

Introduction

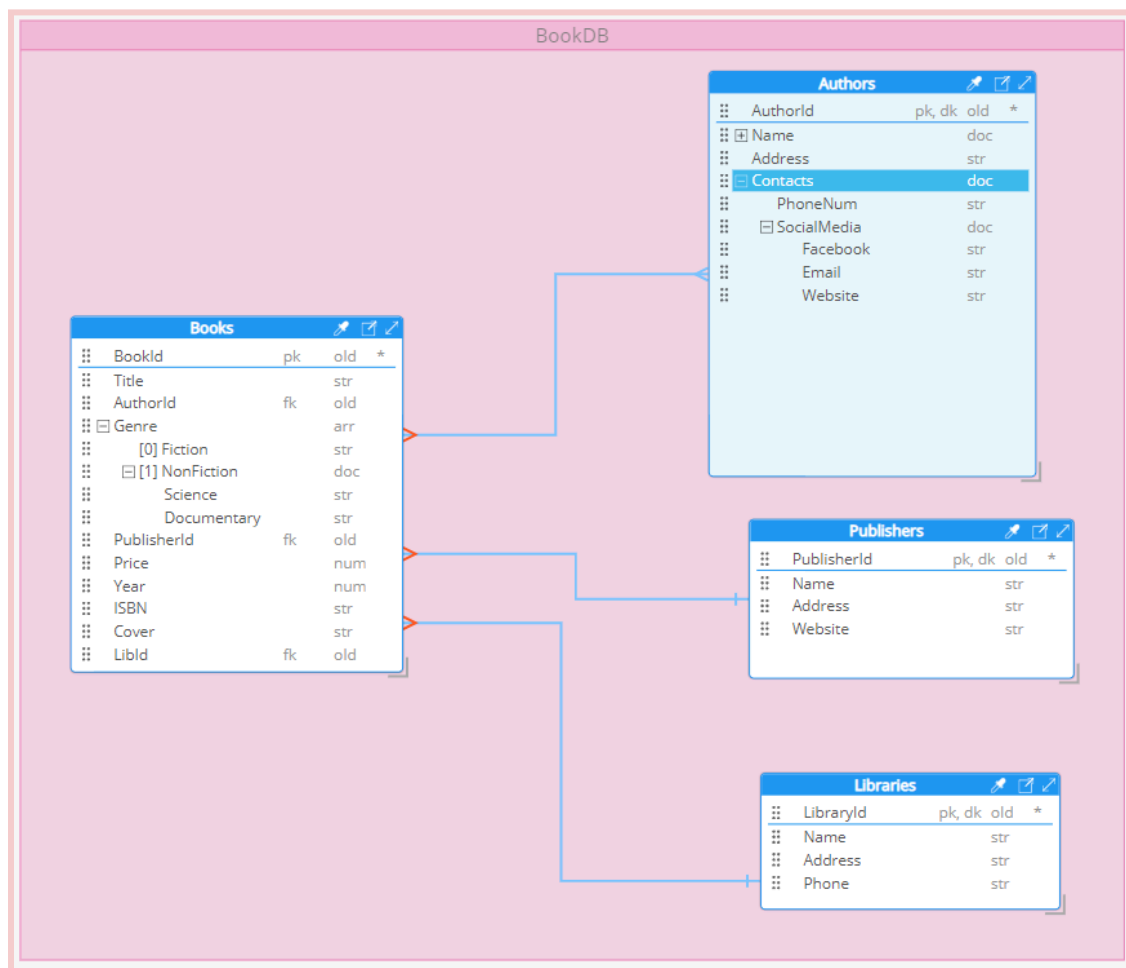
This fullstack project will serve for your own hobby or interest, which you will build over the fullstack course of the entire semester. This will be a website building from scratch that shares your hobby with the world.

→ In this case, I took a book library as my own preference.

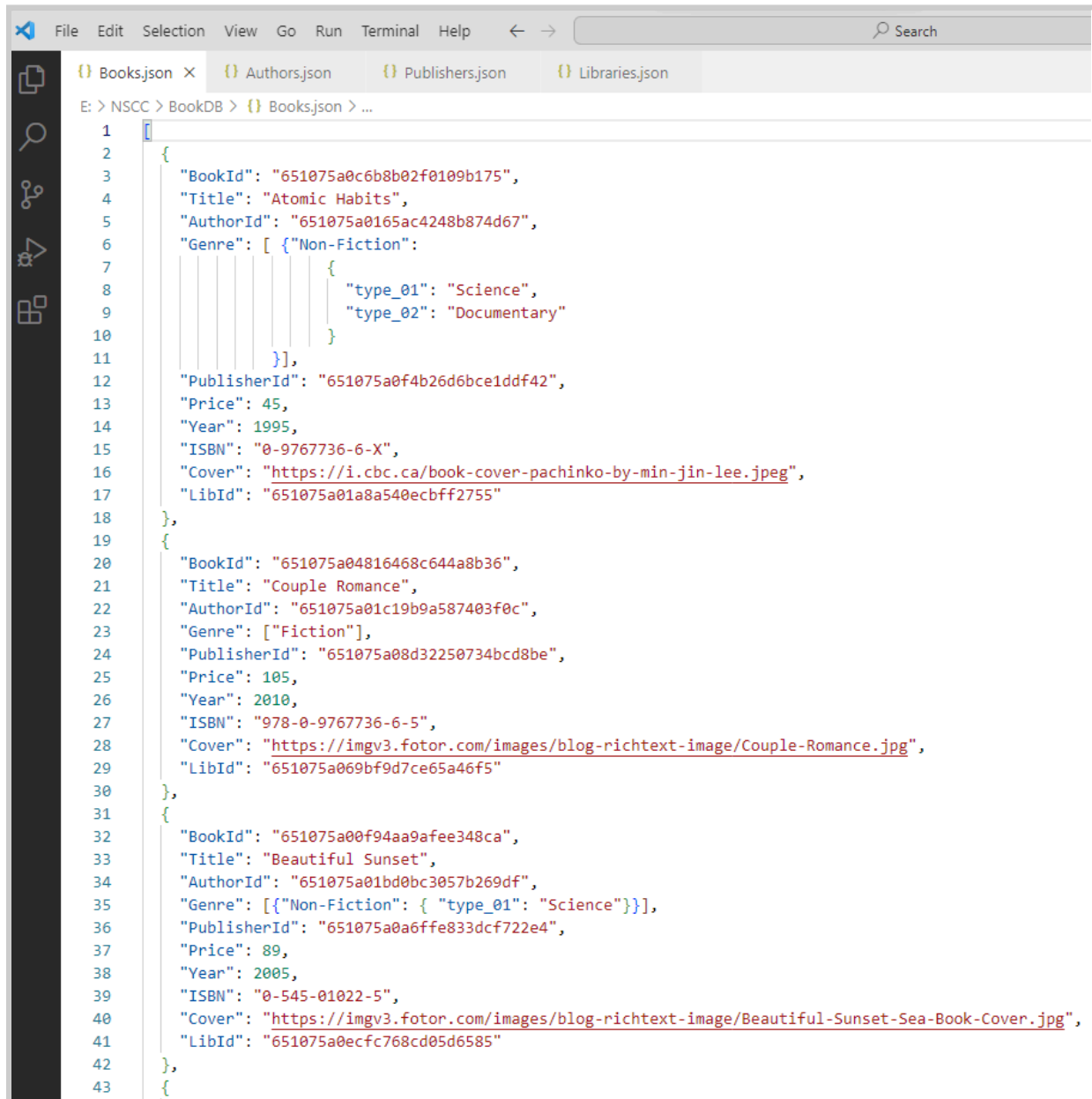
DATABASE PREPARATION

Step 1: Create ERD

- One of your fields must be of type **array**.
- One of your fields must be of type document (ie. **a nested document**).
- One field must be an image (usually an address or **path to an image**).

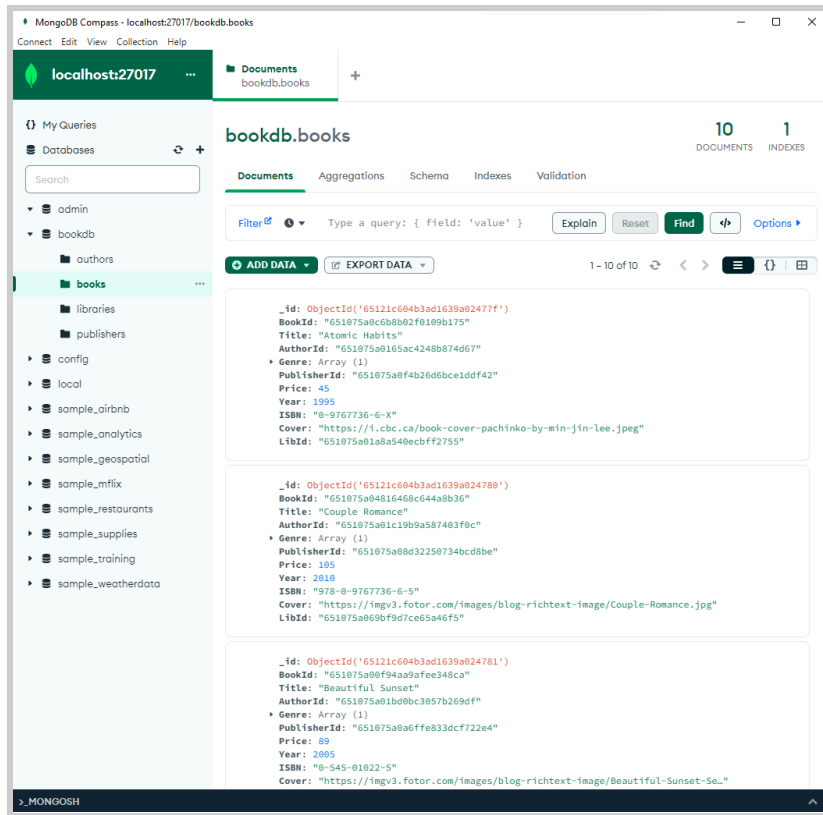


Step 2: Generate Mock Data (Json)

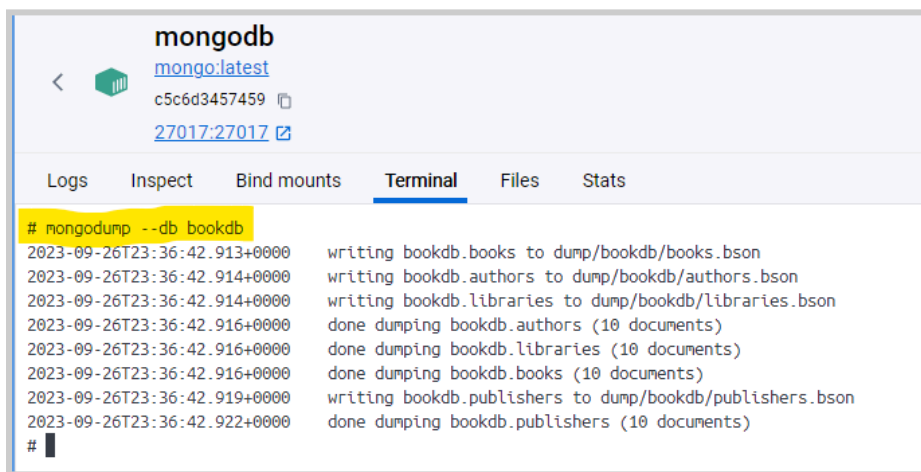


```
1  {
2    "BookId": "651075a0c6b8b02f0109b175",
3    "Title": "Atomic Habits",
4    "AuthorId": "651075a0165ac4248b874d67",
5    "Genre": [ {"Non-Fiction":
6      {
7        "type_01": "Science",
8        "type_02": "Documentary"
9      }
10   ]],
11   "PublisherId": "651075a0f4b26d6bce1ddf42",
12   "Price": 45,
13   "Year": 1995,
14   "ISBN": "0-9767736-6-X",
15   "Cover": "https://i.cbc.ca/book-cover-pachinko-by-min-jin-lee.jpeg",
16   "LibId": "651075a01a8a540ecbff2755"
17 },
18 {
19   "BookId": "651075a04816468c644a8b36",
20   "Title": "Couple Romance",
21   "AuthorId": "651075a01c19b9a587403f0c",
22   "Genre": ["Fiction"],
23   "PublisherId": "651075a08d32250734bcd8be",
24   "Price": 105,
25   "Year": 2010,
26   "ISBN": "978-0-9767736-6-5",
27   "Cover": "https://imgv3.fotor.com/images/blog-richtext-image/Couple-Romance.jpg",
28   "LibId": "651075a069bf9d7ce65a46f5"
29 },
30 {
31   "BookId": "651075a00f94aa9afee348ca",
32   "Title": "Beautiful Sunset",
33   "AuthorId": "651075a01bd0bc3057b269df",
34   "Genre": [{"Non-Fiction": { "type_01": "Science"}}],
35   "PublisherId": "651075a0a6ffe833dcf722e4",
36   "Price": 89,
37   "Year": 2005,
38   "ISBN": "0-545-01022-5",
39   "Cover": "https://imgv3.fotor.com/images/blog-richtext-image/Beautiful-Sunset-Sea-Book-Cover.jpg",
40   "LibId": "651075a0ecfc768cd05d6585"
41 },
42 {
43 }
```

Step 3: Import Data into MongoDB



Step 4: Create a backup of the data



Finally, go to [MongoDB Atlas](#) to create an account and generate a connection string for the database to go online.

→ Then start implementing the following requirements for the app:

Application Requirements

This final assignment will focus on filling out the basic functionality of your Full Stack application. You should have a working application that is secure and informative and gives details about your data. Basic client-side validation will be in place which should reflect the validation rules on the server end.

Phase 1

REQ-001 DISPLAY YOUR DATA ON THE HOMEPAGE OF YOUR FULLSTACK APP (10 PTS)

As demonstrated in class, you should make a call to your API to retrieve your custom data documents from MongoDB. This data should be displayed using the frontend template provided to you and added in class.

- You don't need to display all of your data for each item on this page. Select a subset of data to display in the card. You should however display the image you set for each item.
- From the client template provided, refactor the code to extract the JSX necessary to define a Card on this page and create a component from this JSX called *Card.js*.

REQ-002 DEMONSTRATE SOME INITIAL ROUTING FOR THE APPLICATION (10 PTS)

Using the techniques displayed in class using *react-router-dom*, add some initial routes which direct the navigation to the following areas of the application:

- / - This route should load in the *Main.js* component which in turn loads in instances of your newly created *Card.js* component
- /signin – This route should load in the *SignIn.js* component.
- /register – This route should load in a new component called *Register.js*

REQ-003 COMPLETE COMPONENT CALLED SIGNIN.JS (10 PTS.)

Your starter template provides an incomplete component called *SignIn.js*. Add the required functionality so that the component performs the following:

- Collects data as a controlled form maintaining its own state.
- Submits via either fetch or axios the data to the API.
- Stores the returned JSON web token in LocalStorage or HttpOnly Cookie.
- Redirects the user to the homepage upon successful login.
- Your component does not have to validate at this point.

REQ-004 CREATE A COMPONENT CALLED REGISTER.JS (10 PTS.)

Similar to your *SignIn.js* component, your application will need to be able to register new users. This will allow end-users to register and be added to the users database on the backend.

- This component should include a submission form that takes in all required information to register a user. Refer to your API for that required information.

- Modify your API so that in addition to returning a JSON token using the /login endpoint, you also return a token in the header for a new registration using the /register endpoint. That way a new user is automatically logged in upon registration.
- Stores the JSON web token in Local Storage or HttpOnly Cookie.
- Your component does not have to validate data at this point.

Phase 2

REQ-001 CREATE THE AUTH.JS SERVICE WHICH WILL HANDLE AUTHENTICATION RELATED FUNCTIONALITY

This will be an in-class activity to complete this requirement.

In your Full Stack application, in the client portion of the application (React), in the src folder add a new folder called *services*. In this new folder, create your own *authService.js* file and implement the logic to *login*, *logout*. Add an additional method in the service class called *register*. This will require you to communicate with your API endpoints, send credentials to these endpoints, and handle the responses from the API. This class should also be fully responsible for setting, reading and removing any JWT stored in the browser.

REQ-002 ADD A NEW COMPONENT WHICH HANDLES THE CREATION OF A NEW ITEM IN YOUR DATABASE.

Create a new controlled form component in your React App called *CreateForm.js* which will load in a form which will allow a user to fill in the details and save a new item in the database. For the purposes of this exercise, you can limit the data fields for this portion. (Many of you have defined rather large documents in your database.)

For this phase, if you have very large Models with many required fields, you might wish to disable some of the required fields if you have many in order to get the initial save functionality working. You can add fields into your form and re-enable the additional required fields at a later time.

Create a route in the application which will load in your *CreateForm* component. Demonstrate this by adding a Create New button link somewhere in your Main component which when clicked will send the user to the *CreateForm* component.

REQ-003 PROTECT CREATEFORM.JS WITH A PROTECTED ROUTE

Refer to the class recording for this requirement.

Create a new component called *ProtectedRoutes.js* in your Full Stack application which will check if the user is authenticated before allowing the user to proceed to a specified component. If the user is not authenticated, they should be redirected to the *SignIn* component page.

Once you have created the *ProtectedRoutes* component, change the route structure you created in the previous REQ from a normal route to a protected one. Demonstrate that the user is directed to the *SignIn* page if they are not already authenticated.

Phase 3

REQ-001 ADD BASIC VALIDATION TO THE EXISTING FORMS IN YOUR APPLICATION

Using the approach demonstrated in class, implement the displaying of validation errors for the existing forms in your application. These forms include *SignIn*, *Register*, and *CreateForm*.

You are free to display your validation errors in any way you wish as long as they are coherent and easily understandable.

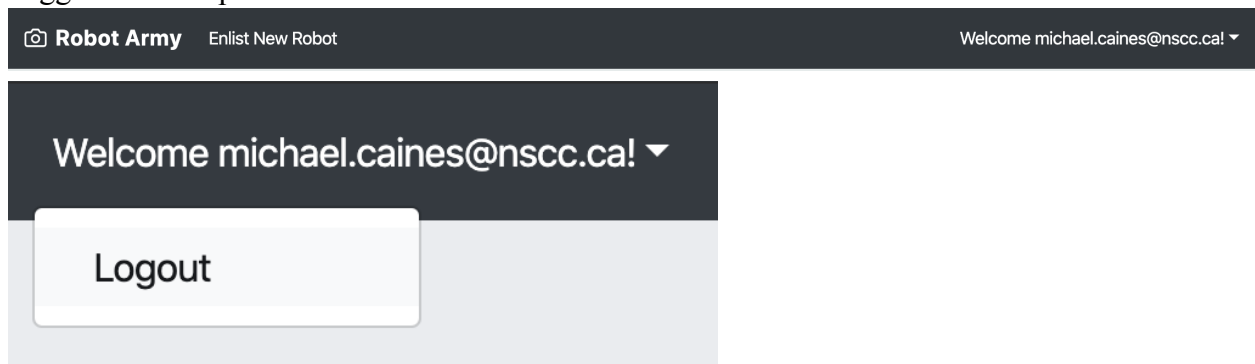
REQ-002 ADD FUNCTIONALITY TO THE NAVBAR TO SHOW CURRENT LOGGED IN/OUT STATUS.

Update the *NavBar* such that it correctly displays whether a user is currently logged in or out of the application. Use your *authService*'s *isAuthenticated* method to determine whether to show either the *Login* and *Register* links, or the *Welcome* dropdown with a *Logout* option. The *NavBar* should immediately update to reflect a changed login status.

Logged Out example...



Logged In Example...



In the client, install and use the *jwt-decode* npm package (<https://www.npmjs.com/package/jwt-decode>) to be able to access the email address encoded in your JWT. (If you're not already including the email address as part of the encrypted payload of your returned JWT, then modify your login and register API endpoints to do so).

REQ-003 ADD EDITFORM COMPONENT

Add a new component called *EditForm* which will show when any edit button is clicked. The

form should pre populate with the existing data from the database via the API.

Appropriate basic validation error display should be implemented.

REQ-004 ADD DELETE FUNCTIONALITY

Clicking on any delete button in the *Card* component should cause the application to prompt the user to confirm deletion. Upon deletion the deleted card should be removed from state and applicable cards re-rendered.

REQ-005 ADD SEARCH FUNCTIONALITY

Customize and modify the provided search form to perform basic search such that at least one field in your data is searched upon. You don't need to modify your API to accommodate this functionality. Simply add a *filter* to the already existing array mapping that you're already doing to render the cards in your Main component. Filtered results should be displayed on the main page.

You are free to decide what criteria you wish to base your search upon but there must be basic functionality applied.