

# Branch Library Management System Requirements Specification

Revision 2854, made 26/01/2012 by tws

September 24, 2012

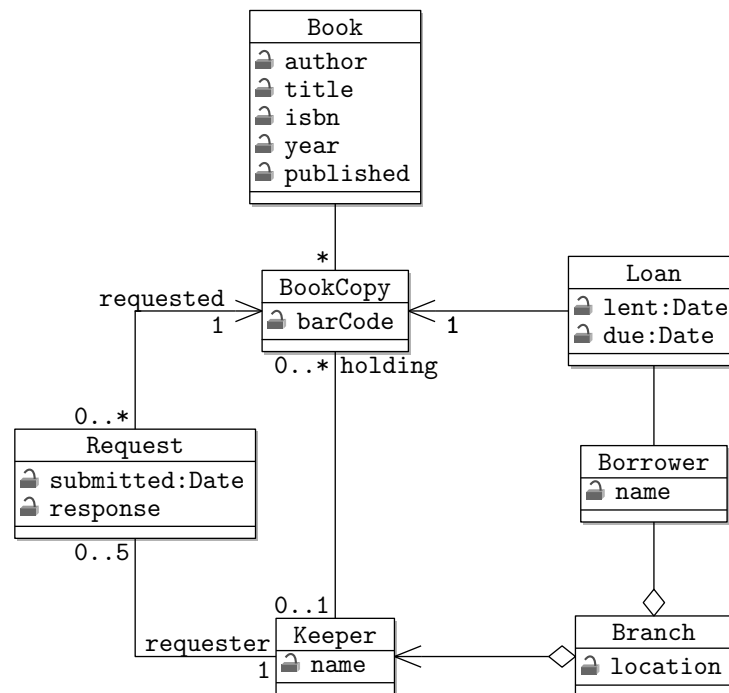
# Contents

<b>1 Problem Description</b>	<b>3</b>
<b>2 Domain Model</b>	<b>4</b>
<b>3 Actors</b>	<b>5</b>
<b>4 Use Cases</b>	<b>6</b>
4.1 Keeper Administration . . . . .	7
4.2 Transferring Books from the Central Library to the Branch . . . . .	10
4.3 Sub-Lending Books . . . . .	14
4.4 Return books to the Central Library . . . . .	18
4.5 Common shared Use Cases . . . . .	22
<b>5 Non Functional Requirements</b>	<b>28</b>

# **1 Problem Description**

A university is considering the introduction of a branch library system in its departments to support staff and students in their work. A computer based system is needed for managing the movement of books from the central library and within the branch. Each branch will receive a collection of books from the main library, typically as a result of a request from a member of staff. These will then be available to loan out to members of staff and students.

## 2 Domain Model



The model illustrates the major domain elements for the branch library system.

The distributed branch is made up of a number of keepers who will hold the books from the central library. Keepers can make requests for books to be transferred from the library, which must be processed by the librarian. Books can be lent to borrowers.

Note that this is not a class diagram of a system design, since many elements are missing, attribute types are unspecified and operations have not been identified. The diagram represents the entities in the system domain that need to be represented, rather than how the system requirements will be realised in a design.

There are several issues resulting from the development of the domain model that need to be addressed by further requirements gathering.

1. What is the life-cycle of a loan - are loans kept after books are returned (e.g. for auditing purposes), or is a loan object destroyed when the book is returned to a keeper?
2. What is the life-cycle of a request, is it disposed of once it has been processed by a librarian?

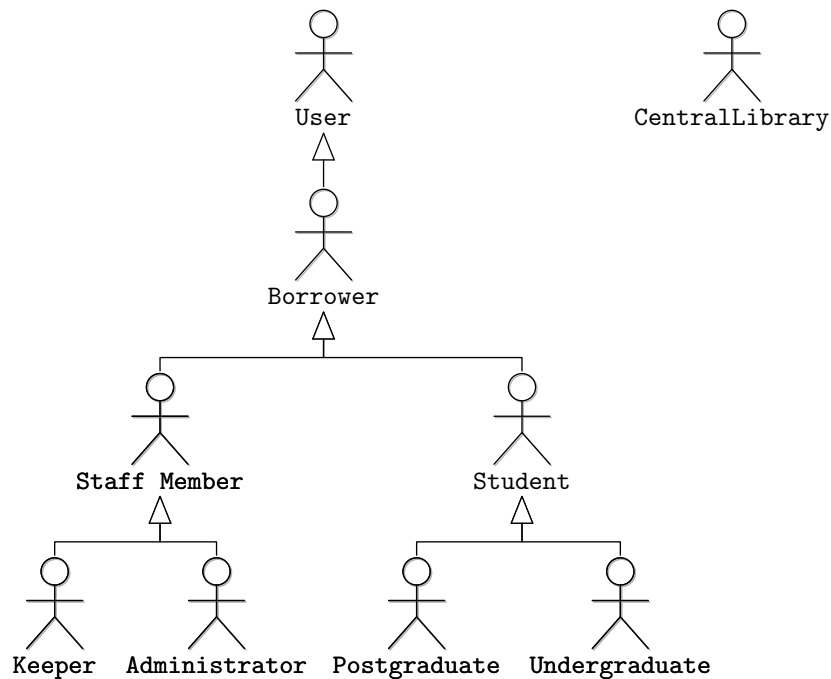


Figure 1: revised actors for the branch library system

### 3 Actors

Figure 1 illustrates the relationships between actor roles in the system. A short summary of the actors is given below:

**User** represents any actor on the system. Is able to search the library system for books and their locations

**CentralLibrary** handles the Central Library functions for distributing books to branches

**Borrower** a user who is registered as a borrower by the CentralLibrary

**Student** a type of borrower with limited borrowing rights

**Staff Member** a type of borrower with the potential to be a Keeper

**Keeper** a member of staff in the department hosting the branch, with the authority to request books from the Central Library and sub-lend books to borrowers

**Administrator** a member of staff in the department hosting the branch, appointed to manage keepers in the branch

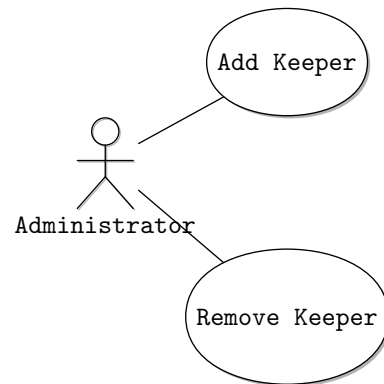
## 4 Use Cases

This section describes the required functionality for the Branch Library System as four groups of related use cases. The core use cases for the system are:

- common utility activities (Section 4.5);
  - Login
  - View Status
  - Logout
  - Search for book
  - Search for user
- keeper administration (Section 4.1);
  - Add keeper
  - Remove keeper
- transferring books to keepers in the branch from the Central Library (Section 4.2);
  - Request book for branch
  - Process request
  - Record receipt of book
- sub-lending books to borrowers by keepers (Section 4.3)
  - Record book lend
  - Request book return (keeper to borrower)
  - Record book return (Branch)
- returning books to the Central Library (Section 4.4)
  - Request book from Branch Library
  - Record book sent
  - Record book return (Central)
- some further use cases have also been identified as potentially useful. The clients do not envisage that these use cases should be implemented in the current construction phase and have not been fully described in the previous elaboration phase.
  - Transfer book between keepers
  - Record book return by borrower to the Central Library
  - Apply fine for late return.

## 4.1 Keeper Administration

### Use Case Diagram



A member of staff appointed in the department as an administrator is responsible for adding and removing members of staff as keepers of books.

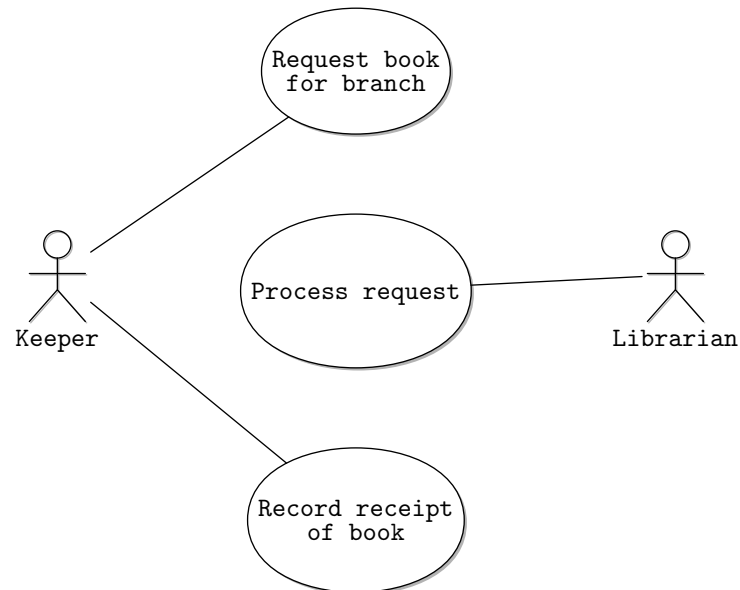
<b>Use case</b>	Add Keeper
<b>Description</b>	<pre> graph TD     Start(( )) --&gt; Login([Login])     Login --&gt; Search([Search for user])     Search --&gt; Decision1{ }     Decision1 -- "found user" --&gt; Change([Change user to keeper])     Change --&gt; End((( )))     Decision1 -- "search again" --&gt; Decision2{ }     Decision2 --&gt; Search </pre> <p>The diagram illustrates the 'Add Keeper' process. It begins with a start node leading to a 'Login' use case. Following 'Login', the process moves to a 'Search for user' use case. A decision diamond follows 'Search for user'. If the user is 'found', the process proceeds to 'Change user to keeper', which then leads to the end node. If the user is not found, the process loops back to 'Search for user' via a 'search again' path.</p>
<b>Rationale</b>	A member of staff appointed in the department as an administrator is responsible for adding and removing members of staff as keepers of books.
<b>Priority</b>	Must have
<b>Status</b>	Not implemented
<b>Actors</b>	<ul style="list-style-type: none"> <li>• Administrator</li> </ul>
<b>Extensions</b>	
<b>Includes</b>	<ul style="list-style-type: none"> <li>• Login</li> <li>• Search for user</li> </ul>
<b>Conditions</b>	<p><b>post</b> The member of staff is registered as a keeper able to receive and lend books.</p>
<b>Non-Functional Requirements</b>	
<b>Scenarios</b>	
<b>Risks</b>	<ul style="list-style-type: none"> <li>• The system may not be able to determine whether a user is a member of staff, eligible to be designated as a keeper.</li> </ul>
<b>User Interface</b>	



Use case	Remove Keeper
Description	<pre> graph TD     Start(( )) --&gt; Login([Login])     Login --&gt; Search([Search for user])     Search --&gt; Decision1{ }     Decision1 -- "found user" --&gt; Change([Change keeper to user])     Change --&gt; End((( )))     Decision1 -- "search again" --&gt; Decision2{ }     Decision2 --&gt; Search </pre> <p>The diagram illustrates the 'Remove Keeper' process. It begins with a start node leading to a 'Login' use case. Following 'Login', the process moves to a 'Search for user' use case. A decision point follows the search. If the user is 'found', the process proceeds to 'Change keeper to user', which then leads to the end node. If the user is not found, the process loops back to the 'Search for user' use case via a 'search again' path.</p>
Rationale	A member of staff appointed in the department as an administrator is responsible for de-authorising members of staff as keepers.
Priority	Should have
Status	Not implemented
Actors	
Extensions	
Includes	<ul style="list-style-type: none"> <li>• Login</li> <li>• Search for user</li> </ul>
Conditions	<p><b>pre</b> The keeper to be removed must exist in the system.</p> <p><b>pre</b> The keeper must not have any books, or have lent books to users.</p> <p><b>post</b> The member of staff is de-authorised to keep and loan books.</p>
Non-Functional Requirements	
Scenarios	
Risks	<ul style="list-style-type: none"> <li>• The system may not be able to determine whether a user is a member of staff, eligible to be designated as a keeper.</li> <li>• Keepers may need to be de-authorised while they still have books on loan to other users.</li> </ul>
User Interface	

## 4.2 Transferring Books from the Central Library to the Branch

### Use Case Diagram



A keeper can request books to be transferred from the Central Library to the branch (in the possession of the keeper). The librarian must process each request. When the book arrives in the branch, the keeper must record receipt of the book before it can be sub-lent.

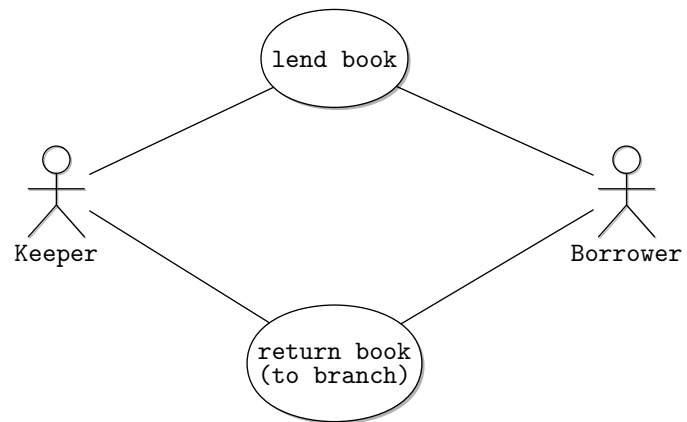
Use case	Request book for branch
Description	<pre> graph TD     Start(( )) --&gt; Login([log in])     Login --&gt; SelectReq([select request book])     SelectReq --&gt; Cond1{ }     Cond1 -- "[connection available?]" --&gt; End1((( )))     Cond1 --&gt; Search([search for books])     Search --&gt; Cond2{ }     Cond2 -- "[books found?]" --&gt; SelectBooks([select books])     SelectBooks --&gt; Confirm([confirm request])     Confirm --&gt; End2((( )))     Cond2 --&gt; Cond3{ }     Cond3 -- "[search again?]" --&gt; End3((( )))     Cond3 --&gt; Search </pre> <p>The diagram illustrates the process of requesting a book for a branch. It begins with a start node leading to the 'log in' use case, followed by 'select request book'. A decision point follows: if the connection is available, the process ends; otherwise, it proceeds to 'search for books'. Another decision point follows the search: if books are found, the user proceeds to 'select books' and then 'confirm request' before ending. If no books are found, a third decision point determines if the user wants to 'search again'. If yes, it loops back to 'search for books'; if no, it ends.</p>
Rationale	Keepers need to be able to request books to be transferred into the branch library.
Priority	Must Have
Status	Not implemented.
Actors	<ul style="list-style-type: none"> <li>• Keeper</li> </ul>
Extensions	
Includes	<ul style="list-style-type: none"> <li>• Login</li> <li>• Search for book</li> </ul>
Conditions	<p><b>post</b> A request for the transfer of the requested book is stored for processing.</p>
Non-Functional Requirements	
Scenarios	
Risks	
User Interface	

Use case	Process Request
Description	<pre> graph TD     Start(( )) --&gt; Login([Login])     Login --&gt; Decision1{ }     Decision1 -- "no pending requests" --&gt; End(( ))     Decision1 --&gt; Access([Access next request])     Access --&gt; Decision2{ }     Decision2 -- approve --&gt; Mark([Mark book record 'in transit'])     Decision2 -- deny --&gt; Enter([Enter comment])     Decision2 -- postpone --&gt; Decision1     Enter --&gt; Send([Send denial])     Mark --&gt; Decision3{ }     Send --&gt; Decision3     Decision3 -- "review next request" --&gt; Decision1     Mark --&gt; Decision1     </pre> <p>The diagram illustrates the 'Process Request' use case. It begins with a start node leading to a 'Login' activity. Following login, a decision point checks for 'no pending requests'. If true, the process ends. If false, it proceeds to 'Access next request'. Another decision point follows, with three paths: 'approve' leads to 'Mark book record 'in transit'', 'deny' leads to 'Enter comment' and then 'Send denial', and 'postpone' loops back to the decision point. Both 'Mark book record 'in transit'' and 'Send denial' lead to a third decision point, which then loops back to the 'no pending requests' decision point via the 'review next request' path.</p>
Rationale	Librarians need to process requests for books from the branch keepers. Keepers need to be notified of the Librarian's decision to approve or deny requests.
Priority	Must Have
Status	Not implemented
Actors	<ul style="list-style-type: none"> <li>• Librarian</li> <li>• Keeper</li> </ul>
Extensions	
Includes	<ul style="list-style-type: none"> <li>• Login</li> </ul>
Conditions	<p><b>post</b> Any request of deny decisions are recorded.</p> <p><b>post</b> The requesting keeper is notified of the librarian's decision.</p>
Non-Functional Requirements	
Scenarios	
Risks	
User Interface	

<b>Use case</b>	Record receipt of book
<b>Description</b>	<pre> graph TD     Start(( )) --&gt; Login(Login)     Login --&gt; ARRecord([Access Request Record])     ARRecord --&gt; ChangeStatus([Change request status to received])     ChangeStatus --&gt; End((( ))) </pre> <p>The diagram shows a vertical flow starting from a solid black circle (start node), followed by an arrow pointing to an oval labeled 'Login'. Another arrow points down to a rounded rectangle labeled 'Access Request Record'. A third arrow points down to another rounded rectangle labeled 'Change request status to received'. Finally, an arrow points down to a bullseye symbol (end node).</p>
<b>Rationale</b>	The keeper is required to explicitly acknowledge receipt of a book before they lend it out to a borrower. The use case supports the tracking and auditing of a book's location.
<b>Priority</b>	Must Have
<b>Status</b>	Not implemented
<b>Actors</b>	<ul style="list-style-type: none"> <li>• Keeper</li> <li>• Librarian</li> </ul>
<b>Extensions</b>	
<b>Includes</b>	<ul style="list-style-type: none"> <li>• Login</li> </ul>
<b>Conditions</b>	<p><b>pre</b> The book must be recorded as 'in transit'.</p> <p><b>post</b> The book is recorded as in the possession of the keeper.</p>
<b>Non-Functional Requirements</b>	
<b>Scenarios</b>	
<b>Risks</b>	
<b>User Interface</b>	

### 4.3 Sub-Lending Books

#### Use Case Diagram



Keepers can sub-lend books in their possession to borrowers.

<b>Use case</b>	Record book lend
<b>Description</b>	<pre> graph TD     Start(( )) --&gt; Login([Login])     Login --&gt; Fork1[ ]     Fork1 --&gt; SearchBook([Search for book])     Fork1 --&gt; SearchUser([Search for user])     SearchBook --&gt; Decision{ }     Decision -- "is keeper" --&gt; Fork2[ ]     SearchUser --&gt; Fork2     Fork2 --&gt; ApproveLoan([Approve loan])     ApproveLoan --&gt; End((( )))     Decision --&gt; End </pre> <p>The diagram illustrates the process of recording a book loan. It begins with a start node leading to a 'Login' use case. Following login, the process splits into two parallel paths: 'Search for book' and 'Search for user'. From 'Search for book', the flow enters a decision diamond. If the user is a 'keeper', the flow proceeds to a join bar before the 'Approve loan' use case. If not a keeper, the flow bypasses the decision and goes directly to the end node. The 'Search for user' path also joins the 'Approve loan' use case. The process concludes with the 'Approve loan' use case leading to the end node.</p>
<b>Rationale</b>	Keepers need to be able to document the lending of books to borrowers. Borrowers are notified when they have been assigned a book.
<b>Priority</b>	Must have
<b>Status</b>	Not implemented
<b>Actors</b>	<ul style="list-style-type: none"> <li>• Keeper</li> <li>• Borrower</li> </ul>
<b>Extensions</b>	
<b>Includes</b>	<ul style="list-style-type: none"> <li>• Login</li> <li>• Search for book</li> <li>• Search for borrower</li> </ul>
<b>Conditions</b>	<b>post</b> The status of the book is changed to 'lent' to named borrower.
<b>Non-Functional Requirements</b>	
<b>Scenarios</b>	
<b>Risks</b>	
<b>User Interface</b>	

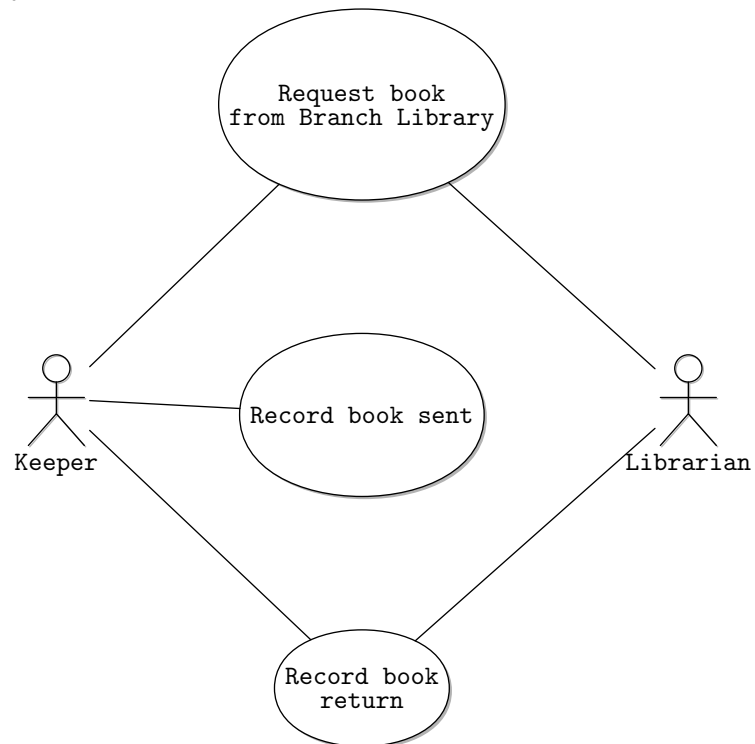
<b>Use case</b>	Request book return (to Branch)
<b>Description</b>	<pre> graph TD     Start(( )) --&gt; Login(Login)     Login --&gt; Search(Search for book)     Search --&gt; Decision{is keeper}     Decision --&gt; Request(Make return request)     Request --&gt; End((( )))     Decision --&gt; End </pre> <p>The diagram illustrates the process of requesting a book return. It begins with a start node leading to a 'Login' use case, followed by 'Search for book'. A decision diamond labeled 'is keeper' follows. If the user is a keeper, the flow proceeds to 'Make return request' and then to the end node. If not a keeper, the flow bypasses the request step and goes directly to the end node.</p>
<b>Rationale</b>	Keepers need to be able to request that books are returned by a borrower.
<b>Priority</b>	Must have
<b>Status</b>	Not implemented
<b>Actors</b>	<ul style="list-style-type: none"> <li>• Keeper</li> <li>• Borrower</li> </ul>
<b>Extensions</b>	
<b>Includes</b>	<ul style="list-style-type: none"> <li>• Login</li> <li>• Search for book</li> </ul>
<b>Conditions</b>	<p><b>pre</b> Book status is set to “lent to named borrower”</p> <p><b>post</b> The borrower receives notification that the keeper wishes for the book to be returned.</p>
<b>Non-Functional Requirements</b>	
<b>Scenarios</b>	
<b>Risks</b>	
<b>User Interface</b>	



<b>Use case</b>	Record book return (to branch)
<b>Description</b>	<pre> graph TD     Start(( )) --&gt; Login([Login])     Login --&gt; Search([Search for book])     Search --&gt; Decision{ }     Decision -- "is keeper" --&gt; Mark([Mark book as returned])     Mark --&gt; End((( ))) </pre> <p>The diagram illustrates the process of recording a book return. It begins with a start node leading to a 'Login' use case, followed by 'Search for book'. A decision point follows, with a path labeled 'is keeper' leading to 'Mark book as returned', which then leads to the end node.</p>
<b>Rationale</b>	Keepers need to be able to document the collection of books from borrowers. Borrowers are notified when a book they have borrowed is recorded as having been returned to the book's keeper.
<b>Priority</b>	Must have
<b>Status</b>	Not implemented
<b>Actors</b>	<ul style="list-style-type: none"> <li>• Keeper</li> <li>• Borrower</li> </ul>
<b>Extensions</b>	
<b>Includes</b>	<ul style="list-style-type: none"> <li>• Login</li> <li>• Search for book</li> <li>• Search for user</li> </ul>
<b>Conditions</b>	<p><b>pre</b> The keeper is responsible for the book.</p> <p><b>post</b> The status of the book is changed to in possession of the keeper.</p>
<b>Non-Functional Requirements</b>	
<b>Scenarios</b>	
<b>Risks</b>	
<b>User Interface</b>	

#### 4.4 Return books to the Central Library

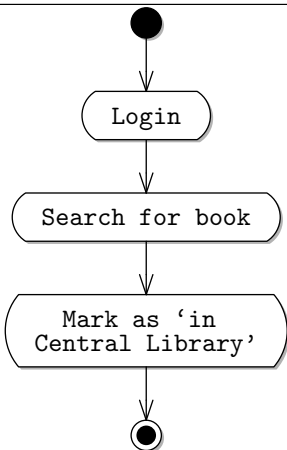
##### Use Case Diagram



Librarians in the Central library are able to request that books stored by branches be returned. Book keepers are responsible for sending books back to the library and can document this action on the system. The librarian records books as having been received back at the Central Library.

<b>Use case</b>	Request book from Branch
<b>Description</b>	<pre> graph TD     Start(( )) --&gt; Login([Login])     Login --&gt; Search([Search for book])     Search --&gt; Decision{ }     Decision -- "in branch" --&gt; Confirm([Confirm Request])     Confirm --&gt; End((( )))     Decision --&gt; End </pre> <p>The diagram illustrates the process of requesting a book from a branch. It begins with a start node leading to a 'Login' use case, followed by a 'Search for book' use case. A decision point follows, with a path labeled 'in branch' leading to a 'Confirm Request' use case, which then leads to the end node. An alternative path from the decision point also leads to the end node.</p>
<b>Rationale</b>	Administrators need to request books be returned from a branch, when a central library borrower needs the book, for example.
<b>Priority</b>	Must Have
<b>Status</b>	Not implemented
<b>Actors</b>	<ul style="list-style-type: none"> <li>• Librarian</li> <li>• Keeper</li> </ul>
<b>Extensions</b>	
<b>Includes</b>	<ul style="list-style-type: none"> <li>• Login</li> <li>• Search for book</li> </ul>
<b>Conditions</b>	<b>post</b> A return request is recorded for the book.
<b>Non-Functional Requirements</b>	
<b>Scenarios</b>	
<b>Risks</b>	
<b>User Interface</b>	

<b>Use case</b>	Record book sent
<b>Description</b>	includegraphicsfigures/refinement/book-to-central/record-book-sent-1
<b>Rationale</b>	This feature allows keepers to document when a book has been sent back to the Central Library.
<b>Priority</b>	Should have. This feature is useful for tracking purposes but is not part of the core work flow of the system.
<b>Status</b>	Not implemented.
<b>Actors</b>	<ul style="list-style-type: none"> <li>• Keeper</li> </ul>
<b>Extensions</b>	
<b>Includes</b>	<ul style="list-style-type: none"> <li>• Login</li> <li>• Search for book</li> </ul>
<b>Conditions</b>	<p><b>pre</b> A return request has been made for the book.</p> <p><b>pre</b> The book is recorded as in the keeper's possession.</p> <p><b>post</b> Book is recorded as being 'in transit' to the Central Library.</p>
<b>Non-Functional Requirements</b>	
<b>Scenarios</b>	
<b>Risks</b>	
<b>User Interface</b>	

<b>Use case</b>	Record book return
<b>Description</b>	 <pre> graph TD     Start(( )) --&gt; Login(Login)     Login --&gt; Search(Search for book)     Search --&gt; Mark(Mark as 'in Central Library')     Mark --&gt; End((( ))) </pre>
<b>Rationale</b>	Supports book tracking by allowing the librarian to record when a book is received back at the central library.
<b>Priority</b>	Must Have
<b>Status</b>	Not implemented
<b>Actors</b>	<ul style="list-style-type: none"> <li>• Librarian</li> </ul>
<b>Extensions</b>	
<b>Includes</b>	<ul style="list-style-type: none"> <li>• Login</li> <li>• Search for book</li> </ul>
<b>Conditions</b>	<p><b>pre</b> A return request has been made for the book.</p> <p><b>pre</b> The book is recorded as 'in transit' or in possession of a Keeper.</p> <p><b>post</b> Book is recorded as being in the Central Library.</p>
<b>Non-Functional Requirements</b>	
<b>Scenarios</b>	
<b>Risks</b>	
<b>User Interface</b>	

## **4.5 Common shared Use Cases**

This section describe a number of use cases which are included elsewhere in the model.

Use case	Login
Description	<pre> graph TD     Start(( )) --&gt; D1{ }     D1 -- "[logged in?]" --&gt; End1((( )))     D1 --&gt; Fork[ ]     Fork --&gt; EnterUsername([enter username])     Fork --&gt; EnterPassword([enter password])     EnterUsername --&gt; Join[ ]     EnterPassword --&gt; Join     Join --&gt; Submit([submit])     Submit --&gt; D2{ }     D2 -- "[correct?]" --&gt; End2((( )))     D2 --&gt; Delay([do delay])     Delay --&gt; Join   </pre>
Rationale	The login use cases permits different levels of access to the system. A username is the unique user ID provided by the Central Library. A password must be up to eight alpha numeric characters.
Priority	Must have
Status	Not implemented
Actors	<ul style="list-style-type: none"> <li>• Librarian</li> <li>• Borrower</li> </ul>
Extensions	
Includes	
Conditions	<b>post</b> The user is logged in if correct credentials are provided
Non-Functional Requirements	
Scenarios	
Risks	
User Interface	

<b>Use case</b>	View Status
<b>Description</b>	A user's status (books borrowed, books lent, privileges, requests for book returns) is displayed.
<b>Rationale</b>	Users' need to be able to view their status within the system.
<b>Priority</b>	Must have
<b>Status</b>	Not implemented
<b>Actors</b>	<ul style="list-style-type: none"> <li>• User</li> </ul>
<b>Extensions</b>	
<b>Includes</b>	
<b>Conditions</b>	<b>pre</b> The user is logged in.
<b>Non-Functional Requirements</b>	
<b>Scenarios</b>	
<b>Risks</b>	
<b>User Interface</b>	



<b>Use case</b>	Logout
<b>Description</b>	The logout use cases changes a users account to logged out, requiring them to re-authenticate to the system. Logout is either invoked by the user or by the internal inactivity timer actor.
<b>Rationale</b>	The logout use case allows a user to leave the system to prevent unauthorised access from an unattended terminal.
<b>Priority</b>	Must have
<b>Status</b>	Not implemented
<b>Actors</b>	<ul style="list-style-type: none"> <li>• Librarian</li> <li>• Borrower</li> <li>• Inactivity Timer</li> </ul>
<b>Extensions</b>	
<b>Includes</b>	
<b>Conditions</b>	<p><b>pre</b> The user is logged in</p> <p><b>post</b> The user is logged out</p>
<b>Non-Functional Requirements</b>	
<b>Scenarios</b>	
<b>Risks</b>	
<b>User Interface</b>	

<b>Use case</b>	Search for book
<b>Description</b>	<pre> graph TD     Start(( )) --&gt; SubmitQuery([Submit query])     SubmitQuery --&gt; FoundBook{ }     FoundBook -- "found book" --&gt; End1((( )))     FoundBook --&gt; SearchAgain{ }     SearchAgain -- "search again" --&gt; SubmitQuery     SearchAgain --&gt; End2(((X))) </pre> <p>Minimally, a user may search by:</p> <ul style="list-style-type: none"> <li>• an initial substring for the title of a book. For example, all books whose title begin with the term “Software Engineering” would be included in the search result for the query “Software Eng”;</li> <li>• the exact identifier of a book.</li> </ul>
<b>Rationale</b>	Allows the branch library to be queried for locations of books with keepers. Note that searching books does not require a user to log in to the system.
<b>Priority</b>	Must have
<b>Status</b>	Not implemented
<b>Actors</b>	<ul style="list-style-type: none"> <li>• User</li> </ul>
<b>Extensions</b>	
<b>Includes</b>	
<b>Conditions</b>	
<b>Non-Functional Requirements</b>	
<b>Scenarios</b>	
<b>Risks</b>	
<b>User Interface</b>	

<b>Use case</b>	Search for user
<b>Description</b>	<pre> graph TD     Start(( )) --&gt; Login(Login)     Login --&gt; SubmitQuery([Submit query])     SubmitQuery --&gt; FoundUser{ }     FoundUser -- found user --&gt; End1((( )))     FoundUser --&gt; SearchAgain{ }     SearchAgain -- search again --&gt; SubmitQuery     SearchAgain --&gt; End2((X)) </pre>
<b>Rationale</b>	Allows the branch library to be queried for information on users by keepers.
<b>Priority</b>	Must have
<b>Status</b>	
<b>Actors</b>	<ul style="list-style-type: none"> <li>• Keeper</li> </ul>
<b>Extensions</b>	
<b>Includes</b>	
<b>Conditions</b>	
<b>Non-Functional Requirements</b>	
<b>Scenarios</b>	
<b>Risks</b>	
<b>User Interface</b>	

## **5 Non Functional Requirements**

The system will need to be able to interact with the central library's existing database of books and users.

1. Any data must be stored in such a way that the storage medium can be changed without changing the system's architecture.
2. The system must interact with the Central Library's database of books.
3. The system must interact with the Central Library's database of users. Access control must be implemented with reference to this database.
4. The user interface to the system should emphasise simplicity and must be minimally functional for the purposes of the final demonstration.