 3, 4.1-4.2
0.0.4	27-02-2015	B. Crispino	Added class diagrams to ch. 5
0.0.5	27-02-2015	A. Hameed	Added text to ch. 6 and 7
0.0.6	27-02-2015	B. Crispino	Added sequence diagrams to ch. 7
0.0.7	27-02-2015	A. Hameed	Added GUI diagrams to ch. 9
0.0.8	01-03-2015	R. Nazario	Added text to ch. 1.1
0.0.9	02-03-2015	A. Hameed	Added text to ch. 1.2, 4.3
1.0.0	02-03-2015	R. Nazario, B. Crispino	Added text to ch. 6
1.0.1	02-03-2015	B. Crispino	Added text to ch. 8
1.0.2	02-03-2015	A. Hameed	Added text to ch. 10 and 11
1.0.3	02-03-2015	R. Nazario, A. Hameed, B. Crispino	Internal review complete
1.0.4	02-03-2015	R. Nazario, A. Hameed, B. Crispino	Internally accepted
1.1.0	02-03-2015	A. Hameed	Fixed document status

Contents

1 Introduction

1.1 System Overview

The software will be used to implement a client-server resource acquisition system, namely Professional Resource Management System (PRMS). This system allows users to access a database of library resources consisting of textbooks, encyclopaedias, articles and technical documents, from which they can acquire resources relevant to their needs. It hides the complexity of a resource acquisition system making the software easy to use. Usability is further increased by offering a front-end GUI for users to log in to and access the system.

The environment identified for the PRMS system is similar to a library system. The software will allow the user to specify one or more of the following to search/filter through the database of resources: document name, document type, subject, publication year and author. The software also enables the user to sort acquired data by document type, subject, publication year and author for efficiency and a better user experience. Being a client-server system, all user requests at the client system will be sent to a server for acknowledgement and a response. System administrators will be the power users with access to the backend of the software.

1.2 Design Objectives

The PRMS system design revolves around the principle of a software which is easy-to-use and efficient for the user. Being a client-server system, the ability to service multiple clients at any given time is also its top priority. Design objectives, therefore, focus heavily on providing the user with the best user experience. The software design will consist of three major components: front-end, back-end and client-server relationship.

The front-end design (client system) will primarily consist of the GUI which the user will interface with directly. Its design will allow any user to be able to both understand and use the software without confusion. The back-end design (server) will provide all the functionality to carry out the various tasks a user may perform at the client side. Major functions of the design include the user account validation and searching, filtering and sorting methodologies. Search and sorting algorithms will be implemented for accuracy and efficiency. The third component; client-server design will be the bond between the front and back end systems. Functionality in the design will require multiple clients to connect to a given server and data transfer between them will need to be kept accurate and efficient with minimal lag time.

With the PRMS system being a network dependant software, the design will have to account for potential network loss and/or failure. The design will also ensure all user information and account privileges are kept private and secure.

1.3 Documents

1.3.1 Reference Documents

[SWE Design I Project Outline]	Naser, H. (2015). Team-project: client-server resource acquisition system [class handout]. Department of Engineering, Lakehead University, Thunder Bay, ON.
[SWE 3670 Textbook]	Schach, S. (2011). *Object-Oriented and Classical Software Engineering* (8th ed., p. 667). New York: McGraw-Hill Education.
[SWE 3050 Textbook]	Lethbridge, T., & Laganiere, R. (2005). *Object-Oriented Software Engineering* (2nd ed., p. 533). Maidenhead, Berkshire: McGraw-Hill Education.

1.3.2 Applicable Documents

[SW Requirements Document]	Hameed, A. Crispino, B., Nazario, R. (2015). PRMS [requirements document]. Department of Engineering, Lakehead University, Thunder Bay, ON.
[SW Specifications Document]	Hameed, A. Crispino, B., Nazario, R. (2015). PRMS [specifications document]. Department of Engineering, Lakehead University, Thunder Bay, ON.

2 Glossary

2.1 Definitions

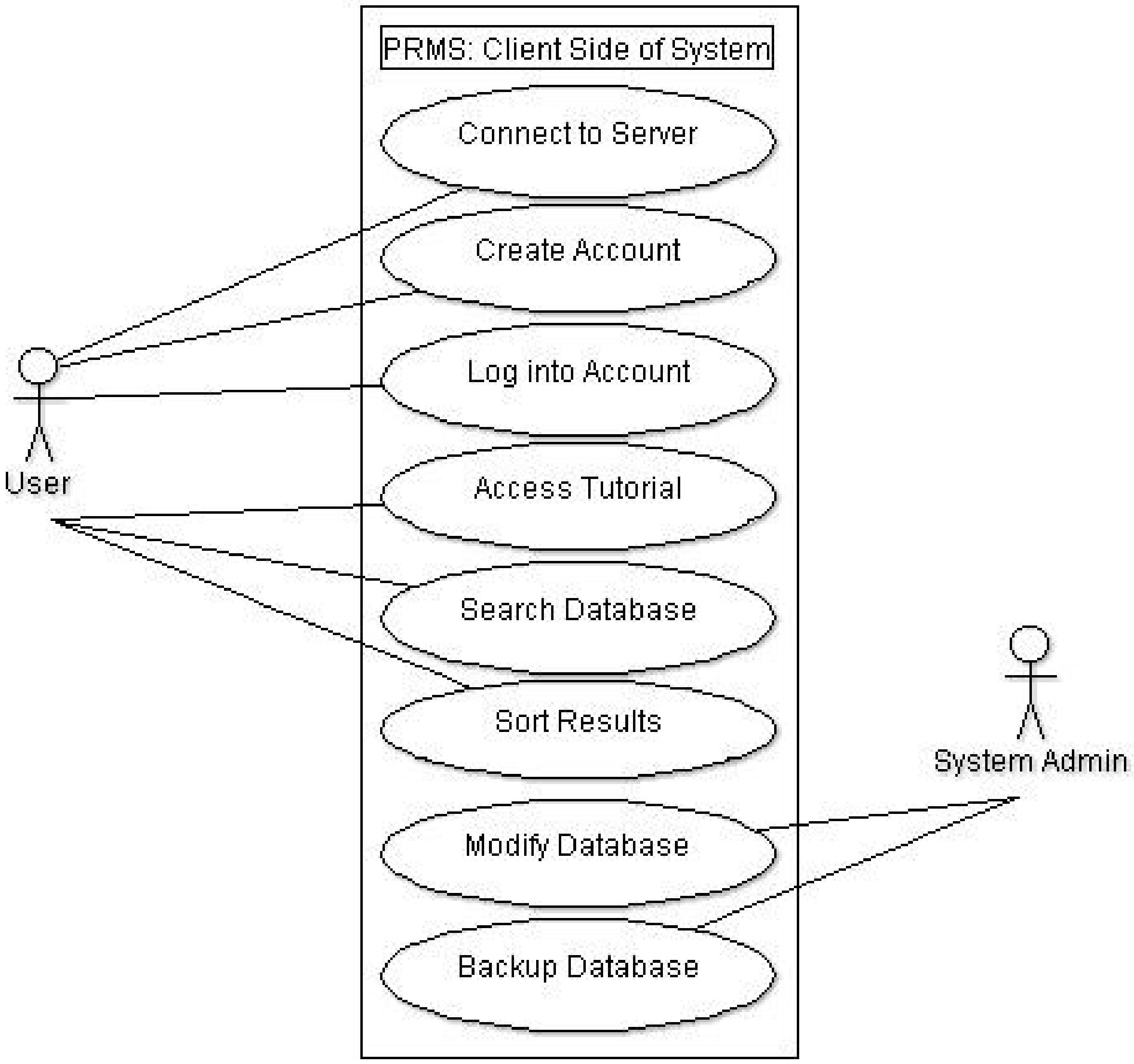
Algorithm	A process or set of rules to be followed in calculations or other problem-solving operations.
Client	Program that is used by all the users and system admins.
PRMS Software	A software implementing a client-server resource acquisition system.
Resource	Material which can be readily drawn from a server and displayed to the client for effective use.
Server	A computer or computer program that manages access to a centralized resource or service in a network (i.e. PRMS database)
System Administrator	The system admin oversees the entire PRMS system and has the right to configure the system, to create and remove admins and resources as well as any other high level configuration.
User	Any individual or system administrator with access to the client machine using PRMS.

2.2 Abbreviations

CASE Tools	Computer-aided Software Engineering Tools
COCOMO	Constructive Cost Model
GUI	Graphical user Interface
IDE	Integrated Development Environment
IP	Internet Protocol
KLOC	Thousands (K) of Lines of Code
PRMS	Professional Resource Management System
SVN	Apache Subversion
TCP	Transmission Control Protocol
UML	Unified Modeling Language
XML	eXtensible Markup Language

3 Use Cases

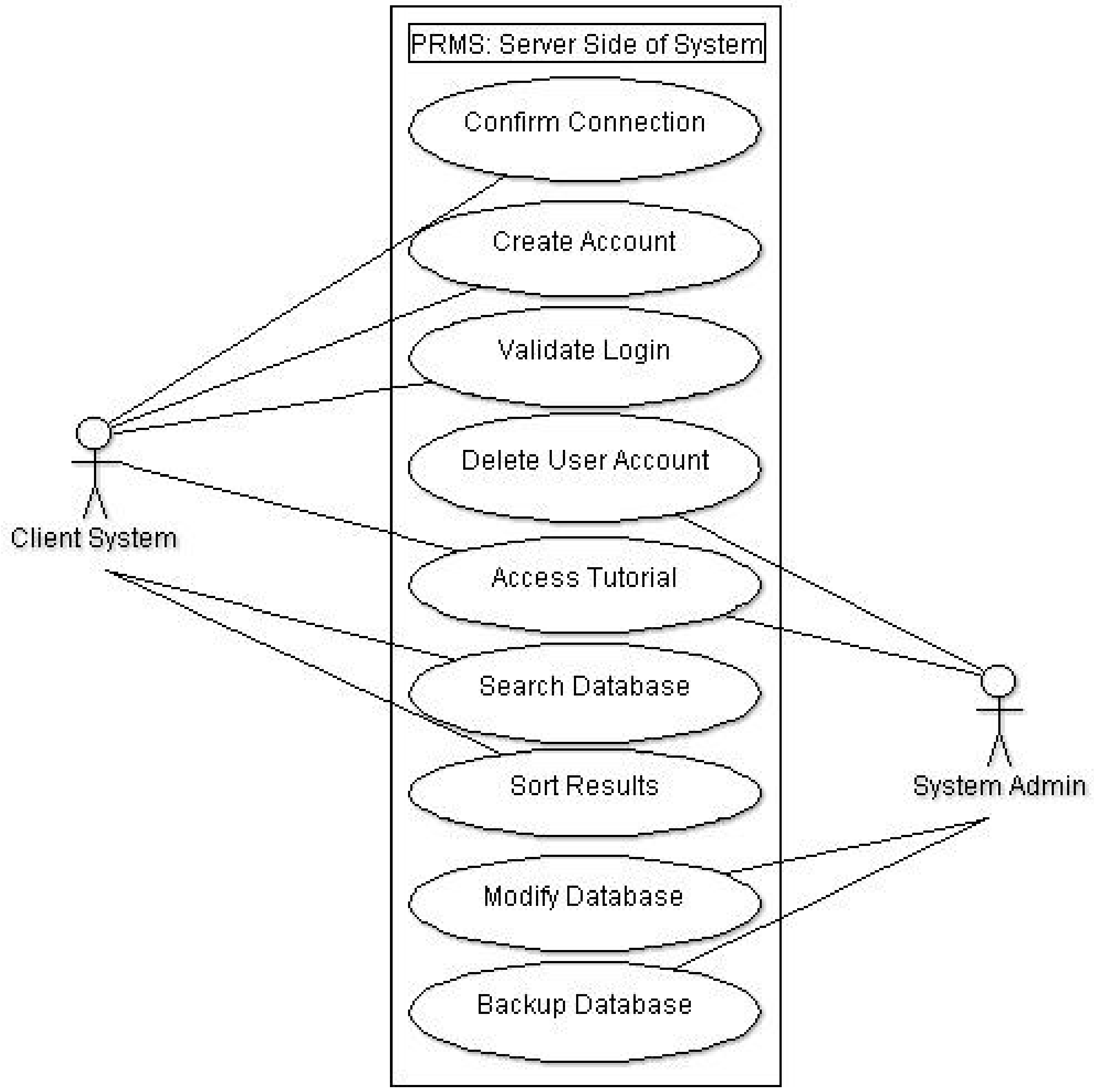
3.1 UML Client-Side Functional Model



3.1.1 Client-Side Use Case Description

The step-by-step (detailed) descriptions for the above client-side use case diagram can be referred to within the PRMS Specifications Document (SRD) referenced in section 1.3.2.

3.2 UML Server-Side Functional Model



3.2.1 Server-Side Use Case Description

The step-by-step (detailed) descriptions for the above server-side use case diagram can be referred to within the PRMS Specifications Document (SRD) referenced in section 1.3.2.

4 Design Overview

4.1 Introduction

The design approach being used for the PRMS system is object-oriented design (OOD). The system will be of a client-server architecture with the ability to simultaneously service multiple users seated across multiple client machines connected to at the very minimum, one server. This chapter will provide you an overview of how the design of this software will be implemented.

4.2 System Architecture

Below are an overview of definitions of the UML class diagrams displayed in section 5.1.

4.2.1 Client Side Classes

Class	Definition
GUI	● connects the client to the various pages of the system ● without the GUI, none of the pages described below (Login, Search, Result) would exist
Login Page	● receives user login information: username and password ● requests validation of authentication from the server ● request to create account is made in the absence of validation
Search Page	● receives search/filter parameters from the client: document name, document type, subject, publication year, author ● requests all resources relevant to the search/filter parameters from database stored on server
Result Page	● displays data acquired from the server ● request to sort by a single parameter: document name, document type, subject, publication year, or author, is sent to server and sorted data is again displayed
Client Connection Controller	● connects to server by IP address and port number ● sends requests and receives data to and from the server

4.2.2 Server Side Classes

Class	Definition
Server Connection Controller	● connects to client by IP address and port number ● receives requests and sends data to and from the client
Account	● contains a list of user/admin accounts ● validates login requests and creates new accounts
Admin	● deletes user accounts ● modifies (create/edit/delete) data within the database ● creates backups of current database
Database Controller	● gets all data that need to be filtered/sorted and sends to client ● without the Database Controller, neither the Search Controller nor the Sort Controller would exist.
Database	● contains all data/resources for the PRMS system
Sort Controller	● applies sorting algorithm to sort data using one of the following parameters: document type, subject, publication year, or author ● received from user and sorted data sent back to client
Search Controller	● sends request to grab all the data from database ● applies search/filtering algorithm using parameters: document name, document type, subject, publication year, and author ● acquired data sent back to the client

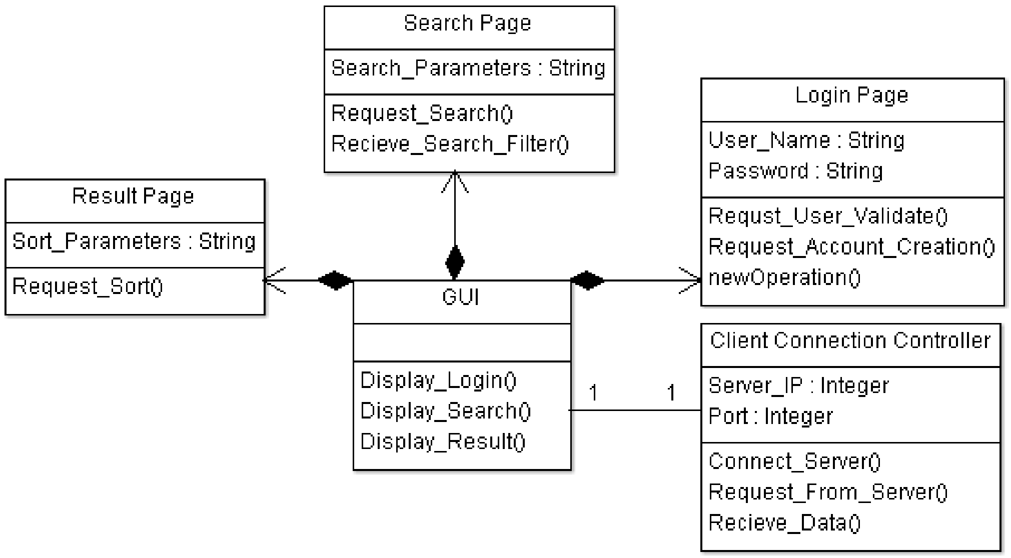
4.3 System Interface Description

The user will interface with only the client-side machine of the client-server PRMS software. The software will consist of a user-friendly, Graphical User Interface (GUI) as pictured in section 8. The server-side of the software will not be visible nor accessible to the general user. Given, the user is a system admin, access to the server-side will be provided via client machine.

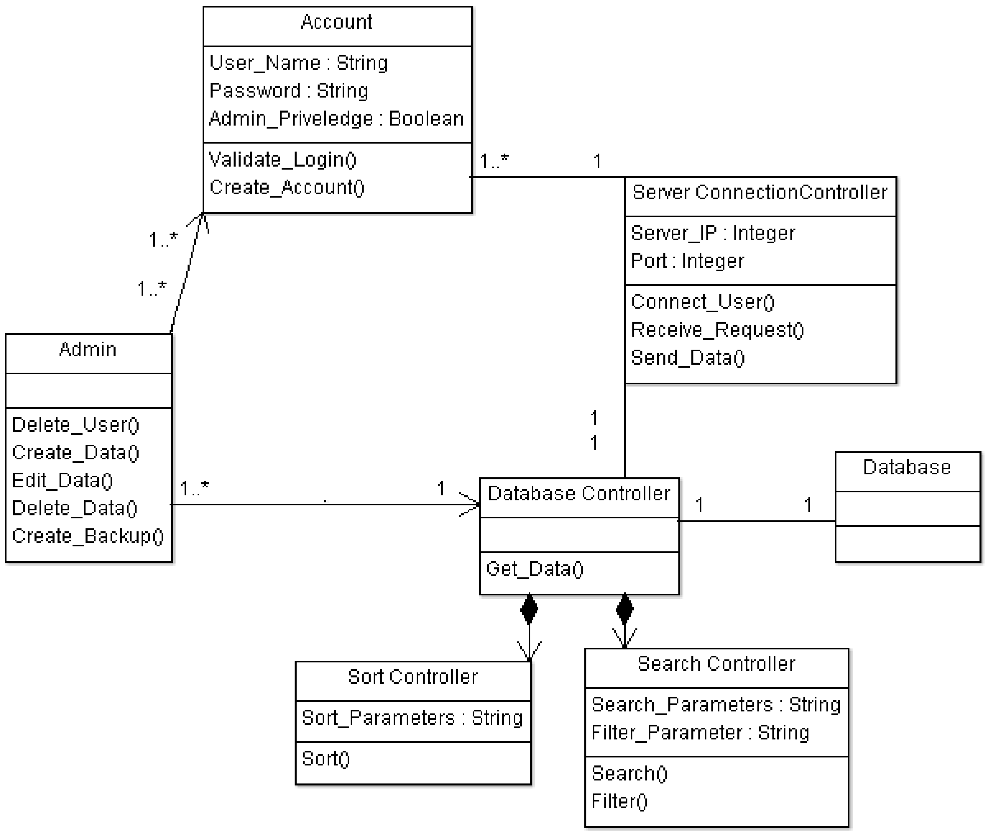
For the GUI, the user begins by requesting a connection to a server by inputting the server IP and port number. Once a connection is established, the user will be required to login/create account where their information will be stored and validated. Upon validation, access to the search page will be granted where the user can search/filter for required resources followed by the ability to sort acquired results for a better user experience. If the user account is labelled admin, extended access to the system will be granted, where the admin may alter settings and backend systems.

5 Object Model

5.1 Client-side Object Model



5.2 Server-side Object Model



6 Object Descriptions

6.1 Objects in the Client-side System

6.1.1 Object: GUI

Purpose: *To model as the gateway to all other classes and conduct all interaction with user*
Constraints: *none*
Persistent: *none*

6.1.1.1 Methods Descriptions

1. Method: *Display_Login( )*
   Return Type: *void*
   Parameters: *none*
   Return value: *none*
   Pre-condition: *The user is not connected to the server at this point*
   Post-condition: *The user has connected to the server at this point*
   Attributes read/used: *none*
   Methods called: *none*
   Processing logic: *The Client Connection Controller method Connect_Server must be invoked by entering the correct Server_IP and Server_Port. The Server Connection Controller which was in Listening Mode accepts then the connection and takes the user to the Login Page through this Display_Login method.*

2. Method: *Display_Search( )*
   Return Type: *void*
   Parameters: *none*
   Return value: *none*
   Pre-condition: *The user has connected to the server and entered valid User_Name and Password in the Login Page and then validate to log into the server*
   Post-condition: *The user has been redirected to the Search Page*
   Attributes read/used: *none*
   Methods called: *Validate_Login*
   Processing logic: *The user enters valid User_Name and Password in the Login Page, passing through the Client Connection Controller to the Server Connection Controller, which goes to the Account Object and call the method Validate_Login. After the success validation of the user, the Client Connection Controller calls the method Display_Search, taking the client to the Search Page.*

3. Method: *Display_Result( )*
   Return Type: *void*
   Parameters: *none*
   Return value: *none*

Pre-condition: *The user has been logged into the system and has performed a request for search*
Post-condition: *The Result Page is shown with the sources related to the search done*
Attributes read/used: *none*
Methods called: *Search*
Processing logic: *The user enters valid keywords for the search in the Search Page, passing through the Client Connection Controller to the Server Connection Controller, having through the Search Controller the Search method invoked. After getting the sources that matches with the keywords provided by the user, the method Display_Result takes the user to the Result Page displaying all the results which are related to the requested search.*

6.1.2 Object: Client Connection Controller

Purpose: *To establish the connection with the server of the software*
Constraints: *It should be a Boolean return of connection failure or success.*
Persistent: *Connection remains established until client request for disconnect acknowledged*

6.1.2.1 Attribute Descriptions

1. Attribute: *Server_IP*
   Type: *integer*
   Description: *Stores the IP address that the server is located on*
   Constraints: *The IP address should be exact*

2. Attribute: Server_*Port*
   Type: *integer*
   Description: *Stores the port number required to connect to the server*
   Constraints: *The port number should be exact*

6.1.2.2 Methods Descriptions

1. Method: *Connect_Server( )*
   Return Type: *Boolean*
   Parameters: *Server_IP and Server_Port are required to request connection to the server*
   Return value: *Success or failure*
   Pre-condition: *Boolean are set as 0, meaning that it is still not connected to the server*
   Post-condition: *Boolean are set as 1, meaning that it is now connected to the server*
   Attributes read/used: *Server_IP and Server_Port*
   Methods called: *Display_Login*
   Processing logic: *After receiving the Server_IP and the Server_Port to request the connection to the server, the Client Connection Controller contacts the server which is reading and waiting for connection requests. If the connection succeeds the Display_Login() method is called.*

2. Method: *Request_From_Server( )*
   Return Type: *Boolean*
   Parameters: *Login credentials; Search parameters; Sort parameters.*
   Return value: *Success or failure*
   Pre-condition: *Client has established connection with a server*
   Post-condition: *Server acknowledges and acts on the received request*
   Attributes read/used: *User_Name, Password, Search_Parameters, Sort_Parameters.*
   Methods called: *Display_Login ; Display_Search ; Display_Result*
   Processing logic: *As soon as the Login Page is displayed, the user enters the values User_Name and Password to require access to the Search Page. If the request succeeds the user is then taken to the Search Page through the method Display_Search. Once the user is in the Search Page, it is possible to enter the Search_Parameters and request the server to search into the database. The success of this operation calls the method Display_Result, taking the user to the Result Page.*

3. Method: *Receive_Data( )*
   Return Type: *Boolean*
   Parameters: *none*
   Return value: *Success or failure*
   Pre-condition: *Client Connection Controller has sent request to server*
   Post-condition: *User request is acknowledged and user is redirected to another page (Login, Search, or Result Page)*
   Attributes read/used: *based on request sent*
   Methods called: *based on request sent*
   Processing logic: *As the Client Connection Controller calls the method Request_From_Server, the server provides the computations of what has been requested. After the request is acknowledged and processed in the Server Connection Controller, the Server sends the data of the result to the Client Connection Controller, which Receive_Data(), and redirects to the required page: Login, Search or Result..*

6.1.3 Object: Login Page

Purpose: *To interact with the user, attaining their username and password to log them on system*
Constraints: *User must have a pre-registered account to login*
Persistent: *User account stays logged in until disconnect requested or 15-minute timeout*

6.1.3.1 Attribute Descriptions

1. Attribute: *User_Name*
   Type: *string*
   Description: *Stores the string name that is associated to the user*
   Constraints: *Must have a valid 'username@lakeheadu.ca' email*

2. Attribute: *Password*
   Type: *string*

Description: *Stores the password associated to the user's account*
Constraints: *Must be a minimum of four digits; alphanumeric*

6.1.3.2 Methods Descriptions

1. Method: *Request_User_Validate( )*
   Return Type: *Boolean*
   Parameters: *User_Name ; Password*
   Return value: *Success or failure*
   Pre-condition: *Client system is connected to a server and user has requested log in*
   Post-condition: *User has been correctly validated and is redirected to the Search Page*
   Attributes read/used: *User_Name ; Password*
   Methods called: *Display_Search( )*
   Processing logic: *As soon as the user gets to the Login Page, it is required (assuming they are already registered on the system) to enter a valid User_Name and Password to be granted access into the system.*

2. Method: *Request_Account_Creation( )*
   Return Type: *Boolean*
   Parameters: *User_Name ; Password (both user input)*
   Return value: *Success or failure*
   Pre-condition: *User at Login Page but is not registered on PRMS system*
   Post-condition: *User successfully creates an account on the system*
   Attributes read/used: *User_Name ; Password*
   Methods called: *Display_Search()*
   Processing logic: *If a user has not registered on the PRMS system, they may create an account from the Login Page to be able to access the server by entering a valid User_Name and a Password.*

6.1.4 Object: Search Page

Purpose: *To request the server to search the database*
Constraints: *none*
Persistent: *User may search an unlimited number of times*

6.1.4.1 Attribute Descriptions

1. Attribute: *Search_Parameters*
   Type: *string*
   Description: *Stores the search parameters (doc name, doc type, subject, etc.) temporarily*
   Constraints: *none*

6.1.4.2 Methods Descriptions

2. Method: *Request_Search( )*
   Return Type: *Boolean*

Parameters: *Search_Parameters*
Return value: *Success or failure*
Pre-condition: *User is on Search Page and enters valid Search_Parameters*
Post-condition: *Valid search takes user to the Result Page displaying the acquired results*
Attributes read/used: *Search_Parameters*
Methods called: *Display_Result ; Receive_Data*
Processing logic: *The user inputs valid Search_Parameters and clicks on the Search button. At this point, the SearchController is invoked and starts to compute the search that has been requested. Upon acquiring the search results requested via the DatabaseController, it sends a message to the GUI indicating a successful search and then forwarding the user to the Result Page.*

6.1.5 Object: Result Page

Purpose: *To display the result of the search that the client has requested*
Constraints: *The search must have been done correctly*
Persistent: *The user is able to sort the results any way they prefer*

6.1.5.1 Attribute Descriptions

1. Attribute: *Sort_Parameters*
   Type: *string*
   Description: *Sort the results displayed in the result page*
   Constraints: *It must have at least two different sources as result*

6.1.5.2 Methods Descriptions

1. Method: *Request_Sort( )*
   Return Type: *void*
   Parameters: *Sort_Parameters*
   Return value: *none*
   Pre-condition: *The user must already be on the Result Page*
   Post-condition: *Result Page is reformulated being sorted according to the user request*
   Attributes read/used: *Sort_Parameters*
   Methods called: *Display_Result*
   Processing logic: *The Request_Sort method is called by the user when the Result Page is shown and the user prefers to sort the acquired results for accuracy or efficiency.*

6.2 Objects in the Server-side System

6.2.1 Object: Server Connection Controller

Purpose: *To wait for a user to request a connection and establish connection upon request*
Constraints: *none*
Persistent: *The Server Connection Controller stays in "Listening Mode" until it is set down*

6.2.1.1 Attribute Descriptions

1. Attribute: *Server_IP*
   Type: *integer*
   Description: *Stores the IP address that the server is located on*
   Constraints: *The IP address should be valid*

2. Attribute: *Server_Port*
   Type: *integer*
   Description: *Stores the port number required to connect to the server*
   Constraints: *The port number should be valid*

6.2.1.2 Methods Descriptions

1. Method: *Connect_User( )*
   Return Type: *Boolean*
   Parameters: *Server_IP ; Server_Port*
   Return value: *Success or failure*
   Pre-condition: *The server is in "Listening Mode"*
   Post-condition: *Client Connection Controller requested and established connection to server*
   Attributes read/used: *Server_IP ; Server_Port*
   Methods called: *Display_Login*
   Processing logic: *The Server Connection Controller which is in Listening Mode receives the request to connect from the user and connects the user, taking the user to the Login Page by calling the method from the GUI Display_Login.*

2. Method: *Receive_Request( )*
   Return Type: *Boolean*
   Parameters: *User_Name ; Password ; Search_Parameters ; Sort_Parameters*
   Return value: *Success or failure*
   Pre-condition: *User must have been connected to the server and be at one of the three valid pages: Login Page, Search Page or Result Page*
   Post-condition: *The action requested by the user is processed by the server and if it succeeded the user is redirected to the page associated to the result of the action requested*
   Attributes read/used: *User_Name ; Password ; Search_Parameters ; Sort_Parameters*
   Methods called: *Display_Search ; Display_Result ; Search ; Filter ; Sort ; Get_Data ;*
   Processing logic: *As the client is connected to the server, it is the time that the client send requests to the server. The user can request to log into the server by providing valid User_Name and Password. Also, the user which is already logged on can request for Search by inputting keywords and clicking on the Search button. Finally, the user can also request for the Sorting process as soon as the client is located in the Result Page.*

3. Method: *Send_Data( )*
   Return Type: *void*
   Parameters: *none*
   Return value: *none*
   Pre-condition: *The user is connected to the server and request to receive data from server*
   Post-condition: *The data has been sent to the user*
   Attributes read/used: *none*
   Methods called: *none*
   Processing logic: *The user send the request to the server to receive data from the database of the server. The server analyzes the data that the user has requested. If it is approved the sending of the data, the server then sends to the user the data.*

6.2.2 Object: Account

Purpose: *To identify the users that will be allowed to log into the system*
Constraints: *the User_Name must end with a -@lakeheadu.ca*
Persistent: *none*

6.2.2.1 Attribute Descriptions

1. Attribute: *User_Name*
   Type: *string*
   Description: *Stores the name that the user will use to access the server*
   Constraints: *This attribute must end as @lakeheadu.ca*
2. Attribute: *Password*
   Type: *string*
   Description: *Stores the password associated to the user's account*
   Constraints: *none*
3. Attribute: *Admin_Priviledge*
   Type: *Boolean*
   Description: *Stores if the user that is logging on has administrator's privileges*
   Constraints: *only one account of the database is allowed to have this attribute as valid*

6.2.2.2 Methods Descriptions

1. Method: *Create_Account( )*
   Return Type: *Boolean*
   Parameters: *User_Name ; Password*
   Return value: *Success or failure*
   Pre-condition: *The user has connected to the server but does not have an account to log on to the server - it is possible then to create an account*
   Post-condition: *If the constraint of User_Name has been satisfied, then the account is successfully created*
   Attributes read/used: *User_Name ; Password*
   Methods called: *none*

Processing logic: *After the client gets the connection with the server, the client is taken to the Login Page. As the user at this point does not have an account registered yet, it is possible to request to the server to create an account to log on the system.*

2. Method: Validate_Account( )
   Return Type: *Boolean*
   Parameters: *User_Name ; Password*
   Return value: *Success or failure*
   Pre-condition: *The user has been connected to the server and already has a registered account to log on the system*
   Post-condition: *If successfully validated, the user is then taken to the Search Page*
   Attributes read/used: *User_Name ; Password ; Admin_Priviledge*
   Methods called: *Display_Search()*
   Processing logic: *After the client gets the connection with the server, the client is taken to the Login Page. As the user at this point already have a registered account to access the server, the user enters the User_Name and Password information that the server shall check to validate as a registered user. If the inputs are validated, the server checks the Admin_Priviledge and takes the user to the Search Page, allowing privilege functions if the user is identified as the administrator of the server.*

6.2.3 Object: Administrator (Admin)

Purpose: *To administrate the system and perform updates whenever the system needs*
Constraints: *The Boolean attribute Admin_Priviledge must be equal to 1 (true)*
Persistent: *none*

6.2.3.1 Methods Descriptions

1. Method: *Delete_User( )*
   Return Type: *void*
   Parameters: *none*
   Return value: *none*
   Pre-condition: *The admin has logged into the system and accessed the list of users that are registered into the system*
   Post-condition: *The admin deletes a user that should not be allowed to log on the system*
   Attributes read/used: *none*
   Methods called: *none*
   Processing logic: *The administrator logs onto the system and click to check the users that are registered to log on the system. As soon as the administrator finds any user that has been registered incorrectly (not a valid @lakeheadu.ca e-mail, for example) the administrator deletes the user from the server, blocking his access to the server.*

2. Method: *Create_Data( )*
   Return Type: *void*

Parameters: *none*
Return value: *none*
Pre-condition: *The administrator has logged into the system and accessed the database*
Post-condition: *The administrator has added new sources in the database*
Attributes read/used: *none*
Methods called: *none*
Processing logic: *The administrator logs onto the system and click to check the database. Then the admin clicks to add new sources to the database. The data is inputted correctly and then added to the database as a new source.*

3. Method: *Edit_Data( )*
Return Type: *void*
Parameters: *none*
Return value: *none*
Pre-condition: *The administrator has logged into the system and accessed the database*
Post-condition: *The administrator has edited a source in the database*
Attributes read/used: *none*
Methods called: *none*
Processing logic: *The administrator logs into the system and click to check the database. Then the admin selects a source that needs to be updated and click to edit it. The admin changes what it was in need to be changed and finish the task.*

4. Method: *Delete_Data( )*
Return Type: *void*
Parameters: *none*
Return value: *none*
Pre-condition: *The administrator has logged into the system and accessed the database*
Post-condition: *The administrator has deleted a source from the database*
Attributes read/used: *none*
Methods called: *none*
Processing logic: *The administrator logs into the system and click to check the database. The admin then selects and deletes a source that needs to be deleted from the database.*

5. Method: *Create_Backup( )*
Return Type: *File*
Parameters: *none*
Return value: *none*
Pre-condition: *The administrator has logged into the system and accessed the database*
Post-condition: *The administrator has received the backup file from the database*
Attributes read/used: *none*
Methods called: *none*

Processing logic: *The administrator logs into the system and click to check the database. Then the admin clicks to receive a backup file from the database as security measure. The server outputs a file with all the sources in the database.*

6.2.4 Object: Database Controller

Purpose: *To control the database, making the software more consistent*
Constraints: *none*
Persistent: *none*

6.2.4.1 Methods Descriptions

1. Method: *Get_Data( )*
   Return Type: *void*
   Parameters: *none*
   Return value: *none*
   Pre-condition: *The user perform a search, filter or sort request through the Request_From_Server method, and the Server Connection Controller request data from the database to the Database Controller*
   Post-condition: *The Database Controller has performed the request and sent the data requested to the Server Connection Controller*
   Attributes read/used: *Search_Parameters; Sort_Parameters; Display_Result*
   Methods called: *Search; Filter; Sort*
   Processing logic: *The user send a request to the server through the Request_From_Server method, invoking the Client Connection Controller, which invokes the Server Connection Controller. The Server Connection Controller receives the request from the client and send the request to the Database Controller. The Database Controller perform the action and send the data requested to the Server Connection Controller, doing the reverse way of information, providing to the client the request in the Result Page.*

6.2.5 Object: Search Controller

Purpose: To Search in the database properly
Constraints: none
Persistent: none

6.2.5.1 Attribute Descriptions

1. Attribute: *Search_Parameters*
   Type: *string*
   Description: *Stores the parameters of the search that must be done in the database*
   Constraints: *none*

2. Attribute: *Filter_Parameter*
   Type: *string*
   Description: *Describes the subject of which the database controller must filter the search*

Constraints: *must be a valid subject: document name; document type; subject; year of publication; author*

6.2.5.2 Methods Descriptions

1. Method: *Search( )*
   Return Type: *void*
   Parameters: *Search_Parameters*
   Return value: *none*
   Pre-condition: *The client has invoked the Request_From_Server method by requesting a search in the Search Page*
   Post-condition: *The Database Controller finishes the Search process and takes the user to the Result Page*
   Attributes read/used: *Search_Parameters*
   Methods called: *Display_Result*
   Processing logic: *The user send a request to the server through the Request_From_Server method, invoking the Client Connection Controller, which invokes the Server Connection Controller. The Server Connection Controller receives the request from the client and send the request to the Database Controller. The Database Controller perform the Search and send the data requested to the Server Connection Controller, doing the reverse way of information, providing to the client the request in the Result Page.*

2. Method: *Filter( )*
   Return Type: *void*
   Parameters: *Filter_Parameter*
   Return value: *none*
   Pre-condition: *The Search method must have been invoked and the main search must have finished. The Filter must select the sources from the main search result*
   Post-condition: *The Result Page is shown after the filtering have occurred*
   Attributes read/used: *Filter_Parameter*
   Methods called: *Display_Result*
   Processing logic: *The user send a request to the server through the Request_From_Server method, invoking the Client Connection Controller, which invokes the Server Connection Controller. The Server Connection Controller receives the request from the client and send the request to the Database Controller. The Database Controller performs the Search and before sending the data requested to the Server Connection Controller, doing the reverse way of information, it filters the results found according to the Filter_Parameter. After the filtering process, the Result Page is then shown to the user.*

6.2.6 Object: Sort Controller

Purpose: *To Sort the Results that have been displayed to the user*
Constraints: *Must be sorted by one of the subjects: Document Name; Document Type; Subject; Publication Year; Author*
Persistent: *none*

6.2.6.1 Attribute Descriptions

1. Attribute: *Sort_Parameter*
   Type: *string*
   Description: *Stores the subject that the user requested to sort the results like*
   Constraints: Must be equal to one of the subjects: Document Name; Document Type; Subject; Publication Year; Author

6.2.6.2 Methods Descriptions

1. Method: *Sort( )*
   Return Type: *void*
   Parameters: *Sort_Parameter*
   Return value: *none*
   Pre-condition: *The Result Page must have been showed to the user already*
   Post-condition: *Result Page is reformulated after the sorting method has been applied*
   Attributes read/used: *Sort_Parameter*
   Methods called: *Display_Result*
   Processing logic: *The Result Page is shown to the user after a Search Request. At this point, the results are not sorted alphabetically, which would make easier to the client to find the source which is being searched. The Sort method is then invoked by the user to provide an alphabetically display of the results.*

6.2.7 Object: Database

Purpose: *To store the sources of the server*
Constraints: *It must be in the correct order of Document Name; Document Type; Subject; Publication Year; Author. Must have only these subjects*
Persistent: *none*

7 Dynamic Model

7.1 Scenarios

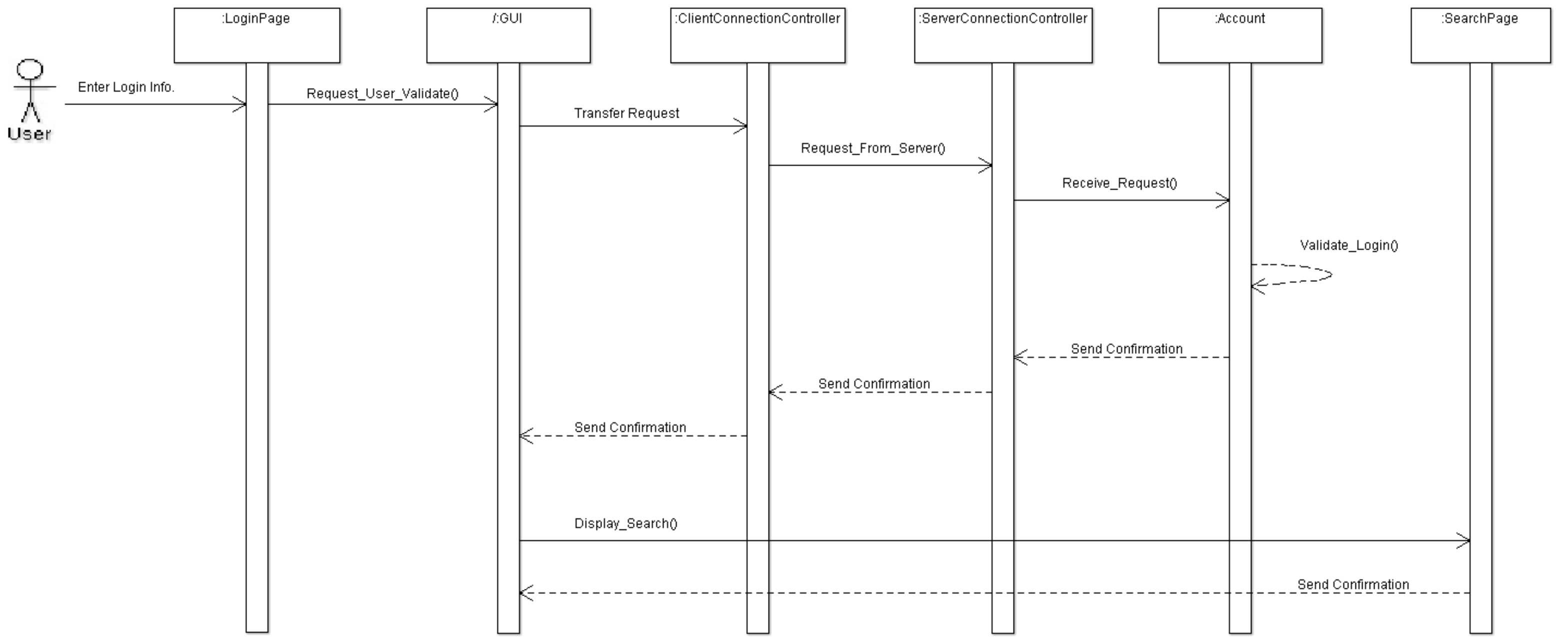
This section models how the system responds when faced with a given scenario.

7.1.1 Scenario: User Login Flow

Description (assuming user is at the Login Page already):

1. User inputs their login information in Login Page on client system
2. Login Page sends the GUI a request to validate/authenticate user
3. GUI transfers the request to Client Connection Controller
4. Client Connection Control accesses the server via Server Connection Controller
5. Server Connection Controller then forwards the request to Account where the user information is validated/authenticated
6. After validating, Account sends a confirmation to the Server Connection Controller which forwards it to Client Connection Controller after which the user is shown GUI
7. Once validated the GUI accesses the Search Page to be displayed to the user

Sequence diagram:

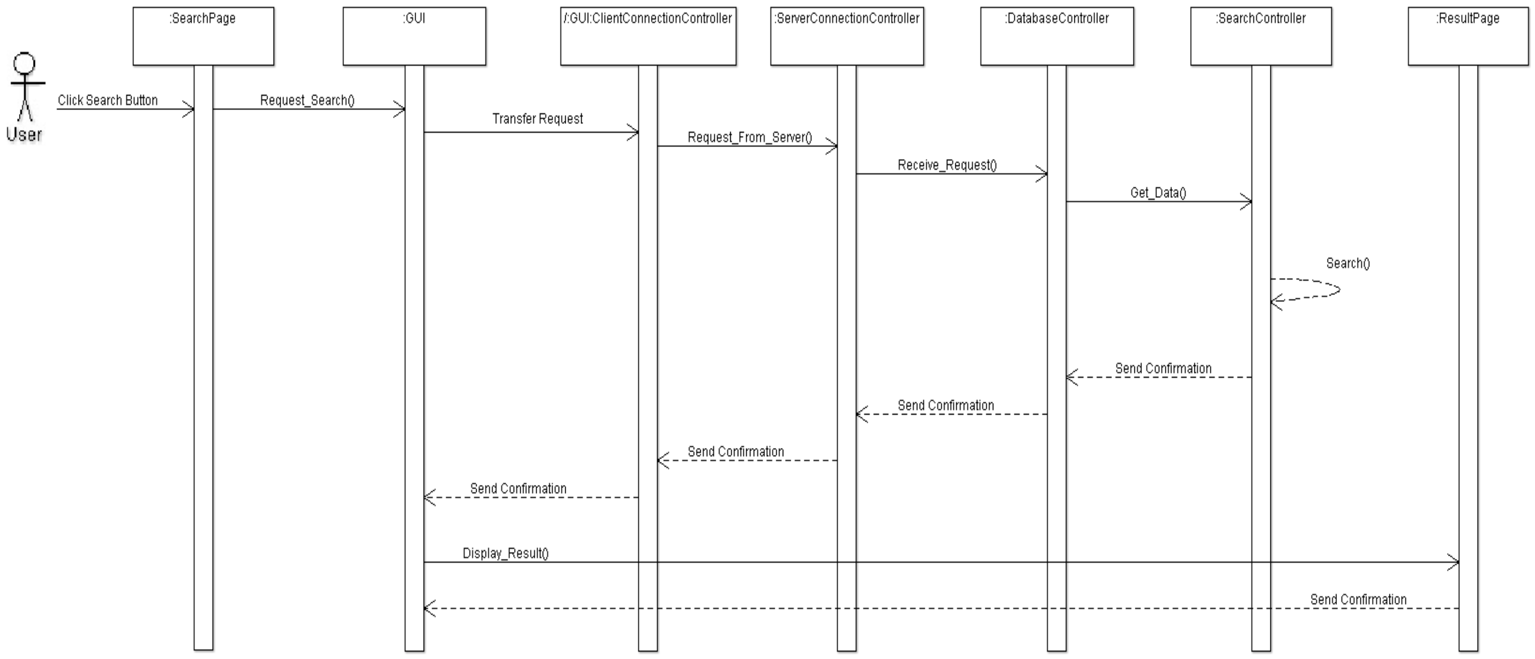


7.1.2 Scenario: User Search Flow

Description (assuming user is at the Search Page already):

1. User inputs search parameters and clicks search
2. Search Page sends the GUI a request to search
3. GUI transfers the request to Client Connection Controller
4. Client Connection Control accesses the server via Server Connection Controller
5. Server Connection Controller then forwards the request to Database Controller
6. Database Controller sends request to Search Controller where it searches the database
7. After searching, Search Controller sends data to the Server Connection Controller which forwards it to Client Connection Controller after which the user is shown GUI
8. Once searched, the GUI accesses the Result Page to be displayed to the user

Sequence diagram:

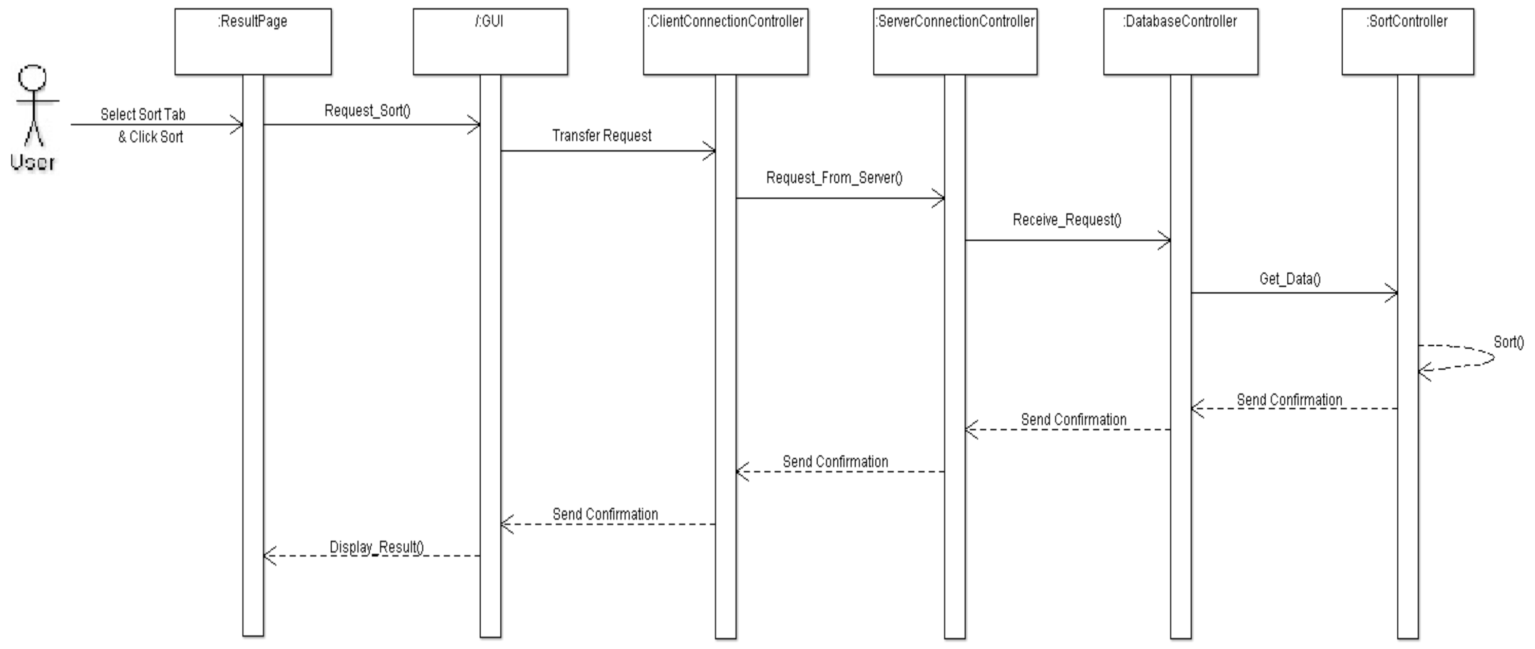


7.1.3 Scenario: User Sort Flow

Description (assuming user is at the Result Page already):

1. User inputs sort parameters and clicks sort
2. Result Page sends the GUI a request to sort
3. GUI transfers the request to Client Connection Controller
4. Client Connection Control accesses the server via Server Connection Controller
5. Server Connection Controller then forwards the request to Database controller
6. Database controller send request to Sort controller where it sorts the data
7. After searching, Sort Controller sends data to the Server Connection Controller which forwards it to Client Connection Controller after which the user is shown GUI
8. Once sorted, the GUI accesses the Result Page to be displayed to the user

Sequence diagram:



8 Pseudocode (PDL)

8.1 Connection to Server PDL

The pseudocode below describes when the client system requests a connection to a server.

```
connect_to_server(int ip_address, int port_number){
        IF server_ip == ip_address AND server_port == port_number THEN
                REPEAT
                        connect_to_client();
                UNTIL client_connected == TRUE;
        ENDIF
}
```

8.2 Login PDL

The pseudocode below describes a user's attempt to log into PRMS on the client system.

```
login (string username, string password) {
        FOR each account in account database
           IF selected_account_username == username
              AND selected_account_password == password THEN
                REPEAT
                    validate_account();
                UNTIL successful_login==TRUE;
           ENDIF
        ENDFOR
}
```

8.3 Search and Filter PDL

The pseudocode below describes the client-side using the search and filter functionality of PRMS. NOTE: assume struct search_output is globally declared and that it contains all data types as the parameters for search.

```
search(string doc_name,string doc_type,string subject,string,int pub_year,string author) {
        vector<search_output> search_data;
        search_data=filter(doc_name,doc_type,subject,pub_year,author);
        RETURN search_data;
}

filter(string doc_name,string doc_type,string subject,string,int pub_year,string author){
        vector<search_output>  temp_data;
        vector<search_output> filtered_data;
        int index_of_database = 0;
        FOR each data set in database
                temp_data.pushback = Get_Data(index_of_database)
                INCREMENT index_of_database;
        ENDFOR
        FOR each data set in temp_data
                IF doc_name == temp_data[index_in_for].name AND
                        doc_type== temp_data[index_in_for].type AND
                        subject == temp_data[index_in_for].subject AND
                        pub_year== temp_data[index_in_for].year AND
                        author== temp_data[index_in_for].author
                THEN
                        filtered_data.pushback(temp_data[index_in_for]);
                ENDIF
        ENDFOR
        RETURN filtered_data
}
```

8.4 Modify Database (Admin) PDL

The pseudocode below describes the system admin (assuming they're already logged in and validated on client system) accessing the database on the server for modifications. Modifications include: create, update and delete.

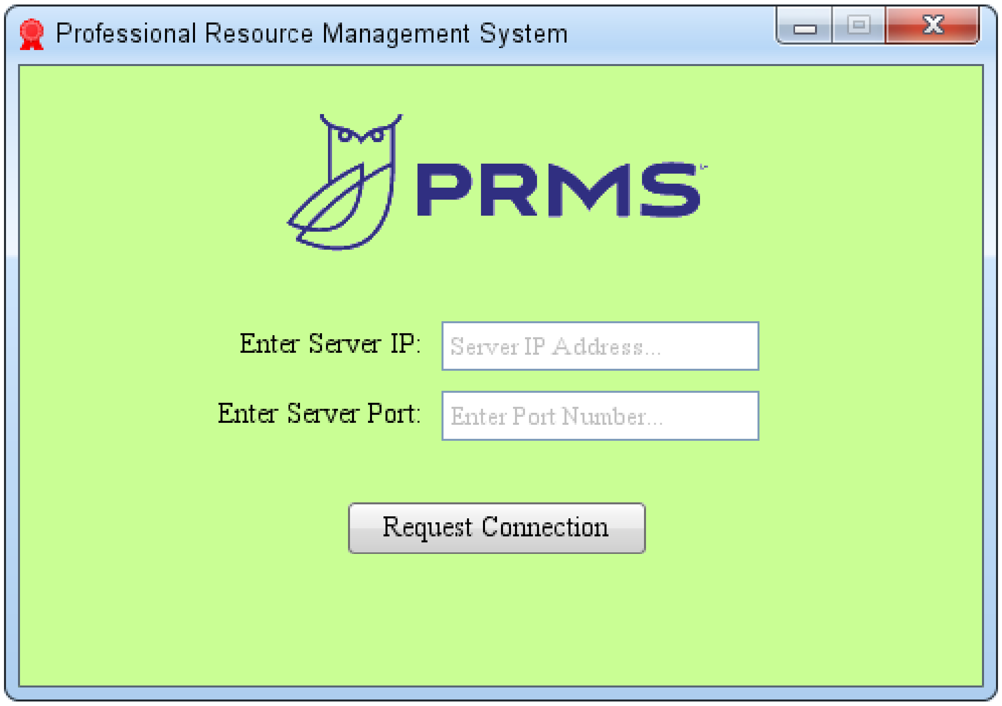
```
create(string doc_name,string doc_type,string subject,string,int pub_year,string author) {
vector<search_output>  temp_data;
search_output  added_data;
int index_of_database = 0;
        FOR each data set in database
                temp_data.pushback = Get_Data(index_of_database)
                INCREMENT index_of_database;
        ENDFOR
        added_data[index].name = doc_name;
        added_data[index].type = doc_type;
        added_data[index].subject = subject;
        added_data[index].year = pub_year;
        added_data[index].author = author;
        temp_data.pushback(added_data)
        save_database(temp_data);
}
```

```
update(int index,string doc_name,string doc_type,string subject,string,int pub_year,string author) {
vector<search_output>  temp_data;
int index_of_database = 0;
```

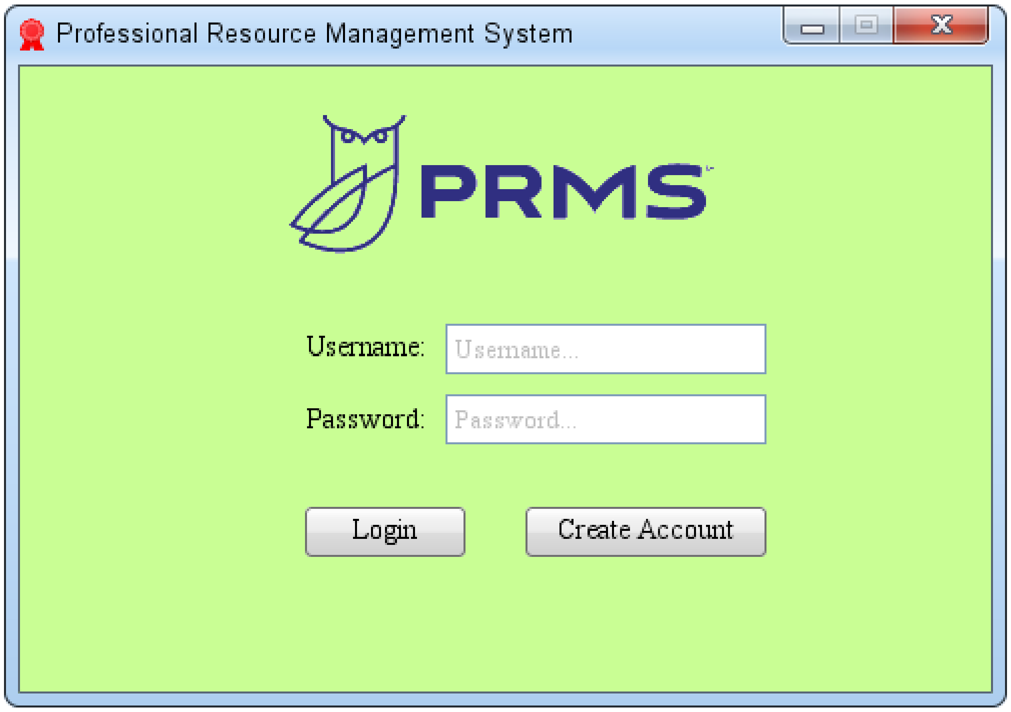
```
        FOR each data set in database
                temp_data.pushback = Get_Data(index_of_database)
                INCREMENT index_of_database;
        ENDFOR
        temp_data[index].name = doc_name;
        temp_data[index].type = doc_type;
        temp_data[index].subject = subject;
        temp_data[index].year = pub_year;
        temp_data[index].author = author;
        save_database(temp_data);
}

delete(int index) {
vector<search_output>  temp_data;
int index_of_database = 0;
        FOR each data set in database
                temp_data.pushback = Get_Data(index_of_database)
                INCREMENT index_of_database;
        ENDFOR
        temp_data.erase(index);
        save_database(temp_data);
}
```

9 Updated Version of GUI

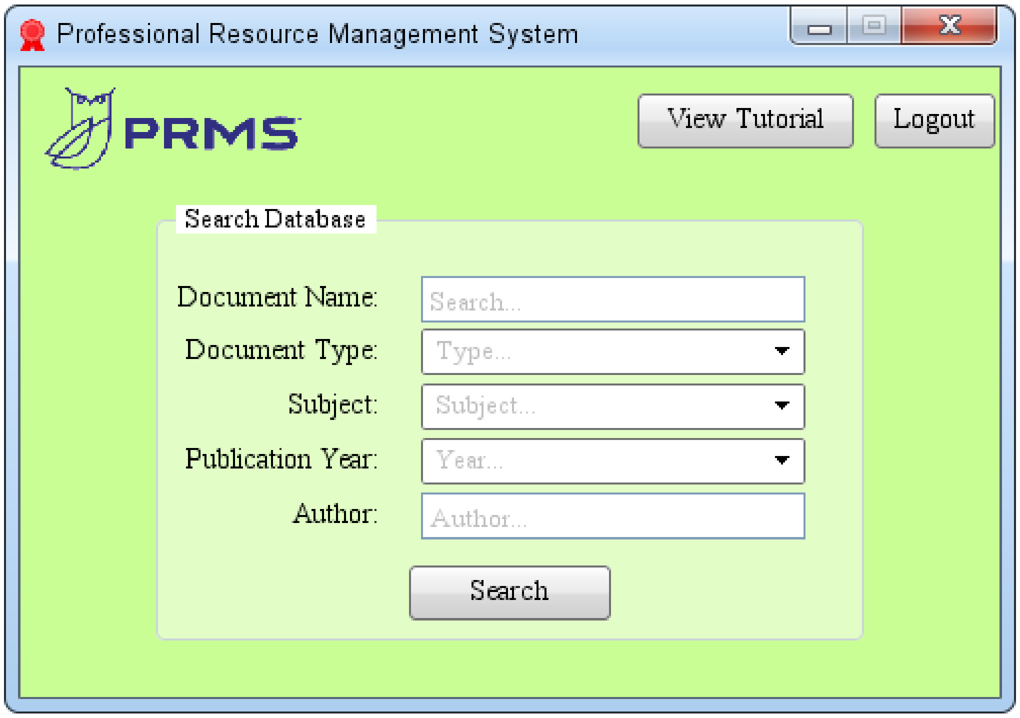
Request Server Connection Page:



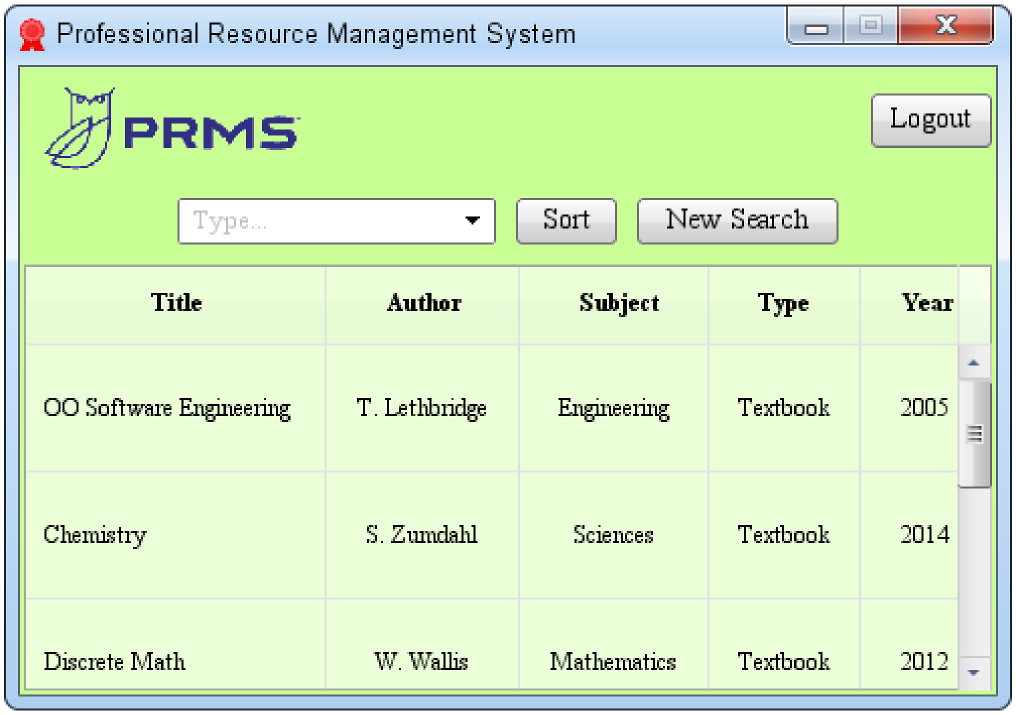
Login Page:



Default (Search) Page:



Results Page:



Title	Author	Subject	Type	Year
OO Software Engineering	T. Lethbridge	Engineering	Textbook	2005
Chemistry	S. Zumdahl	Sciences	Textbook	2014
Discrete Math	W. Wallis	Mathematics	Textbook	2012

10 Updated Project Costs

The intermediate Constructive Cost Model (COCOMO) approach has been utilized to determine the development effort. The following is the intermediate COCOMO calculation for PRMS:

Formulae:

1. Nominal Effort = A*(KLOC)^B, in person-months
2. Effort Multiplier = product of 15 ratings
3. Est. Project Effort = Nominal Effort * Effort Multiplier

Mode: Organic; therefore, **A**=**2.4**, **B**=**1.05** **|** LOC estimate: 3000 (**KLOC** = **3**)
Nominal Effort = 2.4*(3)$^{1.05}$ = **7.60656 (person-months)**

Development Effort Multipliers for PRMS:

Cost Drivers	**Rating**	
Required software reliability	Nominal	1.00
Database size	Low	0.94
Product complexity	High	1.15
Execution time constraint	High	1.11
Main storage constraint	Nominal	1.00
Virtual machine volatility	Nominal	1.00
Computer turnaround time	Nominal	1.00
Analyst capabilities	Nominal	1.00
Applications experience	Low	1.13
Programmer capability	Low	1.17
Virtual machine experience	N/A	-
Programming language experience	Nominal	1.00
Use of modern programming practices	High	0.91
Use of software tools	Nominal	1.00
Required development schedule	Very Low	1.23
	Effort Multiplier:	**1.77566**

Estimated Effort for Project = 7.60656 * 1.77566 = **13.51 person-months**

This estimation results in an approximate team member effort of:
13.51 person-months / 3 persons = **4.50 person-months per team member**

Which, given 1.0 month until project deadline, further approximates each member doing the work of: 4.50 person-months per team member / 1.0 months = **4.50 persons**.

11 Technical Documentation

For the purpose of documentation which was referred to in the making of this document, please see section 1.3 for more information.