|  |
| --- |
| SFWRTECH 4SA3 |
| McLibrary |
| Software Architecture Documentation |

|  |
| --- |
| Mingming Zhang  7-21-2021 |

SFWRTECH 4SA3: Software Architecture

Project Milestone #5 – Implementation

Student Name: Mingming Zhang

Student Number: 400349051

Contents

[1 Introduction 2](#_Toc77791026)

[1.1 Application Overview 2](#_Toc77791027)

[1.2 Technologies 2](#_Toc77791028)

[1.3 Documentation Overview 2](#_Toc77791029)

[1.4 Application Usage 3](#_Toc77791030)

[2 Requirements 3](#_Toc77791031)

[3 Scenario View 4](#_Toc77791032)

[4 Physical View 4](#_Toc77791033)

[5 Development View 5](#_Toc77791034)

[6 Process View 6](#_Toc77791035)

[7 Logical View 8](#_Toc77791036)

[8 Database Schema 9](#_Toc77791037)

# Introduction

In this document, we overview the software architecture for the McLibrary application.

## Application Overview

The McLibrary console application allows users to create a client account and view books from the online library system. It also allows the library administrator to manage the resources data with relative courses.

All e-books in the library have RSA electronic signatures to ensure that users cannot read them without the key. The user can obtain the private key matching his computer ID by borrowing it to open the electronic version of the book. The administrator can manage the increase and decrease of types and quantities in the library. And users can borrow and return electronic books in the library system.

The audience is focused on the librarian and students. The second is the school administrator for the construction area of the library.

## Technologies

The following technologies are used to implement the McLibrary application

|  |  |  |
| --- | --- | --- |
| Technology | Role | Sources |
| Python | Implementation language for all application functionality. | <https://www.python.org/> |
| Redis | Database technology is used to store application data (an in-memory, key-value Database). | <https://redis.io/> |
| Redis Labs | Cloud Redis database-as-a-service provider. | <https://redislabs.com/> |
| Library Genesis | Web API for searching library Genesis data based on title or author. | <https://libgen.is> |
| GenLib API Python Client | Wrapper module for the LibGen API that makes access to the API easier. | <https://pypi.org/project/libgen_api/> |

## Documentation Overview

In section 1.2, we document the requirements for the McLibrary application.

In sections 3 to 7, we use the 4+1 View Model to document the McLibrary application, which involves constructing 7 views focusing on different concerns about the software architecture. We use different types of UML diagrams to document each view. You can read more about the 4+1 View Model in the original paper by Philippe Kruchten:

<https://pe9opel.ucalgary.ca/~far/Lectures/SENG401/PDF/4+1view-architecture.pdf>

## Application Usage

The application can be run with the command **Python3 main.py** from the command line. A configuration file called **config.cfg** must be present, an example of which is provided below:

[Database]

password = J8Kr6ajygygNGg7lppDNyvkXp66AfuSp

host = redis-12493.c275.us-east-1-4.ec2.cloud.redislabs.com

port = 12493

[xxxAPI]

apikey =

[Logging]

file = TRUE

database = TRUE

console = TRUE

log\_filename = log.txt

# Requirements

The McLibrary application was built to satisfy the following requirements.

Table 1 McLibrary Requirements

|  |  |
| --- | --- |
|  | **Functional requirements** |
| **R01** | The system shall track the status of books and remainder in inventory; |
| **R02** | The system shall allow people to create client accounts; |
| **R03** | The system shall allow the administrator to manage the books, including search, create, and delete books; |
| **R04** | Allow client rent and return book; |
| **R05** | Allow client to view current stock and book status; |
| **R06** | Searched data should be retrieved from the LibGen API: The searched book's title, author, year, link. |
|  | **Security requirements** |
| **R07** | The system shall enforce password protection with minimum requirements, eight characters, one number; |
|  | **Traceability requirements** |
| **R08** | The system shall avoid creating a registered user account; |
| **R09** | The system shall avoid creating a duplicated book; |
| **R10** | The system shall judge and modify the booking status according to client operation; |

# Scenario View

**Concerns**: Understanding the central functionality of the system

**Stakeholders**: All stakeholders, but particularly the end-user

**Modelling** **techniques**: UML Use Case Diagram

The users can access these many functions. But the login management is access control to limit the different groups of people has different limits of authority.

The system shall allow the administrator to manage the books, including search, create, and delete books. However, the system shall allow the clients to explore the books in the Database, rent and return books, and take a record into the Database for regular users. See Figure 1 for the Use Case Diagram.

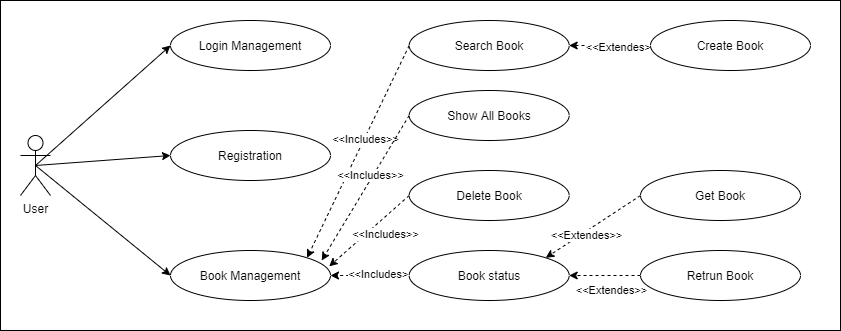


Figure 1 Use Case Diagram (Scenario View)

# Physical View

**Concerns**: Mapping of software to hardware, communication protocols and models related to communication

**Stakeholders**: software architect, software developers

**Modelling** **techniques**: UML Deployment Diagram

The console application communication with the Redis Labs database over TCP port 12493, though this is configuration via the config file if the port needs to be changed in the future. The Redis Labs database is technically hosted in Amazone's EC2 Elastic Computer Lab. The LibGen API is a web service made available over HTTP, and the LibGen API Python client is used to communicate with the API. See Figure 2 for the Deployment Diagram.

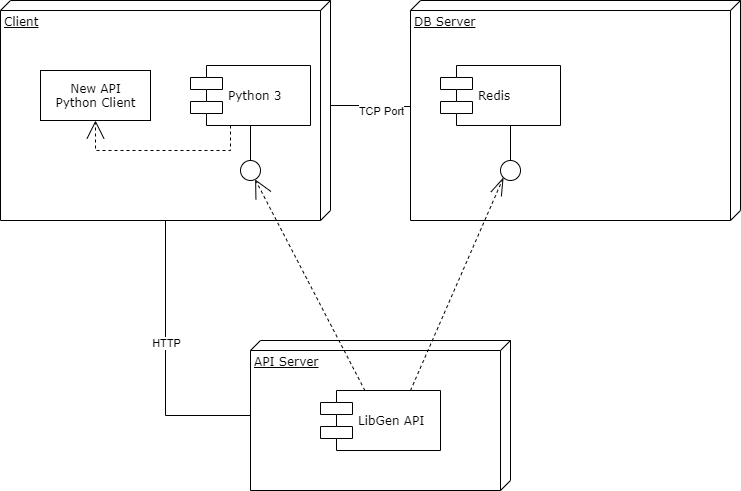


Figure 2 Deployment Diagram (Physical View)

The user can retrieve the book information from the Database and then change the booking status to the Database by rent and return request. For administrators, they can search the relative book name and got the source link from LibGen API, then store the searched information into the Database or delete the book information.

# Development View

**Concerns**: Organization of software modules

**Stakeholders**: software developers, manager

**Modelling** **techniques**: UML Deployment Diagram

The App singleton object is used to read the configuration file and keep track of all read-only application states, including the database connection, LibGen API connection, and a reference to the chain-of-loggers used by the rest of the classes to log application activity. The chain-of-loggers are implemented using a chain-of-responsibility pattern. Each logger instantiates and references the next logger according to whether the configuration file indicates the logger should be used. The template method pattern is used to ensure that the logic for calling the next handler can be defined once in the superclass (Logger) and re-used in the subclasses without using the call super anti-pattern.

The Model-View-Controller pattern is used to handle user interaction, presenting console menus and returning user input to the controller. The controller executes view methods to obtain user input and calls the model functions to handle any database interactions. The view has no state in terms of instance or class variables, so all of its methods are implemented as static methods to recognize and help enforce this property.

The foundation book management class is a parent class of the subclasses with a decorator. It works as an interface. And the subclasses inherit the parent class and have a specific function for each of them. The decorator pattern defines an interface, and the interface doesn't know what functions the subclass has. It's easy to expand and manage all of the functions belonging to the parent class. See Figure 3 for the UML Development View Diagram.

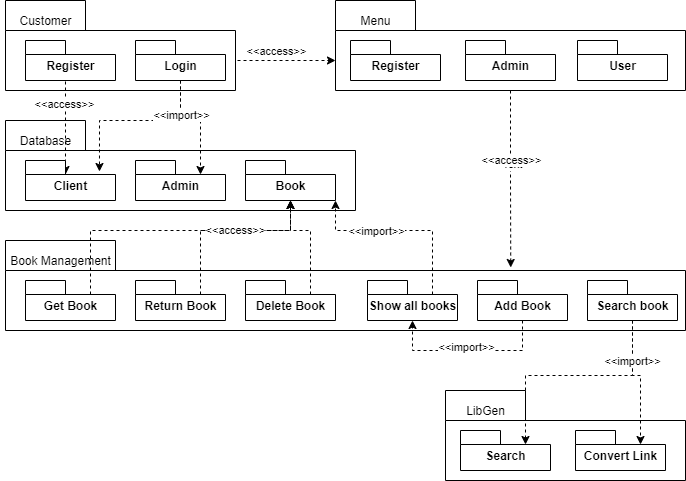


Figure 3 Development Viewpoint Diagram

# Process View

**Concerns**: Runtime communication

**Stakeholders**: software architect

**Modelling** **techniques**: UML Sequence Diagram

The essential runtime communication between the application and the Redis Labs cloud database is between the application and the LibGen API. When administrators log in to the system, they can search and get back the book information from the API. In the meantime, the mirror link should be converted to a downable link. Then, the subsequence of the event is to add the book into the Database. See Figure 4 for the UML Process View Diagram.

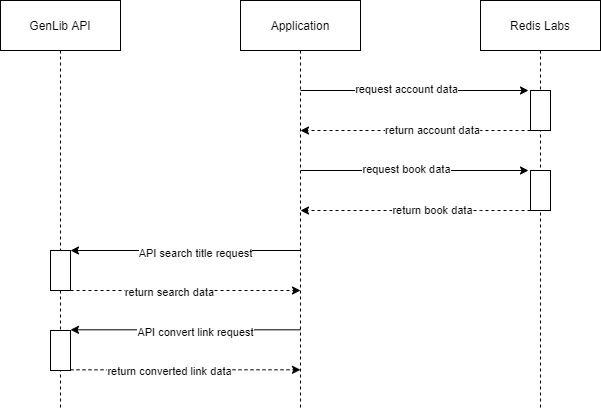


Figure 4 Data Exchange Sequence

In other words, the MVC pattern carries out significant application functionality. There are views for user and administrator interface, a model for database access, and a controller for communication between view and model and book management logic.

The MVC pattern involves the user being displayed a view by the controller, where they can choose options and type data sent to the controller. The controller can carry out the book management logic that is model functions. For instance, the search book request is optionally presented to the user until they successfully log in and decide to do it. The model makes multiple requests to Redis Labs to represent each piece of book data.

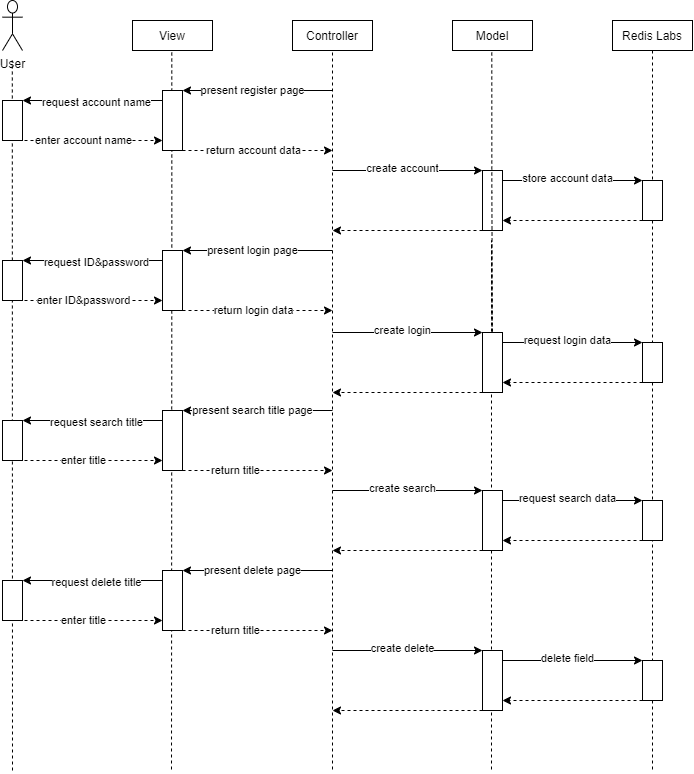


Figure 5 MVC Sequence

# Logical View

**Concerns**: Functional requirements

**Stakeholders**: End-user, software architect

**Modelling** **techniques**: UML Activity Diagram

Administrator has admin menu which supply the highest authority to allow them to manage the increase and decrease of types and quantities in the library. Users should log in to the system to go forward to the correct interface. However, the system shall allow the clients to explore the books in the Database, rent and return books, and take a record into the Database for regular users. And users can borrow and return electronic books in the library system.

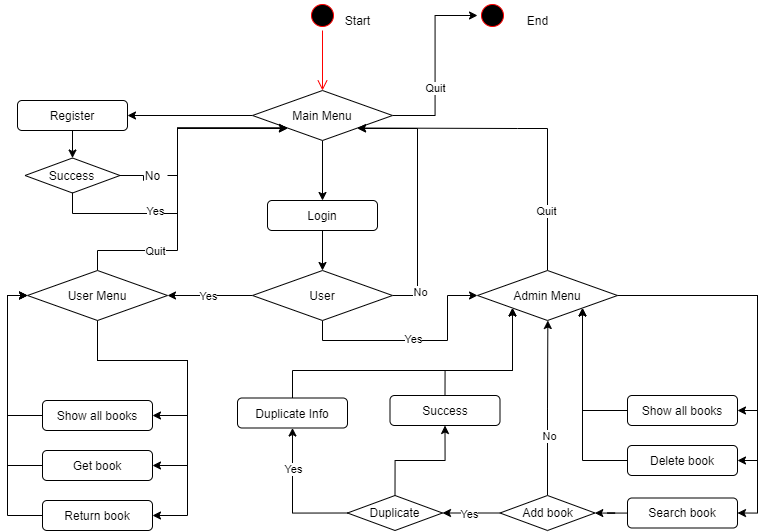


Figure 6 Logical Viewpoint

# Database Schema

Redis is a key-value database that stores data as keys associated with values. One possible value is a hash, which is itself a set of keys and values (however, the keys are called fields in this case to distinguish them). The database schema used in McLibrary is described in the table below.

|  |  |
| --- | --- |
| Key | Value |
| Account | The client account can be created through the console except for the administrator account.   |  |  | | --- | --- | | **Field** | **Value** | | **admin** | admin | | **user** | user\_password | |
| Books | The book's value in the Database has a format requirement as title, author, year, link and status. The book status represents the book is returned (valid) or rent (invalid).   |  |  | | --- | --- | | **Field** | **Value** | | **Book1** | Title, author, year, link, state | | **Book2** | … | |
| Log | A hash is containing log entries the application has created. Each field in the hash is a timestamp for the log message, and the value is the log message content. |