# NYIT

**Spring 2019**

# Project: 01

# Title: Library Management System

**Name:** Salman Siddiqui

**School ID#:** 1204013

**Class ID#:** 20

**Course:** Database Systems

**Course ID:** CSCI-760-M03

**Date:** 05/20/2019

**CONTENTS**

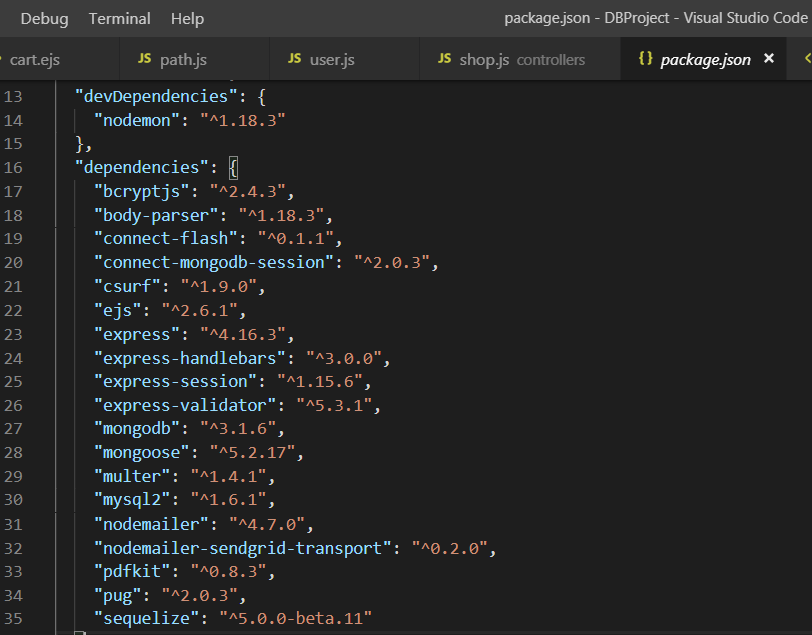
1. Introduction……………………………………………………………………………3
   1. Software……………………………………………………………………………….3
2. System Architecture…………………………………………………………………...4
   1. Entity-Relation Diagram………………………………………………………………4
3. Running the Application………………………………………………………………5
4. Conclusion…………………………………………………………………………….6
5. **Introduction**

The goal of the project was to create a library management system using NodeJs and optional backend database options like mySQL, MongoDB and Postgre to name a few. The library management system is a web application that has CRUD operation feature on the admin side as well as the user side. The user of the library can check in and check out, browse the books, and see details about the book. Operations pertaining to the user end affect database based on the CRUD operation and similarly on the admin side in case of updating the inventory or adding or deleting new books. As mentioned above, it is still under works and we will demonstrate what it can do so far and what the future iterations of the project will look like and what will it be expected to do.

* 1. **Software**

We used the Visual Studio Code text editor to implement the project. Along with that, we installed NodeJs and npm to run the JS scripts seamlessly. Furthermore, as and when needed we, we installed packages from npm to perform smoothly and effectively keeping the demands in mind. For the client side, we used JavaScript framework Express to simplify routes creation for the different pages and ejs. For the database, we used MongoDB with mongoose on JavaScript to facilitate efficiency and create a seamless environment

The above mentioned packages and frameworks were not the only ones implemented. In the package.json file of the project, we can see the various dependencies installed for this project.



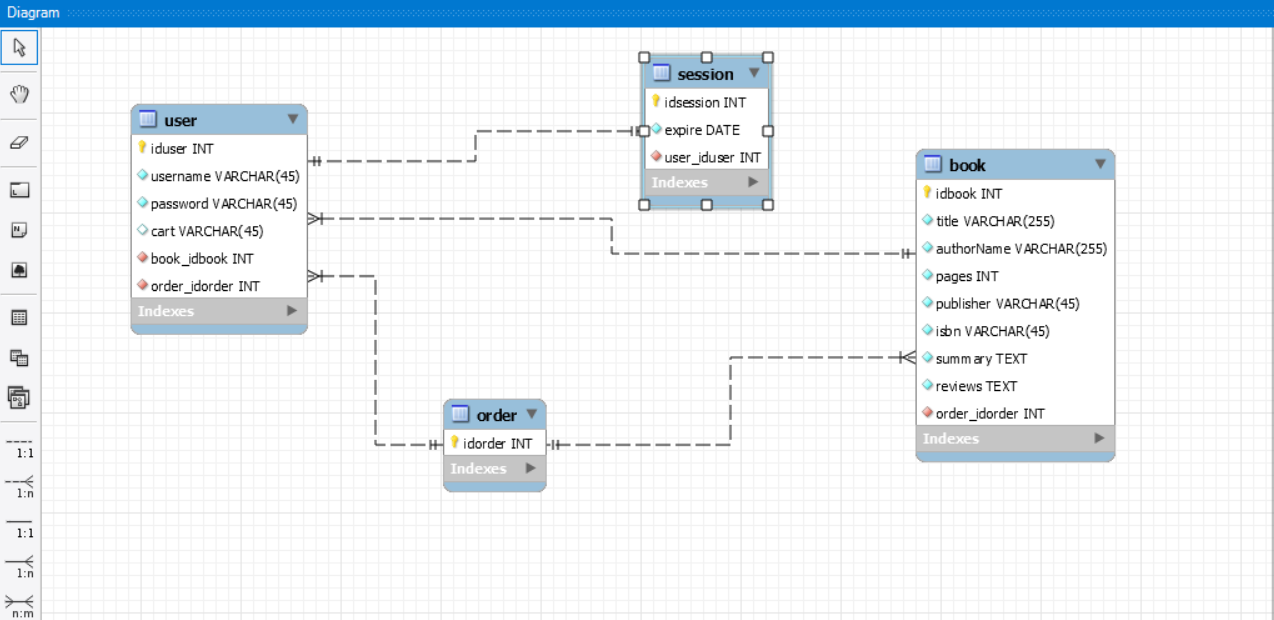
We will go through some of the packages as they are important for the functioning of the module. BCryptjs was used to encrypt the password/cardno of the user in the database. In general, saving scripts and passwords are visible on the database and in order to save the password as a hash we installed bccrptjs from npm to hash and save the password. Mongoose is the framework for mongodb that we use. Connect-mongodb-session was used to create a secure user session that held local cookies and cache for the user and once the user logged out, it deleted the cache for security. Developer dependencies that was used was nodemon which helped in running the server and see the changes made Visual Studio with ease.

1. **System Architecture**

The user is a client, through a web browser that requests services of the library management system. The server responds with the post and get methods created for the routes depending on the user’s requests and actions. The backend database is called into action based on the users action. For example, when a user logs in using his or her own credentials, the information is stored on the database to create a session.

**2.1 Entity-Relation diagram**

For now we are using a basic E-R diagram that serves as a good starting point for the direction we wanted to take for the project.

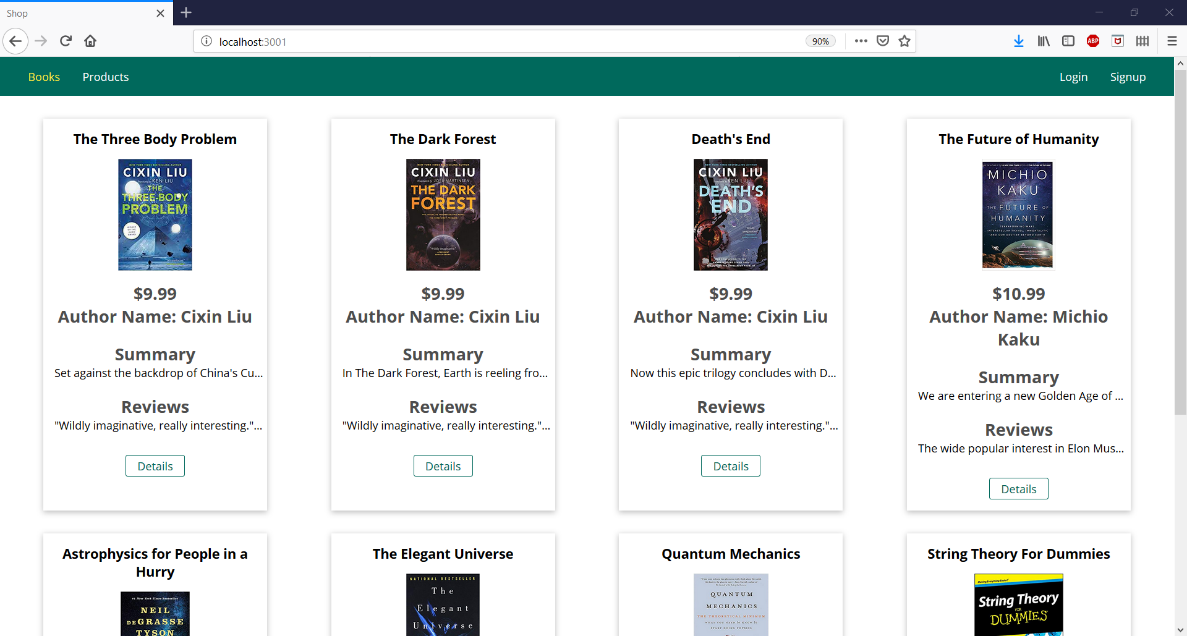


MongoDB is schema-less database hence it is called NoSQL. However, we can still create relations within the documents of a database. We create a database called library that has the following collections:

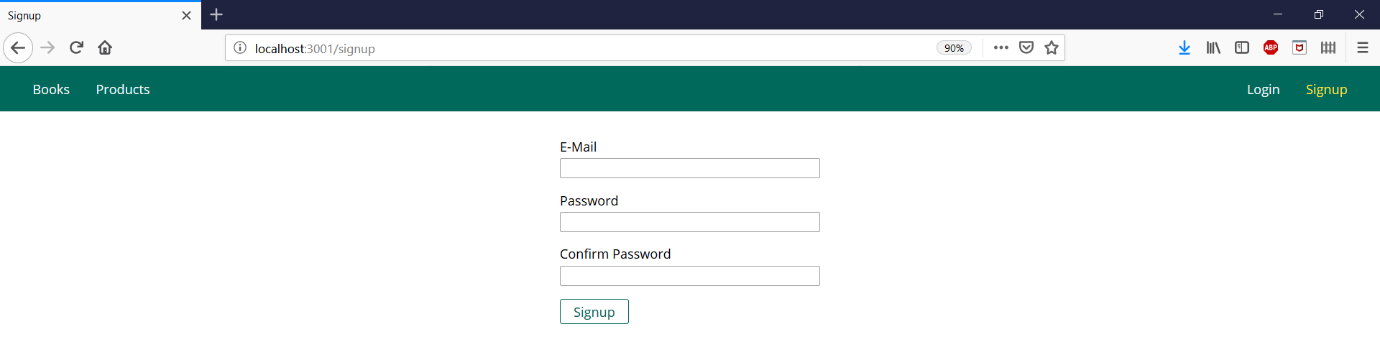
1. User: The user collection has documents in the form of username and password. Furthermore, as we mentioned we can create relations in the database. Therefore we add the books\_id to keep track of the books the user has. Furthermore, we added an order\_id to keep track of the books he has checked out. In the future iterations of the project, the order table will change to check out/check in table.
2. Session: The session collections has documents that are extracted from the user, i.e. his username and password along with the expiry of the session which in this case is the cookie. This creates a secure environment for the user to browse in and not leave him or her vulnerable.
3. Book: The collection contains information about the book on the library database.
4. Order: The order is a collection of the user with respect to the book he has in his library and the checkout/ordered products. It has many to many relation with the user and the product since many users can have many orders and many books.
5. **Running the Application**

We run the application by first navigating to the project folder and run the following command

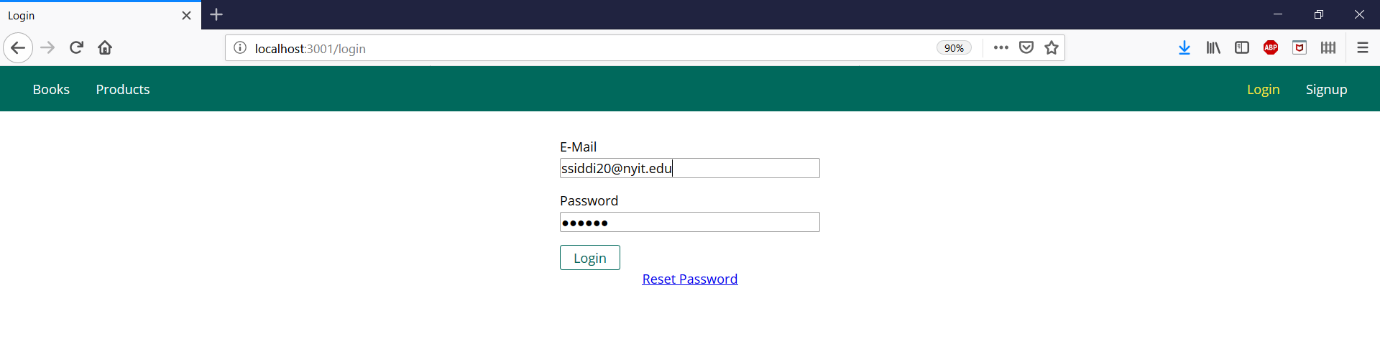
npm-start

This command runs the nodemon for a live server. Next we navigate the port at which the application is listening, which in our case is 3001 on the web browser. 

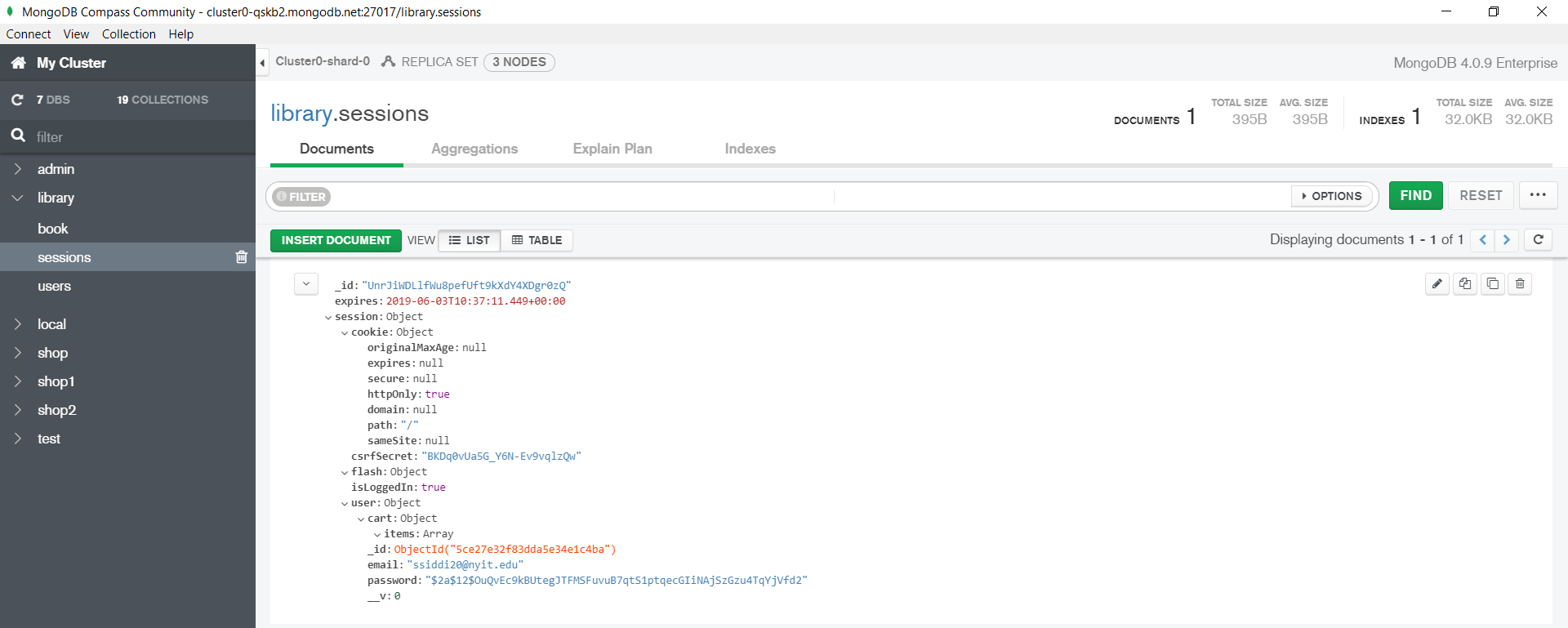
Here are some of the books that are already on the database we created when we started the server. localhost/3001 is the link we are right now which shows the books without the option to checkout or order. This is because the user has not signed on or logged in. Next, we click on login or signup depending on the user.



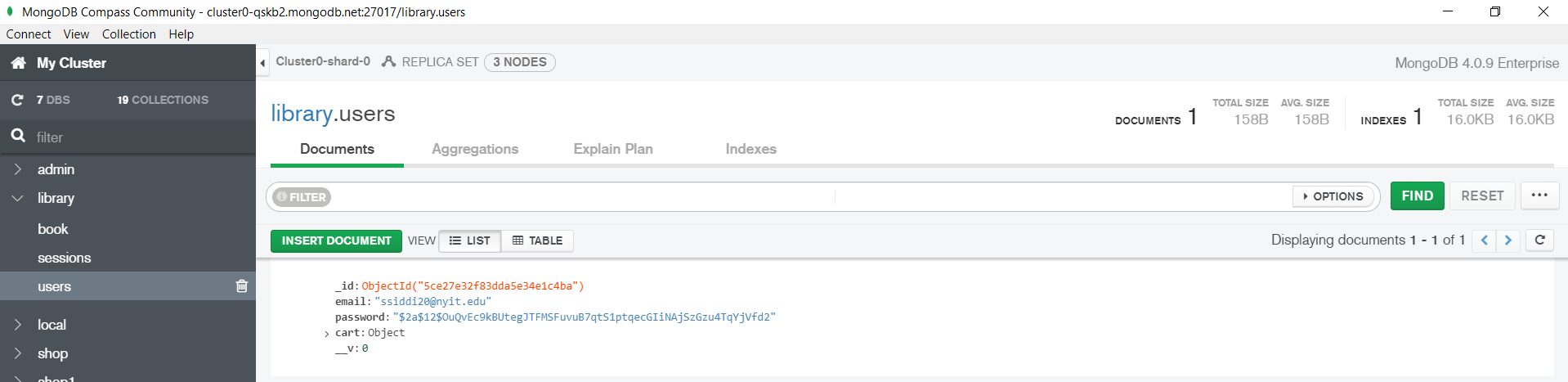
This is the signup page. Since we already have an login user created, we login using our credentials.



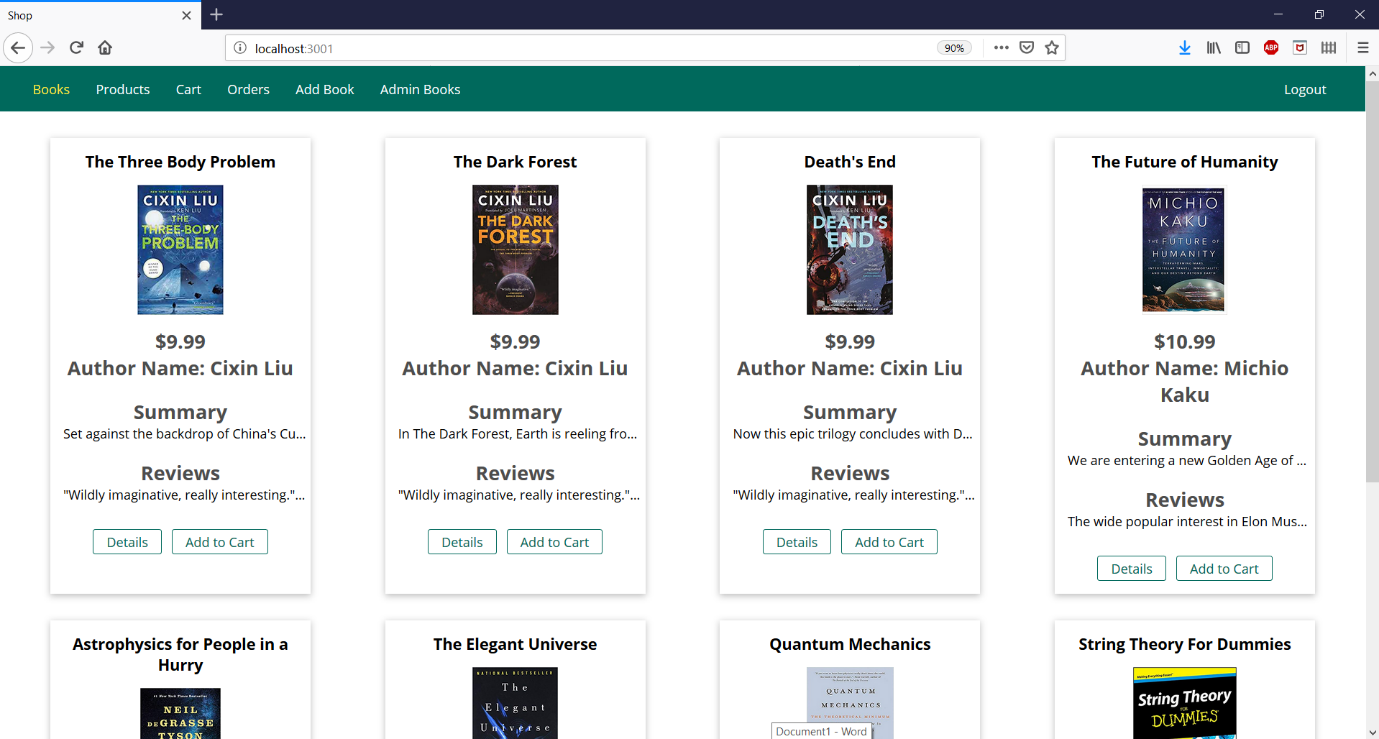
When we login, as mentioned above, we create a session for the particular user. The session has the user email and hashed password. It borrows this information from the users collection



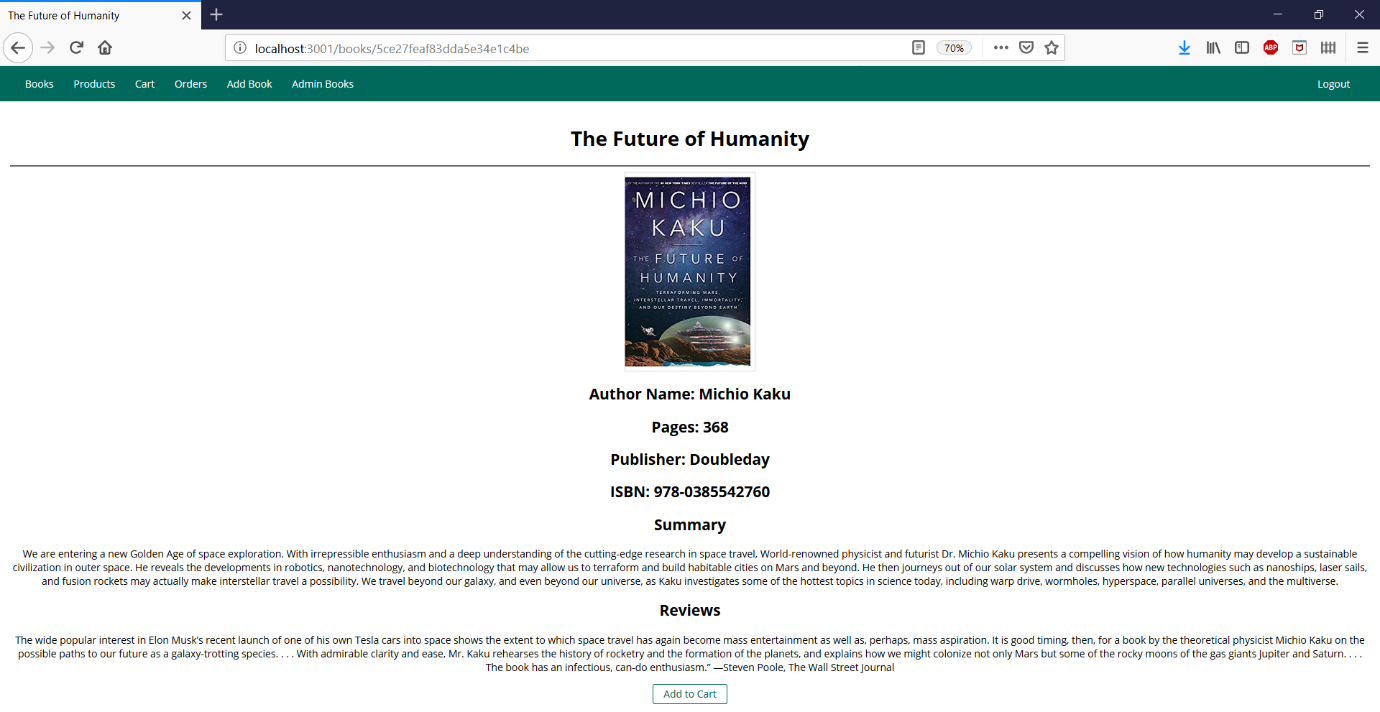
The user collection has cart object which holds the books and orders of that particular user.



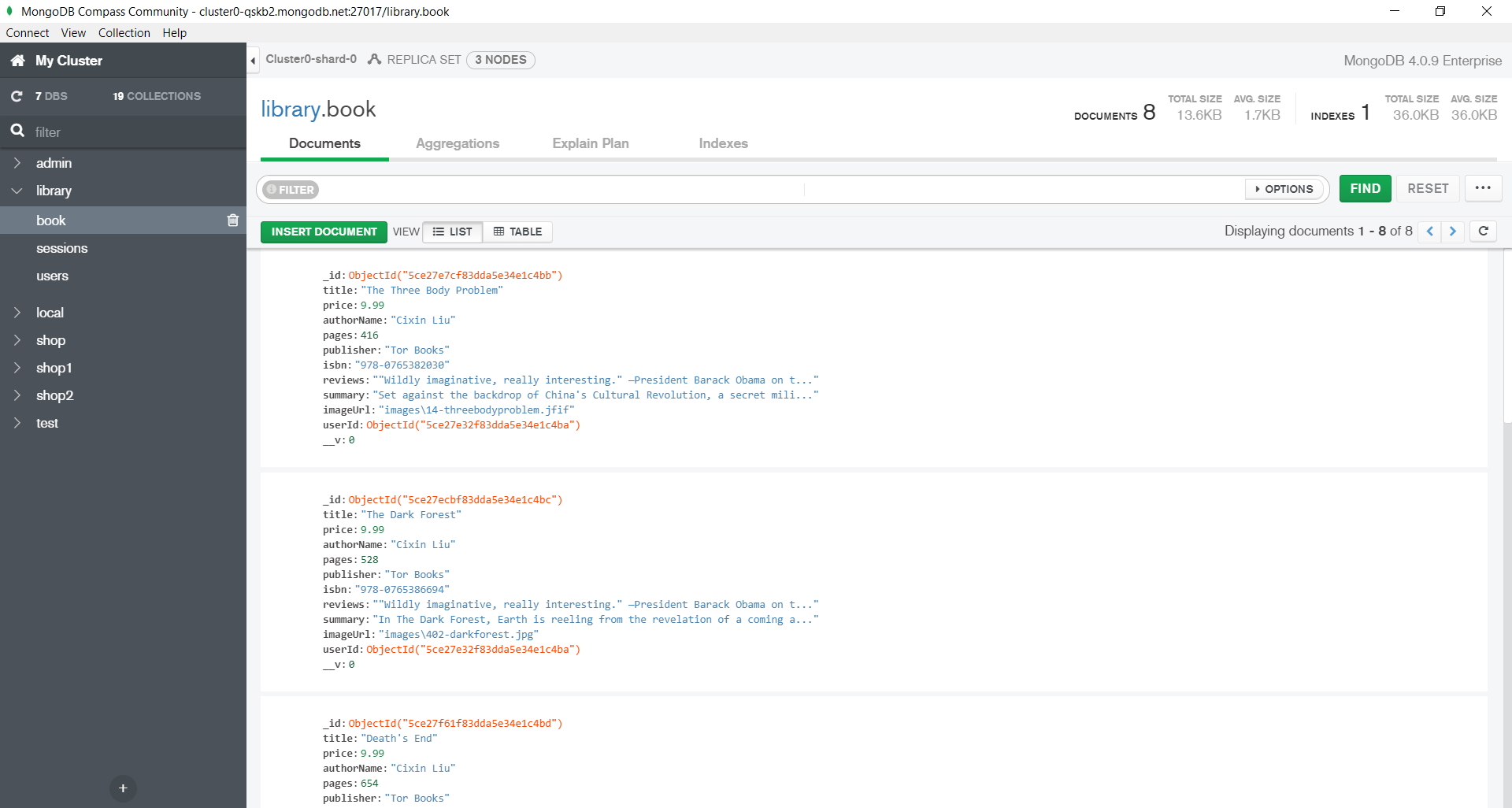
Once we login, we can see the homepage along with the add to cart option



The boxes contain partial information about the book therefore we added a details button to redirect to a page where information is available. After we click on the detail page, we can see the pages, publisher, ISBN, summary and reviews on the book. If this book peaked the user’s interest, he can directly add the book to his or her cart.

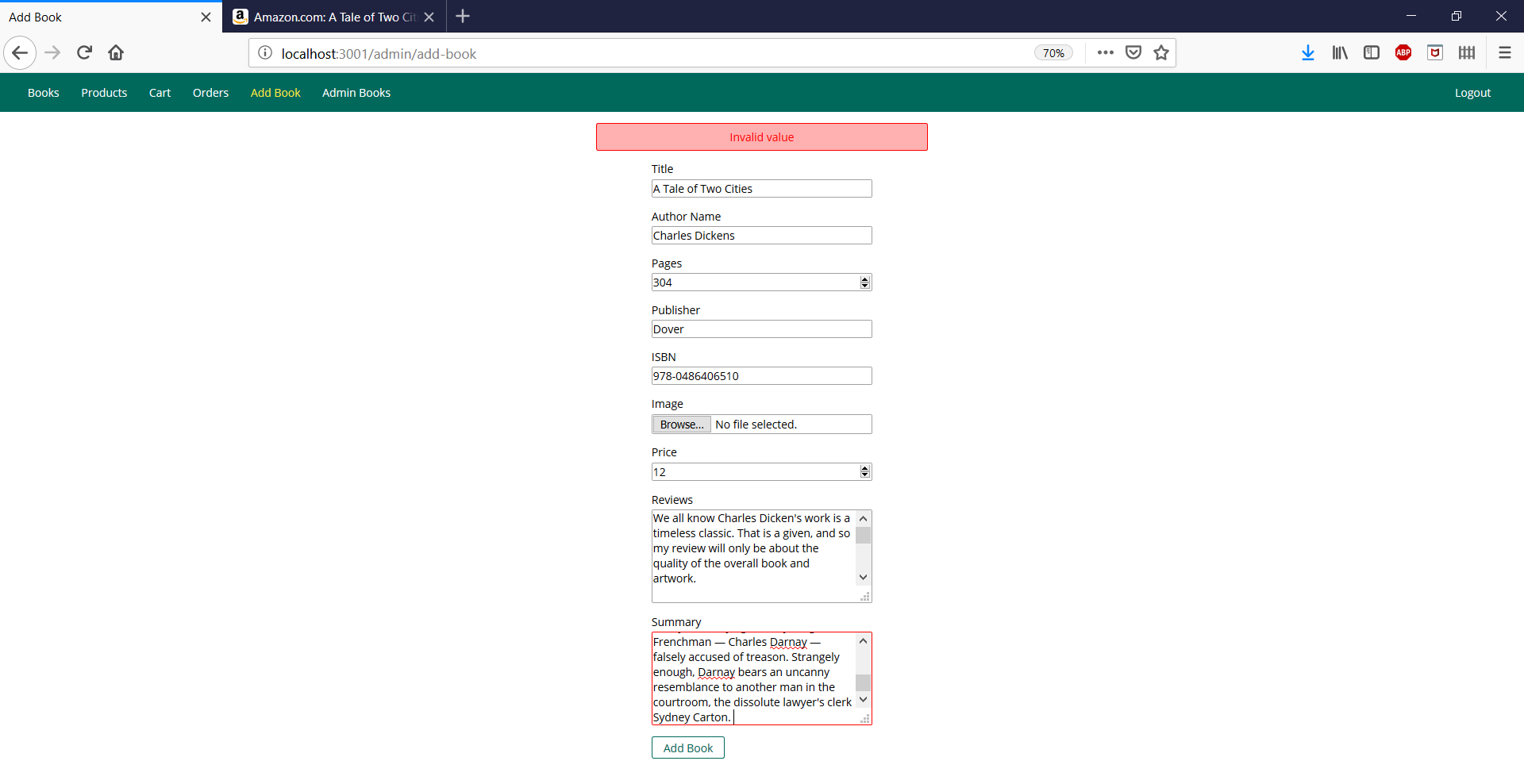


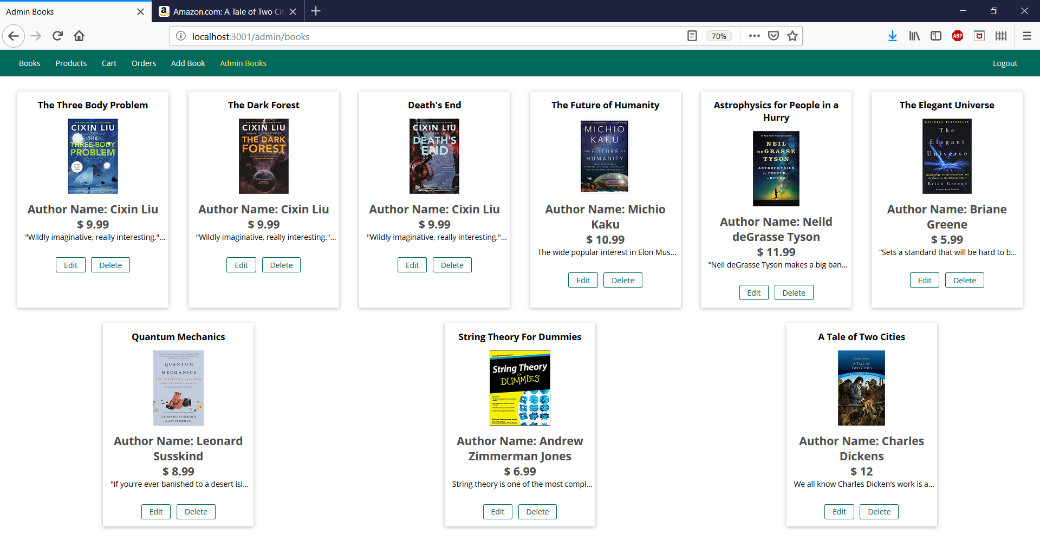
And just as proof we can check the database on MongoDB Atlas to verify certain posts.

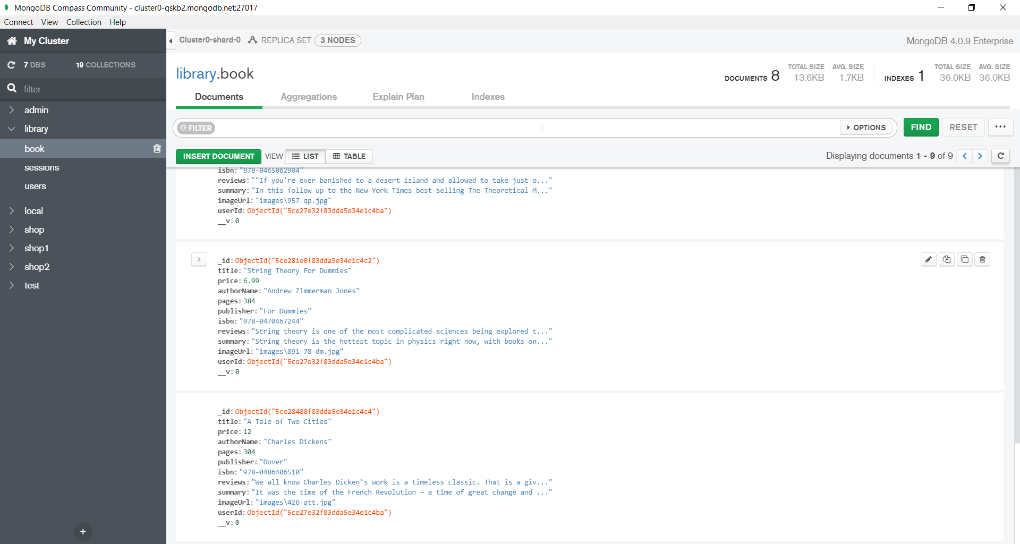


We see that the first three posts are The Three Body Problem, The Dark Forest and Death’s End, all part of a trilogy by Cixin Liu. We can cross verify with the front page above to check if it is correct.

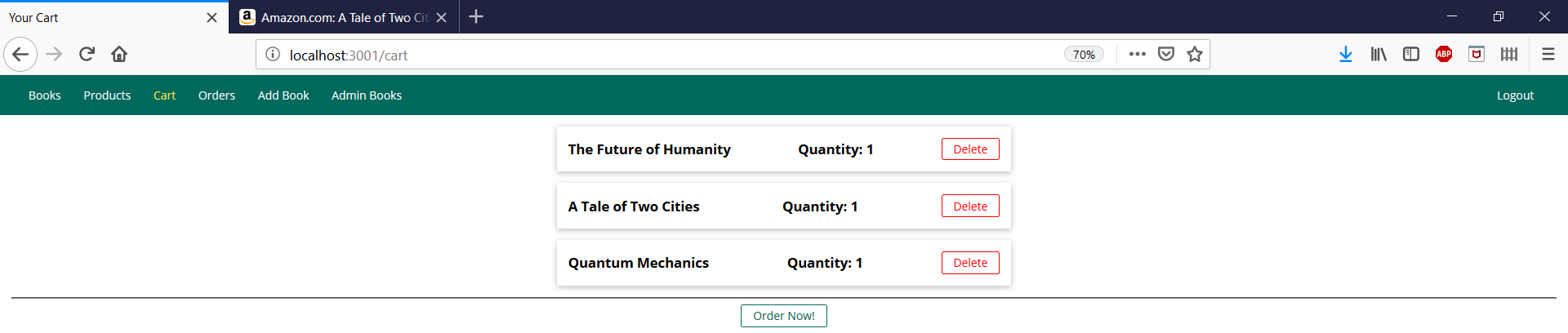
The admin can add more books for the users as we can see below. We added A tale of two cities by Charles Dickens and it shows on the frontend and the backend database.

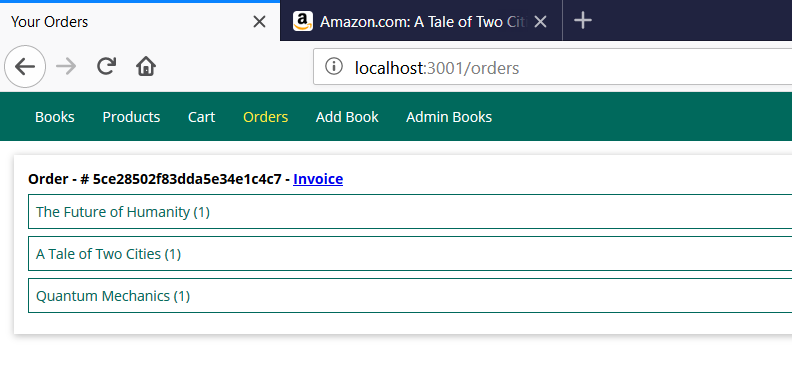




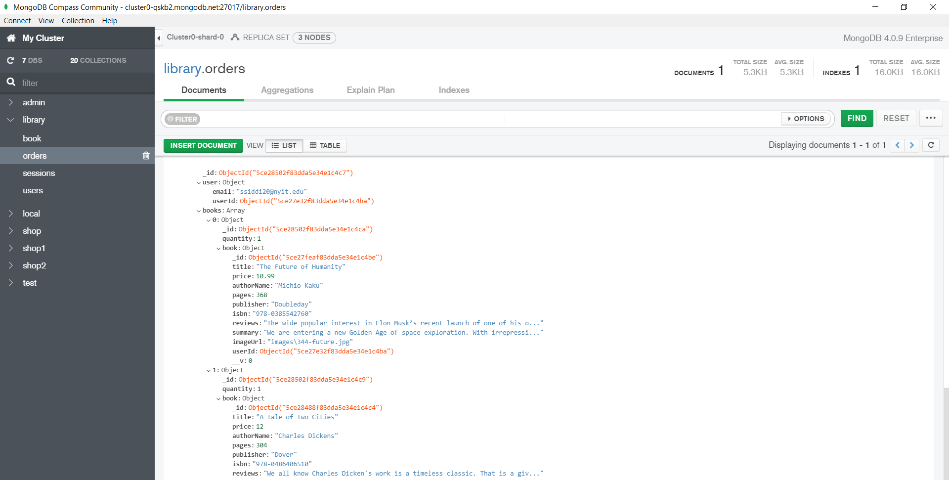


Next we select the books we want to order and can proceed to checkout and place the order for the books. This generates an invoice which when clicked upon gives, the invoice of the order.

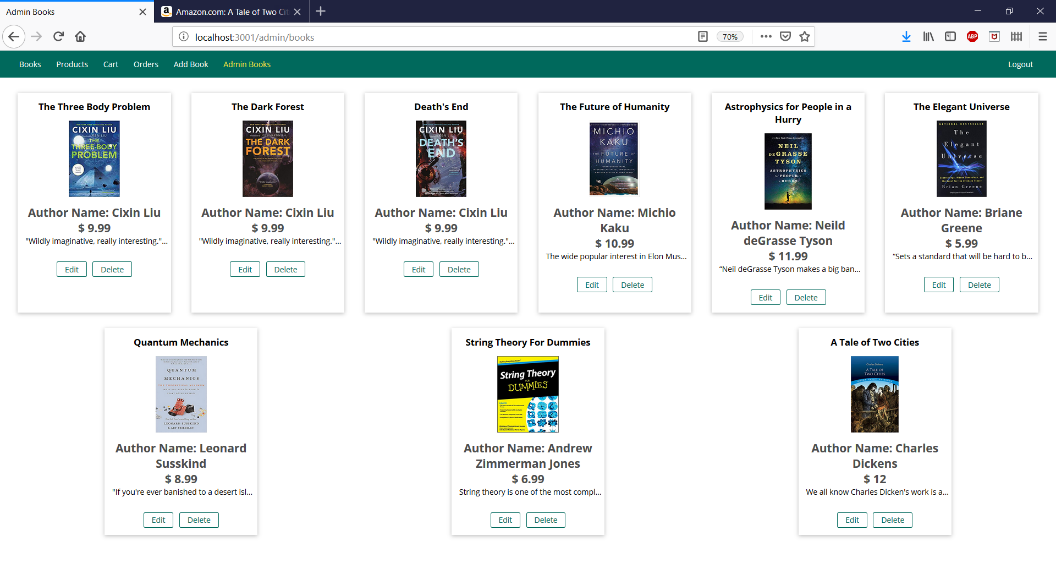




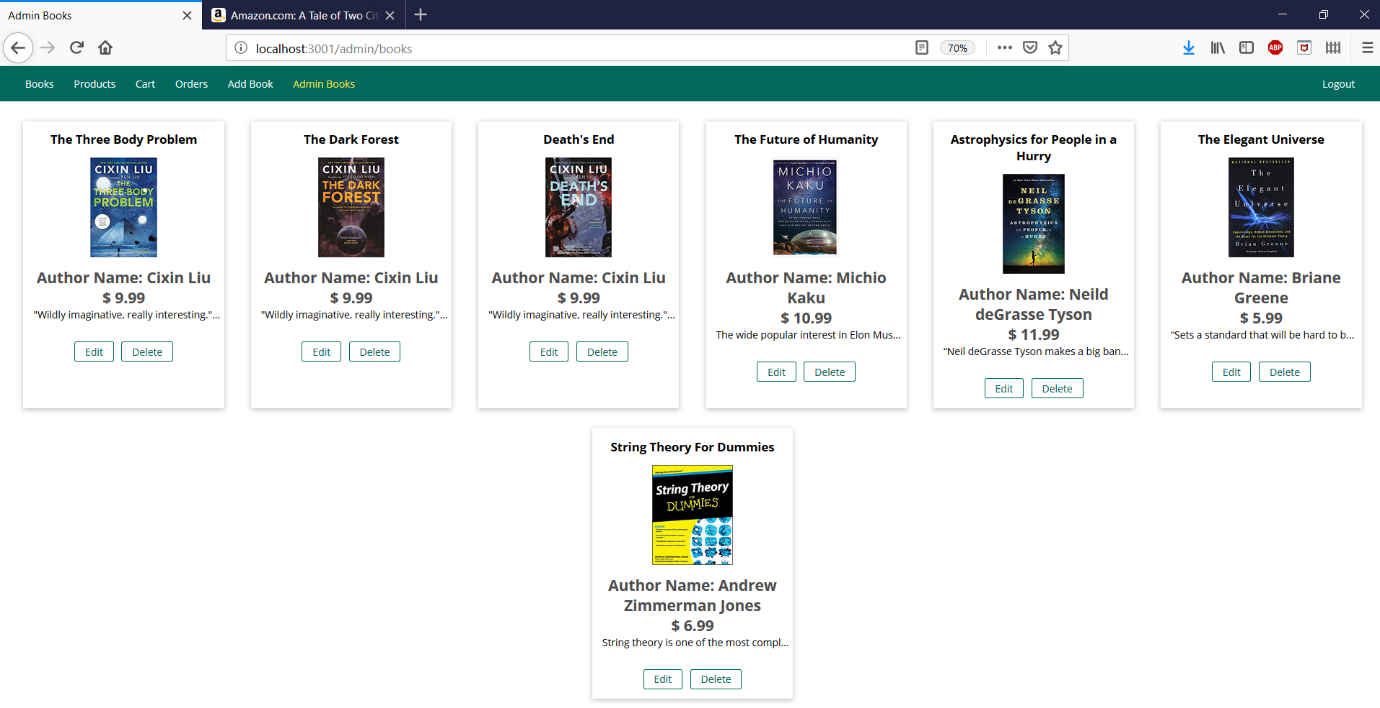
After we place the order, we can check the database to see the order. We can cross check with orders in the database with the invoice above and also see the user the ordered the product.

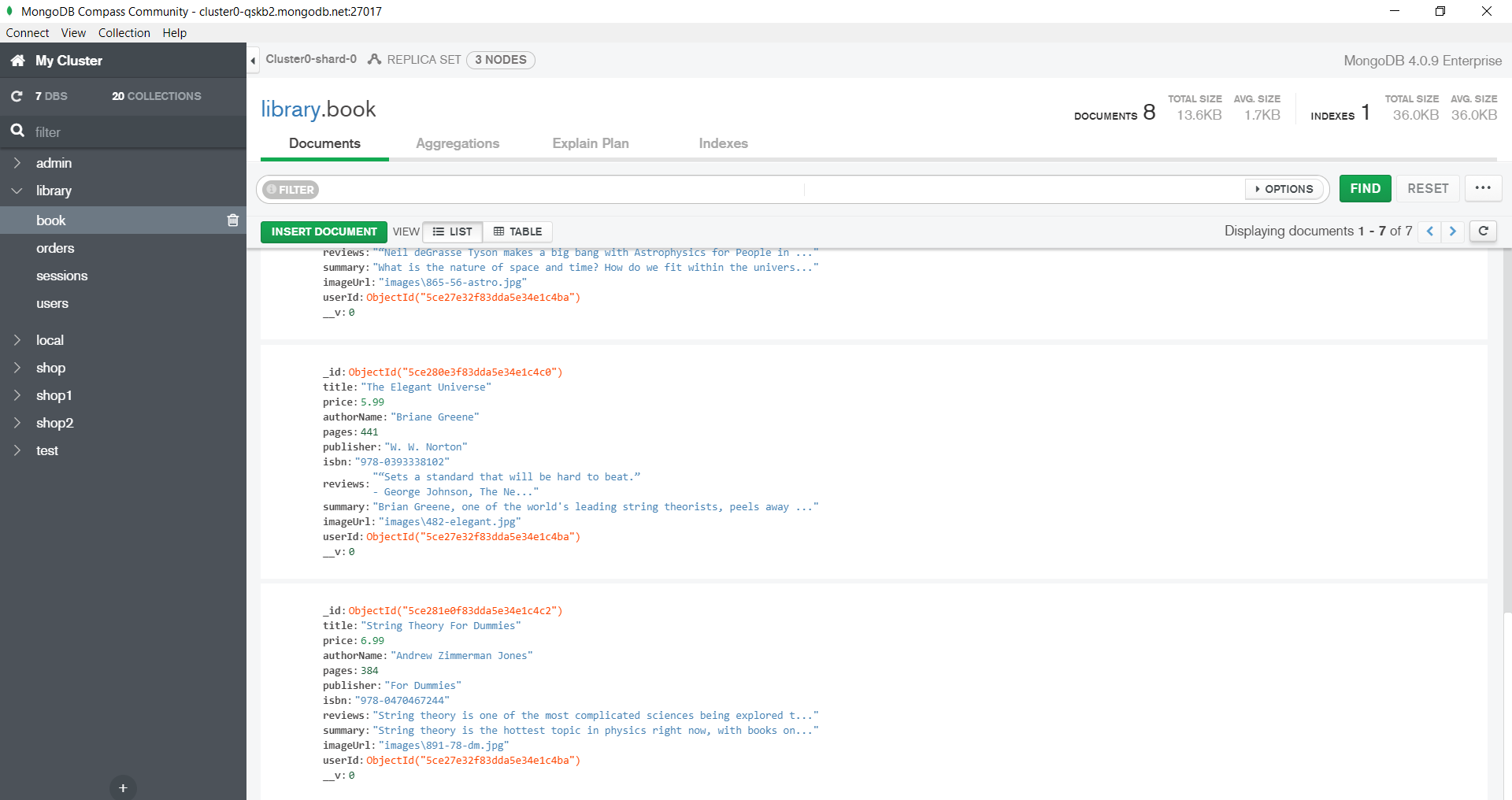


As mentioned above, we can perform CRUD operations. The edit button in the image helps in editing the information about the page. The edit button redirects to a form we have seen above.



The delete button deletes the box from the front end and the document information from the database. We have deleted two book, Quantum Physics and A Tale of Two Cities. We can check the database to verify the deletion. We can observe the document number has reduced to 7





1. **Conclusion**

The application is still developing and required more fine tuned approach to meet the demands. The application will extend to libraries and extract the databases and create a large user database. Currently the application is the first step in the right direction.