COMP10020 - INTERNET TECHNOLOGIES

Bookclub Website

|  |  |
| --- | --- |
| Student Name: | Ziyang Chen |
| Banner ID: | B01656061 |
| Academic Year | 2023-2024 |
| Module | COMP10020 – INTERNET TECHNOLOGIES |
| Term | 1 |

目录

[Bookclub Website 1](#_Toc28931)

[Project Introduction 1](#_Toc10584)

[1. app.js 1](#_Toc2972)

[2. Homepage 2](#_Toc13879)

[3. Login 3](#_Toc10402)

[4. Bookshelf 7](#_Toc1885)

[5. Bookdetails 8](#_Toc25379)

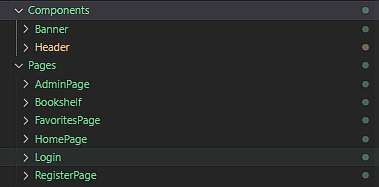
[6. Database 15](#_Toc23070)

[8. Hosting 17](#_Toc16314)

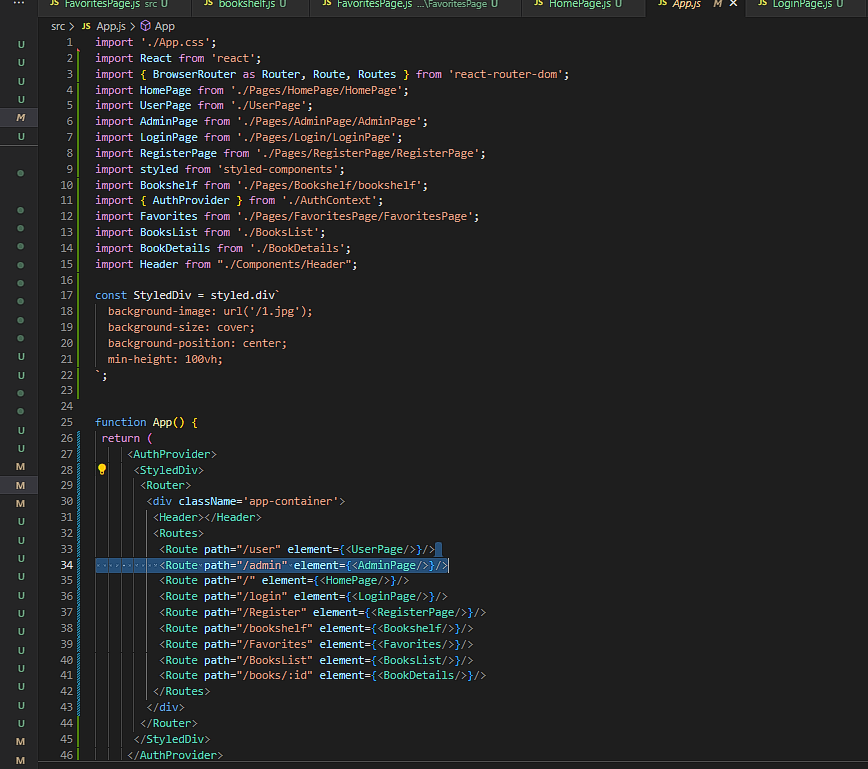
Project Introduction

This project is open to all users who are interested in reading. We welcome all reading enthusiasts to upload their favorite books on this website and express their opinions and recommendations. And users are very welcome to share books between users.

Therefore, this website includes the following pages:



1. app.js



This code is the main component of a web application written using the React framework. It defines the routing structure and appearance style of the application.

“styled-components”: The styled-components library is used to create components and attach styles. This defines a StyledDiv component that will serve as the root container of the application and sets the background image, overlay background size, background position, and ensures it is at least as tall as the viewport height.

“react-router-dom”: Use the react-router-dom library to manage page navigation and routing. It allows applications to switch views without reloading the page.

“BrowserRouter”: A router that uses the HTML5 history API (pushState, replaceState, and popstate events) to keep the UI and URL in sync.

“Routes and Route”: A series of Route components are defined in the Routes component, and each Route corresponds to a path and rendered component.

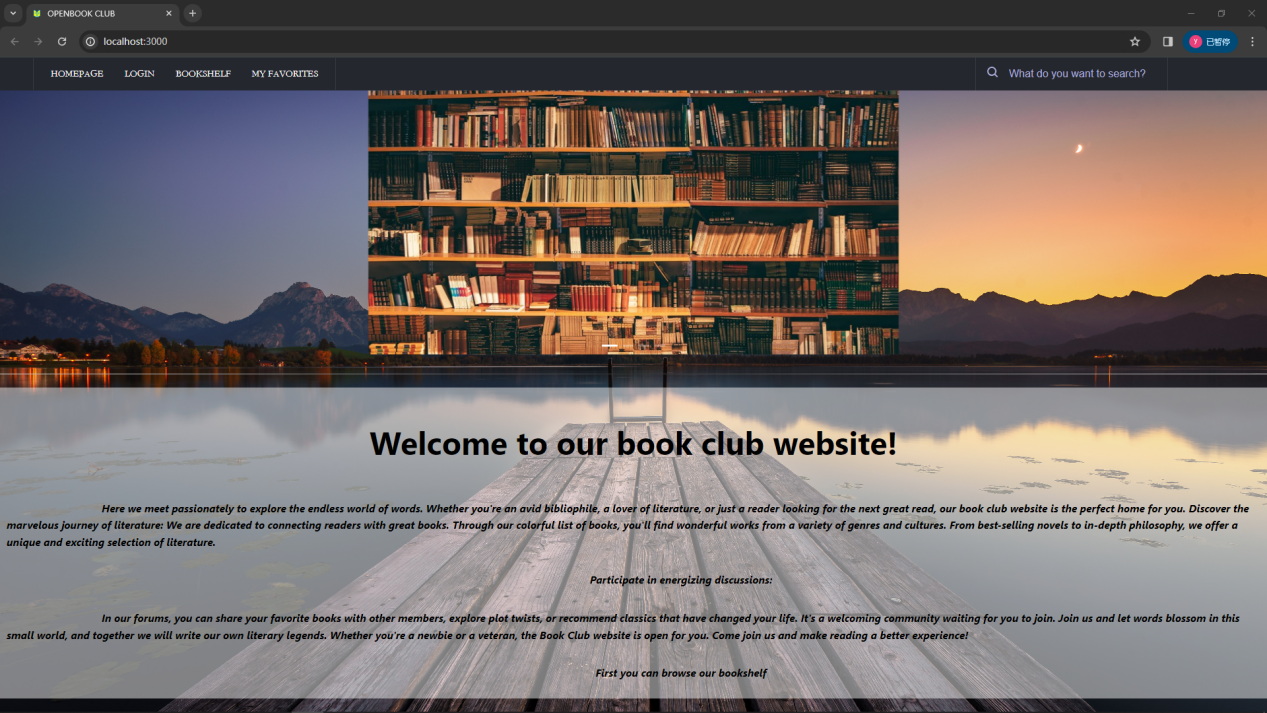
“AuthProvider”: Wrap the entire application in the AuthProvider component, which is a context provider that manages the authentication status of the entire application. This is also convenient for viewing your favorites after logging in, and determining whether the commenter is registered. People who are not registered cannot comment.

“StyledDiv”: This is the root container of the application and all page components will be rendered within this container.

“Header”: May be a component containing a navigation menu or application title.

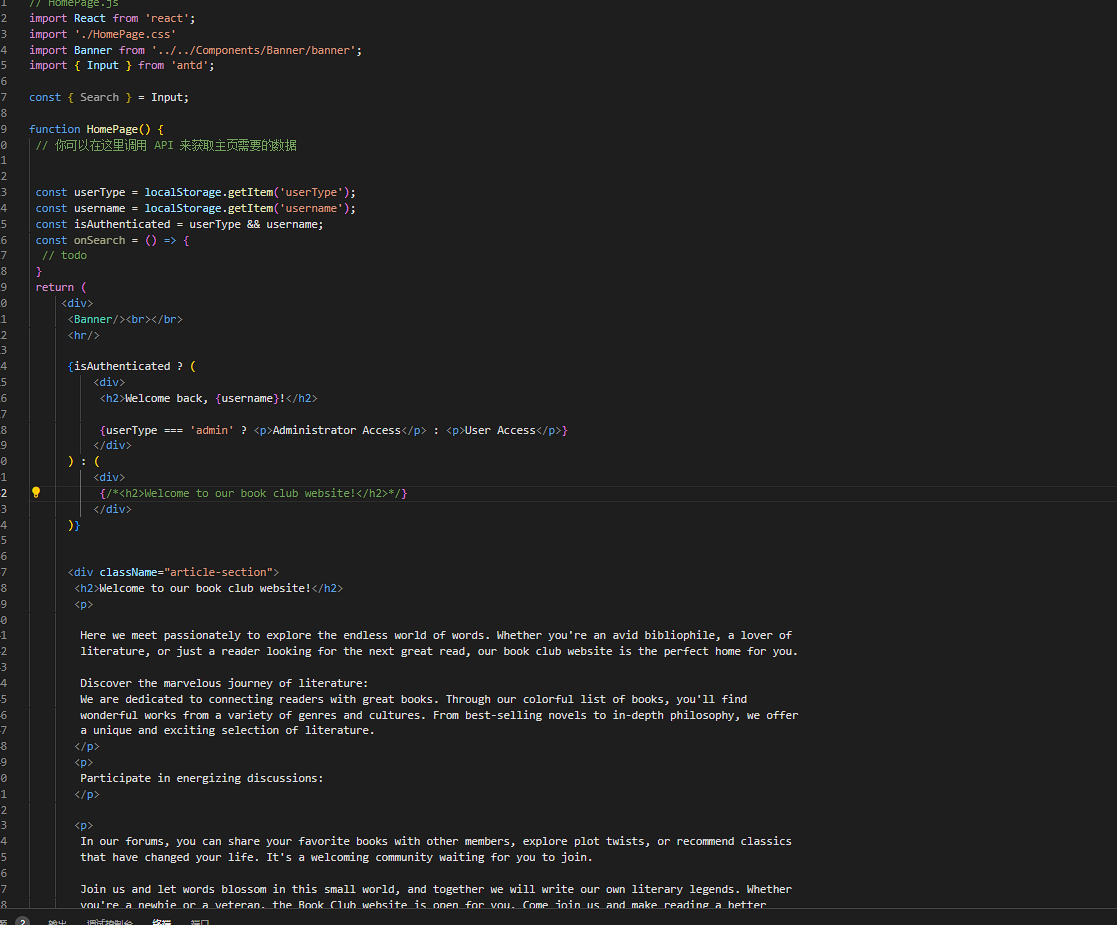
“div” with class app-container: Defines a div with class app-container to wrap routers and routes, which helps keep the layout consistent throughout the application.

1. Homepage

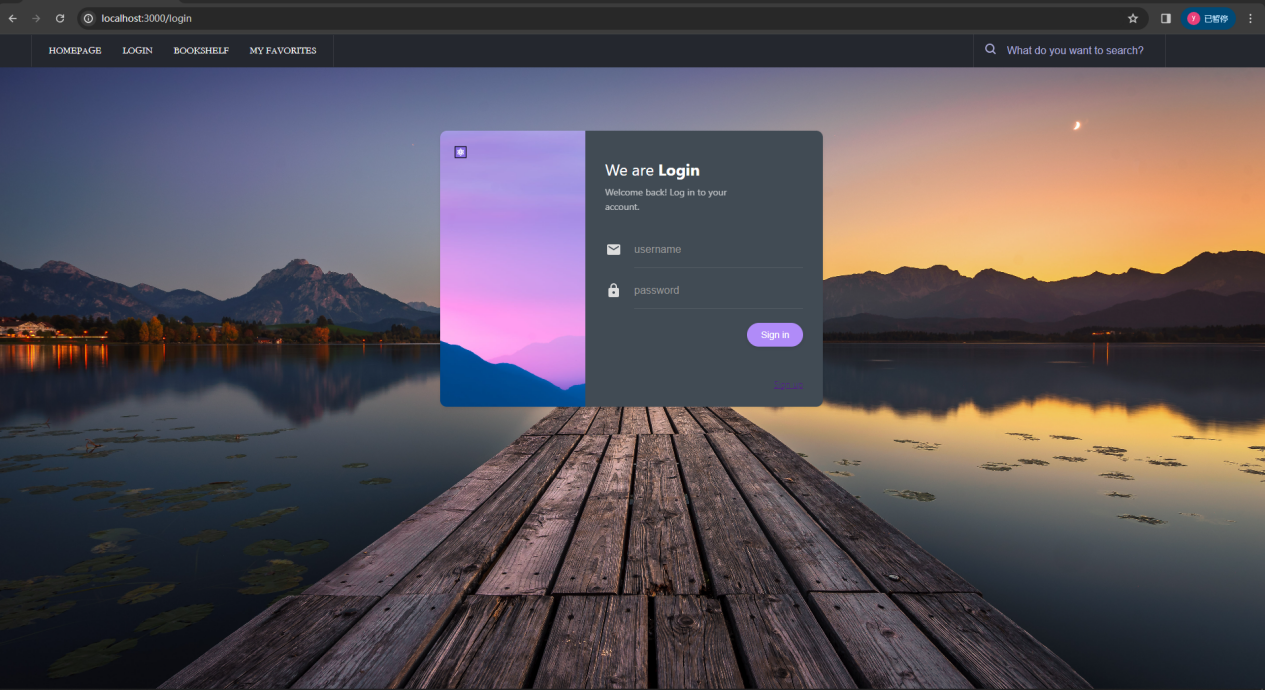


I divided the main interface into a directory page (Homepage, Login, Bookshelf, Myfavourite) at the top, a search bar, a looping component in the center that adds interest to the page, and text to welcome and introduce users at the bottom.

The HomePage component represents a welcome page that introduces the book club and its community. It uses conditional rendering to display different user information, and enhances the interactivity and beauty of the page through UI elements provided by antd. Although there is a skeleton of a search function in the code (onSearch function), the actual logic has not been implemented yet. The purpose of this component is to give users a warm and personalized welcome while introducing the main features of the site and community activities.

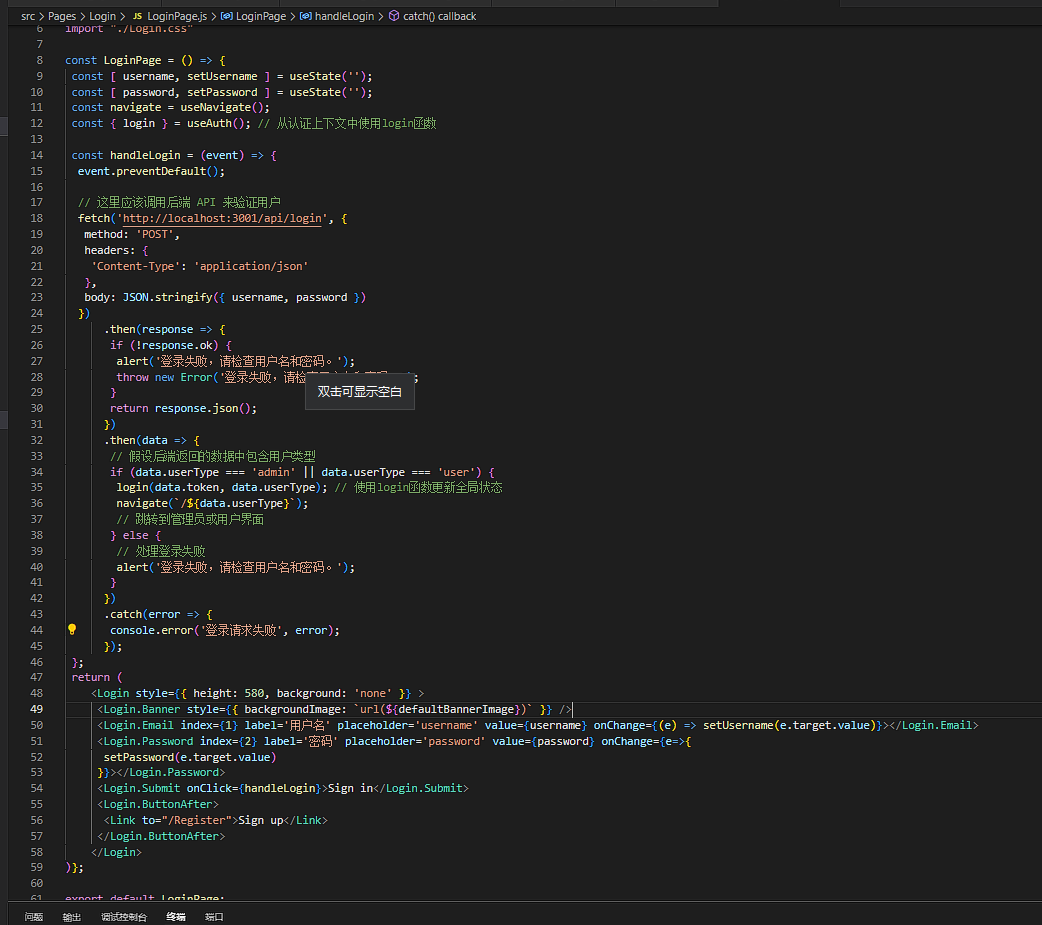


1. Login



In this interface, we provide a login interface and a registration interface in the lower right corner of the main interface.

The login interface requires email and password to log in.



Use “useState” to create username and password states, which are used to store and update the login information entered by the user.

“useNavigate” is a new hook for react-router-dom v6 for page navigation in components.

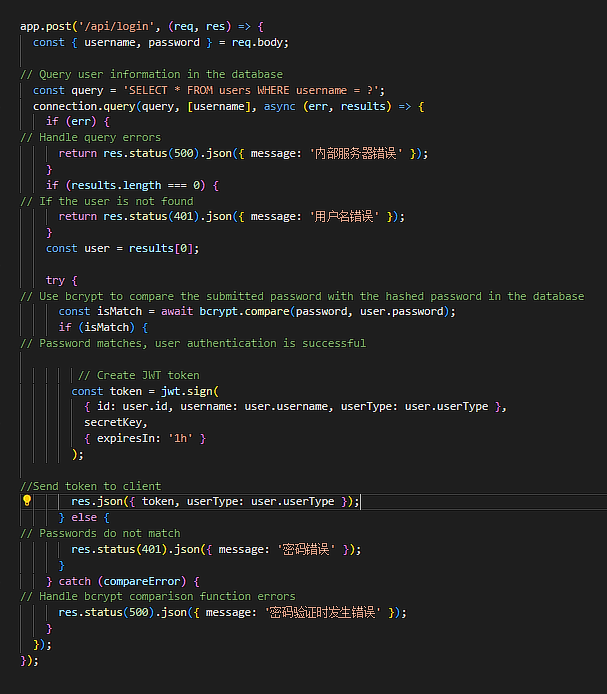
Use a component called Login to render the login screen, which may be imported from the @react-login-page/page3 package.

The Login.Banner subcomponent is used to display the background image of the login page. The background image is introduced through defaultBannerImage.

The Login.Email and Login.Password subcomponents are used to collect the user's username and password. Their values are bound to the component state and updated as the user enters.

The Login.Submit component is used to submit the login form.

Login.ButtonAfter contains a Link component that leads to the registration page and allows users to jump to the registration form.



The backend of the login interface

Use app.post to create a route that handles POST requests with the path '/api/login'.

Extract the username and password from the request body.

Use connection.query to query the database to check if a user with the given username exists.

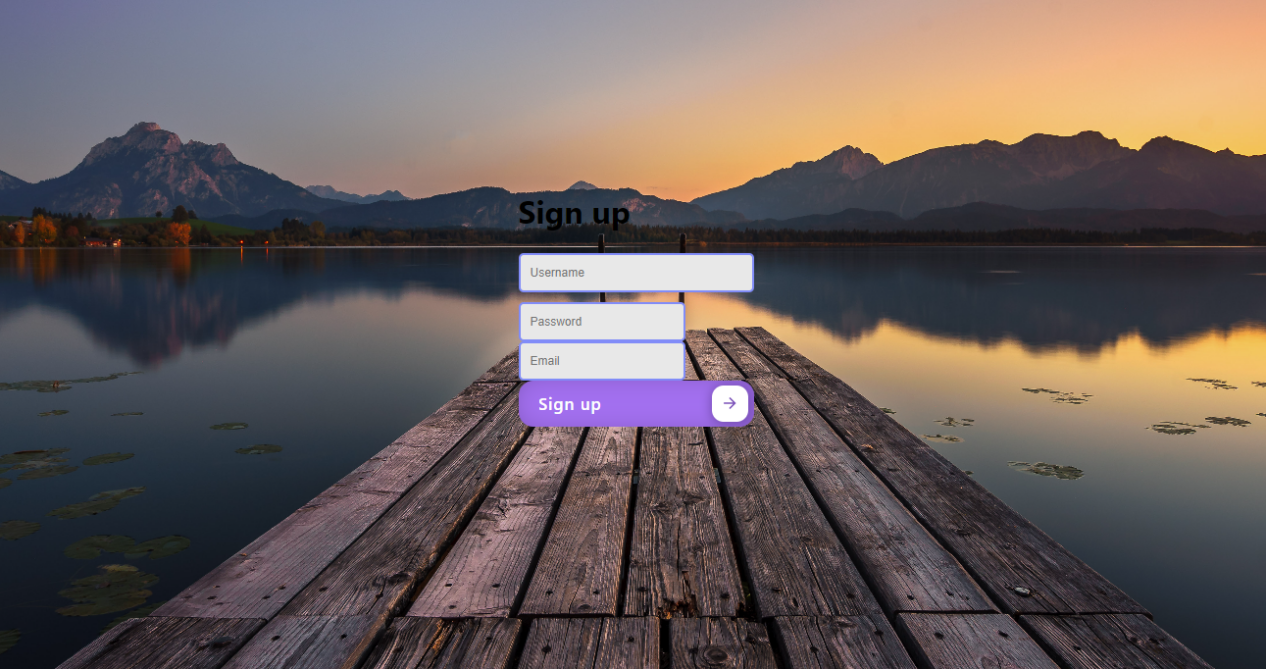
Use parameterized queries ([username]) to prevent SQL injection attacks.

Use bcrypt.compare to compare the submitted password with the hashed password in the database.

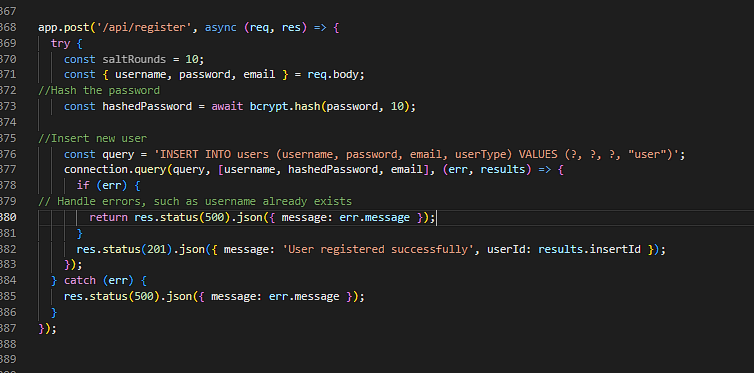
If the passwords match, proceed to the next step of JWT generation; if they do not match, return a 401 status code and prompt that the password is incorrect.

Generate a JWT using the user's ID, username, and userType.

Set the JWT expiration time to 1 hour.



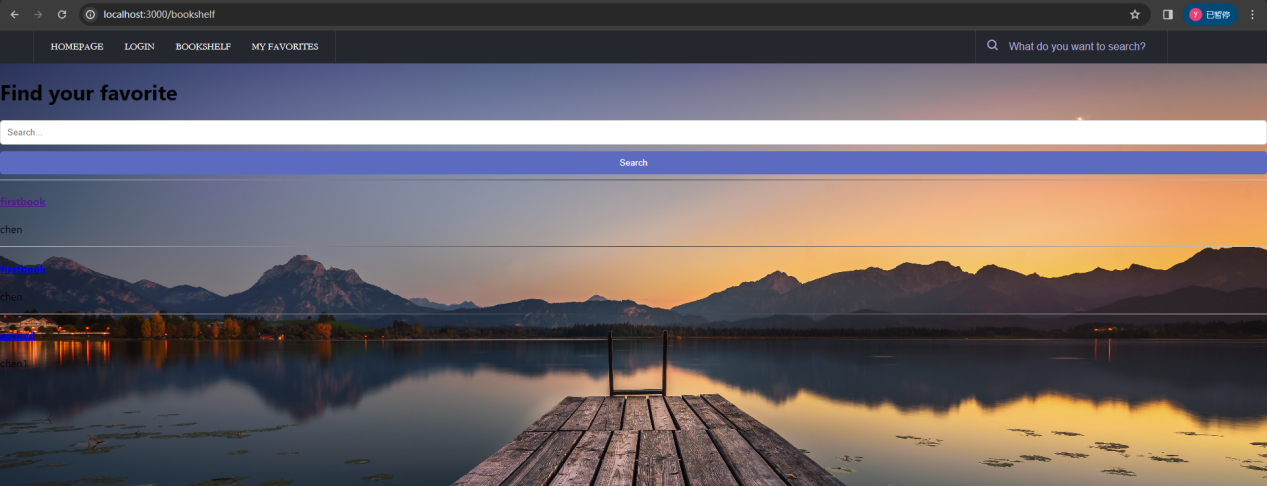
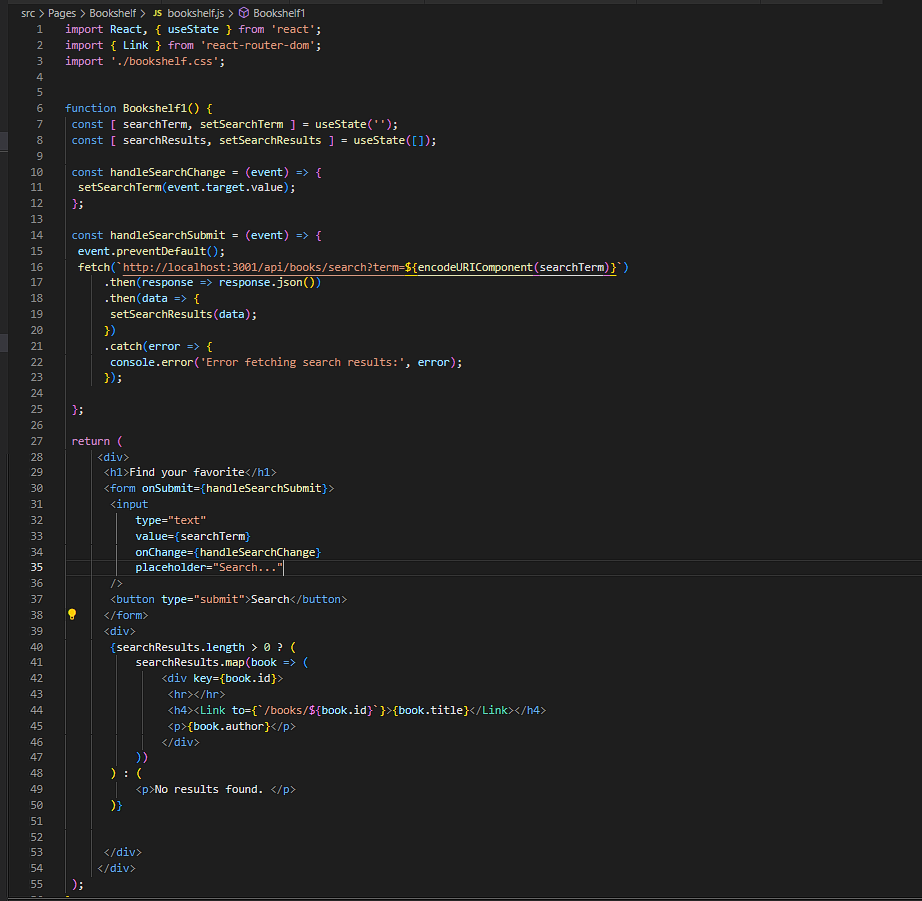
Registration requires username, password, and email.



Use the hash function of the bcrypt library to encrypt the user password, using 10 rounds of salt (security factor).

Construct a SQL query to insert the new user's username, hashed password, email, and user type into the users table. The user type is hard-coded as "user", which means that all users who register through this endpoint will be granted normal user rights.

Use connection.query to perform this insert, passing the username, hashed password, and email as parameters to the query.

1. Bookshelf 

Use useState to create two state variables: searchTerm and searchResults.

searchTerm is used to store search terms entered by the user.

searchResults is used to store search results obtained from the API.

The handleSearchChange function is bound to the onChange event of the input box and is used to update the searchTerm state to reflect the user's input.

The handleSearchSubmit function is bound to the form's onSubmit event. When the user submits the form, the default form submission behavior is prevented and the search operation is performed.

Use the fetch function to send a GET request to the backend API. The requested URL contains the searchTerm entered by the user, and is URL encoded through the encodeURIComponent function.

Parse JSON data from the response and set it into searchResults state.

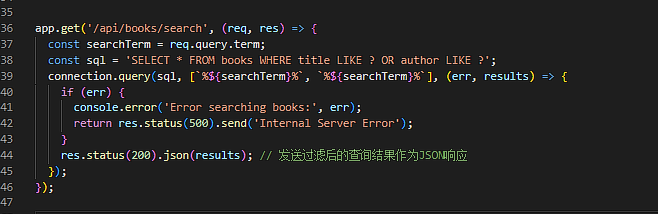
The display content is determined based on the length of searchResults.

If there are search results (searchResults.length > 0), iterate through each result and render a div element with the title and author for each book.

Use the Link component to create a link to the detail page of each book with the path /books/${book.id}.

If there are no search results, a "No results" message is displayed.

Bookshelf1 component provides users with a simple book search function. The user can enter a search term, and after submitting the form, the component requests the search results from the backend API and displays them. This component uses the basic features of React (such as state management and event handling) and the Link component of react-router-dom to implement navigation. Overall, this is a basic example showing how to handle API requests and routing in React.



The search backend

const searchTerm = req.query.term: Extracts the parameter named term from the request's query string. This parameter represents the search term entered by the user.

const sql = 'SELECT \* FROM books WHERE title LIKE ? OR author LIKE ?':

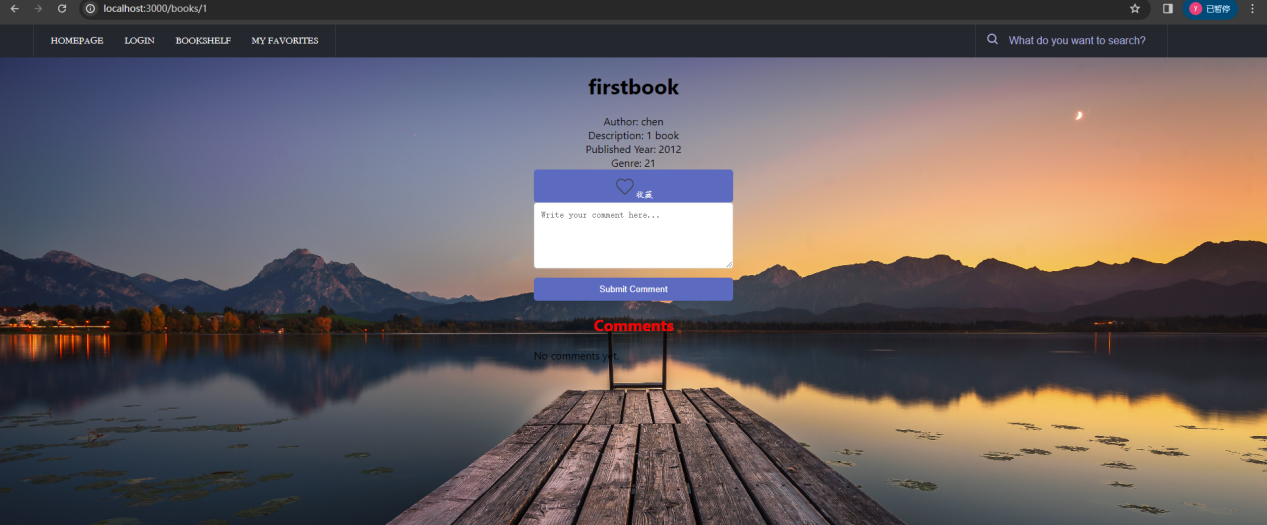
Construct a SQL query statement to search for book titles (titles) or authors (authors) containing search terms in the books table.

Use the LIKE operator and the wildcard character % to achieve partial matching, so that the search term can match any part of the field.

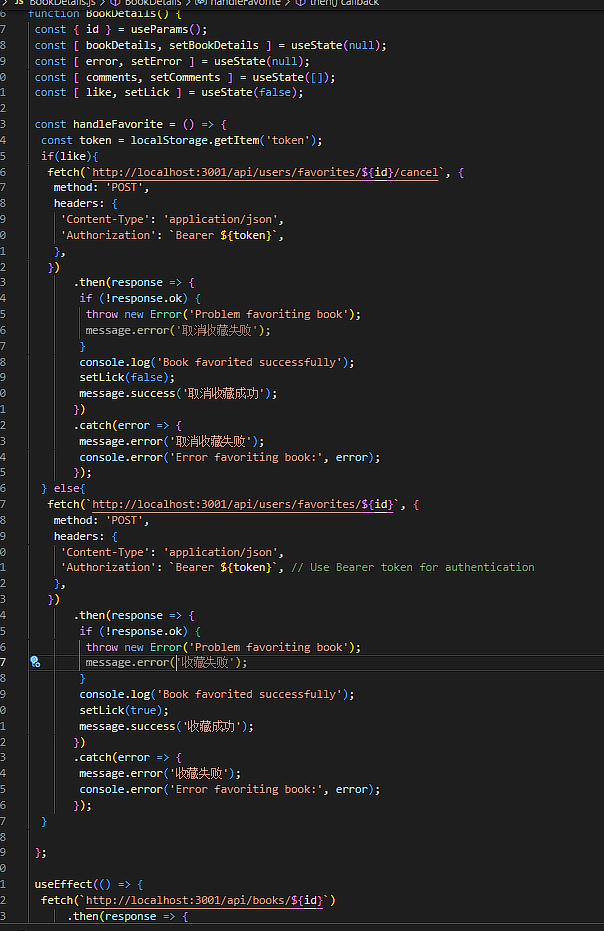
This query is case-insensitive, which means that the search terms and the data in the database may not match the case.

1. Bookdetails

This is the bookshelf interface, which is used to list all books and search based on keywords.This is the bookshelf interface, which is used to list all books and search based on keywords. Displayed book list, showing book name, author, uploader, release time, description



After clicking the title of the list, you will jump to the book details page, which displays the book information in detail. You can add it to the My favorite page through the Add to Favorites button. There is a comment area below where you can add your own comments or browse other books. comments left by users.



Use useParams to get the book ID in the URL.

Use useState to manage book details (bookDetails), error messages (error), comments (comments), and collection status (like).

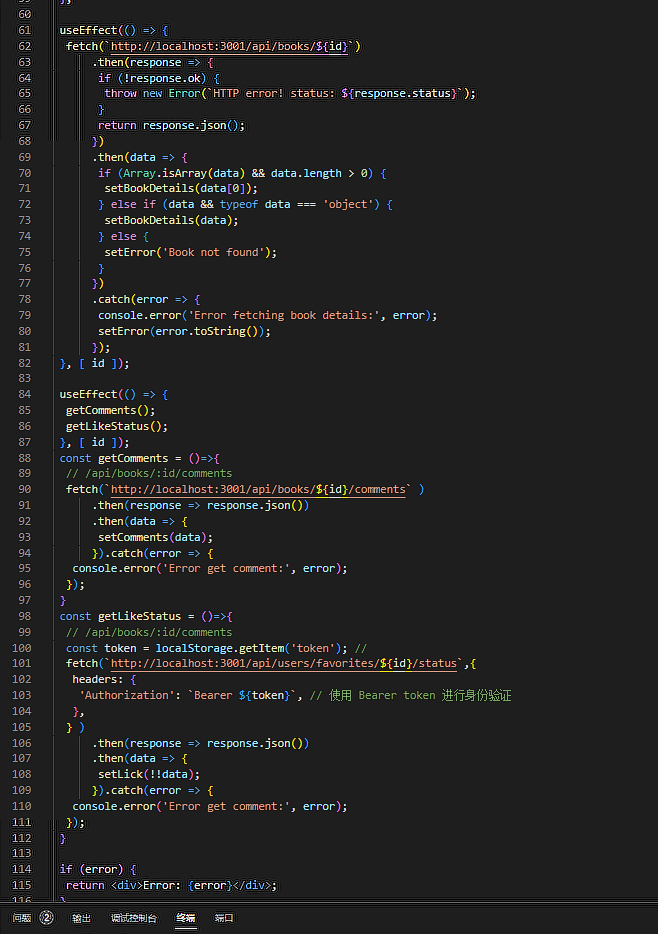
Use the useEffect hook to get the book details and reviews from the API based on the book ID when the component loads.

Another useEffect hook is used to obtain the current user's collection status of the book.

The handleFavorite function is used to handle the logic of favorites and unfavorites.

Determine whether to initiate a collection or cancel a collection request by judging the like status.

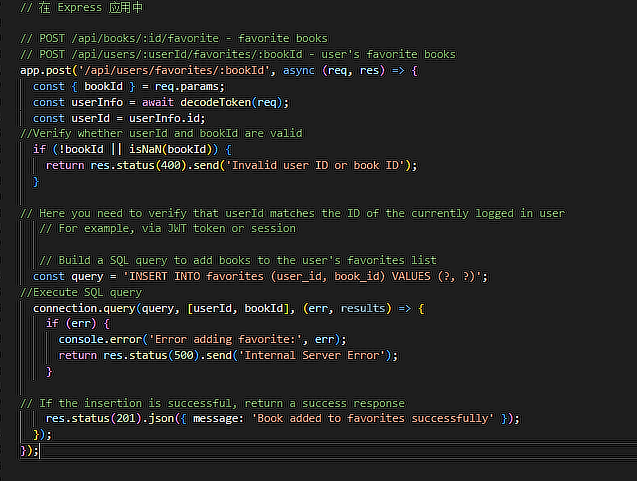
Use the fetch function to interact with the backend API and update the status based on the response.



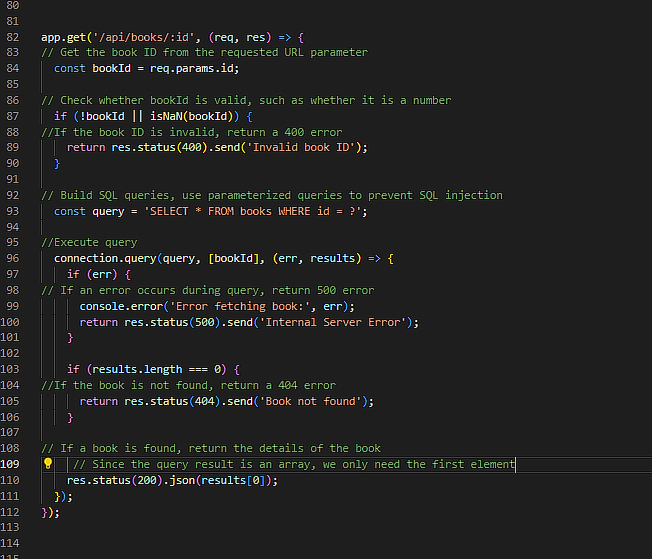
5 interfaces are used here to process data from the backend.



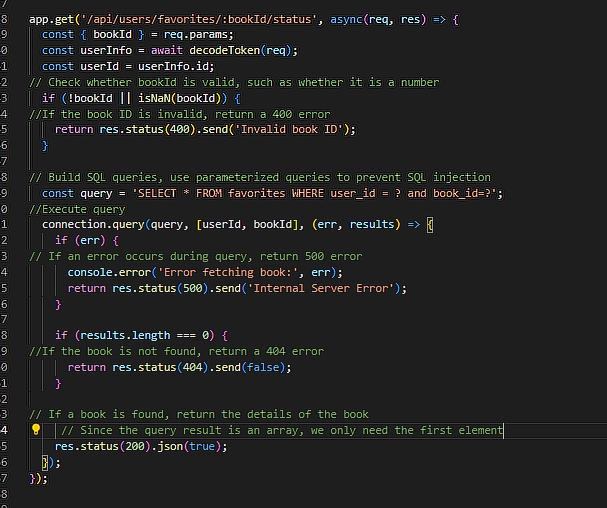
Used to cancel favorites



Used to query the user's collection of books

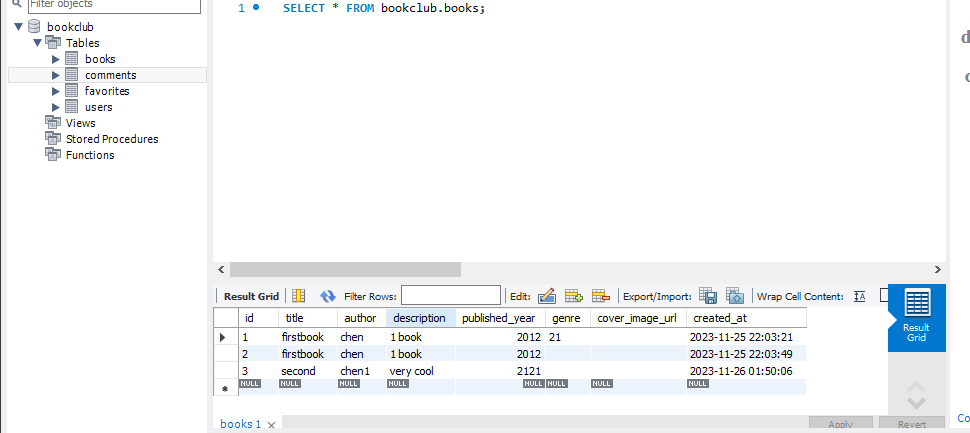


Used to obtain information corresponding to the ID book

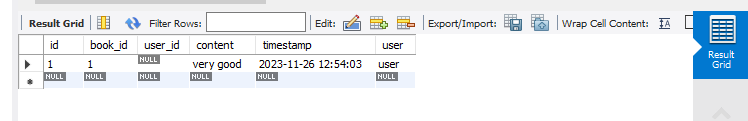


Obtain the corresponding secretary data according to the book ID

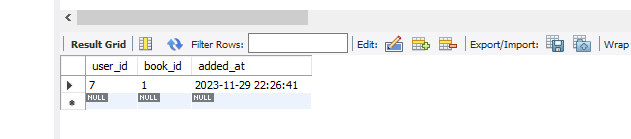
1. Database



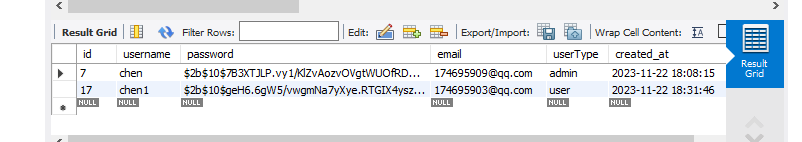
Books



Comments



Favorites



Users

Four tables were created to store book information, user information, a collection form that connects books and users, and a user comment form. Passwords are encrypted for security.

1. Future Development

* Consider introducing artificial intelligence and machine learning technologies to provide more personalized book recommendations and content filtering.
* Develop mobile applications to make it easier for users to access and use website features on mobile devices.
* Online and offline reading clubs and author meetings are held regularly to enhance community cohesion.
* The book database is continuously expanded to cover more types and languages of books to meet the needs of different users.
* Consider integration with existing services:
* Integrate social media APIs to allow users to share their reading experiences through platforms such as Facebook, Twitter, etc.
* Explore cooperation possibilities with online bookstores, such as providing direct book purchase links or e-book downloads.
* Consider partnering with public libraries or academic databases to provide users with richer resources.
* In fact, I thought about directly using the APIs of some large libraries for operations, which is the most comprehensive and safest way. However, due to time issues and technical problems, I finally gave up. If we continue to develop, this should be the first thing to do, because only Enriching book content can provide users with a better experience.
* Google Books API:

The Google Books API allows you to search the full text of a book and obtain book details such as title, author, publisher, publication date, etc. It also provides user reviews, ratings, and preview links.

* Open Library API:

Open Library is a free resource operated by the Internet Archive that provides data on a large number of books. It allows users to retrieve book information through ISBN, LCCN, OLID, etc.

* In order to ensure the safety of users we must do the following:
* Use HTTPS:

Make sure your website uses HTTPS, which means all data transmitted to and from your website is encrypted.

Obtain and install an SSL/TLS certificate to ensure data is encrypted during transmission.

* Database encryption:

Encrypt sensitive data (such as user passwords, personal information, etc.) stored in the database. The password can already be encrypted using a hashing algorithm such as bcrypt. For other sensitive data, symmetric or asymmetric encryption technology can be used for encrypted storage.

* Secure cloud storage service:

Choose a cloud storage service (such as AWS, Azure, Google Cloud) that offers advanced security and compliance. Since I already have an AWS cloud server, I should deploy the project to AWS.

Ensure that cloud services comply with regional or industry data protection regulations (such as GDPR, HIPAA, etc.).

1. Hosting

1.Deploy Node.js and MySQL database

Connect to your EC2 instance using SSH.

Install the Node.js environment.

Install the MySQL database and configure it appropriately (such as setting the root password).

Migrate your local MySQL database to a MySQL database on AWS EC2. This may involve exporting local database data and importing it onto an EC2 instance.

2. Deploy Node.js backend

Upload your Node.js application code to the EC2 instance. You can use Git, SCP, or other file transfer methods.

Install all dependencies required for your Node.js application on the EC2 instance.

If necessary, modify the configuration of your Node.js application to connect to the MySQL instance on EC2.

Make your Node.js app run as a service using PM2, Forever, or similar tools.

3. Deploy React front-end

Build your React app (usually npm run build) to generate a production version of the code.

Upload the static files generated by the build to the EC2 instance.

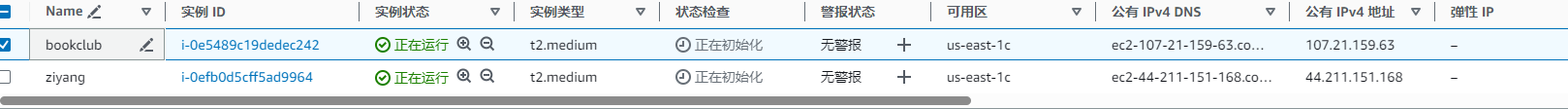
You can choose to use Nginx or Apache as your web server to serve these static files.

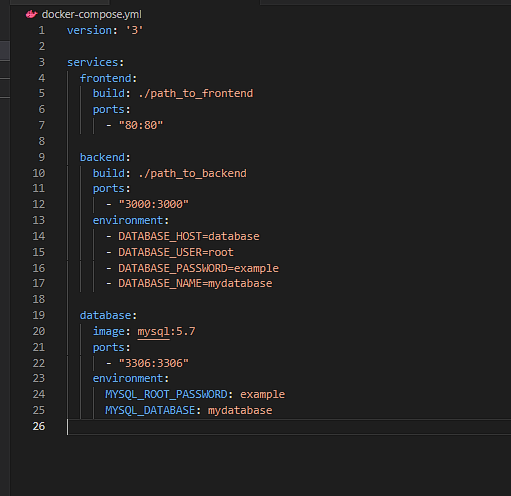
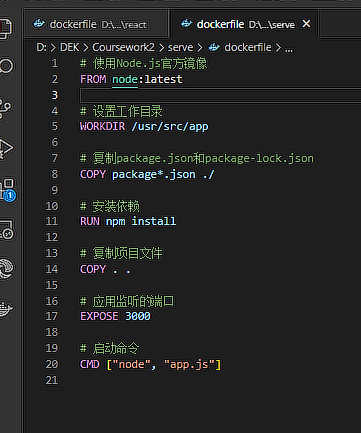
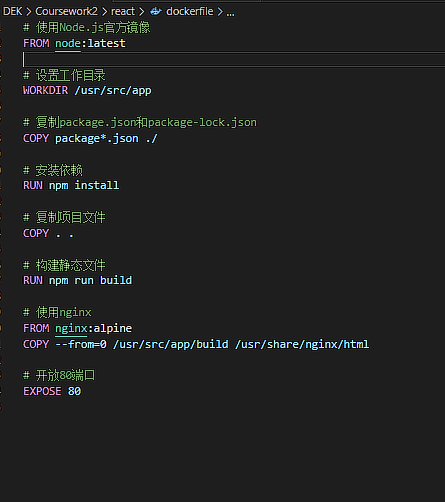
4. Set domain name

If you have a domain name, you can point it to your EC2 instance via Route 53 or another DNS service.Configure Nginx or Apache to handle requests for your domain name.

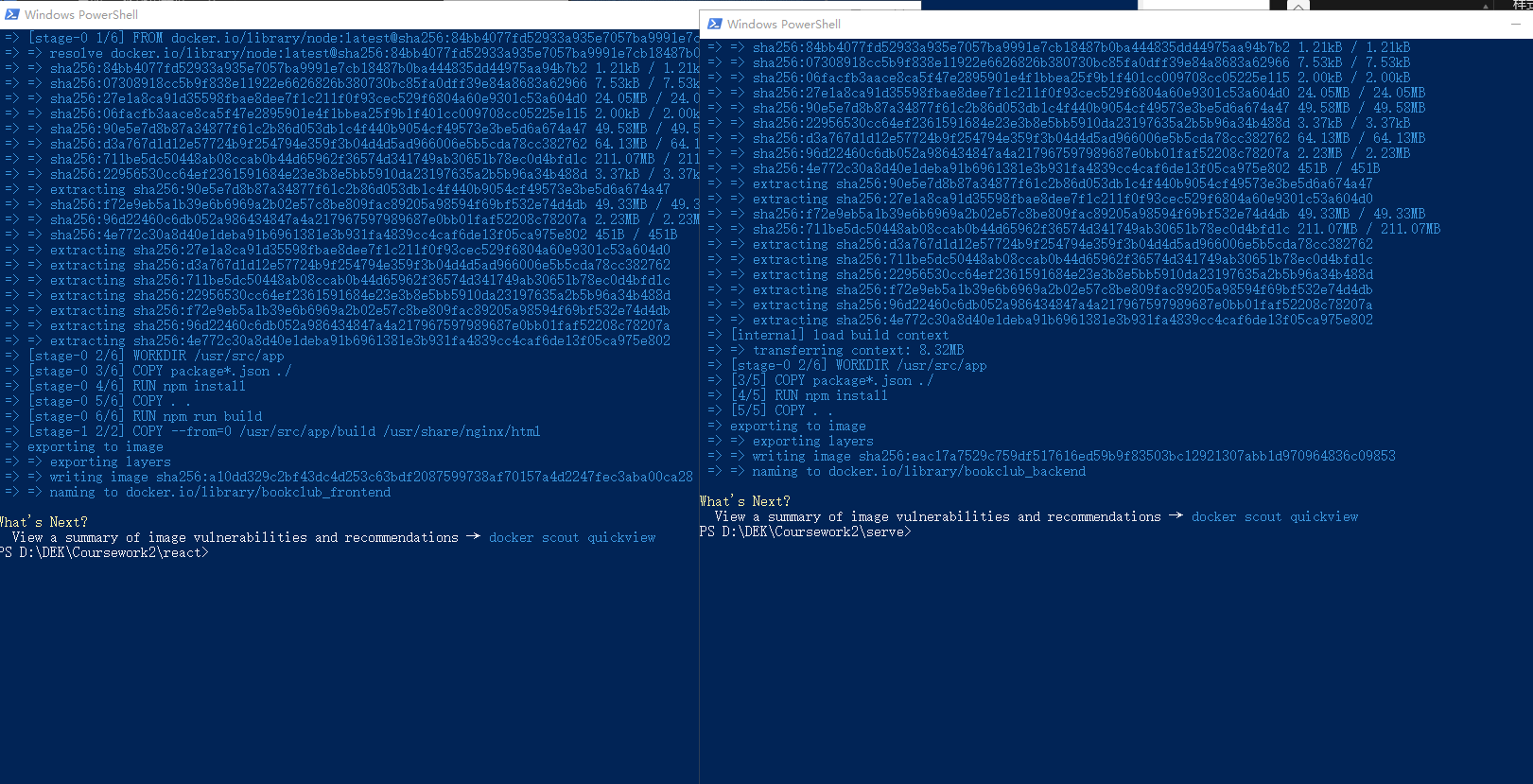
5. Set up HTTPS

For security, it is recommended to set up SSL/TLS encryption. You can get a certificate for free with Let's Encrypt.Configure Nginx or Apache to enable HTTPS.





Use dockers to connect the front-end and back-end so that users can start together and download all dependencies when they start.



Combine front-end and back-end projects using Docker and publish them to GitHub.

1. Environmental consistency

Consistency between development and production environments: Docker ensures consistency between development, testing and production environments, reducing the "can run on my machine" problem.

Dependency management: Docker containers contain all dependencies required to run the project, thereby reducing problems caused by environmental differences.

2. Simplify configuration

Containerization simplicity: Using Docker containers, entire front-end and back-end applications, including their environments and dependencies, can be easily packaged.

Easy to understand and maintain: Docker configuration is usually written into Dockerfile and docker-compose.yml files, which makes the project's configuration clearer and easier to maintain.

3. Improve development efficiency

Quick start and replication: Docker containers can be started and stopped quickly, making the development and testing process more efficient.

Team collaboration: Sharing Docker configurations via GitHub allows team members to start working quickly without spending a lot of time configuring the environment.

4. Easy to deploy and expand

Consistent deployment process: Docker containers can run in any environment that supports Docker, simplifying the deployment process.

Easy to scale and scale: Docker and container orchestration tools such as Kubernetes make it easy to scale and load balance in cloud environments.

5. Version control and documentation

Version control: Put the Docker configuration file on GitHub to enjoy the benefits of version control, such as tracking changes, branch management, etc.

Documentation: Docker files also serve as documentation of environment configuration, helping new members understand project settings.

6. Security and isolation

Runtime isolation: Docker provides an isolated environment for each application, enhancing security.

Dependency isolation: The front end and back end can have independent dependencies, reducing the risk of version conflicts.

By combining Docker and GitHub, you can achieve a more consistent, efficient, and secure development and deployment process. This method is particularly suitable for the development of team projects and complex applications, and can significantly improve team collaboration efficiency and application maintainability.