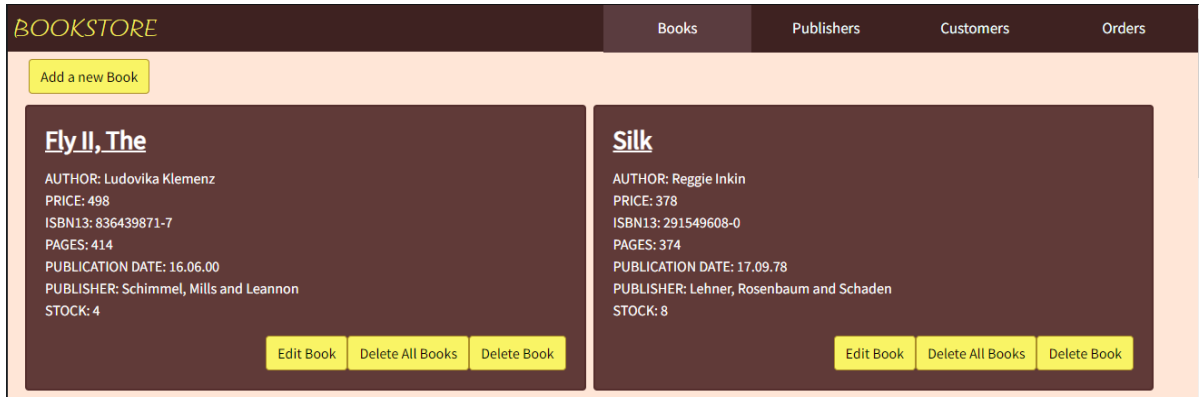
BOOKSTORE

Student: Lupu Maria
Grupa: 1308B
Coordonator: Buțincu Cristian

I. Descrierea proiectului

Prin crearea acestui proiect am urmărit proiectarea unei aplicații care să ofere angajaților unui magazin care vinde cărți, posibilitatea de a gestiona mult mai ușor întreg procesul acestuia. Prin procesul acestuia mă refer la:

- organizarea cărților urmărind cu ușurință datele acestora (autor, preț, numărul de pagini etc.);
- o listă cu toate editurile cu care colaborează magazinul respectiv;
- o listă cu clienții, plus informațiile lor pentru a putea fi livrate cărțile;
- afișarea comenzilor înregistrare și costul total al acestora.



Prin gestionarea procesului mă refer la posibilitatea:

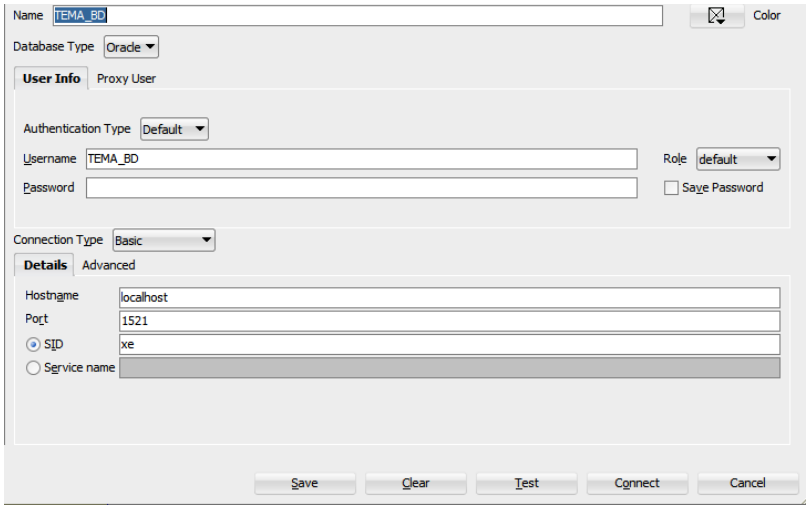
- de a putea modifica detaliile cărților înregistrate;
- de a adăuga mai multe scrieri sau edituri;
- de a șterge cărțile care nu vor mai fi vândute de magazinul respectiv, cât și publicația cu care nu se mai colaborează;
- de a putea anula sau trimite o comandă;
- de a putea vizualiza cărțile dintr-o comandă anume.

Acest procedeu l-am creat prin crearea unei baze de date și am implementat vizual funcționalitatea principalelor funcții de interogare a unei baze de date.

LINK DOCUMENTAȚIE FULL:
https://docs.google.com/document/d/1OmpMG_63RO8BpvPECt-oBopTaggFB4FEJQVLK-Tbz9c/edit?usp=sharing

II. Tehnologiile folosite pentru front-end și back-end

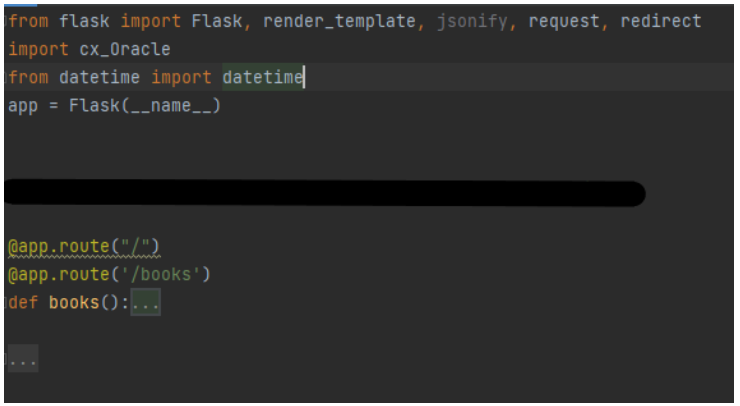
Baza de date din cadrul acestui proiect a fost creată cu ajutorul lui *Oracle Database Express Edition (XE) 11gR2* unde inițial aceasta a fost goală. Am făcut conexiunea ei în *SQL Developer*, lăsând-o localhost.



În acest mediu de dezvoltare integrat pentru lucrul cu SQL în bazele de date Oracle, *SQL Developer*, am creat tabelele și am făcut diverse modificări, interogări asupra ei. Am important și extensia din Python, **cx_Oracle**, care permite accesul la baza de date Oracle.

```
import cx_Oracle
```

IDE-ul utilizat este *PyCharm* unde pentru partea de front-end am folosit framework-ul **Flask** scris în Python. Acesta permite construirea de aplicații web. Un exemplu de linii de cod pentru folosirea Flask-ului:



App routing este folosit pentru a mapa adresa URL cu funcția asociată care este destinată pentru îndeplinirea anumitor task-uri. Aceasta la final va randa un fișier **html** pe baza căreia se generează un template pentru pagina noastră web.

III. Conectarea la baza de date

Un lucru important care trebuie făcut când se lucrează cu o bază de date și diverse librării Python, este crearea unei conexiuni între acestea. Multe modificări sunt realizate folosind conexiunea, care la finalizare este închisă.

Pentru conectare m-am folosit de cx_Oracle.

```
con = cx_Oracle.connect("TEMA_BD", "Maria26", "localhost/xe")
```

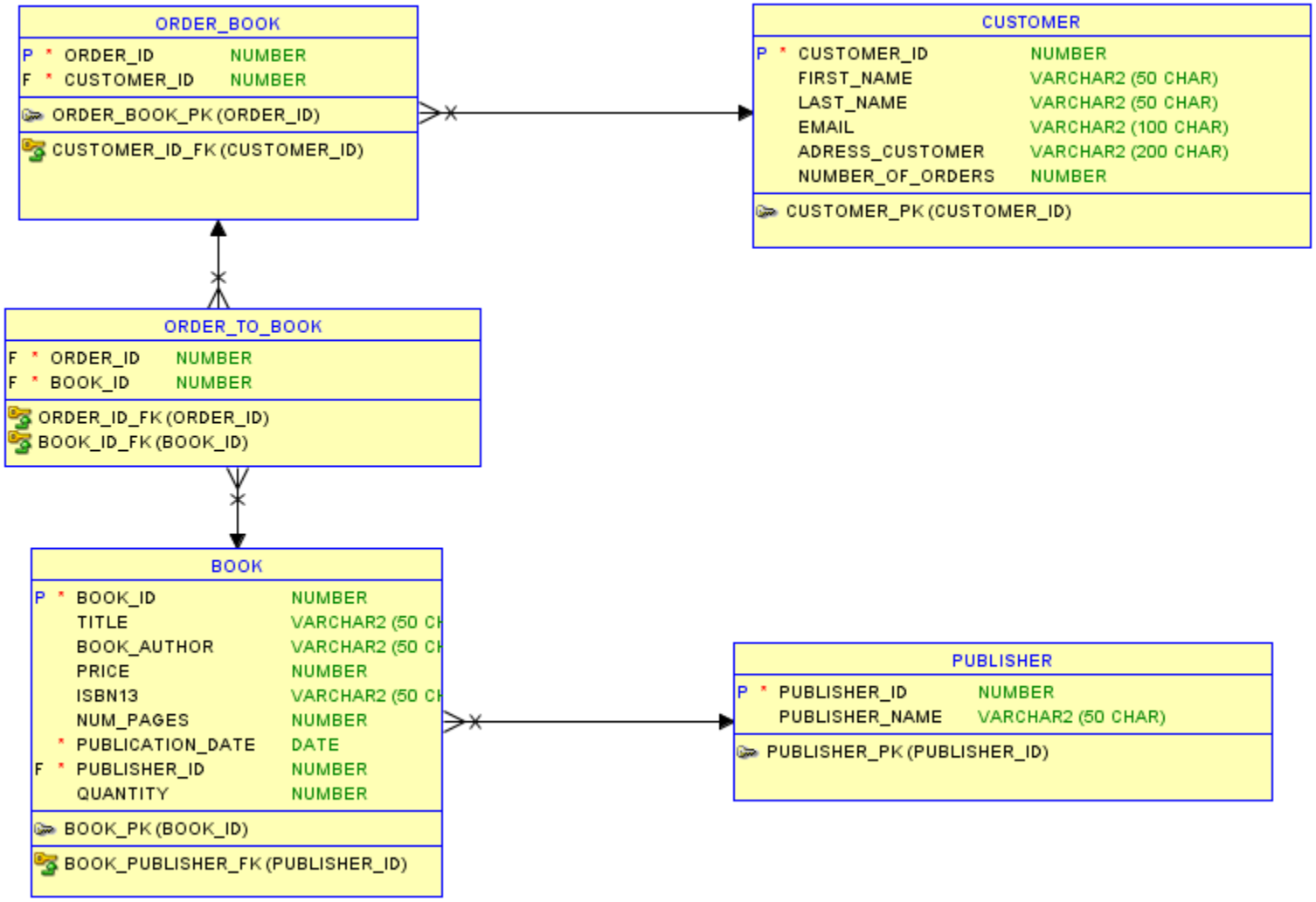
```
if __name__ == '__main__':
    app.run(debug=True)
    con.close()
```

În funcția asociată app routing-ului conexiunea la baza de date se face astfel:

```
@app.route('/publishers')
def publish():
    [redacted]

    sql=con.cursor()
    sql.execute('select * from publisher')

    [redacted]

    sql.close()
```

IV. Structura și inter-relaționarea tabelelor și descrierea constrângerilor

Diagrama ER:



Pentru identificare s-a optat includerea unei chei primare (*Primary key*) pentru fiecare tabelă, reprezentată de câmpul **ID**.

Inițial am creat tabela PUBLISHER care indică editurile cărților cu care colaborează magazinul.

Apoi am realizat tabela BOOK unde salvăm toate cărțile și informațiile lor. Această tabelă conține un *publisher_id* drept constrângere de integritate referențială (*Foreign Key*) deoarece ea se referă la coloana cu o cheie primară, *publisher_id* din tabela PUBLISHER.

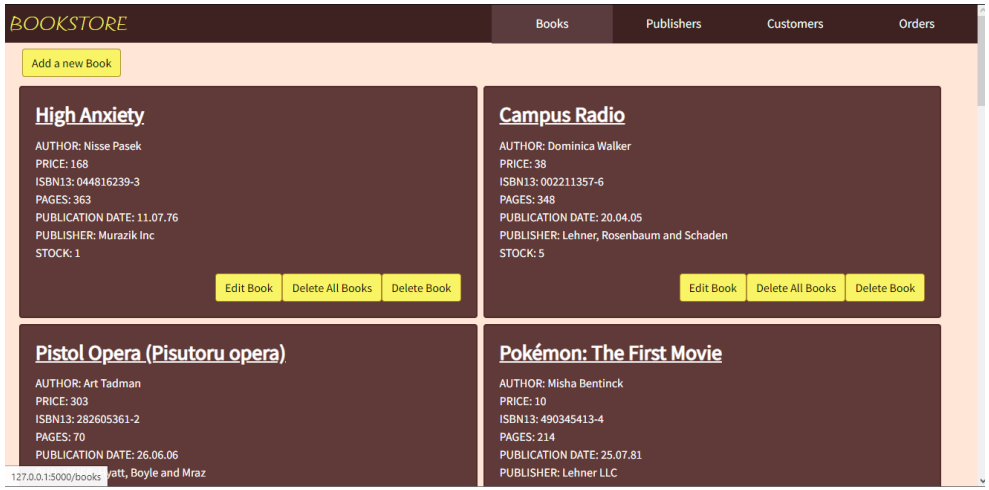
Tabela CUSTOMER salvează toți clienții și datele respective care se găsesc în diagrama de mai sus.

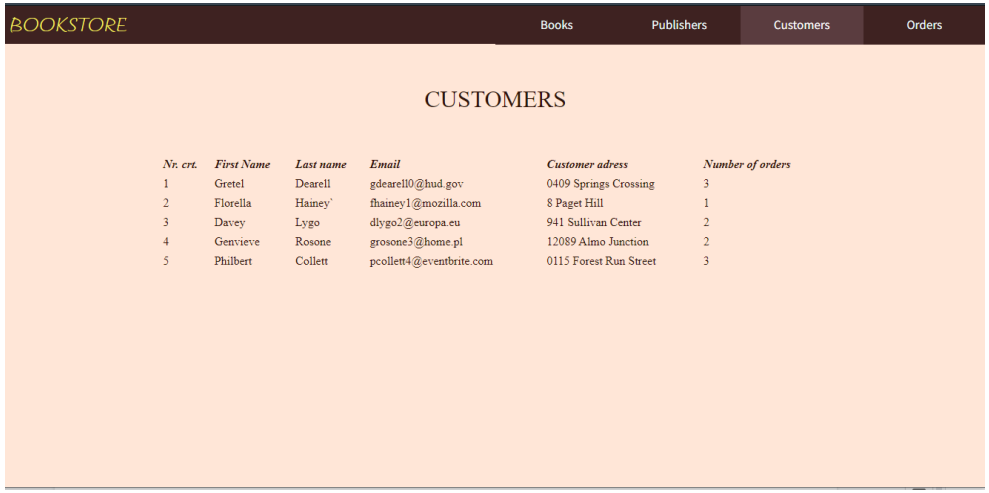
Tabela ORDER_BOOK stochează comenzile realizate de clienți. În aceasta se găsește o cheie externă (FK) *customer_id* care indică la cheia primă din tabela CUSTOMER.

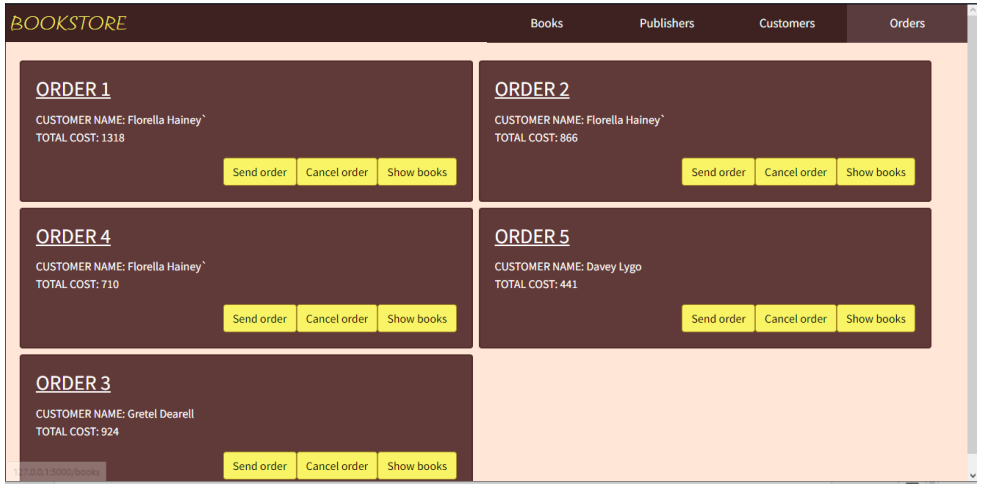
Tabela ORDER_TO_BOOK a fost creată pentru a face o legătură între tabela BOOK și ORDER_BOOK. Magazinul poate avea mai multe comenzi pe o singură carte și mai multe cărți pe o singură comandă. Atunci când o comandă este trimisă, acel *quantity* din tabela BOOK va fi decrementat cu numărul de cărți care au fost vândute.

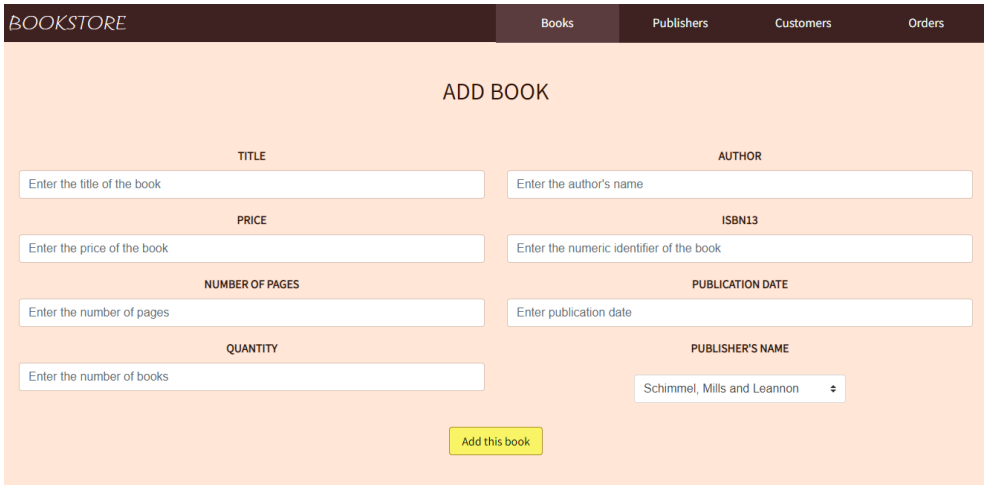
V. Capturi de ecran

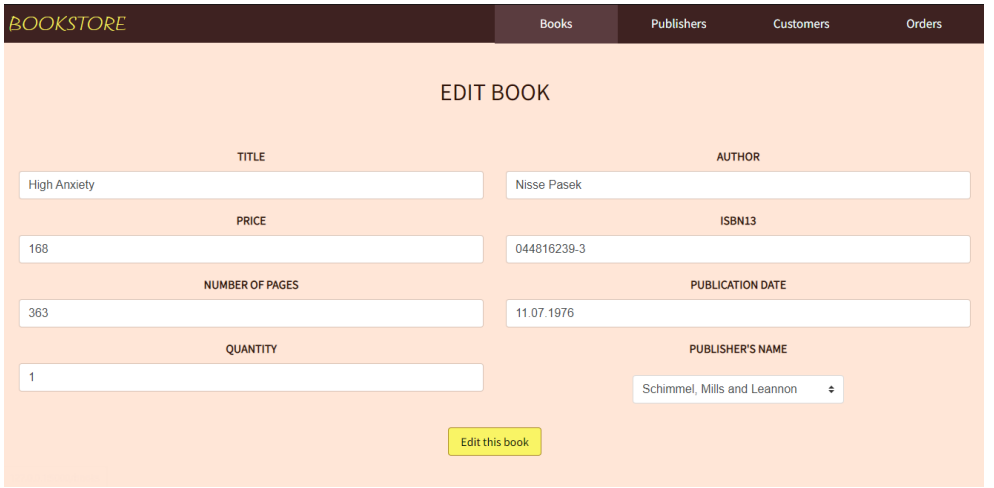
➔ INTERFAȚA GRAFICĂ

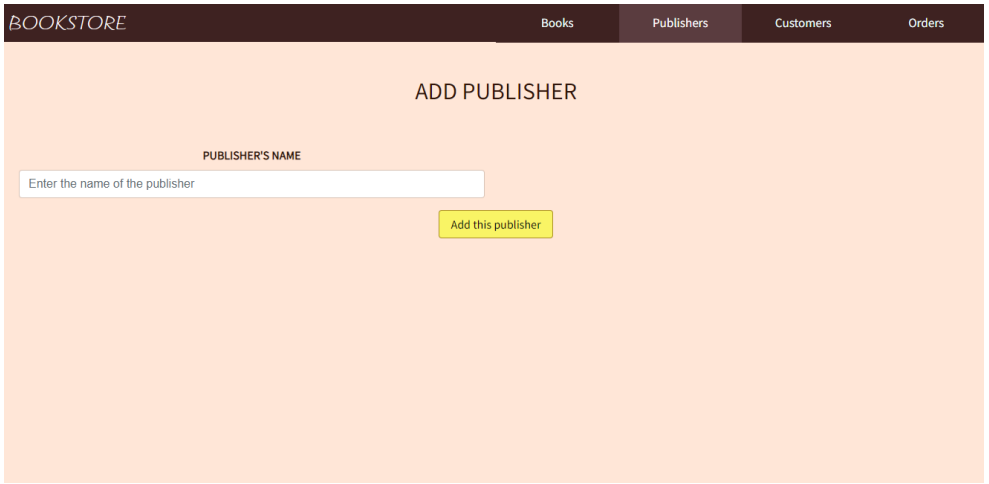


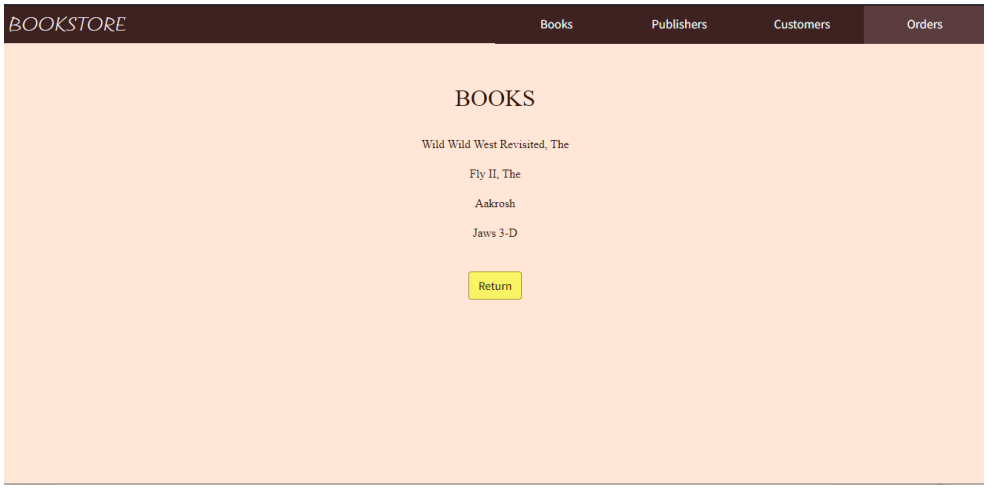





BOOKSTORE
Books
Publishers
Customers
Orders
ADD BOOK
TITLE
Enter the title of the book
AUTHOR
Enter the author's name
PRICE
Enter the price of the book
ISBN13
Enter the numeric identifier of the book
NUMBER OF PAGES
Enter the number of pages
PUBLICATION DATE
Enter publication date
QUANTITY
Enter the number of books
PUBLISHER'S NAME
Schimmel, Mills and Leannon
Add this book


BOOKSTORE
Books
Publishers
Customers
Orders
EDIT BOOK
TITLE
High Anxiety
AUTHOR
Nisse Pasek
PRICE
168
ISBN13
044816239-3
NUMBER OF PAGES
363
PUBLICATION DATE
11.07.1976
QUANTITY
1
PUBLISHER'S NAME
Schimmel, Mills and Leannon
Edit this book


BOOKSTORE
Books
Publishers
Customers
Orders
ADD PUBLISHER
PUBLISHER'S NAME
Enter the name of the publisher
Add this publisher


BOOKSTORE
Books
Publishers
Customers
Orders
BOOKS
Wild Wild West Revisited, The
Fly II, The
Aakrosh
Jaws 3-D
Return

➔ I MPLEMENTARE

main.py books.html

```python
from flask import Flask, render_template, jsonify, request, redirect
import cx_Oracle
from datetime import datetime
app = Flask(__name__)

con = cx_Oracle.connect("TEMA_BD", "Maria26", "localhost/xe")

@app.route("/")
@app.route('/books')
def books():
    books = []
    sql = con.cursor()
    sql.execute('SELECT b.book_id, b.title, b.book_author, b.price, b.ISBN13, b.num_pages, b.publication_date, p.publis

    for result in sql:
        book = {}
        book['book_id'] = result[0]
        book['title'] = result[1]
        book['book_author'] = result[2]
        book['price'] = result[3]
        book['ISBN13'] = result[4]
        book['num_pages'] = result[5]
        book['publication_date'] = datetime.strptime(str(result[6]), '%Y-%m-%d %H:%M:%S').strftime('%d.%m.%y')
        book['publisher_name'] = result[7]
        book['quantity'] = result[8]
        books.append(book)
```

```python
@app.route('/addBook', methods=['GET', 'POST'])
def add_book():
    error = None
    if request.method == 'POST':
        sql = con.cursor()
        sql.execute(f"insert into book (title,book_author,price,ISBN13,num_pages,publication_date,publisher_id,quantity)
        sql.execute('commit')
        return redirect('/books')
    else:
        publishers= []
        sql = con.cursor()
        sql.execute('select publisher_id, publisher_name from publisher')
        for result in sql:
            publishers.append((result[0],result[1]))
        sql.close()

        return render_template('addBook.html', publishers=publishers)


...
```

```python
@app.route('/delAllBooks', methods=['POST'])
def del_Allbook():
    bid = request.form['book_id']
    sql = con.cursor()
    sql.execute('delete from book where book_id=' + bid)
    sql.execute('commit')
    return redirect('/books')

@app.route('/delBooks', methods=['POST'])
def del_book():
    bid = request.form['book_id']
    sql = con.cursor()
    sql.execute('update book set quantity = quantity -1 where book_id=' + bid)
    sql.execute('commit')
    return redirect('/books')
```

```python
@app.route('/delPublishers', methods=['POST'])
def del_publishers():
    publish = request.form['publisher_id']
    sql = con.cursor()
    sql.execute('delete from book where publisher_id=' + publish)
    sql.execute('delete from publisher where publisher_id=' + publish)
    sql.execute('commit')
    return redirect('/publishers')


@app.route('/addPublisher', methods=['GET','POST'])
def add_publisher():
    error = None
    if request.method == 'POST':
        sql = con.cursor()
        sql.execute(f"insert into publisher (publisher_name) values (\'{request.form['publisher_name']}\')")
        sql.execute('commit')
        return redirect('/publishers')
    else:
        return render_template('addPublisher.html')
```