**Project 1**

**Section 1 - Setup**

**1) Heroku**

**a) Created account**

Username: markshou209@gmail.com

Password: Zeepcs50w$

**b) Created New App**

App name: bookreviewsbymark

**C) Configure Add-ons**

- added Heroku Progres

- Chose “Hobby Dev – Free” plan, Clicked “Provision” 🡪 “Heroku Postgres :: Database”

Settings 🡪 View Credentials

Please note that **these credentials are not permanent**.

Heroku rotates credentials periodically and updates applications where this database is attached.

**Host** ec2-107-21-125-209.compute-1.amazonaws.com

**Database** dcmoh597l9rarv

**User** oydqwwvmfkwawe

**Port** 5432

**Password**

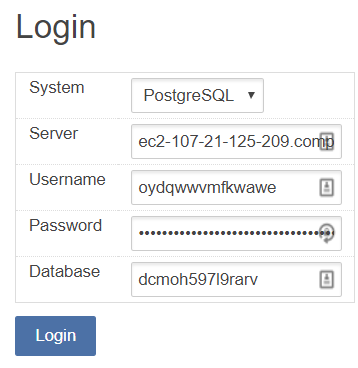
3cb6a5a0d105c929fc4695244b25dfbb1458befabddb2b3c85d254549213b656

**URI** postgres://oydqwwvmfkwawe:3cb6a5a0d105c929fc4695244b25dfbb1458befabddb2b3c85d254549213b656@ec2-107-21-125-209.compute-1.amazonaws.com:5432/dcmoh597l9rarv

**Heroku CLI** heroku pg:psql postgresql-spherical-60683 --app bookreviewsbymark

**d) Log in via Adminer**

<https://adminer.cs50.net/>



- View database structure

**2) Python & Flask**

**a) Python & pip already installed**

**b) Install python library requirements (Flask, SQLAlchemy)**

$ pip3 install -r requirements.txt

**c) Set environment variables**

$ export FLASK\_APP=application.py

$ export FLASK\_ENV=development

$ export DATABASE\_URL="postgres://oydqwwvmfkwawe:3cb6a5a0d105c929fc4695244b25dfbb1458befabddb2b3c85d254549213b656@ec2-107-21-125-209.compute-1.amazonaws.com:5432/dcmoh597l9rarv"

$ echo $DATABASE\_URL

postgres://oydqwwvmfkwawe:3cb6a5a0d105c929fc4695244b25dfbb1458befabddb2b3c85d254549213b656@ec2-107-21-125-209.compute-1.amazonaws.com:5432/dcmoh597l9rarv

\*\*\* Above environment variables can be run as a shell script (environment.sh)

$ source environment.sh

Setting FLASK\_APP to: application.py

Setting DATABSE\_URL to: development

Setting DATABSE\_URL to: postgres://oydqwwvmfkwawe:3cb6a5a0d105c929fc4695244b25dfbb1458befabddb2b3c85d254549213b656@ec2-107-21-125-209.compute-1.amazonaws.com:5432/dcmoh597l9rarv

**d) Run Flask server**

$ flask run

\* Serving Flask app "application.py" (lazy loading)

\* Environment: development

\* Debug mode: on

\* Restarting with stat

\* Debugger is active!

\* Debugger PIN: 308-841-801

\* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

<http://127.0.0.1:5000/> or <http://localhost:5000/>

Project 1: TODO

**3) Goodreads API**

**a) Sign up for Goodreads account**

[www.goodreads.com](http://www.goodreads.com)

username: markshou209@gmail.com

password: Zeepcs50w$

**b) Apply for API key**

<https://www.goodreads.com/api/keys>

Application Name: project1

Company Name: project1

**key:**r1E4UhH6xREe17fpAqAu1g  
**secret:**5zRCaREyy6BGiGoaO8u9R5PVH4NGv10SSPSjYlXM

**c) Test API**

**goodreads\_api.py**

import requests

res = requests.get("https://www.goodreads.com/book/review\_counts.json",

params={"key": "r1E4UhH6xREe17fpAqAu1g", "isbns": "9781632168146"})

print(res.json())

$ python goodreads\_api.py

{'books': [{'id': 29207858, 'isbn': '1632168146', 'isbn13': '9781632168146', 'ratings\_count': 0, 'reviews\_count': 2, 'text\_reviews\_count': 0, 'work\_ratings\_count': 26, 'work\_reviews\_count': 118, 'work\_text\_reviews\_count': 10, 'average\_rating': '4.04'}]}

**Section 2 – Book Review Website**

**0) Files**

a) environment.sh – bash file to set the DATABASE\_URL (postgres server/database link), FLASK\_APP (application.py), and FLASK\_ENV (development)

a) requirements.txt – all the python libraries required for the main python code (application.py)

b) create\_tables.sql – SQL code ran in database to create initial tables (accounts, books, reviews)

c) import.py – python code to import all the values books.csv into our SQL books table (isbn, title, author, year)

d) application.py – main python code that flask runs to serve our website

e) HTML pages:

- layout.html: main css page that is replicated and extended by other pages

- index.html: user login page

- register.html: user registration page

- search.html: main search page after logging in

- book.html: individual book info page where isbn, title, year, author, blurb, goodreads rating/# ratings is displayed. This page also displays book reviews left in our database and allows for current user to leave 1 rating/review.

- 404.html: page not found error page

**1) Create Tables**

a) Log into PostgreSQL database

$ psql postgres://oydqwwvmfkwawe:3cb6a5a0d105c929fc4695244b25dfbb1458befabddb2b3c85d254549213b656@ec2-107-21-125-209.compute-1.amazonaws.com:5432/dcmoh597l9rarv

b) run create\_tables.sql

- this creates 3 tables: “accounts”, “books”, and “reviews”

dcmoh597l9rarv=> \i create\_tables.sql

DROP TABLE IF EXISTS "accounts" CASCADE;

DROP SEQUENCE IF EXISTS accounts\_id\_seq CASCADE;

CREATE TABLE "accounts" (

"id" SERIAL NOT NULL,

"username" VARCHAR PRIMARY KEY NOT NULL,

"password" VARCHAR NOT NULL,

)

DROP TABLE IF EXISTS "books" CASCADE;

DROP SEQUENCE IF EXISTS books\_id\_seq CASCADE;

CREATE TABLE "books" (

"id" SERIAL NOT NULL,

"isbn" VARCHAR PRIMARY KEY NOT NULL,

"title" VARCHAR NOT NULL,

"author" VARCHAR NOT NULL,

"year" INTEGER NOT NULL,

)

DROP TABLE IF EXISTS "reviews" CASCADE;

DROP SEQUENCE IF EXISTS reviews\_id\_seq CASCADE;

CREATE TABLE "reviews" (

"id" SERIAL PRIMARY KEY NOT NULL,

"acc\_id" VARCHAR NOT NULL,

"book\_id" VARCHAR NOT NULL,

"comment" VARCHAR NOT NULL,

"rating" FLOAT NOT NULL,

"date" timestamp DEFAULT now() NOT NULL,

)

**2) Import**

**a) Python script to import rows from books.csv into the “books” table with columns “isbn”, “title”, “author”, and “year”.**

import csv

import os

from sqlalchemy import create\_engine

from sqlalchemy.orm import scoped\_session, sessionmaker

# engine = create\_engine("postgres://oydqwwvmfkwawe:3cb6a5a0d105c929fc4695244b25dfbb1458befabddb2b3c85d254549213b656@ec2-107-21-125-209.compute-1.amazonaws.com:5432/dcmoh597l9rarv")

engine = create\_engine(os.getenv("DATABASE\_URL"))

db = scoped\_session(sessionmaker(bind=engine))

def main():

f = open("books.csv")

reader = csv.reader(f)

# INSERT rows into TABLE "books"

for isbn, title, author, year in reader:

if year != "year":

db.execute("INSERT INTO books (isbn, title, author, year) VALUES

(:isbn, :title, :author, :year)", {"isbn": isbn, "title":

title, "author": author, "year": year})

print(f"Added \"{title}\" to database.")

db.commit()

if \_\_name\_\_ == "\_\_main\_\_":

main()

$ python import.py

…

Added "Starter for Ten" to database.

Added "Imajica" to database.

Added "Emily Climbs" to database.

**b) View new tables in database**

dcmoh597l9rarv=> \d

List of relations

Schema | Name | Type | Owner

--------+-----------------+----------+----------------

public | accounts | table | oydqwwvmfkwawe

public | accounts\_id\_seq | sequence | oydqwwvmfkwawe

public | books | table | oydqwwvmfkwawe

public | books\_id\_seq | sequence | oydqwwvmfkwawe

public | reviews | table | oydqwwvmfkwawe

public | reviews\_id\_seq | sequence | oydqwwvmfkwawe

(6 rows)

dcmoh597l9rarv=> SELECT \* FROM books LIMIT 5;

isbn | title | author | year

------------+-----------------------+------------------+------

0380795272 | Krondor: The Betrayal | Raymond E. Feist | 1998

1416949658 | The Dark Is Rising | Susan Cooper | 1973

1857231082 | The Black Unicorn | Terry Brooks | 1987

0553803700 | I, Robot | Isaac Asimov | 1950

080213825X | Four Blondes | Candace Bushnell | 2000

(5 rows)

**3) Requirements.txt**

- this file contains all the python libraries used in application.py

Flask

Flask-Session

psycopg2

SQLAlchemy

os

re

requests

flask\_bcrypt

bs4

**4) User Registration**

a) In application.py, @app.route(“/”) renders the index.html page which contains code for user registration

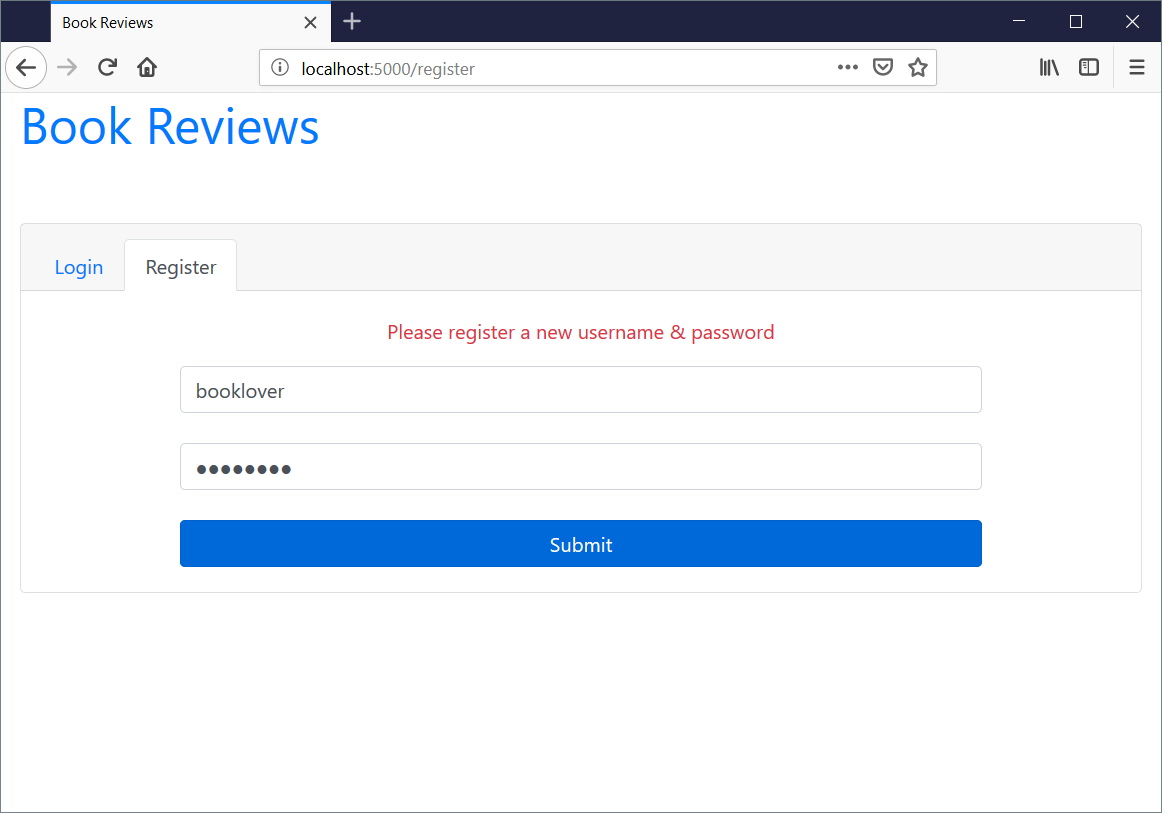
@app.route("/")

def index():

return render\_template("index.html")

**index.html**

- start page with bootstrap css Navigation card template for user Login/Register



from flask\_bcrypt import Bcrypt

@app.route("/signup", methods=["POST"])

def signup():

username = request.form.get("username")

password = request.form.get("password")

passw\_encrypted = bcrypt.generate\_password\_hash(password).decode('utf-8')

if (username is None) or (password is None) or (username == "") or (password is ""):

return render\_template("register.html", message="Please enter a username and password")

elif not username.isalnum() or not len(username) >= 6 or not len(username) <= 30:

return render\_template("register.html", message="Username must be alphanumeric AND between 6 and 30 characters")

elif db.execute("SELECT \* FROM accounts WHERE username = :username", {"username": username}).fetchone() is None:

db.execute("INSERT INTO accounts (username, password) VALUES (:username, :password)", {"username": username, "password": passw\_encrypted})

db.commit()

return render\_template("index.html", message="New user registered. Please sign in below.")

else:

return render\_template("register.html", message="Username already exists. Please choose another.")

- note that when registering, the password will be stored in encrypted hash format using the flask\_bcrypt library.

dcmoh597l9rarv=> select \* from accounts;

id | username | password

----+-----------+--------------------------------------------------------------

51 | booklover | $2b$12$Dn1hwCAGcoCGT4/siVR8uOM9Ka7HZ4S1MUpqBWYCGAgGvgrdaF3wq

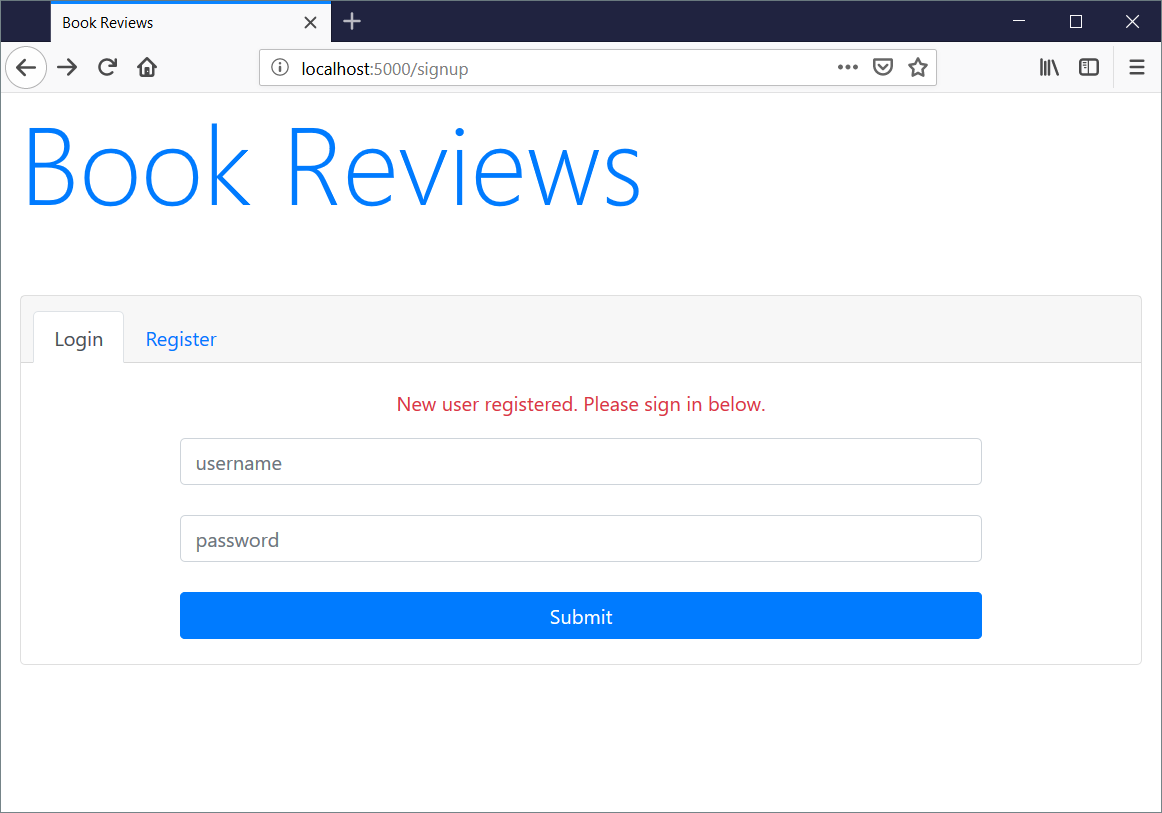
(1 row)

- username “booklover”, password “password” generates a hashed password which is then stored in the database. This is for database security. We do not want to store password as plaintext in the database.

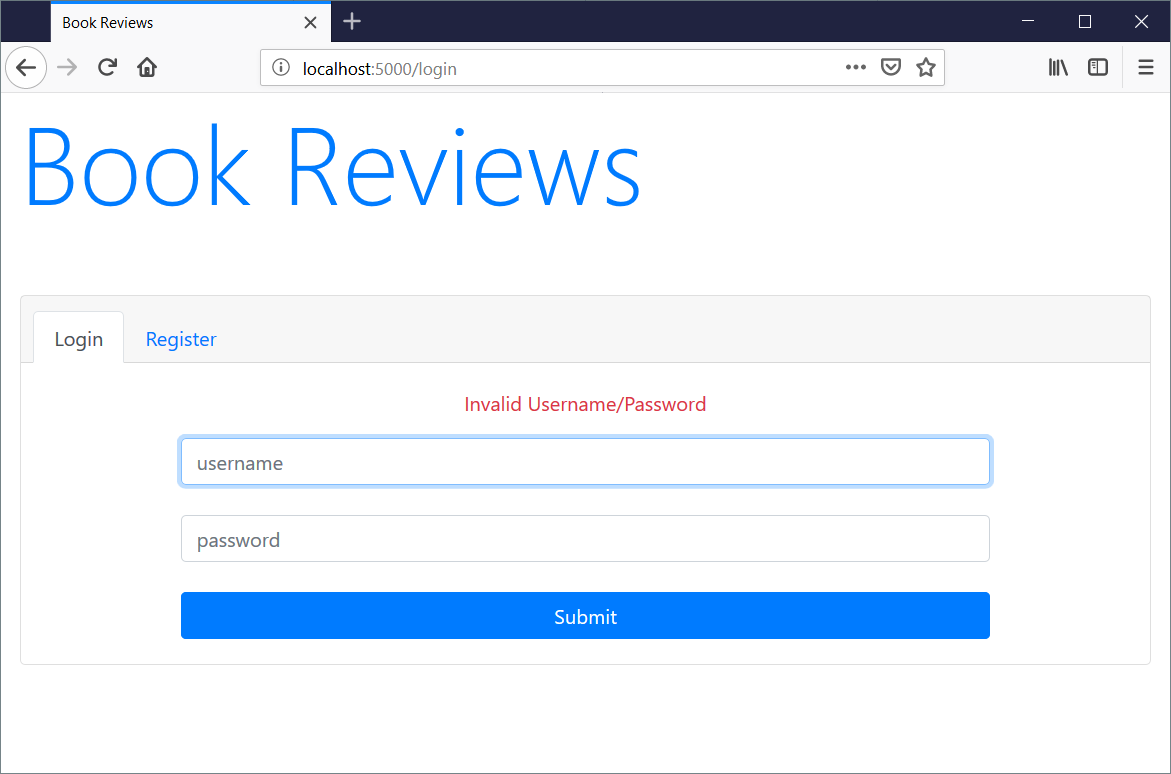
**5) User Login/Logout**

a) login page

- user can log in from the root page “/” or upon registration.



- After clicking “Submit” on the Register tab, the user will be taken to the Login tab to log in.



- if either the username or password is wrong, an error message will be displayed: “Invalid Username/Password”

user = db.execute("SELECT \* FROM accounts WHERE username = :username", {"username": username}).fetchone()

if user is not None:

if bcrypt.check\_password\_hash(user['password'], password) is True:

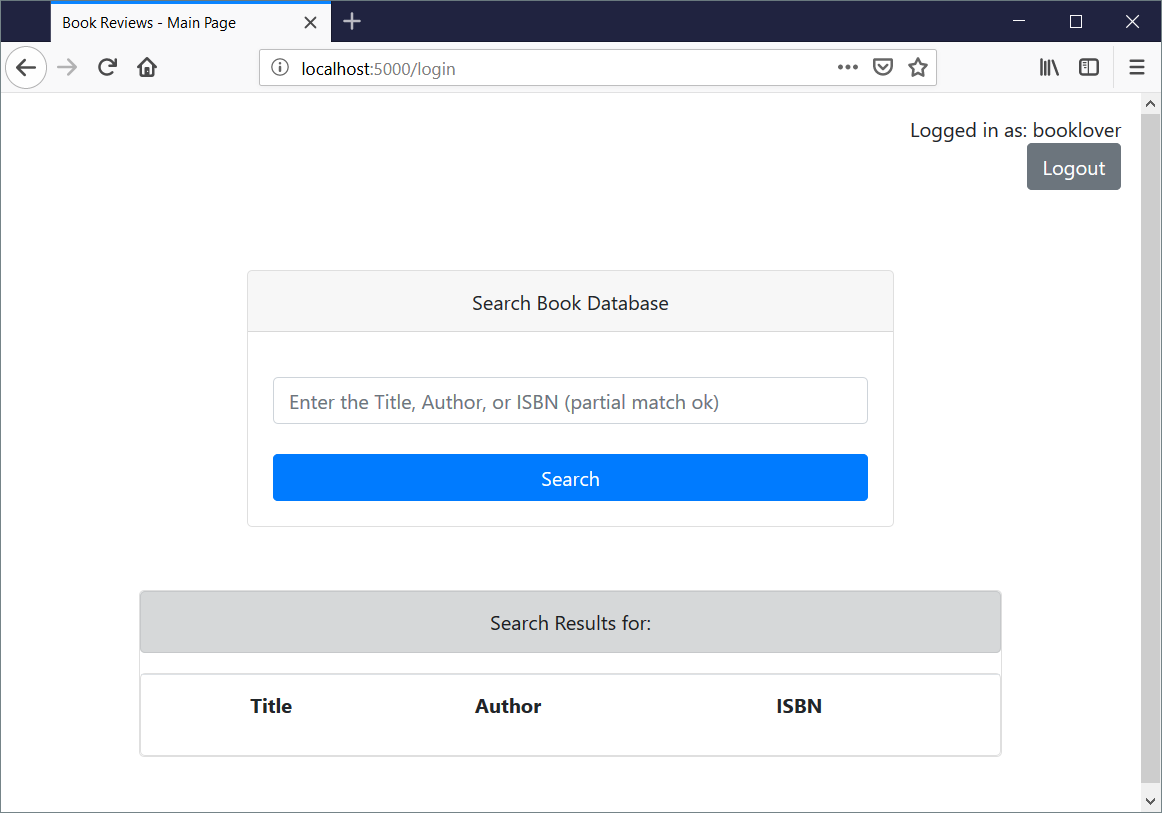
session["user\_id"] = user['id']

session["username"] = user['username']

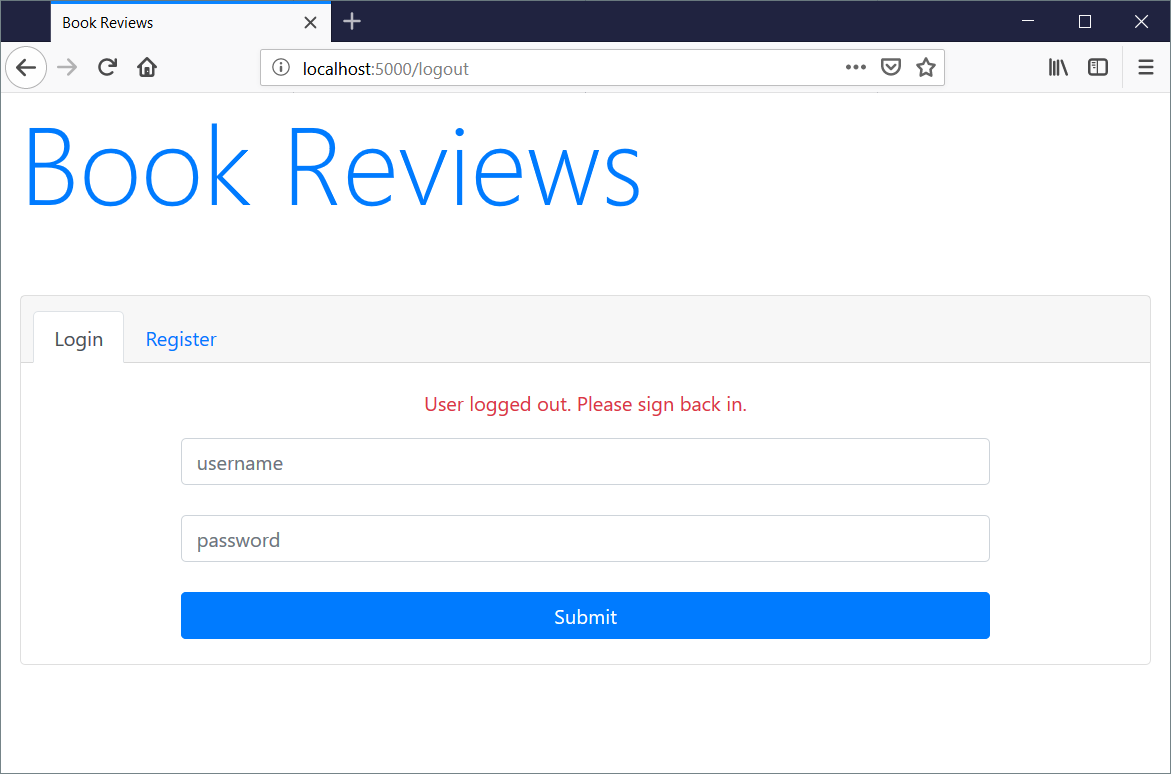
return render\_template("search.html")

return render\_template("index.html", message="Invalid Username/Password")

- user’s password input will be decrypted via bcrypt before checking against stored value in database (stored as hash). If it is directly checked against the database, an error will result.



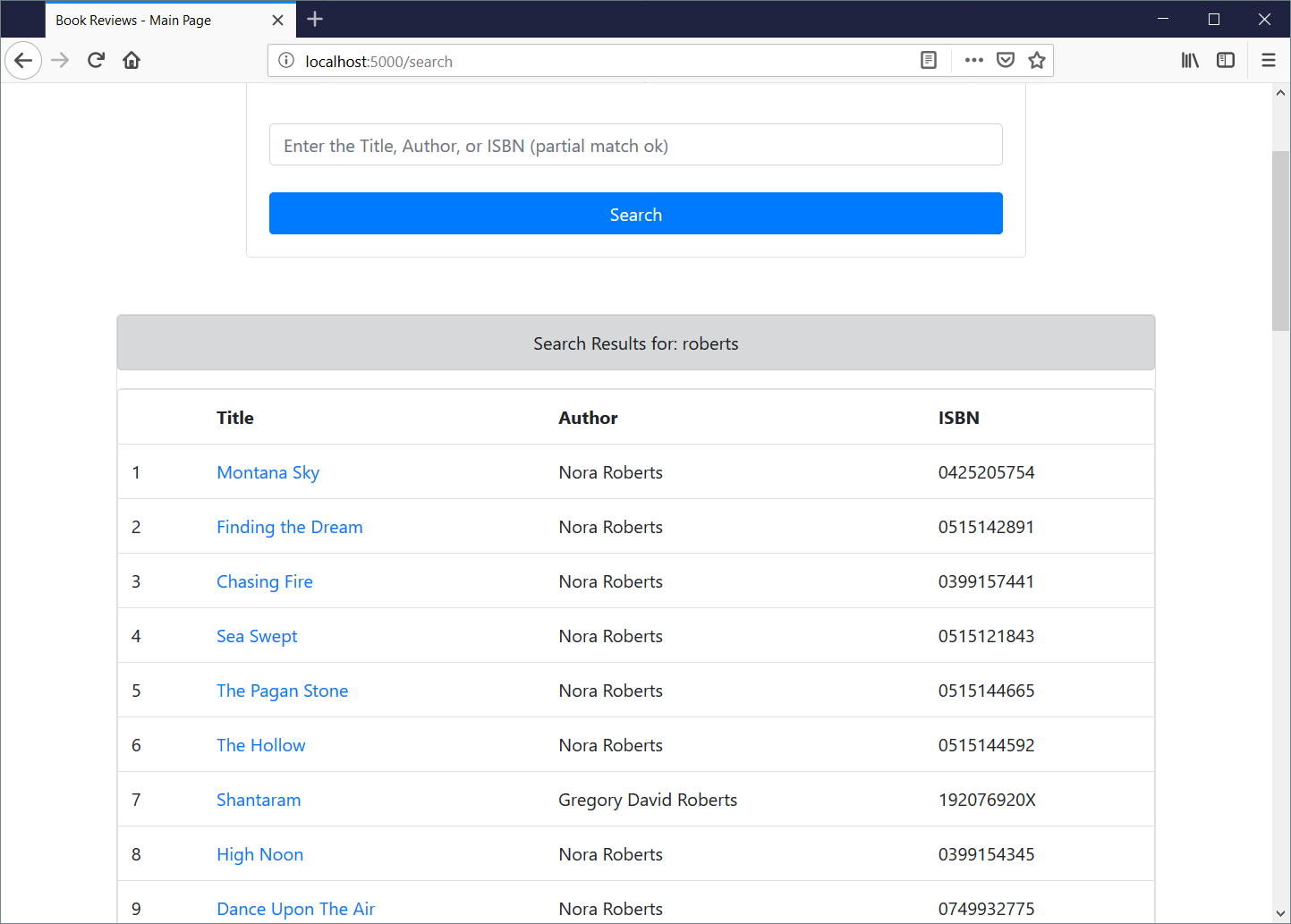
- If the username/password is correct, the user will be taken to the main search page. Note that the user will be assigned a session id, and anything the user does (i.e. ratings & commenting) is logged under his/her user id (ex. “booklover”)



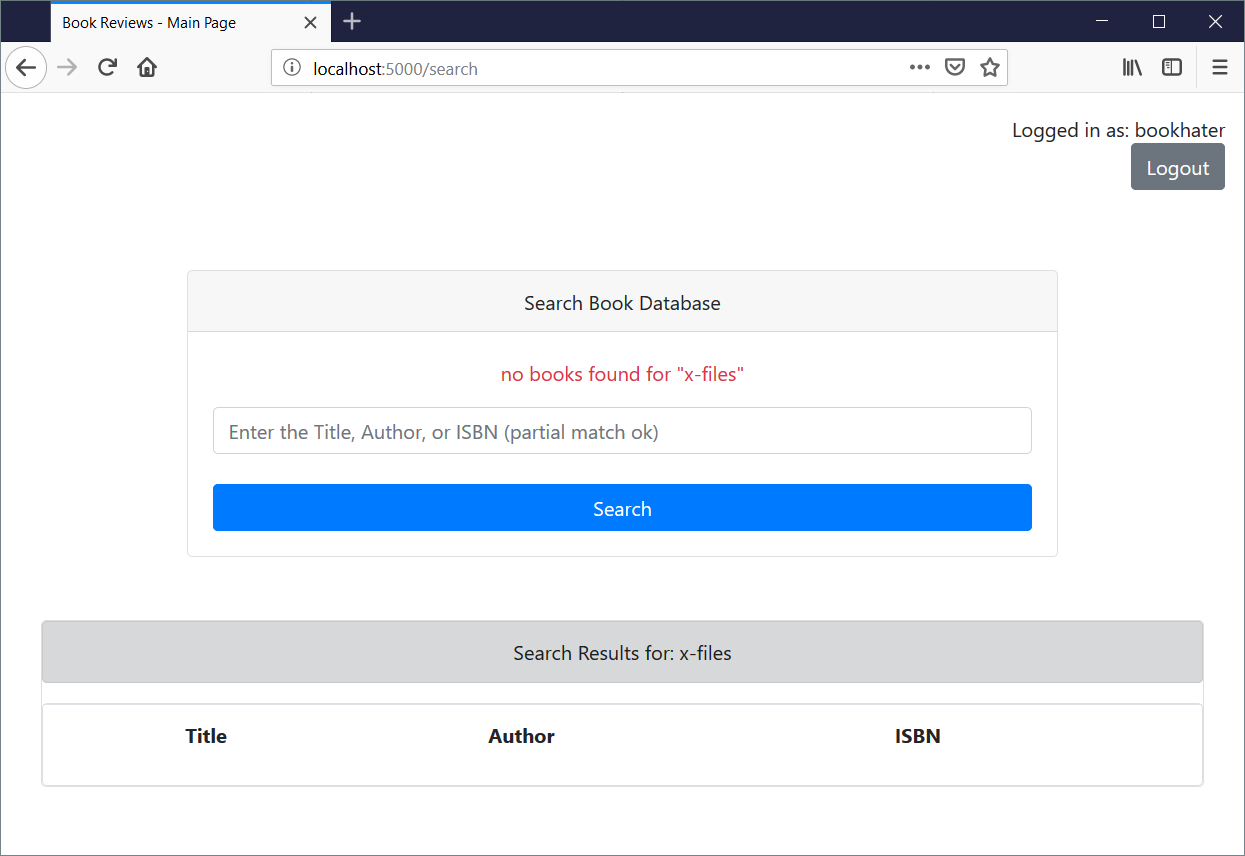
- clicking on “Logout” will log the user out of the page and display a message: “User logged out. Please sign back in.”

**6) Search**

- User can search by Title, Author, or ISBN. Note that search is case insensitive and will display partial matches for searches.



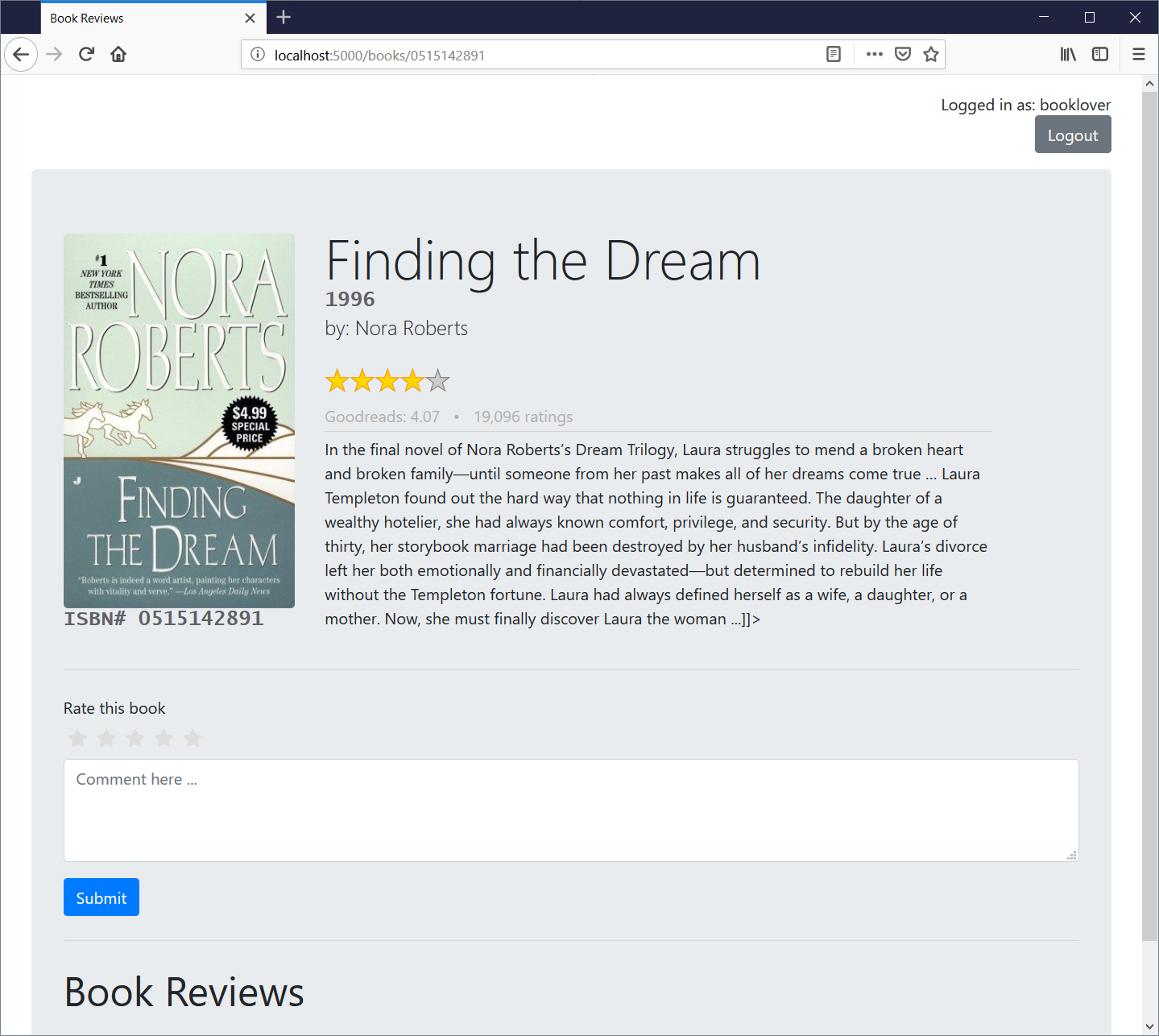
- For example, a search for “roberts” will display results for author “Nora Roberts”, “Gregory David Roberts”, or any title that contains the text “Roberts”. The titles displayed are hotlinks that will take the user to that book’s individual webpage.



- a search for “x-files” returns no results, and an error message is displayed.

**7) Book Page / Goodreads Review Data**

- the user can get to the individual book page via the search page links, or directly typing the ISBN code into the URL, for example, …/books/0515142891



- User can click on a book title link and will be taken to the book page, which is created on the fly via flask. It uses the book.html template and feeds in information: title, year, author, Goodreads rating, Goodreads # of ratings, Goodreads blurb, book cover image from openlibrary.org, and the list of Book Reviews for that book in our database.

- The Goodreads Star rating image is created using CSS and is by percentage (it will display partial star)

- The middle section is for the user to review the book

- The bottom section is to display existing reviews in our database (currently there are none).

# Good Reads Rating

response = requests.get("https://www.goodreads.com/book/review\_counts.json", params={"key": "r1E4UhH6xREe17fpAqAu1g", "isbns": isbn})

data = response.json()

gr\_rating = (data['books'][0]['average\_rating'])

gr\_rating\_pct = "{0:.0%}".format(float(gr\_rating)/5.0)

gr\_ratingcount = (data['books'][0]['work\_ratings\_count'])

gr\_ratingcountcommas = "{:,}".format(gr\_ratingcount)

# Good Reads Book Blurb

xml\_page = requests.get("https://www.goodreads.com/book/isbn/", params={"key": "r1E4UhH6xREe17fpAqAu1g", "isbn": isbn})

soup = BeautifulSoup(xml\_page.text, features='lxml')

description = soup.find\_all('description')[0]

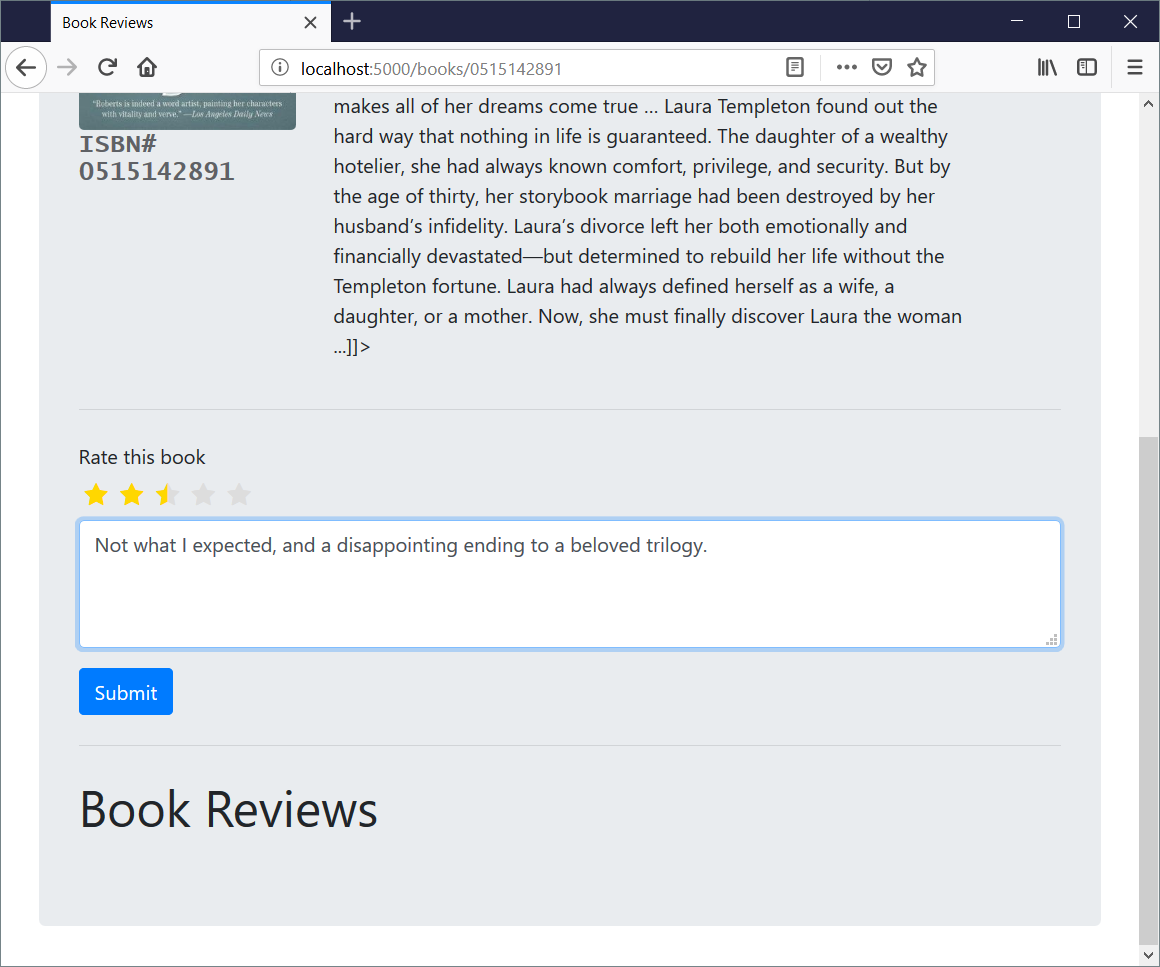
blurb = description.get\_text()

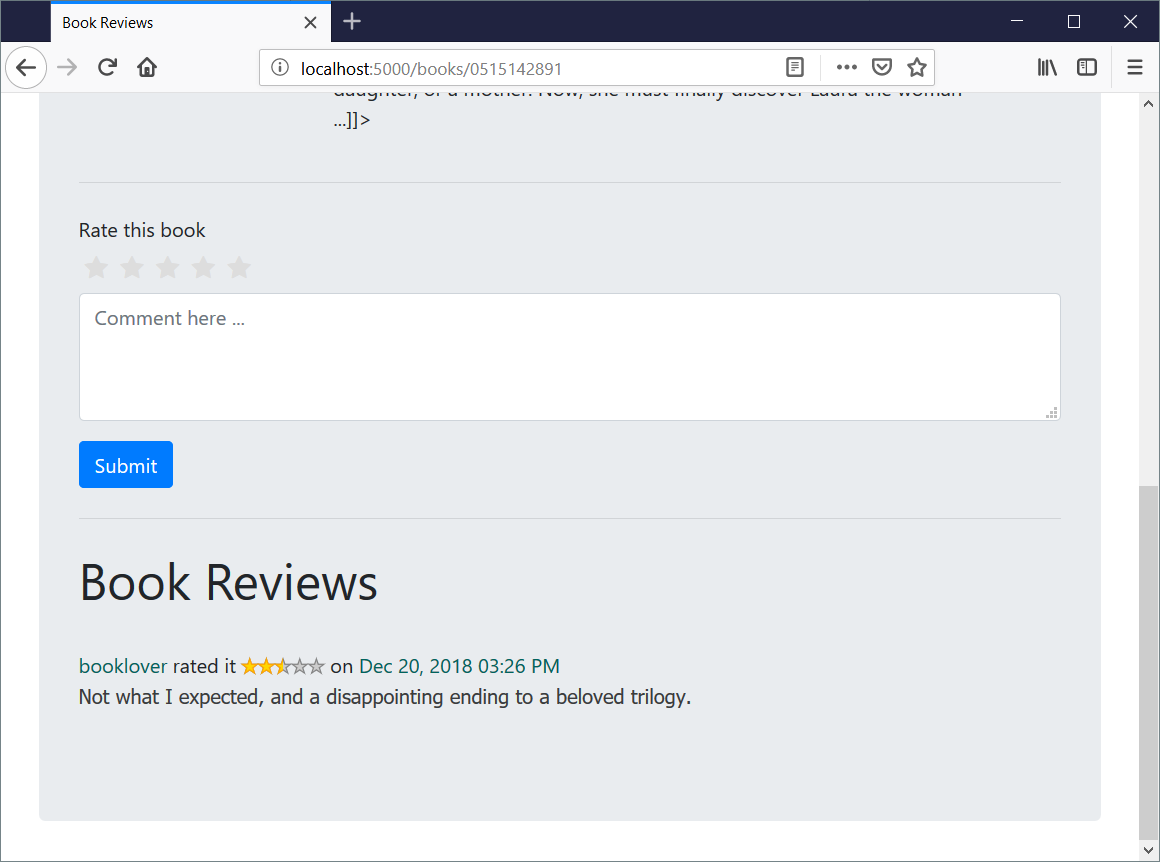
return render\_template("book.html", book\_info=book, reviews=reviews, reviewtime=reviewtime, gr\_rating\_pct=gr\_rating\_pct, rating=gr\_rating, rcount=gr\_ratingcountcommas, blurb=blurb, comment\_error=comment\_error)

- Goodreads ratings is obtained from the Goodreason API. The key is specific to my Goodreads account and is written into the page request. The data is in the form of json or xml, and is parsed out into variables and passed into the book.html page for display

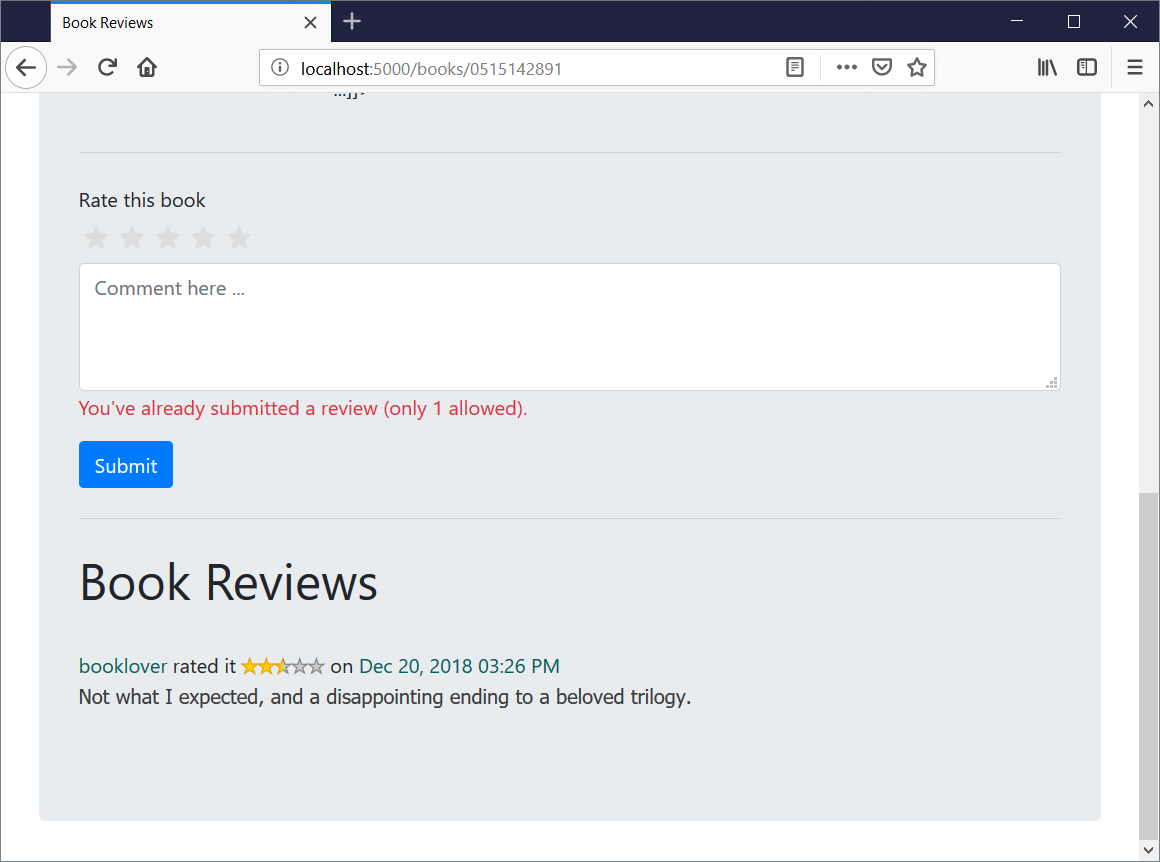
**8) Review Submission**

- the user can rate the book on a star scale (1-5 stars) in ½ star increments. The rating and comment fields are required before the page will allow for submission.

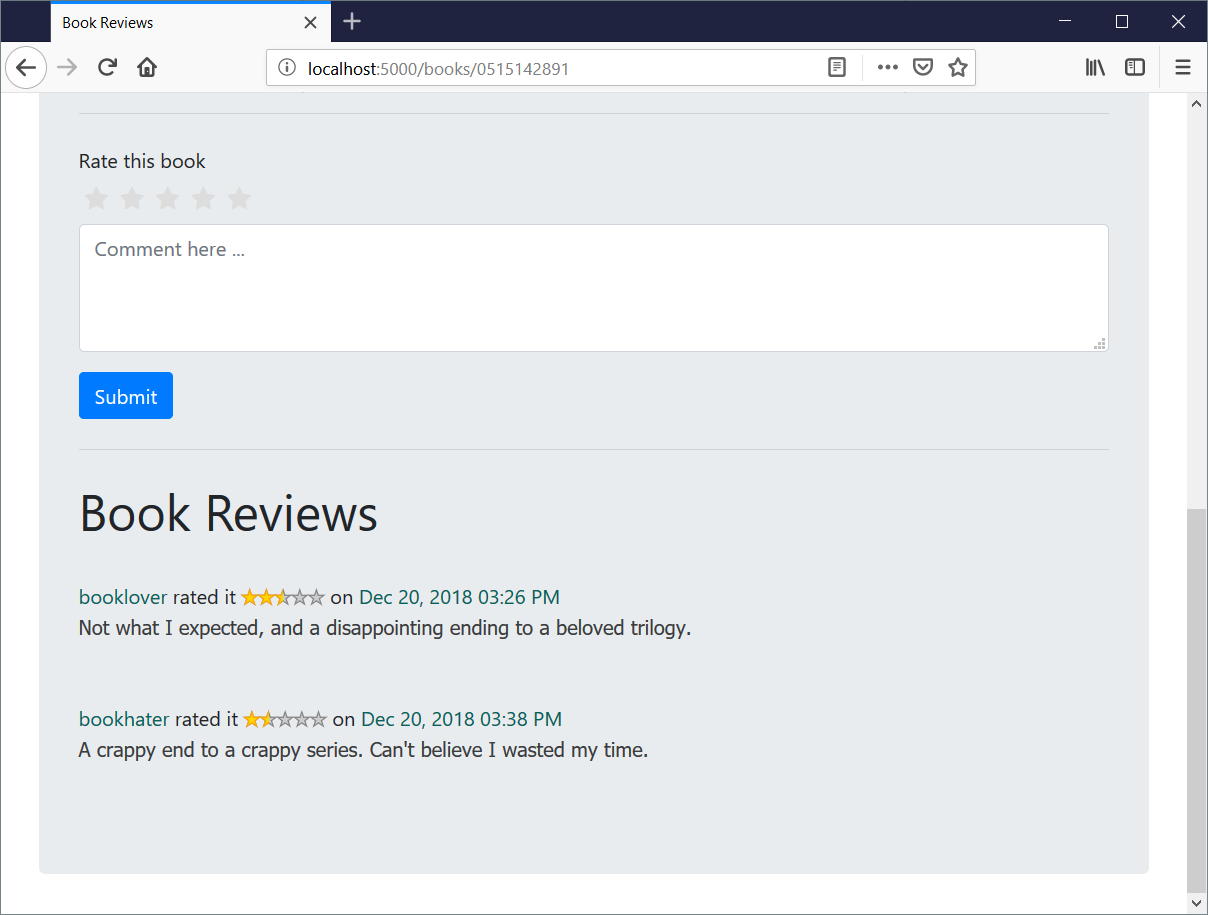




- After submission, the Book Review will be displayed below in chronological order.



- If the same user tries to submit another review, the page will display an error: “You’ve already submitted a review (only 1 allowed).



- after registering a new user and logging in, the new user is able to leave another review on the page.

dcmoh597l9rarv=> select \* from reviews;

id | acc\_id | book\_id | comment | rating | date

----+-----------+------------+----------------------------+--------+---------------------------

49 | booklover | 0515142891 | Not what I expected, and a | 2.5 | 2018-12-20 15:26:41.081455

disappointing ending to a

beloved trilogy.

50 | bookhater | 0515142891 | A crappy end to a crappy | 1.5 | 2018-12-20 15:38:02.64063

series. Can't believe I

wasted my time.

(2 rows)

- these are the reviews entered in the database

**9) API Access**

- when the user makes a GET request to my website’s /api/<isbn> route, the website returns a JSON response containing the book’s title, author, publication date, ISBN number, review count, and average score.

@app.route("/api/<string:isbn>", methods=["GET", "POST"])

def bookapi(isbn):

book = db.execute("SELECT \* FROM books WHERE isbn = :isbn", {"isbn": isbn}).fetchone()

reviews = db.execute("SELECT COUNT(comment) FROM reviews WHERE book\_id = :isbn", {"isbn": isbn}).fetchone()

avg\_rating = db.execute("SELECT AVG(rating) FROM reviews WHERE book\_id = :isbn", {"isbn": isbn}).fetchone()

if not book:

abort(404)

return jsonify({

"title": book.title,

"author": book.author,

"year": book.year,

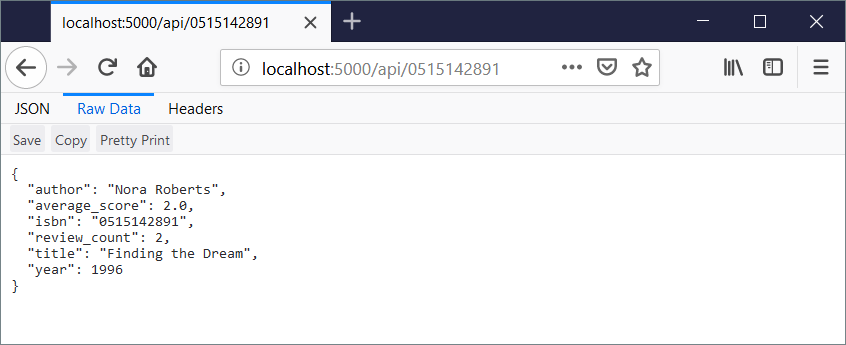
"isbn": book.isbn,

"review\_count": reviews.count,

"average\_score": avg\_rating.avg

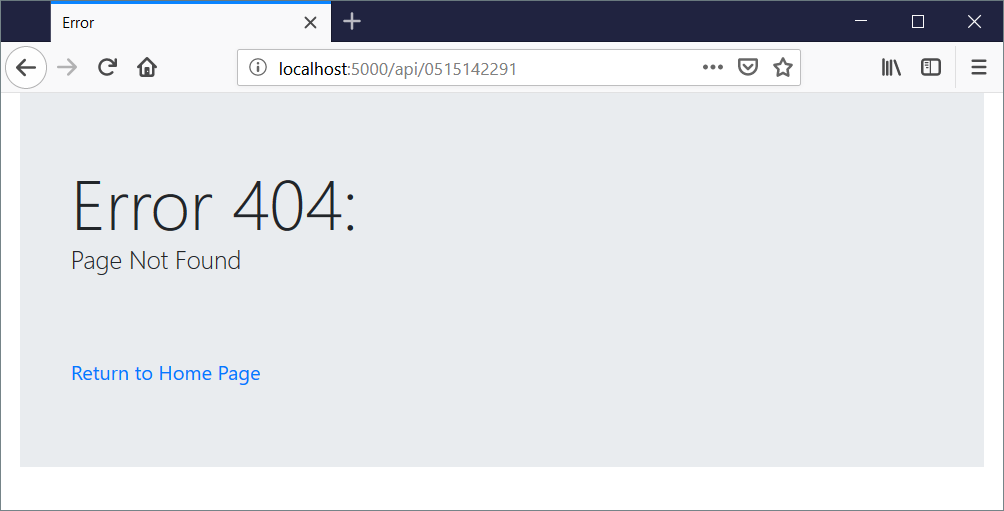
})

- note that the review\_count and average\_score are values from our created database, not Goodreads data.



**10) 404 Error**

- if the API request is for an ISBN that does not exist in our database, the website returns a 404 error.



**Section 3 – Deploying Website**

**1) Push files onto Github**