**Vision Document for “Library Management System”**

****

**Team members:**

* **Bao Quoc Nguyen - 614207**
* **Deivison Mozer de Souza - 614634**
* **Luis Santiago Escobar Ramirez - 614615**
* **Phong Lu - 614616**

# 

Table of Contents

[**1. Introduction 4**](#_eqrlqrlhpwk8)

[**2. Positioning 6**](#_scd21fm9ktzy)

[2.1. Problem Statement 6](#_n23svxec4d0w)

[2.2. Product Position Statement 6](#_334m3sa7i56o)

[**3. Stakeholder Descriptions 7**](#_pecisb9u8tm3)

[3.1. Stakeholder Summary 7](#_qiuo6oxd0qb1)

[3.2. User Environment 7](#_am51q37nl80)

[**4. Product Overview 8**](#_ed24rh4knarb)

[4.1. Product Perspective 8](#_t0hgjlerko0n)

[4.2. Assumptions and Dependencies 8](#_kf3l1fy5zz00)

[4.3. Needs and Features 9](#_nr1otnoo1pi2)

[4.4. Alternatives and Competition 11](#_9yv331st8jxv)

[**5. Other Product Requirements 11**](#_z6w9gi51jouk)

# 

# 

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 07/01/2023 | 0.1 | Initial version | All team members |
|  |  |  |  |

# 

# Introduction

Library management system (LMS) is a web-based application designed to digitally transform the library operations at the university (not a public library), enhance the efficiency of library administration and benefit the users with ease of use.

LMS manages 3 different roles of user (student, librarian, and administrator). Based on the logged in user’s role, the system will provide a set of functionalities accordingly.

Every user after logging in is allowed to update their profile (contact number, address) and account (user name, password)

The administrator role is the highest role in the role’s hierarchy. They can perform the functionality of both librarians and students. However, their main responsibilities are to manage:

* System configuration:
  + number of days is free to borrow
  + fine for overdue (per day)
  + maximum number of books can borrow
  + number of overdues to be suspended (etc. 3)
  + number of overdues to be locked
  + number of suspended days
  + number of days to be reminded
* User: CRUD (create, update, delete) librarian / administrator
* Author: CRUD
* Book and book category: CRUD
* Add book copy
* Unlock student account

The librarian’s role is to:

* List checkout records (all, by specific day, by student id…)
* List overdue books
* Check the next availability of the book by different criterias (name, isbn, authors..)
* Process the checkout and returned book
* Charge overdue (refer to Assumption section for details)

Finally, the student role is to:

* Self register for their own account. The student identity will be the student ID
* List of checkout books own by the student
* Check the availability of the expectation book, and find the physical location.
* Get the book physically and contact the librarian in person (at the service desk) to perform checkout.
* Contact the librarian in person to return the book.

There’s also a schedulers running at the backend at the end of the day to:

* reactivate the account if the suspended time is over.
* sending reminder to student by email if the due date in the next 3 days

**Optional features:** we will try to implement these feature if we have enough time

* Student / librarian can propose the new book to the library
* Statistic on book category/ author / book name
* Statistic on checkout records
* Students can send the feedback to the library administration team.
* Test plan, test cases
* Logging centralization using Elasticsearch ELK

# Positioning

## Problem Statement

|  |  |
| --- | --- |
| The problem of | The administrators, and librarians manage the books and the books status. |
| Affects | the library, books, students, librarians, and administrators. |
| the impact of which is | Keeping track of the books’ status is hard.  The lending books process is complicated, time consuming by paperwork. |
| a successful solution would be | a system that to keep track of the books’ status efficiently with ease-of-use |

## Product Position Statement

|  |  |
| --- | --- |
| For | librarians, students, administrators. |
| Who | digitalize the book lending operations to be more accurate, efficient. |
| The (product name) | LMS |
| That | Check the availability of the book instantly, and simplify the administration operations efficiently. Benefits the users in many different perspectives |
| Unlike | CodeAchi, Libero |
| Our product | Lightweight, effective, user friendly, cutting edge technology, easy to integrate with other existing systems, easy to scale up/down for every single service, easy to enhance and adapt new technologies since its architecture is based on Spring Cloud standard framework. |

# 

# Stakeholder Descriptions

## Stakeholder Summary

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Responsibilities** |
| **Administrators (Admin)** | Admins manage the information related to book, system configuration data, and administration users (Administrators, librarians) | * Setting up the system with configuration data * CRUD book categories, books, authors, book copies * CRUD administrators/ librarian * Unlock student account |
| **Librarians** | Librarians handle check-out and returning books | * Handle checkout books * Handle returning books * Handle overdue * List out checkout records |
| **Students** | Users can search for books, check availability, borrow and return the books. | * Self-register their own account and maintain the profile correctly * Borrow the books and return the books |
| **Developers** | Developers implement the system based on the requirement specification | * Implement the software features * Write the unit test for the features they’ve developed |
| **Testers** | Testers test the system and make sure the features are working as expected | * Prepare the test plan * Perform testing and provide the test report |

## User Environment

The system is a web-based application based on micro-services architecture. The development is divided by front-end and back-end groups with 2 members for each group.

The back-end system is developed based on the Java programming language. It’s an independent programming language so it would be able to be deployed on any platform (Windows, Linux..). React is used for front-end development so the user interface can run on any of browsers (Chrome, Safari, Internet Explorer…)

The LMS runs on its own at the moment. In the future, however, it would be better if LMS is able to:

* integrate with the existing student administration system to validate the student ID, and student names.
* interage with the existing payment system for overdue charges

# Product Overview

## Product Perspective

The software itself is an independent and self-contained software. At first, it can stand-alone but could be extended by:

* integrate with the existing system (student administration, payment)
* use Memory Database (Hazelcast, Redis etc.) to cache the category data, configurable data… to tune up the performance.

## Assumptions and Dependencies

We’re using Java as an independent platform programming language for the back-end so it will be available for almost all operating systems. React as a front-end UI which makes it available for almost any browser running on web browsers (Safari, Chrome…)

Assumptions:

* The administrator(s) account will be set up as predefined data in the database.
* In case the users forgot the password, they will use the forgot password function, entering the valid student id, name and user ID, then the new password will be sent to their email.
* The administrator will handle lost, damaged books by updating the book copy.
* Payment system for the overdue, lost or damage book is out of project scope. It’ll be handled separately.
* Verifying the validity of student ID, name, email is out of project scope.
* Student self-registration account will be automatically accepted if the student ID doesn’t exist in the system.
* Unlocking account handling is out of project scope.
* Same copy of the book will be physically located in the same aisle/section. The book location information is just for reference purposes only (etc. A10-1, B10-2..)
* Checkout and return books must be done at the library service desk.
* Junit will be used at back end end-point and test scope focus on main flow only (end-point and service level)
* Since the back-end services can be run multiple instances simultaneously by scaling up or down and logging centralization is out of project scope due to time frame limit.
* CICD pinepine (Jenkins… etc.) integrated with SonarQube source code analysis is out of project scope.
* The proxy at the back-end will be used with an internal load balancing algorithm in case multiple instances are running.
* High availability and fault tolerance can be partially supported but will not be in the project scope.
* Multiple clusters will be out of project scope

## Needs and Features

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No** | **Problem** | **Need** | **Priority** | **Features** | **Planned Release** |
| **Common features** | | | | | |
| 1 | Login | Users log into the LMS | 1 | Users enter user name and password |  |
| 2 | Update profile information | update user profile - except for the status | 2 | Users can update name, email, address |  |
| 3 | Update contact information | update the contact information | 2 | Users can update login user id and password |  |
| 4 | Find the books to borrow | search to find expect book | 1 | users can search for a book by multiple criteria |  |
| 5 | Forgot password | reset password | 2 | users want to reset their password |  |
| **Administrator** | | | | | |
| 6 | Add librarian / or administrator to the system | CRUD the administrators and librarians | 1 | The admin can manage the administrators and librarians |  |
| 7 | Manage the authors | CRUD the authors | 1 | The admin can manage the authors list |  |
| 8 | Manage the book categories | CRUD the books categories | 1 | The admin can manage the book categories |  |
| 9 | Manage the book | CRUD the books | 1 | The admin can manage the book information |  |
| 10 | Manage the book copy | manage the copy of the books (add more copy or reduce the copy because of lost or damage) | 1 | The admin can manage the copy of the book |  |
| 11 | Manage configuration table | manage configuration table items to avoid hard-coding (number of books can be borrowed concurrently etc.) | 1 | The admin can manage the configurable items |  |
| 12 | List of students | list of students | 1 | The admin can search for particular student |  |
| 13 | Unlock student account | Admin can unlock a student account | 1 | The admin can unlock the student account. |  |
| 14 | Checkout the book on behalf on librarian or for themself | Checkout a book | 1 | The admin can check out a book |  |
| 15 | Return the book on behalf on librarian or for themself | Return the book | 1 | The admin can check in the book |  |
| 16 | List check out records by specific criteria (date range, student..etc) | List the checked out books | 1 | The admin can list out the checkout records |  |
| 17 | List overdue books by specific criteria (category, student..etc) | List the overdue books | 1 | The admin can list the overdue books. |  |
| 18 | Check the next availability of the book based on the the due date | Display the possible available date | 2 | The admin can check the next availability of the book |  |
| 19 | Reset password for users in case they forgot | Reset password for users | 2 | Reset password for users |  |
| **Librarian** | | | | | |
| 20 | Check out the book for students or for themself | Students that have no restriction should be able to borrow books. | 1 | The librarian can check out a book |  |
| 21 | Return the book for students or for themself | return the book to the library | 1 | The librarian can return the book to the library |  |
| 22 | List checkout records by multiple criteria | Display books are currently borrowed | 1 | The librarian can list out checkout records |  |
| 23 | List overdue books | Display overdue books | 1 | The librarian can list the overdue books |  |
| 24 | Check the availability of the book on the due date | Display books next available date | 1 | The librarian can check the next available date for a book. |  |
| **Student** | | | | | |
| 25 | Register for their own account | students can register for their account | 1 | The student can register their user information and credentials |  |
| 26 | List of checkout books are borrowed from the library | list out the checkout books | 1 | The student can list out the books they have borrowed |  |
| **Schedulers - Batch processing** | | | | | |
| 27 | Remind the student for due date | sending the reminder | 2 | Send reminder to student by email |  |
| 28 | Change the status of student from suspended to available if the suspended period is over | change the student status to available | 2 | change the student status to available |  |

## Alternatives and Competition

There are a few commercial products available in the market (CodeAchi, Libero etc.) but they’re not quite fit with the requirement because of:

* Project scope is a lot bigger than our needs.
* Pricy
* Difficult to customize
* Difficult to enhance the functionality
* Difficult to integrate the the existing systems
* Maintenance fee

# Other Product Requirements

The backend will be mostly compatible and deployed with any platform (Windows, Linux…)

MySQL is currently being used but can be ported to any other relational databases: MSSQL Server, Oracle, DB2..etc

The user interface will be mainly used as Chrome in the first phase but can be workable with any other type of browsers (Safari, IE, MS Edge…etc) in the future.

Multiple instances of particular service can be scaled up or down. Load balancing should be used in case multiple instances are running.

Multiple clusters can be easily supported to increase High Availability and fault tolerance in the future as the need grows.

The user-friendly interface allows the users to quickly perform operations without detailed training.

The system will prevent any unauthorized access.

Server registry should be used to monitor the number of microservices instances.

Configuration server should be used to centralize the micro-services configuration.

For every system request, less than 3 seconds to get the response.