DATA-236 Sec 12 - Distributed Systems for Data Engineering
HOMEWORK 3

Nandhakumar Apparsamy
018190003

GitHub -

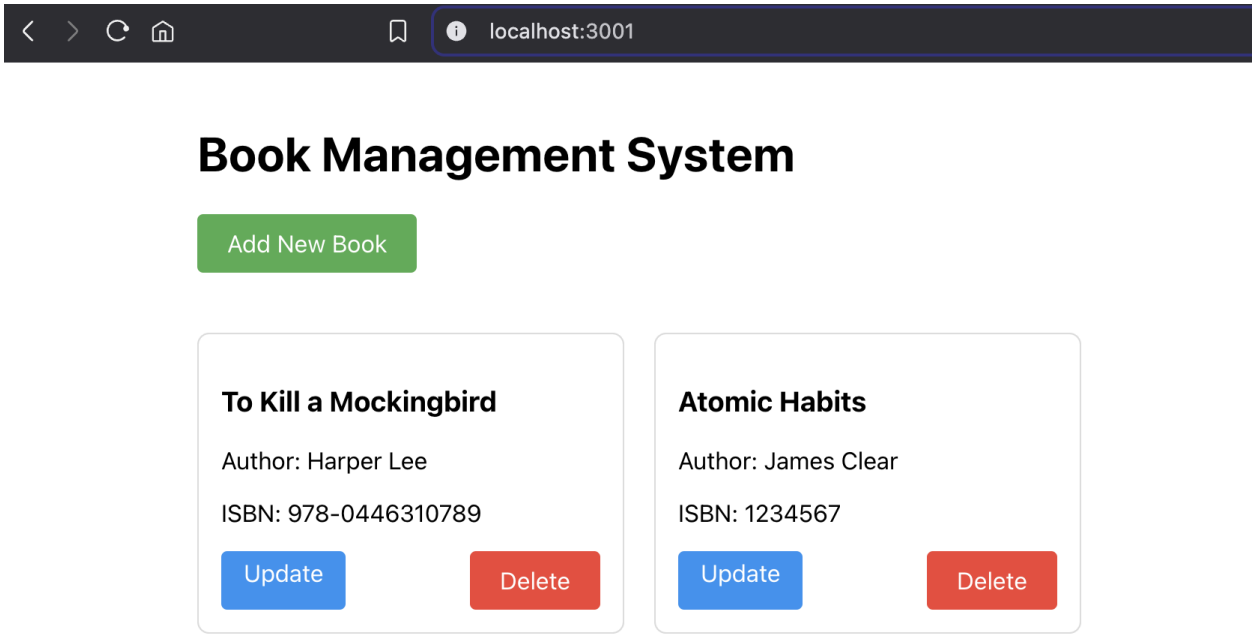
Q1. React (10 Points)

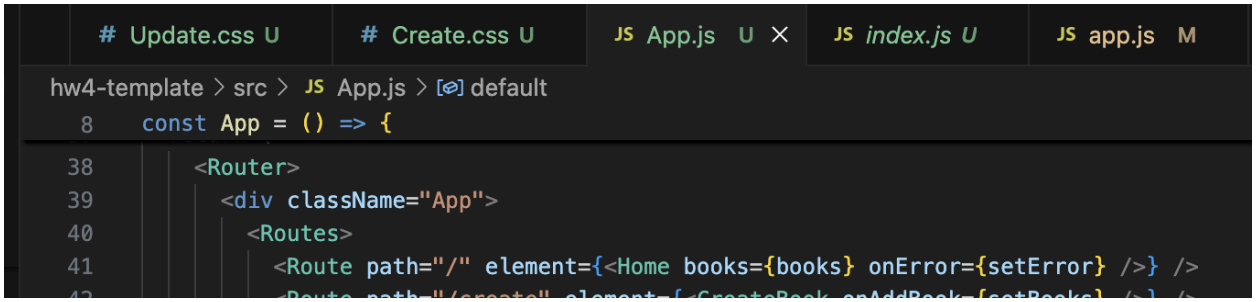
Create a Book Management App where users can add, update, and delete books.

I. Home Page **(1 Point)**

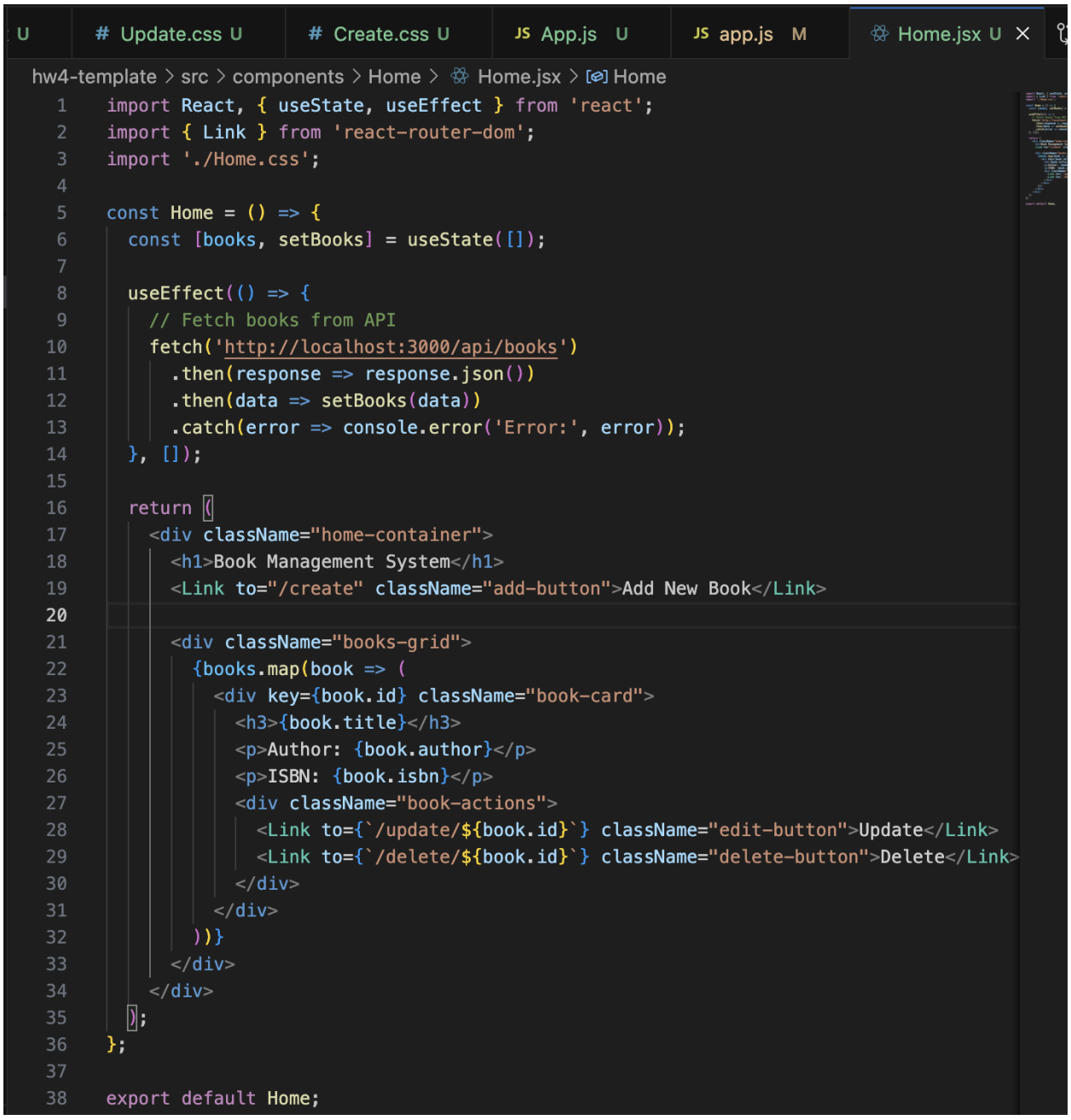
Write the necessary code to create the 'Home.jsx' component which will display all the books.

The Home component should be rendered when the user is on the root route '/'.




Update.css U
Create.css U
JS App.js U
JS index.js U
JS app.js M
hw4-template > src > JS App.js > default
const App = () => {
<Router>

<Routes>
<Route path="/" element={<Home books={books} onError={setError} />} />


U
Update.css U
Create.css U
JS App.js U
JS app.js M
Home.jsx U
hw4-template > src > components > Home > Home.jsx > Home
import React, { useState, useEffect } from 'react';
import { Link } from 'react-router-dom';
import './Home.css';
const Home = () => {
const [books, setBooks] = useState([]);
useEffect(() => {
// Fetch books from API
fetch('http://localhost:3000/api/books')
.then(response => response.json())
.then(data => setBooks(data))
.catch(error => console.error('Error:', error));
}, []);
return (

<h1>Book Management System</h1>
<Link to="/create" className="add-button">Add New Book</Link>

{books.map(book => (

<h3>{book.title}</h3>
Author: {book.author}
ISBN: {book.isbn}

<Link to={`/update/${book.id}`} className="edit-button">Update</Link>
<Link to={`/delete/${book.id}`} className="delete-button">Delete</Link>

))}

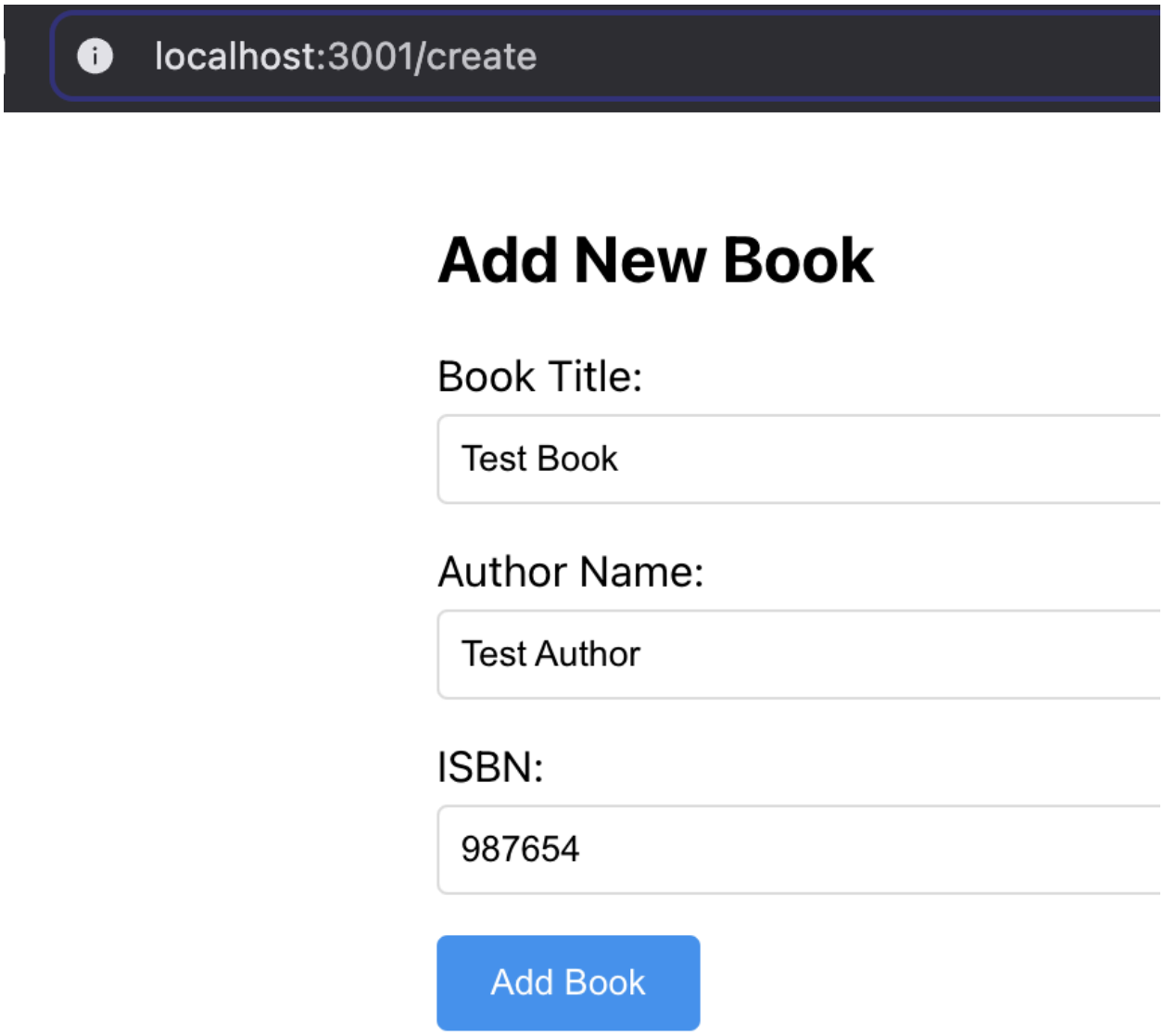
);
};
export default Home;

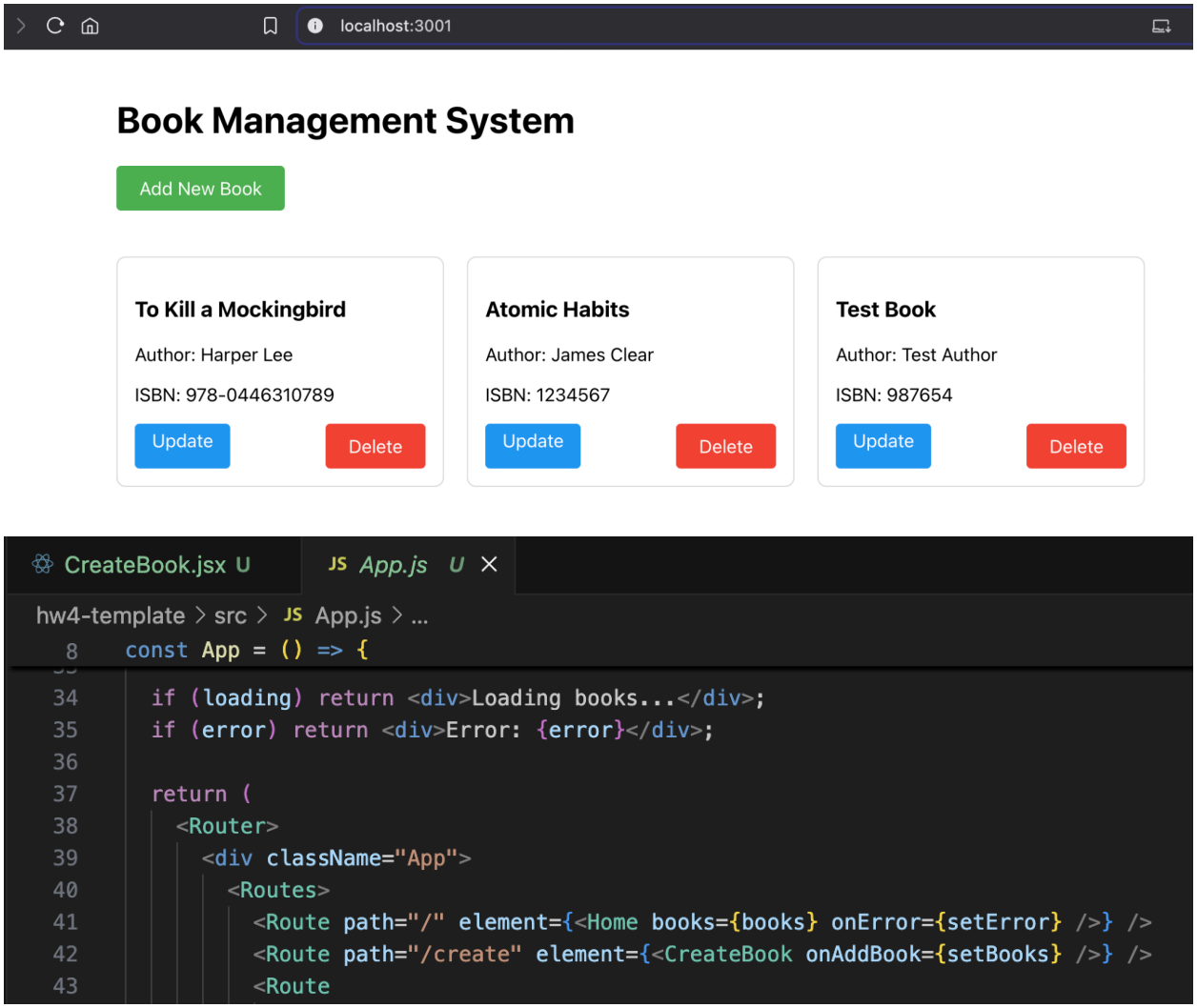
II. Add a New Book **(2 Points)**

Write the necessary code to create the ‘CreateBook.jsx’ component. The component should be rendered when the user is on the ‘/create’ route.

The component should accept props (similar to demo) to add the new book and have the following:

- An input field for entering the Book Title.
- An input field for entering the Author Name.
- A submit button labeled "Add Book". When the user clicks the Add book button:
- A new book should be added to the book list with an auto-incremented book ID.
- The user should be redirected to the home page to see the updated book list.





CreateBook.jsx U App.js U

hw4-template > src > App.js > ...

```
8   const App = () => {
34    if (loading) return <div>Loading books...</div>;
35    if (error) return <div>Error: {error}</div>;
36
37    return (
38      <Router>
39        <div className="App">
40          <Routes>
41            <Route path="/" element={<Home books={books} onError={setError} />} />
42            <Route path="/create" element={<CreateBook onAddBook={setBooks} />} />
43            <Route
```

CreateBook.jsx U | App.js U

hw4-template > src > components > Create > CreateBook.jsx > ...

```jsx
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import './CreateBook.css';

const CreateBook = ({ onAddBook }) => {
  const [title, setTitle] = useState('');
  const [author, setAuthor] = useState('');
  const [isbn, setIsbn] = useState('');
  const [error, setError] = useState(null);
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError(null);

    try {
      const response = await fetch('http://localhost:3000/api/books', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify({
          title,
          author,
          isbn
        }),
      });

      if (!response.ok) {
        throw new Error('Network response was not ok');
      }

      const newBook = await response.json();
      onAddBook(newBook);
      navigate('/');
    } catch (error) {
      setError('Failed to create book. Please try again.');
      console.error('Error:', error);
    }
  };
```

CreateBook.jsx U | App.js U

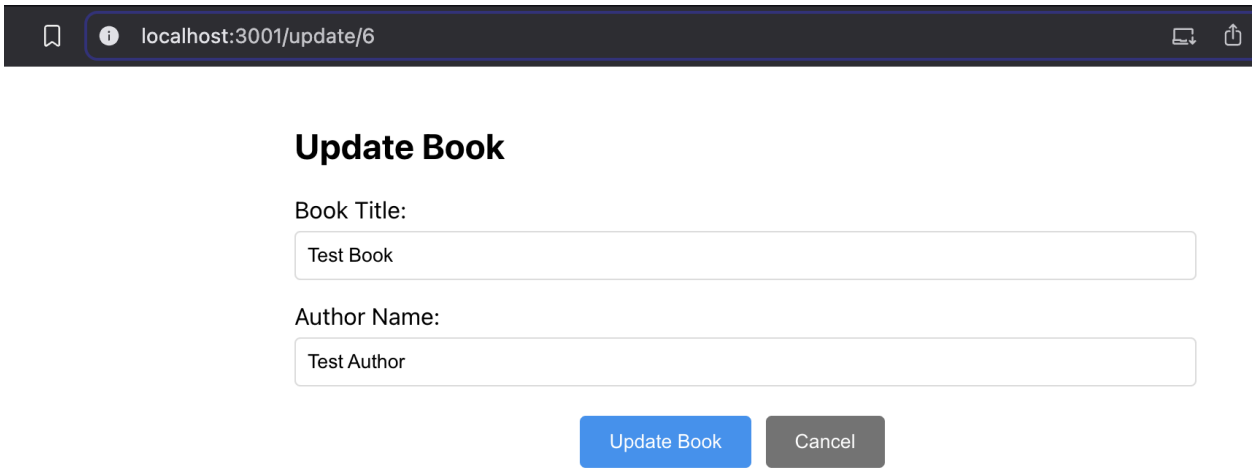
hw4-template > src > components > Create > CreateBook.jsx > ...

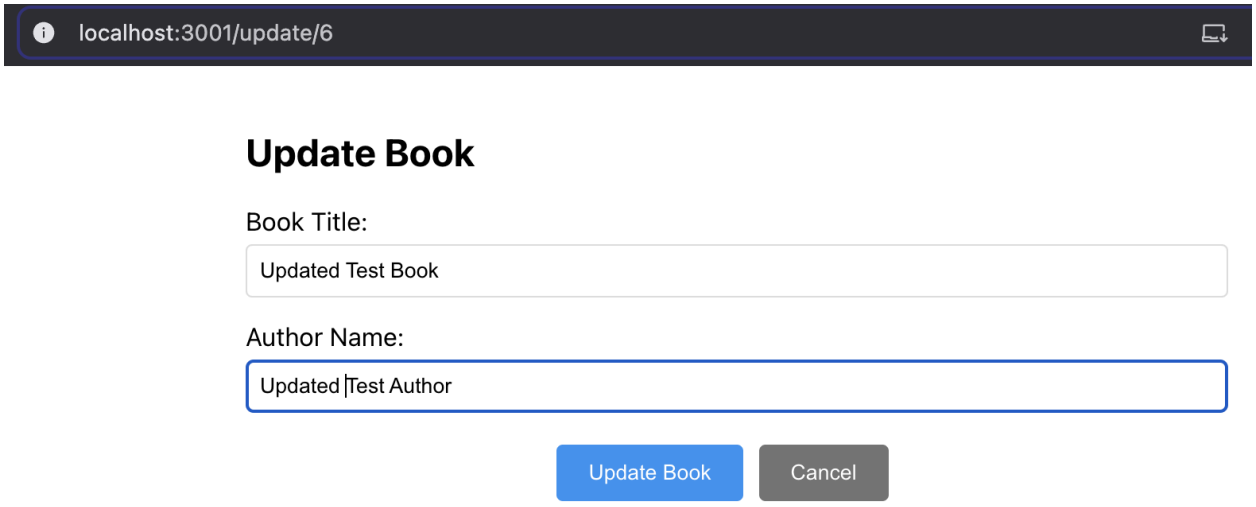
```jsx
  return (
    <div className="create-book-container">
      <h2>Add New Book</h2>
      {error && <div className="error-message">{error}</div>}
      <form onSubmit={handleSubmit}>
        <div className="form-group">
          <label htmlFor="title">Book Title:</label>
          <input
            type="text"
            id="title"
            value={title}
            onChange={(e) => setTitle(e.target.value)}
            required
          />
        </div>
        <div className="form-group">
          <label htmlFor="author">Author Name:</label>
          <input
            type="text"
            id="author"
            value={author}
            onChange={(e) => setAuthor(e.target.value)}
            required
          />
        </div>
        <div className="form-group">
          <label htmlFor="isbn">ISBN:</label>
          <input
            type="text"
            id="isbn"
            value={isbn}
            onChange={(e) => setIsbn(e.target.value)}
          />
        </div>
        <button type="submit" className="submit-button">Add Book</button>
      </form>
    </div>
  );
};

```

III. Update Book **(2 Points)**
Write the necessary code to create the ‘UpdateBook.jsx’ component. The component should be rendered when the user is on the ‘/update’ route.
The component should accept props (similar to demo) to update the new book and have the following:

- An input field for Book Title.
- An input field for Author Name.
- A submit button labeled "Update Book".

When the user clicks the Update Book button the book within context should be updated and the user should be redirected to the home page showing updated book list.





Book Management System

Add New Book

To Kill a Mockingbird

Author: Harper Lee

ISBN: 978-0446310789

Update Delete

Atomic Habits

Author: James Clear

ISBN: 1234567

Update Delete

Updated Test Book

Author: Updated Test Author

ISBN: 987654

Update Delete

UpdateBook.jsx U

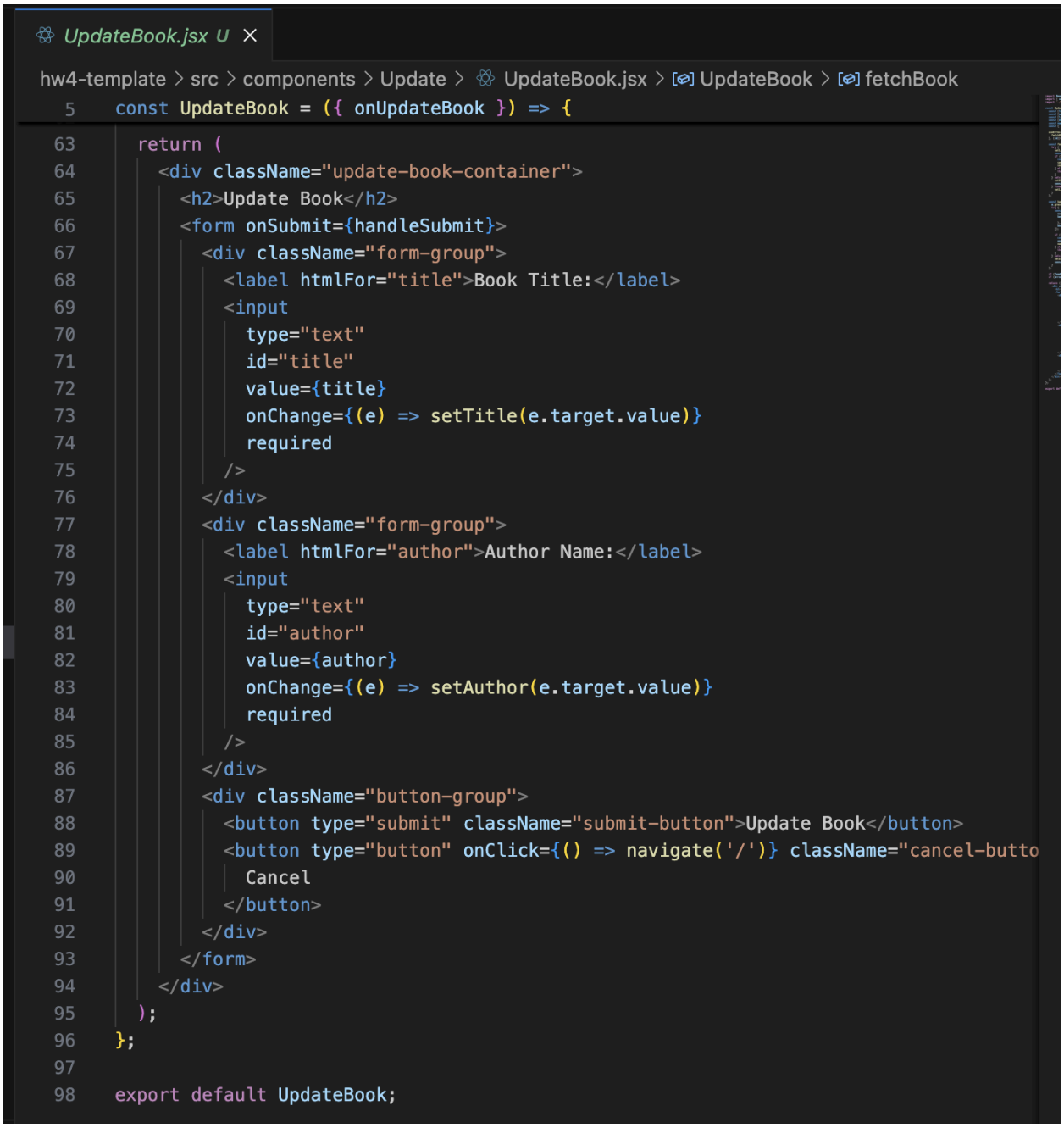
hw4-template > src > components > Update > UpdateBook.jsx > UpdateBook > fetchBook

```jsx
import React, { useState, useEffect } from 'react';
import { useNavigate, useParams } from 'react-router-dom';
import './UpdateBook.css';

const UpdateBook = ({ onUpdateBook }) => {
  const [title, setTitle] = useState('');
  const [author, setAuthor] = useState('');
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const navigate = useNavigate();
  const { id } = useParams();

  useEffect(() => {
    fetchBook();
  }, [id]);

  const fetchBook = async () => {
    try {
      setLoading(true);
      const response = await fetch(`http://localhost:3000/api/books/${id}`);
      if (response.ok) {
        const book = await response.json();
        setTitle(book.title);
        setAuthor(book.author);
      } else {
        throw new Error('Book not found');
      }
    } catch (error) {
      setError(error.message);
      console.error('Error fetching book:', error);
    } finally {
      setLoading(false);
    }
  };
```

UpdateBook.jsx U

hw4-template > src > components > Update > UpdateBook.jsx > UpdateBook > fetchBook

```
5   const UpdateBook = ({ onUpdateBook }) => {
36    const handleSubmit = async (e) => {
37      e.preventDefault();
38      try {
39        const response = await fetch(`http://localhost:3000/api/books/${id}`, {
40          method: 'PUT',
41          headers: {
42            'Content-Type': 'application/json',
43          },
44          body: JSON.stringify({ title, author }),
45        });
46
47        if (response.ok) {
48          const updatedBook = await response.json();
49          onUpdateBook && onUpdateBook(updatedBook);
50          navigate('/');
51        } else {
52          throw new Error('Failed to update book');
53        }
54      } catch (error) {
55        setError(error.message);
56        console.error('Error:', error);
57      }
58    };
59
60    if (loading) return <div>Loading...</div>;
61    if (error) return <div className="error-message">{error}</div>;
62
```



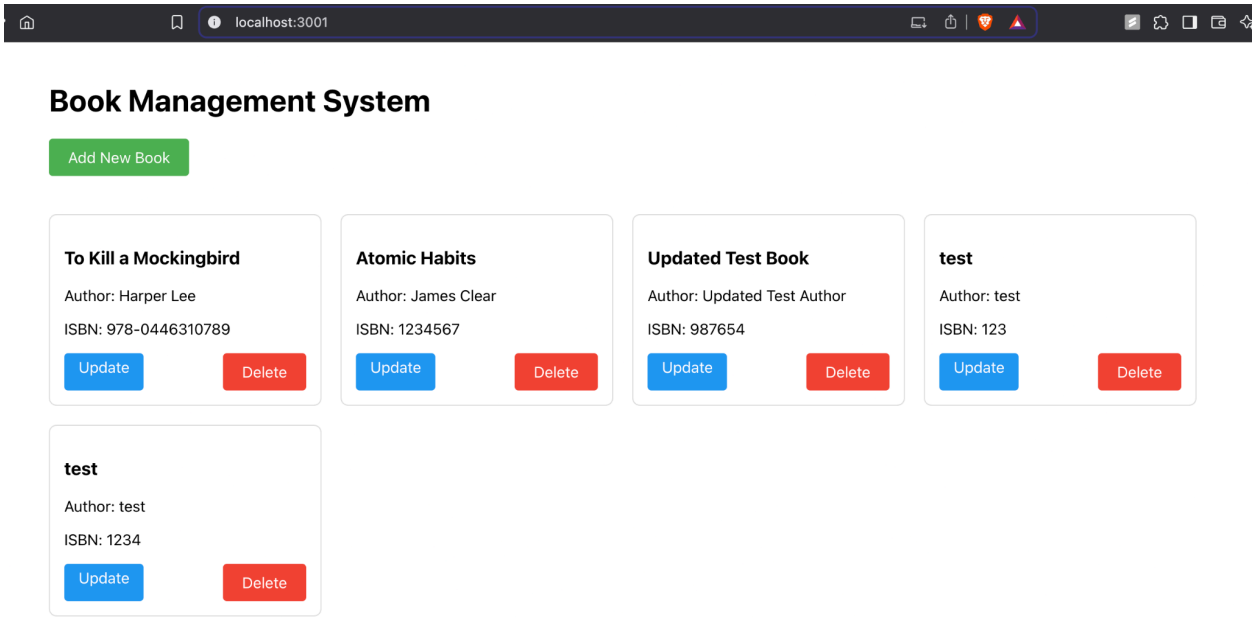
IV . Delete the Book **(2 Points)**

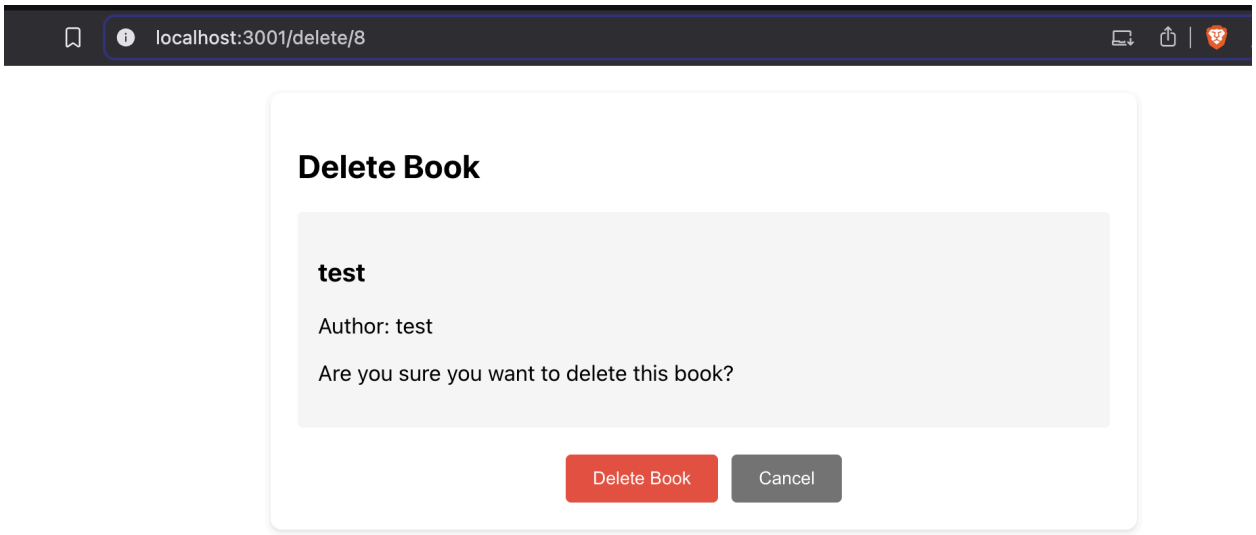
Write the necessary code to create the ‘DeleteBook.jsx’ component. The component should be rendered when the user is on the ‘/delete’ route.

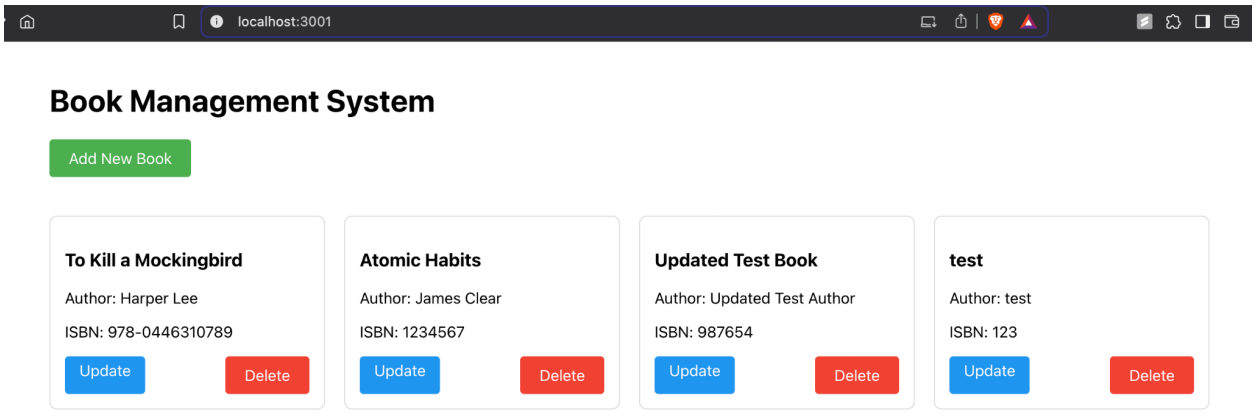
The component should accept props (similar to demo) to delete a book and have the following:

● A button labeled "Delete Book".

When the user clicks the Delete Book button the book in context should be deleted and the user should be redirected to the home page showing the updated book list.


localhost:3001
Book Management System
Add New Book
To Kill a Mockingbird
Author: Harper Lee
ISBN: 978-0446310789
Update
Delete
Atomic Habits
Author: James Clear
ISBN: 1234567
Update
Delete
Updated Test Book
Author: Updated Test Author
ISBN: 987654
Update
Delete
test
Author: test
ISBN: 123
Update
Delete
test
Author: test
ISBN: 1234
Update
Delete

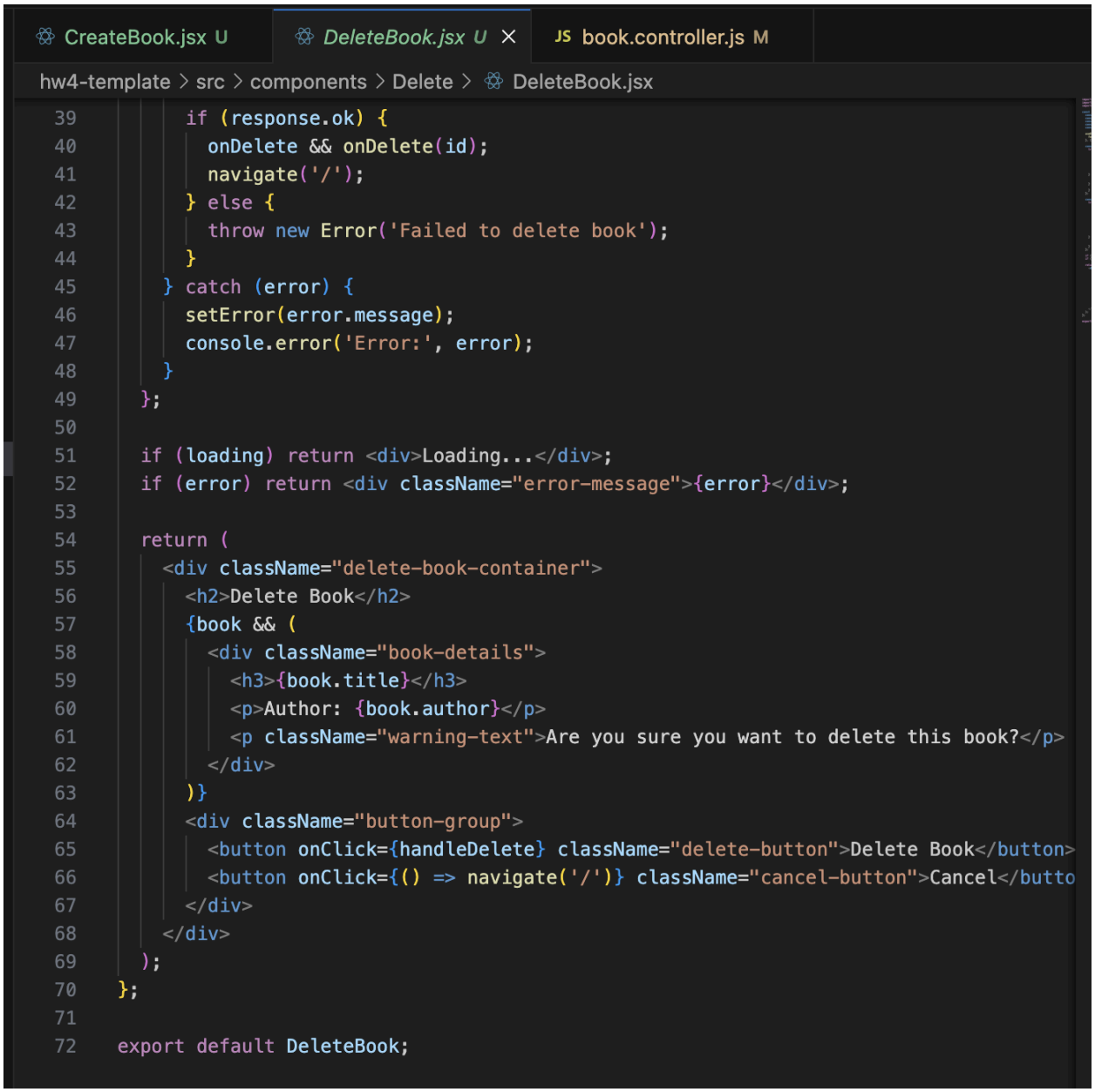

localhost:3001/delete/8
Delete Book
test
Author: test
Are you sure you want to delete this book?
Delete Book
Cancel


localhost:3001
Book Management System
Add New Book
To Kill a Mockingbird
Author: Harper Lee
ISBN: 978-0446310789
Update
Delete
Atomic Habits
Author: James Clear
ISBN: 1234567
Update
Delete
Updated Test Book
Author: Updated Test Author
ISBN: 987654
Update
Delete
test
Author: test
ISBN: 123
Update
Delete

CreateBook.jsx U | DeleteBook.jsx U | book.controller.js M

hw4-template > src > components > Delete > DeleteBook.jsx

```jsx
import React, { useState, useEffect } from 'react';
import { useNavigate, useParams } from 'react-router-dom';
import './DeleteBook.css';

const DeleteBook = ({ onDelete }) => {
  const [book, setBook] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const navigate = useNavigate();
  const { id } = useParams();

  useEffect(() => {
    fetchBookDetails();
  }, [id]);

  const fetchBookDetails = async () => {
    try {
      const response = await fetch(`http://localhost:3000/api/books/${id}`);
      if (response.ok) {
        const data = await response.json();
        setBook(data);
      } else {
        throw new Error('Book not found');
      }
    } catch (error) {
      setError(error.message);
      console.error('Error fetching book:', error);
    } finally {
      setLoading(false);
    }
  };

  const handleDelete = async () => {
    try {
      const response = await fetch(`http://localhost:3000/api/books/${id}`, {
        method: 'DELETE',
      });
```



V . For all the above questions make sure to:

● Pass props for the Create, Update, and Delete components. **(1 Point)**

```jsx
<Router>
  <div className="App">
    <Routes>
      <Route path="/" element={<Home books={books} onError={setError} />} />
      <Route path="/create" element={<CreateBook onAddBook={setBooks} />} />
      <Route
        path="/update/:id"
        element={<UpdateBook books={books} setBooks={setBooks} />}
      />
      <Route
        path="/delete/:id"
        element={<DeleteBook books={books} setBooks={setBooks} />}
      />
    </Routes>
  </div>
</Router>
```

- Use hooks like useState and useEffect wherever necessary. **(1 Point)**

```jsx
const App = () => {
  const [books, setBooks] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    fetchBooks();
  }, []);

  const fetchBooks = async () => {
    try {
      setLoading(true);
      const response = await fetch('http://localhost:3000/api/books');
      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
      }
      const data = await response.json();
      setBooks(data);
    } catch (error) {
      setError('Failed to fetch books. Please ensure the server is running.');
      console.error('Error:', error);
    } finally {
      setLoading(false);
    }
  };
```

```
const CreateBook = ({ onAddBook }) => {
  const [title, setTitle] = useState('');
  const [author, setAuthor] = useState('');
  const [isbn, setIsbn] = useState('');
  const [error, setError] = useState(null);
  const navigate = useNavigate();
```

```
const UpdateBook = ({ onUpdateBook }) => {
  const [title, setTitle] = useState('');
  const [author, setAuthor] = useState('');
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const navigate = useNavigate();
  const { id } = useParams();

  useEffect(() => {
    fetchBook();
  }, [id]);
```

```
const DeleteBook = ({ onDelete }) => {
  const [book, setBook] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const navigate = useNavigate();
  const { id } = useParams();

  useEffect(() => {
    fetchBookDetails();
  }, [id]);
```

- User React-Router-Dom for routing. **(1 Point)**

Home.jsx U

hw4-template > src > components > Home > Home.jsx

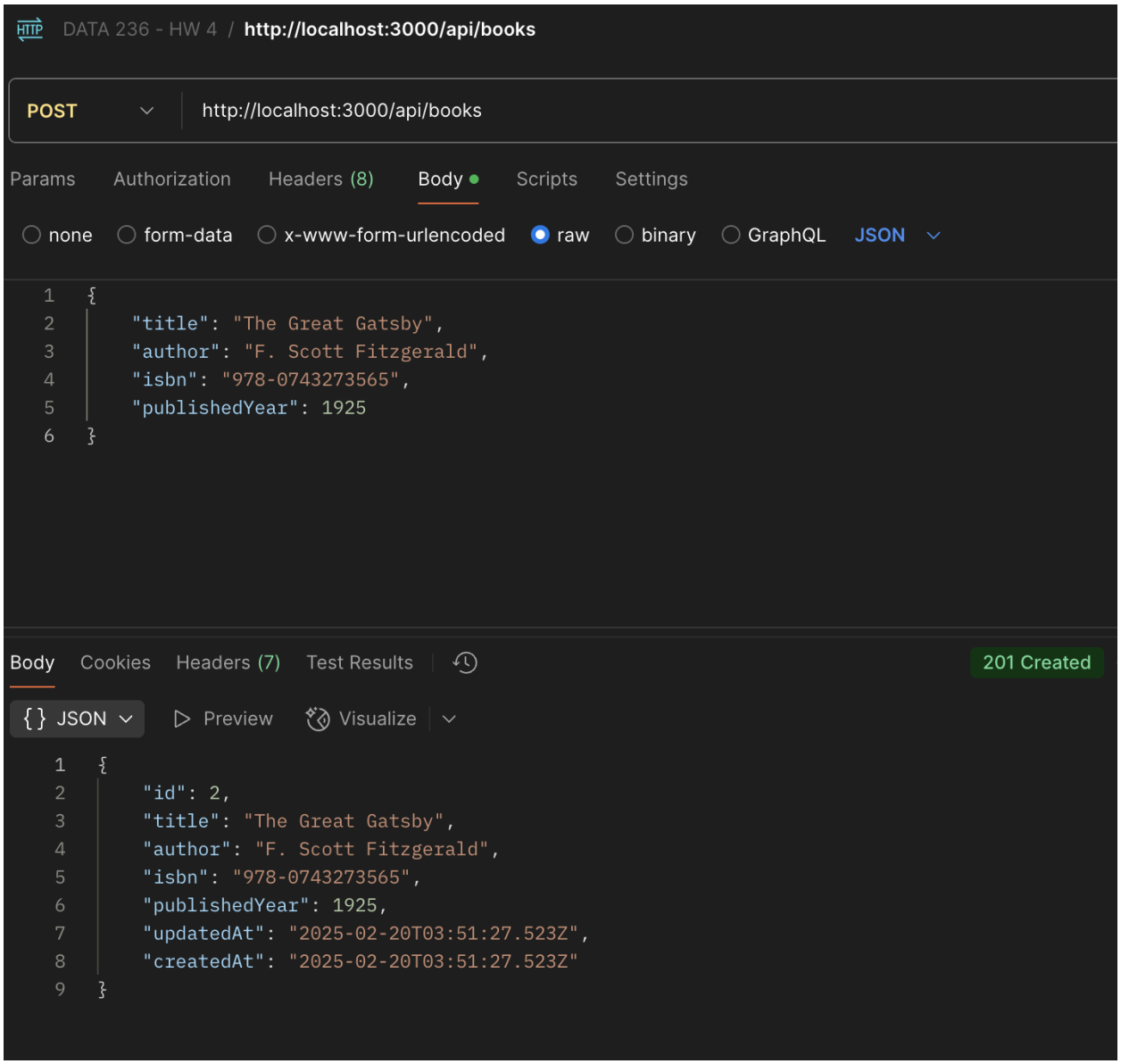
```jsx
import React, { useState, useEffect } from 'react';
import { Link } from 'react-router-dom';
import './Home.css';

const Home = () => {
  const [books, setBooks] = useState([]);

  useEffect(() => {
    // Fetch books from API
    fetch('http://localhost:3000/api/books')
      .then(response => response.json())
      .then(data => setBooks(data))
      .catch(error => console.error('Error:', error));
  }, []);

  return (
    <div className="home-container">
      <h1>Book Management System</h1>
      <Link to="/create" className="add-button">Add New Book</Link>

      <div className="books-grid">
        {books.map(book => (
          <div key={book.id} className="book-card">
            <h3>{book.title}</h3>
            <p>Author: {book.author}</p>
            <p>ISBN: {book.isbn}</p>
            <div className="book-actions">
              <Link to={`/update/${book.id}`} className="edit-button">Update</Link>
              <Link to={`/delete/${book.id}`} className="delete-button">Delete</Link>
            </div>
          </div>
        ))}
      </div>
    </div>
  );
};

export default Home;
```

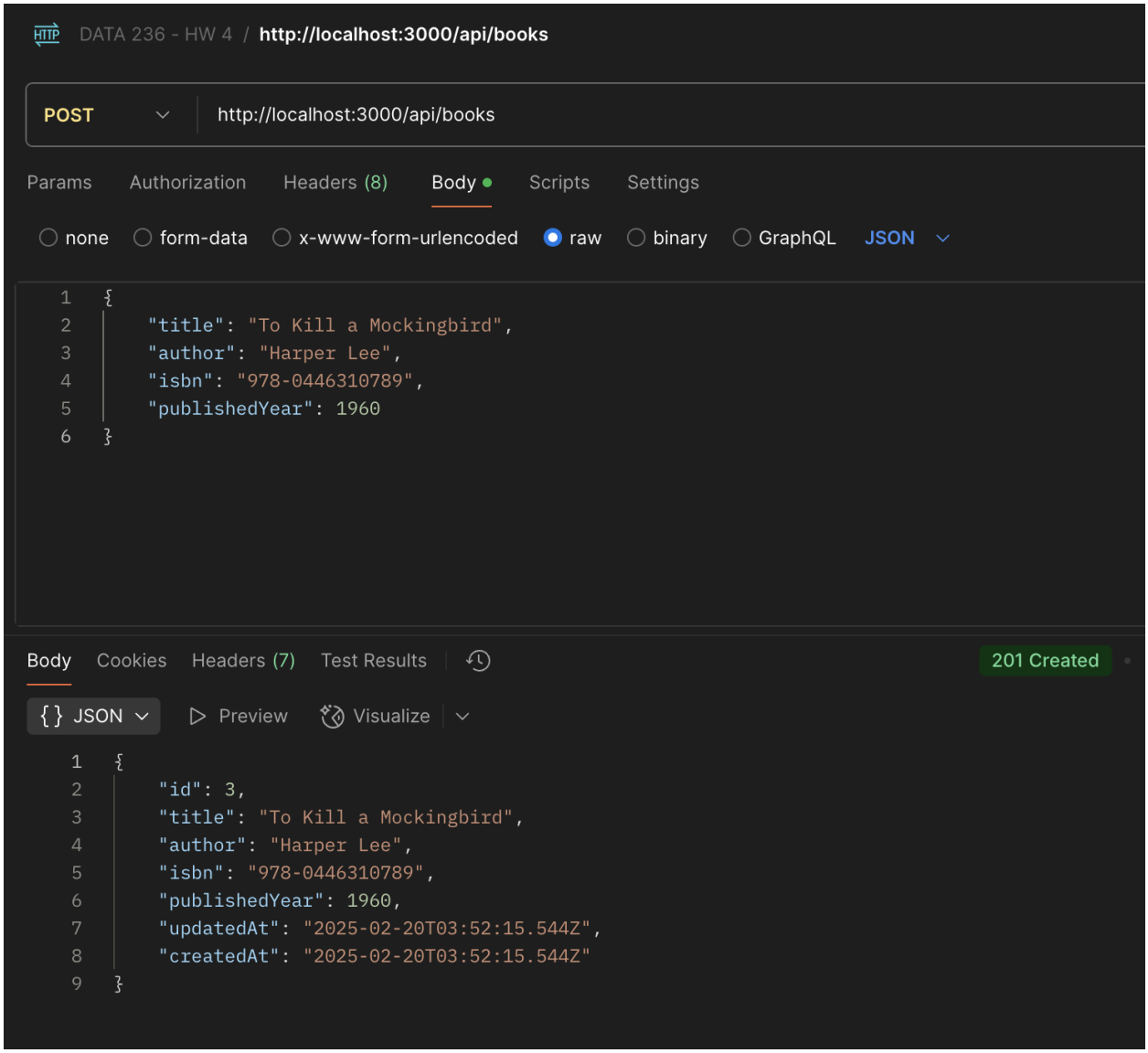
Q2. MySQL (10 points)

In this question, you will apply the concepts learned in class to build a simple CRUD (Create, Read, Update, Delete) application using Node.js and MySQL using Sequelize
Create an appropriate book Model, *View(Optional*), and Controller.

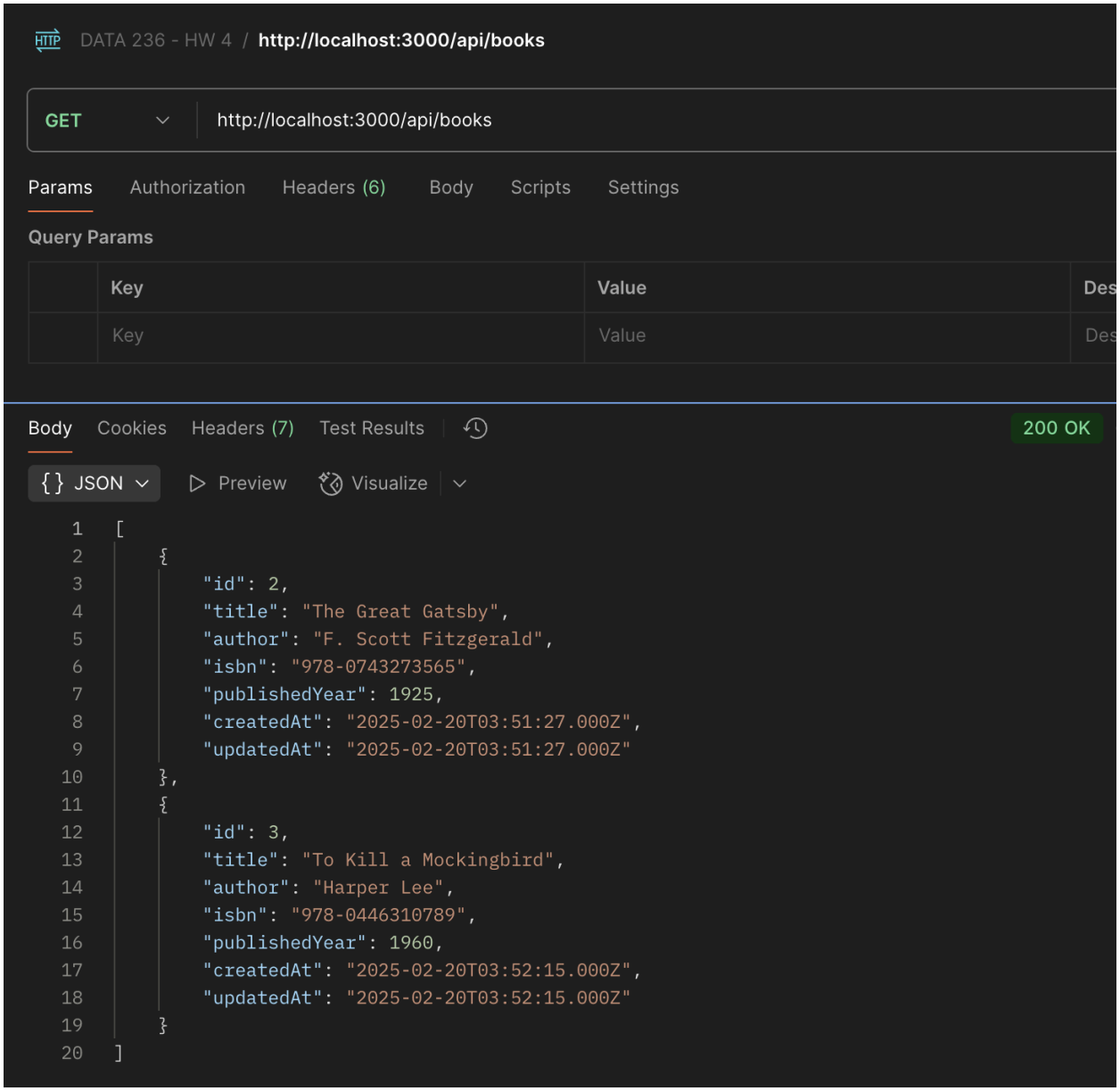
You are required to create a Book Management System that allows users to:

- Add a new book (POST)

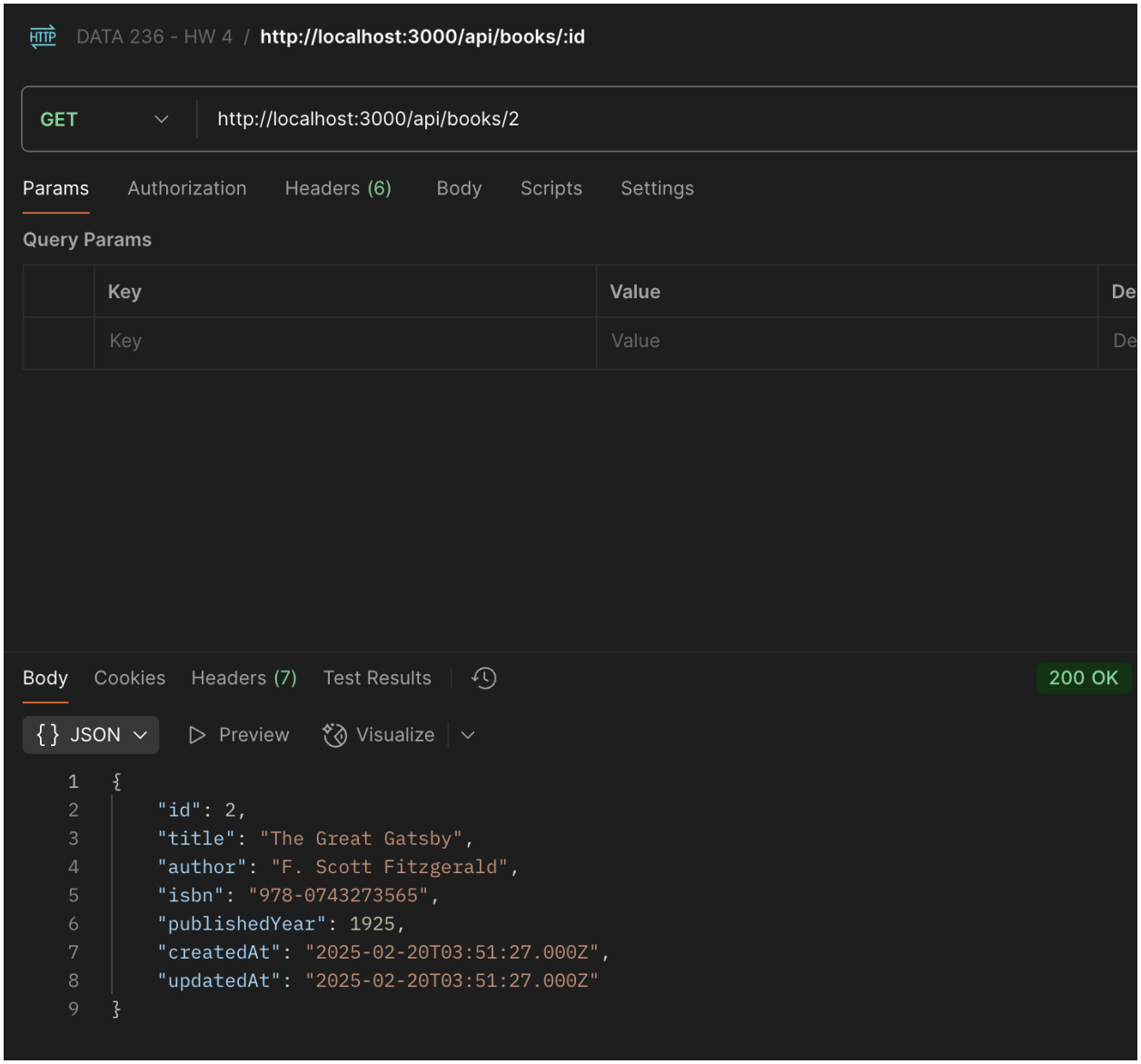



DATA 236 - HW 4 / http://localhost:3000/api/books
POST
http://localhost:3000/api/books
Params
Authorization
Headers (8)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"title": "To Kill a Mockingbird",
"author": "Harper Lee",
"isbn": "978-0446310789",
"publishedYear": 1960
}
Body
Cookies
Headers (7)
Test Results
201 Created
JSON
Preview
Visualize
{
"id": 3,
"title": "To Kill a Mockingbird",
"author": "Harper Lee",
"isbn": "978-0446310789",
"publishedYear": 1960,
"updatedAt": "2025-02-20T03:52:15.544Z",
"createdAt": "2025-02-20T03:52:15.544Z"
}

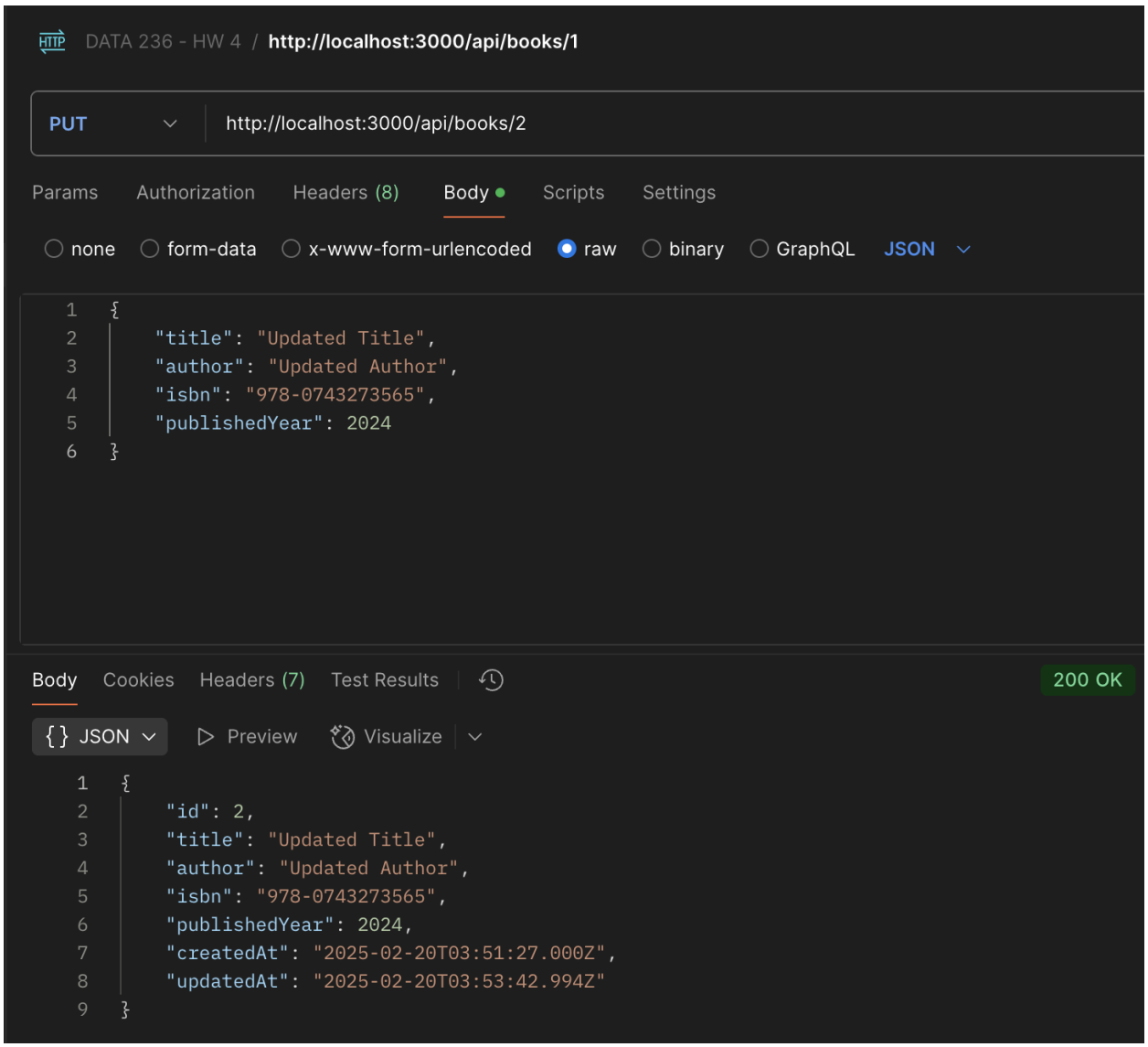
- View all books (GET)

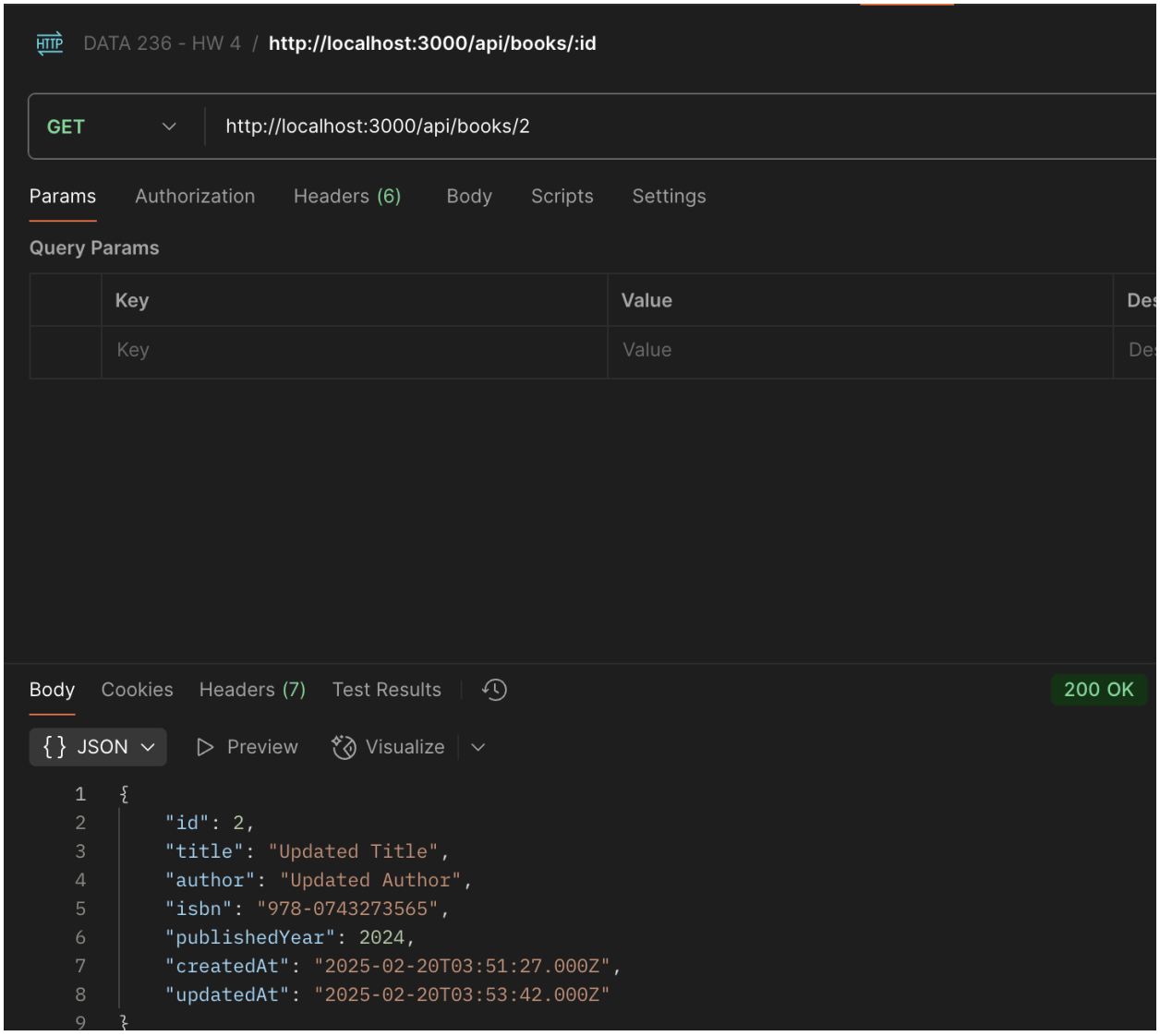


- View a book by ID (GET)

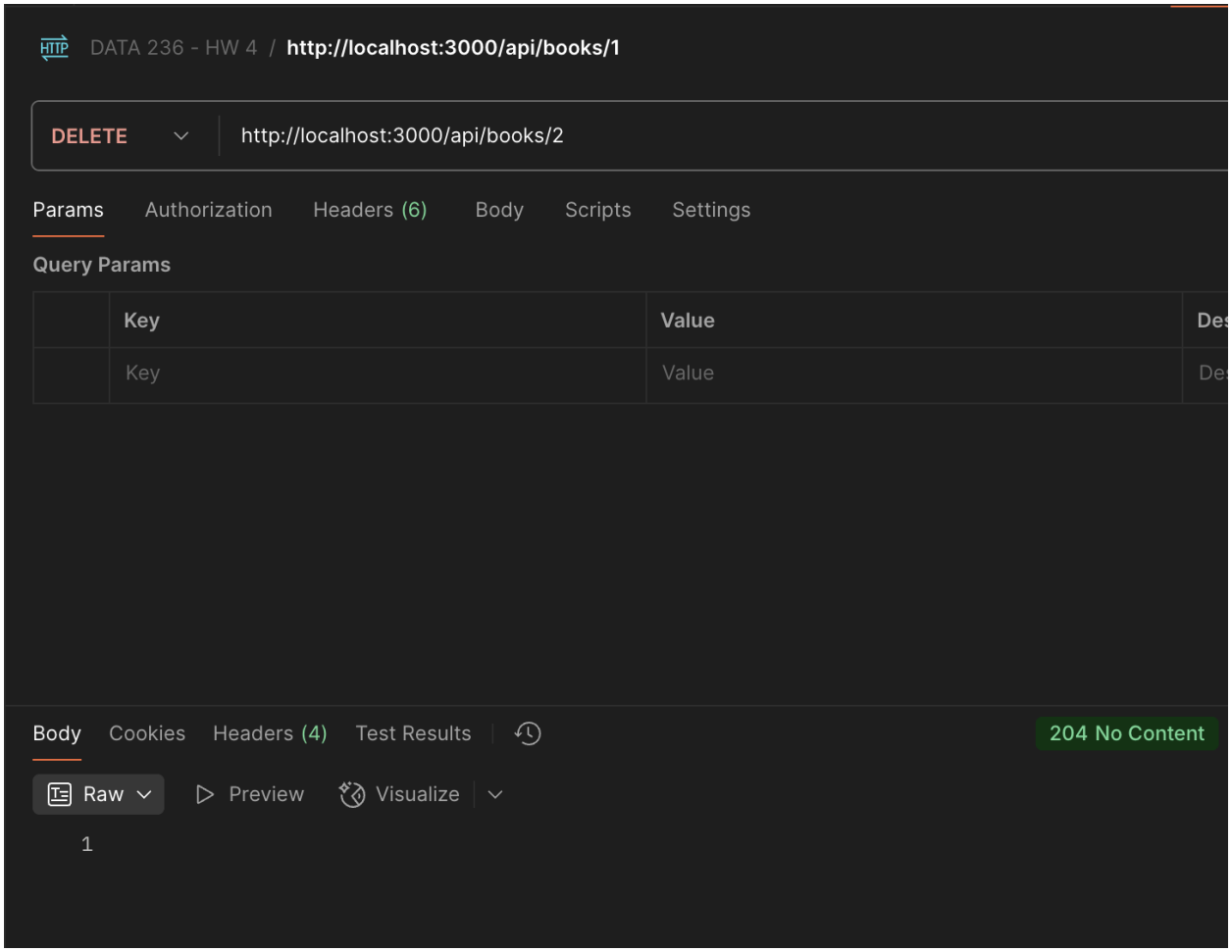


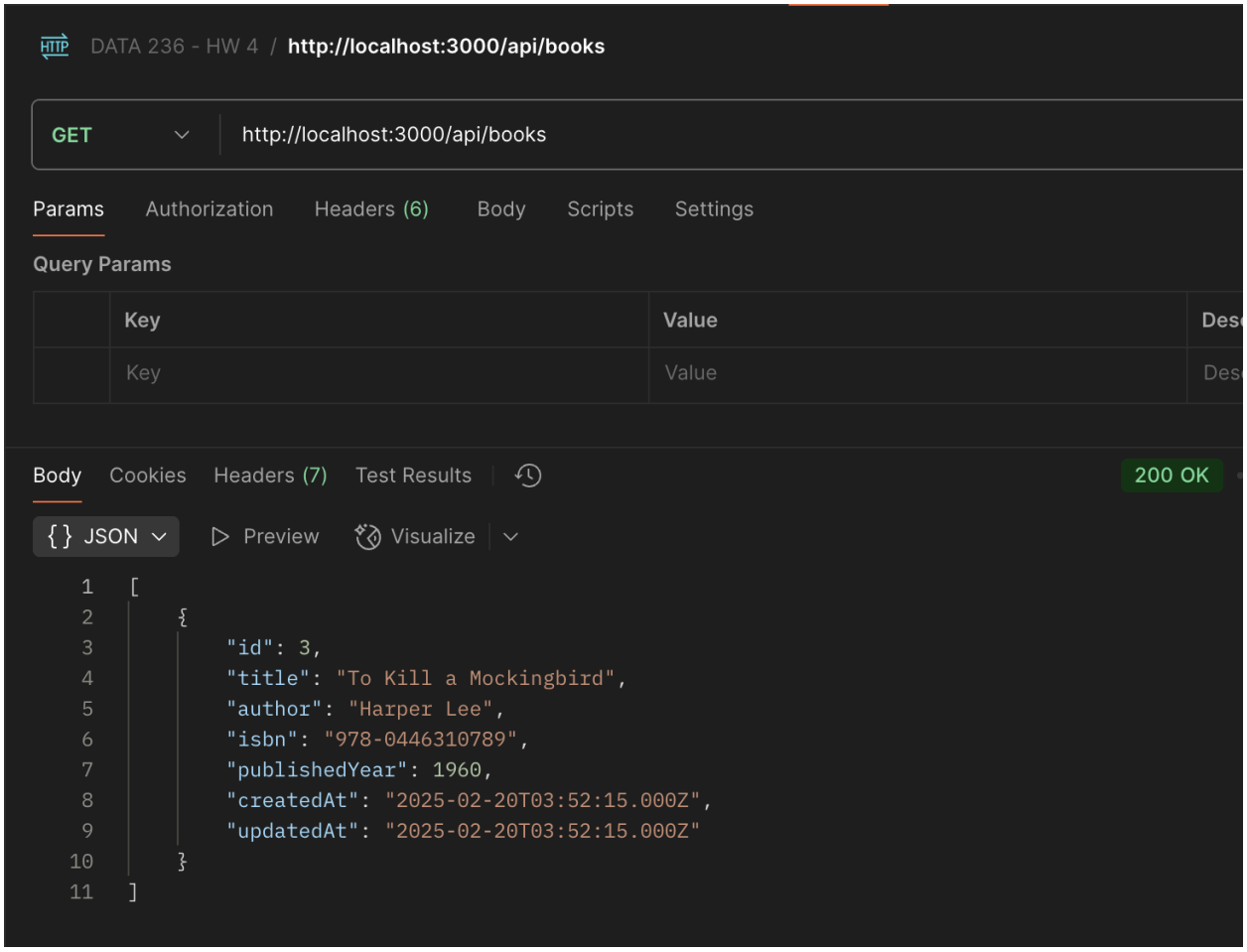
- Update book details (PUT)



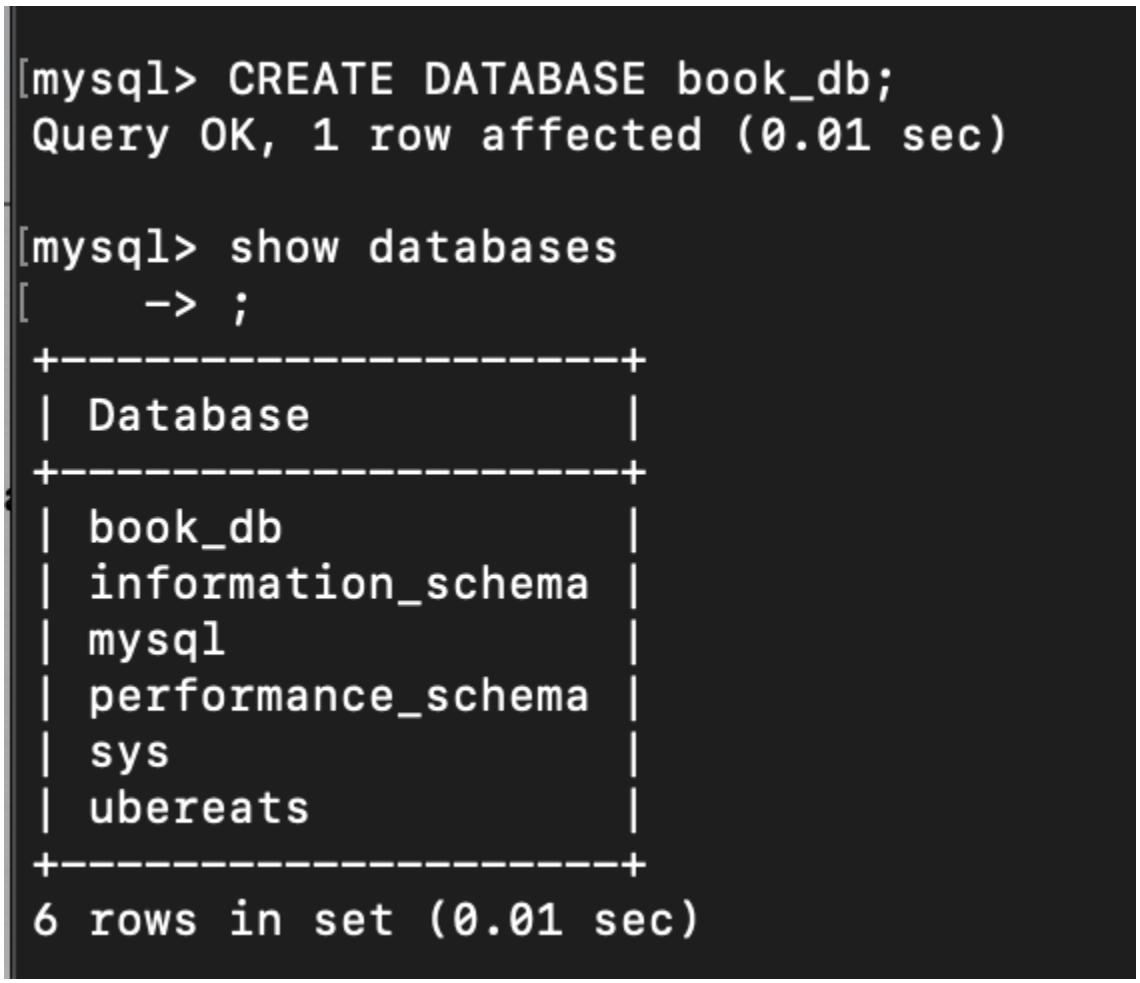

DATA 236 - HW 4 / http://localhost:3000/api/books/:id
GET
http://localhost:3000/api/books/2
Params
Authorization
Headers (6)
Body
Scripts
Settings
Query Params
Key
Value
Key
Value
Body
Cookies
Headers (7)
Test Results
200 OK
JSON
Preview
Visualize
{
"id": 2,
"title": "Updated Title",
"author": "Updated Author",
"isbn": "978-0743273565",
"publishedYear": 2024,
"createdAt": "2025-02-20T03:51:27.000Z",
"updatedAt": "2025-02-20T03:53:42.000Z"
}

- Delete a book (DELETE)




DATA 236 - HW 4 / http://localhost:3000/api/books
GET
http://localhost:3000/api/books
Params
Authorization
Headers (6)
Body
Scripts
Settings
Query Params
Key
Value
Key
Value
Body
Cookies
Headers (7)
Test Results
200 OK
JSON
Preview
Visualize
[
{
"id": 3,
"title": "To Kill a Mockingbird",
"author": "Harper Lee",
"isbn": "978-0446310789",
"publishedYear": 1960,
"createdAt": "2025-02-20T03:52:15.000Z",
"updatedAt": "2025-02-20T03:52:15.000Z"
}
]

Create a MySQL database named **book_db** with a table called **books** for this task.

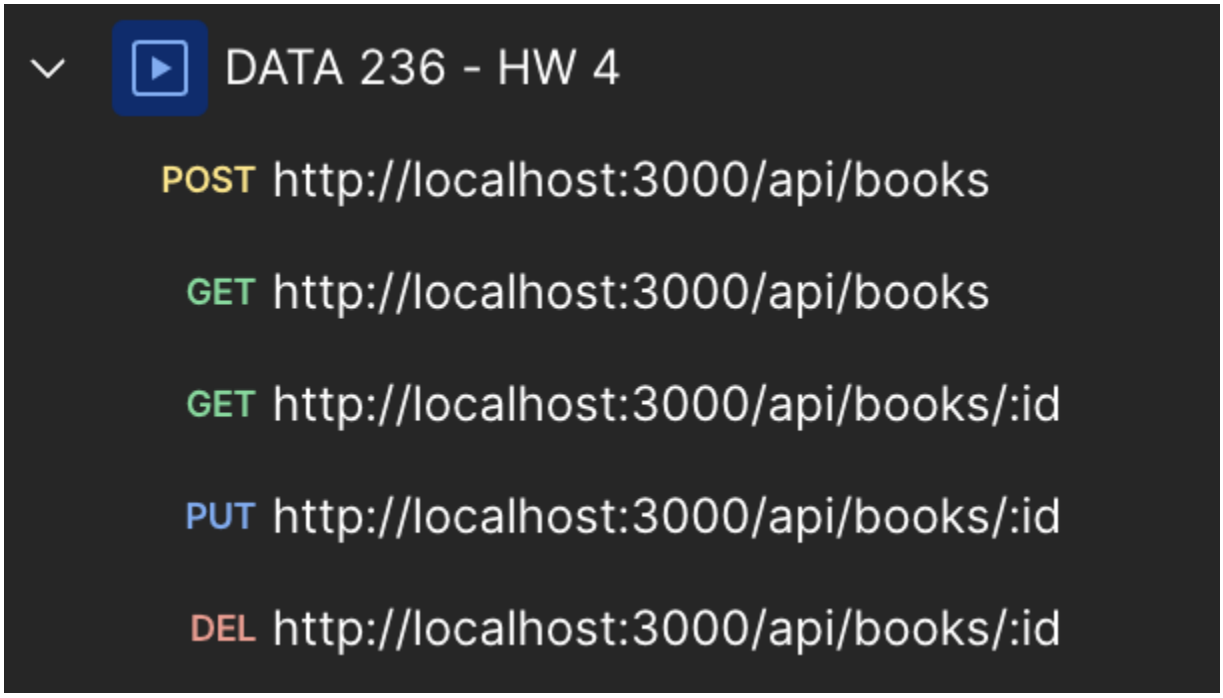


```
mysql> use book_db;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+------------------+
| Tables_in_book_db |
+------------------+
| Books            |
+------------------+
1 row in set (0.01 sec)

mysql> describe booksl
    -> ;
ERROR 1146 (42S02): Table 'book_db.booksl' doesn't exist
mysql> describe books;
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| id            | int          | NO   | PRI | NULL    | auto_increment |
| title         | varchar(255) | NO   |     | NULL    |                |
| author        | varchar(255) | NO   |     | NULL    |                |
| isbn          | varchar(255) | YES  | UNI | NULL    |                |
| publishedYear | int          | YES  |     | NULL    |                |
| createdAt     | datetime     | NO   |     | NULL    |                |
| updatedAt     | datetime     | NO   |     | NULL    |                |
+---------------+--------------+------+-----+---------+----------------+
7 rows in set (0.01 sec)
```

mysql-nodejs > JS book.model.js > ...

```
1   const { DataTypes } = require('sequelize');
2   const sequelize = require('./config/database');
3
4   const Book = sequelize.define('Book', {
5     title: {
6       type: DataTypes.STRING,
7       allowNull: false
8     },
9     author: {
10      type: DataTypes.STRING,
11      allowNull: false
12    },
13    isbn: {
14      type: DataTypes.STRING,
15      unique: true
16    },
17    publishedYear: {
18      type: DataTypes.INTEGER
19    }
20  });
21
22  module.exports = Book;
```

You are free to name your API endpoints appropriately.



Submit the Postman API response screenshots for each operation. Each completed operation will get 2 points. Also, submit a screenshot of the database and project folder structure.