# 21 MERN: Book Search Engine

## Your Task

Your assignment this week is emblematic of the fact that most modern websites are driven by two things: data and user demands. This shouldn't come as a surprise, as the ability to personalize user data is the cornerstone of real-world web development today. And as user demands evolve, applications need to be more performant.

This week, you'll take starter code with a fully functioning Google Books API search engine built with a RESTful API, and refactor it to be a GraphQL API built with Apollo Server. The app was built using the MERN stack with a React front end, MongoDB database, and Node.js/Express.js server and API. It's already set up to allow users to save book searches to the back end.

To complete the assignment, you'll need to do the following:

1. Set up an Apollo Server to use GraphQL queries and mutations to fetch and modify data, replacing the existing RESTful API.

2. Modify the existing authentication middleware so that it works in the context of a GraphQL API.

3. Create an Apollo Provider so that requests can communicate with an Apollo Server.

4. Deploy your application to Heroku with a MongoDB database using MongoDB Atlas. Use the Deploy with Heroku and MongoDB Atlas walkthrough for instructions.

## User Story

```
AS AN avid reader
I WANT to search for new books to read
SO THAT I can keep a list of books to purchase
```

## Acceptance Criteria

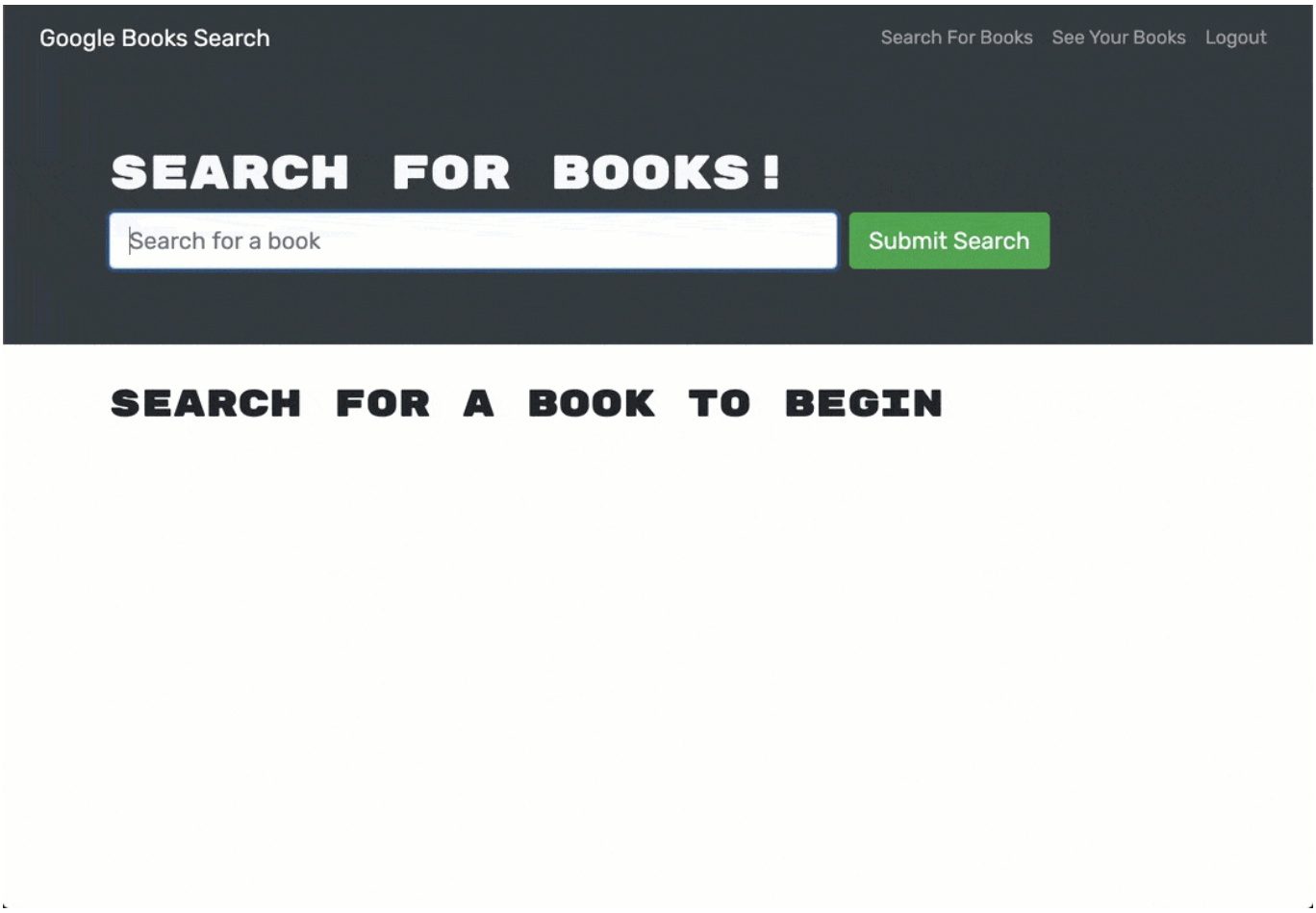
```
GIVEN a book search engine
WHEN I load the search engine
THEN I am presented with a menu with the options Search for Books and Login/Signup
and an input field to search for books and a submit button
WHEN I click on the Search for Books menu option
THEN I am presented with an input field to search for books and a submit button
WHEN I am not logged in and enter a search term in the input field and click the
submit button
THEN I am presented with several search results, each featuring a book's title,
author, description, image, and a link to that book on the Google Books site
WHEN I click on the Login/Signup menu option
THEN a modal appears on the screen with a toggle between the option to log in or
sign up
```

```
WHEN the toggle is set to Signup
THEN I am presented with three inputs for a username, an email address, and a
password, and a signup button
WHEN the toggle is set to Login
THEN I am presented with two inputs for an email address and a password and login
button
WHEN I enter a valid email address and create a password and click on the signup
button
THEN my user account is created and I am logged in to the site
WHEN I enter my account’s email address and password and click on the login button
THEN I the modal closes and I am logged in to the site
WHEN I am logged in to the site
THEN the menu options change to Search for Books, an option to see my saved books,
and Logout
WHEN I am logged in and enter a search term in the input field and click the
submit button
THEN I am presented with several search results, each featuring a book’s title,
author, description, image, and a link to that book on the Google Books site and a
button to save a book to my account
WHEN I click on the Save button on a book
THEN that book’s information is saved to my account
WHEN I click on the option to see my saved books
THEN I am presented with all of the books I have saved to my account, each
featuring the book’s title, author, description, image, and a link to that book on
the Google Books site and a button to remove a book from my account
WHEN I click on the Remove button on a book
THEN that book is deleted from my saved books list
WHEN I click on the Logout button
THEN I am logged out of the site and presented with a menu with the options Search
for Books and Login/Signup and an input field to search for books and a submit
button
```

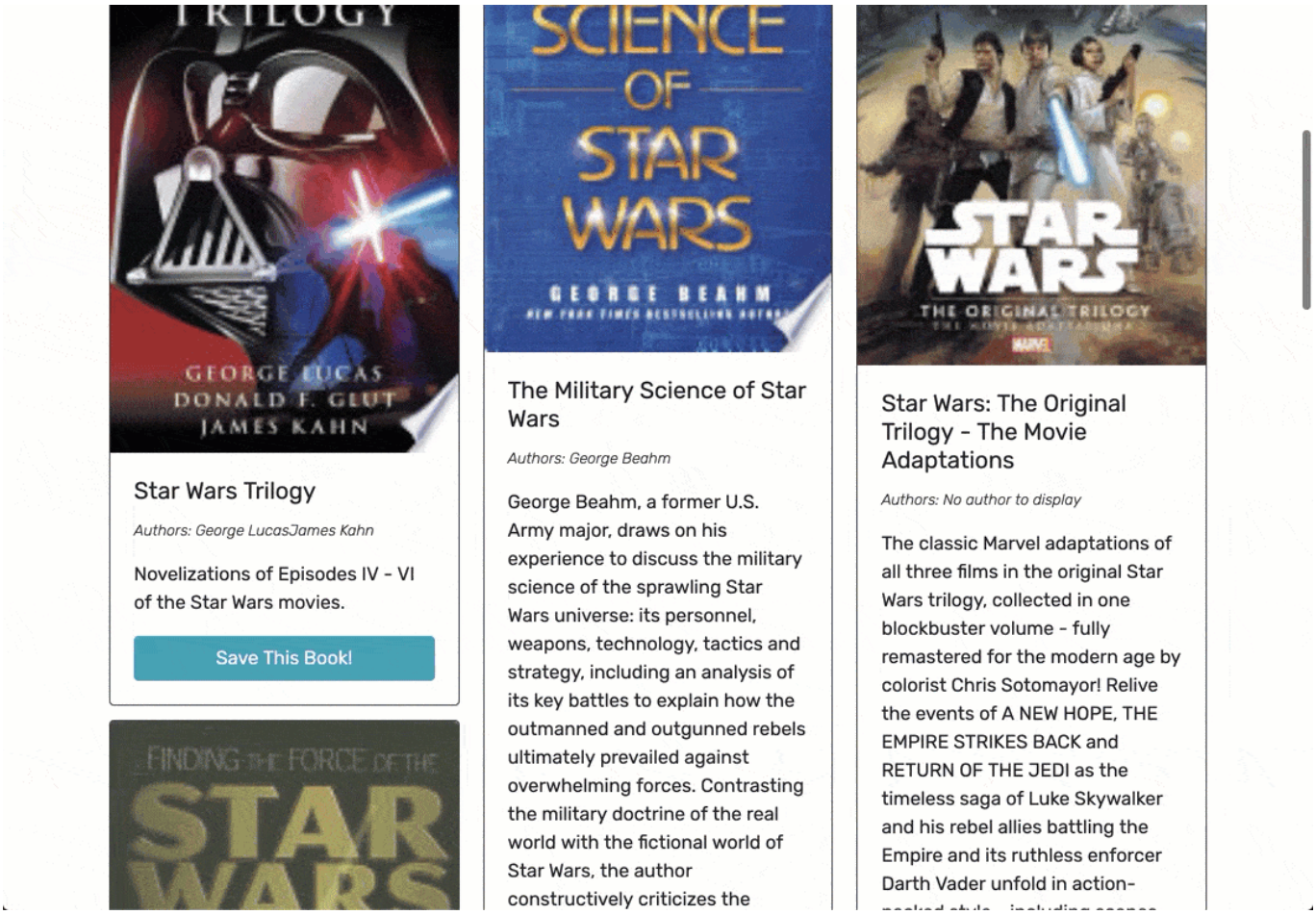
## Mock-Up

Let's start by revisiting the web application's appearance and functionality.

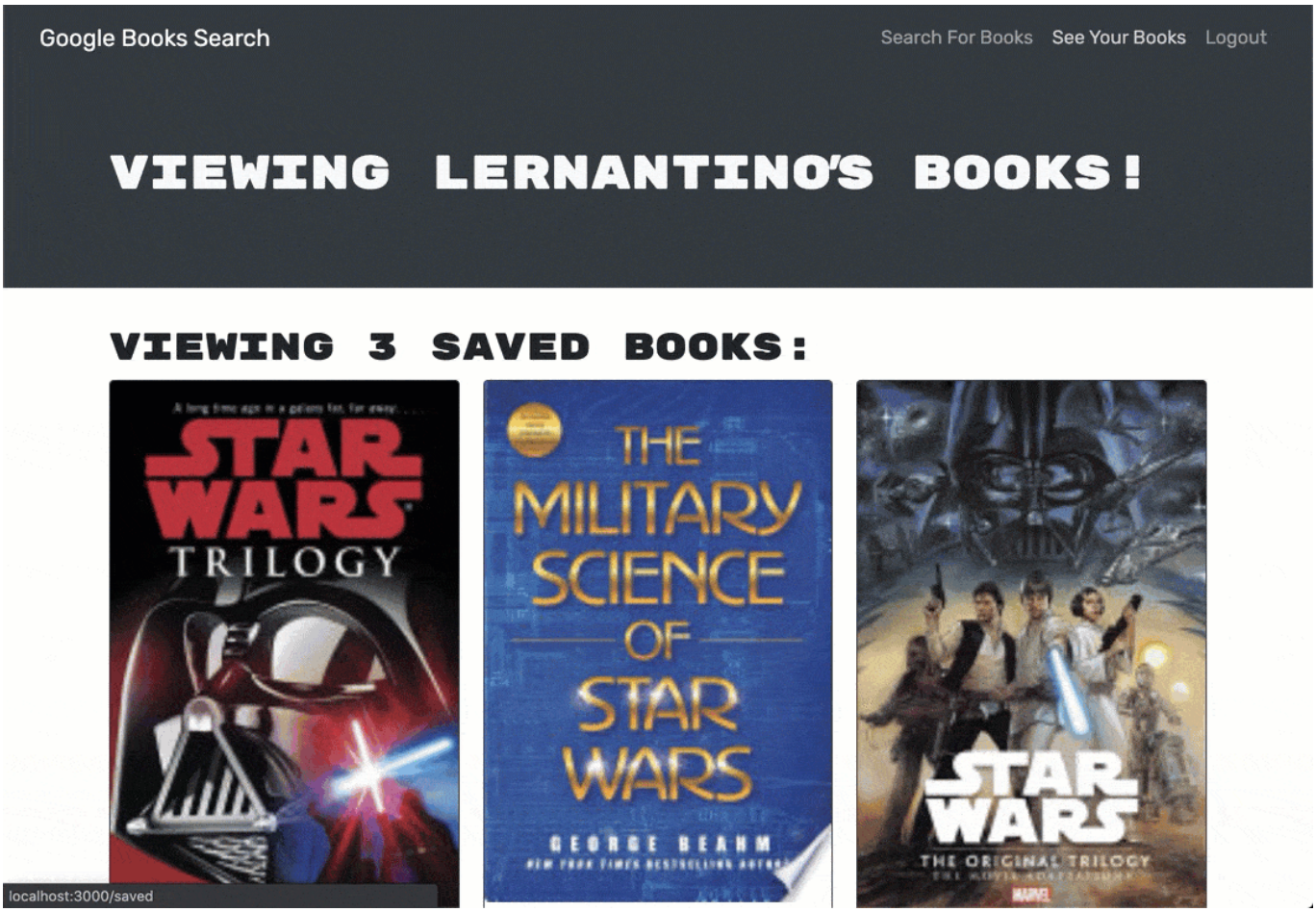
As you can see in the following animation, a user can type a search term (in this case, "star wars") in a search box and the results appear:




The user can save books by clicking "Save This Book!" under each search result, as shown in the following animation:



A user can view their saved books on a separate page, as shown in the following animation:




## Getting Started

In order for this application to use a GraphQL API, you'll need to refactor the API to use GraphQL on the back end and add some functionality to the front end. The following sections contain details about the files you'll need to modify on the back end and the front end.

**Important**: Make sure to study the application before building upon it. Better yet, start by making a copy of it. It's already a working application—you're converting it from RESTful API practices to a GraphQL API.

### Back-End Specifications

You'll need to complete the following tasks in each of these back-end files:

- `auth.js`: Update the auth middleware function to work with the GraphQL API.
- `server.js`: Implement the Apollo Server and apply it to the Express server as middleware.
- `Schemas` directory:
    - `index.js`: Export your typeDefs and resolvers.
    - `resolvers.js`: Define the query and mutation functionality to work with the Mongoose models.

      **Hint**: Use the functionality in the `user-controller.js` as a guide.
    - `typeDefs.js`: Define the necessary `Query` and `Mutation` types:

- `Query` type:
  - `me`: Which returns a `User` type.
- `Mutation` type:
  - `login`: Accepts an email and password as parameters; returns an `Auth` type.
  - `addUser`: Accepts a username, email, and password as parameters; returns an `Auth` type.
  - `saveBook`: Accepts a book author's array, description, title, bookId, image, and link as parameters; returns a `User` type. (Look into creating what's known as an `input` type to handle all of these parameters!)
  - `removeBook`: Accepts a book's `bookId` as a parameter; returns a `User` type.
- `User` type:
  - `_id`
  - `username`
  - `email`
  - `bookCount`
  - `savedBooks` (This will be an array of the `Book` type.)
- `Book` type:
  - `bookId` (Not the `_id`, but the book's `id` value returned from Google's Book API.)
  - `authors` (An array of strings, as there may be more than one author.)
  - `description`
  - `title`
  - `image`
  - `link`
- `Auth` type:
  - `token`
  - `user` (References the `User` type.)

## Front-End Specifications

You'll need to create the following front-end files:

- `queries.js`: This will hold the query `GET_ME`, which will execute the `me` query set up using Apollo Server.

- `mutations.js`:
  - `LOGIN_USER` will execute the `loginUser` mutation set up using Apollo Server.
  - `ADD_USER` will execute the `addUser` mutation.
  - `SAVE_BOOK` will execute the `saveBook` mutation.
  - `REMOVE_BOOK` will execute the `removeBook` mutation.

Additionally, you'll need to complete the following tasks in each of these front-end files:

- `App.js`: Create an Apollo Provider to make every request work with the Apollo Server.
- `SearchBooks.js`:
  - Use the Apollo `useMutation()` Hook to execute the `SAVE_BOOK` mutation in the `handleSaveBook()` function instead of the `saveBook()` function imported from the `API` file.
  - Make sure you keep the logic for saving the book's ID to state in the `try...catch` block!
- `SavedBooks.js`:
  - Remove the `useEffect()` Hook that sets the state for `UserData`.
  - Instead, use the `useQuery()` Hook to execute the `GET_ME` query on load and save it to a variable named `userData`.
  - Use the `useMutation()` Hook to execute the `REMOVE_BOOK` mutation in the `handleDeleteBook()` function instead of the `deleteBook()` function that's imported from `API` file. (Make sure you keep the `removeBookId()` function in place!)
- `SignupForm.js`: Replace the `addUser()` functionality imported from the `API` file with the `ADD_USER` mutation functionality.
- `LoginForm.js`: Replace the `loginUser()` functionality imported from the `API` file with the `LOGIN_USER` mutation functionality.

## Grading Requirements

> **Note**: If a Challenge assignment submission is marked as "0", it is considered incomplete and will not count towards your graduation requirements. Examples of incomplete submissions include the following:
>
> - A repository that has no code
> - A repository that includes a unique name but nothing else
> - A repository that includes only a README file but nothing else
> - A repository that only includes starter code

This Challenge is graded based on the following criteria:

### Technical Acceptance Criteria: 40%

- Satisfies all of the preceding acceptance criteria plus the following:
    - Has an Apollo Server that uses GraphQL queries and mutations to fetch and modify data, replacing the existing RESTful API.
    - Use an Apollo Server and apply it to the Express.js server as middleware.
    - Include schema settings for resolvers and typeDefs as outlined in the Challenge instructions.
    - Modify the existing authentication middleware to work in the context of a GraphQL API.
    - Use an Apollo Provider so that the application can communicate with the Apollo Server.
    - Application must be deployed to Heroku.

### Deployment: 32%

- Application deployed at live URL.
- Application loads with no errors.
- Application GitHub URL submitted.
- GitHub repository contains application code.

### Application Quality: 15%

- User experience is intuitive and easy to navigate.
- User interface style is clean and polished.
- Application resembles the mock-up functionality provided in the Challenge instructions.

### Repository Quality: 13%

- Repository has a unique name.
- Repository follows best practices for file structure and naming conventions.
- Repository follows best practices for class/id naming conventions, indentation, quality comments, etc.
- Repository contains multiple descriptive commit messages.
- Repository contains high-quality README file with description, screenshot, and link to the deployed application.

## Review

You are required to submit BOTH of the following for review:

- The URL of the functional, deployed application.

- The URL of the GitHub repository. Give the repository a unique name and include a README describing the project.

---