Software Requirements Specification

for

Library

Prepared by Axel Caliari, Emmanuel Gay, Jiahong Wu and Victor Wang

Concordia University

23/05/2018

Table of Contents

1. Introduction 1

1.1 Purpose 1

1.2 Intended Audience and Reading Suggestions 1

1.3 Project Scope 1

1.4 References 1

2. Overall Description 2

2.1 Product Perspective 2

2.2 Product Features 2

2.3 User Classes and Characteristics 2

2.4 Operating Environment 2

2.5 Design and Implementation Constraints 2

2.6 User Documentation 3

2.7 Assumptions and Dependencies 3

2.8 Budget 3

3. System Features 3

3.1 Project Features 3

3.2 Use Case 1 4

4

4. External Interface Requirements 4

4.1 User Interfaces 4

4.2 Hardware Interfaces 4

4.3 Software Interfaces 5

4.4 Communications Interfaces 5

5. Other Nonfunctional Requirements 5

5.1 Performance Requirements 5

5.2 Safety Requirements 5

5.3 Security Requirements 5

5.4 Software Quality Attributes 6

6. Other Requirements 6

# Introduction

## Purpose

The purpose of this document is to give a detailed description of the requirements for the “Library” software. It will illustrate the purpose and complete declaration for the development of the system. This document is primarily intended to be proposed to a project manager for its approval and a reference for developing the first version of the system for the development team.

Please do note that this project is realized by students, consequently people with higher expertise, experience and knowledge may find the application not plausible in a business or practical environment, not well organized and optimized, or incomplete. The purpose of this document is to showcase utilization of UML and design patterns that we, the students, have learnt and applied.

(Note to the professor: Please forgive us if the project seems to be much more database-focused than object-focused.)

## Project Scope

The Library a Java-based application designed for easy handling of librarian-specific tasks. With just a few clicks, the user can add, remove or search for a book in the library, browse the catalog, and make reservations for customers, making both the librarian’s and the borrower’s life easier. Items, customers and active reservations of book in the library are stored in a database and can be sorted in various ways by the user. The application is only accessible by registered librarians with the valid login credentials.

## Definitions, acronyms, and abbreviations

|  |  |
| --- | --- |
| User | Someone who interacts with the application |
| Admin | In this application, the admin is the user, and thus the librarian |
| Customer | Borrower, someone who borrows a book from the library |
| CRUD | Create, read, update and delete. User interface conventions facilitating viewing, searching and changing information |
| MVC | Model-view-control. Design pattern used for Library |
| GUI | Graphical User Interface |

# Overall Description

## Product Perspective

This system is a data-centric application. The main part, the Library application, will be used as a platform for interactions between the user and the database.

Desktop software

Database server

User

Library application

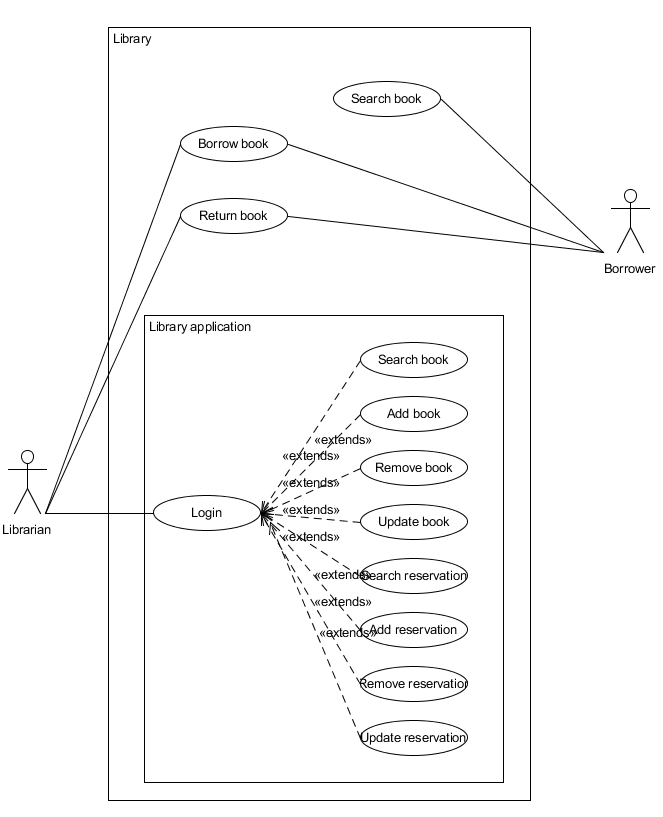
Database

*Figure 1 – Block diagram*

The Library application is what controls what’s in the system. Indeed, it defines the types of objects that are manipulated and who manipulates them, in our example the books and the administrators. It also will need to communicate with the database, sending information but also retrieving it for the user to consult. It will also have to be able to modify existing information in the database and update it.

## Main functionalities

Only one type of user interacts with the system: the librarian. In our application, he is considered as the administrator of the system, having the rights to manipulate every type of information that is present in the system: books, customers, reservations. The administrator can add a book to the library, define its type by selecting one of the available options, then entering the title, the author; he can add customers who have borrowed books by providing their first and last name; he can add a reservation for a certain book, providing the borrower’s identification, the borrow date and the return date, and determine if the book has been returned or not. He can also remove a book from the library or remove a customer from the database.



*Figure 2 – Use case diagram*

## Operating Environment

This application is ran using Java, so it is compatible with all operating systems. The current test build is developed using Eclipse, with additional libraries for MySQL connections and plugins for the User Interface (javafx).

# System Features

## Project Features

Our take on this application is a classic one frequently used in data-centric projects: the MVC design pattern. Consequently, the application is separated into three main packages: Model, Control/View, and Persistence.

Model is the central component, containing the application’s behavior domain independent of the user interface. It controls and manages the data of the system, as in what describes:

- a book (title, author, id etc.) and its type (whether it is a magazine, a novel or manual)

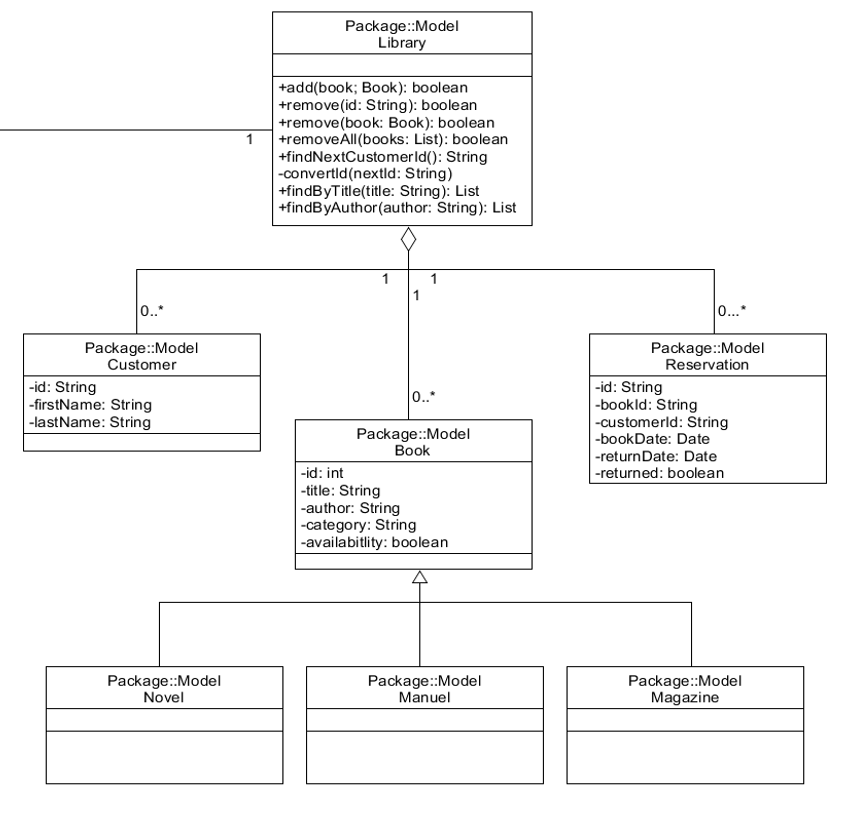
- an administrator (login credentials)

- a customer (first name, last name, id)

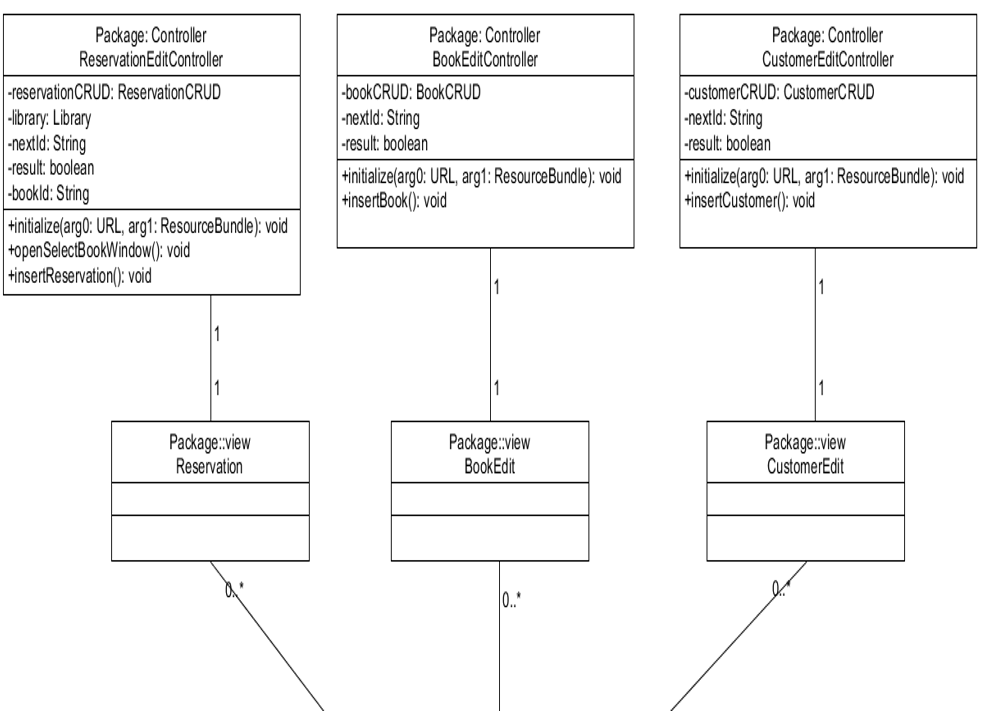
- a reservation (book id, customer id, id, book and return date, status as in if the book has been returned or not)

- what is in the library (array of books, customers and reservations)

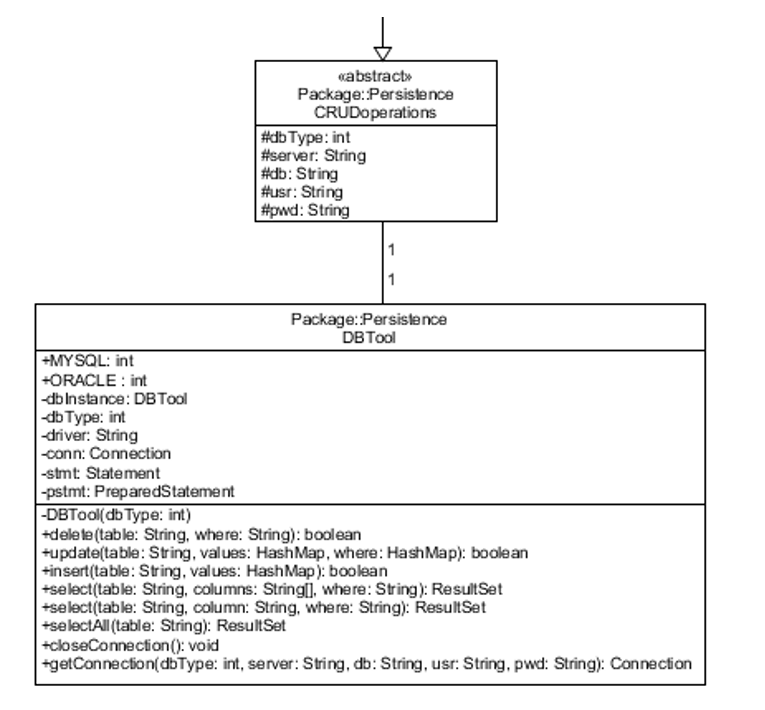
Control/View, named simply Controller, has the dual purpose of controlling the output representation of information (View) and managing input and converting it for the Model or View (Controller). As such, this is where the plugin javafx has been prominently used for handling the GUI. Each main actions (select a book, edit a reservation, edit a customer) and



*Figure 3 –Model class diagram*



*Figure 3 –Controller/View class diagram*



*Figure 4 – Persistence class diagram*

## Use Case 1

<Itemize the detailed functional requirements associated with this feature. These are the software capabilities that must be present in order for the user to carry out the services provided by the feature, or to execute the use case. Include how the product should respond to anticipated error conditions or invalid inputs. Requirements should be concise, complete, unambiguous, verifiable, and necessary. Use “TBD” as a placeholder to indicate when necessary information is not yet available.>

<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.> REQ-1: REQ-2:

## 

# External Interface Requirements

## User Interfaces

<Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, and so on. Define the software components for which a user interface is needed. Details of the user interface design should be documented in a separate user interface specification.>

## Hardware Interfaces

<Describe the logical and physical characteristics of each interface between the software product and the hardware components of the system. This may include the supported device types, the nature of the data and control interactions between the software and the hardware, and communication protocols to be used.>

## Software Interfaces

<Describe the connections between this product and other specific software components (name and version), including databases, operating systems, tools, libraries, and integrated commercial components. Identify the data items or messages coming into the system and going out and describe the purpose of each. Describe the services needed and the nature of communications. Refer to documents that describe detailed application programming interface protocols. Identify data that will be shared across software components. If the data sharing mechanism must be implemented in a specific way (for example, use of a global data area in a multitasking operating system), specify this as an implementation constraint.>

## Communications Interfaces

<Describe the requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on. Define any pertinent message formatting. Identify any communication standards that will be used, such as FTP or HTTP. Specify any communication security or encryption issues, data transfer rates, and synchronization mechanisms.>

# Other Nonfunctional Requirements

## Performance Requirements

<If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.>

## Safety Requirements

<Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product’s design or use. Define any safety certifications that must be satisfied.>

## Security Requirements

<Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication requirements. Refer to any external policies or regulations containing security issues that affect the product. Define any security or privacy certifications that must be satisfied.>

## Software Quality Attributes

<Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.>

# Other Requirements

<Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project.>

Appendix A: Glossary

<Define all the terms necessary to properly interpret the SRS, including acronyms and abbreviations. You may wish to build a separate glossary that spans multiple projects or the entire organization, and just include terms specific to a single project in each SRS.>

Appendix B: Analysis Models

<Optionally, include any pertinent analysis models, such as data flow diagrams, class diagrams, state-transition diagrams, or entity-relationship diagrams.>

Appendix C: Issues List

< This is a dynamic list of the open requirements issues that remain to be resolved, including TBDs, pending decisions, information that is needed, conflicts awaiting resolution, and the like.>