Library Management System

Ebrahim Golriz

December 2023

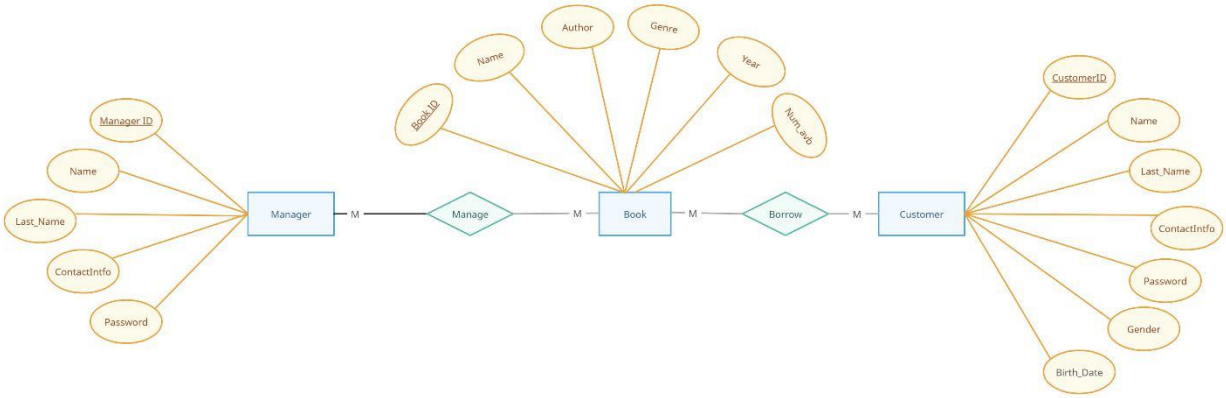
To implement a library system, we need some basic entities:

- **Book**: Information about each book and the number of each book in the library. There must be at least 1 copy of each book in the library.
- **Customer**: Contains personal information about the customer and the books he has borrowed. The customer can search through the library books, select the book he wants and borrow it. He also has access to the list of books he has borrowed and can return them to the library.
- **Manager**: Contains the personal information of the administrator; the administrator has access to the list of books and customers. He can see the books that a particular customer is currently borrowing.

In a hypothetical scenario,

The customer must register and log in with a password and Log in to the program, then he can view the available books and search based on the ID, Filter their various features. Then he can borrow the book he wants.

The manager can change book information or add a new book.



ER Chart Intended for this syste

Required tables created in *SQL Server:*

CreateTables.sql

```sql
CREATE TABLE books (
    b_id INT IDENTITY(1,1) PRIMARY KEY,
    b_name VARCHAR(100),
    author VARCHAR(100),
    genre VARCHAR(50),
    p_year INT,
    num_avb INT
);

CREATE TABLE managers (
    m_id INT IDENTITY(1,1) PRIMARY KEY,
    m_name VARCHAR(100),
    m_lastname VARCHAR(100),
    m_contactinfo VARCHAR(100),
    m_password VARCHAR(50)
);


CREATE TABLE customers (
    c_id INT IDENTITY(1,1) PRIMARY KEY,
    c_name VARCHAR(100),
    c_lastname VARCHAR(100),
    c_contactinfo VARCHAR(100),
    c_password VARCHAR(50),
    c_gender VARCHAR(10),
    c_birthdate DATE
);


CREATE TABLE customers_books (
    c_id INT,
    b_id INT,
    Borrow_Date DATE,
    FOREIGN KEY (c_id) REFERENCES customers(c_id),
    FOREIGN KEY (b_id) REFERENCES books(b_id),
    PRIMARY KEY (c_id, b_id)
);

CREATE TABLE customers_books_history (
    c_id INT,
    b_id INT,
    date_borrowed DATE,
    date_returned DATE default '9999-12-31',
    PRIMARY KEY (c_id, b_id, date_borrowed),
    FOREIGN KEY (c_id) REFERENCES customers(c_id),
    FOREIGN KEY (b_id) REFERENCES books(b_id)
);
```

Using sqlalchemy ,We add the database and its tables to the program in the Python environment.

```python
# Define the database connection string
db_connection_string = 'mssql+pyodbc://DESKTOP-3KLDFB9/LibraryDB?driver=ODBC+Driver+17+for+SQL+Server'

# Create the SQLAlchemy engine using PyODBC
db_engine = create_engine(db_connection_string)

# Create metadata
metadata = MetaData()

managers_table = Table('managers', metadata, autoload_with=db_engine)
customers_table = Table('customers', metadata, autoload_with=db_engine)
books_table = Table('books', metadata, autoload_with=db_engine)
customers_books_table = Table('customers_books', metadata, autoload_with=db_engine)
customers_books_history_table = Table('customers_books_history', metadata, autoload_with=db_engine)

```

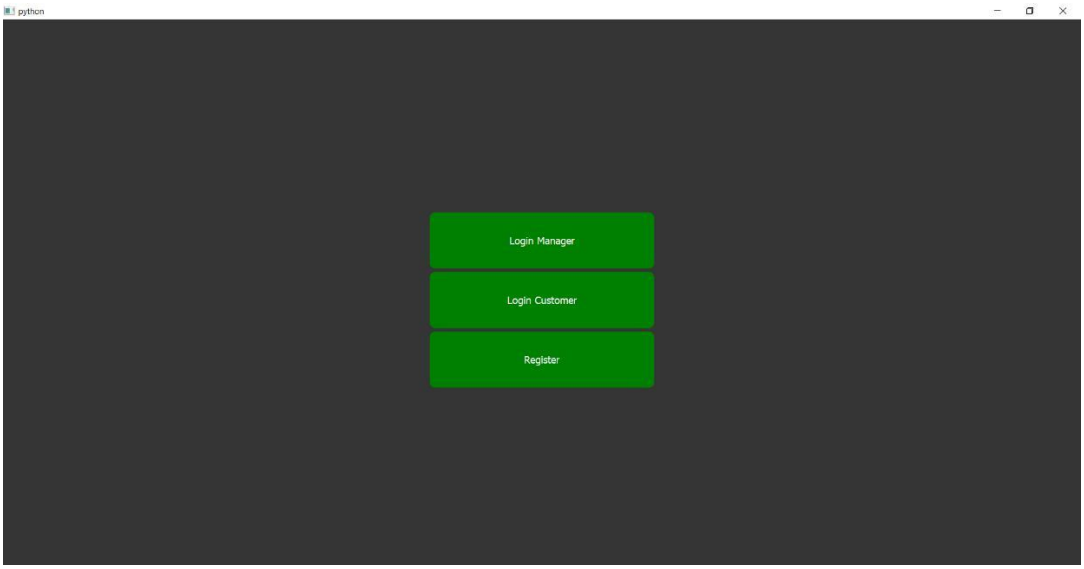
This section also includes the necessary preparations for creating a user interface using PyQt:

```python
def main():
    app = QApplication(sys.argv)



    app.setStyle('Fusion')

    palette = QPalette()
    palette.setColor(QPalette.Window, QColor(53, 53, 53))
    palette.setColor(QPalette.WindowText, Qt.white)
    palette.setColor(QPalette.Base, QColor(25, 25, 25))
    palette.setColor(QPalette.AlternateBase, QColor(53, 53, 53))
    palette.setColor(QPalette.ToolTipBase, Qt.white)
    palette.setColor(QPalette.ToolTipText, Qt.white)
    palette.setColor(QPalette.Text, Qt.white)
    palette.setColor(QPalette.Button, QColor(53, 53, 53))
    palette.setColor(QPalette.ButtonText, Qt.white)
    palette.setColor(QPalette.BrightText, Qt.red)
    palette.setColor(QPalette.Link, QColor(42, 130, 218))
    palette.setColor(QPalette.Highlight, QColor(116, 215, 112))
    palette.setColor(QPalette.HighlightedText, Qt.black)

    app.setPalette(palette)

    page_controller = PageController()
    page_controller.get_widget().show()
    sys.exit(app.exec_())

if __name__ == '__main__':
    main()

```

Now, by running a simple scenario, we will examine how the program works and how it interacts with the created database.

Customer

At the beginning of the program, we are presented with the following page:



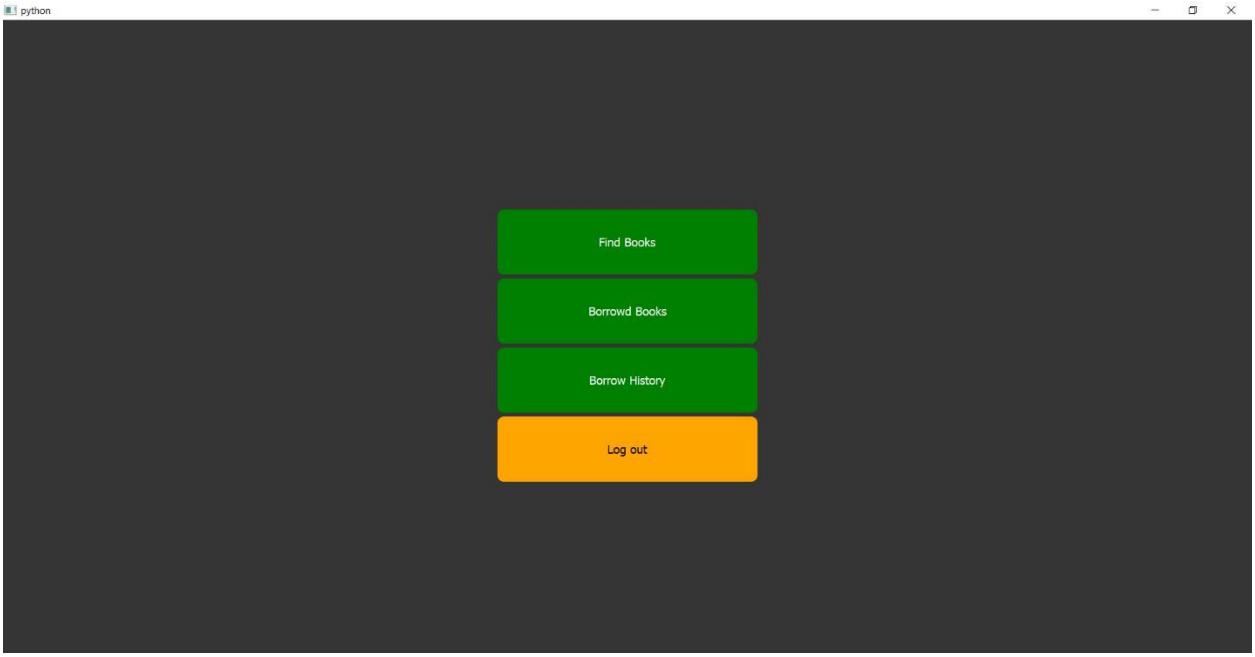
By pressing the button Register, We enter the following page:

On this page, the user enters their information and presses the button **Submit**. The entered information, by creating a query and executing it, is converted into a new row or instance of the table **customers.**

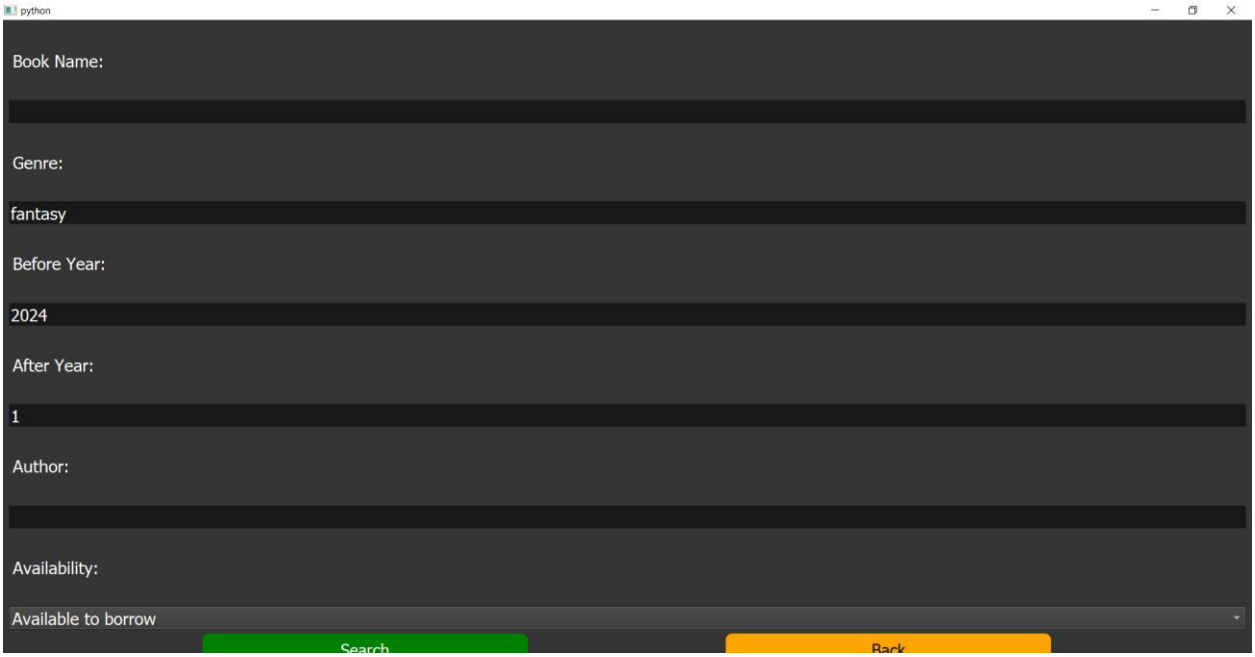
```python
        # Get the input values
        name = self.name_input.text()
        lastname = self.lastname_input.text()
        contactinfo = self.contactinfo_input.text()
        password = self.password_input.text()
        gender = self.gender_input.currentText()
        birthdate = self.birthdate_input.date()


        if not name or not lastname or not contactinfo or not password or not gender:
            QMessageBox.warning(self, 'Input Error', 'All fields must be filled.')
            return
        if birthdate.year() >= 2004:
            QMessageBox.warning(self, 'Input Error', 'Birthdate must be before 2004.')
            return
        birthdate = self.birthdate_input.date().toString("yyyy-MM-dd")

        query = customers_table.insert().values(
            c_name=name,
            c_lastname=lastname,
            c_contactinfo=contactinfo,
            c_password=password,
            c_gender=gender,
            c_birthdate=birthdate
        )

        with db_engine.connect() as connection:
            result = connection.execute(query)
            connection.commit()
```

After the information is confirmed, during a message, the user is given an ID that they can use to log in to the system.

The user will now be redirected to their home page:



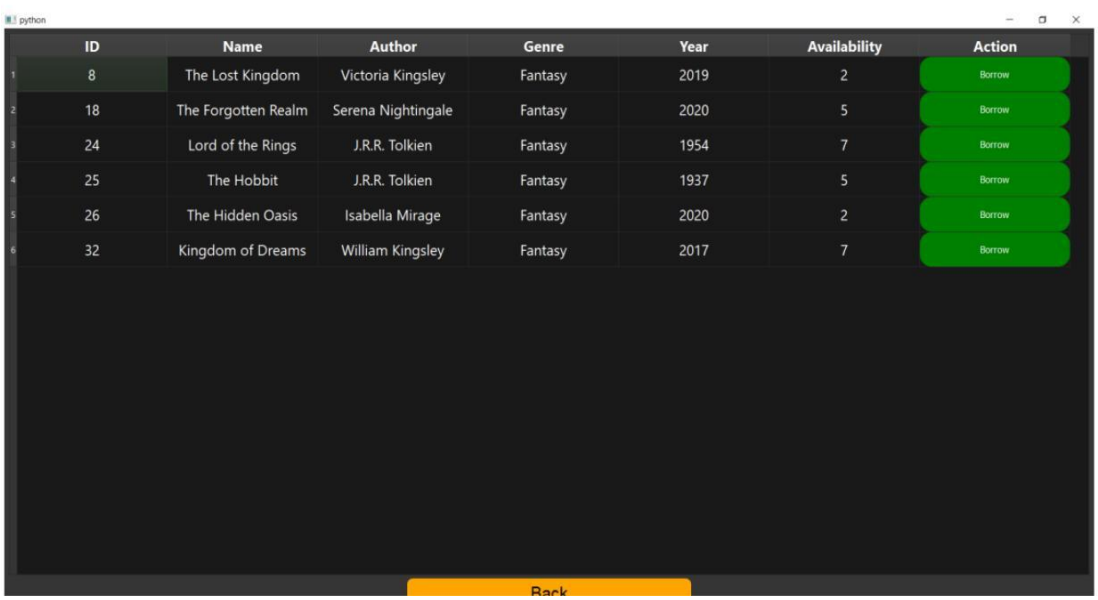
By pressing the button **Find Books**, The following page will open where the user can apply the filters they need to the list of books in the table **books.**



Available to borrow option Only shows books that the library have more than 1 of.

Search command with filters applied by a query is executed, the result is the rows fetched from the table and If the size value is zero, it means that there was no book with the applied filters in the books table:

```python
def submit_search(self):
    # Get the input values
    b_name = self.b_name_input.text()
    genre = self.genre_input.text()
    before_year = self.before_year_input.text()
    if (not before_year):
        before_year = 2024
    after_year = self.after_year_input.text()
    if (not after_year):
        after_year = 1
    author = self.author_input.text()
    availability = self.availability_input.currentText()
    query = select(books_table).where(
        (books_table.c.b_name.like(f'%{b_name}%')) &
        (books_table.c.genre.like(f'%{genre}%')) &
        (books_table.c.p_year <= int(before_year)) &
        (books_table.c.p_year >= int(after_year)) &
        (books_table.c.author.like(f'%{author}%')) &
        (
            (books_table.c.num_avb > 1)
            if availability == 'Available to borrow'
            else True
        )
    )
    with db_engine.connect() as connection:
        result = connection.execute(query)
        rows = result.fetchall()
        connection.commit()

    # Check if any books were found
    if len(rows) == 0:
        QMessageBox.warning(self, 'No Books Found', 'No books were found that match your search criteria.')
    else:
```



List of books shown to the user

Now by pressing the button **Borrow** For each book, it is first checked whether the user has permission to borrow that book; The conditions are:

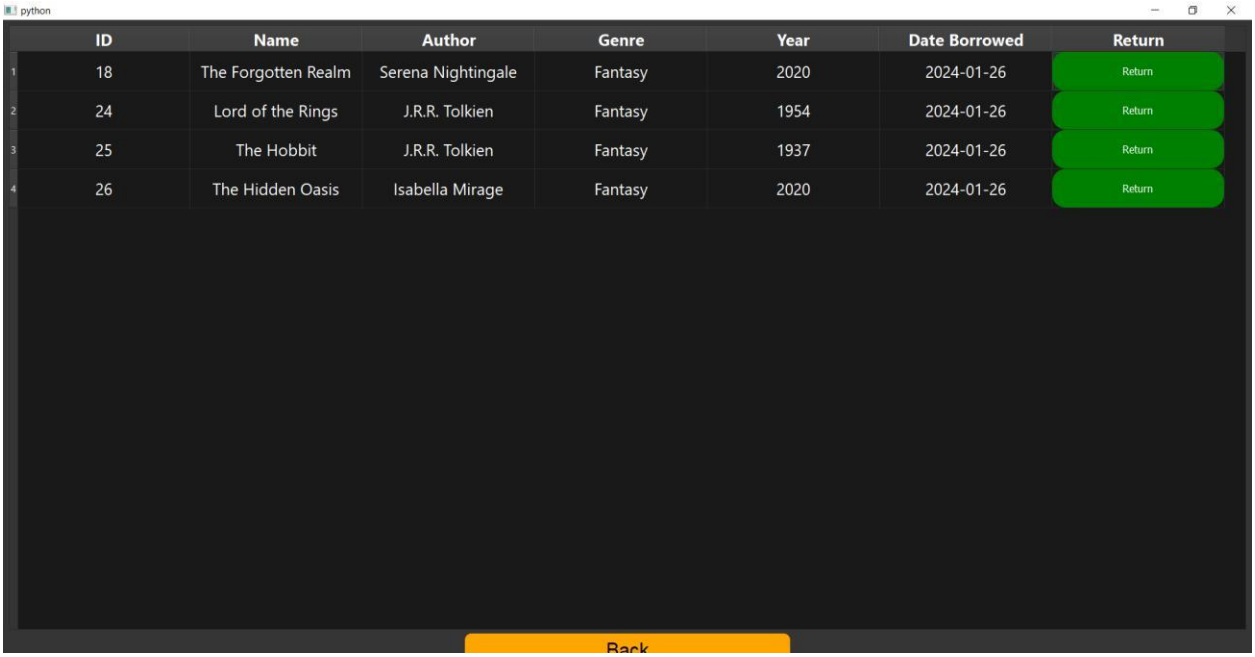
1. The stock level of the desired book must be at least 2.
2. The user has not currently borrowed that book.
3. The user cannot borrow and return a book on the same day and then want to borrow that book again on the same day.

```python
with db_engine.connect() as connection:

    # Check the availability of the book
    query = select(books_table.c.num_avb).where(books_table.c.b_id == book_id)

    num_avb_item = self.table.item(row, 5)

    if num_avb <= 1:
        QMessageBox.warning(self, 'Book Unavailable', f'This is the last available stock of book {book_id}, "{book_name}". Can not be borrowed.')
        return
    query = select(customers_books_table.c).where(
        (customers_books_table.c.c_id == self.customer_id) &
        (customers_books_table.c.b_id == book_id)
    )
    result = connection.execute(query)
    row = result.fetchone()
    if row is not None:
        QMessageBox.warning(self, 'Book Already Borrowed', f'You have already borrowed one copy of book {book_id}, "{book_name}".')
        return

    # Check if the book has already been borrowed by the customer on the same day
    query = select(customers_books_history_table.c).where(
        (customers_books_history_table.c.c_id == self.customer_id) &
        (customers_books_history_table.c.b_id == book_id) &
        (customers_books_history_table.c.date_borrowed == date.today())
    )
    result = connection.execute(query)
    row = result.fetchone()
    if row is not None:
        QMessageBox.warning(self, 'Book Already Borrowed Today and then returned, ', f'You have already borrowed book {book_id}, "{book_name}" today and returned it.')
        return
```

Next, if the above conditions are met, first Insert book and customer ID in customers_books table along with the date of the day. Then the same values are also entered in customers_books_history table. After that, the inventory level of that book decreases by one unit:

```python
        # Insert the customer_id and book_id into the customers_books table
        query = customers_books_table.insert().values(
            c_id=self.customer_id,
            b_id=book_id,
            Borrow_Date=date.today()
        )
        connection.execute(query)

        #  Insert the customer_id and book_id into the customers_books_history table
        query = customers_books_history_table.insert().values(
            c_id=self.customer_id,
            b_id=book_id,
            date_borrowed = date.today()
        )
        connection.execute(query)
        # update the book's availablity
        query = (
            update(books_table).
            where(books_table.c.b_id == book_id).
            values(num_avb=books_table.c.num_avb - 1)
        )
        connection.execute(query)
        connection.commit()


    self.find_books_page.submit_search()
    QMessageBox.information(self, 'Book Borrowed', f'Customer {self.customer_id} borrowed book {book_id}, "{book_name}".')
```

By pressing the button **Borrowed Books** On the user's home page, we are presented with the following page, which is a list of books that the customer has purchased. Currently borrowed, along with book details and borrowing date:



This list is created by creating a select query in which the tables customers_books_table and books_table with the same ID are joined together, the rows where the user ID is the same as the ID in customers_books are selected, and the desired columns are picked:

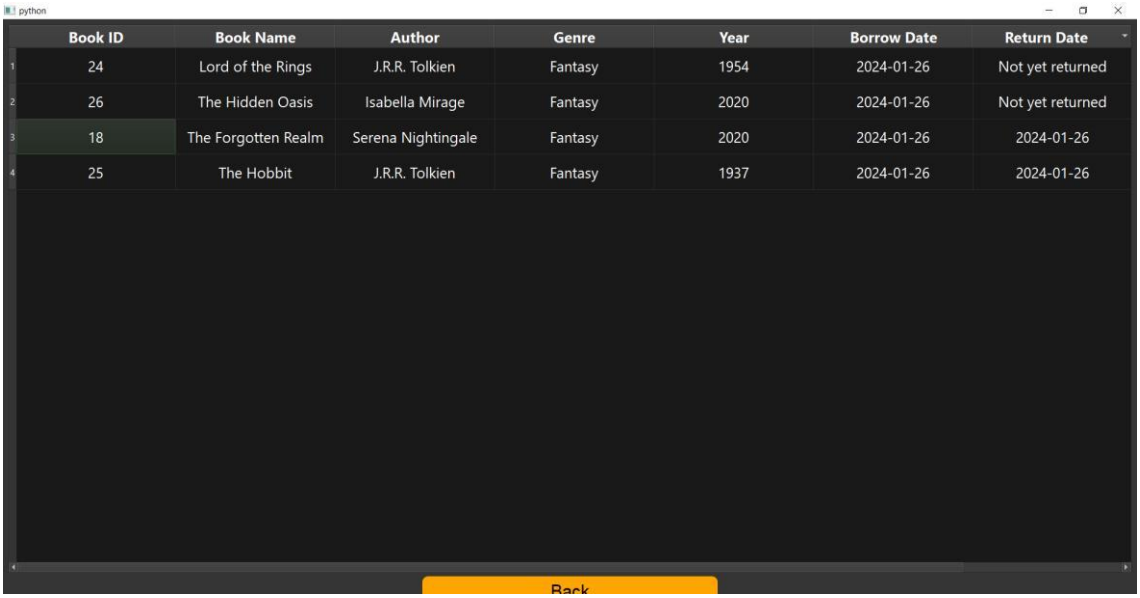
```
def find_borrowed_books(self):
    # Join the tables and select the columns
    query = (
        select(
            books_table.c.b_id,
            books_table.c.b_name,
            books_table.c.author,
            books_table.c.genre,
            books_table.c.p_year,
            customers_books_table.c.Borrow_Date
        )
        .join_from(customers_books_table, books_table, customers_books_table.c.b_id == books_table.c.b_id)
        .where(customers_books_table.c.c_id == self.customer_id)
    )

    with db_engine.connect() as connection:
        result = connection.execute(query)
        rows = result.fetchall()

    if len(rows) == 0:
        QMessageBox.warning(self, 'No Books Found', 'You do not currently have a borrowed book')
        self.page_controller.show_customer_main_page()
```

If there are no rows as a result of this fetch, it means the user has not currently borrowed a book.

By pressing the **Return** button in the list of books borrowed by the user, the **customers_books_history** table is first updated, and for the desired row (given the customer ID and book and the return date, which is equivalent to the default date), the value of the **date_returned** column is changed to the current date; then the inventory of that book is increased by one unit, and that desired row is deleted from the **customers_books** table, given the customer ID and book.

```python
with db_engine.connect() as connection:

    # update customers_books_history table
    query = update(customers_books_history_table).where(
    customers_books_history_table.c.b_id == book_id,
    customers_books_history_table.c.c_id == self.customer_id,
    customers_books_history_table.c.date_returned == '9999-12-31'
    ).values(
        date_returned = date.today()
    )
    connection.execute(query)

    # update the book's availablity
    query = update(books_table).where(books_table.c.b_id == book_id).values(num_avb = books_table.c.num_avb + 1)
    connection.execute(query)

    # remove item from customers_books table
    query = delete(customers_books_table).where(
    customers_books_table.c.b_id == book_id,
    customers_books_table.c.c_id == self.customer_id
    )

    connection.execute(query)

    connection.commit()
```

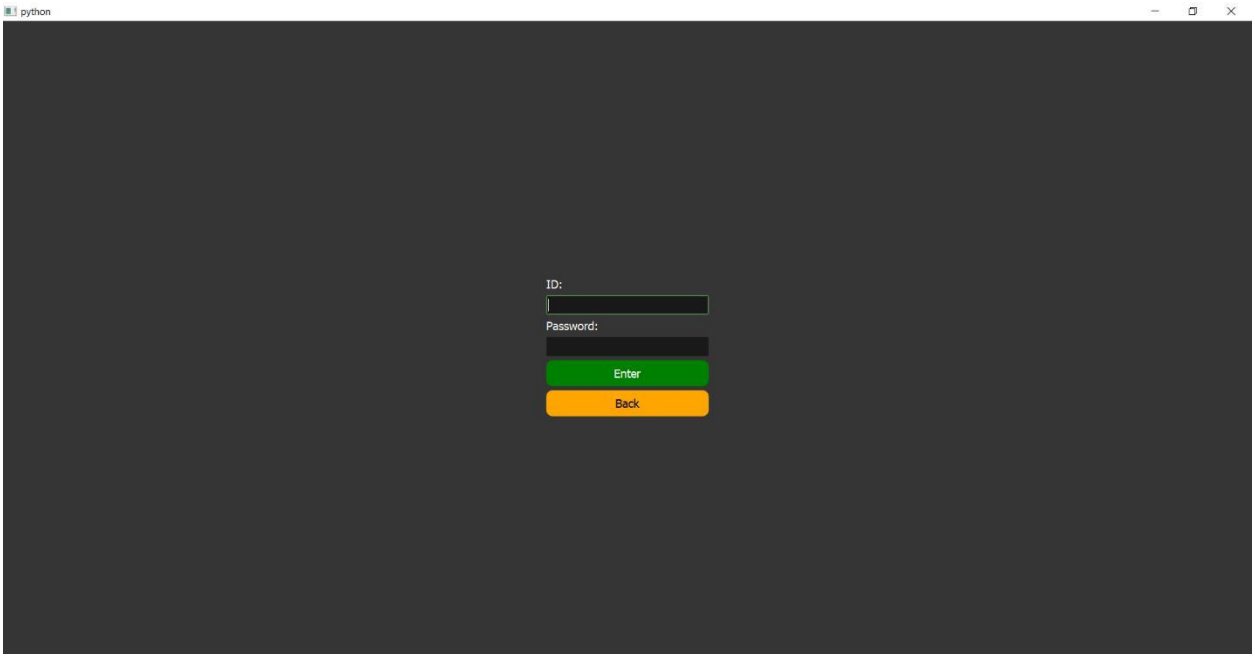
By pressing the **Borrow History** button on the user's home page, the rows related to the user are fetched from the **customers_books_history** table according to the select statement explained in the previous section and displayed to the user:



	Book ID	Book Name	Author	Genre	Year	Borrow Date	Return Date
1	24	Lord of the Rings	J.R.R. Tolkien	Fantasy	1954	2024-01-26	Not yet returned
2	26	The Hidden Oasis	Isabella Mirage	Fantasy	2020	2024-01-26	Not yet returned
3	18	The Forgotten Realm	Serena Nightingale	Fantasy	2020	2024-01-26	2024-01-26
4	25	The Hobbit	J.R.R. Tolkien	Fantasy	1937	2024-01-26	2024-01-26

Manager

The manager must first log in to their account (an instance of the administrator in the administrators table is already entered into the database)

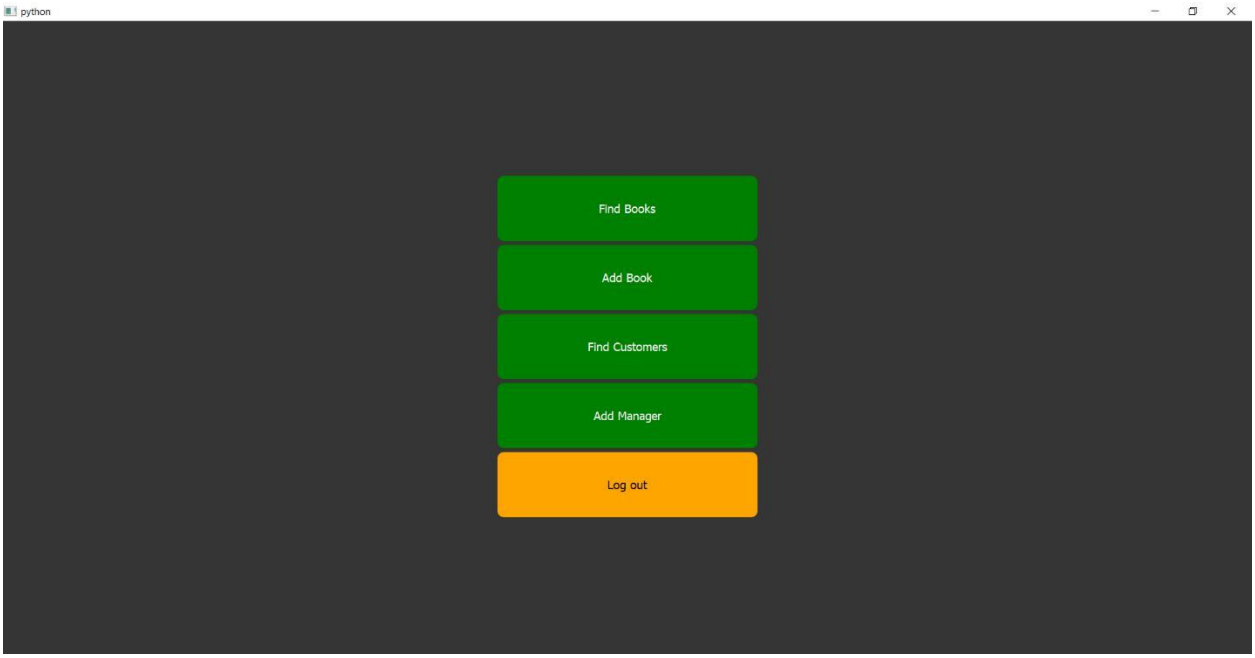


By entering the information, the ID and Password entered in the managers table are selected and if such a row exists, the manager is taken to the main management page:

```
with db_engine.connect() as connection:
    query = select(managers_table).where(
        (managers_table.c.m_id == entered_id) & (managers_table.c.m_password == entered_password)
    )
    result = connection.execute(query).fetchall()

    if len(result) > 0:
        QMessageBox.information(self, 'Login Status', 'Welcome!')
        self.page_controller.managermainpage.manager_id = entered_id
        self.page_controller.show_manager_main_page()
    else:
        QMessageBox.warning(self, 'Login Status', 'User not found.')
        self.line_edit_id.clear()
        self.line_edit_password.clear()
```

User login is done in the same way.

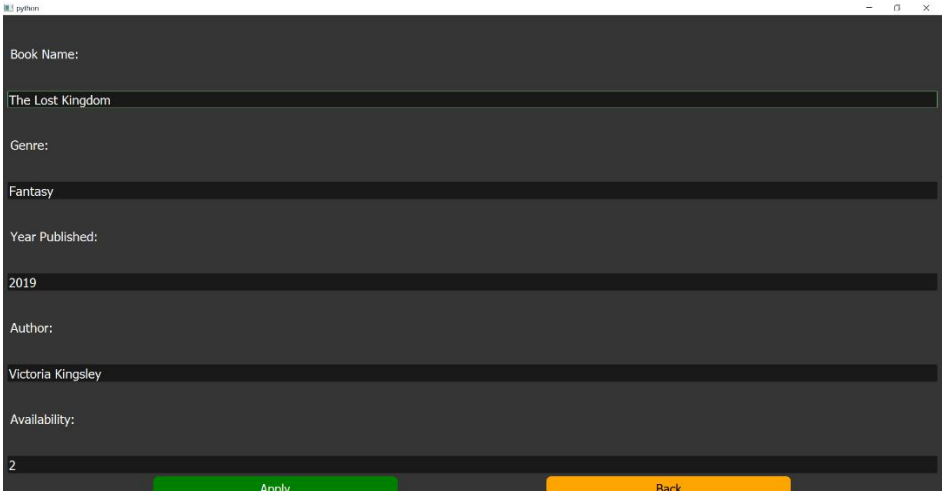
The admin home page looks like this:



The manager's home page looks like this: By pressing the **Find Books** button, we are redirected to the same page that was shown to the user to filter the books, and after applying the necessary filters, the list of books is shown to the administrator:

As you can see, for each book, there are two buttons for the manager, **Edit** and List of people who have currently **borrowed** that book.

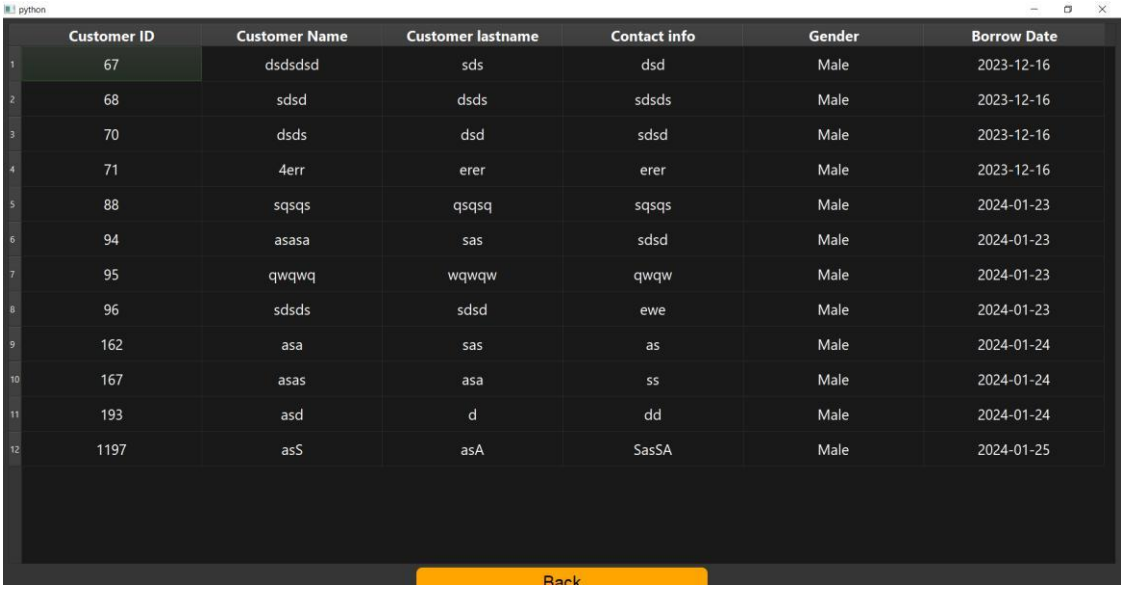
By pressing the button **Edit** The following page is shown to the administrator where he can change the book information:



By confirming the entered information, the values of the properties of that book are replaced with the entered values, and the table **books** will be updated.

```
def applychanges(self):
    b_name = self.b_name_input.text()
    genre = self.genre_input.text()
    author = self.author_input.text()


    try:
        p_year = int(self.year_published_input.text())
        num_avb = int(self.availability_input.text())
    except ValueError:
        QMessageBox.warning(self, 'Input Error', 'Year and Availability must be numbers.')
        return


    if not b_name or not genre or not p_year or not author or not num_avb :
        QMessageBox.warning(self, 'Input Error', 'All fields must be filled.')
        return

    update_stmt = update(books_table).where(books_table.c.b_id == self.book_id).values(
        b_name=b_name,
        genre=genre,
        p_year=p_year,
        author=author,
        num_avb=num_avb
    )

    with db_engine.connect() as connection:
        result = connection.execute(update_stmt)
        connection.commit()
```

By pressing the button **Current Borrowers**, a list of customers who have currently borrowed that book will be displayed:

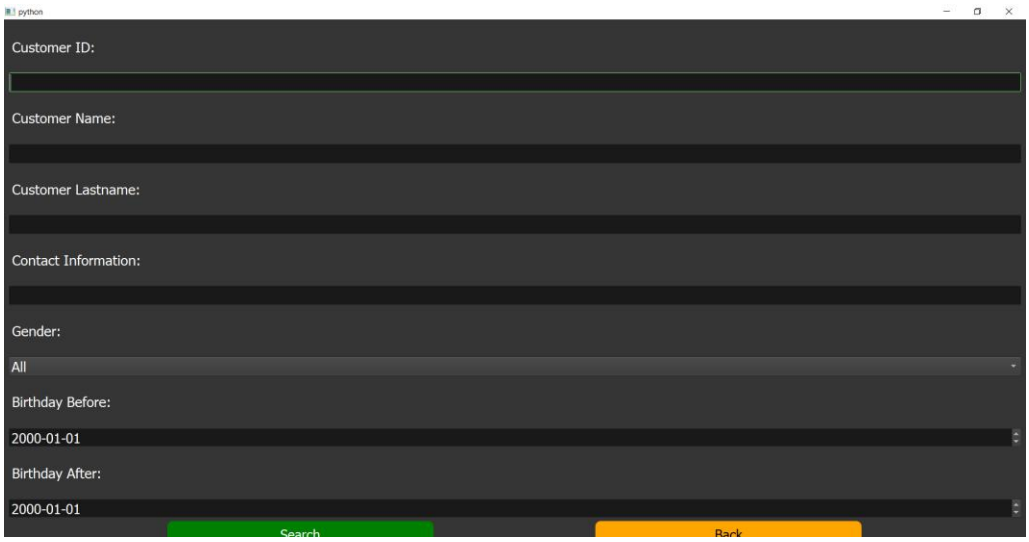


	Customer ID	Customer Name	Customer lastname	Contact info	Gender	Borrow Date
1	67	dsdsdsd	sds	dsd	Male	2023-12-16
2	68	sdsd	dsds	sdsds	Male	2023-12-16
3	70	dsds	dsd	sdsd	Male	2023-12-16
4	71	4err	erer	erer	Male	2023-12-16
5	88	sqsqs	qsqsq	sqsqs	Male	2024-01-23
6	94	asasa	sas	sdsd	Male	2024-01-23
7	95	qwqwq	wqwqw	qwqw	Male	2024-01-23
8	96	sdsds	sdsd	ewe	Male	2024-01-23
9	162	asa	sas	as	Male	2024-01-24
10	167	asas	asa	ss	Male	2024-01-24
11	193	asd	d	dd	Male	2024-01-24
12	1197	asS	asA	SasSA	Male	2024-01-25

This operation is performed by joining the two tables **customers_books** and **customers**, and selecting those rows where the ID of the desired book is equal to the same ID in **customers_books.**

```
def load_data(self):
    self.table.setSortingEnabled(False)
    self.table.clear()

    query = (
        select(
            customers_table.c.c_id,
            customers_table.c.c_name,
            customers_table.c.c_lastname,
            customers_table.c.c_contactinfo,
            customers_table.c.c_gender,
            customers_books_table.c.Borrow_Date,
        )
        .join_from(customers_books_table, customers_table, customers_books_table.c.c_id == customers_table.c.c_id)
        .where(customers_books_table.c.b_id == self.book_id)
    )

    with db_engine.connect() as connection:
        result = connection.execute(query)
        rows = result.fetchall()

    if len(rows) == 0:
        QMessageBox.warning(self, 'No current borrowers', 'This books has no current borrowers')
        self.page_controller.show_booklist_page()
```

By pressing the button **Find Customers**, The following page is displayed, giving the administrator the ability to filter customers based on their characteristics:



This is done by executing the select command with the condition entered by the administrator:

```python
def submit_search(self):
    # Get the input values
    c_id_text = self.c_id_input.text()
    c_name = self.c_name_input.text()
    c_lastname = self.c_lastname_input.text()
    c_contactinfo = self.c_contactinfo_input.text()
    c_gender = self.c_gender_input.currentText()
    c_birthdate_before = self.birthdate_before_input.date().toString("yyyy-MM-dd")
    c_birthdate_after = self.birthdate_after_input.date().toString("yyyy-MM-dd")

    query = select(customers_table).where(
        (customers_table.c.c_name.like(f'%{c_name}%')) &
        (customers_table.c.c_lastname.like(f'%{c_lastname}%')) &
        (customers_table.c.c_contactinfo.like(f'%{c_contactinfo}%')) &
        (customers_table.c.c_birthdate.between(c_birthdate_after, c_birthdate_before))
    )
    if c_id_text:
        try:
            c_id = int(c_id_text)
        except ValueError:
            QMessageBox.warning(self,'Input Error', 'ID must be a number.')
            return
        query = query.where(customers_table.c.c_id == c_id)

    if c_gender != "All":
        query = query.where(customers_table.c.c_gender == c_gender)

    with db_engine.connect() as connection:
        result = connection.execute(query)
        rows = result.fetchall()
        connection.commit()

    # Check if any customers were found
    if len(rows) == 0:
        QMessageBox.warning(self, 'No Customers Found', 'No customers were found that match your search criteria.')
```

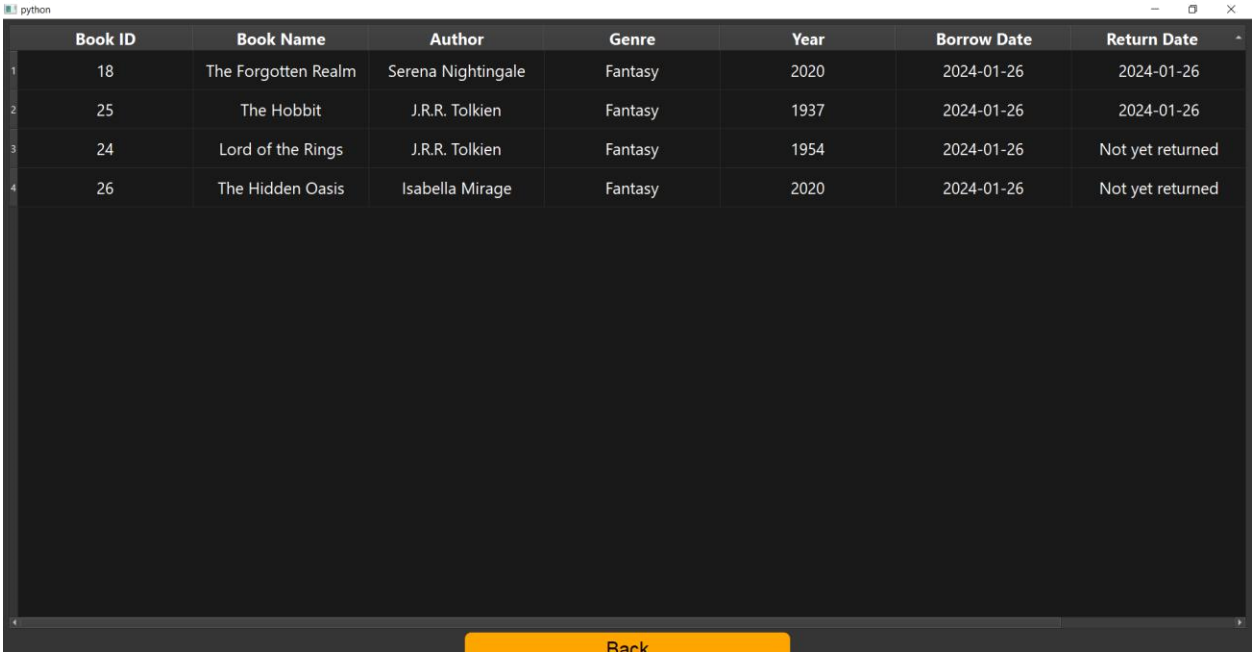
python

	ID	Name	Lastname	Contact info	Password	gender	Birthdate	Action
1	24	asasasa	sasa	sasas	asasasas	Male	2000-01-01	Books
2	25	ssdf	sdfsdf	sdfs	dfsdfsdf	Male	2000-01-01	Books
3	26	utyjtjytj	tyjty	jtyjyt	jtyjty	Male	2000-01-01	Books
4	27	asdasd	asdsadsa	dasd	asdsad	Male	2000-01-01	Books
5	28	sdasd	asdas	dasdasd	sadsad	Male	2000-01-01	Books
6	29	dvsdff	dsfd	fsdfs	dfsdfsd	Male	2000-01-01	Books
7	30	asdsad	sdasdas	dasdas	dasdasdas	Male	2000-01-01	Books
8	31	sdasd	asdasd	asdasd	asdasd	Male	2000-01-01	Books
9	32	sdadasd	dasdasd	sdas	asda	Male	2000-01-01	Books
10	33	sdadasd	dasdasd	sdas	asda	Male	2000-01-01	Books
11	34	sdadasd	dasdasd	sdas	asda	Male	2000-01-01	Books
12	35	f	f	f	f	Male	2000-01-01	Books
13	36	sas	as	asas	as	Male	2000-01-01	Books
14	37	yhj	jgh	hjgh	hjg	Male	2000-01-01	Books
15	38	asAS	SASAS	ASA	AS	Male	2000-01-01	Books

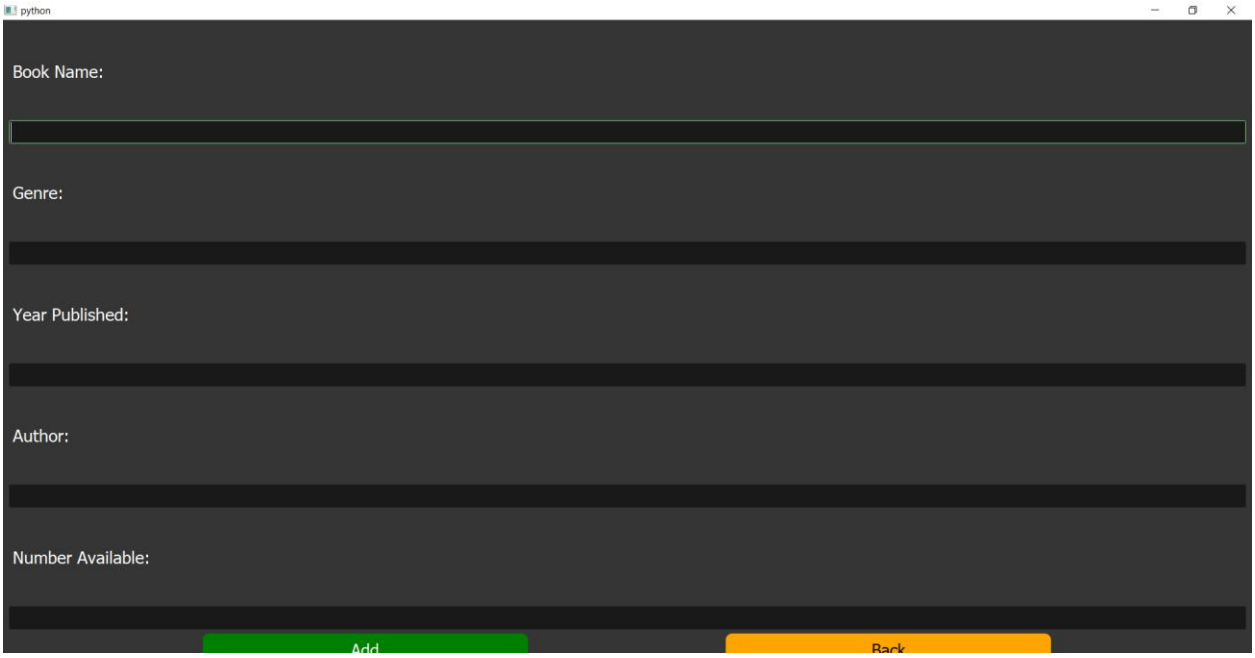
Back

In the list shown to the administrator, for each customer there is a **Books** button that shows the books that the customer has borrowed so far.

```python
def load_data(self):
    self.table.setSortingEnabled(False)
    self.table.clear()

    query = (
        select(
            books_table.c.b_id,
            books_table.c.b_name,
            books_table.c.author,
            books_table.c.genre,
            books_table.c.p_year,
            customers_books_history_table.c.date_borrowed,
            customers_books_history_table.c.date_returned
        )
        .join_from(customers_books_history_table, books_table, customers_books_history_table.c.b_id == books_table.c.b_id)
        .where(customers_books_history_table.c.c_id == self.customer_id)
    )

    with db_engine.connect() as connection:
        result = connection.execute(query)
        rows = result.fetchall()

    if len(rows) == 0:
        QMessageBox.warning(self, 'No Books Found', 'No books have been borrowed yet')
```

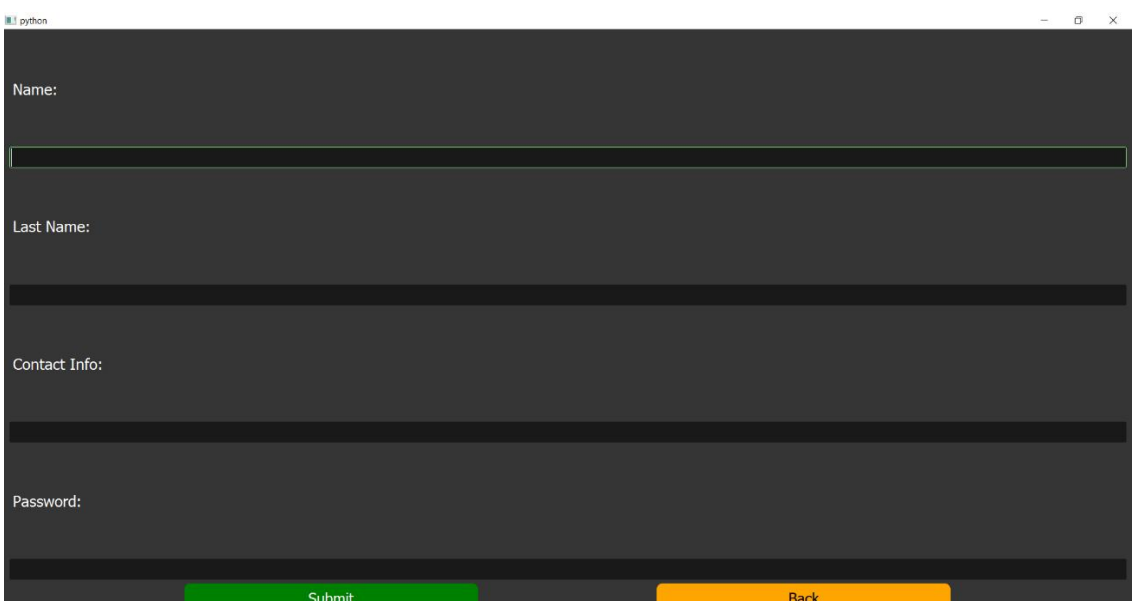


By pressing the button **Add Book** On the main page of the administrator, the following page is displayed, by entering the information of a new book, that book will be added to Table **books.**



```python
def addbook(self):

    # Get the input values
    b_name = self.b_name_input.text()
    genre = self.genre_input.text()
    author = self.author_input.text()
    try:
        year = int(self.year_input.text())
        availability = int(self.availability_input.text())
    except ValueError:
        QMessageBox.warning(self, 'Input Error', 'Year and Availability must be numbers.')
        return


    if not b_name or not genre or not year or not author or not availability :
        QMessageBox.warning(self, 'Input Error', 'All fields must be filled.')
        return


    query = books_table.insert().values(
        b_name = b_name,
        author = author,
        genre = genre,
        p_year = year,
        num_avb = availability
    )

    with db_engine.connect() as connection:
        result = connection.execute(query)
        connection.commit()
```

By pressing the button **Add Manager**, the following page will be displayed and by entering the details and confirming it, a new administrator instance will be created.



```python
def submit_registration(self):

    # Get the input values
    name = self.name_input.text()
    lastname = self.lastname_input.text()
    contactinfo = self.contactinfo_input.text()
    password = self.password_input.text()



    if not name or not lastname or not contactinfo or not password:
        QMessageBox.warning(self, 'Input Error', 'All fields must be filled.')
        return


    query = managers_table.insert().values(
        m_name=name,
        m_lastname=lastname,
        m_contactinfo=contactinfo,
        m_password=password,

    )

    with db_engine.connect() as connection:
        result = connection.execute(query)
        connection.commit()
```