Group 4 Final Presentation

Jose Alvarez, Emmanuel Aurelio, Alyssa Lian Bacay, Tyler Blicharz, Fatima Pepino, Alyssa Simon

The Problem

The University of Riverside (UoR) is currently working with an outdated library system- card catalogs. Each book has multiple copies of cards printed for them with certain information- such as the title, author, ISBN number, aisle number, etc. To find a book requested by a patron, librarians would have to comb over copies on copies of book cards. Now, UoR is looking to update and computerize their database management system.

You have been hired to develop UoR's Library Management System using object-oriented software engineering principles. The system should allow library staff to manage the library's collection of books, as well as patrons' borrowing records.

- The system should allow library staff to add, update, and delete books from the library's collection.
- The system should allow library staff to add, update, and delete patrons' borrowing records.
- The system should allow the library staff to search for books by title, author, or genre.
- The system should track the availability of books.
- The system should keep a record of all borrowing transactions and generate reports on library usage.

Requirements

Business Model

The University of Riverside's library aims to provide students access to information and resources. It is funded by the school. Some of its resources include book lending, community building services, workshops, and more. In the case of UoR, the librarian(s) oversee all relevant activities. For the purposes of this project, we are only concerned with the book lending operations.

Business Model pt. 2

How does book lending work?

All the library's books are organized by genre, which are then broken into aisles. A patron is welcome to browse the aisles and choose a book at their own pace. However, they are also able to approach the librarian and request specific books, which the librarian can then search up.

From there, the patron will be able to check books out via the librarian. When they are finished, they will return the books to the librarian, who will then mark the books as returned.

Business Model pt. 3

A patron can only borrow up to two books at a time.

If a patron does not return a book by its due date, the book is considered overdue and a fine is charged to the patron's account.

If the patron has an overdue book, they will not be able to borrow another book until the late book is returned.

If a patron loses a book, they will also be charged a fine.

GLOSSARY

Inventory: collection of games that are available for checkout by patrons.

Patron Account: an account a patron signs up for that allows them to borrow games from the inventory

Patron: person who has an account and is authorized to check out games.

Due date: the date which a borrowed book must be returned to the store.

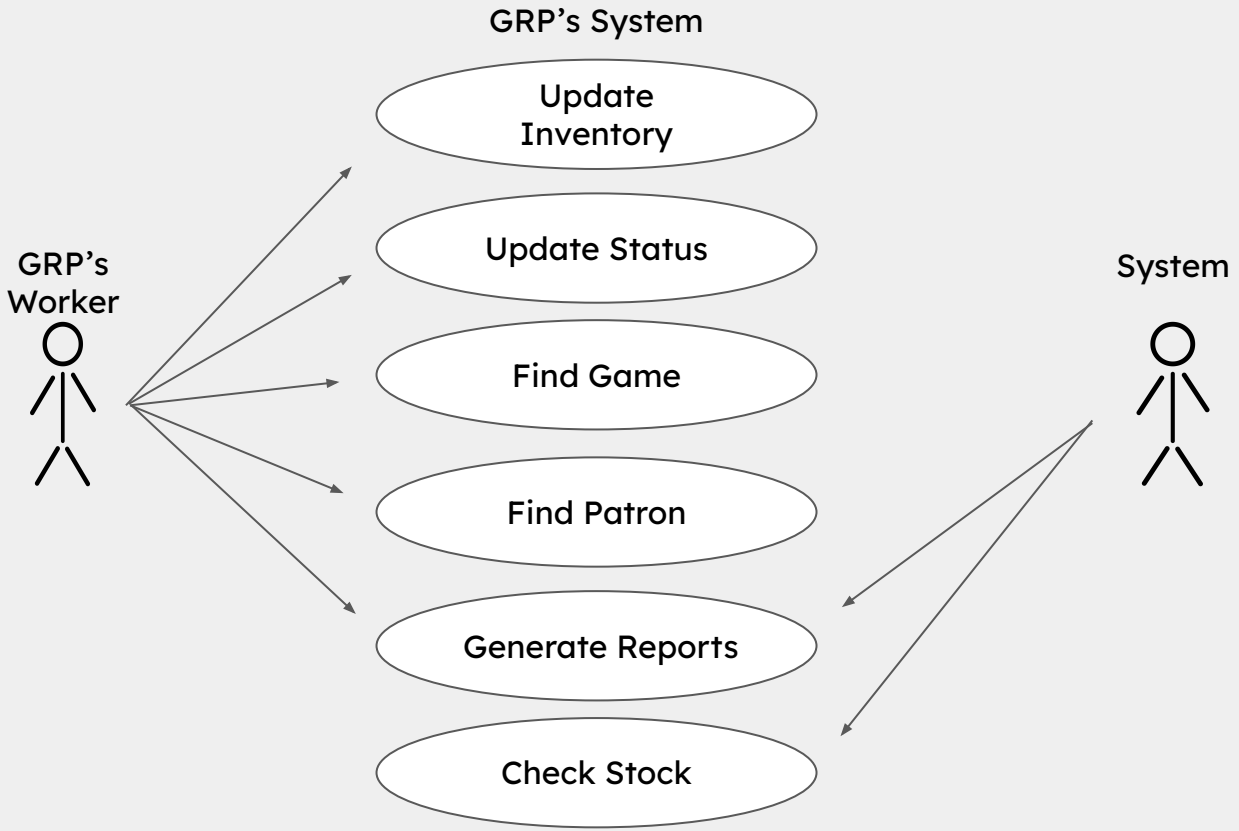
Overdue: a game that is not returned by the due date.

Admin account: privileged user account that allows staff to manage system.

Availability: the status of a game indicating whether it is currently available for checkout.

Fine: fee assessed to a patron for a late return or failure to return a borrowed game.

First Iteration UML Diagram



Update Inventory

Brief Description

The *Update Inventory* use case allows the workers to add or delete games from their records.

Step-by-Step Description

1. The worker can add a new game to the database.
    a. To add the game, the worker needs to enter the title of the game, the developer, the ISBN, the console, and the aisle where the game is found.
2. The worker can also remove games from the database.

Update Status

Brief Description

The *Update Status* use case allows the worker to update whether a game is available, checked out, lost, or overdue through an admin account.

Step-by-Step Description

1. From the Find Game use case, the worker chooses to edit the status of the game OR scans the games code and is prompted to edit the status of it.
    a. If a game is to be checked out :
        i. The worker cans/pulls up patrons account.
        ii. From the date of the checkout, the due date of the game is calculated and added to the system.
        iii. The status of the game is considered “checked out” and is temporarily removed from the count of available game.
    b. If a game is to be returned :
        i. The game is returned to the count of available game.
    c. If a game is lost :
        i. A fine is charged to the patron’s account.
        ii. The game is permanently removed from the count of available games.
    d. If a game is overdue:
        i. A fine is charged to the patron’s account.

FIND BOOKS

Brief Description The *Find Book* use case allows the library staff to search for a book by its title, author, genre, or ISBN and return all relevant information.
Step-by-Step Description 1. Librarian searches for a book in a search bar with either title, author, genre, or ISBN. 2. The system returns all relevant information of the book, including title, author, genre, ISBN, and location in the library. 3. In the search result, if the patron asks to check the availability of the book, there will be a button prompting “Check Availability”.

FIND PATRON

Brief Description

The *Find Patron Records* use case will allow the librarian to scan in the patron's ID, check their profile/records.

Step-by-Step Description

1. The patron gives the librarian their student of staff ID.
    a. If they do not have their ID card on hand, they will give their ID number.
2. The librarian scans the ID card or searches the ID number.
3. If the patron exists, their profile and borrowing history shows up.
    a. The current book(s) borrowed show up, if any.
    b. If there's a relevant late fine for a borrowed book, that also shows up.

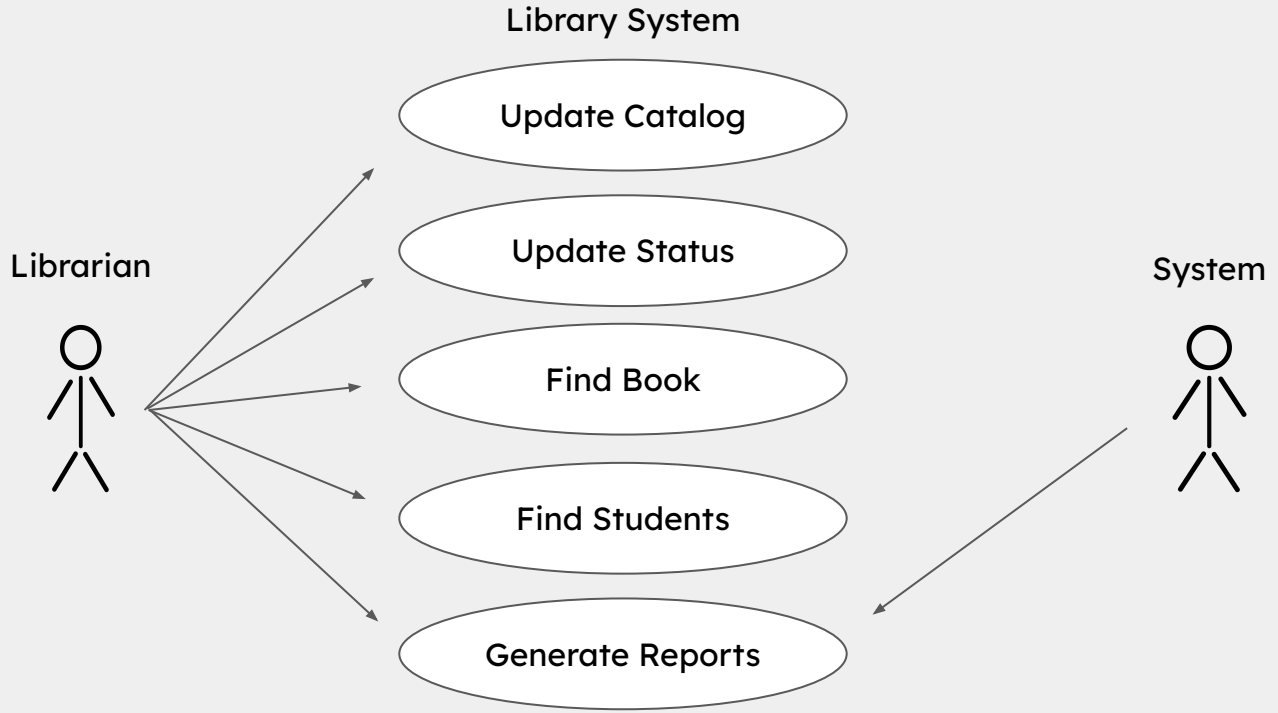
Generate Reports

Brief Description The *Generate Reports* use case allows the librarian at the end of the day to see the total number of books that were borrowed, returned, and marked overdue.
Step-by-Step Description 1. By the end of the day, each book borrowed and returned will be logged into the library database. 2. If a patron's book return date was today, and they did not return, their account will have the late fee applied too.

CHECK AVAILABILITY

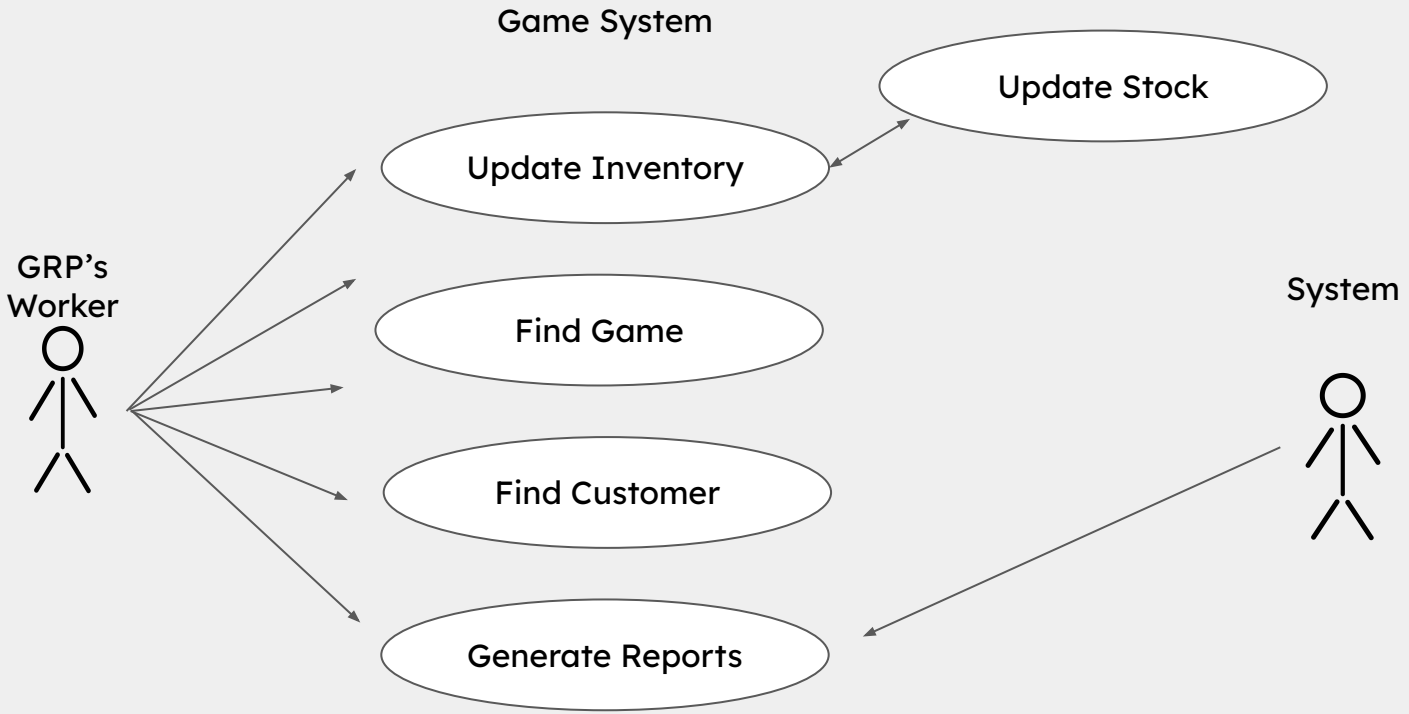
Brief Description The *Check Availability* use case allows the librarian to check the availability of a book that a student is asking for.
Step-by-Step Description 1. From the Find Book use case, the librarian clicks on the “Check Availability” button. 2. Library management system retrieves information on the book’s availability from the database and displays the information on the library’s screen.

Second Iteration UML Diagram



The check availability use case was found unnecessary- its steps were absorbed by the find book use case.

SECOND ITERATION UML DIAGRAM



The Update Status use case was found to be an extension of the Update Catalog use case.

ANALYSIS

FUNCTIONAL MODEL

Patron Finds Book In Library

a. The patron walks into the library, whether they know what they want or not.
b. The patron walks around the library, where all books are sectioned off into different aisles, depending on the genre.
c. The patron finds a book that they want, *The Hunger Games* by Suzanne Collins in the Sci-Fiction selection on Aisle C.
d. The patron finishes looking around and then brings it to the front desk.
e. Library staff scans the patron's student card.
f. The librarian inputs the borrowing book's ISBN into the system.
g. The librarian checks the book out, which marks the borrowing date in the system.
h. The librarian gives the patron their receipt with a return date on it.

Functional Model

Patron Asks Librarian For Book

a. The patron enters the library and the patron asks the librarian for The Hunger Games.

b. The librarian opens the find book interface.

c. The librarian searches for the book through the library system via title and author name.

d. The system searches through the library's catalog for The Hunger Games.

e. The system finds The Hunger Games in its records.

f. The user interface returns to the librarian the location of The Hunger Games and current availability.

g. The librarian tells the patron the aisle of where the book is located.

FUNCTIONAL MODEL

Adding Books into Catalog

a. The library receives new books to add to inventory, via new releases, donations, etc.

b. The librarian now needs to add the new books to the database.

c. The librarian opens the interface, and clicks add *New Book.*

d. The librarian then enters the title of the book, the author, the ISBN, and the genre.

e. From there, the librarian is able to find out what aisle where the book will be found, and enters that into the database.

Functional Model

Library Does Not Have Requested Book

a. The patron enters the library and asks the librarian for The Hunger Games.
b. The librarian opens the find book interface.
c. The librarian searches for The Hunger Games via the title.
d. The system searches through the library's catalog for The Hunger Games.
e. The system does not find The Hunger Games.
f. The user interface returns to the librarian that The Hunger Games is not found.
g. The librarian alerts the patron that the library does not have The Hunger Games.

FUNCTIONAL MODEL

Deleting Books from Catalog

a. The librarian searches *The Hunger Games* by Suzanne Collins in the system.

b. The system searches through the library's catalog for *The Hunger Games*.

c. The user interface returns to the librarian the location of *The Hunger Games* and current availability.

d. The librarian chooses to edit the status of *The Hunger Games*.

e. The librarian removes it from the system.

FUNCTIONAL MODEL

Print Library Reports

a. The university requests a report on the number of books checked out on a certain date.
b. The Librarian opens up the interface to access the database
c. The Librarian accesses the data of the day that was requested
d. The data of that day is displayed.
e. The Librarian prints out the data
f. The Librarian sends the data to the university.

Noun Extraction

The **Library Management System** allows the **librarian** to keep track of all the **books** in the **library**. The **librarian** can add to or delete **books** from the library's **catalog**. They are also able to check out **books** when they are borrowed. The **system** can also be used to generate **reports** on library **activity**.

Nouns : Library Management System (LMS), librarian, books, library, catalog, system, report, activity

Exclude : librarian, library, LMS, activity, system

Candidate Classes : books, report, catalog

Class Responsibility Cards

Report Class
Responsibilities 1. See how many games have been checked out. 2. See how many games have been removed/added 3. See how many games have been returned. 4. See how many games have been issued as overdue. 5. See how many visitors have been at the GRP. 6. Email report.
Collaboration(s) 1. Books Class 2. Catalog Class 3. Patron Class

Inventory Class
Responsibilities 1. Search up a game title. 2. Check to see if a game is available. 3. See how many copies are available of a certain game. 4. See a number of games by a specific developer. 5. Adding a game 6. Deleting a game
Collaborations(s) 1. Game Class 2. Patron Class 3. Report Class

Class Responsibility Cards

Patron Class

Responsibilities

1. Retrieve the patron name.
2. Retrieve patron phone number.
3. Retrieve patron borrowing records.
4. Update patron borrowing records.
5. Pay late fine.
6. Waive late fine.

Collaboration(s)

1. Game Class
2. Inventory Class
3. Report Class

Book Class

Responsibilities

1. Retrieve the book genre.
2. Retrieve the book title.
3. Retrieve book ISBN.
4. Look up the name of the author.
5. Check where the book is located.

Collaboration(s)

1. Catalog Class
2. Report Class
3. Patron Class

Class Responsibility Cards 2nd Iteration

Catalog Class

Responsibilities

1. Search up a game title.
2. Check to see if a game is available.
3. Check where a game is located.
4. See how many copies are available of a certain game.
5. See a number of games by a specific developer.
6. Retrieve information on a game..

Collaborations(s)

1. Games Class
2. Patron Class
3. Report Class

Game Class

Responsibilities

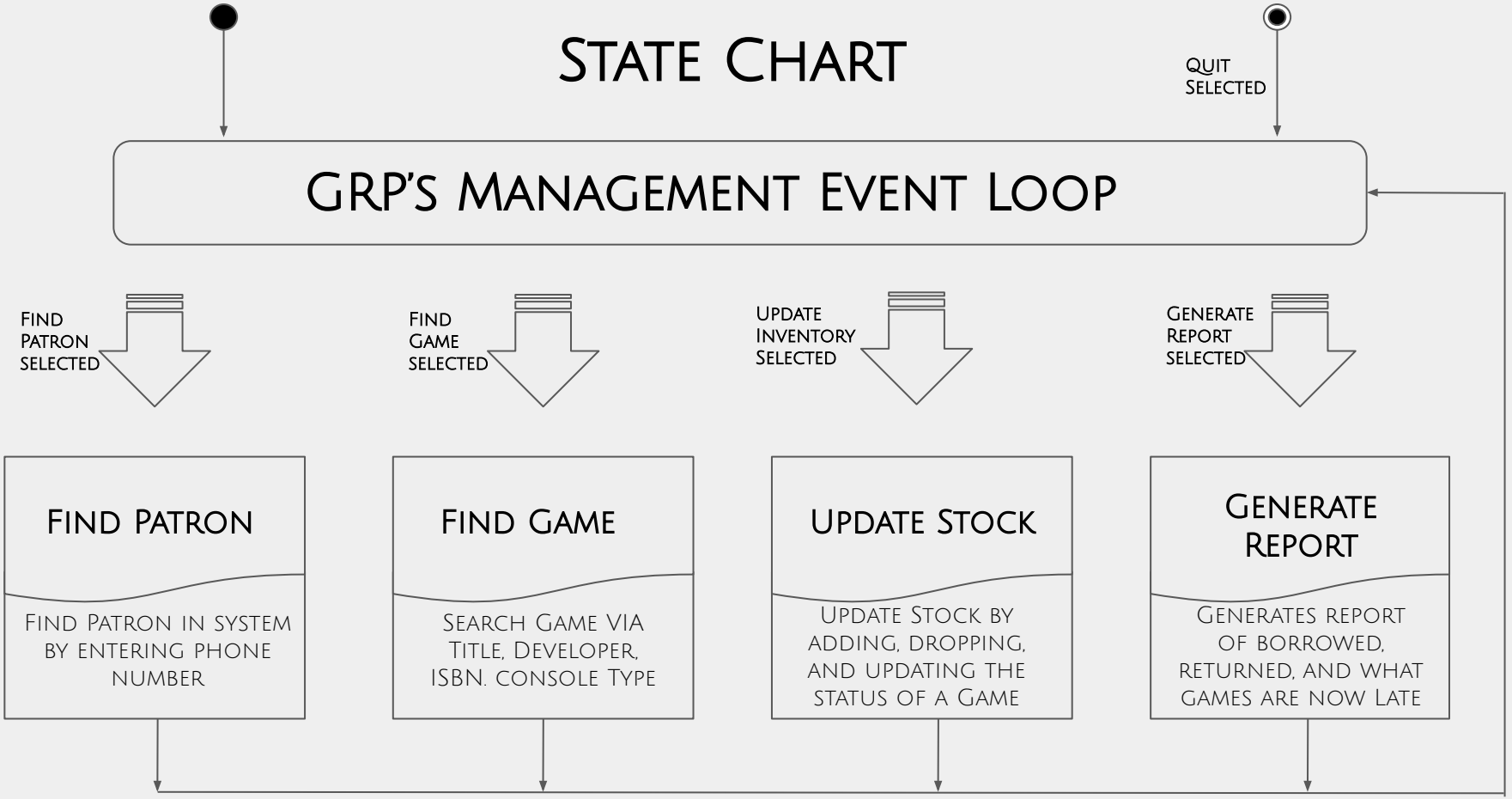
1. Store the game console type.
2. Store the game title.
3. Store game ISBN.
4. Add a game.
5. Delete a game.

Collaboration(s)

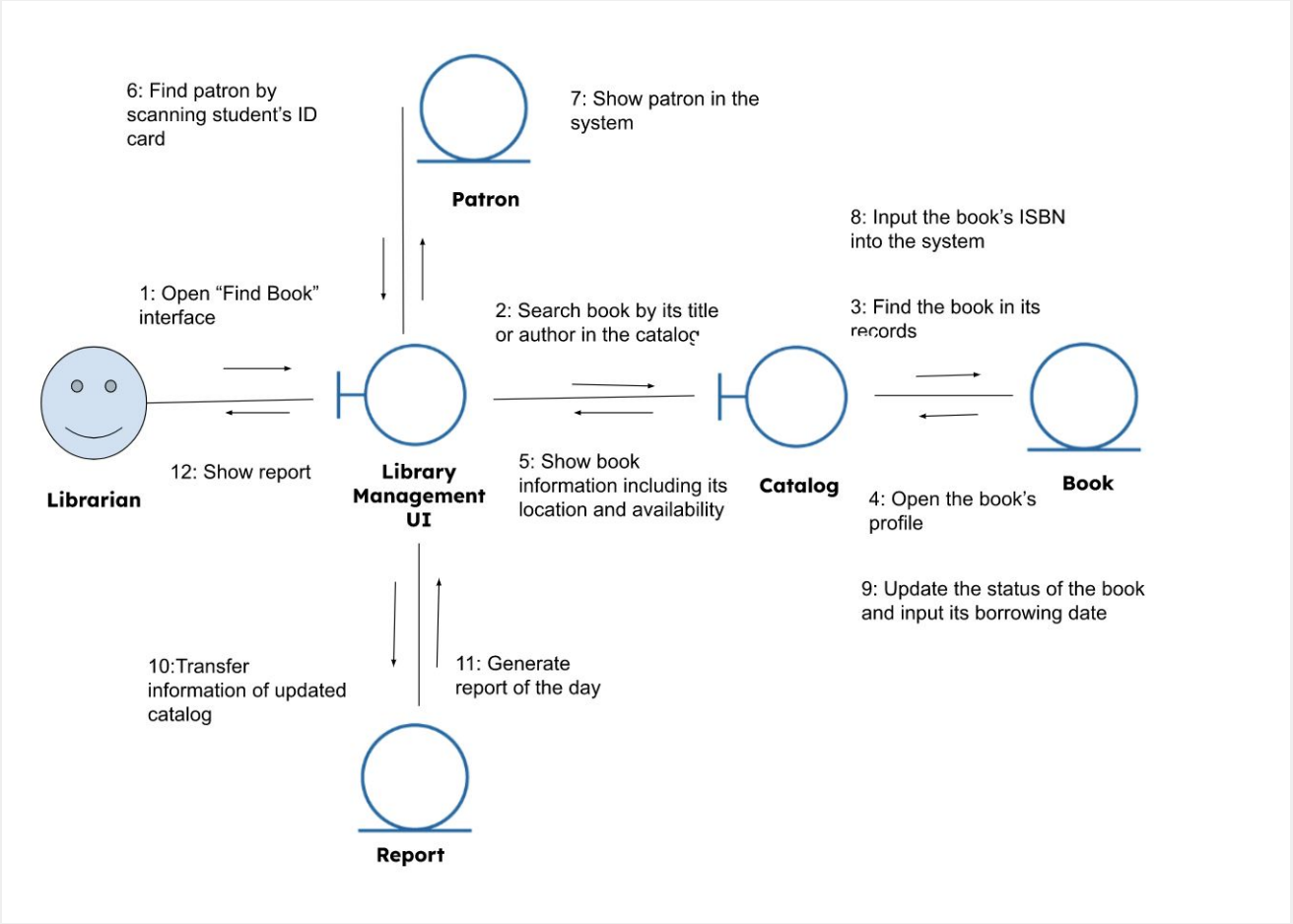
1. Inventory Class
2. Report Class
3. Patron Class

Changes were needed to ensure each class had **data coupling-**
The catalog class now can only retrieve data from the book class, where all the data will be stored.

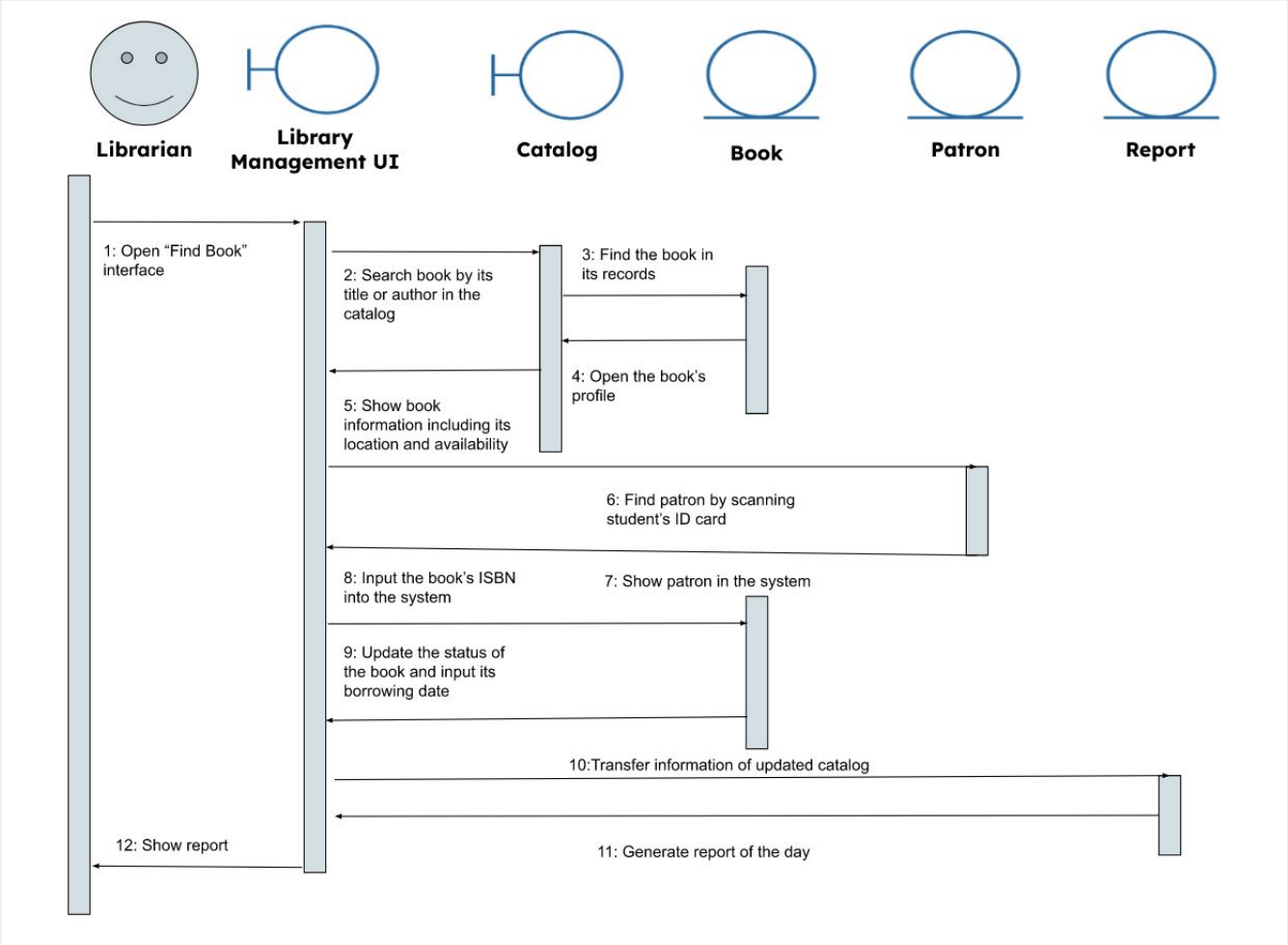
State Chart


Quit Selected
GRP's Management Event Loop
Find Patron Selected
Find Game Selected
Update Inventory Selected
Generate Report selected
Find Patron
Find Patron in system by entering phone number
Find Game
Search Game via Title, Developer, ISBN. console Type
Update Stock
Update Stock by adding, dropping, and updating the status of a Game
Generate Report
Generates report of borrowed, returned, and what games are now Late

Communication Diagram



SEQUENCE DIAGRAM



Design

PSEUDOCODE

```
public class book() {
        public array bookInfo[6];
        public matrix database;

        public static void addBook(title, author, isbn, genre, aisle){
                receive user input for book info (format as Title, Author, ISBN, Genre, Aisle)

                split string into bookInfo
                        set bookInfo[6] to 0

                if database contains bookInfo
                        bookInfo[6]++
                else
                        add bookInfo array to database
                        bookInfo[6] = 1
        }

        public static void deleteBook(title, author, isbn, genre, aisle){
                receive user input for book info (format as Title, Author, ISBN, Genre, Aisle)

                split string into bookInfo array

                for index in database
                        if bookInfo matches(true)
                                delete array from database
                                booksDeleted++
                        if at the end of the index and booksDeleted = 0
                                print “No match was found.”
        }

}
```

Code

Showcase