CS 3200 Database Design
Project Final Report

PriviteraDRizzoMRizzoN

0 - Table of Contents

1 - README for Creating and Running Project

Please note this readme exists as README.md in the top level directory submitted with the code. Some of the links will work better there. All paths within the project repo are relative to the top level directory. This is the one that contains the src/, docs/, and install/ directories within it.

Specifications & Prerequisites:

Libraries:

The only libraries needed are for the python backend of the project. They can all be seen within the requirements.txt file inside the install directory. Relative to the top level directory of the project: install/requirements.txt. However, there are scripts that can be run which install all of the libraries for you. Run install/setup.sh (Unix based Operating Systems) or install/setup.bat (Windows Machines). They will handle the creation of a python virtual environment with the libraries of the correct version.

Additionally, the Javascript code is using the JQuery Library uploaded within the project code itself. The user does not need to obtain it themselves. Similarly, a CSS styling library/framework for the web page called Bulma is used by the project. It is included within the project files included in submission.

Software:

MySQLWorkbench and MySQLServer are needed on the machine running the project in order to create the schema from the dump file (including its data, procedures, and functions). The python backend will connect to the database in order to create the user interface and obtain data. This can be accomplished the same way as throughout the semester with the complete dump file being run.

A download of Python Version 3.9. Please download it from this page: https://www.python.org/downloads/release/python-390/ and select the option at the bottom of the page matching the Operating System of the machine running.

Setup Steps:

1. (For Windows Only) Download Python3.9
   a. **Note:** Using a different and or earlier version of python will result in indeterminate results of the project. Possibly even the code not working. Please be sure your version is Python 3.9.
   b. Please run **py -0** in the command prompt / cmd to see a list of installed python versions.

        i. Please confirm that you see **3.9-64 *** or **3.9-64**. If not, the installation was not successful.
    c. If you are using Ubuntu or a Debian distribution, the above install will be handled by install/setup.sh.
2. Run setup scripts. Pick the correct one based on the operating system:
    a. **Windows: install/setup.bat**
        i. Note: this can be done through the command prompt or clicking on the .bat file using file explorer.
    b. **Ubuntu/Debian/Windows Git-Bash: install/setup.sh**
    c. This handles the complete setup of a virtual environment with the correct libraries/packages for the project.
    d. The virtual environment will live in the **cs3200-venv** directory relative to the top level of this project
    e. Depending on the operating system, the exact location of the python interpreter will change. However, the start scripts will handle that, and the user does not need to figure it out.
3. If testing the late fee calculation functionality:
    a. After running the dump file to create the schema, please run the sql script at **/src/database/show_overdue_functionality.sql.**
    b. The script adds another procedure to the database which can checkout a book on a specific day. It then checks out a book on a specific day in the past with a very low checkout length. The book is checked out by user nickrizzo.
    c. This is added as an additional feature because the group believed the existence of a procedure to checkout a book on a specific day was inappropriate. The design of the application is such that a user checks out a book the moment they are checking it out. There should NOT be an interface by which they can checkout a book on a different day. Hence, this procedure could not be included in the main dump file. However, for the ease of validating that our late fee calculation is correct, the group added the procedure in this limited capacity.
    d. If the steps in View Profile user flow are followed for the nickrizzo user, the late fee will be displayed in the table and can be verified.

Running/Starting the Application Server:

1. **Windows: start.bat**
    a. Using the command prompt. This file is located at the top level of the submitted project code.
    b. Note: the application can also be started by clicking on the file itself in the file explorer.
2. **Ubuntu/Debian/Windows Git-Bash: ./start.sh**
    a. Using some form of command line terminal. This file is located at the top level of the submitted project code.

3. Each script will call \`main.py\` using the python virtual environment setup.
4. Open the landing page to begin interacting with the frontend / client side
    a. Most likely, **http://localhost:8080/**
    b. When the server starts it will also print this url in case port 8080 is taken.
5. **NOTE On Windows Machines:**
    a. The first time the program is run, Windows Firewall might block the application, please click "**Allow Access**". This issue should not happen again.
    b. When endin g the program please do \`Ctrl+C\` then respond to "**Terminate batch job (Y/N)?**" with \`y\` and click enter.

2 - Technical Specifications

To run this application, the user must have Windows 10 or Ubuntu/Debian Linux Distributions. Users must also have python 3.90 installed (see Setup Steps: for installing it). Additionally, an internet browser such as Google Chrome must be installed. Chromium based browsers are all confirmed to work, but exhaustive testing was not done on other browsers such as Firefox. MySQL Server and Workbench 8.0 CE must also be installed (as per the instructions given by the course CS3200).

Due to the fact there is a lot of code in this project, the group decided to modularize the python structure into classes that manage their respective tasks. Hence, all functionality that involves directly interacting with the database (stored procedures, stored functions, etc...) are all contained within the src/backend/db_manager.py file (and its DB_Manager class). This class is inherited by the User_Manager class, which in-turn is inherited by the WebApp class found in main.py. Calls to the database interface functions can be seen in main.py as self.<function> where function is defined within DB_Manager. For example, checkout_book, search_for_book, etc… Thus, if a grader wants to see the exact point where the python code interacts with the database, please look through db_manager.py. Aside from the stored procedures and functions within the schema itself, the python backend does not interact with the database at ALL. Therefore, by looking at the stored procedures and functions within the provided dump file (and then the generated schema), it is possible to see all of the interfaces to the database available to the backend.

Understanding the Project File Structure:

The file structure of the project source code (once the setup scripts are run) is as follows:
cs3200-venv/
docs/
install/
src/
README.md
start.bat
Start.sh

The directories and or files needed to understand how the project runs are described below. Please do not modify this file structure when examining the project, as various directories interact with one another with the precondition that this structure is maintained.

Virtual Environment Directory (cs3200-venv)

The exact format of the venv directory changes based on the operating system. Essentially, within this directory, a python interpreter with the correct python version (3.9) and correctly versioned packages will be stored within it. If the operating system is Windows this will be *cs3200-venv/Scripts/python.exe*, and if a Linux based Operating System *cs3200-venv/bin/python*. The python interpreter calls the *main.py* file in *src/backend*. This knowledge is not required as running the *start.sh* or *start.bat* scripts will determine the correct pathing for the user and run the program.

Install Directory

This directory contains the *requirements.txt* file which describes all non-standard python modules needed for the project to be run successfully. The file also specifies which version they should be. The *setup.bat* and *setup.sh* files can be run based on Operating System to create the virtual environment directory and setup the project run system.

Src Directory

This directory contains all source code needed by the project to run successfully. It has three main subdirectories: *database, backend, and frontend*.

The database directory will almost never get used aside for internal testing. The one exception is testing the late fee calculation functionality, in which case please see Step 3 of the Setup Steps Section of this report.

The backend directory contains all python code needed by the project to perform its complete functionality. The structure of this directory should NOT get modified at anypoint. The project server code is run by starting *main.py*, but running one of the start scripts will do this for a user. *main.py* imports various 3rd party libraries (obtained via *requirements.txt* and the setup scripts) and the project defined programs contained within the *src/backend/* directory. Moreover, the Web Application serves frontend code (Javascript, HTML templates, and CSS) relative to the location of *main.py*.

The frontend directory is broken down further into 2 subdirectories, *templates* and *static***.** Templates contain all of the html templates which are served and rendered by Flask and the Web Application in main.py. It should be noted that many of these files are seemingly empty when their corresponding URL's are very full. The reason is that using Flask functionality such as WTF-Forms (Flask's method for creating HTML forms) and flask-Table automatically generates the HTML and CSS code and inserts it into the HTML templates at time of rendering using Jinja2. For example, in *searchResult.html*, *{{ result_table }}* is a Jinja2 variable. When the */search_book* route is given a book to search for, it queries the database, formats the results into a Flask Table, and uses the following Flask API (*render_template*) to insert the Flask Table into the Jinja2 variable *result_table* as the page is rendered.

```
return render_template("searchResult.html",
                       title="Library Search Results",
                       book_title_searched=form.book_title.data,
                       result_table=search_table)
```

Such use of Jinja2 and built-in Flask functionality can be seen throughout the project.

The *frontend/static* directory contains all CSS, Javascript, and image files needed to be served by the python backend/server. This is why the directory structure should not be modified, the serving from python will break if these directories are not in the correct locations.

Please note that there are very few CSS files. That is due to the usage of the CSS Bulma framework. The group cannot take credit for the *src/frontend/static/css/bulma.min.css* file. It is the file given by the open source Bulma project framework. Its GitHub can be accessed at https://github.com/jgthms/bulma, and the min file was obtained from https://cdn.jsdelivr.net/npm/bulma@0.7.2/css/bulma.min.css . It is simply included in the *base.html* file. That file is in turn extended by every other HTML file - see the first line of every html file: *{% extends "base.html" %}*. By finding appropriate ways to use the Bulma CSS classes and styles, the project maintains a consistent color scheme on every webpage.

Likewise, there are very few Javascript files in this project. This is because the project relies on switching pages frequently and does not require Javascript to modify small aspects of a static page. The few Javascript files do use the common external library called jQuery. The library is included in the project at *src/frontend/static/js/extern/jquery-3.5.1.min.js*. Once again, the group does not take any credit for this Javascript file. Its GitHub can be found at https://github.com/jquery/jquery and the min file used in this project was obtained from https://code.jquery.com/jquery-3.5.1.min.js. The Javascript code used in this project is mainly for controlling the upper navigation bar and hamburger menu. Additionally, it is responsible for limiting the options in the registration drop down menus. When a given Library System is selected, a database procedure is called to obtain names of the libraries that exist within the system. The Javascript code then modifies the Library dropdown to contain only those libraries.

3 - Current Conceptual Design

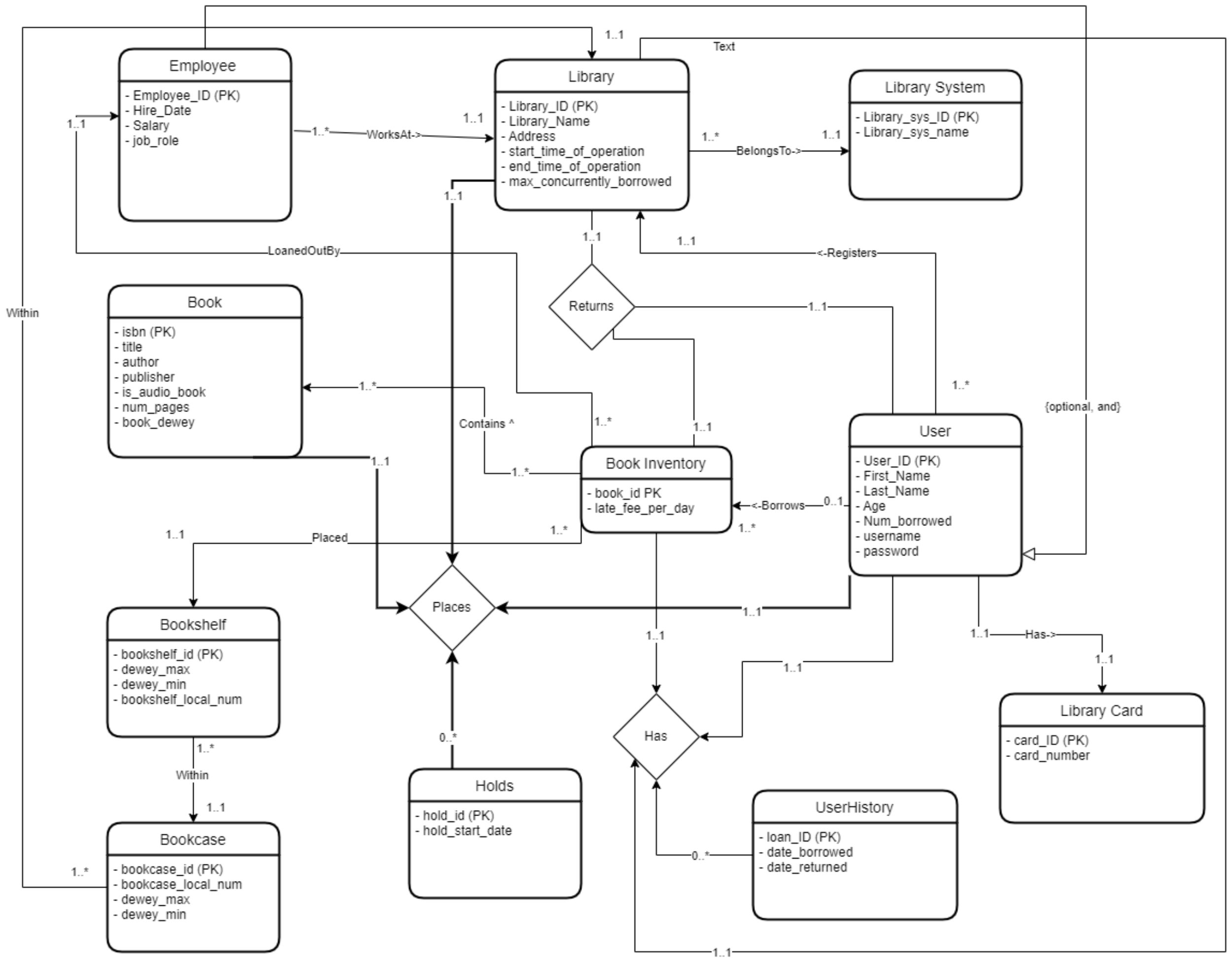


Figure 1: UML Diagram of the Library Management System

4 - Logical Design

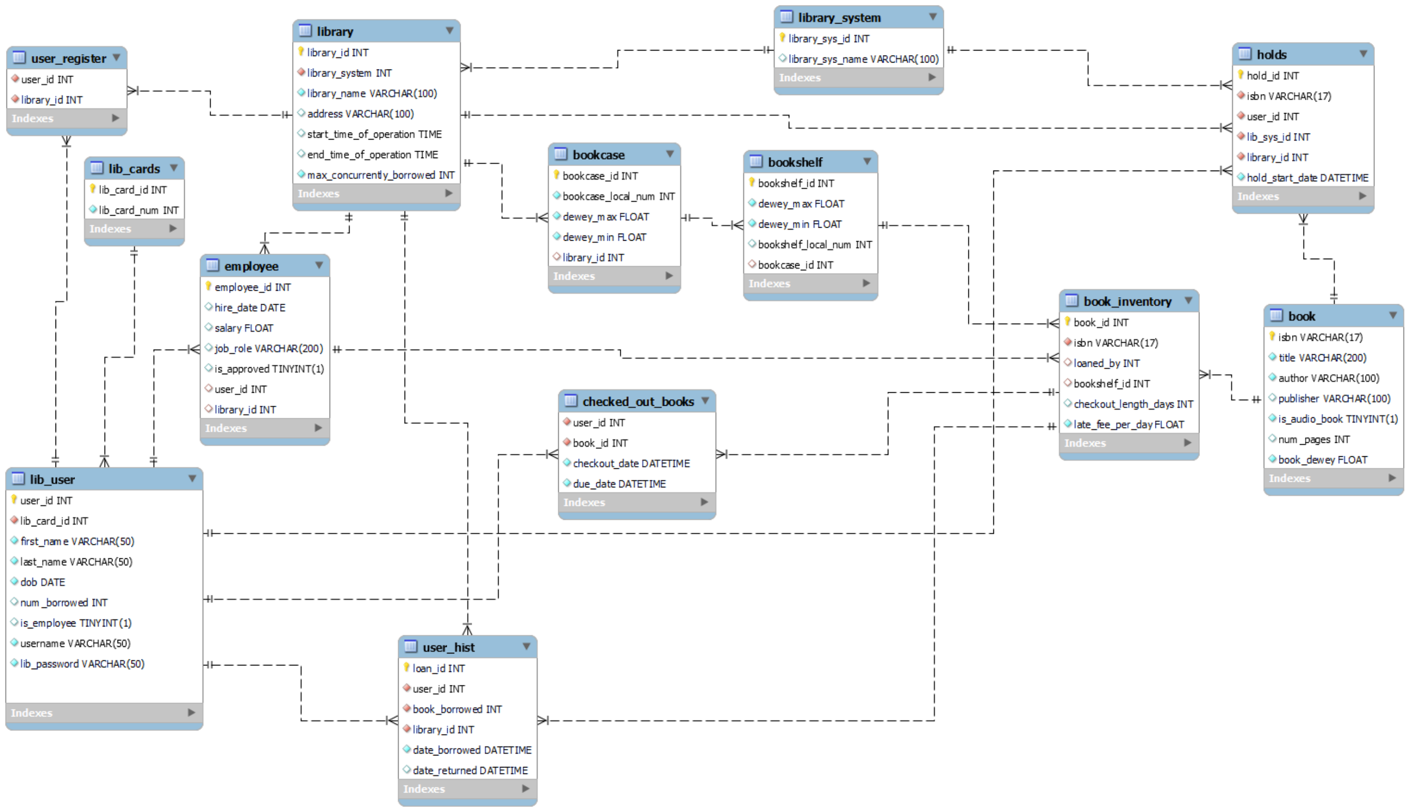


Figure 2: Logical Design Diagram of the Library Management System Schema

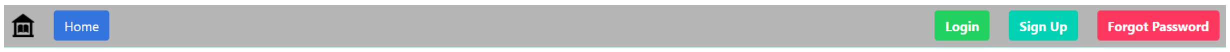
5 - User Flow of System

There are a lot of actions that can be taken as a user. The options change if that user is also an employee of a given library. The user flow will be broken up into 3 parts: flow for registering as an employee/user, all ways to interact/flow with the system as a user, all ways to interact/flow with the system as an employee.

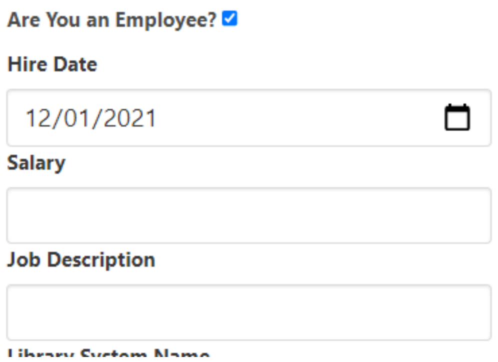
Login Flow

This assumes the account being logged into has already been created. Please see Registering for details on how to do that. The system comes with some preloaded users (also employees). Their login information can be seen in Appendix A and Appendix B.

Registering a User/Employee

- Go to http://localhost:8080/register (can get there via the Sign Up Button)
    - 

    - Fill out all the fields and click Register. Validation will inform the user if there are errors that must be corrected.
    - Note: if registering as an employee, check the box, more data must be entered.



Are You an Employee? ✓

Hire Date

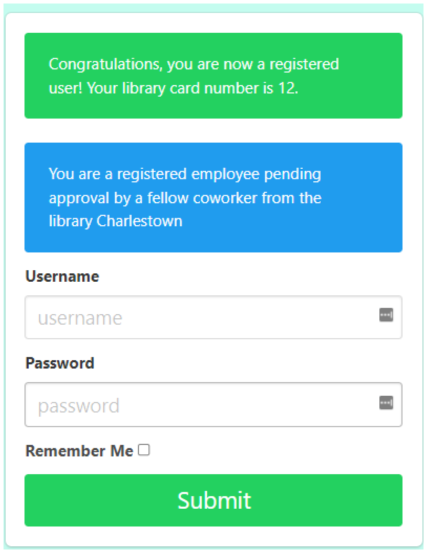
12/01/2021

Salary

Job Description

Figure 3: Registering as Employee Specific Fields

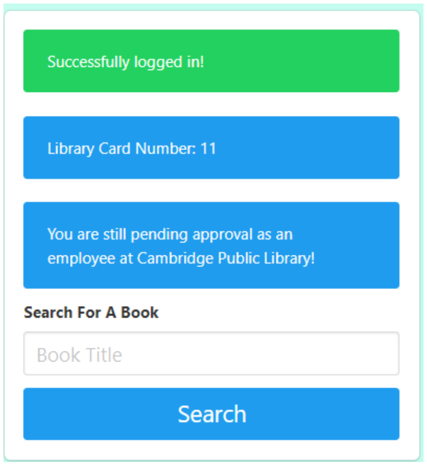
- If the registration is successful you will be taken back to the sign in page.


Congratulations, you are now a registered user! Your library card number is 12.

You are a registered employee pending approval by a fellow coworker from the library Charlestown

Figure 4: Example After Registration Pending Approval as Employee

  - The display regarding approval will ONLY appear if registering as an employee.
- To be approved as an employee, a fellow employee from the same library (not system) must login and approve you. Please see Appendix A for a list of employee logins for each respective library. If another employee is registered for any of the given libraries, simply login to the corresponding pre-approved employee account for that library, and approve (or deny) that new employee.
- The flow is register as employee->login as approved employee->approve new employee ->login as new employee->perform employee actions
- The flow is register as employee->login as approved employee->deny new employee ->login as new employee->(only have access to standard employee actions)
- If the new employee attempts to login while they are still pending, they will receive the following message:


Successfully logged in!

Library Card Number: 11

You are still pending approval as an employee at Cambridge Public Library!

Figure 5: Example View of Logging in as Pending Employee

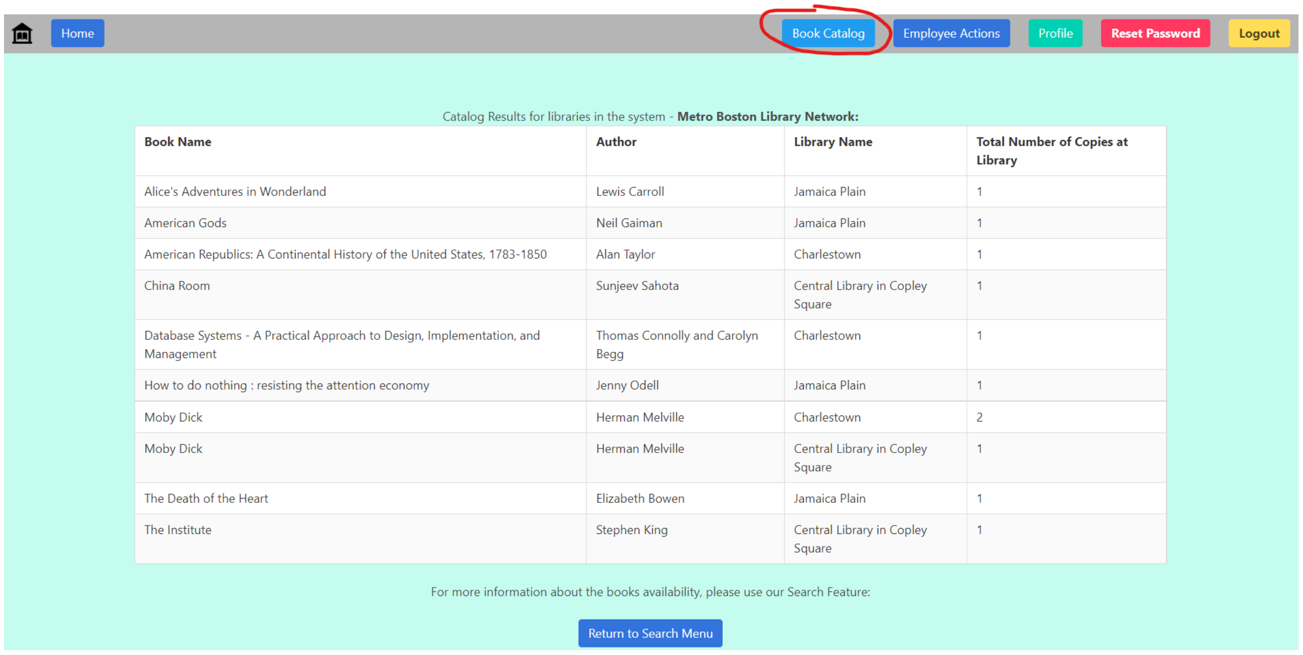
Forgot/Reset Password

To reset or change a user's password one needs to know their library card number and username. Ideally - and in the real world - a password reset would only be allowed with email validation, but since that is out of the scope of this project the group went with a simpler and less ideal solution. To reset one's password, one must simply click the red "Forgot Password" or "Reset Password" (depending on if they are logged in or not) button on the top right corner of the navigation bar.

Potential User Actions/Flows

These are actions that can be taken by ALL users (regardless if they are an employee or not) after registration. Login is required before any of these actions. The website itself will force a redirect to login if a user attempts these actions first. For grading and testing purposes, these screenshots are taken by some of the default user/employee accounts made during schema creation so that they are easily replicable if desired. Specifically, the users are "nickrizzo" and "holdacct" who both belong to "Charlestown" library within the "Metro Boston Library Network" library system (see Appendix A and Appendix B). Feel free to create a new account(s) belonging to this library and system to achieve the same results!

Viewing Library Catalog



Catalog Results for libraries in the system - **Metro Boston Library Network:**

Book Name	Author	Library Name	Total Number of Copies at Library
Alice's Adventures in Wonderland	Lewis Carroll	Jamaica Plain	1
American Gods	Neil Gaiman	Jamaica Plain	1
American Republics: A Continental History of the United States, 1783-1850	Alan Taylor	Charlestown	1
China Room	Sunjeev Sahota	Central Library in Copley Square	1
Database Systems - A Practical Approach to Design, Implementation, and Management	Thomas Connolly and Carolyn Begg	Charlestown	1
How to do nothing : resisting the attention economy	Jenny Odell	Jamaica Plain	1
Moby Dick	Herman Melville	Charlestown	2
Moby Dick	Herman Melville	Central Library in Copley Square	1
The Death of the Heart	Elizabeth Bowen	Jamaica Plain	1
The Institute	Stephen King	Central Library in Copley Square	1

For more information about the books availability, please use our Search Feature:

Figure 6: View of Library Catalog

Once logged in, a user can visualize all the books that exist in their library system (not just their individual library). Since the current search capability is limited to exact string matches, the current best way to actually find and checkout or place a book is to find it in the catalog first and use the exact "Book Name" in the search box. Regardless, the catalog is a great way to visualize the available books in a user's library system in a concise and consistent manner.

Searching For a Book

In order for a user to actually checkout or place a hold on a book they have to first find the book using the web app's search feature. To search, navigate to the homepage (using the "Home" button or going to the root of the application) and simply type the name of the book in the search input box. As stated previously, the best way to search for books is to find it in the book catalog in order to copy and paste the title exactly into the search box.

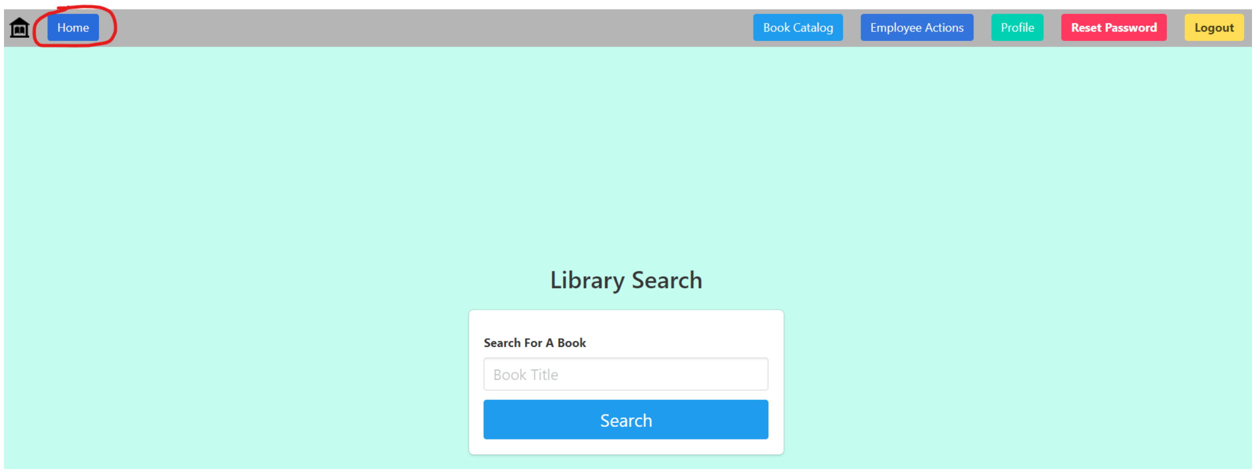
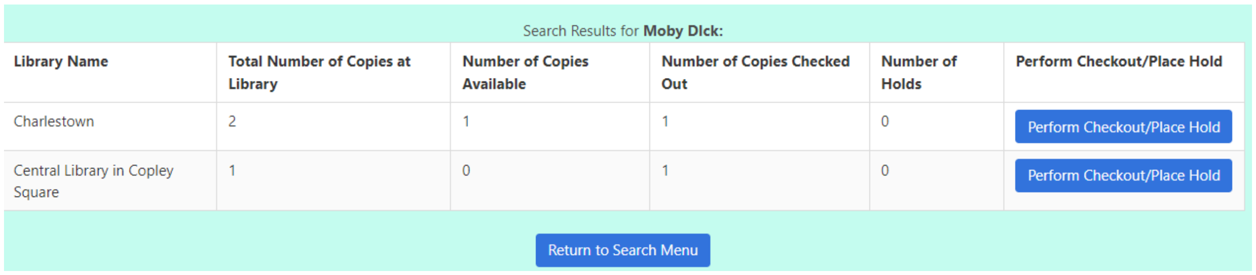


Figure 7: View of Main Search Page

Checking Out / Placing a Hold on a Book

To check out a book, it must first be searched for (see Searching For a Book). The search results table will provide the option to check out a book or place it on hold if it was found. For example, in Figure 8, there are 2 libraries in the library system the "holdacct" user belongs to that have copies of "Moby Dick". Charlestown Library has 1 copy left, so clicking checking out would result in the book getting checked out (see Figure 9). However, Central Library in Copley Square has all of its copies checked out already, so clicking checkout would result in a hold getting placed (see Figure 10). When a user who has a copy of Moby Dick from Central Library returns it, holdacct user will automatically have it checked out for them. Note that both Figure 9 and 10 are the results of clicking the blue button on the right most column, and are mutually exclusive.

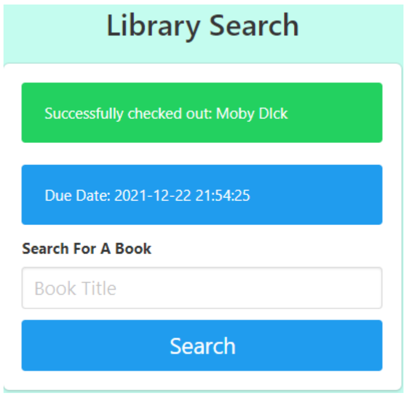
Two copies of the same book cannot be checked out / placed on hold. For example, if holdacct placed themselves on hold (Figure 10), and then tried to checkout the book from Charlestown (like in Figure 9), it would fail and the messages in Figure 11 would be displayed.



Search Results for **Moby DIck:**

Library Name	Total Number of Copies at Library	Number of Copies Available	Number of Copies Checked Out	Number of Holds	Perform Checkout/Place Hold
Charlestown	2	1	1	0	Perform Checkout/Place Hold
Central Library in Copley Square	1	0	1	0	Perform Checkout/Place Hold

Figure 8: View of Search Results for Moby Dick On holdacct User

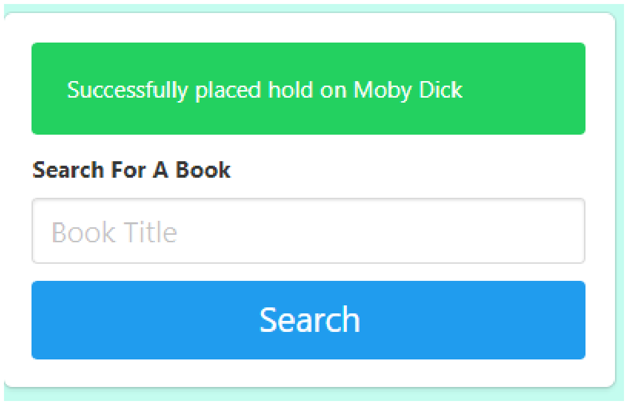


Library Search

Successfully checked out: Moby DIck

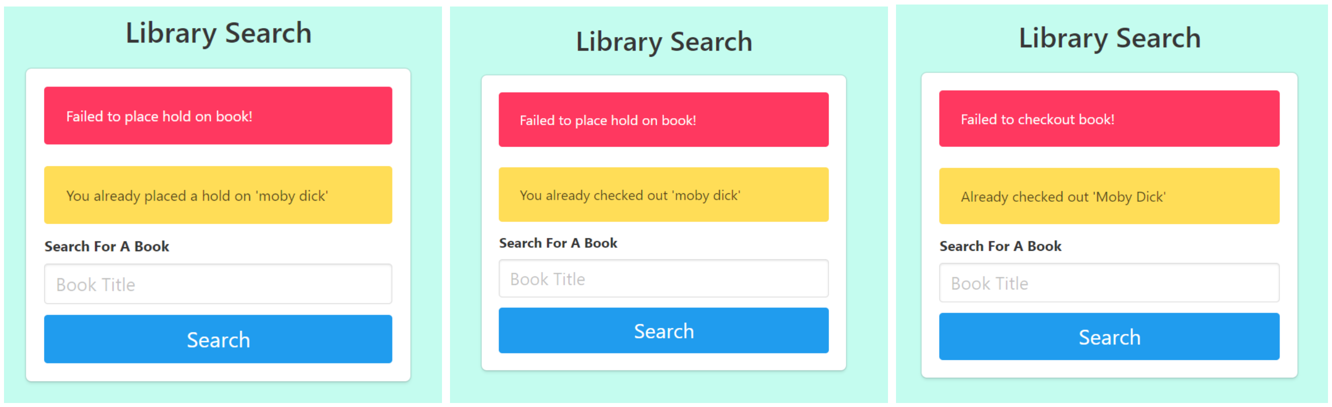
Due Date: 2021-12-22 21:54:25

Figure 9: Results of Checking Out Moby Dick From Charlestown Library as holdacct



Successfully placed hold on Moby Dick

Figure 10: Example Result of Placing Hold On Moby Dick from Central Library as holdacct



Failed to place hold on book!

You already placed a hold on 'moby dick'

Failed to place hold on book!

You already checked out 'moby dick'

Failed to checkout book!

Already checked out 'Moby Dick'

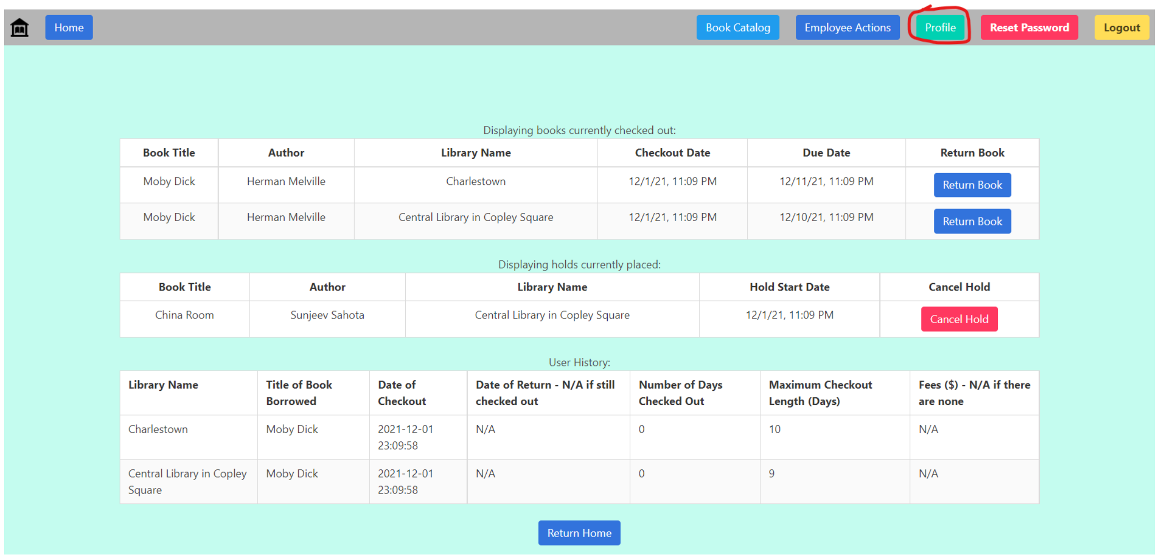
Figure 11: Example Failure for Attempting to Check Out / Place Hold on Multiple Copies of Moby Dick (with holdacct)

The holdacct user (see Appendix B for login information) is a great account to login to to play around with the checkout/hold functionality of the system. Try clicking the checkout/hold button on Moby Dick and it will be placed on hold.

Interacting with a Book Already Checked Out / On Hold

One particularly useful functionality of the web application is that it prevents a user from accidentally checking out a book twice or placing duplicate holds. If the user accidentally clicks the checkout/hold button or because they genuinely forgot they already have it, then the system will provide a prompt saying they already have it and not proceed. This interaction can also be seen via Figure 11 above.

View Profile



Displaying books currently checked out:

Book Title	Author	Library Name	Checkout Date	Due Date	Return Book
Moby Dick	Herman Melville	Charlestown	12/1/21, 11:09 PM	12/11/21, 11:09 PM	Return Book
Moby Dick	Herman Melville	Central Library in Copley Square	12/1/21, 11:09 PM	12/10/21, 11:09 PM	Return Book

Displaying holds currently placed:

Book Title	Author	Library Name	Hold Start Date	Cancel Hold
China Room	Sunjeev Sahota	Central Library in Copley Square	12/1/21, 11:09 PM	Cancel Hold

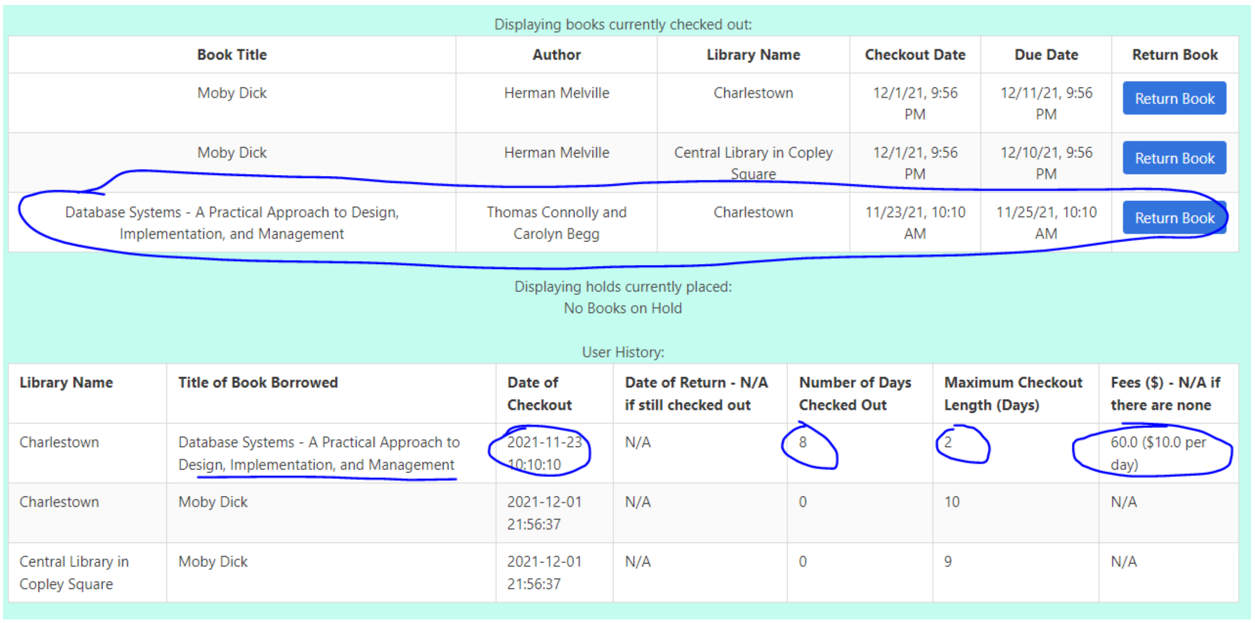
User History:

Library Name	Title of Book Borrowed	Date of Checkout	Date of Return - N/A if still checked out	Number of Days Checked Out	Maximum Checkout Length (Days)	Fees ($) - N/A if there are none
Charlestown	Moby Dick	2021-12-01 23:09:58	N/A	0	10	N/A
Central Library in Copley Square	Moby Dick	2021-12-01 23:09:58	N/A	0	9	N/A

Figure 12: User profile page (for account nickrizzo)

A user can also see which books they currently have checked out or on hold and their past checkout history by visiting their profile page (as seen in Figure 12). The profile page is navigated to by clicking on the "Profile" button on the top right or by typing in the url "http://localhost:8080/profile". This page is extremely useful (and the only way) for interacting with existing holds and check outs.

If the additional script src/database/show_overdue_functioanlity.sql was run during setup (described in Setup step 3), the nickrizzo user will have an additional book checked out and an additional row in User History that shows a late fee. This can be seen in Figure 13. Please note that at the time of generating this result, the day was 12/1/2021, which is 8 days after the checkout date of 11/23/2021. As the maximum checkout length is 2 days, the book has been overdue for 6 days. Hence, the late fee of $60 (with $10 per day) is accurate. Demonstrating the late fee calculation is correct. Readers of this report can feel free to validate this result personally.



Displaying books currently checked out:

Book Title	Author	Library Name	Checkout Date	Due Date	Return Book
Moby Dick	Herman Melville	Charlestown	12/1/21, 9:56 PM	12/11/21, 9:56 PM	Return Book
Moby Dick	Herman Melville	Central Library in Copley Square	12/1/21, 9:56 PM	12/10/21, 9:56 PM	Return Book
Database Systems - A Practical Approach to Design, Implementation, and Management	Thomas Connolly and Carolyn Begg	Charlestown	11/23/21, 10:10 AM	11/25/21, 10:10 AM	Return Book

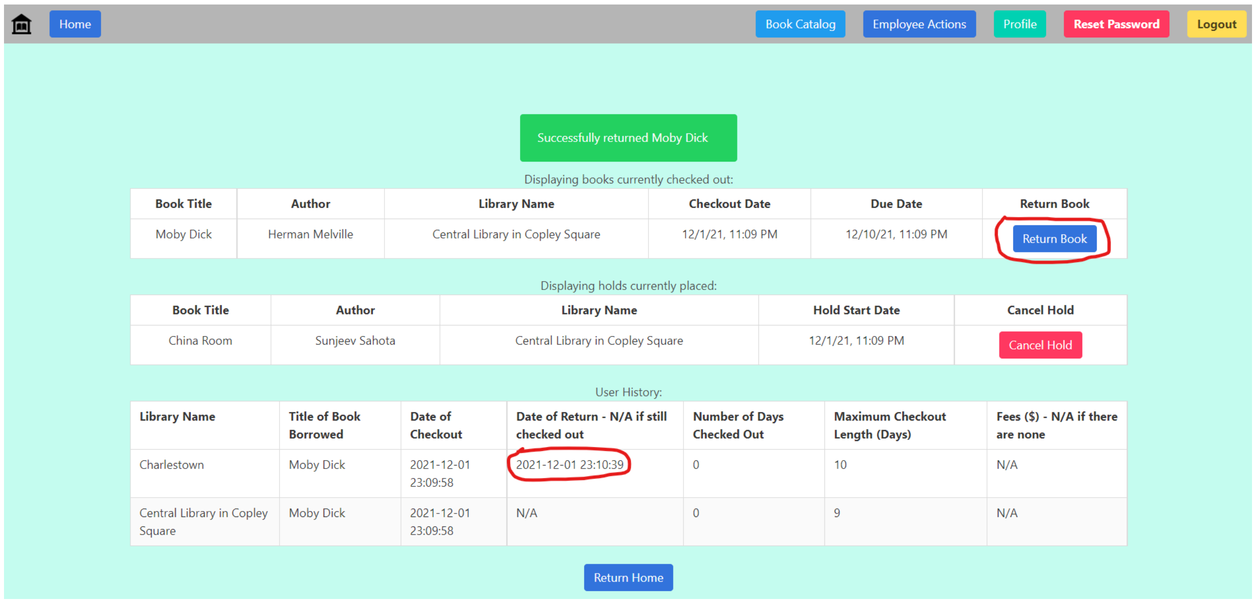
Displaying holds currently placed:
No Books on Hold

User History:

Library Name	Title of Book Borrowed	Date of Checkout	Date of Return - N/A if still checked out	Number of Days Checked Out	Maximum Checkout Length (Days)	Fees ($) - N/A if there are none
Charlestown	Database Systems - A Practical Approach to Design, Implementation, and Management	2021-11-23 10:10:10	N/A	8	2	60.0 ($10.0 per day)
Charlestown	Moby Dick	2021-12-01 21:56:37	N/A	0	10	N/A
Central Library in Copley Square	Moby Dick	2021-12-01 21:56:37	N/A	0	9	N/A

Figure 13: Demonstration of Late Fee Calculation with nickrizzo User

Returning a Book or Hold



Successfully returned Moby Dick

Displaying books currently checked out:

Book Title	Author	Library Name	Checkout Date	Due Date	Return Book
Moby Dick	Herman Melville	Central Library in Copley Square	12/1/21, 11:09 PM	12/10/21, 11:09 PM	Return Book

Displaying holds currently placed:

Book Title	Author	Library Name	Hold Start Date	Cancel Hold
China Room	Sunjeev Sahota	Central Library in Copley Square	12/1/21, 11:09 PM	Cancel Hold

User History:

Library Name	Title of Book Borrowed	Date of Checkout	Date of Return - N/A if still checked out	Number of Days Checked Out	Maximum Checkout Length (Days)	Fees ($) - N/A if there are none
Charlestown	Moby Dick	2021-12-01 23:09:58	2021-12-01 23:10:39	0	10	N/A
Central Library in Copley Square	Moby Dick	2021-12-01 23:09:58	N/A	0	9	N/A

Return Home

Figure 14: Returning a Book (compare against Figure 12)

If a user wanted to return a book, they would navigate to their profile page (as described in the profile section) and click the “Return Book” button next to the book they want to return as shown in Figure 14. The process is exactly the same for cancelling a hold. The system will provide an alert to the user if the return or cancellation is successful (like in Figure 14) or was not able to be completed. Interestingly, when a user returns a book, the “User History” table in the profile is immediately updated such that the “Date of Return” for the returned book contains the current datetime. This can be seen by comparing the “User History” table between Figure 12 and 14 by focusing on the row containing Charlestown’s copy of “Moby Dick” since it was returned in between Figure 12 & 14’s screenshots.

Potential Employee Actions/Flows

Please note that these are all in addition to the regular user actions an employee can perform. These employee specific actions are all on the /employee_actions page that is easily accessed via the Employee Actions button available to all *approved* logged in employees. Figure 15 shows the button that is only available to employees and takes them to Employee Specific options like Verifying new employees and Adding Books.

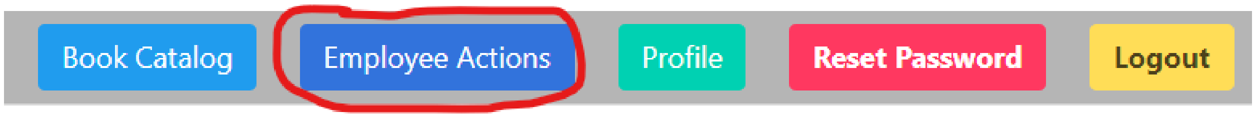
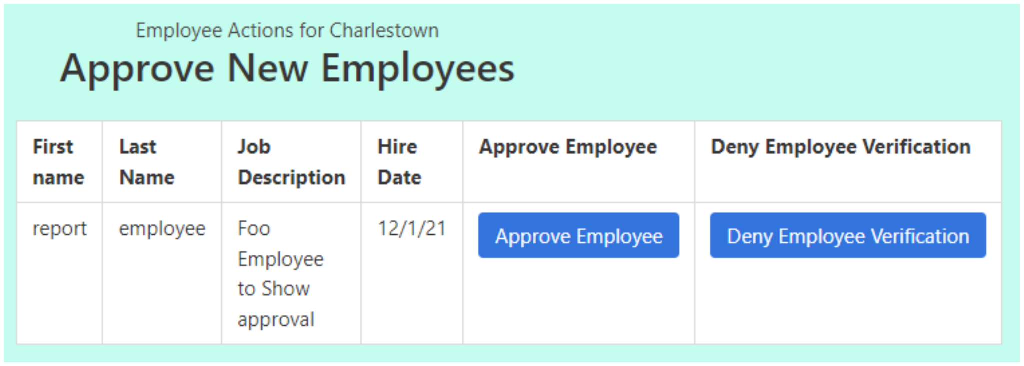


Figure 15: Show of Employee Actions Button

Approve / Deny New Employees

When a user registers themself as an employee, they must be approved by an employee at the same library who is already approved and has all privileges (see Figure 4 of Registering a User/Employee). For example, if user "foo" (with name "report employee") registers as an employee at Charlestown Library, the employee with username nickrizzo who is already approved at Charlestown Library can login and approve or deny foo (see Appendix A). This can be seen in Figure 16 where foo can either be verified/approved or denied as an employee.



Employee Actions for Charlestown

Approve New Employees

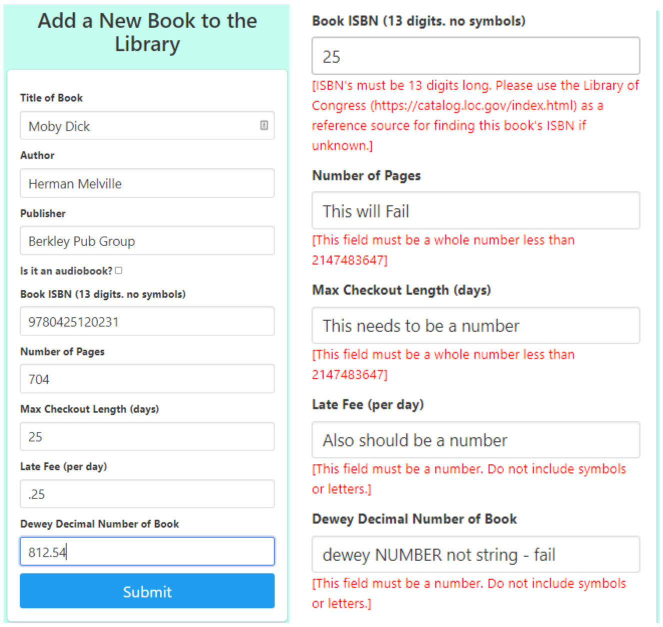
First name	Last Name	Job Description	Hire Date	Approve Employee	Deny Employee Verification
report	employee	Foo Employee to Show approval	12/1/21	Approve Employee	Deny Employee Verification

Figure 16: Approval View Within Employee Actions for User nickrizzo

If the Approve button is clicked by nickrizzo, the next time foo logs in, they will also have access to the Employee Actions button, Employee Actions page, and all existing Employee Privileges. If they are denied, they will keep their existing user privileges, but not be granted any Employee Privileges. When the user/employee nickrizzo clicks either option, they will be kept on the current page, but foo will be removed from the table. If there are multiple users awaiting approval, all other rows will remain in the table.

Add New Book to Library

Only employees can add new books to the library. From the Employee Actions page, the form can be filled out to add a new book. All the fields are required, and the user will be guided with each field's requirements if they are not met. Figure 17 (on the left) shows an example adding another copy of Moby Dick (wow that sure is a popular book) to the Charlestown Library via the nickrizzo account. When this form is submitted, a new row is inserted into the book_inventory table with all of this information. Based on the dewey number given and the dewey number range of the libraries bookshelves/bookcases, a correct bookshelf position is picked (see the add_new_book procedure). If this is the first time a book with this ISBN is added, it would also be inserted into the book table.


Add a New Book to the Library

Title of Book	Moby Dick
Author	Herman Melville
Publisher	Berkley Pub Group
Is it an audiobook?	☐
Book ISBN (13 digits. no symbols)	9780425120231
Number of Pages	704
Max Checkout Length (days)	25
Late Fee (per day)	.25
Dewey Decimal Number of Book	812.54

Book ISBN (13 digits. no symbols)	25
	[ISBN's must be 13 digits long. Please use the Library of Congress (https://catalog.loc.gov/index.html) as a reference source for finding this book's ISBN if unknown.]
Number of Pages	This will Fail
	[This field must be a whole number less than 2147483647]
Max Checkout Length (days)	This needs to be a number
	[This field must be a whole number less than 2147483647]
Late Fee (per day)	Also should be a number
	[This field must be a number. Do not include symbols or letters.]
Dewey Decimal Number of Book	dewey NUMBER not string - fail
	[This field must be a number. Do not include symbols or letters.]

Figure 17&18: Valid Form & Invalid to Add A New Book

Whereas, Figure 18 (on the right) shows the types of errors that will be received if the fields are incorrect. The ISBN must be 13 digits long. References are provided to determine the ISBN of a given book. Similarly, the Number of Pages, Max Checkout Mength, Late Fee Per Day, and Dewey Decimal Number should be integer or float values. Entering strings instead will cause the book form to be rejected. Late fee and dewey number can have fractional values as well.

6 - Lessons Learned

Technical expertise gained

For this project, github was used to allow multiple project members to collaboratively work on the program files. After a few technical difficulties during the setup and installation of git/GitHub, it provided a much more effective file sharing system than something such as sharing files in a Google Drive folder. A second technical skill developed during this project was the use of MySQL for a DBMS. Designing and creating the entire library database, including all tables, functions, and procedures needed to interface with the web application, increased the proficiency in SQL for all project members. Previous classwork assignments using MySQL involved interacting with an already created database, so the opportunity to create one from scratch was a great learning opportunity. Another technical expertise was the interfacing of MySQL with frontend and backend languages during the creation of the web application for this project.

Some additional concepts that group members learned was how to use python's Flask module and some of its add-ons (Flask-Forms/Flask-WTF, Flask-Tables, and Flask-Login) to create a full functional user web application. While some group members had previous experiences with simplistic Flask Applications, this project was definitely more involved as it had more moving parts. Specifically, Flask-Login added middleware (i.e. cookies and secure logins) to facilitate the ability for users to create accounts and login with help of the MySQL Database. On top of learning all these Flask concepts the group also had to learn how to interface between the database's stored procedures and the web applications backend & frontend.

Insights, time management insights, data domain insights etc.

This project was very informative in that group members have experience working on web application server frameworks (in C, C++, javascript, & python), but have never integrated it with an SQL based database. In turn, this project was very educational as members had to learn how to create HTTP endpoints capable of getting, updating, and deleting data from the database. Moreover, the complicated nature in which SQL procedures usually had to reference a specific user meant many joins needed to be done to ensure that the available books being shown were books that were actually accessible for that user. Additionally, the concepts of registering, logging in, and remembering users are fairly new concepts for the group so there was a fairly steep initial learning curve. However, now that the group has had exposure to how to handle these middleware functionalities (at least with Flask and Flask-Login) any future projects will be much easier to understand and write.

One issue that was encountered by the group were unexpected procedures and functions that were needed by the python backend to provide functionality to users that were not originally expected. The group tried to rigorously plan out what procedures and functions were needed with certain inputs and outputs in the early planning stage of the project. Despite this well made plan, it very readily became apparent that additional procedures were needed simply to display the available & checked out books as well as the currently placed holds for any given user on their profile page. At the onset, the group forgot that users would need to see their current account status in order to interact with it to perform additional checkouts, holds, cancellations, etc. This unexpected roadblock hampered the group's planning as work was divided evenly prior to realizing more work was needed.

Overall, the group was able to produce a very impressive product that everyone is proud of. Unfortunately, it was not a trivial matter to produce this web application and it required a significant amount of time. Luckily the group started working on the project early and lived by the concept of "fail hard, fail early." Getting the registration and login pages to work correctly with the database in the very beginning was a very challenging task that was hard to perfect. However, once this task was completed and the group learned how Flask-Forms, Flask-WTF (an abbreviation for Flask-Tables), and Flask-Login worked and were interfaced with the overall application, future functionality additions were much simpler to add as they required tools the group was already familiar with.

Another problem the group encountered was that many procedures and functions depended on the results of past procedures and inserted data. For example, to properly test the search_for_book procedure, the database has the precondition of there being books in a library. This also requires there being bookcases and bookshelves, which in turn requires there being libraries and library systems. In order to do that, the procedure to add/insert that piece of data into the database had to be written and tested (a simple INSERT INTO would not work because positioning based on dewey numbers had to be achieved for bookcases and shelves). Hence, before writing and testing the search_for_book procedure, at least 4 other procedures had to be written, and then data had to be found to insert into the database (books with all their fields, real libraries with information, etc... This was not part of the original timing estimate for the task.

Moreover, if one procedure stopped working because of various changes to the SQL procedures/schema and python code, then many other procedures would stop working as well. To handle these hiccups, the group ended up adding extensive error checking - both in python and in the SQL procedures - such that the program would alert users to failures and move on instead of simply silently crashing. Once the group finished writing the program, extensive tests were done to further prove all the functionality works as expected. Despite the achievement of getting this web app fully functioning, the group probably took on too much work since the time commitments were very large for this project.

Realized or contemplated alternative design / approaches to the project

There are a lot of joins in the project. For example, navigating from a tuple in book_inventory to its given library. It requires joins to go from tables book_inventory->bookshelf->bookcase->library using joins on the Foreign Keys and unique keys of each respective table. This works, but is most likely not scalable if the data becomes sufficiently large. Hence, the schema is a good candidate for switching to a NoSQL database such as MongoDB. The chaining of joins is necessitated by 3rd Normal Form tables and the proper conceptual design of the Entities and Tables, but it leads to unnecessary overhead in the deployed system.

Additionally, the backend language (currently Python) can be replaced with a more natively optimized language such as C++. Though the overhaul might be time consuming, C++ web frameworks, like pistache, offer much better performance for HTTP REST oriented Web Applications (see its open-source code at https://github.com/pistacheio/pistache).

Document any code not working in this section

There is no code in this project that is not working. All code works as expected and all testing confirmed this. All potential sql errors in python are caught with try/except's. There might be edge cases that fail, but no attempts to find them have been successful. If other errors are found, please feel free to reach out to the students who worked on the project at their respective emails (see below). Thank you.
Matthew Rizzo - rizzo.ma@northeastern.edu
Nicholas Rizzo - rizzo.n@northeastern.edu
Domenic Privitera - privitera.d@northeastern.edu

7 - Future Work

Planned uses of the database

This database system designed by PriviteraRizzoRizzo will serve as a book management system for libraries in a library network, i.e. all public libraries in the City of Boston. This system will allow a library to locate where books are located amongst its branches, and to keep track of what users have checked out what books.

The data being tracked includes each library branch in a network, users who have library cards in the network, the history of books checked out by users in the system, employees of each library, books owned by each library, and the location of each book in a particular library.

Members of libraries within a system that uses the application will be able to search for books available at the library, request a book for any library in the system, place a hold for any book in the system, track the due date of a given loaned book, track the availability of a book, and other functionalities of that nature.

The project will allow library users the ability to perform these commands completely via a web page which will display information to them, and allow for inputs as needed.

Potential areas for added functionality

Currently, the title of a book must be exactly what the user types in the search bar. One potential added functionality would be to display results with a similar book title to what the user had entered. A second improvement would be to add a new class of employees, managers, and have them be the only employees allowed to approve a new employee account. Providing tighter control on who is allowed to promote a user account to an employee account would reduce the likelihood of unauthorized access to the management system.

Appendices

Appendix A - Employee Logins

Please note these are all properly approved employees. Hence, they can be used to approve future employees at their respective libraries. For ease, they are included in order of their library ID as well

Library Name	username	password
Charlestown	nickrizzo	pwd
Central Library in Copley Square	central_employee	central_pwd
Jamaica Plain	jamaica_employee	jamaica_pwd
Plymouth Public Library	mattrizzo	pwd
Kingston Public Library	kingston_employee	kingston_pwd
Cambridge Public Library	dompriv	pwd

Appendix B - Other Preloaded User Logins

Library Name	username	password
Charlestown	holdacct	pwd