CP476 Project Report

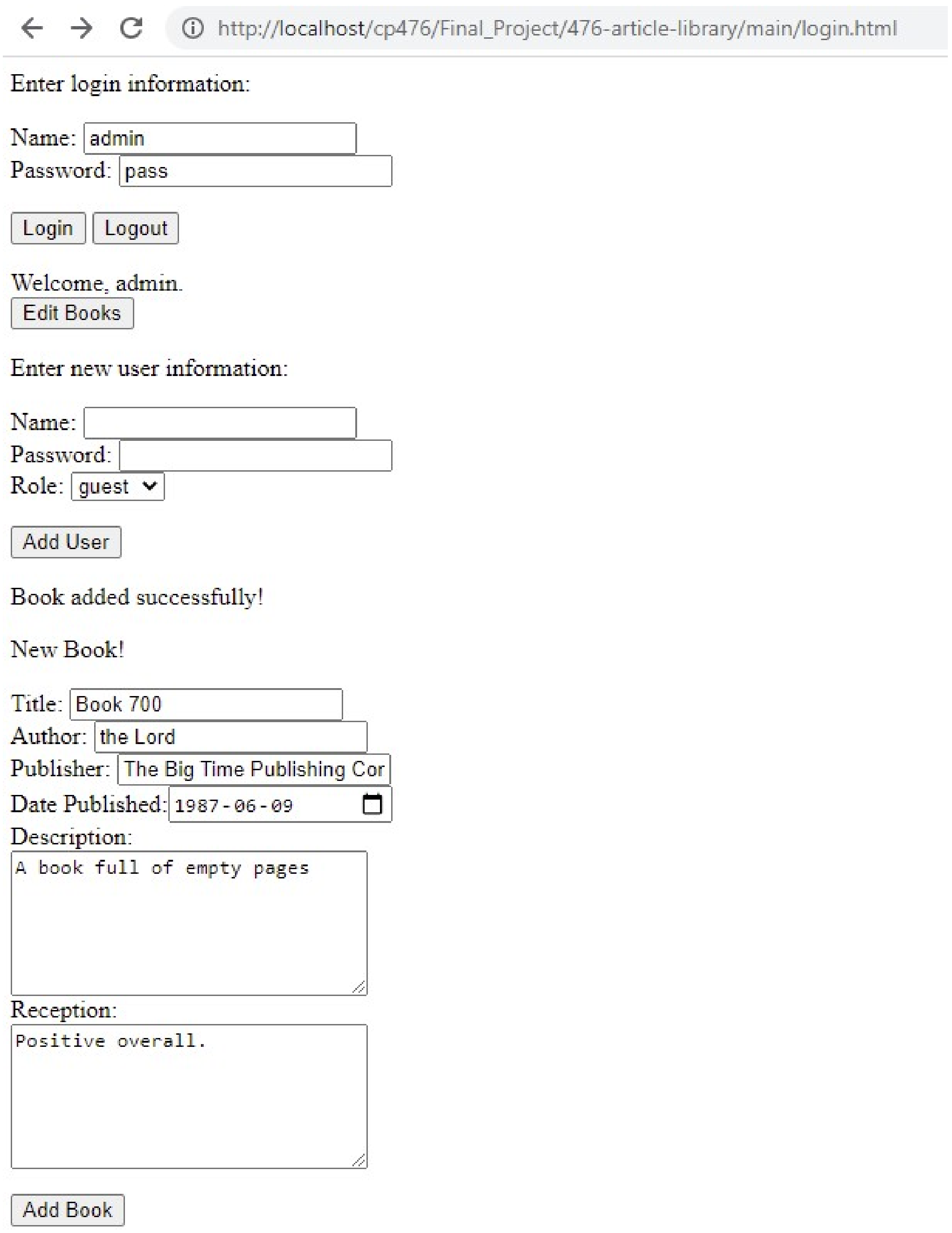
Authors

Michael Child-Wynne-Jones, Morgenne Besenschek, Chang Xing Li

Proof of Accomplishment: System design & implementation

P2.1 Client side component & UI [25/30/*]

While our UI is not stylish, it is responsive and robust. We use JQuery to create dynamic, responsive webpages, which are tailored to a user's permissions and which use cookies to keep users logged in for 30 minutes.



http://localhost/cp476/Final_Project/476-article-library/main/login.html

Enter login information:

Name: admin
Password: pass

Login Logout

Welcome, admin.
Edit Books

Enter new user information:

Name:
Password:
Role: guest

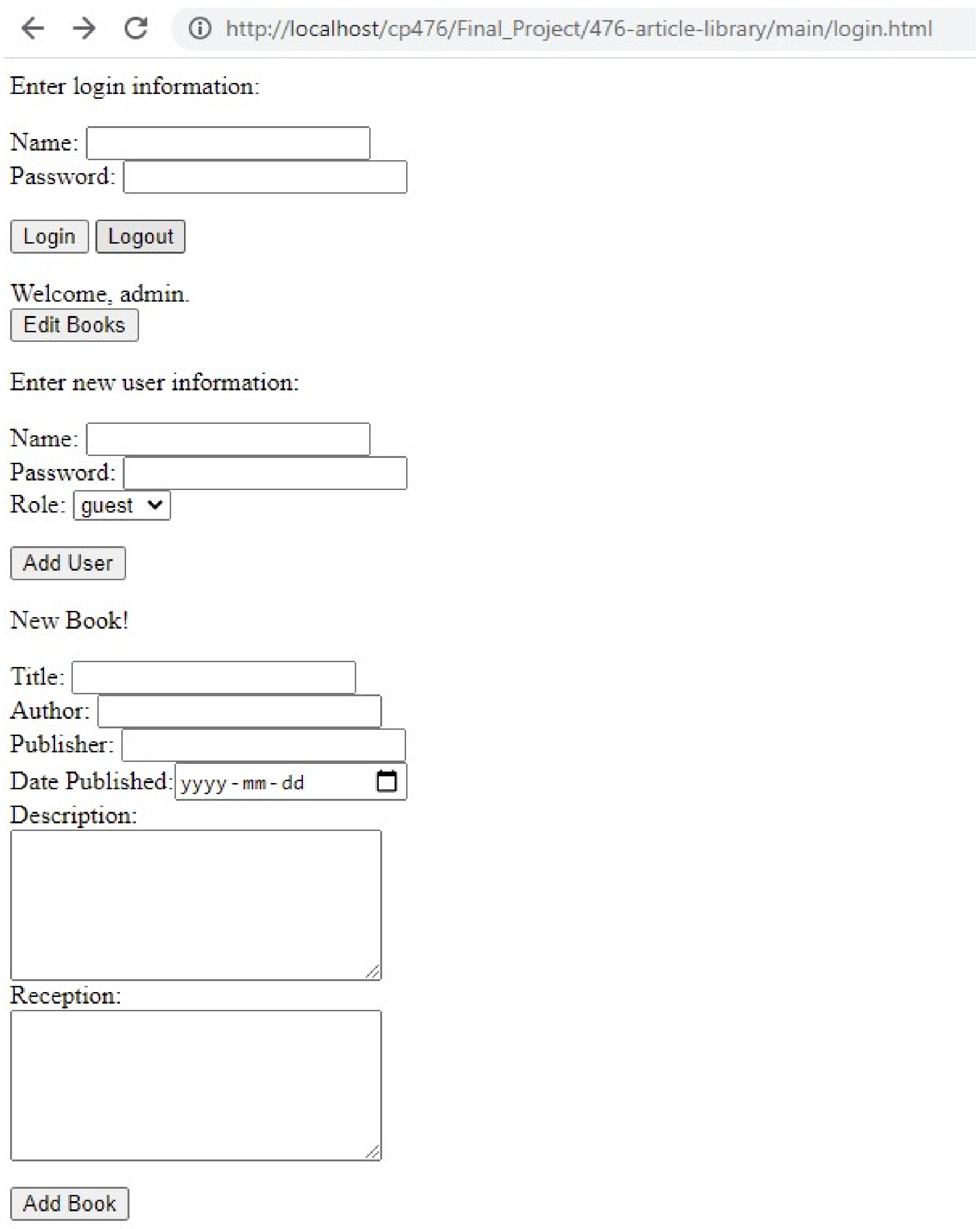
Add User

Book added successfully!

New Book!

Title: Book 700
Author: the Lord
Publisher: The Big Time Publishing Cor
Date Published: 1987-06-09
Description:
A book full of empty pages
Reception:
Positive overall.

Add Book


http://localhost/cp476/Final_Project/476-article-library/main/login.html

Enter login information:

Name:
Password:

Login Logout

Welcome, admin.
Edit Books

Enter new user information:

Name:
Password:
Role: guest

Add User

New Book!

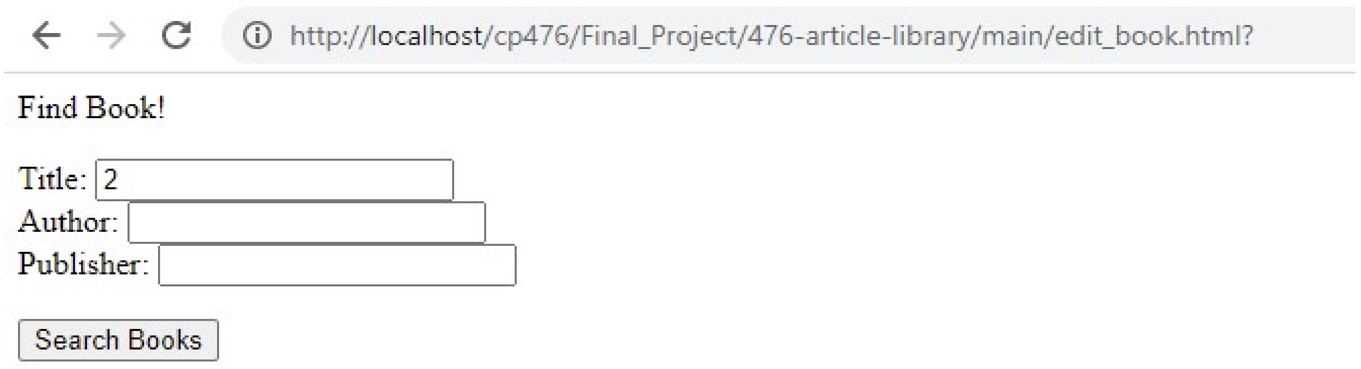
Title:
Author:
Publisher:
Date Published: yyyy - mm - dd
Description:

Reception:

Add Book

P2.2 Server side CGI components [60/60/*]

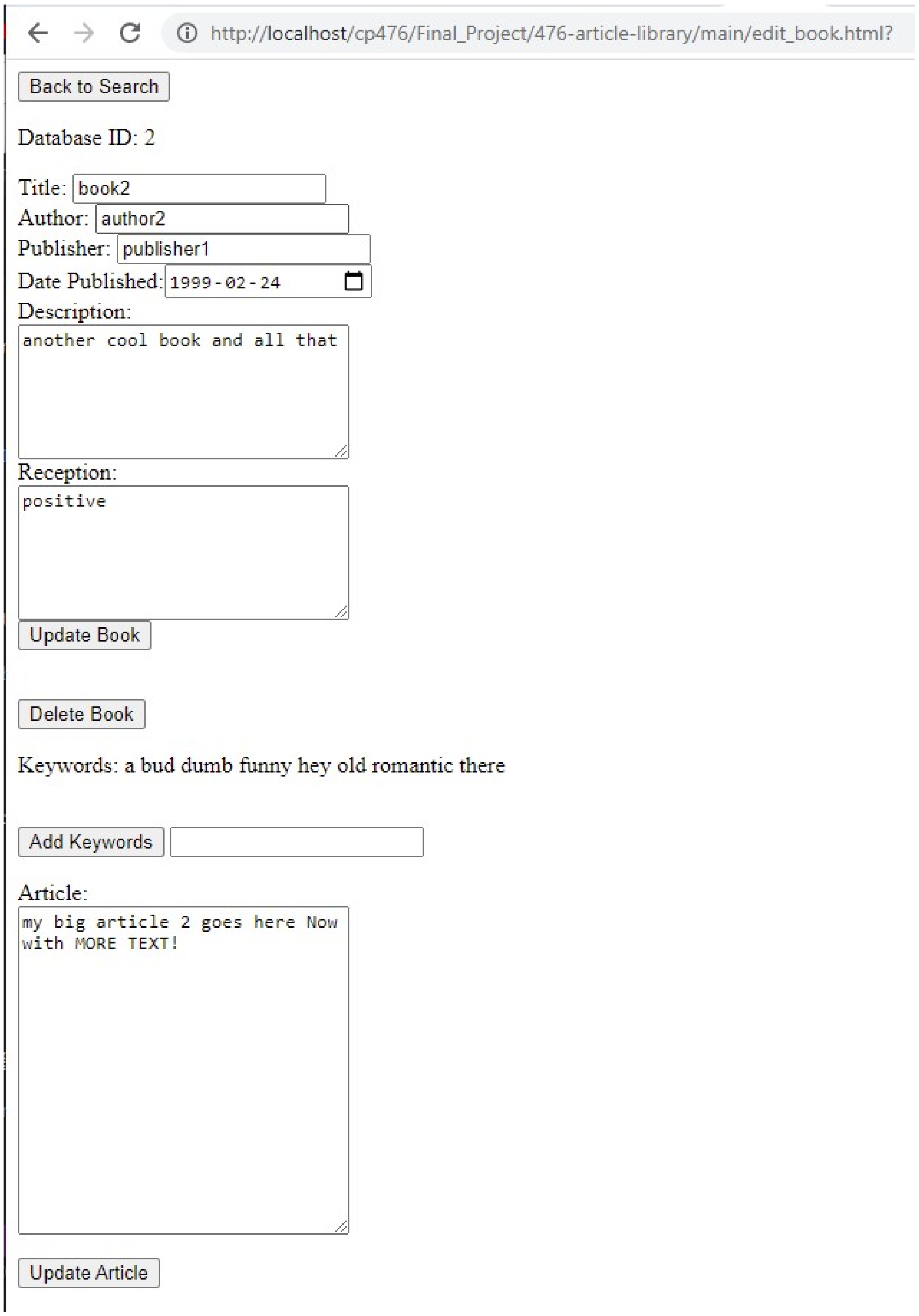
Our page use AJAX to communicate with the server side components, which are written in PHP. For example, the login page sends its data to the server, which checks against the database, and sends a role back if valid.



http://localhost/cp476/Final_Project/476-article-library/main/edit_book.html?

Find Book!

Title: 2
Author:
Publisher:


http://localhost/cp476/Final_Project/476-article-library/main/edit_book.html?
Back to Search

Database ID: 2

Title:	book2
Author:	author2
Publisher:	publisher1
Date Published:	1999-02-24
Description:	another cool book and all that
Reception:	positive

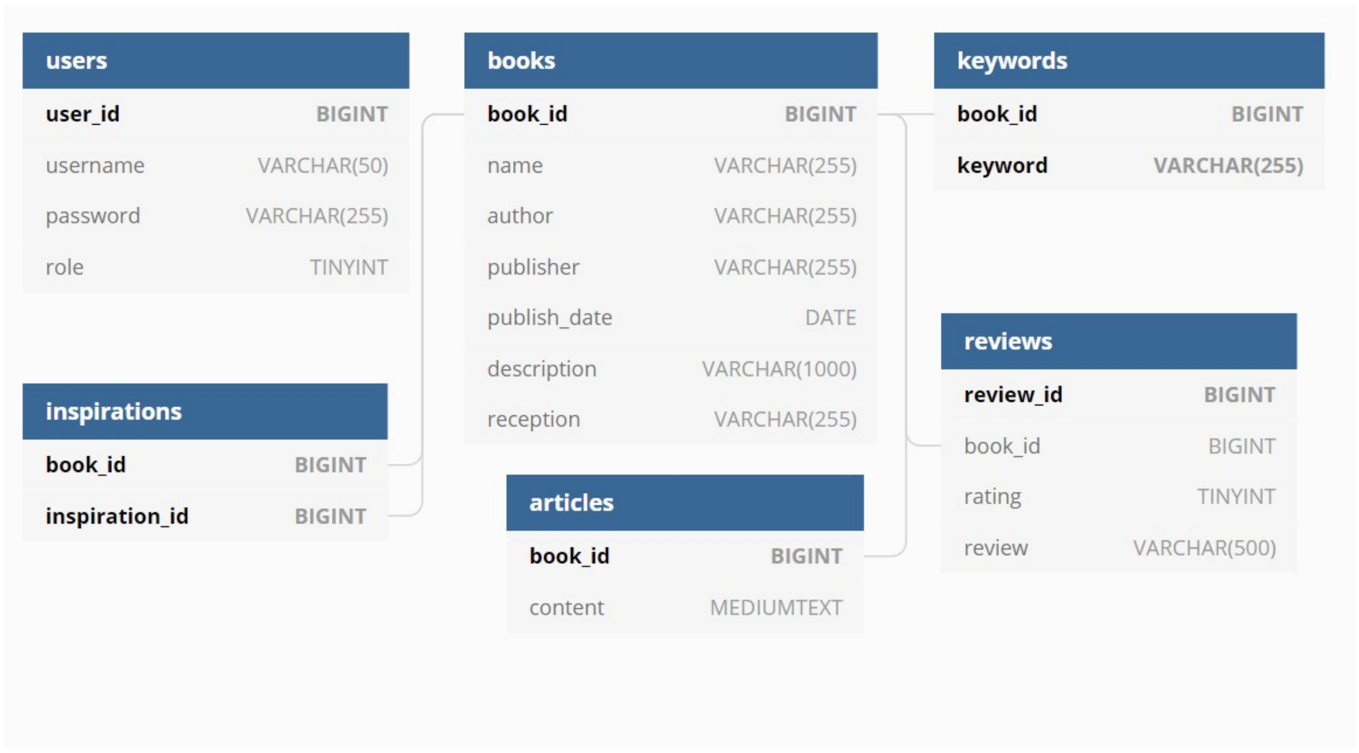
Keywords: a bud dumb funny hey old romantic there

Article:

```
my big article 2 goes here Now
with MORE TEXT!
```

P2.3 Database tier design, data, usage [30/30/*]

Our database has 6 tables:



users	
user_id	BIGINT
username	VARCHAR(50)
password	VARCHAR(255)
role	TINYINT

books	
book_id	BIGINT
name	VARCHAR(255)
author	VARCHAR(255)
publisher	VARCHAR(255)
publish_date	DATE
description	VARCHAR(1000)
reception	VARCHAR(255)

keywords	
book_id	BIGINT
keyword	VARCHAR(255)

reviews	
review_id	BIGINT
book_id	BIGINT
rating	TINYINT
review	VARCHAR(500)

inspirations	
book_id	BIGINT
inspiration_id	BIGINT

articles	
book_id	BIGINT
content	MEDIUMTEXT

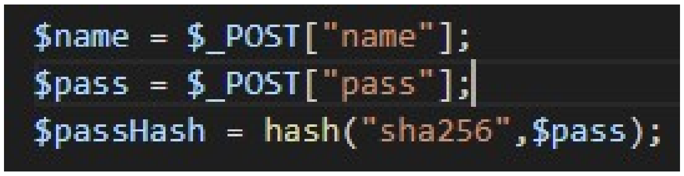
Database Design

The users table is a completely independant table for storing login information and roles.

The main table is books, as it stores the information for each unique book entry in the database. The other four tables are child tables that store tangential information about books, usually in a one-to-many relationship. Keywords stores individual keywords mapped to book ids, reviews stores individual reviews for each book, articles stores a unique article for each book, and inspirations store a link between one book and another (if the other book was an inspiration/source for the book).

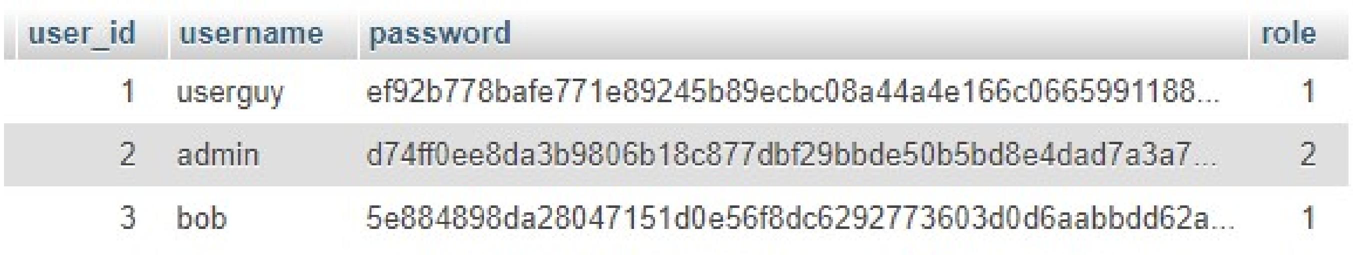
P2.4 New features and tools [25/30/*]

Using the 256-bit Secure Hashing Algorithm, we are able to store user passwords as hashes, allowing us to verify passwords without an intruder being able to steal them from our storage.



```
$name = $_POST["name"];
$pass = $_POST["pass"];
$passHash = hash("sha256",$pass);
```

Password hashing implementation



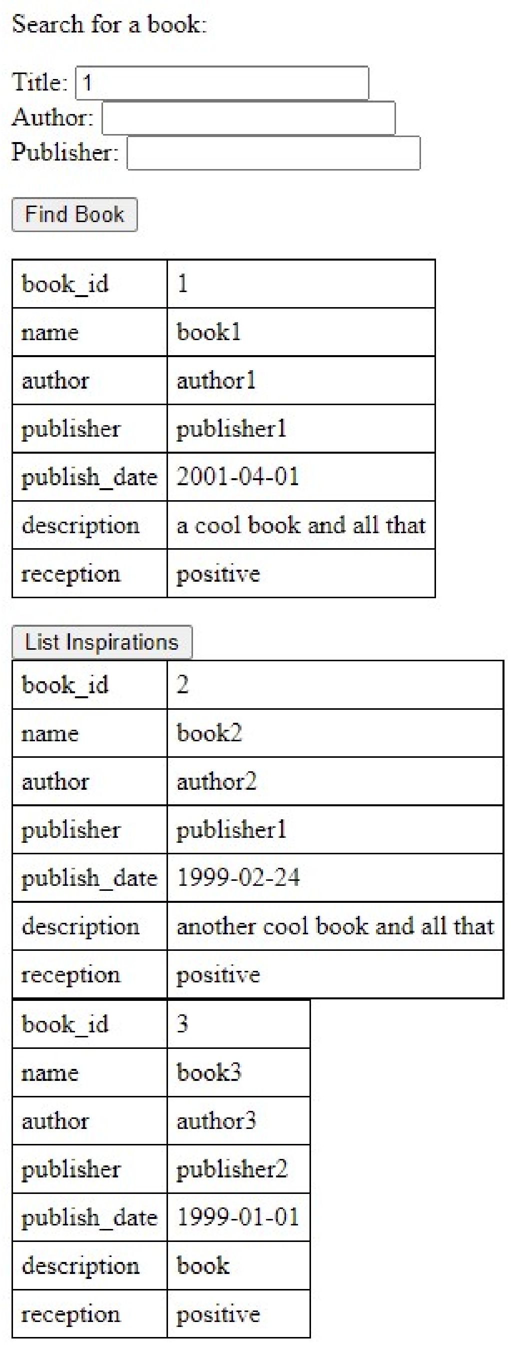
user_id	username	password	role
1	userguy	ef92b778bafe771e89245b89ecbc08a44a4e166c0665991188...	1
2	admin	d74ff0ee8da3b9806b18c877dbf29bbde50b5bd8e4dad7a3a7...	2
3	bob	5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a...	1

Hashes stored in database

However, our project does not make use of some other useful tools, such as Docker, and we believe this is a small shortcoming.

P2.5 Problem solving algorithms [30/30/*]

Books in our webapp have a component known as Inspirations. If a book was inspired by (or references) another book, we can enter that inspiration into the database's inspirations table. We implemented a BFS algorithm to retrieve stored inspirations from the database and create a list of inspirations, custom for each individual book. The core of this algorithm is taking the inspirations of each book, and adding those inspirations to the list in order. So, if a user wants an inspirations list for book 1, they will see book 2 (which inspired book 1) as well as book 3 (which inspired book 2). We believe this feature is useful for a curious reader to see a chain of influence through famous works. Ideally (with more time), we would implement this as a tree instead of a list, to better visualise the degrees of relation among the inspirations.


Search for a book:

Title: 1
Author:
Publisher:

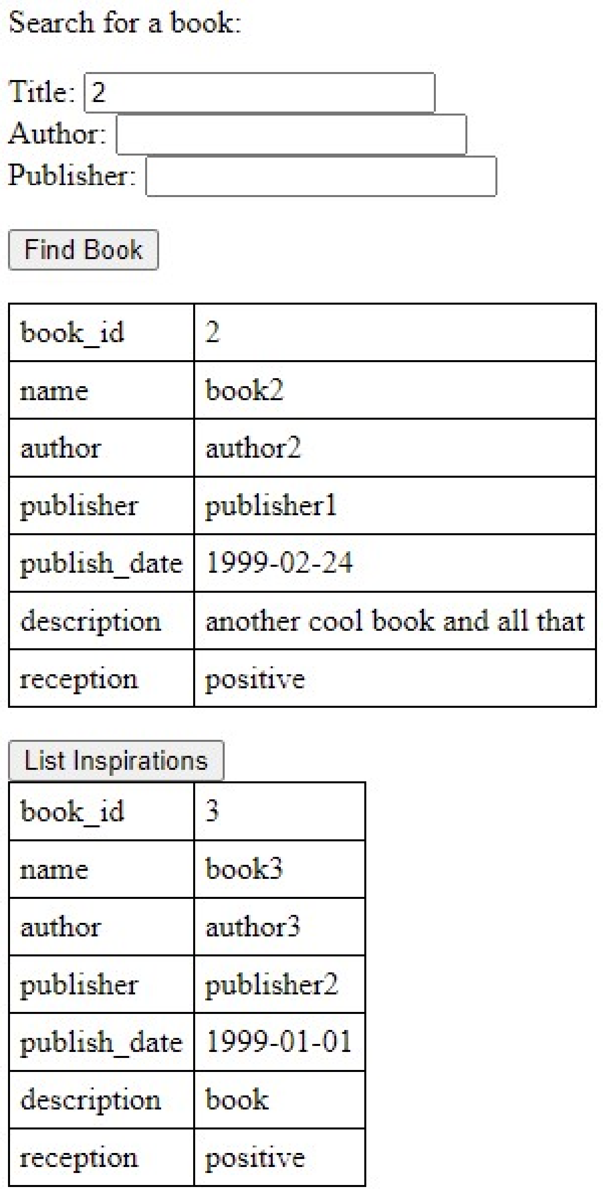
Find Book

book_id	1
name	book1
author	author1
publisher	publisher1
publish_date	2001-04-01
description	a cool book and all that
reception	positive

List Inspirations

book_id	2
name	book2
author	author2
publisher	publisher1
publish_date	1999-02-24
description	another cool book and all that
reception	positive

book_id	3
name	book3
author	author3
publisher	publisher2
publish_date	1999-01-01
description	book
reception	positive


Search for a book:

Title: 2
Author:
Publisher:

Find Book

book_id	2
name	book2
author	author2
publisher	publisher1
publish_date	1999-02-24
description	another cool book and all that
reception	positive

List Inspirations

book_id	3
name	book3
author	author3
publisher	publisher2
publish_date	1999-01-01
description	book
reception	positive

P2.6 Efficiency and robustness [20/20/*]

Our webapp is very responsive with almost no UI lag of any kind. In situations where excessive output or data may be generated (such as getting an inspiration list in 1000s), we've placed reasonable limits on how much data is retrieved from the database (usually a limit of 100 rows) reducing all worst-case performance.