Library Management Software

Acknowledgement

First and foremost I thank god almighty for showering his blessings during the course of this project.

I thank Fr. Salvin Augustine SJ Director of Loyola School and Fr Roy Alex SJ Principal of Loyola School for providing us with computer labs and other facilities that aided in our project.

I express my sincere gratitude to our esteemed computer teacher Ms Roshy under whose guidance this paper was written. I thank her for her enthusiasm for the subject and for her guidance and support throughout every step both in this project and throughout the curriculum. I also thank her for her advice and constant encouragement.

Gratitude is extended to the open-source communities of Python and SQL, whose wealth of resources, documentation, and forums empowered us to overcome technical hurdles and implement robust features. CustomTkinter and Tkinter as GUI toolkits, deserves special acknowledgment for simplifying the creation of an intuitive user interface. I also thank all the other module creators for enriching python with awesome functionality

I further thank all of the staff members and our fellow classmates for their support and cooperation extended to us during this work

I owe our sincere gratitude towards Loyola School for their excellent academic undertaking and towards CBSE for making this happen. I also express our deepest gratitude to our parents.

Index

Sl. No.	Description	Page No.
1	Introduction to Python	1
2	Project Synopsis	2
3	Source Images	4
4	Source Code (SQL)	5
5	Source Code (Text)	9
6	Source Code (Python)	10
7	Output Screens	51
8	Bibliography	69

Introduction to Python

Python is an interpreted high-level general purpose programming language. Its design philosophy emphasizes code readability with its use of significant indentation. Its language constructs as well as its object- oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s, as a successor to the ABC programming and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features. Such as list comprehensions and a cycle-detecting garbage collection system (in addition to reference counting). Python 3.0 was released in 2008 and was a major revision of the language that is not backward- compatible. Python 2 was discontinued with the version 2.7.18 in 2020.

Python consistently ranks as one of the most popular programming language due to its simplicity and versatality. Python was designed to be easy to understand and intuitive. It was created with the intention of people reading code as they do plain English, so it is excellent for beginners and there is only a small learning curve to it.

Synopsis

Managing a library requires keeping a data log book and the librarian must maintain and update this book throughout the operation of the library. Every time a book is taken or a new member is added the librarian has to perform a continuous stream of repetitive tasks to perform the action. This monotonous process may become difficult and time consuming for the librarian.

This project aims to simplify and make easy the task of maintaining a library with the help of a computer database and an easy to use graphical user interface. Strenuous effort had been put to ensure the correctness and security of the data being entered. This software is intended for use only to the librarian or the administrator of the library and has built in functions to easily perform administrative tasks and search up data.

The main features of this software is as follows:

- Issue books to members
- Return books to members
- Calculate fine when returning
- Reserve books on demand
- Compute availability of book
- Compute eligibility of member to lent a book
- Add / Remove members
- Manage catalogs of books, authors and publishers
- Other utilities

The code for this project is split into two – **'database_functions.py'** and **'main.py'.** The former contains the code for the fronted *i.e.,* customtkinter and only focuses on the gui and functions for entering data and checking the partial integrity of entered data. The latter contains the backend code *i.e.,* the SQL queries and functions to calculate and encode data. The functions in the latter are imported in the former to complete the software.

A simple yet effective method of error handling has also been incorporated in the project. In the backend, errors are finely categorised into groups and each error group has a unique number as given in the source. For example, operation success is denoted by 99. The backend functions forwards the error – if any - to the fronted where it gets displayed appropriately.

Softwares used for project: Python 3.12.0, MySQL community edition 8.0.35
Python modules – ctktable, pil, ctkxyframe, datetime, customtkinter, tkinter

This project's **source code** can be found on:
https://drive.proton.me/urls/TJ7K52EN2W#ZygDArSPGYLA

The database '**Library**' used for this project contains the following tables:

members	
PK	**mid INTEGER AUTO_INCREMENT**
	name VARCHAR(30) NOT NULL
	address VARCHAR(150)
	phone BIGINT
	gender VARCHAR(15)
	class INTEGER(1)
	no_of_books_rented INTEGER DEFAULT 0
	CHECK (class IN (1, 2, 3))

books	
PK	**bid INTEGER AUTO_INCREMENT**
FK	**aid INTEGER**
FK	**pid INTEGER**
	title VARCHAR(50) NOT NULL
	genre VARCHAR(30)
	type VARCHAR(30)
	isbn BIGINT(13)
	availability BOOL
	edition INTEGER
	no_of_copies INTEGER
	description VARCHAR(255)
	location VARCHAR(100)
	FOREIGN KEY(aid) REFERENCES authors(aid)
	FOREIGN KEY(pid) REFERENCES publishers(pid));

authors	
PK	**aid INTEGER AUTO_INCREMENT**
	name VARCHAR(30) NOT NULL
	gender VARCHAR(15)
	dob DATE
	country VARCHAR(15)
	info VARCHAR(255)
	phone BIGINT
	contact VARCHAR(255)

reservation	
FK	**bid INTEGER NOT NULL**
FK	**mid INTEGER NOT NULL**
	date_reserved DATE NOT NULL
	reservation_end_date DATE NOT NULL
	FOREIGN KEY(bid) REFERENCES books(bid)
	FOREIGN KEY(mid) REFERENCES members(mid));

publishers	
PK	**pid INTEGER AUTO_INCREMENT**
	name VARCHAR(50)
	contact VARCHAR(255)
	details VARCHAR(255)

loan	
FK	**bid INTEGER NOT NULL**
FK	**mid INTEGER NOT NULL**
	date_taken DATE NOT NULL
	due_date DATE NOT NULL
	FOREIGN KEY(bid) REFERENCES books(bid)
	FOREIGN KEY(mid) REFERENCES members(mid));

statistics	
PK	**bid INTEGER NOT NULL**
	no_of_loans INTEGER NOT NULL DEFAULT 0
	month INTEGER NOT NULL
	FOREIGN KEY(bid) REFERENCES books(bid));

credentials	
PK	**username VARCHAR(20)**
	passwd VARCHAR(30) NOT NULL

Source Images

(1) – ‘icons/ icon.png’

(2) – ‘icons/image.png’

Source Code (SQL)

(1) 'tables.sql' – SQL file containing instructions to create tables in the database.

```
 1  CREATE TABLE members (
 2   mid INTEGER AUTO_INCREMENT,
 3   name VARCHAR(30) NOT NULL,
 4   address VARCHAR(150),
 5   phone BIGINT,
 6   gender VARCHAR(15),
 7   class INTEGER(1),
 8   no_of_books_rented INTEGER DEFAULT 0,
 9   CHECK (class IN (1, 2, 3)),
10   PRIMARY KEY(mid));
11
12  CREATE TABLE authors (
13   aid INTEGER AUTO_INCREMENT,
14   name VARCHAR(30) NOT NULL,
15   gender VARCHAR(15),
16   dob DATE,
17   country VARCHAR(15),
18   info VARCHAR(255),
19   phone BIGINT,
20   contact VARCHAR(255),
21   PRIMARY KEY(aid));
22
23  CREATE TABLE publishers (
24   pid INTEGER AUTO_INCREMENT,
25   name VARCHAR(50),
26   contact VARCHAR(255),
27   details VARCHAR(255),
28   PRIMARY KEY(pid));
29
30  CREATE TABLE books (
31   bid INTEGER AUTO_INCREMENT,
32   aid INTEGER,
33   pid INTEGER,
34   title VARCHAR(50) NOT NULL,
35   genre VARCHAR(30),
36   type VARCHAR(30),
37   isbn BIGINT(13),
38   availability BOOL,
39   edition INTEGER,
```

```
 no_of_copies INTEGER,
 description VARCHAR(255),
 location VARCHAR(100),
 PRIMARY KEY(bid),
 FOREIGN KEY(aid) REFERENCES authors(aid),
 FOREIGN KEY(pid) REFERENCES publishers(pid));

CREATE TABLE loan (
 bid INTEGER NOT NULL,
 mid INTEGER NOT NULL,
 date_taken DATE NOT NULL,
 due_date DATE NOT NULL,
 FOREIGN KEY(bid) REFERENCES books(bid),
 FOREIGN KEY(mid) REFERENCES members(mid));

CREATE TABLE reservation (
 bid INTEGER NOT NULL,
 mid INTEGER NOT NULL,
 date_reserved DATE NOT NULL,
 reservation_end_date DATE NOT NULL,
 FOREIGN KEY(bid) REFERENCES books(bid),
 FOREIGN KEY(mid) REFERENCES members(mid));

CREATE TABLE statistics (
 bid INTEGER NOT NULL,
 no_of_loans INTEGER NOT NULL Dbooks(bid));

CREATE TABLE credentials (
 username VARCHAR(20),
 passwd VARCHAR(30) NOT NULL,
 PRIMARY KEY(username));EFAULT 0,
 month INTEGER NOT NULL,
 FOREIGN KEY(bid) REFERENCES books(bid));

CREATE TABLE credentials (
 username VARCHAR(20),
 passwd VARCHAR(30) NOT NULL,
 PRIMARY KEY(username));
```

(2) 'values.sql' – SQL file containing sample test values to perform and test the program on.

```
 1  INSERT INTO credentials (username, passwd) VALUES
 2  ('devanandan', 'kakakaka'),
 3  ('anantha', 'kookookookoo');
 4
 5  INSERT INTO publishers (name, contact, details) VALUES
 6  ('Scholastic Press', 1234567890, 'Leading publisher of
children''s books'),
 7  ('Penguin Random House', 2345678901, 'Major publisher of adult
and children''s books'),
 8  ('Hachette Book Group', 3456789012, 'Major publisher of
institutional knowledge books');
 9
10  INSERT INTO members (name, address, phone, gender, class) VALUES
11  ('John Doe', '123 Main Street', 4598465494, 'Male', 1),
12  ('Jane Doe', '456 Elm Street', 4842168451, 'Female', 2),
13  ('Peter Smith', '789 Oak Street', 489752542, 'Male', 3),
14  ('Susan Jones', '101 Maple Street', 8952152156, 'Female', 1),
15  ('David Brown', '202 Pine Street', 87841584544, 'Male', 2),
16  ('Elizabeth Green', '303 Elm Street', 487487875, 'Female', 3),
17  ('Michael Williams', '404 Oak Street', 545488884, 'Male', 1),
18  ('Sarah Johnson', '505 Maple Street', 1489744946, 'Female', 3),
19  ('William Thomas', '606 Pine Street', 994719894, 'Male', 3),
20  ('Catherine Anderson', '707 Elm Street', 8998465165, 'Female',
1);
21
22  INSERT INTO authors (name, gender, dob, country, info, phone,
contact) VALUES
23  (
24      'J.K. Rowling', 'Female', '1965-07-31',
25      'United Kingdom', 'Best-selling author known for Harry
Potter series',
26  4567890, 'jkrowling@example.com'
27    ),
28    (
29      'George R.R. Martin', 'Male', '1948-09-20',
30      'United States', 'Author of A Song of Ice and Fire series',
31  6543210, 'grrmartin@example.com'
32    ),
33    (
34      'Harper Lee', 'Female', '1926-04-28',
35      'United States', 'Author of To Kill a Mockingbird',
36  8901234, 'harperlee@example.com'
37    );
38
39  INSERT INTO books (aid, pid, title, genre, type, isbn,
```

```
availability,
    edition, no_of_copies, description, location) VALUES
    (
  1, 'Harry Potter and the Chamber of Secrets',
      'Fantasy', 'Fiction', 9780747538493,
      true, 1, 5, 'Second book in the Harry Potter series',
      'Library A Shelf 15 Row 2'
    ),
    (
  1, 'Harry Potter and the Prisoner of Azkaban',
      'Fantasy', 'Fiction', 9780439136365,
      true, 1, 3, 'Third book in the Harry Potter series',
      'Library A'
    ),
    (
  2, 'A Game of Thrones', 'Fantasy',
      'Fiction', 9780553381689, true, 1,
  'First book in A Song of ice and Fire series',
      'Library B'
    ),
    (
  2, 'A Clash of Kings', 'Fantasy',
      'Fiction', 9780553381696, true, 2,
  'Second book in A Song of Ice and Fire series',
      'Library B'
    ),
    (
  3, 'To Kill a Mockingbird', 'Fiction',
      'Novel', 9780446310789, true, 1, 4,
      'Classic novel set during the Great Depression',
      'Library C'
    );
```

Source Code (Text)

(1) – 'errors.txt' – a text file used to map arbitrary error numbers to their corresponding errors. These errors numbers are used in the program to manage and forward errors within functions.

```
0 - no match found i.e. no similar data/entry in table
1 - no entries/data in table i.e. table is empty
2 - username not found
3 - password is incorrect
4 - books limit reached
5 - book not available
7 - no loan exist
8 - member cannot hold book
9 - maximum input length exceeded
10 - member has unreturned book
11 - only one book left
99 - success
```

Source Code (Python)

(1) ‘create_database.py’ – python script used to create the database and inital tables using ‘tables.sql’ in mysql.

```
 1  from mysql.connector import connect
 2
 3  connection = connect(user="root", password="8235",
host="127.0.0.1", database="library")
 4  cursor = connection.cursor()
 5
 6  with open('tables.sql', 'r') as tables:
 7      sql = tables.read()
 8      commands = sql.split(';')
 9      for command in commands:
10          cursor.execute(command)
11          connection.commit()
12
13  with open('values.sql', 'r') as values:
14      sql = values.read()
15      commands = sql.split(';')
16      for command in commands:
17          cursor.execute(command)
18          connection.commit()
```

(2) ‘database_functions.py’ – pyton script containing all the backend functions used for manipulating the database.

```
from rapidfuzz.fuzz import (
    partial_ratio,
    partial_token_set_ratio,
    WRatio,
    token_set_ratio,
)
from mysql.connector import connect
from datetime import date, timedelta

connection = connect(user="root", password="8235",
host="127.0.0.1", database="library")
cursor = connection.cursor()

###############
# credentials #
###############
def check_credentials(username, password):
    username_exists = False
    password_correct = False
    cursor.execute("SELECT * FROM credentials;")
    for row in cursor:
        if username == row[0]:
            username_exists = True
            if password == row[1]:
                password_correct = True

    if not username_exists:
        out = 2
    elif not password_correct:
        out = 1
    else:
        out = 99

    return out


# print(check_credentials('devanandan','kakakaka'))
# print(check_credentials('sd','65'))

####################
# common functions #
####################


def similarity(str1, str2):
```

```
45      lstr1 = "".join([*filter(str.isalnum, str1.lower())])
46      lstr2 = "".join([*filter(str.isalnum, str2.lower())])
47      r1 = partial_ratio(lstr1, lstr2)
48      r2 = partial_token_set_ratio(lstr1, lstr2)
49      r3 = WRatio(lstr1, lstr2)
50      r4 = token_set_ratio(lstr1, lstr2)
51      return max([r1, r2, r3, r4])
52
53
54  def out(array, error):
55      if len(array) != 0:
56          out = array
57      else:
58          out = error
59
60      return out
61
62  def search_book_by_title(title):
63      BIDs = []
64      cursor.execute("SELECT title, bid FROM books;")
65      for row in cursor:
66          if similarity(row[0], title) >= 71:
67              BIDs.append(row[1])
68      #    print(BIDs)
69      return out(BIDs, 0)
70
71
72  # title = input("title: ")
73  # print(search_book_by_title(title))
74
75  def search_book_by_author(author):
76      BIDs = []
77      AIDs = []
78      cursor.execute("SELECT name, aid FROM authors;")
79      for row in cursor:
80          if similarity(row[0], author) >= 71:
81              AIDs.append(row[1])
82      for AID in AIDs:
83          cursor.execute(f"SELECT bid FROM books WHERE aid =
{AID};")
84          for row in cursor:
85              BIDs.append(row[0])
86
87      return out(BIDs, 0)
88
89
90  # author = input("author: ")
91  # print(search_book_by_author(author))
92
```

```
 93  def search_book_by_description(description):
 94      BIDs = []
 95      cursor.execute("SELECT description, bid FROM books;")
 96      for row in cursor:
 97          if similarity(row[0], description) >= 71:
 98              BIDs.append(row[1])
 99
100      return out(BIDs, 0)
101
102
103  # description = input("description: ")
104  # print(search_book_by_description(description))
105
106  def search_author_by_name(name):
107      AIDs = []
108      cursor.execute("SELECT name, aid FROM authors;")
109      for row in cursor:
110          if similarity(row[0], name) >= 70:
111              AIDs.append(row[1])
112
113      return out(AIDs, 0)
114
115
116  # name = input("name: ")
117  # print(search_author_by_name(name))
118  Default Paragraph Style
119  def search_publisher_by_name(name):
120      PIDs = []
121      cursor.execute("SELECT name, pid FROM publishers;")
122      for row in cursor:
123          if similarity(row[0], name) >= 70:
124              PIDs.append(row[1])
125
126      return out(PIDs, 0)
127
128
129  # name = input("name: ")
130  # print(search_publisher_by_name(name))
131
132  def get_book_info(BID):
133      details = {}
134      cursor.execute(f"SELECT * FROM books WHERE bid = {BID};")
135      for row in cursor:
136          details["aid"] = row[1]
137          details["pid"] = row[2]
138          details["title"] = row[3]
139          details["genre"] = row[4]
140          details["type"] = row[5]
141          details["isbn"] = row[6]
```

```
142             details["availability"] = row[7]
143             details["edition"] = row[8]
144             details["no_of_copies"] = row[9]
145             details["description"] = row[10]
146             details["location"] = row[11]
147         return out(details, 0)
148 
149 
150     # print(get_book_info(1))
151     # print(get_book_info(6))
152 
153     def get_member_info(MID):
154         details = {}
155         cursor.execute(f"SELECT * FROM members WHERE mid = {MID};")
156         for row in cursor:
157             details["name"] = row[1]
158             details["address"] = row[2]
159             details["phone"] = row[3]
160             details["gender"] = row[4]
161             details["class"] = row[5]
162             details["no_of_books_rented"] = row[6]
163         return out(details, 0)
164 
165     # print(get_member_info(1))
166     # print(get_member_info(11))
167 
168     def get_author_info(AID):
169         details = {}
170         cursor.execute(f"SELECT * FROM authors WHERE aid = {AID};")
171         for row in cursor:
172             details["name"] = row[1]
173             details["gender"] = row[2]
174             details["dob"] = str(row[3])
175             details["country"] = row[4]
176             details["info"] = row[5]
177             details["phone"] = str(row[6])
178             details["contact"] = row[7]
179         return out(details, 0)
180 
181     def get_publisher_info(PID):
182         details = {}
183         cursor.execute(f"SELECT * FROM publishers WHERE pid =
184 {PID};")
185         for row in cursor:
186             details["name"] = row[1]
187             details["contact"] = row[2]
188             details["details"] = row[3]
189         return out(details, 0)
```

```
190
191  def display_books(limit, offset, aid="'%'", pid="'%'"):
192      books = []
193
194      if "'%'" not in [aid, pid]:
195          error = 0
196      else:
197          error = 1
198
199      cursor.execute(f"SELECT bid, title, genre, type, isbn,
availability, edition, no_of_copies, location, description FROM
books WHERE aid LIKE {aid} AND pid LIKE {pid} LIMIT {limit} OFFSET
{offset};")
200      for row in cursor:
201          books.append(row)
202
203      return out(books, error)
204
205
206  #print(display_books(2,2))
207  # print(display_books(0,0,2))
208  # print(display_books(0,0,5))
209  #print(display_books(10, 5))
210
211  def display_members(limit, offset):
212      members = []
213      cursor.execute(f"SELECT * FROM members LIMIT {limit} OFFSET
{offset};")
214      for row in cursor:
215          members.append(row)
216
217      return out(members, 1)
218
219
220  # print(display_members(2,0))
221
222
223  def display_authors(limit, offset):
224      authors = []
225      cursor.execute(f"SELECT * FROM authors LIMIT {limit} OFFSET
{offset};")
226      for row in cursor:
227          row = list(row)
228          row[3] = str(row[3])
229          authors.append(row)
230
231      return out(authors, 1)
232
233
```

```
234  # print(display_authors(2,0))
235
236  def display_publishers(limit, offset):
237      publishers = []
238      cursor.execute(f"SELECT * FROM publishers LIMIT {limit}
OFFSET {offset};")
239      for row in cursor:
240          publishers.append(row)
241
242      return out(publishers, 1)
243
244
245  # print(display_publishers(2,1))
246
247  def lend(bid, mid, timespan, date_obj=date.today()):
248      book_info = get_book_info(bid)
249      member_info = get_member_info(mid)
250      date = str(date_obj)
251      date = "'" + date + "'"
252      return_date = str(date_obj + timedelta(days=timespan))
253      return_date = "'" + return_date + "'"
254      if book_info != 0 and member_info != 0:
255          if book_info["availability"] > 0:
256              if member_info["no_of_books_rented"] <
member_info["class"]:
257                  cursor.execute(
258                      f"INSERT INTO loan VALUES({bid}, {mid},
{date}, {return_date});"
259                  )
260                  cursor.execute(
261                      f"UPDATE books SET availability =
availability - 1 WHERE bid = {bid};"
262                  )
263                  cursor.execute(
264                      f"UPDATE members SET no_of_books_rented =
no_of_books_rented + 1 WHERE mid = {mid};"
265                  )
266                  cursor.execute(
267                      f"UPDATE statistics SET no_of_loans =
no_of_loans + 1 WHERE bid = {bid};"
268                  )
269                  cursor.execute(
270                      f"UPDATE statistics SET month = {date}
WHERE bid = {bid};"
271                  )
272                  connection.commit()
273                  out = 99
274              else:
275                  out = 4
```

```
276             else:
277                 out = 5
278         else:
279             out = 0
280 
281         return out
282 
283 
284     # print(lend(54,54,54))
285     # print(lend(2,2,30))
286 
287     def calculate_fine(bid, mid, fee_per_day):
288         book_info = get_book_info(bid)
289         member_info = get_member_info(mid)
290         if book_info != 0 and member_info != 0:
291             loans = []
292             cursor.execute(f"SELECT due_date FROM loan WHERE bid =
{bid} AND mid = {mid};")
293             out = None
294             for row in cursor:
295                 loans.append(row)
296             if len(loans) == 0:
297                 out = 7
298             else:
299                 for loan in loans:
300                     today = date.today()
301                     due_date = str(loan[0]).split("-")
302                     due_date = date(int(due_date[0]),
int(due_date[1]), int(due_date[2]))
303                     delta_date = today - due_date
304                     try:
305                         delta_date = int(str(delta_date)[0:2])
306                     except:
307                         delta_date = int(str(delta_date)[0:3])
308                     if delta_date > 0:
309                         out = str(delta_date * fee_per_day)
310                     elif type(out) != str:
311                         out = 99
312                     else:
313                         out = int(out)
314                         if delta_date * fee_per_day > out:
315                             out = str(delta_date * fee_per_day)
316         else:
317             out = 0
318 
319         return out
320 
321 
322     # print(calculate_fine(4,8,10))
```

```
323
324  def return_book(bid, mid, fee_per_day):
325      fine = calculate_fine(bid, mid, fee_per_day)
326      if fine != 0:
327          cursor.execute(f"DELETE FROM loan WHERE bid={bid} AND
mid = {mid};")
328          cursor.execute(
329              f"UPDATE books SET availability = availability + 1
WHERE bid = {bid};"
330          )
331          cursor.execute(
332              f"UPDATE members SET no_of_books_rented =
no_of_books_rented - 1 WHERE mid = {mid};"
333          )
334          connection.commit()
335          out = fine
336      else:
337          out = fine
338
339      return out
340
341
342  # print(return_book(4,8,10))
343
344  def reserve_book(bid, mid, timespan, date_obj=date.today()):
345      member_info = get_member_info(mid)
346      book_info = get_book_info(bid)
347      date = str(date_obj)
348      date = "'" + date + "'"
349      rs_date = str(date_obj + timedelta(days=timespan))
350      rs_date = "'" + rs_date + "'"
351      if member_info != 0 and book_info != 0:
352          if book_info["availability"] > 0:
353              if member_info["class"] > 1:
354                  if member_info["no_of_books_rented"] <
member_info["class"]:
355                      cursor.execute(
356                          f"INSERT INTO reservation VALUES({bid},
{mid}, {date}, {rs_date});"
357                      )
358                      cursor.execute(
359                          f"UPDATE books SET availability =
availability - 1 WHERE bid = {bid};"
360                      )
361                      cursor.execute(
362                          f"UPDATE members SET no_of_books_rented
= no_of_books_rented + 1 WHERE mid = {mid};"
363                      )
364                      cursor.execute(
```

```
365                             f"UPDATE statistics SET no_of_loans =
no_of_loans + 1 WHERE bid = {bid};"
366                         )
367                         cursor.execute(
368                             f"UPDATE statistics SET month = {date}
WHERE bid = {bid};"
369                         )
370                         out = 99
371                     else:
372                         out = 4
373                 else:
374                     out = 8
375             else:
376                 out = 5
377         else:
378             out = 0
379         return out
380
381
382     # print(reserve_book(9, 5, 10))
383
384     def add_member(details):
385         try:
386             data = (
387                 details["name"],
388                 details["address"],
389                 details["phone"],
390                 details["gender"],
391                 details["class"],
392             )
393
394             query = (
395                 "INSERT INTO members (name, address, phone, gender,
class) VALUES"
396                 + "('"
397                 + data[0]
398                 + "','"
399                 + data[1]
400                 + "',"
401                 + str(data[2])
402                 + ",'"
403                 + data[3]
404                 + "',"
405                 + str(data[4])
406                 + ");"
407             )
408             cursor.execute(query)
409             connection.commit()
410             out = 99
```

```
411       except:
412           out = 9
413       return out
414
415
416   # print(add_member({'name':'devanandan', 'address':'kavadithala
manikandeswaram po', 'phone':94001063800, 'gender':'Male',
'class':1}))
417   # print(add_member({'name':'yadunandan', 'address':'peroorkada
manikandeswaram po', 'phone':9400106380, 'gender':'Male',
'class':1}))
418   # print(add_member({'name': 'Devanandan N. Byju', 'address':
'Vazhayila, Manikandeswaram P. O. TVPM', 'phone': 9400106380,
'gender': 'Male', 'class': 3}))
419
420   def del_member(mid):
421       names = []
422       cursor.execute(f"SELECT name FROM members WHERE
mid={mid};")
423       for name in cursor:
424           names.append(name)
425       if len(names) != 0:
426           details = get_member_info(mid)
427           if details["no_of_books_rented"] > 0:
428               out = 10
429           else:
430               cursor.execute(f"DELETE FROM members WHERE mid =
{mid};")
431               out = 99
432       else:
433           out = 0
434       connection.commit()
435       return out
436
437
438   # print(del_member(29))
439   # print(del_member(11))
440
441   def get_book_types():
442       types = []
443       cursor.execute("SELECT DISTINCT type FROM books;")
444       for book_type in cursor:
445           types.append(book_type[0])
446
447       return out(types, [])
448
449   def get_author_names():
450       names = []
451       cursor.execute("SELECT DISTINCT name FROM authors;")
```

```
452         for name in cursor:
453             names.append(name[0])
454 
455         return out(names, [])
456 
457     def get_publishers():
458         publishers = []
459         cursor.execute("SELECT DISTINCT name FROM publishers;")
460         for publisher in cursor:
461             publishers.append(publisher[0])
462 
463         return out(publishers, [])
464 
465     def add_book(details):
466         try:
467             author_name = "'"+details["author"]+"'"
468             cursor.execute(f"SELECT aid FROM authors WHERE name
LIKE {author_name};")
469             aid = cursor.fetchone()
470             publisher_name = "'"+details["publisher"]+"'"
471             cursor.execute(f"SELECT pid FROM publishers WHERE name
LIKE {publisher_name};")
472             pid = cursor.fetchone()
473             pid, aid = pid[0], aid[0]
474 
475             data = (
476                 aid,
477                 pid,
478                 details["title"],
479                 details["genre"],
480                 details["type"],
481                 details["isbn"],
482                 1,
483                 details["edition"],
484                 1,
485                 details["description"],
486                 details["location"]
487             )
488 
489             query = (
490                 "INSERT INTO books (aid, pid, title, genre, type,
isbn, availability, edition, no_of_copies, description, location)
VALUES"
491                 + "("
492                 + str(data[0])
493                 + ","
494                 + str(data[1])
495                 + ",'"
496                 + data[2]
```

```
            + "','"
            + data[3]
            + "','"
            + data[4]
            + "',"
            + str(data[5])
            + ","
            + str(data[6])
            + ","
            + str(data[7])
            + ","
            + str(data[8])
            + ",'"
            + data[9]
            + "','"
            + data[10]
            + "');"
        )
        cursor.execute(query)
        connection.commit()
        out = 99
    except:
        out = 9
    return out

#print(add_book({"title":"Harry Potter and the deathly hallows", "genre":"fiction - magic", "type":"Fiction", "isbn":5231568955566, "edition":1, "description":"harry potter book", "location":"Library shelf C", "author": "Harper Lee", "publisher": "Scholastic Press" }))

def delete_book(BID):
    book_details = get_book_info(BID)
    if book_details != 0:
        cursor.execute(f"DELETE FROM books WHERE bid = {BID};")
        connection.commit()
        out = book_details
    else:
        out = 0
    return out

def decrement_book(BID):
    book_details = get_book_info(BID)
    if book_details != 0:
        if book_details["no_of_copies"] > 1:
            cursor.execute(f"UPDATE books SET no_of_copies = no_of_copies - 1 WHERE bid = {BID};")
            connection.commit()
            out = 99
```

```
541             else:
542                 out = 11
543         else: out = 0
544 
545         return out
```

(3) – 'main.py' Python script containing the main graphical user interface and code.

```
  1 import customtkinter as ctk
  2 from tkinter import PhotoImage, Frame, CENTER
  3 from database_functions import *
  4 from CTkTable import *
  5 from CTkXYFrame import *
  6 from PIL import Image
  7 
  8 # app appearance
  9 ctk.set_appearance_mode("System")
 10 ctk.set_default_color_theme("dark-blue")
 11 ctk.set_window_scaling(1)
 12 
 13 root = ctk.CTk()
 14 
 15 
 16 # theme swticher
 17 def switch_theme(choice):
 18     ctk.set_appearance_mode(choice)
 19 
 20 
 21 # fonts
 22 font_1 = ctk.CTkFont(
 23     family="Opensans",
 24     size=19,
 25 )
 26 font_2 = ctk.CTkFont(family="Alegreya", size=23)
 27 font_3 = ctk.CTkFont(family="Gentium Book Plus", size=52)
 28 
 29 # configure window
 30 root.title("Library Manager")
 31 icon = PhotoImage(file="icons/icon.png")
 32 root.iconphoto(True, icon)
 33 root.minsize(1050, 650)
 34 
 35 # window grid configuration (1x2)
 36 root.rowconfigure(0, weight=1)
 37 root.columnconfigure(0, weight=1)
 38 root.columnconfigure(1, weight=15)
```

```
39
40  # issue frame
41  IssueFrame = ctk.CTkFrame(root, corner_radius=0)
42
43  # (6x3)
44  IssueFrame.columnconfigure(0, weight=1)
45  IssueFrame.columnconfigure(1, weight=2)
46  IssueFrame.columnconfigure(2, weight=1)
47  IssueFrame.rowconfigure(0, weight=1)
48  IssueFrame.rowconfigure(6, weight=2)
49
50
51  def issue():
52      global error_frame
53      mid = mid_entry.get()
54      bid = bid_entry.get()
55      timespan = timespan_entry.get()
56      try:
57          mid = int(mid)
58          bid = int(bid)
59          timespan = int(timespan)
60          x = lend(bid, mid, timespan)
61          if x == 99:
62              error = "Issue success!"
63              mid_entry.delete(0, "end")
64              bid_entry.delete(0, "end")
65              timespan_entry.delete(0, "end")
66          elif x == 4:
67              error = "Error! maximum number of rents reached"
68          elif x == 5:
69              error = "Error! book is not currently available"
70          elif x == 0:
71              error = "Error! book or member not found"
72          try:
73              error_frame.grid_remove()
74          except:
75              pass
76          error_frame = ctk.CTkFrame(IssueFrame)
77          error_frame.grid(row=6, column=0, columnspan=3,
pady=20, sticky="n")
78          error_message = ctk.CTkLabel(error_frame, text=error,
font=font_1)
79          error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)
80      except:
81          try:
82              error_frame.grid_remove()
83          except:
84              pass
```

```
  85          error_frame = ctk.CTkFrame(IssueFrame)
  86          error_frame.grid(row=6, column=0, columnspan=3,
pady=20, sticky="n")
  87          error_message = ctk.CTkLabel(
  88              error_frame, text="Error! Invalid entries",
font=font_1
  89          )
  90          error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)
  91
  92
  93  issue_label = ctk.CTkLabel(IssueFrame, text="Issue a book",
font=font_2)
  94  mid_entry = ctk.CTkEntry(IssueFrame, placeholder_text="Enter
member ID", font=font_1)
  95  bid_entry = ctk.CTkEntry(IssueFrame, placeholder_text="Enter
book ID", font=font_1)
  96  timespan_entry = ctk.CTkEntry(
  97      IssueFrame, placeholder_text="Enter timespan", font=font_1
  98  )
  99
 100  issue_label.grid(row=1, column=1, sticky="nsew", pady=7)
 101  mid_entry.grid(row=2, column=1, sticky="nsew", pady=7)
 102  bid_entry.grid(row=3, column=1, sticky="nsew", pady=7)
 103  timespan_entry.grid(row=4, column=1, sticky="nsew", pady=7)
 104
 105  issue_button = ctk.CTkButton(IssueFrame, text="Issue",
font=font_1, command=issue)
 106  issue_button.grid(row=5, column=1, sticky="nsw", pady=7)
 107
 108  # return frame
 109  ReturnFrame = ctk.CTkFrame(root, corner_radius=0)
 110
 111  ReturnFrame.columnconfigure(0, weight=1)
 112  ReturnFrame.columnconfigure(1, weight=2)
 113  ReturnFrame.columnconfigure(2, weight=1)
 114  ReturnFrame.rowconfigure(0, weight=1)
 115  ReturnFrame.rowconfigure(6, weight=2)
 116
 117
 118  def return_book_():
 119      global error_frame_2
 120      mid = mid_entry_2.get()
 121      bid = bid_entry_2.get()
 122      fee_per_day = fee_entry.get()
 123      try:
 124          mid = int(mid)
 125          bid = int(bid)
 126          fee_per_day = int(fee_per_day)
```

```
 127           x = return_book(bid, mid, fee_per_day)
 128           if x == 99:
 129               error = "Return success!"
 130               mid_entry_2.delete(0, "end")
 131               bid_entry_2.delete(0, "end")
 132               fee_entry.delete(0, "end")
 133           elif x == 0:
 134               error = "Error! book or member not found"
 135           elif x == 7:
 136               error = "Error! this book was not rented to this
member"
 137           else:
 138               error = f"Book returned. Warning! Fine of {x}
exists"
 139           try:
 140               error_frame_2.grid_remove()
 141           except:
 142               pass
 143           error_frame_2 = ctk.CTkFrame(ReturnFrame)
 144           error_frame_2.grid(row=6, column=0, columnspan=3,
pady=20, sticky="n")
 145           error_message = ctk.CTkLabel(error_frame_2,
text=error, font=font_1)
 146           error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)
 147       except:
 148           try:
 149               error_frame_2.grid_remove()
 150           except:
 151               pass
 152           error_frame_2 = ctk.CTkFrame(ReturnFrame)
 153           error_frame_2.grid(row=6, column=0, columnspan=3,
pady=20, sticky="n")
 154           error_message = ctk.CTkLabel(
 155               error_frame_2, text="Error! Invalid entries",
font=font_1
 156           )
 157           error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)
 158
 159
 160   return_label = ctk.CTkLabel(ReturnFrame, text="Return books",
font=font_2)
 161   mid_entry_2 = ctk.CTkEntry(ReturnFrame,
placeholder_text="Enter member ID", font=font_1)
 162   bid_entry_2 = ctk.CTkEntry(ReturnFrame,
placeholder_text="Enter book ID", font=font_1)
 163   fee_entry = ctk.CTkEntry(
 164       ReturnFrame, placeholder_text="Enter fine per day",
```

```
font=font_1
 165  )
 166  return_button = ctk.CTkButton(
 167      ReturnFrame, text="Return", font=font_1,
command=return_book_
 168  )
 169
 170  return_label.grid(row=1, column=1, sticky="nsew", pady=7)
 171  bid_entry_2.grid(row=3, column=1, sticky="nsew", pady=7)
 172  mid_entry_2.grid(row=2, column=1, sticky="nsew", pady=7)
 173  fee_entry.grid(row=4, column=1, sticky="nsew", pady=7)
 174  return_button.grid(row=5, column=1, sticky="nsw", pady=7)
 175
 176  # reserve frame
 177  ReserveFrame = ctk.CTkFrame(root, corner_radius=0)
 178
 179  ReserveFrame.columnconfigure(0, weight=1)
 180  ReserveFrame.columnconfigure(1, weight=2)
 181  ReserveFrame.columnconfigure(2, weight=1)
 182  ReserveFrame.rowconfigure(0, weight=1)
 183  ReserveFrame.rowconfigure(6, weight=2)
 184
 185
 186  def reserve():
 187      global error_frame_3
 188      mid = r_mid_entry.get()
 189      bid = r_bid_entry.get()
 190      timespan = r_time_entry.get()
 191      try:
 192          mid = int(mid)
 193          bid = int(bid)
 194          timespan = int(timespan)
 195          x = reserve_book(bid, mid, timespan)
 196          if x == 99:
 197              error = "Reservation success!"
 198              r_mid_entry.delete(0, "end")
 199              r_bid_entry.delete(0, "end")
 200              r_time_entry.delete(0, "end")
 201          elif x == 0:
 202              error = "Error! book or member not found"
 203          elif x == 8:
 204              error = "Error! this member cannot hold a book"
 205          elif x == 5:
 206              error = "Error! book is not currently available"
 207          elif x == 4:
 208              error = "Error! maximum number of reservations
reached"
 209          try:
 210              error_frame_3.grid_remove()
```

```
 211          except:
 212              pass
 213          error_frame_3 = ctk.CTkFrame(ReserveFrame)
 214          error_frame_3.grid(row=6, column=0, columnspan=3,
pady=20, sticky="n")
 215          error_message = ctk.CTkLabel(error_frame_3,
text=error, font=font_1)
 216          error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)
 217      except:
 218          try:
 219              error_frame_3.grid_remove()
 220          except:
 221              pass
 222          error_frame_3 = ctk.CTkFrame(ReserveFrame)
 223          error_frame_3.grid(row=6, column=0, columnspan=3,
pady=20, sticky="n")
 224          error_message = ctk.CTkLabel(
 225              error_frame_3, text="Error! Invalid entries",
font=font_1
 226          )
 227          error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)
 228
 229
 230  reserve_label = ctk.CTkLabel(ReserveFrame, text="Reserve
books", font=font_2)
 231  r_mid_entry = ctk.CTkEntry(
 232      ReserveFrame, placeholder_text="Enter member ID",
font=font_1
 233  )
 234  r_bid_entry = ctk.CTkEntry(
 235      ReserveFrame, placeholder_text="Enter book ID",
font=font_1
 236  )
 237  r_time_entry = ctk.CTkEntry(
 238      ReserveFrame, placeholder_text="Enter timespan",
font=font_1
 239  )
 240  reserve_button = ctk.CTkButton(
 241      ReserveFrame, text="Reserve", font=font_1, command=reserve
 242  )
 243
 244  reserve_label.grid(row=1, column=1, sticky="nsew", pady=7)
 245  r_mid_entry.grid(row=2, column=1, sticky="nsew", pady=7)
 246  r_bid_entry.grid(row=3, column=1, sticky="nsew", pady=7)
 247  r_time_entry.grid(row=4, column=1, sticky="nsew", pady=7)
 248  reserve_button.grid(row=5, column=1, sticky="nsw", pady=7)
 249
```

```
 250  # catalog query frame
 251  CatalogQueryFrame = ctk.CTkFrame(root, corner_radius=0)
 252  CatalogQueryFrame.columnconfigure(0, weight=1)
 253  CatalogQueryFrame.rowconfigure(0, weight=1)
 254
 255  tabview_3 = ctk.CTkTabview(CatalogQueryFrame)
 256  tabview_3.grid(row=0, column=0, sticky="nsew")
 257
 258  books_tab = tabview_3.add("Books")
 259  books_tab.rowconfigure(0, weight=1)
 260  books_tab.columnconfigure(0, weight=1)
 261  tabview_4 = ctk.CTkTabview(books_tab)
 262  tabview_4.grid(row=0, column=0, sticky="nsew")
 263  display_books_tab = tabview_4.add("Books List")
 264  display_books_tab.rowconfigure(0, weight=1)
 265  display_books_tab.columnconfigure(0, weight=1)
 266  books_frame = CTkXYFrame(display_books_tab)
 267  books_frame.grid(row=0, column=0, sticky="nsew")
 268  value2 = [["ID", "Title", "Genre", "Type", "ISBN",
"Availability", "Edition", "No_of_copies", "Location",
"Description"]]
 269  books_table = CTkTable(books_frame, row=0, column=0,
values=value2, font=font_1)
 270  books_table.grid(row=0, column=0, padx=10, sticky="nsew")
 271  n1 = 0
 272  def load_more_books():
 273      global error_frame_5
 274      global n1
 275      try:
 276          error_frame_5.grid_remove()
 277      except:
 278          pass
 279      books = display_books(10, n1)
 280      if books != 1:
 281          for i in range(len(books)):
 282              books_table.add_row(list(books[i]))
 283          n1 += len(books)
 284      else:
 285          error_frame_5 = ctk.CTkFrame(books_frame)
 286          error_message = ctk.CTkLabel(error_frame_5,
text="Cannot load more results!")
 287          error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)
 288          error_frame_5.grid(row=2, column=0, sticky="nsw",
padx=10, pady=10)
 289  load_more_books()
 290  load_more_books_btn = ctk.CTkButton(books_frame, text="Load
More", command=load_more_books)
 291  load_more_books_btn.grid(row=1, column=0, sticky="nw",
```

```
padx=15, pady=15)
 292  def refresh_books():
 293      global n1
 294      global error_frame_5
 295      try:
 296          error_frame_5.grid_remove()
 297      except:
 298          pass
 299      books_table.delete_rows(range(1, n1 + 1))
 300      n1 = 0
 301      load_more_books()
 302
 303
 304  refresh_books_btn = ctk.CTkButton(books_frame, text="Refresh
Results", command=refresh_books)
 305  refresh_books_btn.grid(row=1, column=0, sticky="nw", padx=165,
pady=15)
 306
 307  search_books_tab = tabview_4.add("Search Book")
 308  search_books_tab.columnconfigure(0, weight=1)
 309  search_books_tab.rowconfigure(0, weight=1)
 310  search_books_frame = ctk.CTkScrollableFrame(search_books_tab)
 311  search_books_frame.grid(sticky="nsew", row=0, column=0)
 312  search_books_frame.columnconfigure(0, weight=10)
 313  search_books_frame.columnconfigure(1, weight=1)
 314  search_label = ctk.CTkLabel(search_books_frame, text="Search
book by:", anchor="w", font=font_1)
 315  search_label.grid(row=0, column=0,sticky="nsew",padx=7,
pady=7)
 316  search_type = ctk.CTkOptionMenu(search_books_frame,
font=font_1, values=['Title', 'Author', 'Description'])
 317  search_type.grid(row=0, column=1, sticky="nsew", padx=7,
pady=7)
 318  books_entry = ctk.CTkEntry(search_books_frame,
placeholder_text=f"Search book", font=font_1)
 319  books_entry.grid(sticky="new", row=1, column=0, padx=(10,2),
pady=5)
 320
 321  def search_book():
 322      global error_frame_12
 323      book_name = books_entry.get()
 324      stype = search_type.get()
 325      if stype == "Author":
 326          x = search_book_by_author(book_name)
 327      elif stype == "Title":
 328          x = search_book_by_title(book_name)
 329      else:
 330          x = search_book_by_description(book_name)
 331      if x == 0:
```

```
 332           error = "No matches found"
 333           publisher_entry.delete(0, "end")
 334       else:
 335           error = ""
 336           authn = 1
 337           for bid in x:
 338               y = get_book_info(bid)
 339               error += f"Result {str(authn)} \n"
 340               authn +=1
 341               error += f" AID: {y['aid']} \n PID: {y['pid']} \n
Title: {y['title']} \n Genre: {y['genre']} \n Type: {y['type']} \n
ISBN: {y['isbn']} \n Edition: {y['edition']} \n Description:
{y['description']}\n\n"
 342       try:
 343           error_frame_12.grid_remove()
 344       except:
 345           pass
 346       error_frame_12 = ctk.CTkFrame(search_books_frame)
 347       error_frame_12.grid(row=2, column=0, columnspan=2,
pady=20, padx=10, sticky="n")
 348       error_message = ctk.CTkLabel(error_frame_12, text=error,
font=font_1, justify="left")
 349       error_message.grid(row=0, column=0, sticky="nsew", pady=7,
padx=15)
 350
 351
 352   s_book_button = ctk.CTkButton(search_books_frame,
text="Search", command=search_book)
 353   s_book_button.grid(row=1, column=1, padx=(5,5), pady=7,
sticky="new")
 354
 355
 356   authors_tab = tabview_3.add("Authors")
 357   authors_tab.rowconfigure(0, weight=1)
 358   authors_tab.columnconfigure(0, weight=1)
 359   tabview_5 = ctk.CTkTabview(authors_tab)
 360   tabview_5.grid(row=0, column=0, sticky="nsew")
 361   display_authors_tab = tabview_5.add("Authors List")
 362   display_authors_tab.rowconfigure(0, weight=1)
 363   display_authors_tab.columnconfigure(0, weight=1)
 364   authors_frame = CTkXYFrame(display_authors_tab)
 365   authors_frame.grid(row=0, column=0, sticky="nsew")
 366   value3 = [["ID", "Name", "Gender", "Date of Birth", "Country",
"Info", "Phone", "Contact"]]
 367   authors_table = CTkTable(authors_frame, row=0, column=0,
values=value3, font=font_1)
 368   authors_table.grid(row=0, column=0, padx=10, sticky="nsew")
 369
 370   n2 = 0
```

```
 371  def load_more_authors():
 372      global error_frame_6
 373      global n2
 374      try:
 375          error_frame_6.grid_remove()
 376      except:
 377          pass
 378      authors = display_authors(10, n2)
 379      if authors != 1:
 380          for i in range(len(authors)):
 381              authors_table.add_row(list(authors[i]))
 382          n2 += len(authors)
 383      else:
 384          error_frame_6 = ctk.CTkFrame(authors_frame)
 385          error_message = ctk.CTkLabel(error_frame_6,
text="Cannot load more results!")
 386          error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)
 387          error_frame_6.grid(row=2, column=0, sticky="nsw",
padx=10, pady=10)
 388  load_more_authors()
 389  load_more_authors_btn = ctk.CTkButton(authors_frame,
text="Load More", command=load_more_authors)
 390  load_more_authors_btn.grid(row=1, column=0, sticky="nw",
padx=15, pady=15)
 391  def refresh_authors():
 392      global n2
 393      global error_frame_6
 394      try:
 395          error_frame_6.grid_remove()
 396      except:
 397          pass
 398      authors_table.delete_rows(range(1, n2 + 1))
 399      n2 = 0
 400      load_more_authors()
 401
 402
 403  refresh_authors_btn = ctk.CTkButton(authors_frame,
text="Refresh Results", command=refresh_authors)
 404  refresh_authors_btn.grid(row=1, column=0, sticky="nw",
padx=165, pady=15)
 405  search_authors_tab = tabview_5.add("Search Author")
 406  search_authors_tab.columnconfigure(0, weight=1)
 407  search_authors_tab.rowconfigure(0, weight=1)
 408  search_authors_frame =
ctk.CTkScrollableFrame(search_authors_tab)
 409  search_authors_frame.grid(sticky="nsew", row=0, column=0)
 410  search_authors_frame.columnconfigure(0, weight=10)
 411  search_authors_frame.columnconfigure(1, weight=1)
```

```
 412  author_entry = ctk.CTkEntry(search_authors_frame,
placeholder_text="Search author by name", font=font_1, width=600)
 413  author_entry.grid(sticky="new", row=0, column=0, padx=(10,2),
pady=5)
 414
 415  def search_author():
 416      global error_frame_10
 417      author_name = author_entry.get()
 418      x = search_author_by_name(author_name)
 419      if x == 0:
 420          error = "No matches found"
 421          author_entry.delete(0, "end")
 422      else:
 423          error = ""
 424          authn = 1
 425          for aid in x:
 426              y = get_author_info(aid)
 427              error += f"Result {str(authn)} \n"
 428              authn +=1
 429              error += f" Name: {y['name']} \n Gender:
{y['gender']} \n Date of Birth: {y['dob']} \n Country:
{y['country']} \n Info: {y['info']} \n Phone: {y['phone']}\n\n"
 430      try:
 431          error_frame_10.grid_remove()
 432      except:
 433          pass
 434      error_frame_10 = ctk.CTkFrame(search_authors_frame)
 435      error_frame_10.grid(row=1, column=0, columnspan=2,
pady=20, padx=10, sticky="n")
 436      error_message = ctk.CTkLabel(error_frame_10, text=error,
font=font_1, justify="left")
 437      error_message.grid(row=0, column=0, sticky="nsew", pady=7,
padx=15)
 438
 439
 440  s_author_button = ctk.CTkButton(search_authors_frame,
text="Search", command=search_author)
 441  s_author_button.grid(row=0, column=1, padx=(5,5), pady=7,
sticky="new")
 442
 443
 444  publishers_tab = tabview_3.add("Publishers")
 445  publishers_tab.rowconfigure(0, weight=1)
 446  publishers_tab.columnconfigure(0, weight=1)
 447  tabview_6 = ctk.CTkTabview(publishers_tab)
 448  tabview_6.grid(row=0, column=0, sticky="nsew")
 449  display_publishers_tab = tabview_6.add("Publishers List")
 450  display_publishers_tab.rowconfigure(0, weight=1)
 451  display_publishers_tab.columnconfigure(0, weight=1)
```

```
 452  publishers_frame =
ctk.CTkScrollableFrame(display_publishers_tab)
 453  publishers_frame.grid(row=0, column=0, sticky="nsew")
 454  value4 = [["ID", "Name", "Contact", "Details"]]
 455  publishers_table = CTkTable(publishers_frame, row=0, column=0,
values=value4, font=font_1)
 456  publishers_table.grid(row=0, column=0, padx=10, sticky="nsew")
 457
 458
 459
 460  n3 = 0
 461  def load_more_publishers():
 462      global error_frame_7
 463      global n3
 464      try:
 465          error_frame_7.grid_remove()
 466      except:
 467          pass
 468      publishers = display_publishers(10, n3)
 469      if publishers != 1:
 470          for i in range(len(publishers)):
 471              publishers_table.add_row(list(publishers[i]))
 472          n3 += len(publishers)
 473      else:
 474          error_frame_7 = ctk.CTkFrame(publishers_frame)
 475          error_message = ctk.CTkLabel(error_frame_7,
text="Cannot load more results!")
 476          error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)
 477          error_frame_7.grid(row=2, column=0, sticky="nsw",
padx=10, pady=10)
 478  load_more_publishers()
 479  load_more_publishers_btn = ctk.CTkButton(publishers_frame,
text="Load More", command=load_more_publishers)
 480  load_more_publishers_btn.grid(row=1, column=0, sticky="nw",
padx=15, pady=15)
 481  def refresh_publishers():
 482      global n3
 483      global error_frame_7
 484      try:
 485          error_frame_7.grid_remove()
 486      except:
 487          pass
 488      publishers_table.delete_rows(range(1, n3 + 1))
 489      n3 = 0
 490      load_more_publishers()
 491
 492
 493  refresh_publishers_btn = ctk.CTkButton(publishers_frame,
```

```
text="Refresh Results", command=refresh_publishers)
 494  refresh_publishers_btn.grid(row=1, column=0, sticky="nw",
padx=165, pady=15)
 495  search_publishers_tab = tabview_6.add("Search Publisher")
 496  search_publishers_tab.columnconfigure(0, weight=1)
 497  search_publishers_tab.rowconfigure(0, weight=1)
 498  search_publishers_frame =
ctk.CTkScrollableFrame(search_publishers_tab)
 499  search_publishers_frame.grid(sticky="nsew", row=0, column=0)
 500  search_publishers_frame.columnconfigure(0, weight=10)
 501  search_publishers_frame.columnconfigure(1, weight=1)
 502  publisher_entry = ctk.CTkEntry(search_publishers_frame,
placeholder_text="Search publisher by name", font=font_1)
 503  publisher_entry.grid(sticky="new", row=0, column=0,
padx=(10,2), pady=5)
 504
 505  def search_publisher():
 506      global error_frame_11
 507      publisher_name = publisher_entry.get()
 508      x = search_publisher_by_name(publisher_name)
 509      if x == 0:
 510          error = "No matches found"
 511          publisher_entry.delete(0, "end")
 512      else:
 513          error = ""
 514          authn = 1
 515          for pid in x:
 516              y = get_publisher_info(pid)
 517              error += f"Result {str(authn)} \n"
 518              authn +=1
 519              error += f" Name: {y['name']} \n Contact:
{y['contact']} \n Details: {y['details']}\n\n"
 520      try:
 521          error_frame_11.grid_remove()
 522      except:
 523          pass
 524      error_frame_11 = ctk.CTkFrame(search_publishers_frame)
 525      error_frame_11.grid(row=1, column=0, columnspan=2,
pady=20, padx=10, sticky="n")
 526      error_message = ctk.CTkLabel(error_frame_11, text=error,
font=font_1, justify="left")
 527      error_message.grid(row=0, column=0, sticky="nsew", pady=7,
padx=15)
 528
 529
 530  s_publisher_button = ctk.CTkButton(search_publishers_frame,
text="Search", command=search_publisher)
 531  s_publisher_button.grid(row=0, column=1, padx=(5,5), pady=7,
sticky="new")
```

```
532
533 # catalog update frame
534 CatalogUpdateFrame = ctk.CTkFrame(root, corner_radius=0)
535 CatalogUpdateFrame.columnconfigure(0, weight=1)
536 CatalogUpdateFrame.rowconfigure(0, weight=1)
537
538 tabview_8 =ctk.CTkTabview(CatalogUpdateFrame)
539 tabview_8.grid(row=0, column=0, sticky="nsew")
540 book_tab = tabview_8.add("Books")
541 book_tab.rowconfigure(0, weight=1)
542 book_tab.columnconfigure(0, weight=1)
543 tabview_7 = ctk.CTkTabview(book_tab)
544 tabview_7.grid(row=0, column=0, sticky="nsew")
545 add_book_tab = tabview_7.add("Add Book")
546 add_book_tab.columnconfigure(0, weight=3)
547 add_book_tab.columnconfigure(1, weight=1)
548 add_book_tab.columnconfigure(2, weight=2)
549 add_book_tab.columnconfigure(3, weight=3)
550 add_book_tab.rowconfigure(0, weight=1)
551 add_book_tab.rowconfigure(12, weight=2)
552
553 book_types = get_book_types()
554 author_names = get_author_names()
555 publishers = get_publishers()
556
557 def add_a_book():
558     global error_frame_13
559     title_b = title.get()
560     isbn_b = isbn.get()
561     genre_b = genre.get()
562     type_b = book_type.get()
563     edition_b = edition.get()
564     description_b = description.get()
565     location_b = location.get()
566     author_b = author.get()
567     publisher_b = publisher.get()
568     try:
569         isbn_b = int(isbn_b)
570         edition_b = int(edition_b)
571         details = dict()
572         details["title"] = title_b
573         details["isbn"] = isbn_b
574         details["genre"] = genre_b
575         details["type"] = type_b
576         details["edition"] = edition_b
577         details["description"] = description_b
578         details["location"] = location_b
579         details["author"] = author_b
580         details["publisher"] = publisher_b
```

```
            x = add_book(details)
            if x == 99:
                error = "Book successfully added!"
                title.delete(0, "end")
                isbn.delete(0, "end")
                genre.delete(0, "end")
                edition.delete(0, "end")
                description.delete(0, "end")
                location.delete(0, "end")
            elif x == 9:
                error = "Error! maximum length exceeded"
            try:
                error_frame_13.grid_remove()
            except:
                pass
            error_frame_13 = ctk.CTkFrame(add_book_tab)
            error_frame_13.grid(row=12, column=0, columnspan=4,
pady=20, sticky="n")
            error_message = ctk.CTkLabel(error_frame_13,
text=error, font=font_1)
            error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)
        except:
            try:
                error_frame_13.grid_remove()
            except:
                pass
            error_frame_13 = ctk.CTkFrame(add_book_tab)
            error_frame_13.grid(row=12, column=0, columnspan=4,
pady=20, sticky="n")
            error_message = ctk.CTkLabel(
                error_frame_13, text="Error! Invalid entries",
font=font_1
            )
            error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)


    add_book_label = ctk.CTkLabel(add_book_tab, font=font_2,
text="Add a new book")
    isbn = ctk.CTkEntry(add_book_tab, font=font_1,
placeholder_text="Book's ISBN")
    title = ctk.CTkEntry(add_book_tab, font=font_1,
placeholder_text="Book's title")
    genre = ctk.CTkEntry(add_book_tab, font=font_1,
placeholder_text="Book's genre")
    type_label = ctk.CTkLabel(add_book_tab, font=font_1,
text="Book Type:", anchor="w")
    book_type = ctk.CTkComboBox(add_book_tab, font=font_1,
```

```
values=book_types)
 619  edition = ctk.CTkEntry(add_book_tab, font=font_1,
placeholder_text="Book Edition")
 620  description = ctk.CTkEntry(add_book_tab, font=font_1,
placeholder_text="Book Description")
 621  location = ctk.CTkEntry(add_book_tab, font=font_1,
placeholder_text="Book's shelf location")
 622  author_label = ctk.CTkLabel(add_book_tab, font=font_1,
text="Author:", anchor="w")
 623  author = ctk.CTkOptionMenu(add_book_tab, font=font_1,
values=author_names)
 624  publisher_label = ctk.CTkLabel(add_book_tab, font=font_1,
text="Publisher:", anchor="w")
 625  publisher = ctk.CTkOptionMenu(add_book_tab, font=font_1,
values=publishers)
 626  add_book_button = ctk.CTkButton(add_book_tab, font=font_1,
text="Add Book", command=add_a_book)
 627
 628  add_book_label.grid(row=1, column=1, columnspan=2,
sticky="nsew", pady=4)
 629  isbn.grid(row=2, column=1, columnspan=2, sticky="nsew",
pady=4)
 630  title.grid(row=3, column=1, columnspan=2, sticky="nsew",
pady=4)
 631  genre.grid(row=4, column=1, columnspan=2, sticky="nsew",
pady=4)
 632  type_label.grid(row=5, column=1, sticky="nsew", pady=4)
 633  book_type.grid(row=5, column=2, sticky="nsew", pady=4)
 634  edition.grid(row=6, column=1, columnspan=2, sticky="nsew",
pady=4)
 635  description.grid(row=7, column=1, columnspan=2, sticky="nsew",
pady=4)
 636  location.grid(row=8, column=1, columnspan=2, sticky="nsew",
pady=4)
 637  author_label.grid(row=9, column=1, sticky="nsew", pady=4)
 638  author.grid(row=9, column=2, sticky="nsew", pady=4)
 639  publisher_label.grid(row=10, column=1, sticky="nsew", pady=4)
 640  publisher.grid(row=10, column=2, sticky="nsew", pady=4)
 641  add_book_button.grid(row=11, column=1, sticky="nw", pady=4)
 642
 643  del_book_tab = tabview_7.add("Remove a Book")
 644  del_book_tab.columnconfigure(0, weight=1)
 645  del_book_tab.columnconfigure(1, weight=2)
 646  del_book_tab.columnconfigure(2, weight=1)
 647  del_book_tab.rowconfigure(0, weight=1)
 648  del_book_tab.rowconfigure(4, weight=2)
 649
 650  def decrement_book_count():
 651      global error_frame_15
```

```
652       bid = book_id_entry.get()
653       try:
654           bid = int(bid)
655           x = decrement_book(bid)
656           if x == 0:
657               error = "Error! book or member not found"
658           elif x == 11:
659               error = "Error! only one book left"
660           elif x == 99:
661               error = "Book count decremented by 1"
662               book_id_entry.delete(0, "end")
663           try:
664               error_frame_15.grid_remove()
665               error_frame_14.grid_remove()
666           except:
667               pass
668           error_frame_15 = ctk.CTkFrame(del_book_tab)
669           error_frame_15.grid(row=4, column=0, columnspan=3,
pady=20, sticky="n")
670           error_message = ctk.CTkLabel(error_frame_15,
text=error, font=font_1, justify="left")
671           error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)
672       except:
673           try:
674               error_frame_15.grid_remove()
675               error_frame_14.grid_remove()
676           except:
677               pass
678           error_frame_15 = ctk.CTkFrame(del_book_tab)
679           error_frame_15.grid(row=4, column=0, columnspan=3,
pady=20, sticky="n")
680           error_message = ctk.CTkLabel(
681               error_frame_15, text="Error! invalid entries",
font=font_1
682           )
683           error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)
684   def delete_a_book():
685       global error_frame_14
686       bid = book_id_entry.get()
687       try:
688           bid = int(bid)
689           x = delete_book(bid)
690           if x == 0:
691               error = "Error! book or member not found"
692           else:
693               error = "Book successfully deleted!"
694               error += f"\n\n Title: {x["title"]}\nISBN:
```

```
{x["isbn"]}\nDescription: {x["description"]}\nEdition:
{x["edition"]}"
 695                 book_id_entry.delete(0, "end")
 696             try:
 697                 error_frame_14.grid_remove()
 698                 error_frame_15.grid_remove()
 699             except:
 700                 pass
 701             error_frame_14 = ctk.CTkFrame(del_book_tab)
 702             error_frame_14.grid(row=4, column=0, columnspan=3,
pady=20, sticky="n")
 703             error_message = ctk.CTkLabel(error_frame_14,
text=error, font=font_1)
 704             error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)
 705         except:
 706             try:
 707                 error_frame_14.grid_remove()
 708                 error_frame_15.grid_remove()
 709             except:
 710                 pass
 711             error_frame_14 = ctk.CTkFrame(del_book_tab)
 712             error_frame_14.grid(row=4, column=0, columnspan=3,
pady=20, sticky="n")
 713             error_message = ctk.CTkLabel(
 714                 error_frame_14, text="Error! invalid entries",
font=font_1
 715             )
 716             error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)
 717
 718
 719     del_book_label = ctk.CTkLabel(del_book_tab, font=font_2,
text="Remove a book")
 720     book_id_entry = ctk.CTkEntry(del_book_tab, font=font_1,
placeholder_text="Book ID")
 721     decrement_book_btn = ctk.CTkButton(del_book_tab, font=font_1,
text="Decrement book count", command=decrement_book_count)
 722     del_book_btn = ctk.CTkButton(del_book_tab, font=font_1,
text="Delete book", command=delete_a_book)
 723
 724     del_book_label.grid(row=1, column=1, pady=7, sticky="nsew")
 725     book_id_entry.grid(row=2, column=1, pady=7, sticky="nsew")
 726     decrement_book_btn.grid(row=3, column=1, pady=7, sticky="nsw")
 727     del_book_btn.grid(row=3, column=1, pady=7, sticky="nsw",
padx=(235,0))
 728
 729     # member query frame
 730     MemberQueryFrame = ctk.CTkFrame(root, corner_radius=0)
```

```
 731
 732  MemberQueryFrame.columnconfigure(0, weight=1)
 733  MemberQueryFrame.rowconfigure(0, weight=1)
 734
 735  tabview_1 = ctk.CTkTabview(MemberQueryFrame)
 736  tabview_1.grid(row=0, column=0, sticky="nsew")
 737
 738  display_tab = tabview_1.add("Display Members")
 739  dsply_frame = CTkXYFrame(display_tab)
 740  dsply_frame.grid(row=0, column=0, sticky="nsew")
 741  value = [["ID", "Name", "Address", "Phone", "Gender", "Class",
"No. of books rented"]]
 742  member_table = CTkTable(dsply_frame, row=0,column=0,
values=value, font=font_1)
 743  member_table.grid(row=0, column=0, padx=10, sticky="nsew")
 744
 745  n = 0
 746  def load_more():
 747      global error_frame_4
 748      global n
 749      try:
 750          error_frame_4.grid_remove()
 751      except:
 752          pass
 753      members = display_members(10, n)
 754      if members != 1:
 755          for i in range(len(members)):
 756              member_table.add_row(list(members[i]))
 757          n += len(members)
 758      else:
 759          error_frame_4 = ctk.CTkFrame(dsply_frame)
 760          error_message = ctk.CTkLabel(error_frame_4,
text="Cannot load more results!")
 761          error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)
 762          error_frame_4.grid(row=2, column=0, sticky="nsw",
padx=10, pady=10)
 763
 764
 765  load_more()
 766  load_more_btn = ctk.CTkButton(dsply_frame, text="Load More",
command=load_more)
 767  load_more_btn.grid(row=1, column=0, sticky="nw", padx=15,
pady=15)
 768
 769
 770  def refresh():
 771      global n
 772      global error_frame_4
```

```
 773      try:
 774          error_frame_4.grid_remove()
 775      except:
 776          pass
 777      member_table.delete_rows(range(1, n + 1))
 778      n = 0
 779      load_more()
 780
 781
 782  refresh_btn = ctk.CTkButton(dsply_frame, text="Refresh
Results", command=refresh)
 783  refresh_btn.grid(row=1, column=0, sticky="nw", padx=165,
pady=15)
 784
 785  display_tab.columnconfigure(0, weight=1)
 786  display_tab.rowconfigure(0, weight=1)
 787
 788  search_tab = tabview_1.add("View Member Details")
 789  search_tab.columnconfigure(0, weight=10)
 790  search_tab.columnconfigure(1, weight=1)
 791
 792  query_input = ctk.CTkEntry(search_tab, placeholder_text="Enter
member ID")
 793  query_input.grid(row=0, column=0, sticky="new", padx=(15, 3),
pady=15)
 794
 795
 796  def search_member():
 797      global error_frame_5
 798      mid = query_input.get()
 799      try:
 800          mid = int(mid)
 801          x = get_member_info(mid)
 802          if x == 0:
 803              error = "Error! member not found"
 804          else:
 805              error = f" Name: {x['name']} \n Address:
{x['address']} \n Phone: {x['phone']} \n Gender: {x['gender']} \n
Class: {x['class']} \n No. of books rented:
{x['no_of_books_rented']}"
 806          try:
 807              error_frame_5.grid_remove()
 808          except:
 809              pass
 810          error_frame_5 = ctk.CTkFrame(search_tab)
 811          error_frame_5.grid(
 812              row=1, column=0, columnspan=2, pady=20,
sticky="new", padx=15
 813          )
```

```
            error_message = ctk.CTkLabel(
                error_frame_5, text=error, font=font_1,
justify="left"
            )
            error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)
        except:
            try:
                error_frame_5.grid_remove()
            except:
                pass
            error_frame_5 = ctk.CTkFrame(search_tab)
            error_frame_5.grid(
                row=1, column=0, columnspan=2, pady=20,
sticky="new", padx=15
            )
            error_message = ctk.CTkLabel(
                error_frame_5, text="Error! Invalid entries",
font=font_1
            )
            error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)


    search_btn = ctk.CTkButton(search_tab, text="Search",
command=search_member)
    search_btn.grid(row=0, column=1, sticky="new", padx=(5, 15),
pady=15)

    # member update frame
    MupdateFrame = ctk.CTkFrame(root, corner_radius=0)

    MupdateFrame.columnconfigure(0, weight=1)
    MupdateFrame.rowconfigure(0, weight=1)

    tabview_2 = ctk.CTkTabview(MupdateFrame)
    tabview_2.grid(row=0, column=0, sticky="nsew")

    add_tab = tabview_2.add("Add Members")
    add_tab.rowconfigure(0, weight=1)
    add_tab.rowconfigure(8, weight=2)
    add_tab.columnconfigure(0, weight=4)
    add_tab.columnconfigure(1, weight=1)
    add_tab.columnconfigure(2, weight=2)
    add_tab.columnconfigure(3, weight=4)


    def add_a_member():
        global error_frame_6
```

```
 856       name = member_name_entry.get()
 857       address = address_entry.get()
 858       phone = phone_entry.get()
 859       gender = gender_entry.get()
 860       m_class = membership_entry.get()
 861       try:
 862           phone = int(phone)
 863           m_class = int(m_class)
 864           details = dict()
 865           details["name"] = name
 866           details["address"] = address
 867           details["phone"] = phone
 868           details["gender"] = gender
 869           details["class"] = m_class
 870           x = add_member(details)
 871           if x == 99:
 872               error = "Member succesfully added!"
 873               member_name_entry.delete(0, "end")
 874               address_entry.delete(0, "end")
 875               phone_entry.delete(0, "end")
 876           elif x == 9:
 877               error = "Error! maximum input length exceeded"
 878           try:
 879               error_frame_6.grid_remove()
 880           except:
 881               pass
 882           error_frame_6 = ctk.CTkFrame(add_tab)
 883           error_frame_6.grid(row=8, column=0, columnspan=4,
pady=20, sticky="n")
 884           error_message = ctk.CTkLabel(error_frame_6,
text=error, font=font_1)
 885           error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)
 886       except:
 887           try:
 888               error_frame_6.grid_remove()
 889           except:
 890               pass
 891           error_frame_6 = ctk.CTkFrame(add_tab)
 892           error_frame_6.grid(row=8, column=0, columnspan=4,
pady=20, sticky="n")
 893           error_message = ctk.CTkLabel(
 894               error_frame_6, text="Error! Invalid entries",
font=font_1
 895           )
 896           error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)
 897
 898
```

```
 899  add_member_label = ctk.CTkLabel(add_tab, text="Add a Member",
font=font_2)
 900  member_name_entry = ctk.CTkEntry(add_tab,
placeholder_text="Member name", font=font_1)
 901  address_entry = ctk.CTkEntry(add_tab,
placeholder_text="Member's adress", font=font_1)
 902  phone_entry = ctk.CTkEntry(
 903      add_tab, placeholder_text="Member's phone number",
font=font_1
 904  )
 905  gender_label = ctk.CTkLabel(add_tab, text="Member's Gender:",
font=font_1, anchor="w")
 906  gender_entry = ctk.CTkOptionMenu(
 907      add_tab, font=font_1, values=["Male", "Female", "Other"]
 908  )
 909  membership_label = ctk.CTkLabel(
 910      add_tab, text="Membership Class:", font=font_1, anchor="w"
 911  )
 912  membership_entry = ctk.CTkOptionMenu(add_tab, font=font_1,
values=["1", "2", "3"])
 913  add_member_button = ctk.CTkButton(
 914      add_tab, text="Add Member", font=font_1,
command=add_a_member
 915  )
 916
 917  add_member_label.grid(row=1, column=1, columnspan=2,
sticky="nsew", pady=7)
 918  member_name_entry.grid(row=2, column=1, columnspan=2,
sticky="nsew", pady=7)
 919  address_entry.grid(row=3, column=1, columnspan=2,
sticky="nsew", pady=7)
 920  phone_entry.grid(row=4, column=1, columnspan=2, sticky="nsew",
pady=7)
 921  gender_label.grid(row=5, column=1, sticky="nsew", pady=7)
 922  gender_entry.grid(row=5, column=2, sticky="nsew", pady=7)
 923  membership_label.grid(row=6, column=1, sticky="nsew", pady=7)
 924  membership_entry.grid(row=6, column=2, sticky="nsew", pady=7)
 925  add_member_button.grid(row=7, column=1, sticky="nsw", pady=7)
 926
 927  del_tab = tabview_2.add("Remove Members")
 928  del_tab.rowconfigure(0, weight=1)
 929  del_tab.rowconfigure(4, weight=2)
 930  del_tab.columnconfigure(0, weight=2)
 931  del_tab.columnconfigure(1, weight=3)
 932  del_tab.columnconfigure(2, weight=2)
 933
 934
 935  def del_a_member():
 936      global error_frame_7
```

```
 937       mid = member_id.get()
 938       sticky_value = "n"
 939       try:
 940           mid = int(mid)
 941           y = get_member_info(mid)
 942           x = del_member(mid)
 943           if x == 99:
 944               error = "Member successfully removed!\n\n"
 945               details = f" Name: {y['name']} \n Address:
{y['address']} \n Phone: {y['phone']} \n Gender: {y['gender']} \n
Class: {y['class']} \n No. of books rented:
{y['no_of_books_rented']}"
 946               error += details
 947               member_id.delete(0, "end")
 948               sticky_value="new"
 949           elif x == 0:
 950               error = "Error! member not found"
 951           elif x == 10:
 952               error = "Error! member has unreturned book"
 953           try:
 954               error_frame_7.grid_remove()
 955           except:
 956               pass
 957           error_frame = ctk.CTkFrame(del_tab)
 958           error_frame.grid(row=4, column=1, columnspan=1,
pady=20, sticky=sticky_value)
 959           error_message = ctk.CTkLabel(error_frame, text=error,
font=font_1, justify="left")
 960           error_message.grid(row=0, column=0, columnspan=3,
sticky="nsew", pady=7, padx=15)
 961       except:
 962           try:
 963               error_frame_7.grid_remove()
 964           except:
 965               pass
 966           error_frame = ctk.CTkFrame(del_tab)
 967           error_frame.grid(row=4, column=1, columnspan=1,
pady=20, sticky=sticky_value)
 968           error_message = ctk.CTkLabel(
 969               error_frame, text="Error! Invalid entries",
font=font_1
 970           )
 971           error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)
 972
 973
 974   del_member_label = ctk.CTkLabel(del_tab, text="Remove a
Member", font=font_2)
 975   member_id = ctk.CTkEntry(del_tab, placeholder_text="Member ID
```

```
for removal", font=font_1)
    del_member_button = ctk.CTkButton(
        del_tab, text="Remove Member", font=font_1,
command=del_a_member
    )

    del_member_label.grid(row=1, column=1, sticky="nsew", pady=7)
    member_id.grid(row=2, column=1, sticky="nsew", pady=7)
    del_member_button.grid(row=3, column=1, sticky="nsw", pady=7)

    # greet frame
    GreetFrame = ctk.CTkFrame(root, corner_radius=0)
    greet_label = ctk.CTkLabel(GreetFrame, text="Library Manager
v1.0", font=font_3)
    greet_label.place(relx=0.5, rely=0.6, anchor=CENTER)
    image =
ctk.CTkImage(light_image=Image.open("icons/image.png"), size=(160,
160))
    image_label = ctk.CTkLabel(GreetFrame, image=image, text="")
    image_label.place(relx=0.5, rely=0.4, anchor=CENTER)

    # change the frame on button press function
    active_frame = IssueFrame

    def change_frame(frame):
        global active_frame
        if frame != active_frame:
            active_frame.grid_remove()
            frame.grid(row=0, column=1, sticky="nsew")
            active_frame = frame


    change_frame(GreetFrame)

    # sidebar
    sidebar = ctk.CTkFrame(root, corner_radius=0)
    sidebar.grid(row=0, column=0, sticky="nsew")

    # sidebar grid configuration(1x12)
    sidebar.columnconfigure(0, weight=1)
    sidebar.rowconfigure(12, weight=5)

    seperator = Frame(sidebar, width=2, bg="#606060")
    seperator.place(relx=0.99, relheight=1)

    label_1 = ctk.CTkLabel(sidebar, text="Circulation",
font=font_2)
    button_1 = ctk.CTkButton(
        sidebar, text="Issue", font=font_1, command=lambda:
```

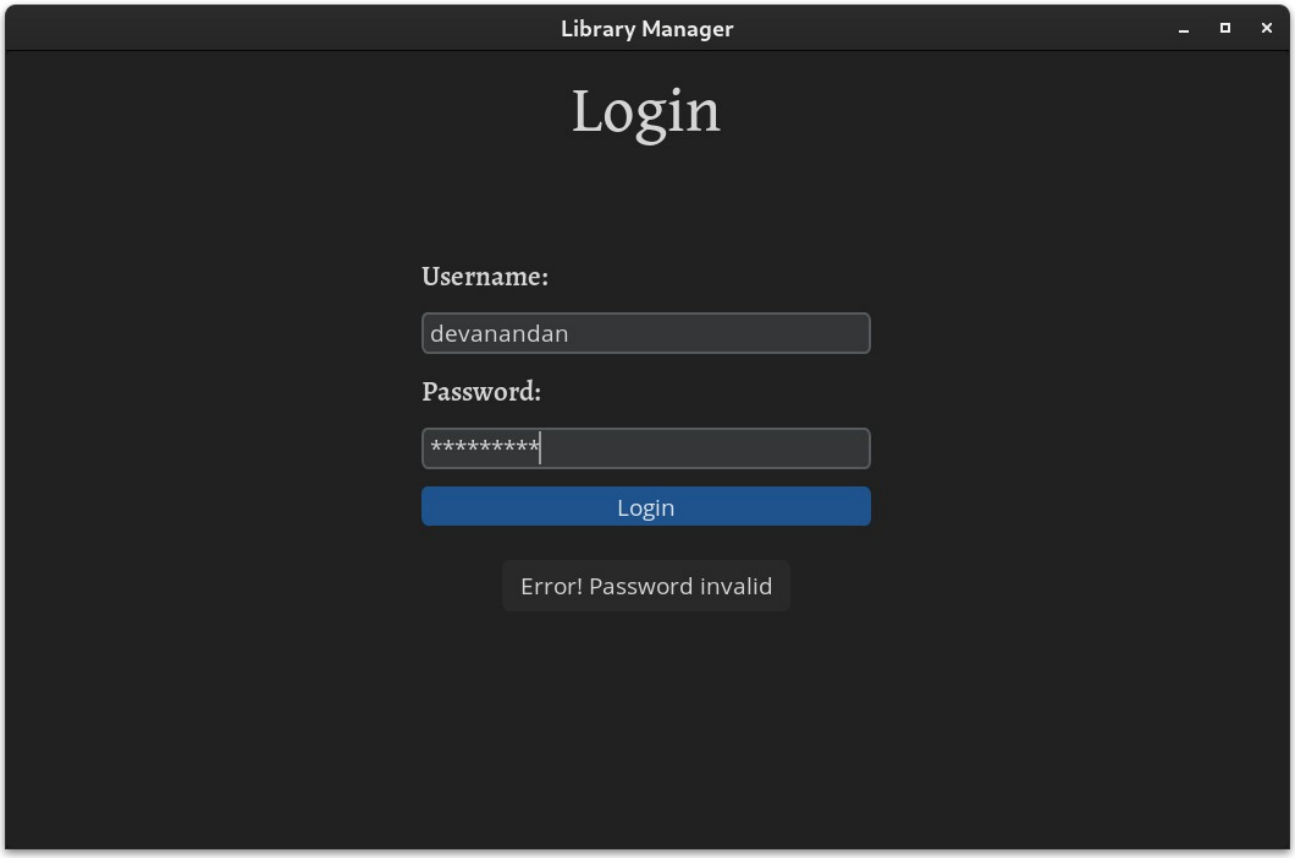
```
change_frame(IssueFrame)
1019  )
1020  button_2 = ctk.CTkButton(
1021      sidebar, text="Return", font=font_1, command=lambda:
change_frame(ReturnFrame)
1022  )
1023  button_3 = ctk.CTkButton(
1024      sidebar, text="Reserve", font=font_1, command=lambda:
change_frame(ReserveFrame)
1025  )
1026  label_2 = ctk.CTkLabel(sidebar, text="Catalog", font=font_2)
1027  button_4 = ctk.CTkButton(sidebar, text="Query", font=font_1,
command=lambda: change_frame(CatalogQueryFrame))
1028  button_5 = ctk.CTkButton(sidebar, text="Update records",
font=font_1, command=lambda: change_frame(CatalogUpdateFrame))
1029  label_3 = ctk.CTkLabel(sidebar, text="Members", font=font_2)
1030  button_6 = ctk.CTkButton(
1031      sidebar, text="Query", font=font_1, command=lambda:
change_frame(MemberQueryFrame)
1032  )
1033  button_7 = ctk.CTkButton(
1034      sidebar,
1035      text="Update records",
1036      font=font_1,
1037      command=lambda: change_frame(MupdateFrame),
1038  )
1039  label_4 = ctk.CTkLabel(sidebar, text="Statistics",
font=font_2)
1040  button_8 = ctk.CTkButton(sidebar, text="Book popularity",
font=font_1)
1041  theme_button = ctk.CTkOptionMenu(
1042      sidebar,
1043      values=["System", "Light", "Dark"],
1044      command=switch_theme,
1045      anchor="center",
1046      font=font_1,
1047  )
1048
1049  label_1.grid(row=0, sticky="nsew", padx=30, pady=(10, 2))
1050  button_1.grid(row=1, sticky="nsew", padx=30, pady=2)
1051  button_2.grid(row=2, sticky="nsew", padx=30, pady=2)
1052  button_3.grid(row=3, sticky="nsew", padx=30, pady=2)
1053  label_2.grid(row=4, sticky="nsew", padx=30, pady=2)
1054  button_4.grid(row=5, sticky="nsew", padx=30, pady=2)
1055  button_5.grid(row=6, sticky="nsew", padx=30, pady=2)
1056  label_3.grid(row=7, sticky="nsew", padx=30, pady=2)
1057  button_6.grid(row=8, sticky="nsew", padx=30, pady=2)
1058  button_7.grid(row=9, sticky="nsew", padx=30, pady=2)
1059  label_4.grid(row=10, sticky="nsew", padx=30, pady=2)
```

```
1060  button_8.grid(row=11, sticky="nsew", padx=30, pady=2)
1061  theme_button.grid(row=13, padx=30, sticky="nsew", pady=(2,
19))
1062
1063  # login frame for signing in
1064  LoginFrame = ctk.CTkFrame(root)
1065  LoginFrame.grid(row=0, column=0, columnspan=2, sticky="nsew")
1066
1067  # login frame grid configuration (7x3)
1068  LoginFrame.columnconfigure(0, weight=3)
1069  LoginFrame.columnconfigure(1, weight=2)
1070  LoginFrame.columnconfigure(2, weight=3)
1071
1072  LoginFrame.rowconfigure(1, weight=2)
1073  LoginFrame.rowconfigure(7, weight=5)
1074
1075
1076  def login():
1077      # Validate the username and password
1078      global error_frame
1079      username = username_entry.get()
1080      password = password_entry.get()
1081      returned = check_credentials(username, password)
1082      if returned == 99:
1083          # Login successful
1084          LoginFrame.destroy()
1085
1086      else:
1087          # Login failed
1088          # Display an error message
1089          try:
1090              error_frame.grid_remove()
1091          except:
1092              pass
1093          error_frame = ctk.CTkFrame(LoginFrame)
1094          error_frame.grid(row=7, column=0,
columnspan=3,pady=20, sticky="n")
1095          if returned == 2:
1096              error_message = ctk.CTkLabel(
1097                  error_frame, text="Error! Username not found",
font=font_1
1098              )
1099          else:
1100              error_message = ctk.CTkLabel(
1101                  error_frame, text="Error! Password invalid",
font=font_1
1102              )
1103          error_message.grid(row=0, column=0, sticky="nsew",
pady=7, padx=15)
```

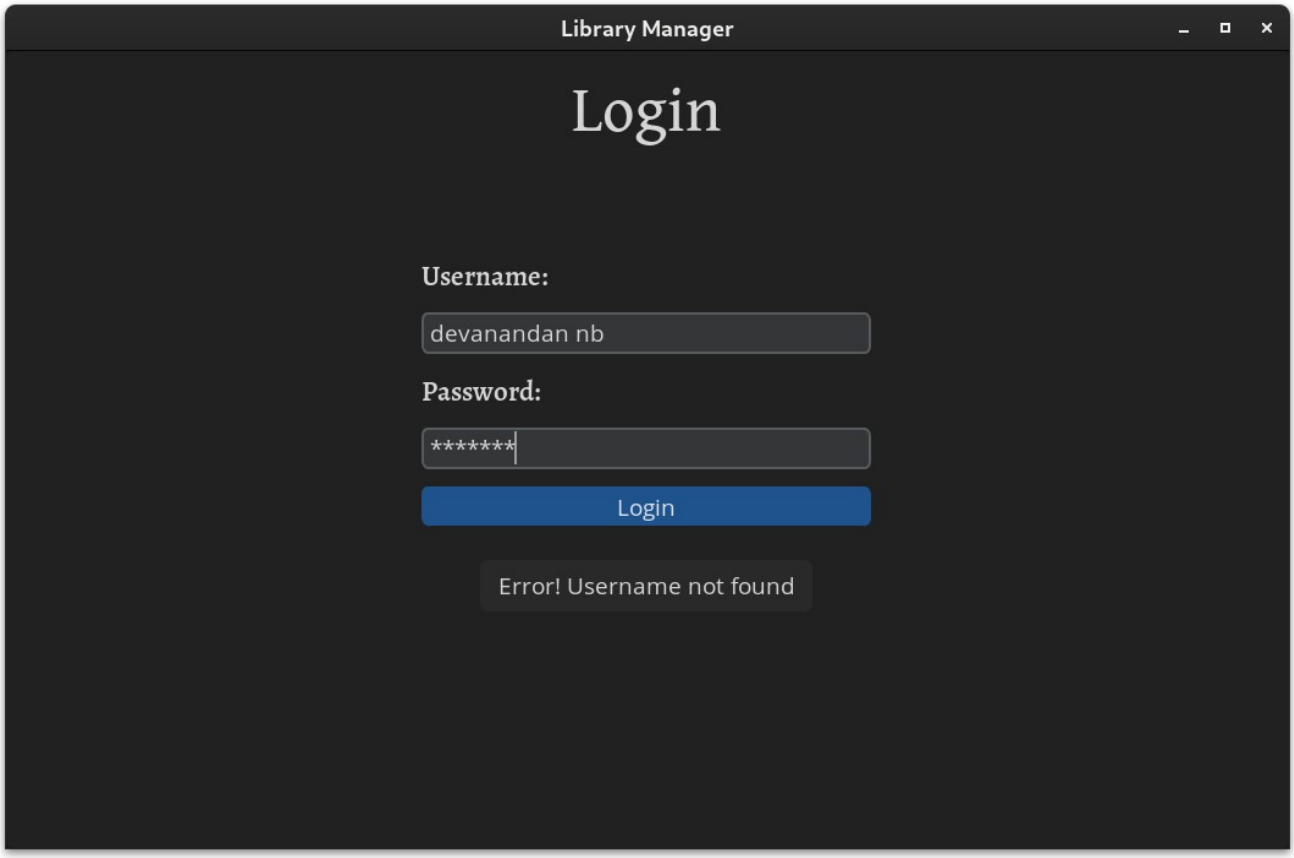
```
1104
1105
1106  # Create the login form
1107  title_label = ctk.CTkLabel(LoginFrame, text="Login",
font=font_3)
1108  username_label = ctk.CTkLabel(LoginFrame, text="Username:",
font=font_2)
1109  username_entry = ctk.CTkEntry(LoginFrame,
placeholder_text="username", font=font_1)
1110  password_label = ctk.CTkLabel(LoginFrame, text="Password:",
font=font_2)
1111  password_entry = ctk.CTkEntry(
1112      LoginFrame, placeholder_text="password", show="*",
font=font_1
1113  )
1114  login_button = ctk.CTkButton(LoginFrame, text="Login",
command=login, font=font_1)
1115
1116  # Place the widgets on the frame
1117  title_label.grid(row=0, column=1, sticky="nsew", pady=7)
1118  username_label.grid(row=2, column=1, sticky="nsw", pady=7)
1119  username_entry.grid(row=3, column=1, sticky="nsew", pady=7)
1120  password_label.grid(row=4, column=1, sticky="nsw", pady=7)
1121  password_entry.grid(row=5, column=1, sticky="nsew", pady=7)
1122  login_button.grid(row=6, column=1, sticky="nsew", pady=7)
1123
1124  # run
1125  root.mainloop()
```

Output Screens

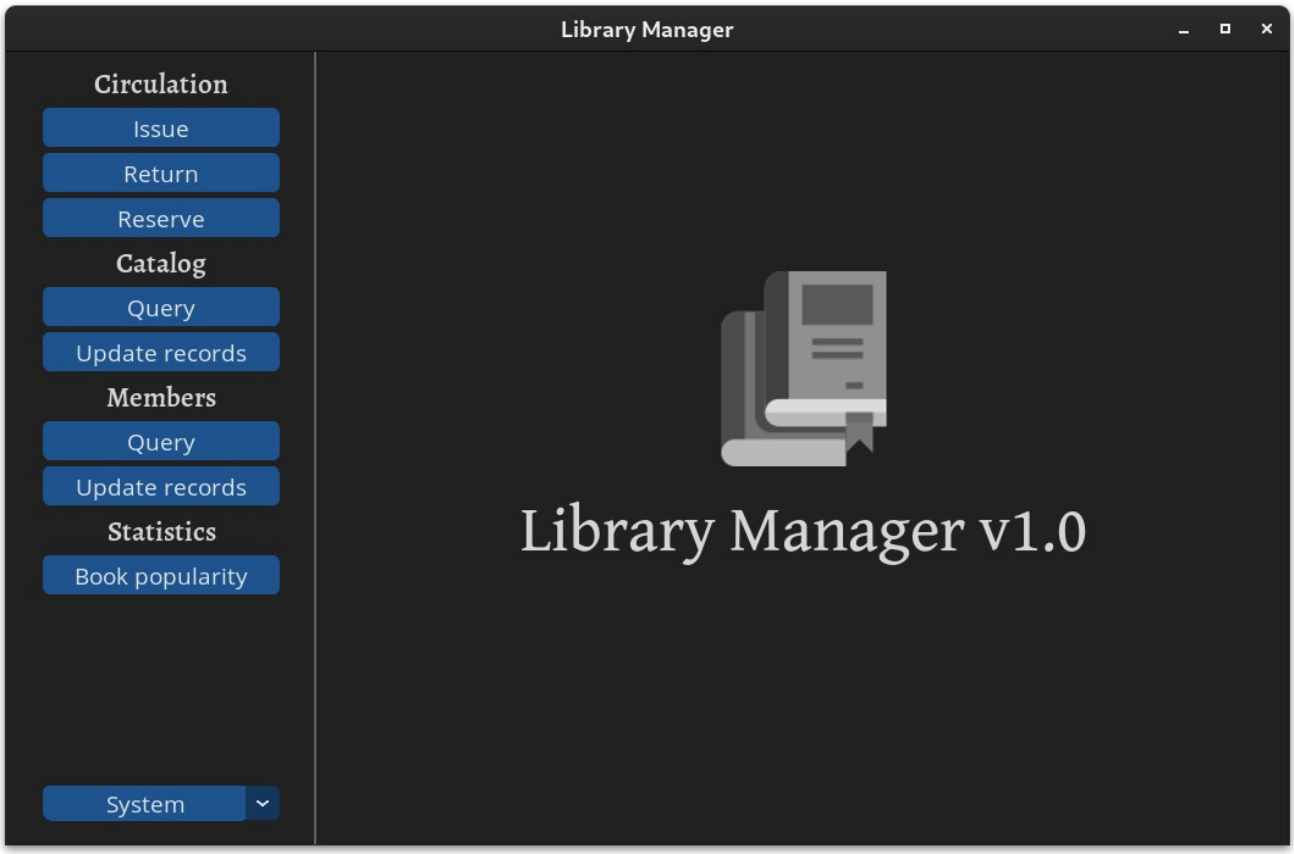
- Login Screen with username and password entries(now showing error on invalid password)



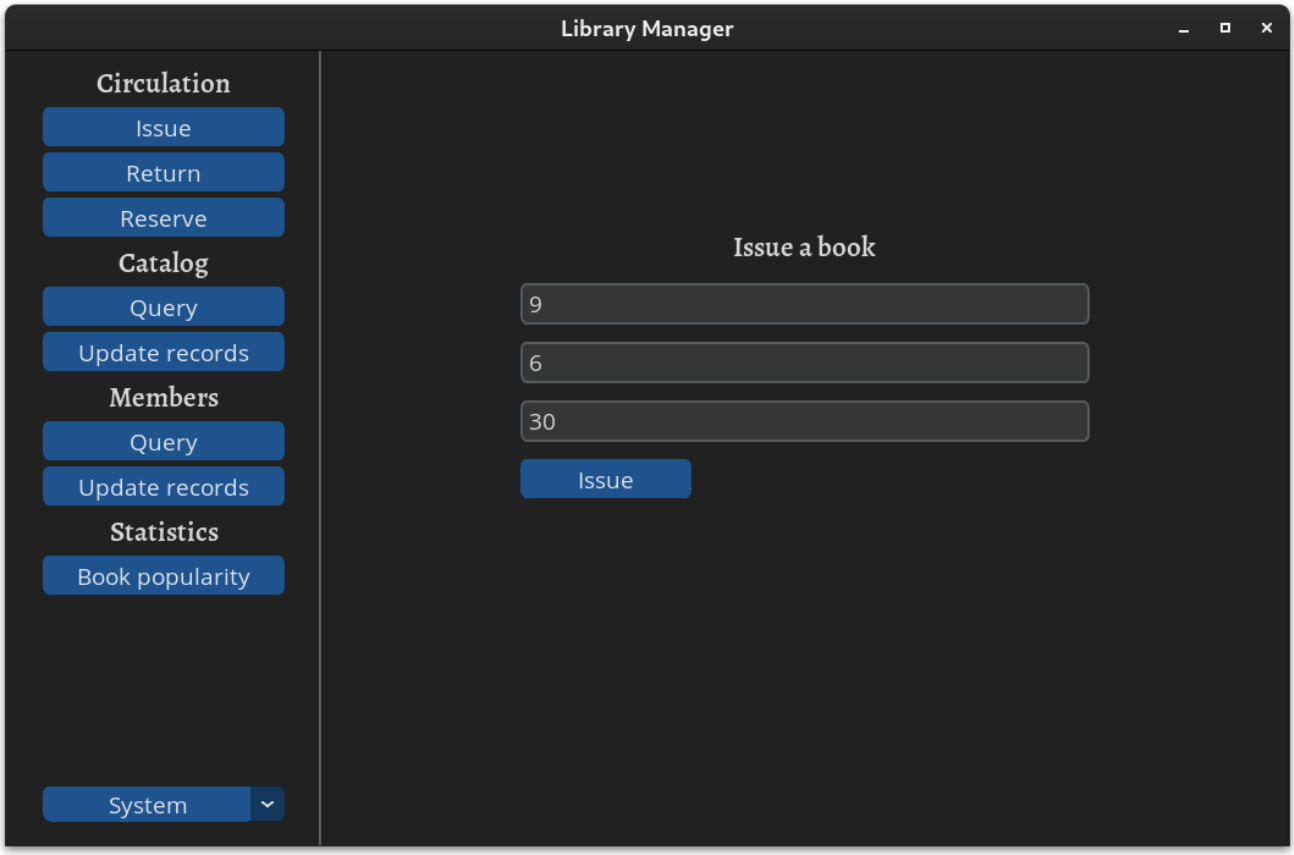
- Error message on invalid credentials (username)



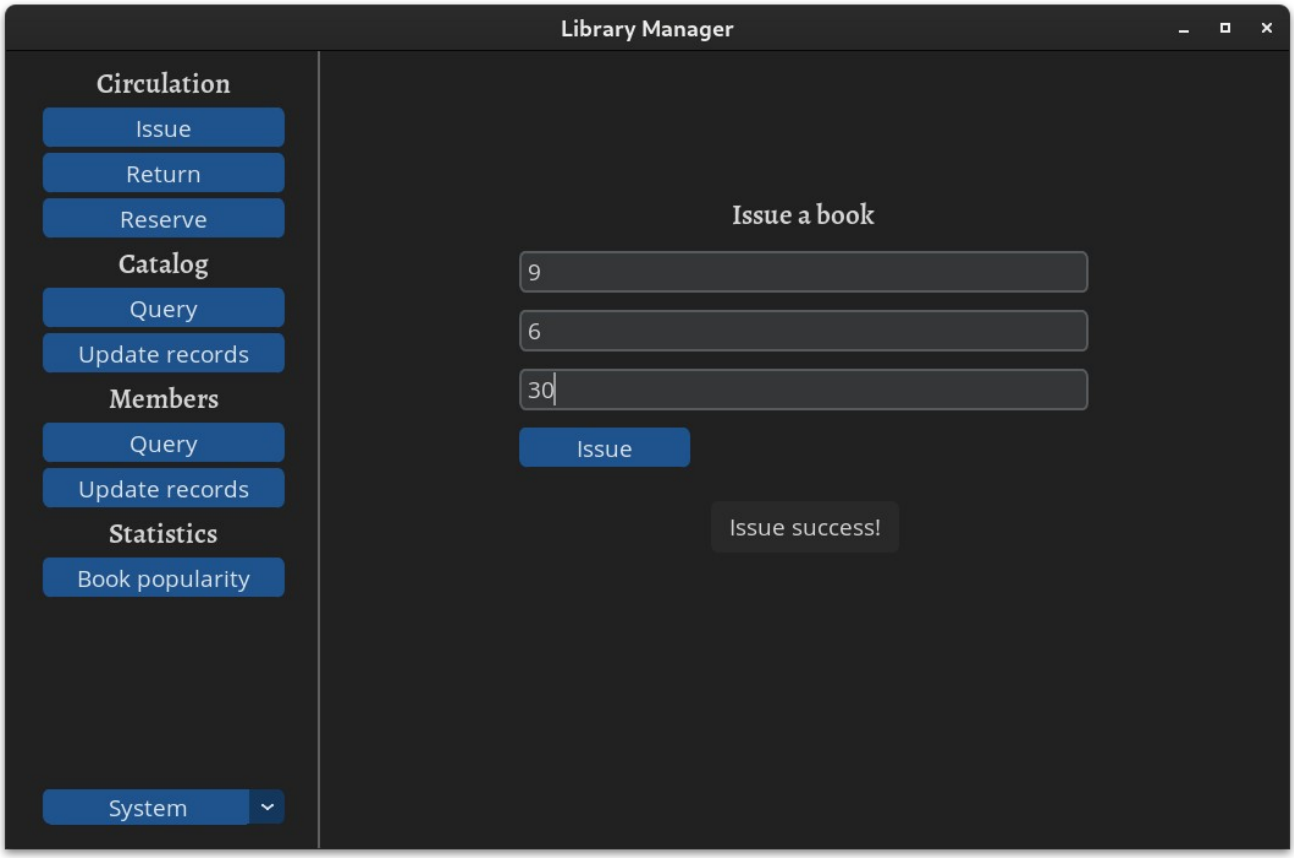
- Greet Screen of the app



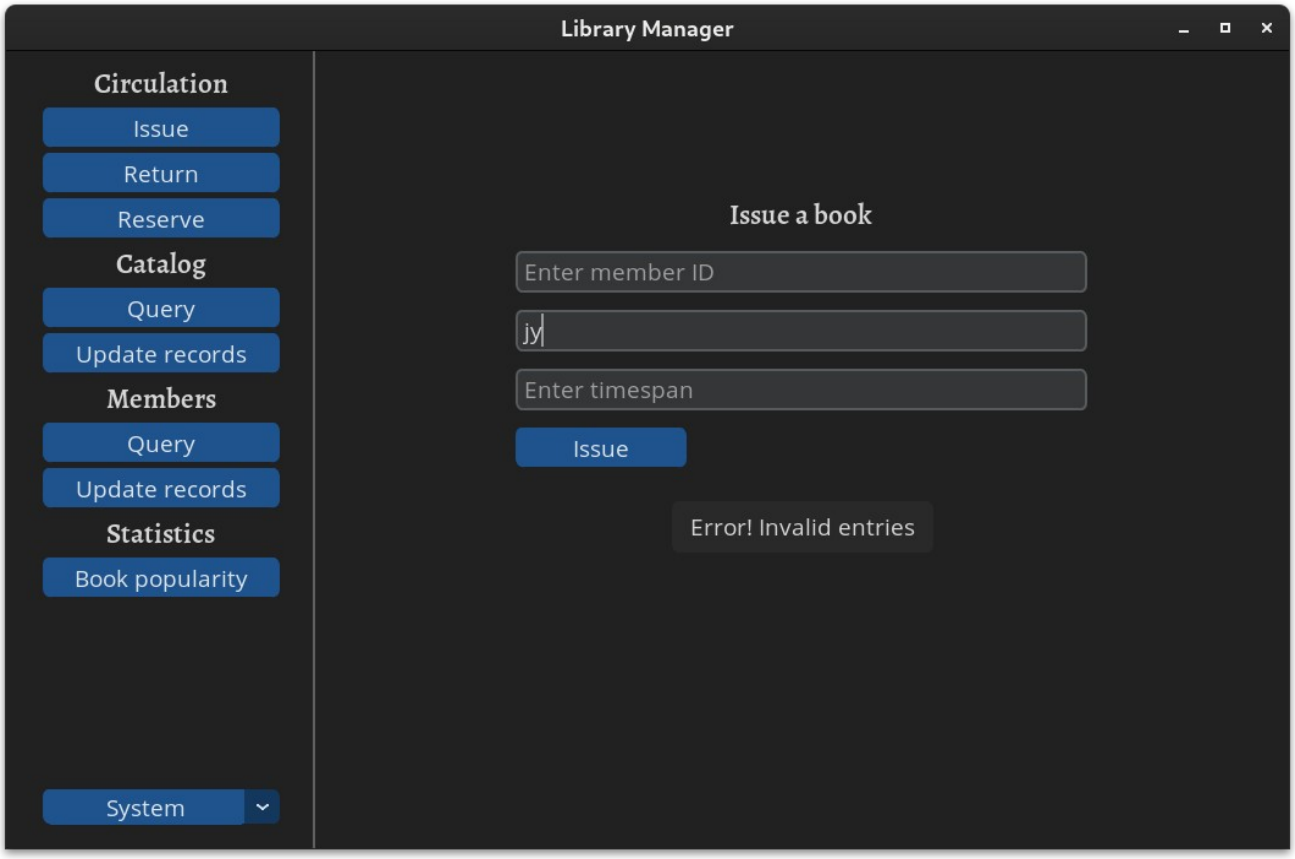
- Issueing a book



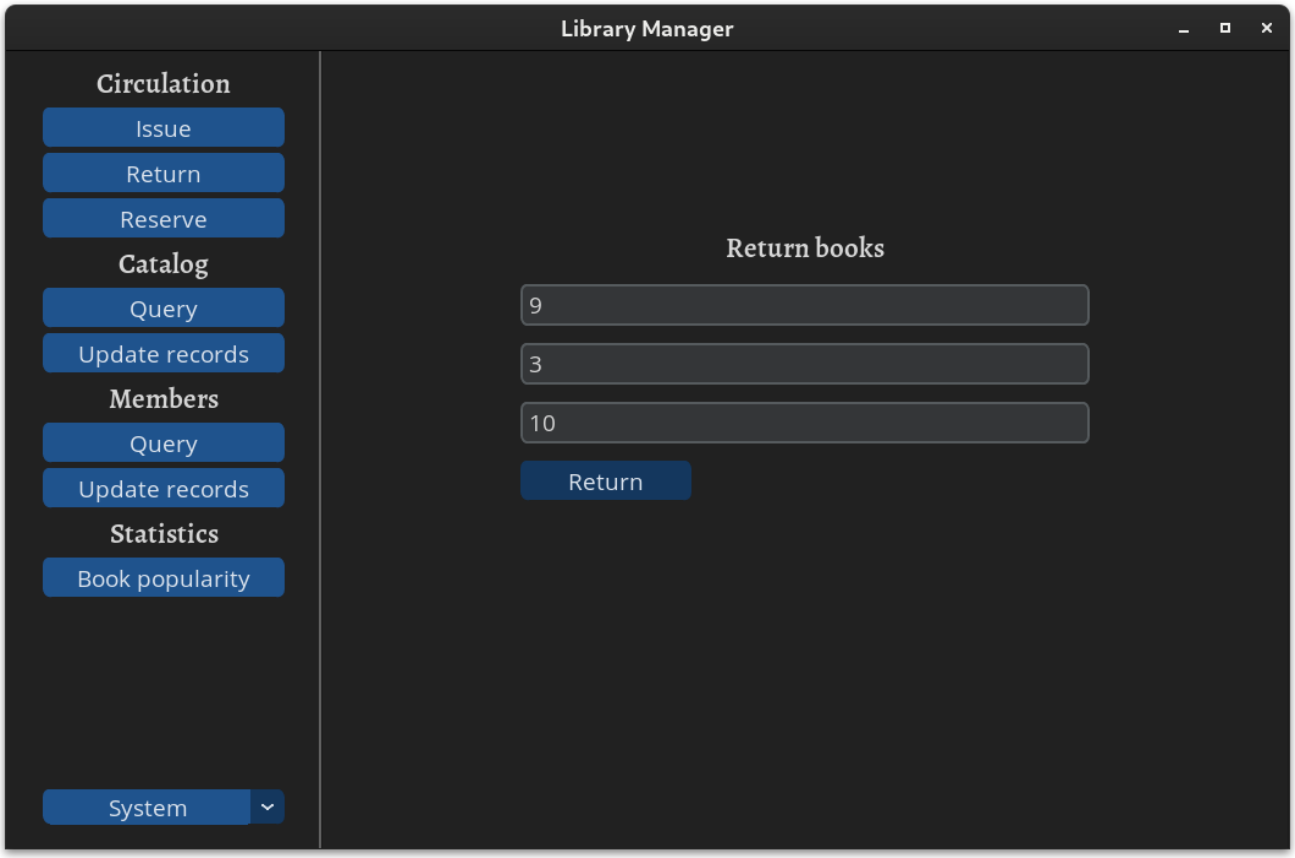
- Issue success



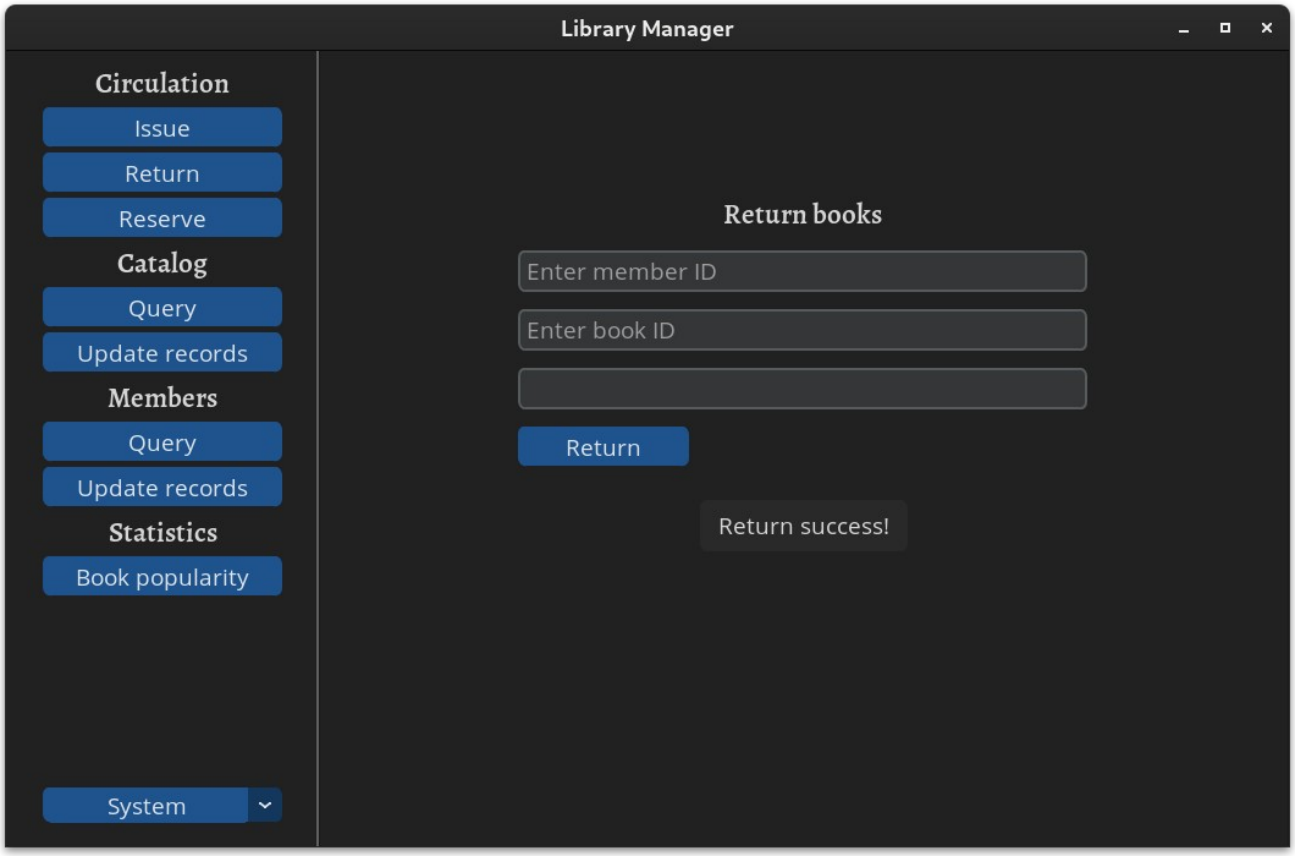
- Error on invalid entries when issuing a book



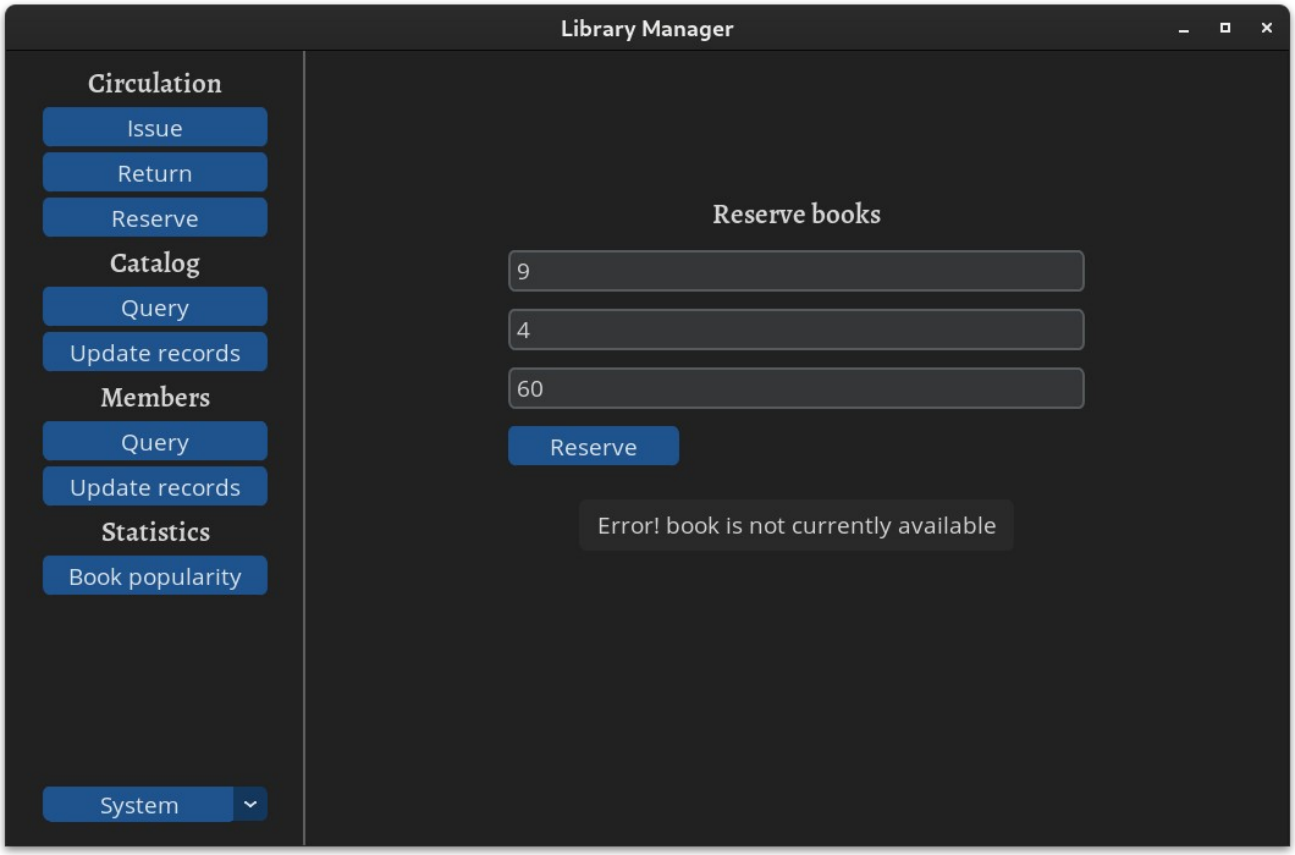
- Entering values to return a book



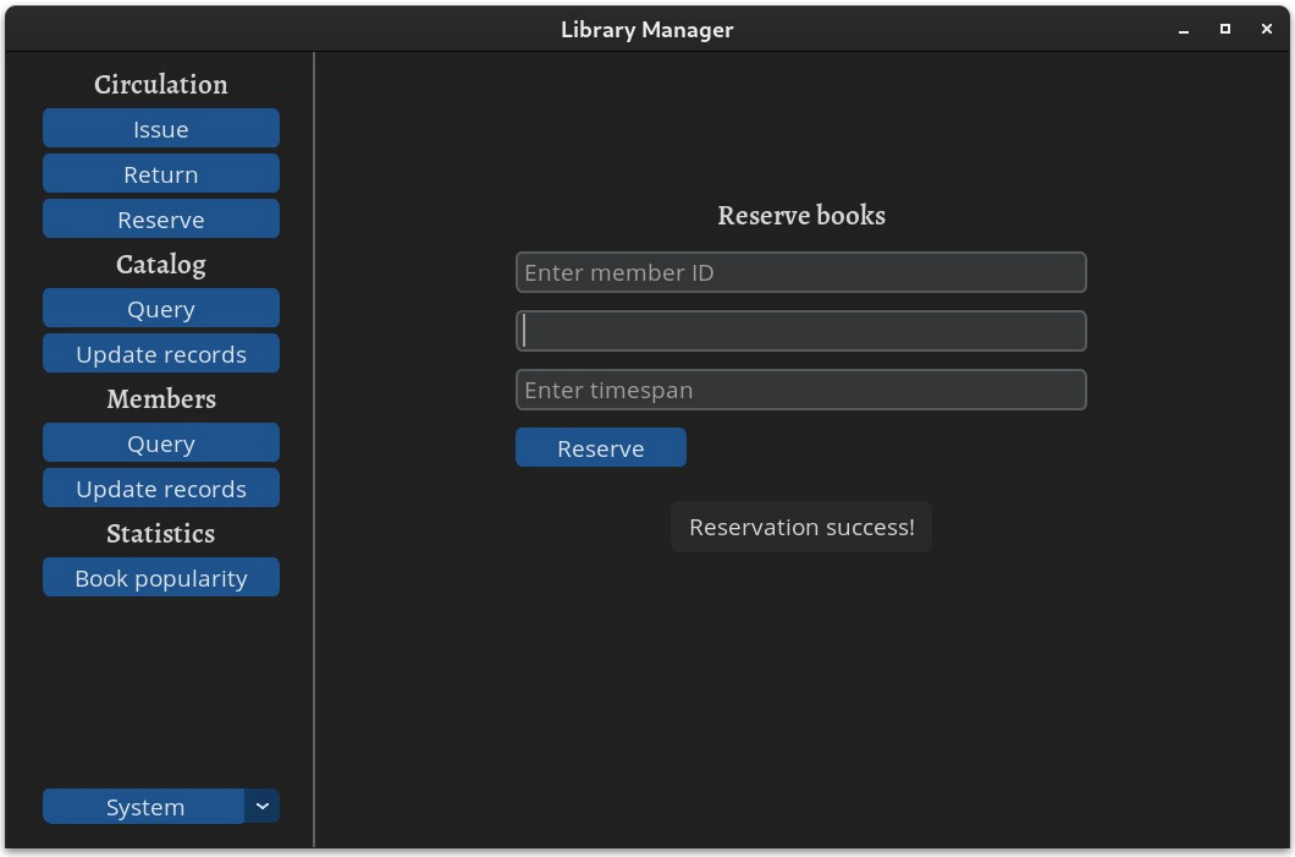
- Return success



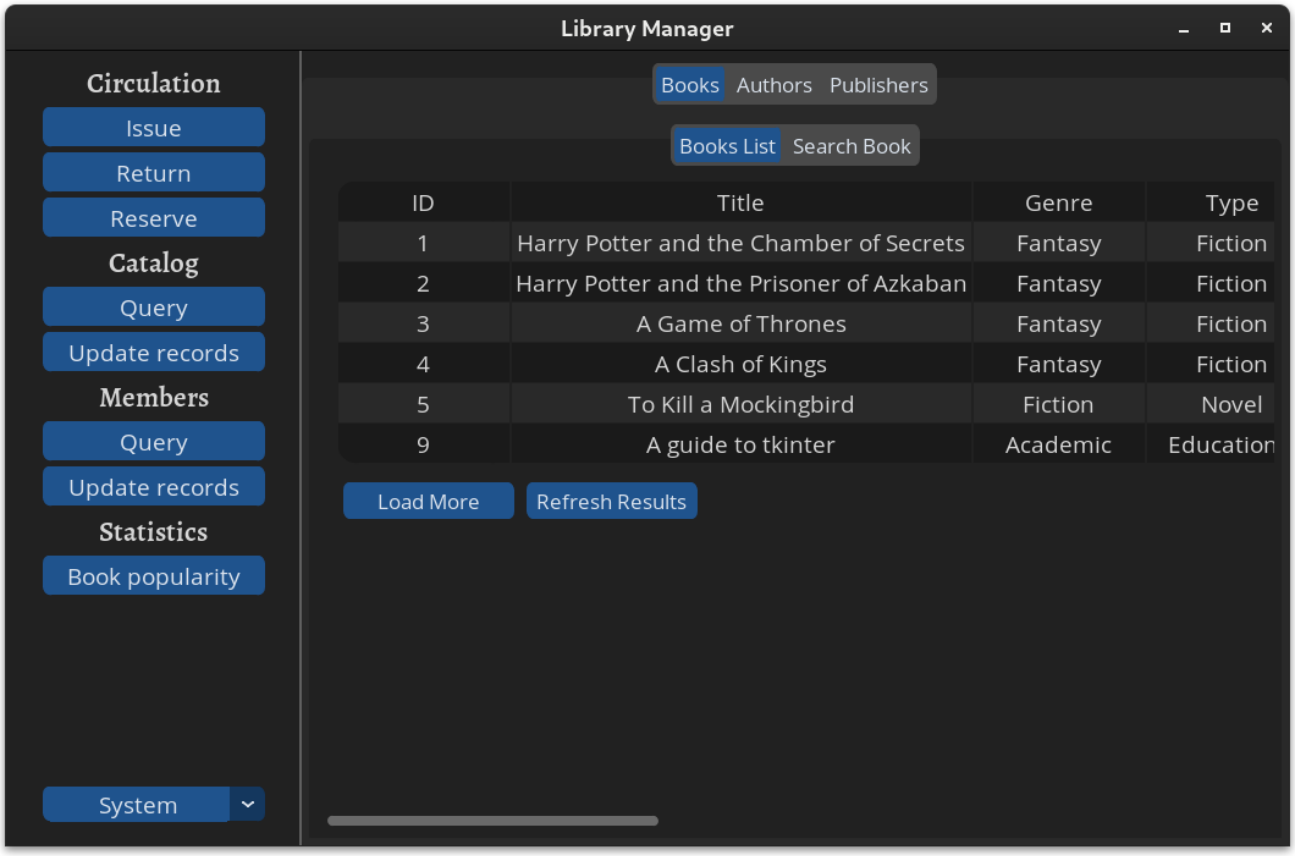
- Reserving a book (error because book supply is limited / not available)



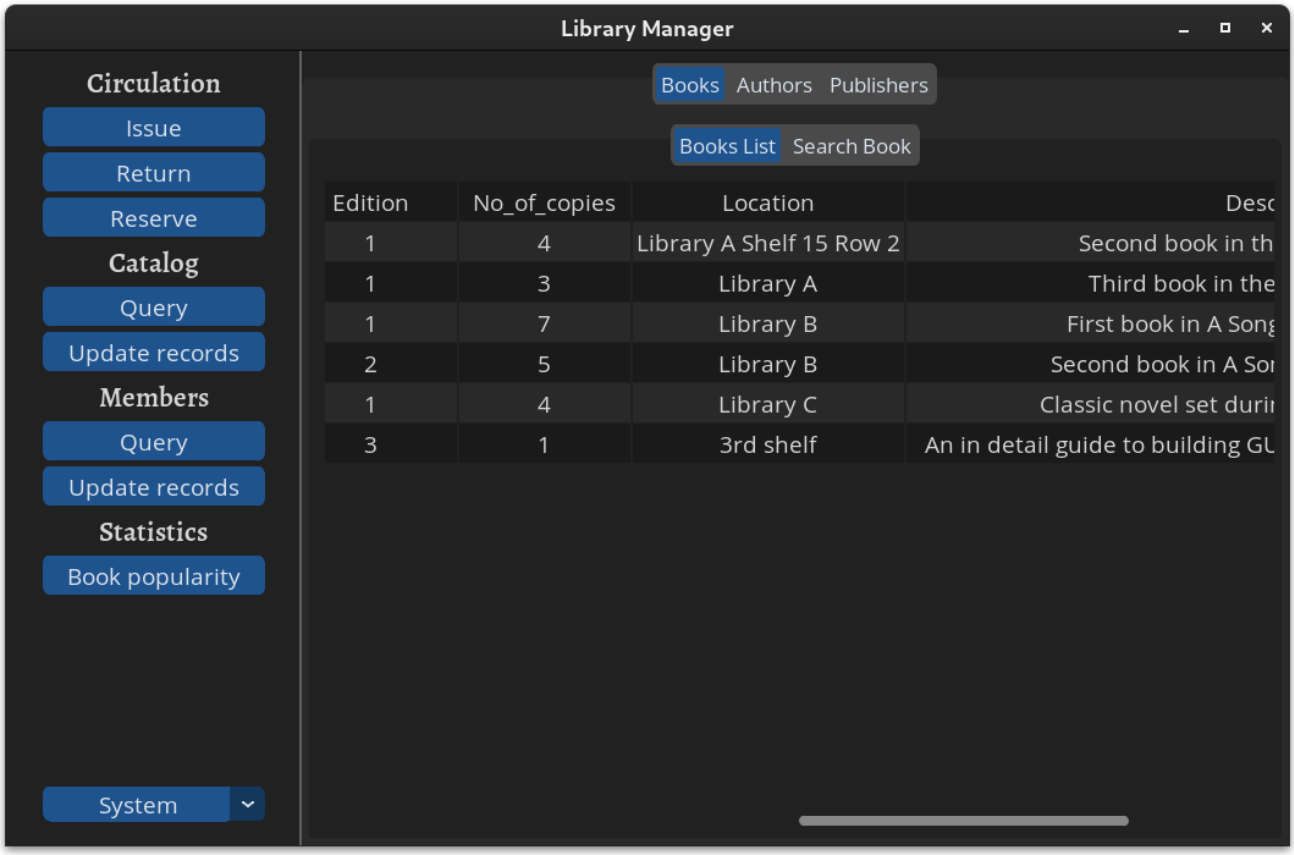
- Book reservation success



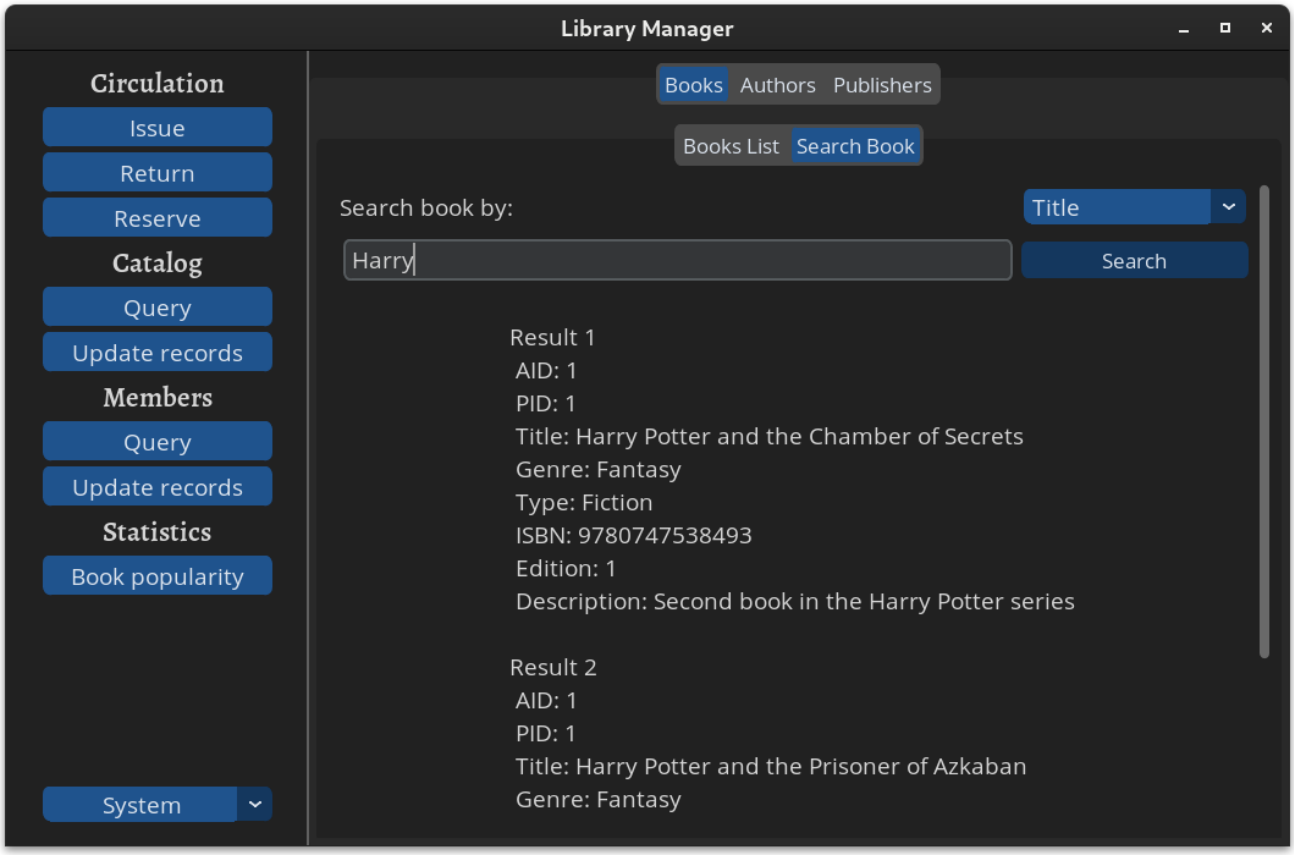
- Books list



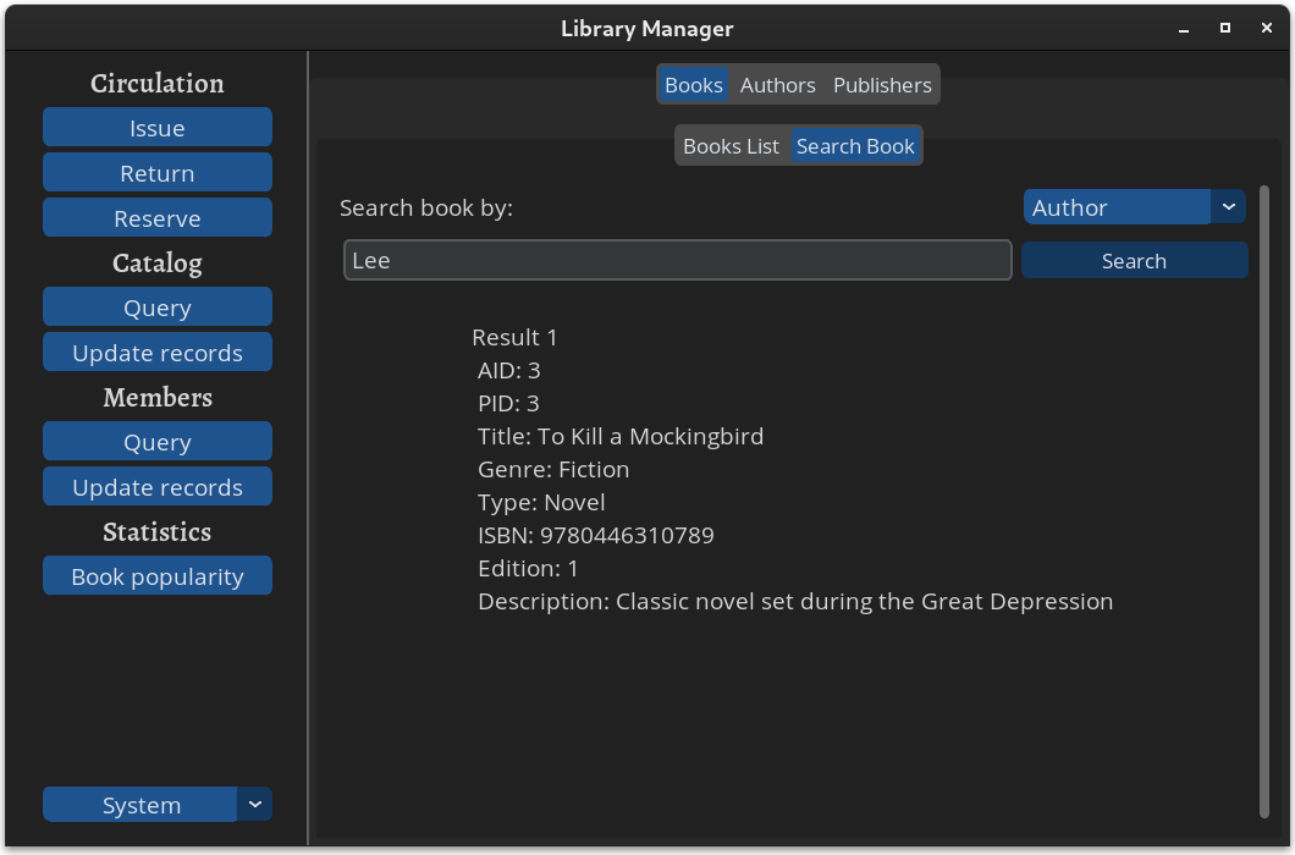
- Scrolling through books list page



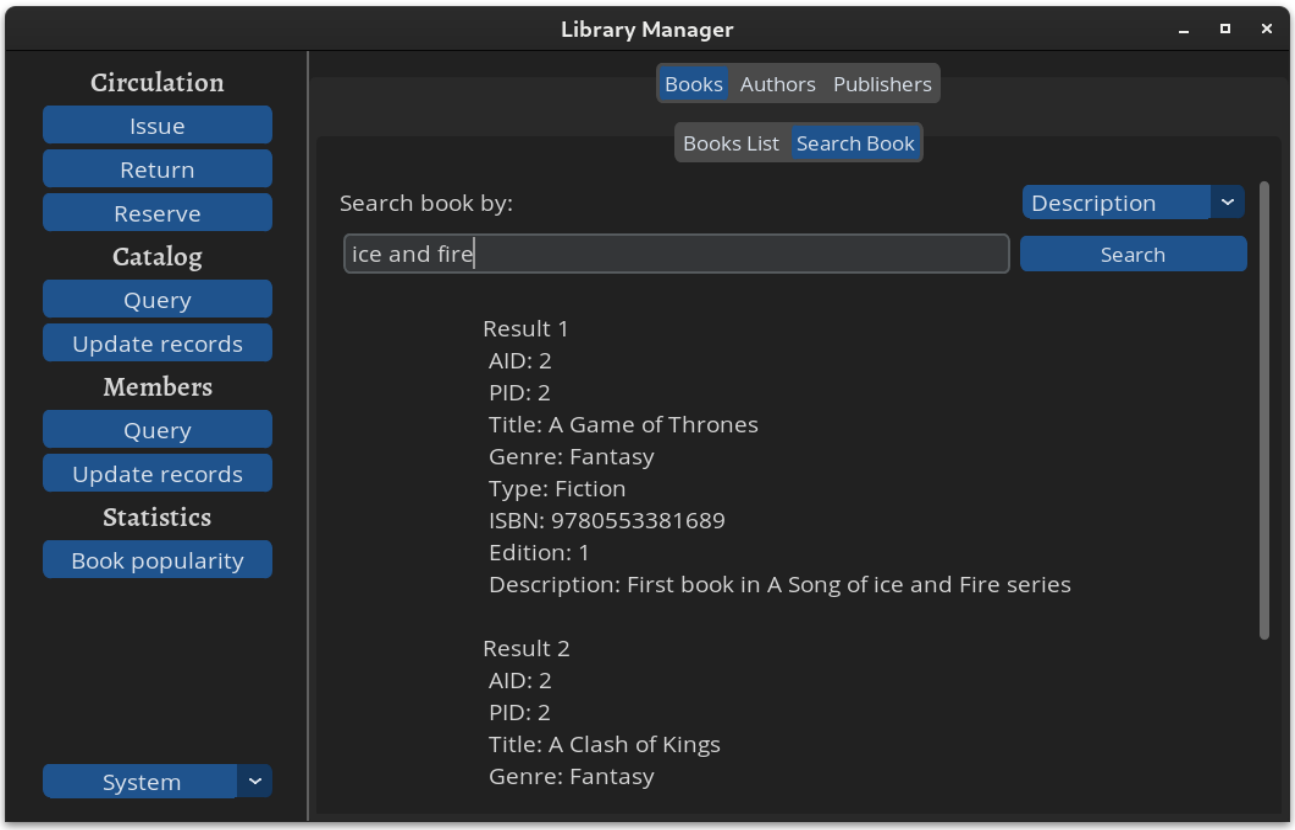
- Searching a book by it's title



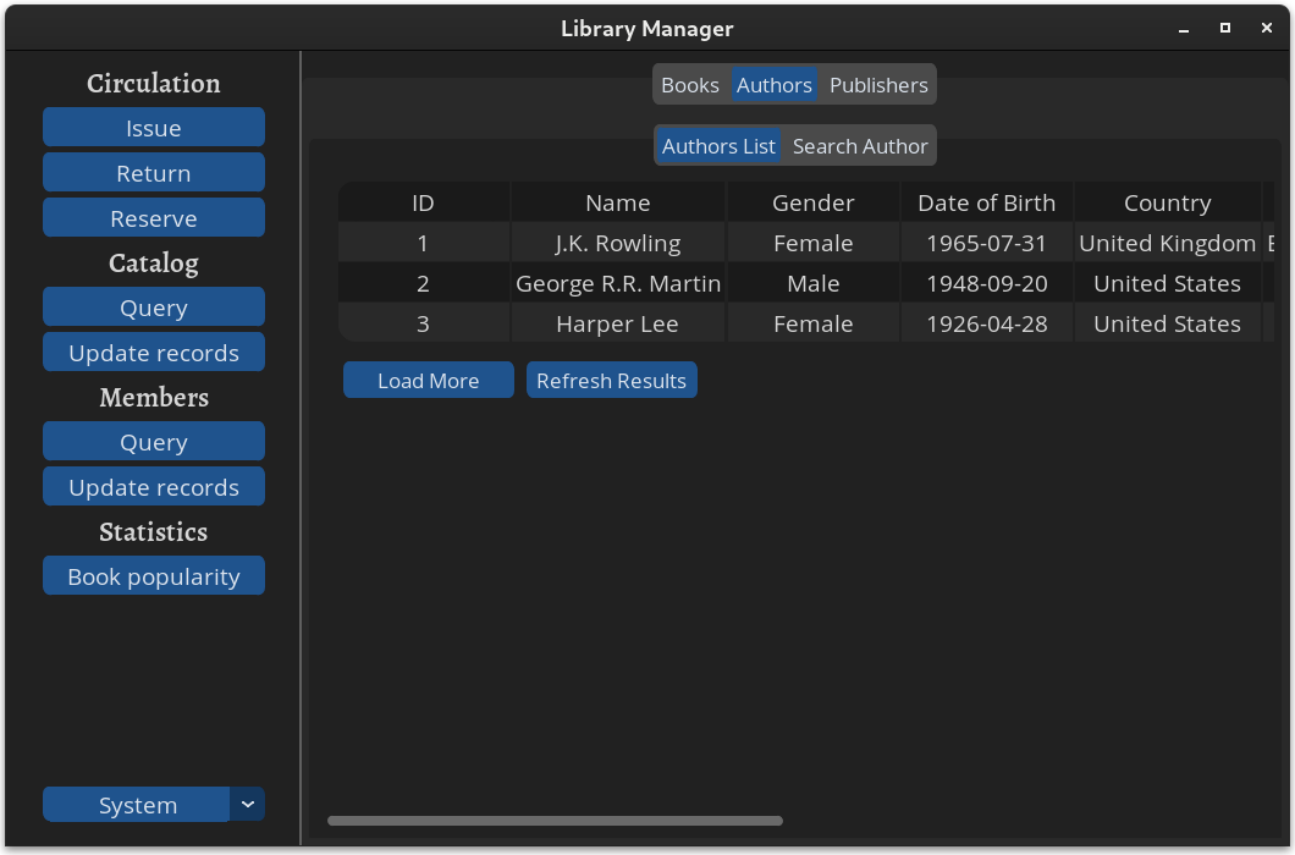
- Searching a book by it's author



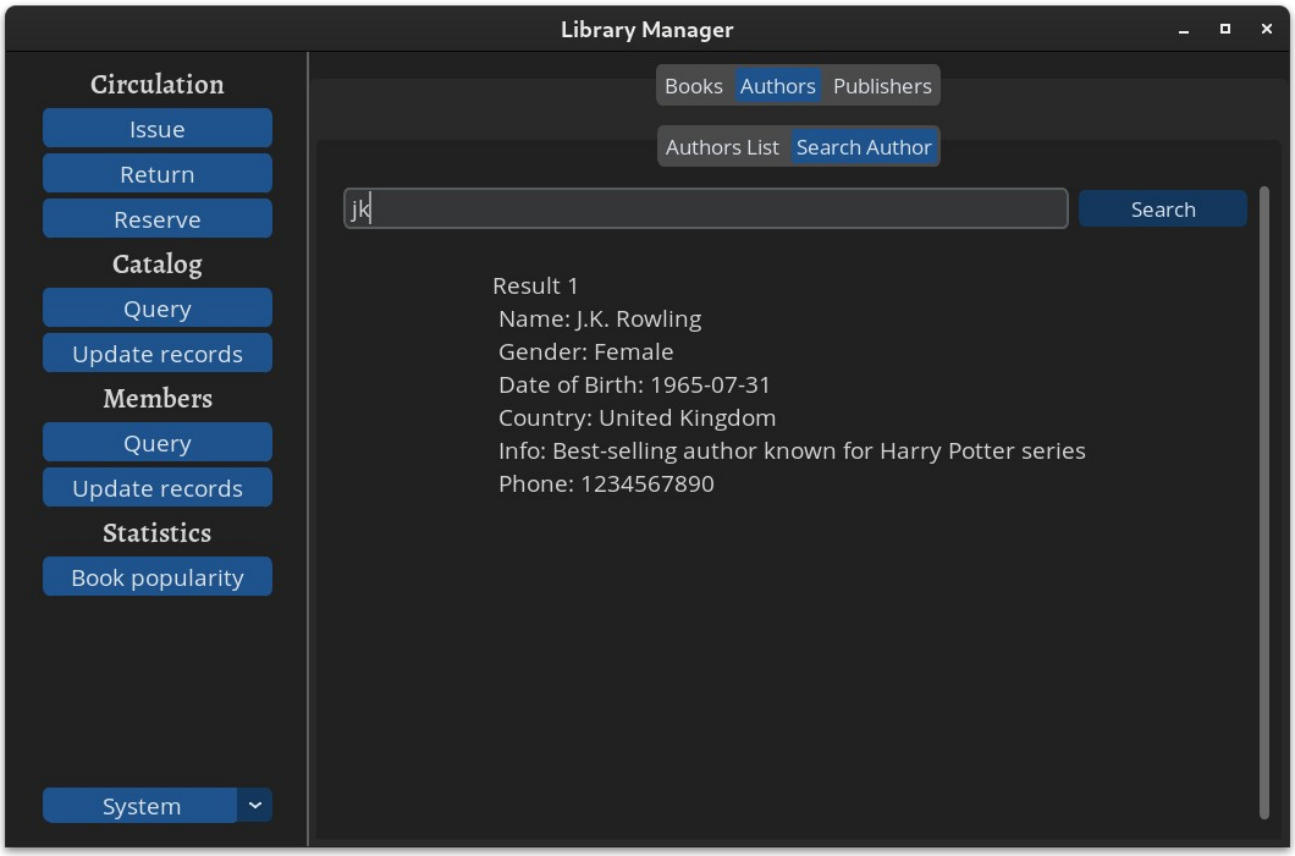
- Searching a book by it's description



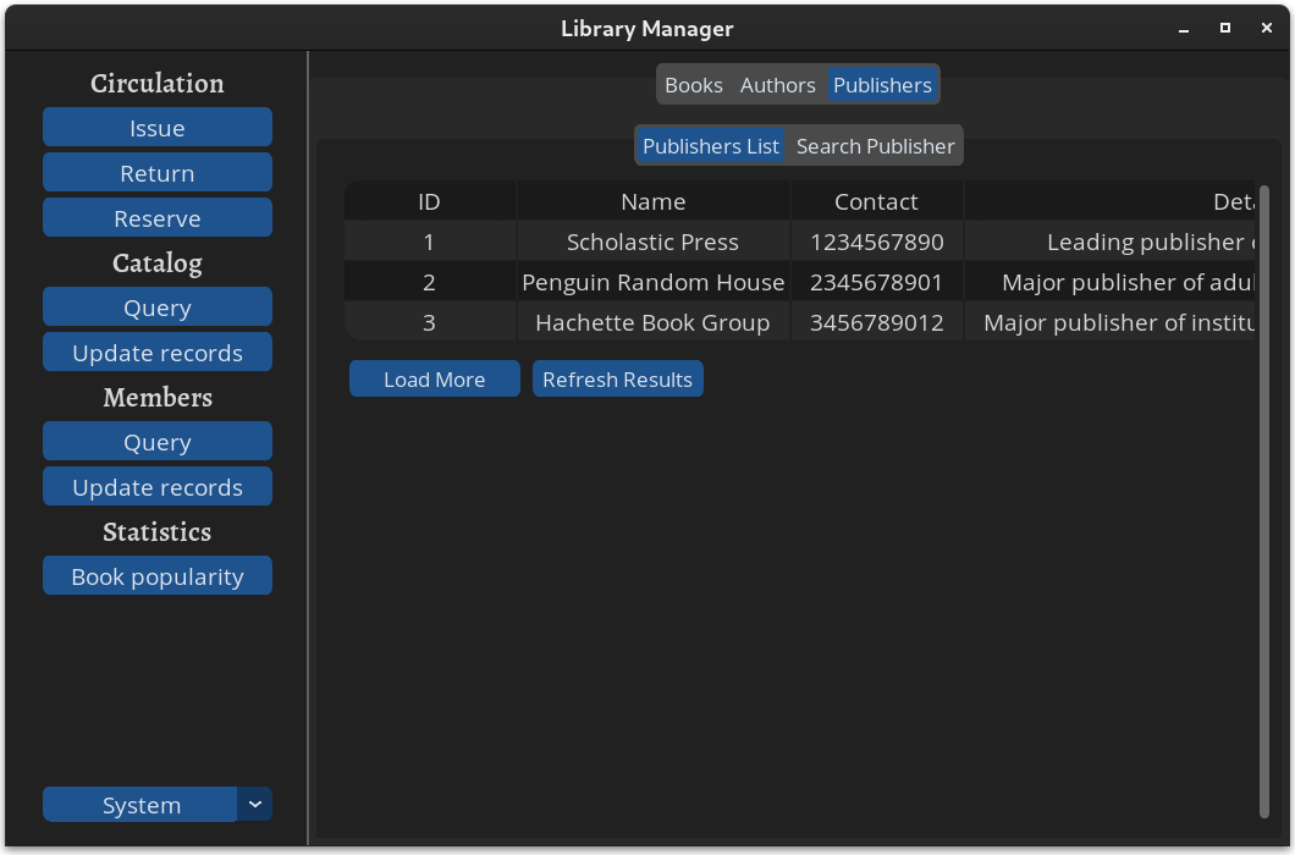
- Authors list



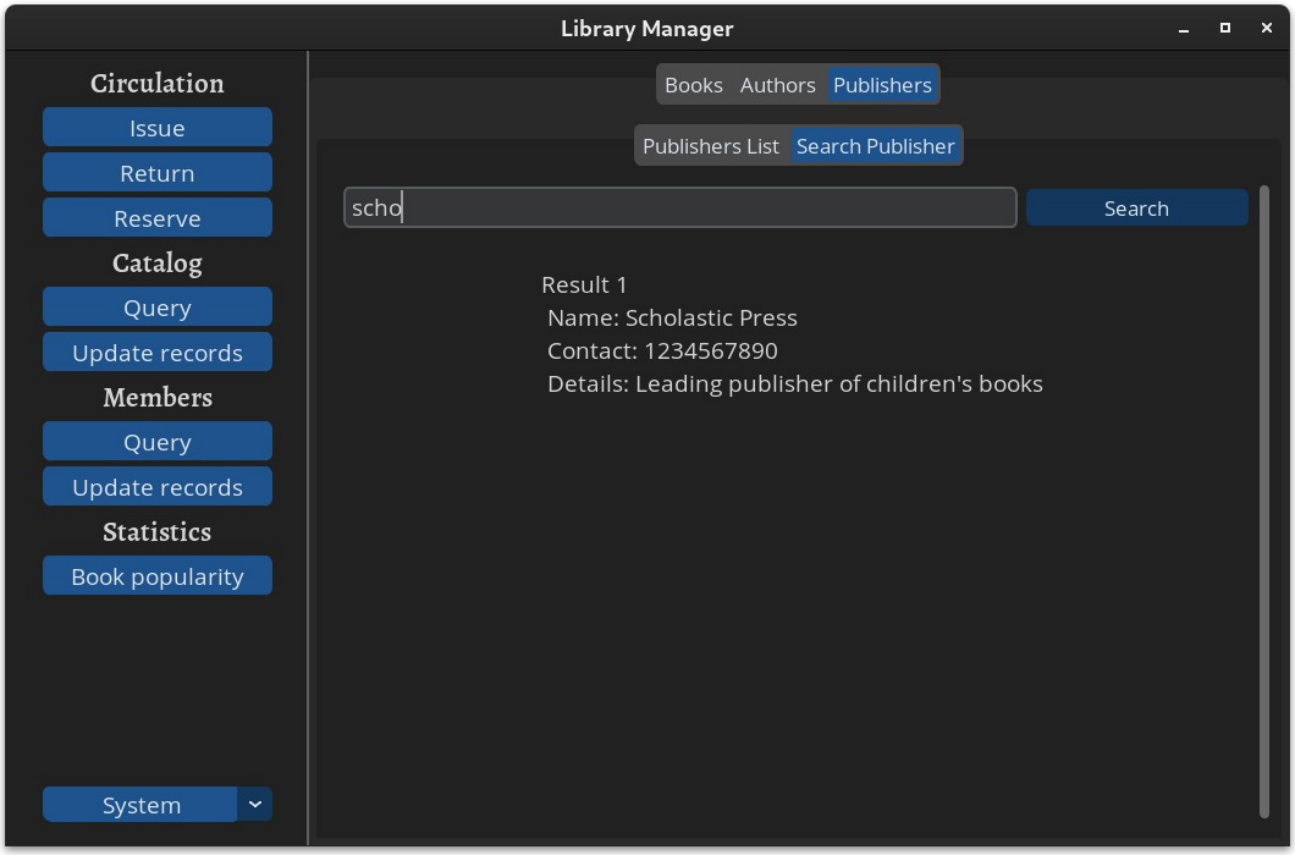
- Searching author by name



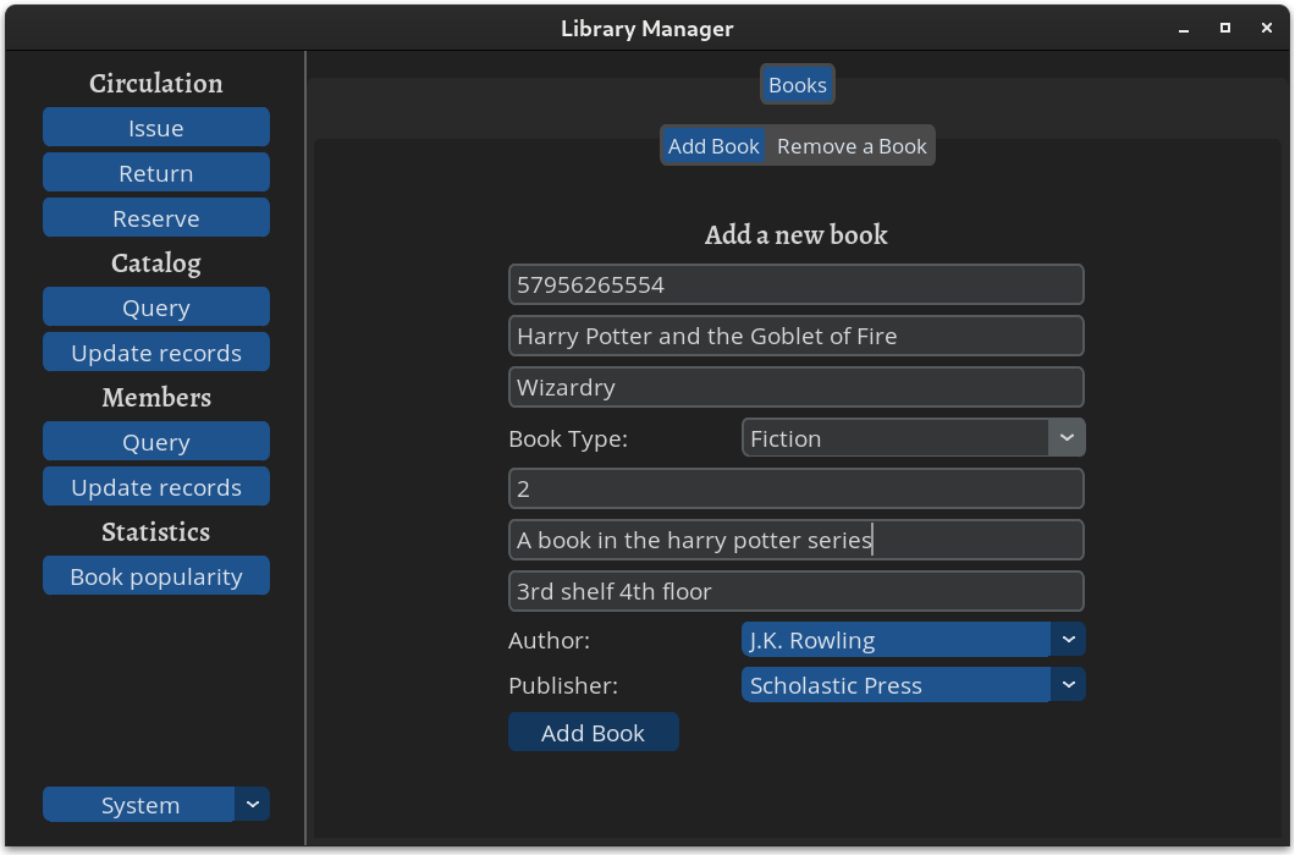
- Publishers list



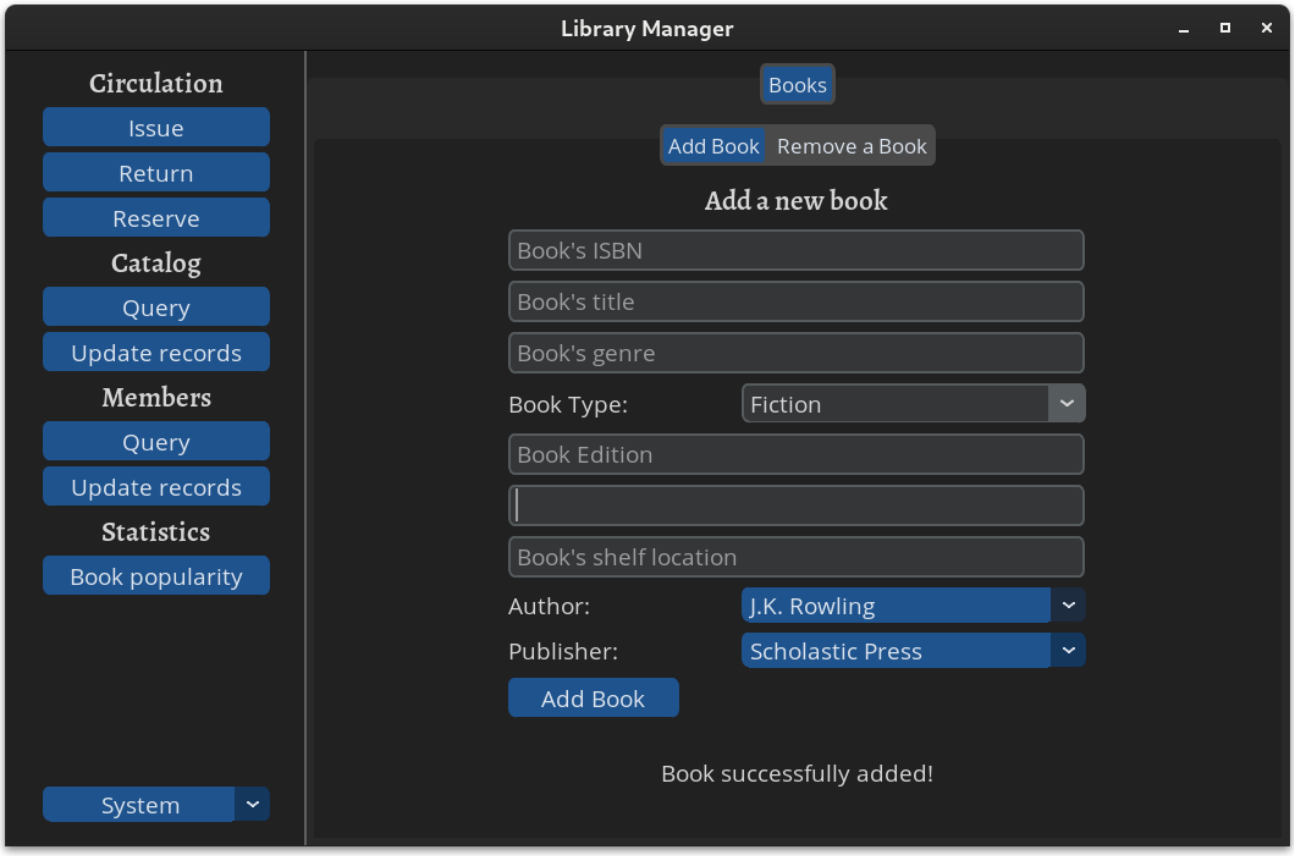
- Searching publisher by name



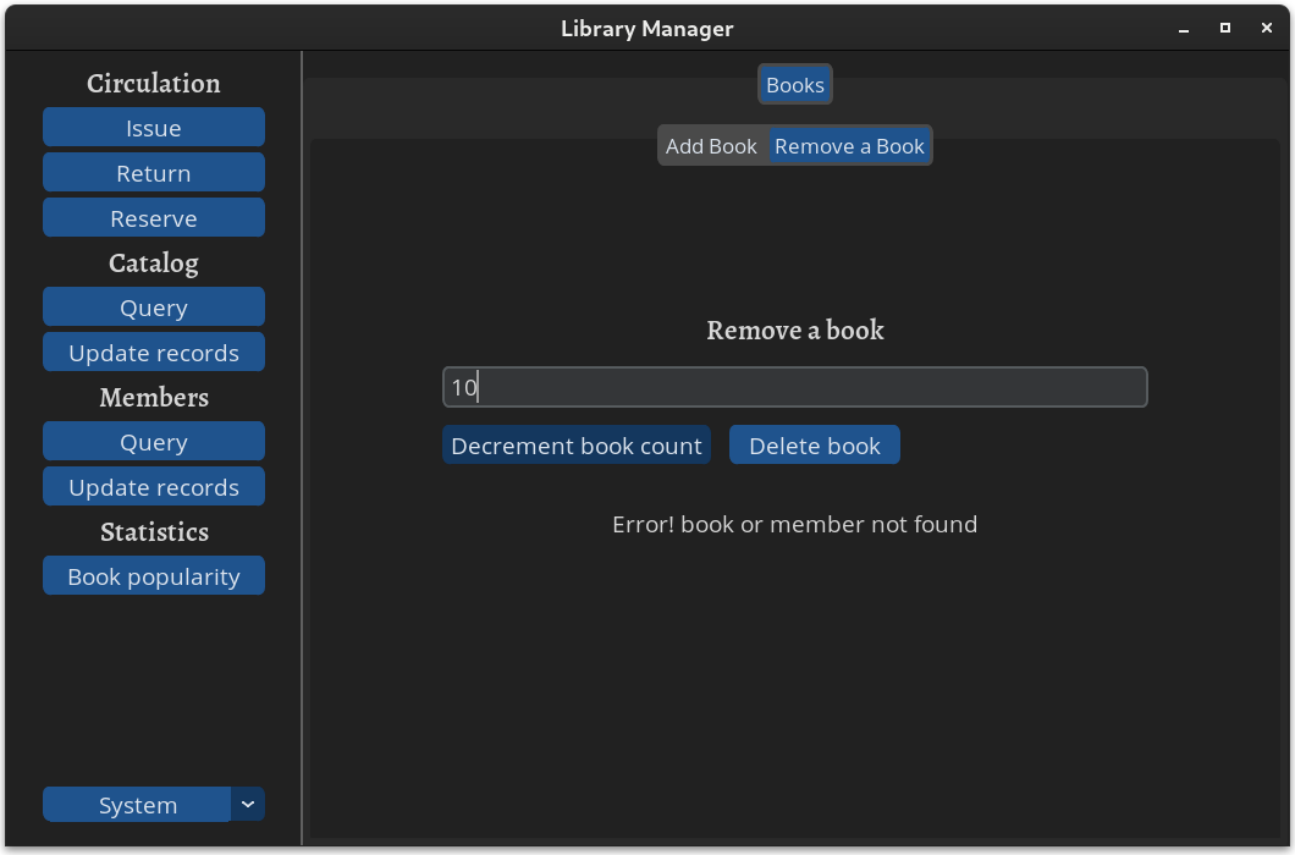
- Adding a new book via multiple entries



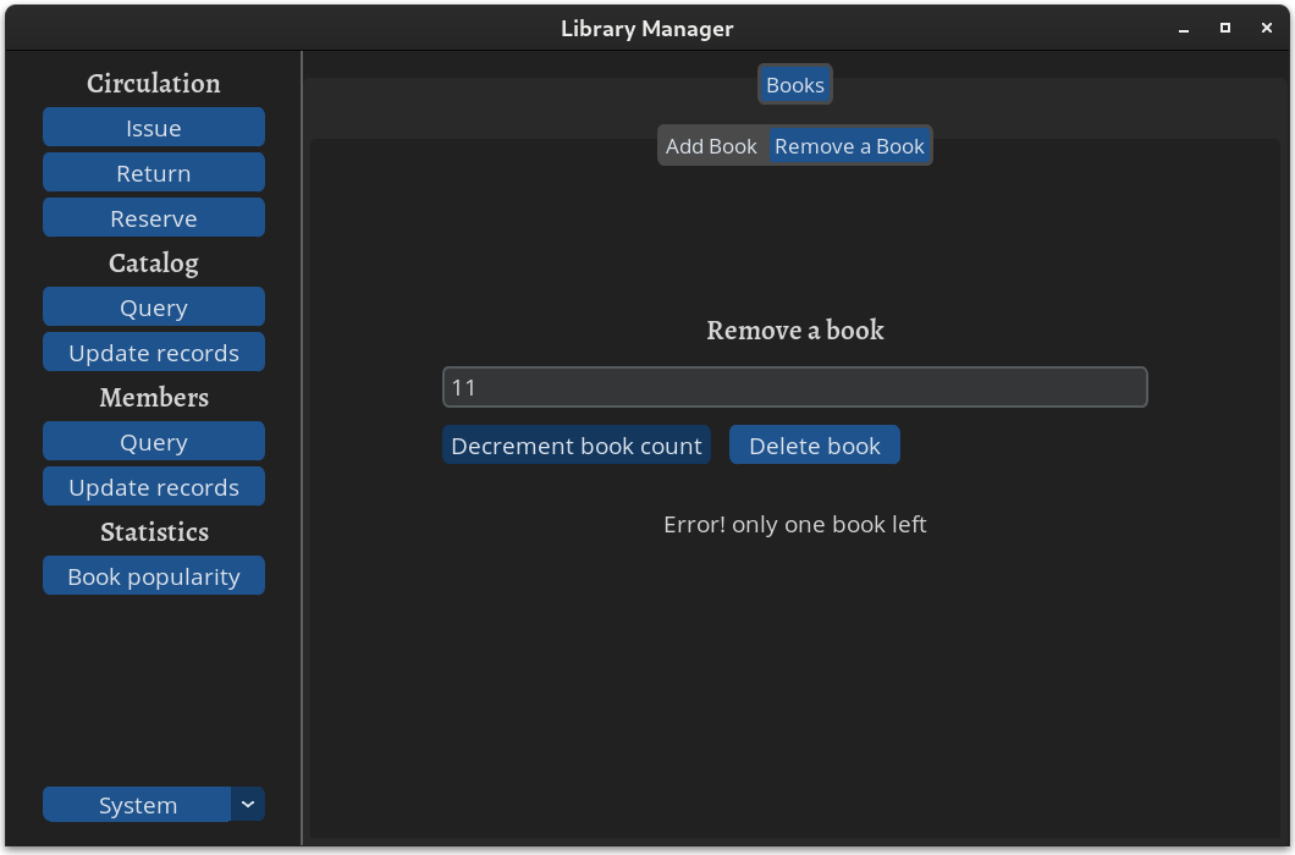
- Books successfully added



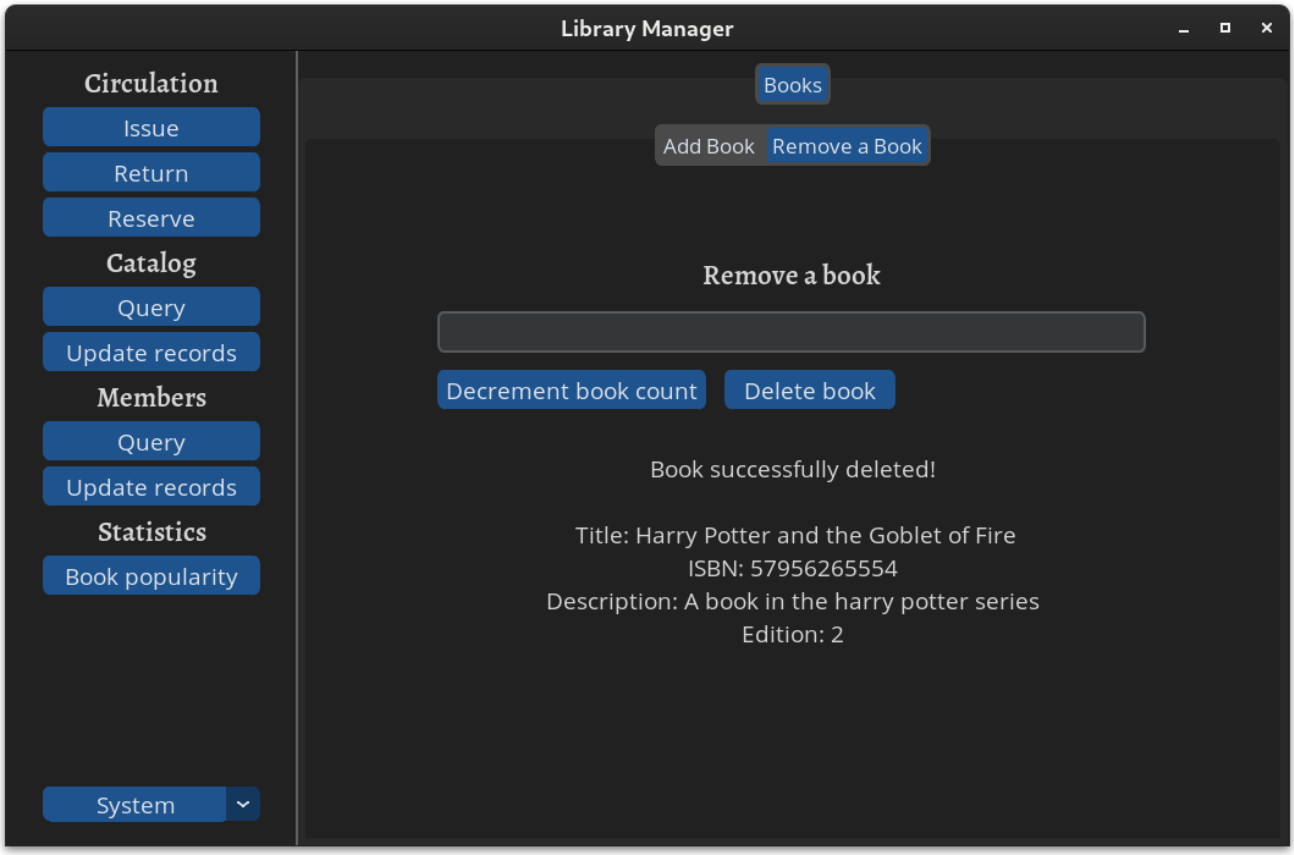
- Removing a book function returning an error



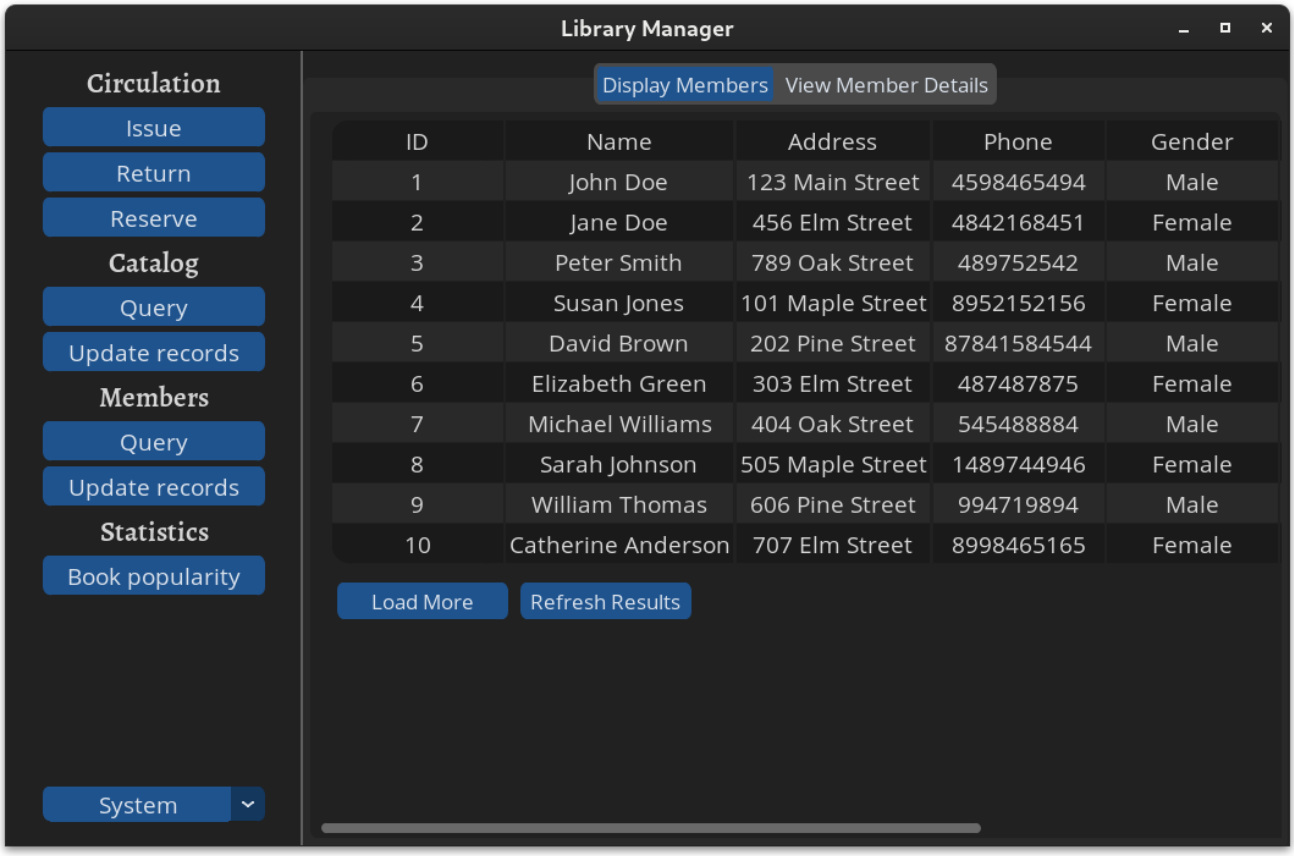
- Error on decrementing book count – only one book left



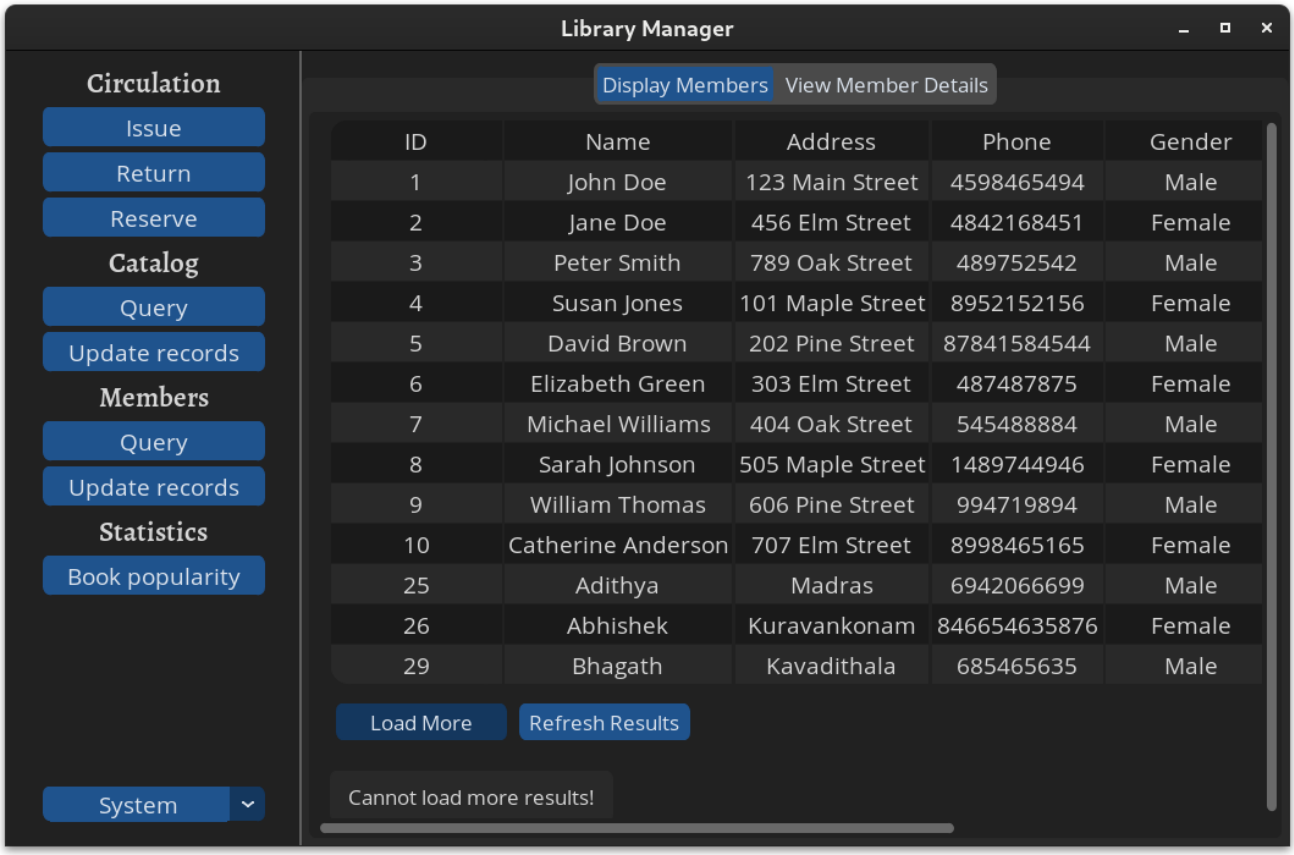
- Books successfully removed and it’s information displayed



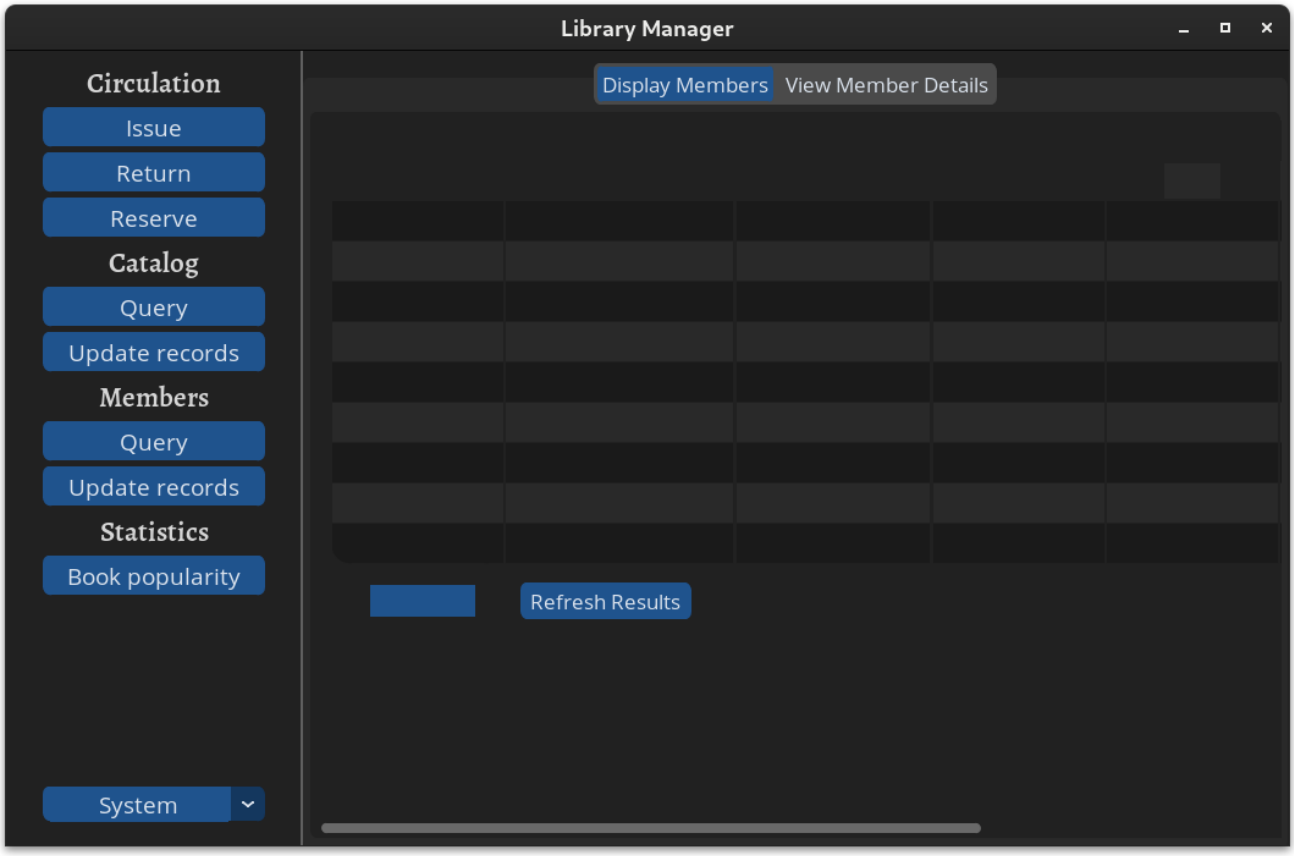
- Members list



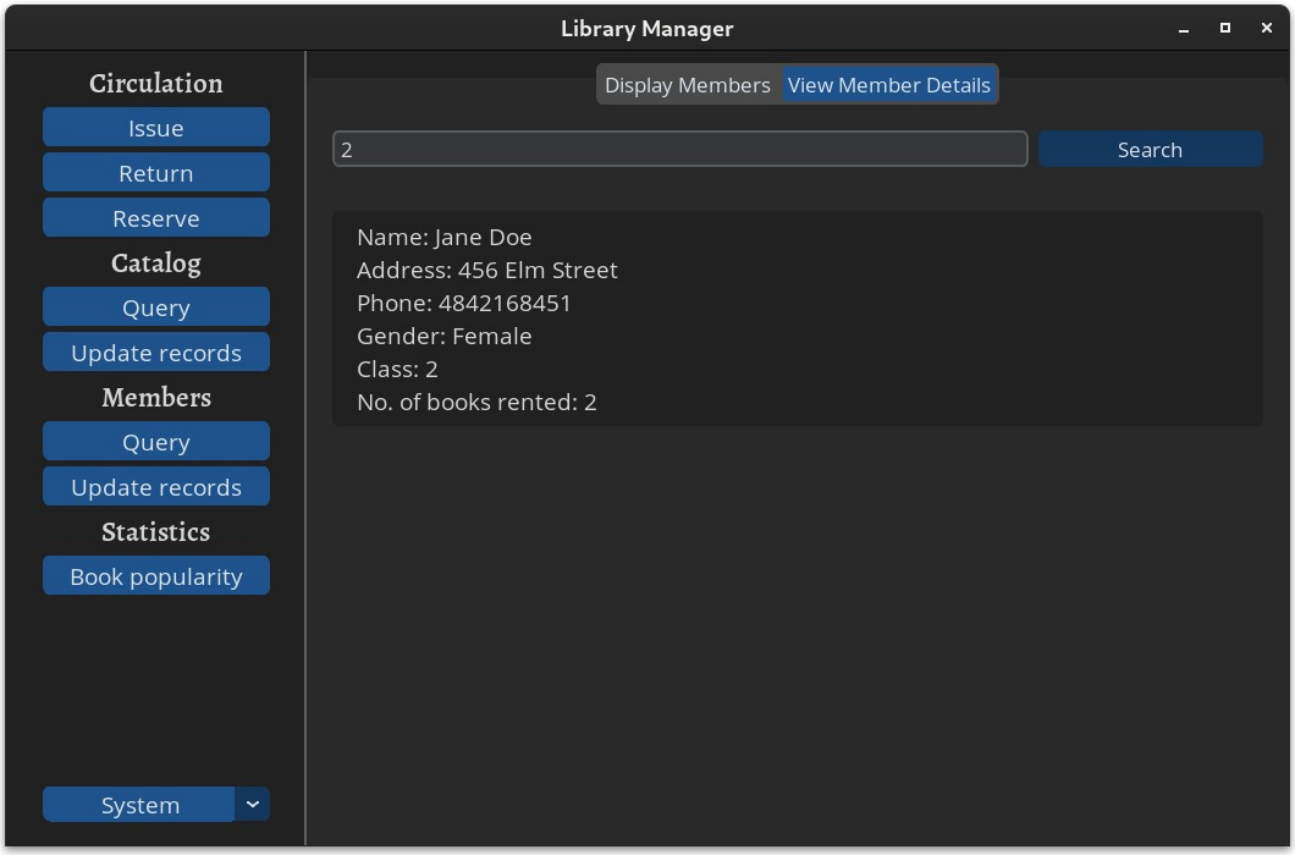
- Error while clicking load more button – maximum records reached



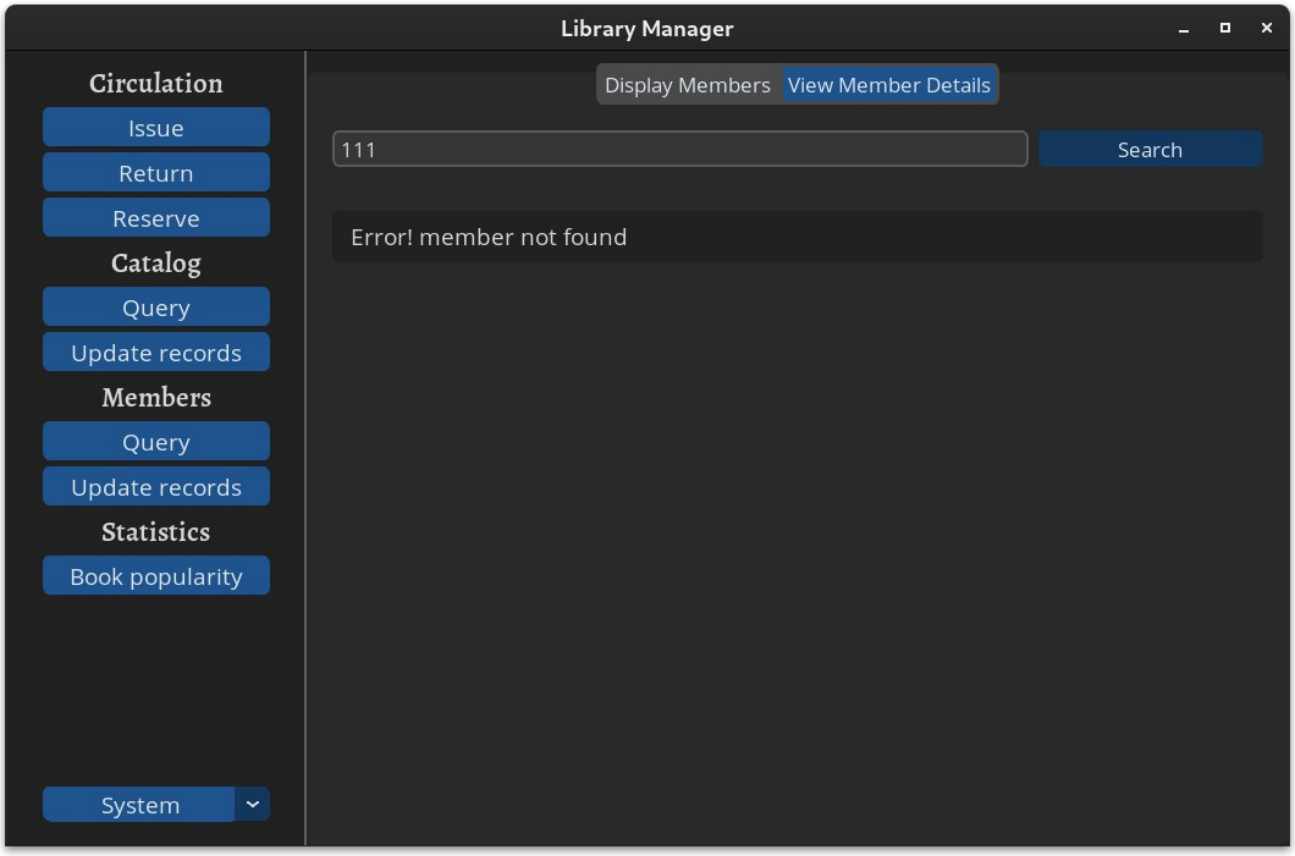
- Members list being refreshed



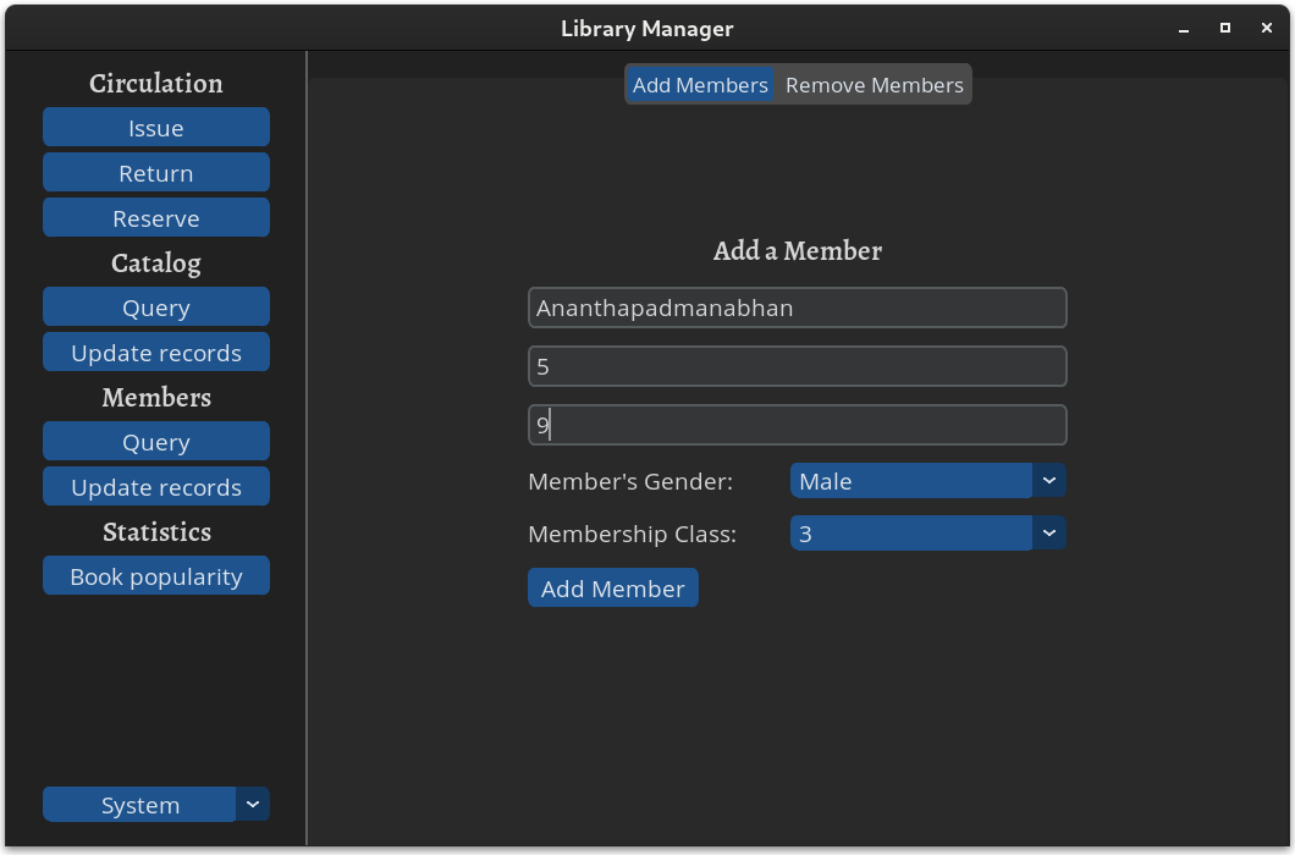
- Member details being displayed



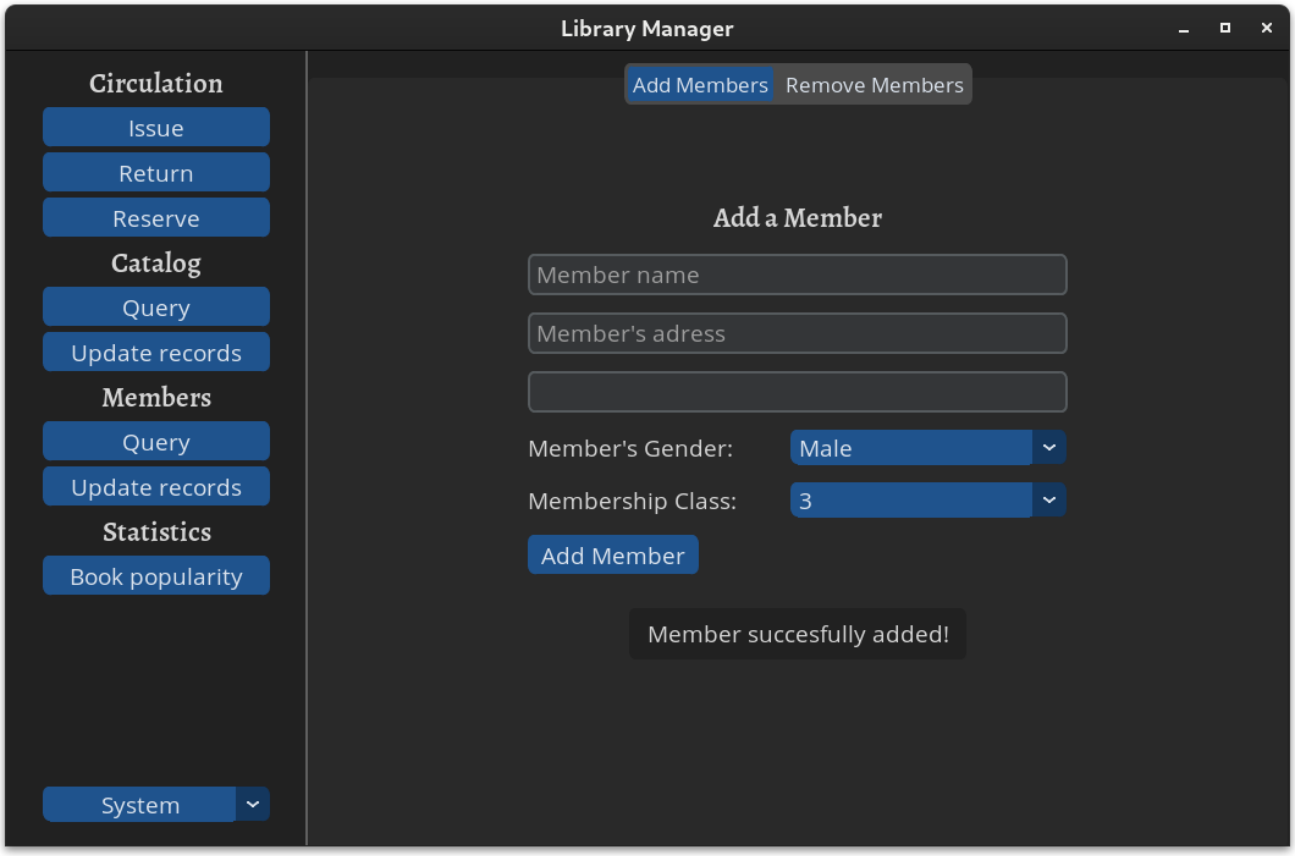
- Error while displaying member – member not found



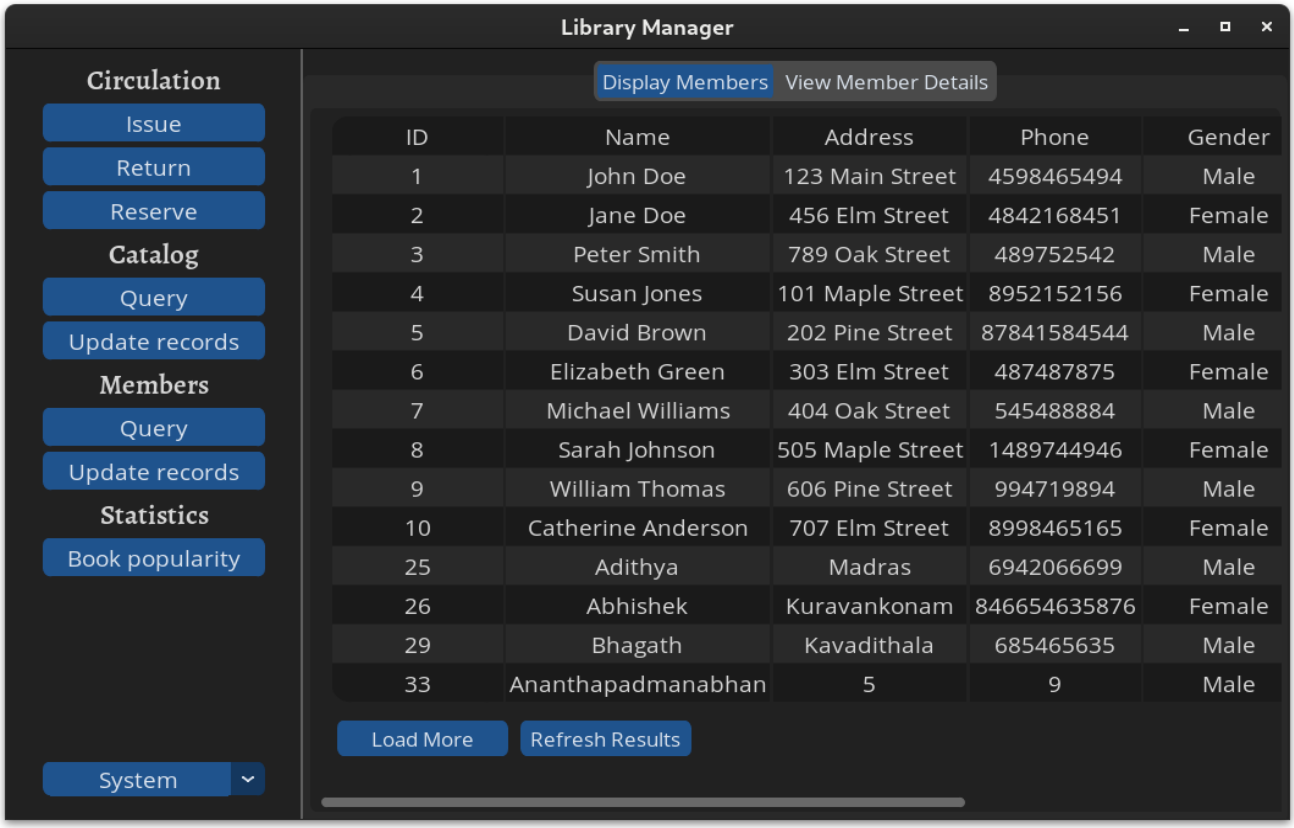
- Member being added using entries



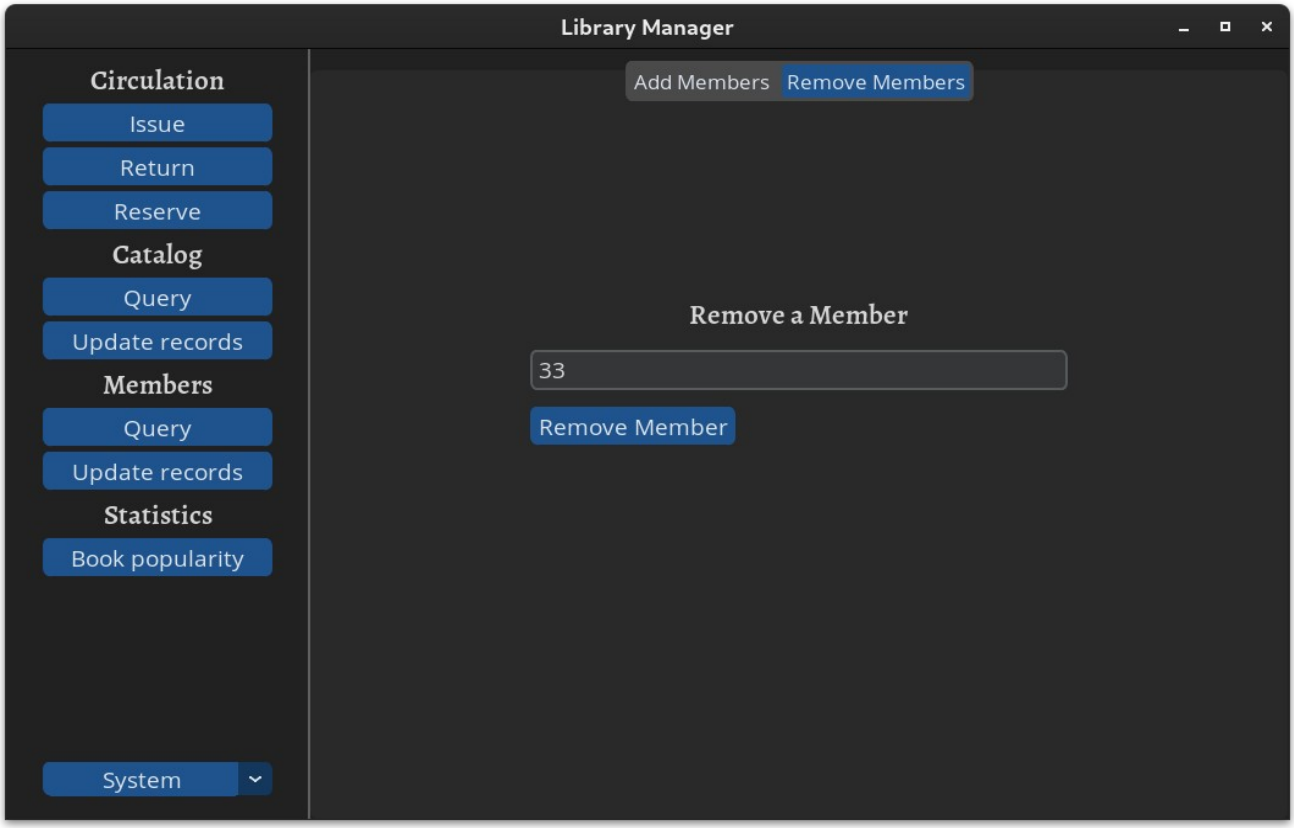
- Member successfully added



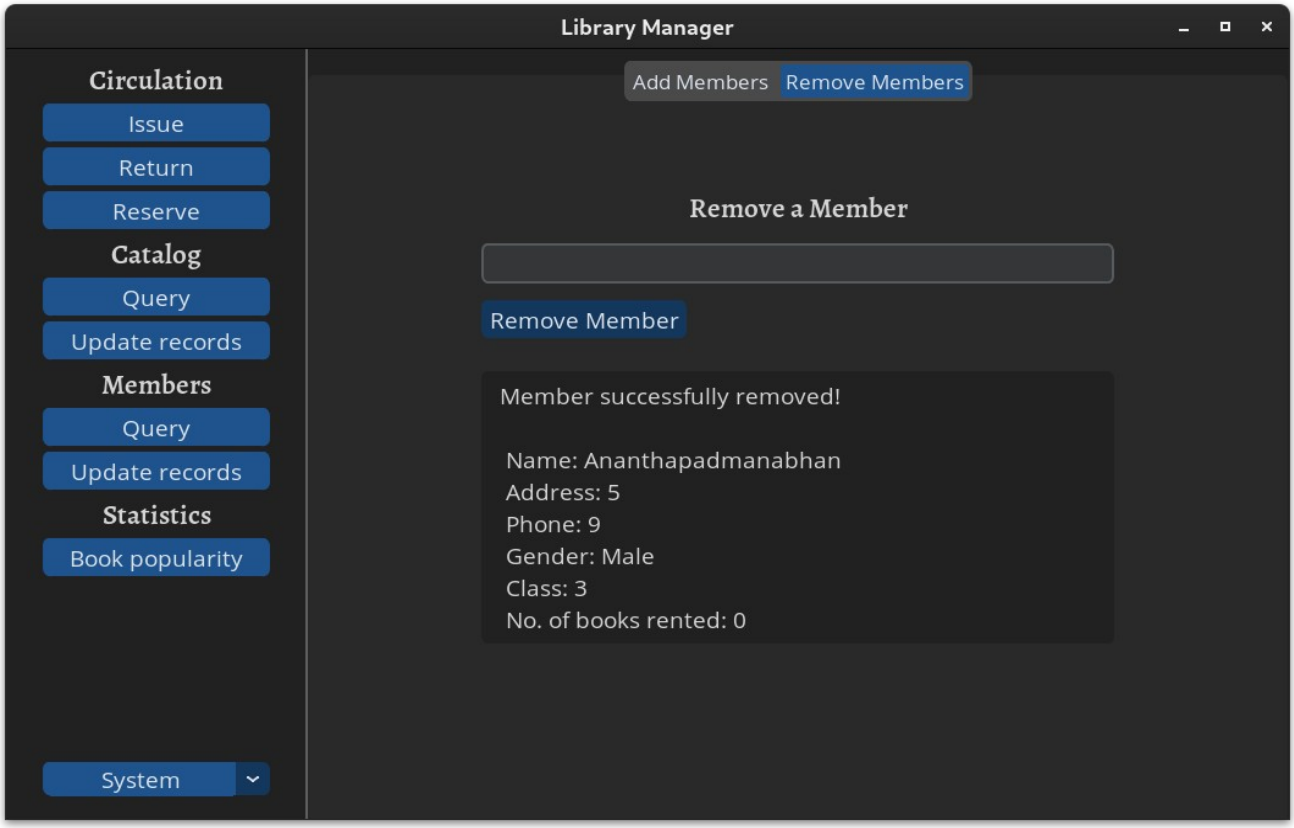
- Members list refreshed after adding a member



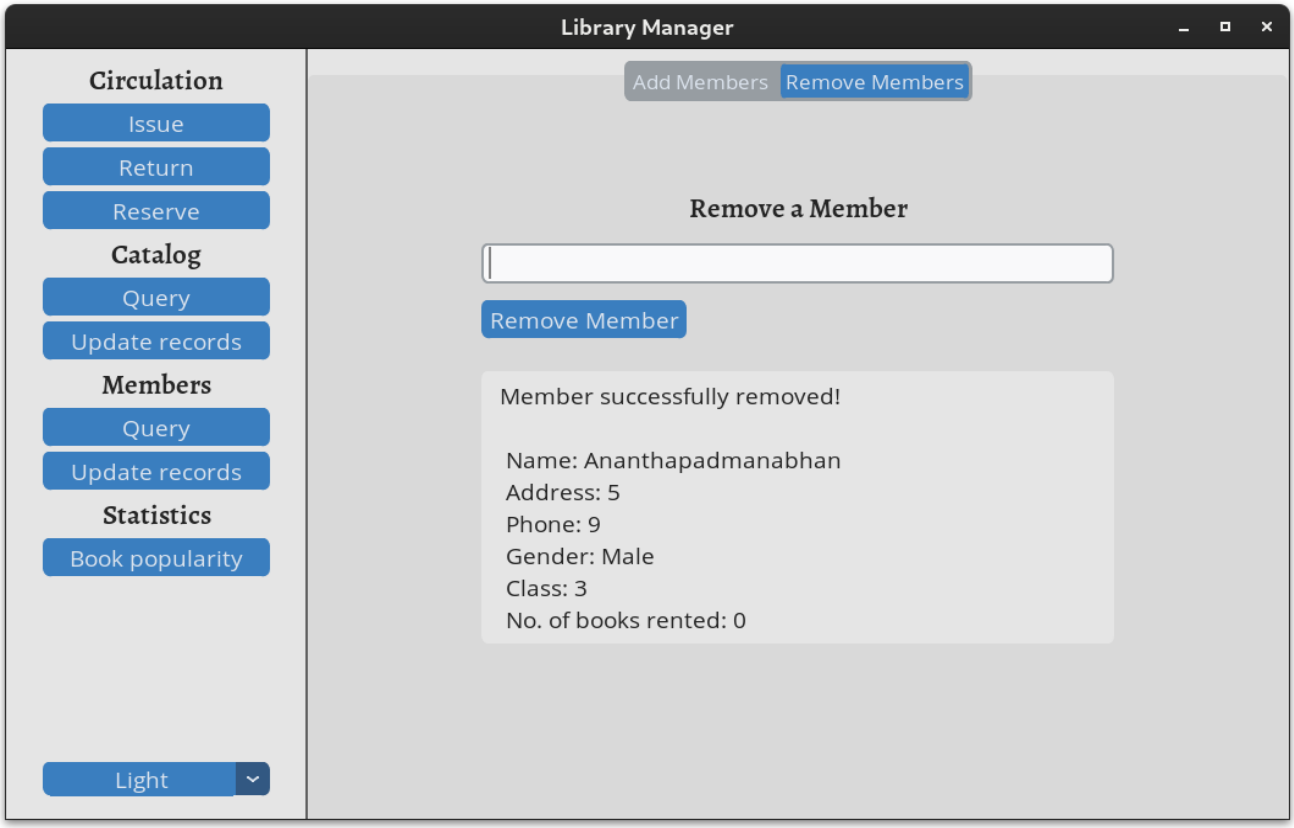
- Removing a member



- Member successfully removed and information displayed



- Library Manager app in light mode(courtesy of customtkinter)



Bibliography

- Python.org
- Computer Science with Python – Sumita Arora
- https://customtkinter.tomschimansky.com/documentation/
- https://github.com/Akascape/CTkXYFrame
- https://github.com/Akascape/CTkTable