Bookstore Application using MEAN Stack

Team 4

Scott Zhou

Shanshan Yu

Anil Erturk

CS628 Full-Stack Development II

City University of Seattle

[zhouscott@cityuniversity.edu](mailto:zhouscott@cityuniversity.edu)

[yushanshan@cityuniversity.edu](mailto:yushanshan@cityuniversity.edu)

erturkanil@cityuniversity.edu

**Abstract**

MEAN stack consists of MongoDB, Express, Angular, and Node. It is a JavaScript stack that helps build dynamic web applications. Because these four technologies support programs that run in JavaScript, they could be used to implement both server-side and client-side in one language. We will design and implement a bookstore application with the MEAN stack. This application can perform CRUD (create, read, update, and delete) operations, hence making it dynamic. Users can check the list or detail, modify and delete the book.

**Keywords:** AngularJS, Bookstore Application, Express, MongoDB, Node

**1. INTRODUCTION**

A bookstore application should allow the administrator to manage their books in stock and show customers the books' information for sale or rent. A sophisticated bookstore application such as Amazon bookstore can support millions of simultaneous requests. In this article, we will discuss how to build a bookstore application in industry standards, and the required technologies to be used in the project.

**1.1 REQUIREMENT**

The bookstore application should have a public interface that can interact with customers, a private interface that helps the book administrator to manage the books, and a database to store the books’ information. The administration side should allow the administrator to perform CRUD operations.

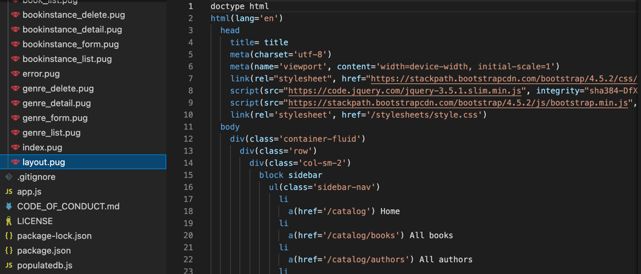
The application should support 1,000 users' visits simultaneously, and the application code should allow building in living PaaS platform easily.

**2. VIEWS**

The bookstore program is going to use Bootstrap and Pug template to build our front-end. Express has a built-in feature to support static template files in the application. The template engine uses actual values in the template file instead of variables to convert the template into an HTML file sent to the client. There are many popular template engines compatible with Express, such as Pug, Mustache, and EJS. The default template generated by Express application generator is Jade, which has been renamed Pug.

Set the application setting properties like views or view engine in app.js to render template files. All template files are in the views directory, this will default the views directory in the application root directory. (Figure 1)

There is no need to define the engine or load the template engine module after the view engine is set up because Express will handle it internally.

  
Figure 1. Layout view

Pug is a high-performance template engine inspired by HTML and implemented with JavaScript. It offers port to other languages such as Java, Python, Ruby, etc. Pug simplifies the syntax to make it easy, fast, and flexible to reuse the HTML code. Pug also gives it up to the power of Bootstrap.

Bootstrap is the world’s most popular framework to build a responsive design web application with a template starter page. We simply include the script tag wrap with query and Bootstrap to import it. It is a collection of tools for building web applications, and a free open-source project which can be found on GitHub. The reason Bootstrap is gaining popularity is not only because of its flexibility and ease of use, but it also helps make sure the design is responsive and compatible with many different browsers. Especially when using JavaScript, it provides rich extensibility, built-in support for jQuery plugins and a programmatic JavaScript API (Application Programming Interface).

**3. MODELS**

To achieve the goal of making a website dynamic, the data must have a place other than the controller where it can be saved and retrieved by any controller thread. The internet works in a multithreaded manner. When multiple requests arrive at the server, multiple threads of the controller are created. These separate instances cannot access each other’s values. This is the first reason the server requires a shared space where the data can be stored and retrieved by all threads.

This shared space could be a text file or a database. Servers use a database to have random access capability as opposed to sequential access capability. This database is required to make the website dynamic because if the data is hard coded in the view, the website will be static. The server will not be able to serve different versions of the same view to the client. If the data is in the controller, the server can serve a static webpage or serve a dynamic webpage with limited temporary data that exists in the thread. To increase the scope of the controller, it can be given access to the database. Now the controller can serve dynamic webpages with the data it and the database have. This is the second reason the server requires a database, to be dynamic and persistent.

So, to sum up, the database is required so the data can be stored and retrieved, hence making the server dynamic and persistent. And, the data can be retrieved by all threads, hence allowing multi-threading which is needed on the internet.

The MVC pattern Divides the software into three parts. Controller functions do certain tasks such as communicating with the database through models, passing key-value pairs to a view, and sending that view to the client. Views are dynamically generated HTML pages.

The requests first hit app.js file, which then forwards the requests to routers. The same app.js file also creates the connection with the MongoDB database using Mongoose. Mongoose also allow data types and restrictions to be specified in the schema. Mongoose schemas specify the format of the database records. This way the data stays in one format and brings the database closer to the backend developer. The view engines such as Pug also turn views into a more familiar format, hence bringing the front end closer to the backend developer. These technologies bring all three parts of the software closer, hence enabling full stack development.

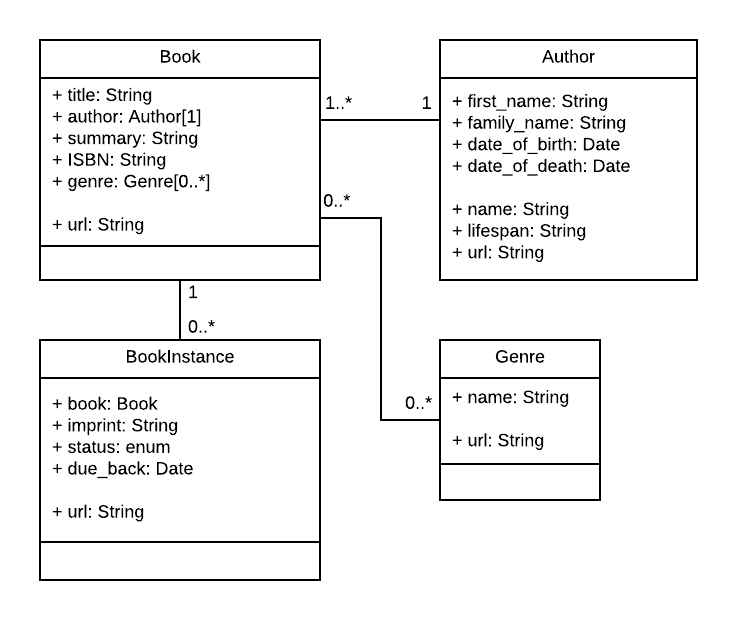


Figure 2. model design

Before the database and schemas can be created, the structure of the database needs to be decided. Above can be seen the structure for the database. There are four classes. All the relations are between the Book class and other classes (Author, Genre, and BookInstance). Figure 2 indicates the relationship between the four items and the four items’ attributes.

Each path is one instance variable of a class and its properties. The schema is made up of all the paths of the class. A document is an instance of this schema, which means the document is an object. A collection contains multiple documents for a class.

There are four schemas that each correspond to one of these classes on the database. The schemas are book, author, genre, and bookinstance.

The first model is the author model. The dependencies for the model are imported first. Note that the DateTime class is required for this model.

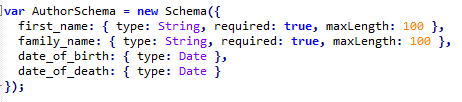


Schema class is imported from the mongoose package.

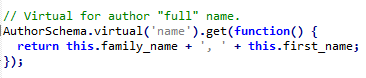


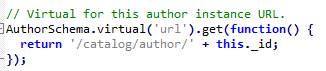
Then a new schema object is created using the constructor of the Schema class from the imported mongoose package (mongoose.Schema). The constructor is given the variables to the schema.

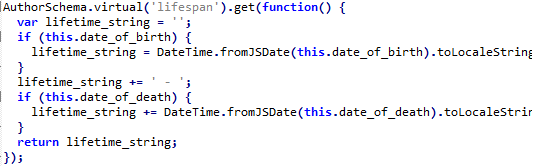
This model has five variables. first\_name, family\_name, date\_of\_birth, and date\_of\_death. Each variable has at least one attribute. Each variable has a type of attribute. The types of first\_name and family\_name is String meanwhile the types of date\_of\_birth and date\_of\_death is Date. This is an example of composition because an object has a reference to another object. first\_name and family\_name also have required and maxLength attributes. They both have true for the value of required and 100 for the maxLength.



Author model also has getFullName, getID, fullName lifespan, and getDate methods.











And lastly, the model is exported with the name “Author”. This is how the model is accessed from the controllers.



The second model is the book model. The dependencies for the model are imported first. Note that the DateTime class is not required for this model.

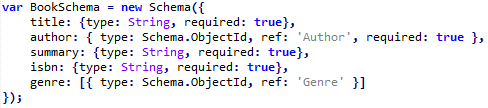


Schema class is imported from the mongoose package.

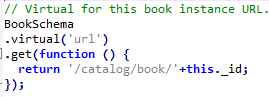


Then a new schema object is created using the constructor of the Schema class from the imported mongoose package (mongoose.Schema). The constructor is given the variables to the schema.

This model has five variables. title, author, summary, isbn, and genre. Each variable has at least one attribute. Each variable has a type of attribute. The types of title, summary, and isbn are String. The type of author is the Author schema “type: Schema.ObjectId, ref: 'Author'”. The type of genre is the Genre model “type: Schema.ObjectId, ref: 'Genre'”. This is an example of composition because an object has a reference to other objects. Also, title, author, summary, and isbn variables have the required attributes with the value true.



Book model also has a getID method.



And lastly, the model is exported with the name “Author”. This is how the model is accessed from the controllers.



The third model is the bookInstance model. The dependencies for the model are imported first. Note that the DateTime class is required for this model.

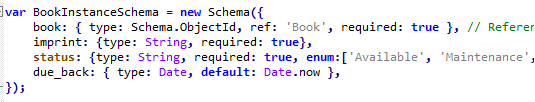


Schema class is imported from the mongoose package.

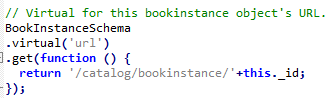


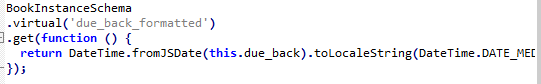
Then a new schema object is created using the constructor of the Schema class from the imported mongoose package (mongoose.Schema). The constructor is given the variables to the schema.

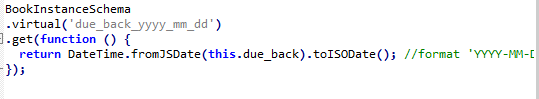
This model has four variables. book, imprint, status, and due\_back. Each variable has at least one attribute. Each variable has a type of attribute. The types of imprint and status are String. The type of due\_back is Date. The type of book is the Book schema “type: Schema.ObjectId, ref: 'Book'”. This is an example of composition because an object has a reference to other objects. Also, book, imprint, and status variables have the required attributes with the value true. Status and due\_back variables have a default value. The default value of the status variable is “Maintenance”. The default value of due\_back variable is “Date.now”. The status variable is an enum variable that can have one of the following values “enum:['Available', 'Maintenance', 'Loaned', 'Reserved']”.



BookInstance model also has getID, due\_back\_formatted, and due\_back\_yyyy\_mm\_dd methods.







And lastly, the model is exported with the name “BookInstance”. This is how the model is accessed from the controllers.



The fourth model is the genre model. The dependencies for the model are imported first. Note that the DateTime class is not required for this model.



Schema class is imported from the mongoose package.

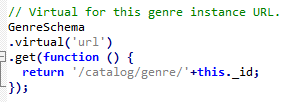


Then a new schema object is created using the constructor of the Schema class from the imported mongoose package (mongoose.Schema). The constructor is given the variables to the schema.

This model has one variable. name. The variable has multiple attributes. The type of name is String. Also, name variable has the required attributes with the value true. The name variable also has a minLength attribute with the value of 3 and a maxLength attribute with the value of 100.



BookInstance model also has getID method.



And lastly, the model is exported with the name “Genre”. This is how the model is accessed from the controllers.



**4. CONTROLLERS**

In the MCV model, the controller’s role is to get the requests from routes, obtain data from the model, combine the data with views, and return it to the client in the browser.

The bookstore application has four controllers, the first one is authorController.js.

Text

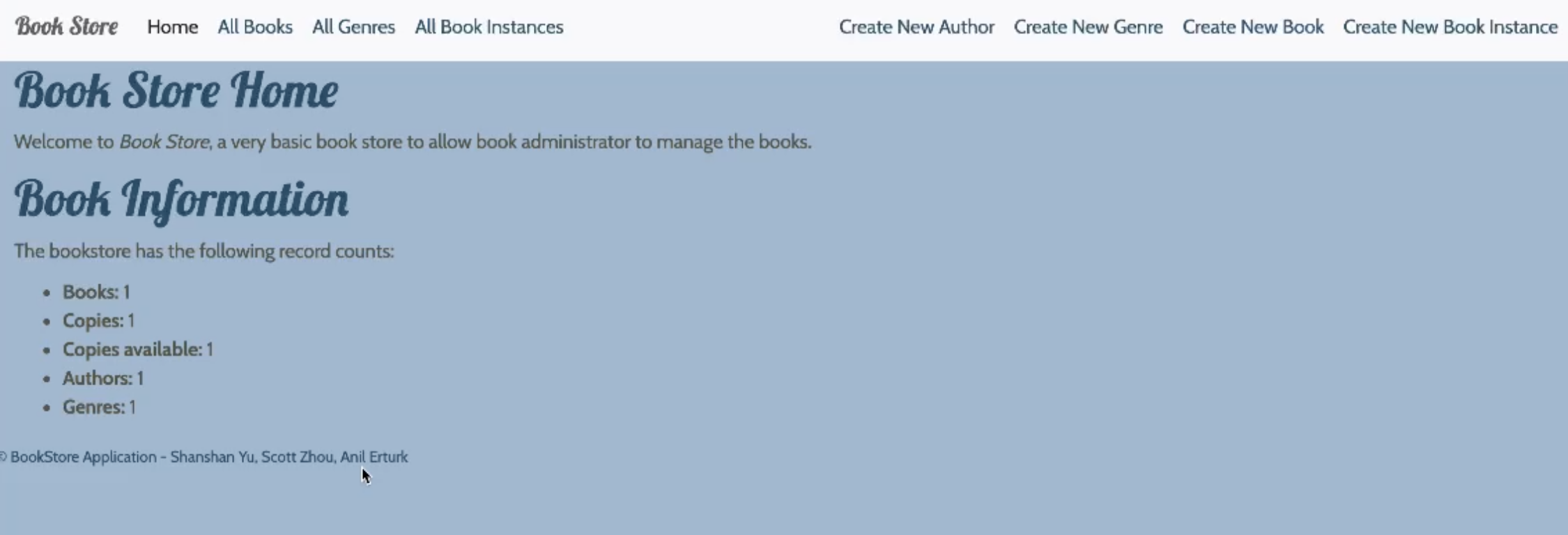
Description automatically generatedFigure 3. authorController.js

As figure 3 shows, the author controller requests author data from model “Author” and renders the author\_list to view. The CRUD functions for the author model are also embedded in the authorController.js, which allows the administrator to manage the author's information. The file also includes error trapping in the functions. If the controller cannot find information from the model, it will show “Author not found” on the browser.

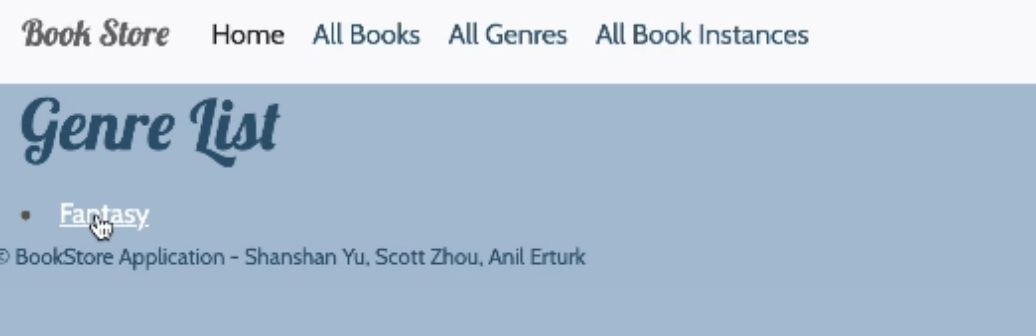
The rest of the controllers are the bookInstance controller, Genre controller, and bookController.js. As the names indicate, those controllers both contain CRUD functions but for book instances, books’ genres, and book. They also have error trapping to show the correct information for different situations.

The four controllers support the bookstore application response request. They provide back-end service of the application.

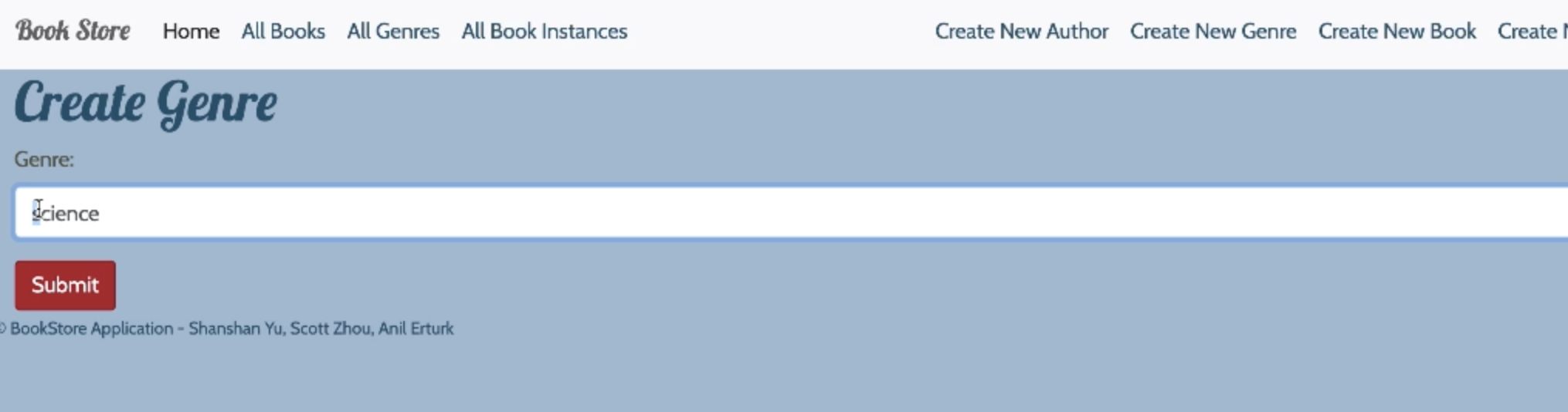
**5. DEMO**



Here is an example of our web application main page. We have an overview of all the data in the database under the Book Information section at the bottom left. The top navigation bar gives the option to view all books, all genres, and all book instances, also creating each of them respectively.



This is the interface when user click All Genres, it will show all the genre list and they are clickable in order to see the detail of each item.



When the user clicks Create New Genres, it will prompt the user to input and submit the required information. After submission, the page will direct to the confirmation page which offers two options: either update or delete the current item.

**6. REFERENCE**

Anblicks. (2020, November 7). 9 Fascinating Reasons to choose MEAN stack. https://www.anblicks.com/blog/9-reasons-why-to-choose-mean-stack/

Bootstrap. (n.d.). Bootstrap documentation. Get bootstrap. https://getbootstrap.com/docs/4.0/getting-started/introduction/

Express. (n.d.). Using template engines with Express. Expressjs.com. Retrieved from: <https://expressjs.com/en/guide/using-template-engines.html>

Hibbard, J. (2019, May 16). A Guide to the Pug HTML Template Preprocessor. SitePoint. https://www.sitepoint.com/a-beginners-guide-to-pug/

Invensis Learning. (2020, December 16). Scrum vs Kanban | Differences & Similarities Between Scrum & Kanban | Invensis Learning. Youtube. https://www.youtube.com/watch?v=pxxmSLJj8FQ

MDN Web Docs. (2021). Express Tutorial Part 4: Routes and controllers. Retrieved from: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express\_Nodejs/routes

Souza, D. (2020, June 1). Getting started with Pug. Logrocket. https://blog.logrocket.com/getting-started-with-pug/