WINTER 2020 COMP3005 PROJECT
Online Bookstore Application

Zijian Zhen
101087006
zijianzhen@cmail.carleton.ca

Jiujiu Duan
101094923
jojoduan@cmail.carleton.ca

1 Conceptual Design

The ER-Diagram of the database is shown in Figure 1.

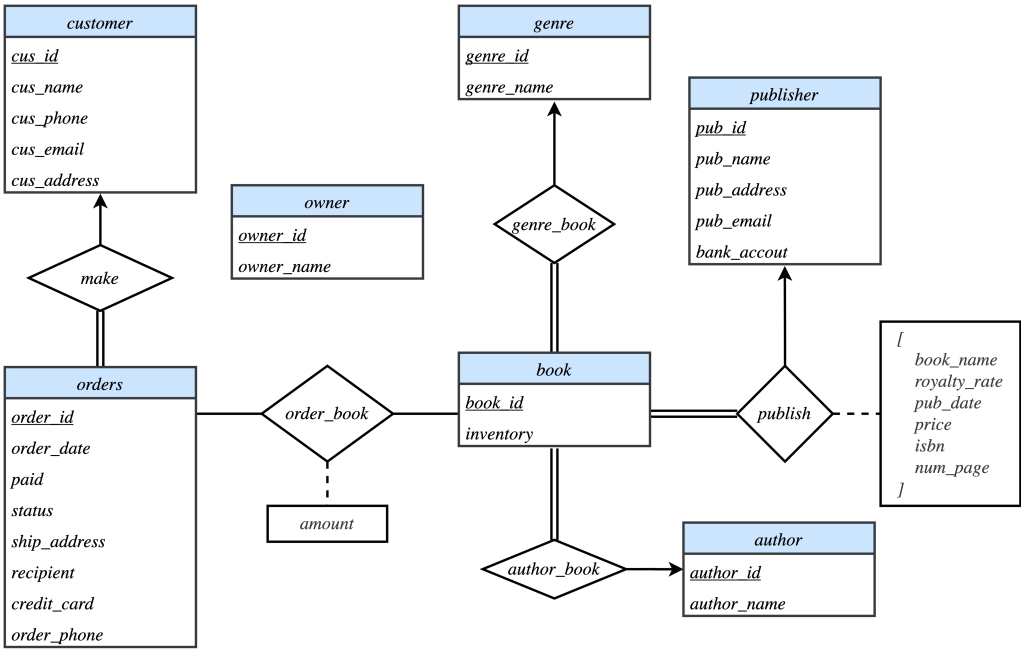

Figure 1: ER-Diagram

Assumption and explanation:

1. The Book Store app applies the Google account API for login and registration, so, password is not needed in our design(it's associated with your Google account). In this way, no duplicate email address is allowed in different account.

2. There should be a relationship set between *owners* and the entity set *bookstore*. But considering that the database is only applied in one bookstore, so, the entity set *bookstore* and the relationship set between *owner* and *bookstore* are omitted from our design. However, *owners* is not removable, since the management system (for bookstore owners/managers) need this entity to check administrative authority.

3. We assume that the royalty rate, publish data, price, ISBN, the total number of pages and price of a book are determined when the publisher publish the book. So, we set them as attributes of the relationship set *publish*

4. Attribute *inventory* in *book* corresponds to the remaining quantity of book in the book store.

5. Attribute *royalty_rate* in *publish* corresponds to the percentage of the sales to be transferred to the publishers.

6. Attribute *amount* in *order_book* corresponds to the total number of a specific book in the given order.

2 Reduction to Relation Schemas

After reducing the ER-diagram of the database, the relation schemas is listed below:

- *customer(cus_id, cus_name, cus_phone, cus_email, cus_address)*
- *genre(genre_id, genre_name)*
- *author(author_id, author_name)*
- *owners(owner_id, owner_name)*
- *publisher(pub_id, pub_name, pub_phone, pub_email, pub_address, bank_account)*
- *orders(order_id, cus_id, order_date, paid, status, ship_address, recipient, credit_card, order_phone)*
- *book(book_id, inventory, pub_id, author_id, genre_id, book_name, royalty_rate, pub_date, price, isbn, num_page, on_shelf)*
- *order_book(order_id, book_id, amount)*

3 Normalization of Relation Schemas

Subsection **3.1** to Subsection **3.8** shows the tests of BCNF normal form for each of the tables. According to the tests, all tables in the database are in a BCNF normal from.

3.1 customer

customer(cus_id, cus_name, cus_phone, cus_email, cus_address)

Let the relation schema *customer* be $R(A, B, C, D, E)$,
where R =*customer*, A =*cus_id*, B =*cus_name*, C =*cus_phone*, D =*cus_email*, E =*customer_address*

According to assumption 1, we have, $A \to BCDE$, $D \to ABCE$
Since A, D is unique for each customer (as what we assumed), we have, A, D are candidate keys.
We conclude that, $R(A, B, C, D, E)$ is in BCNF.

3.2 genre

genre(genre_id, genre_name)

Let the relation schema *genre* be $R(A, B)$,
where R =*genre*, A =*genre_id*, B =*genre_name*

We have, $A \to B$, $B \to A$.
Obviously, both A and B are candidate key.
We conclude that, $R(A, B)$ is in BCNF.

3.3 author

author(author_id, author_name)

Let the relation schema *author* be $R(A, B)$,
where R =*genre*, A =*genre_id*, B =*genre_name*

We have, $A \to B$. We can't get $B \to A$ since different author may have a same name.
Obviously, A is candidate key.
We conclude that, $R(A, B)$ is in BCNF.

3.4 owners

owners(owner_id, owner_name)

Let the relation schema *owners* be $R(A, B)$,
where R =*owners*, A =*owner_id*, B =*owner_name*

We have, $A \to B$, obviously, A is candidate key. And if we $B \to A$, then B would be also candidate key.
We conclude that, $R(A, B)$ is in BCNF.

3.5 publisher

publisher(pub_id, pub_name, pub_phone, pub_email, pub_address, bank_account)

Let the relation schema *publisher* be $R(A, B, C, D, E, F)$,
where R =*publisher*, A =*pub_id*, B =*pub_name*, C =*pub_phone*, D =*pub_email*, E =*pub_address*, F =*bank_account*

We have, $A \to BCDEF$, $B \to ACDEF$, $C \to ABDEF$, $D \to ABCEF$, $E \to ABCDF$, $F \to ABCDE$.
Obviously, A, B, C, D, E, F are all candidate keys.
We conclude that, $R(A, B, C, D, E, F)$ is in BCNF. as

3.6 orders

orders(order_id, cus_id, order_date, paid, status, ship_address, recipient, credit_card, order_phone)

Let the relation schema *orders* be $R(A, B, C, D, E, F, G, H, I)$,
where R =*orders*, A =*order_id*, B =*cus_id*, C =*order_date*, D =*paid*, E =*status*, F =*ship_address*, G =*recipient*, H =*credit_card*, I =*order_phone*

We have, $A \to BCDEFGHI$, and there is no other functional dependency we can get.
Obviously, A is candidate keys.
We conclude that, $R(A, B, C, D, E, F)$ is in BCNF.

3.7 book

book(book_id, inventory, pub_id, author_id, genre_id, book_name, royalty_rate, pub_date, price, isbn, num_page, on_shelf)

Let the relation schema *book* be $R(A, B, C, D, E, F, G, H, I, J, K, L)$,
where R =*book*, A =*book_id*, B =*inventory*, C =*pub_id*, D =*author_id*, E =*genre_id*, F =*book_name*, G =*royalty_rate*, H =*pub_date*, I =*price*, J =*isbn*, K =*num_page*, L =*on_shelf*

We have, $A \to BCDEFGHIJKL$, $J \to ABCDRFGHIKL$
And A, J are candidate keys.
We conclude that, $R(A, B, C, D, E, F, G, H, I, J, K, L)$ is in BCNF.

3.8 order_book

order_book(order_id, book_id, amount)

Let the relation schema *order_book* be $R(A, B, C)$,
where $R =$*order_book*, $A =$*order_id*, $B =$*book_id*, $C =$*amount*

We have, $AB \rightarrow C$, and there is no other functional dependency we can have.
Obviously, AB is candidate key.
We conclude that, $R(A, B, C, D, E, F)$ is in BCNF.

4 Database Schema Diagram

Figure 2 shows the Schema DIagram for the database.

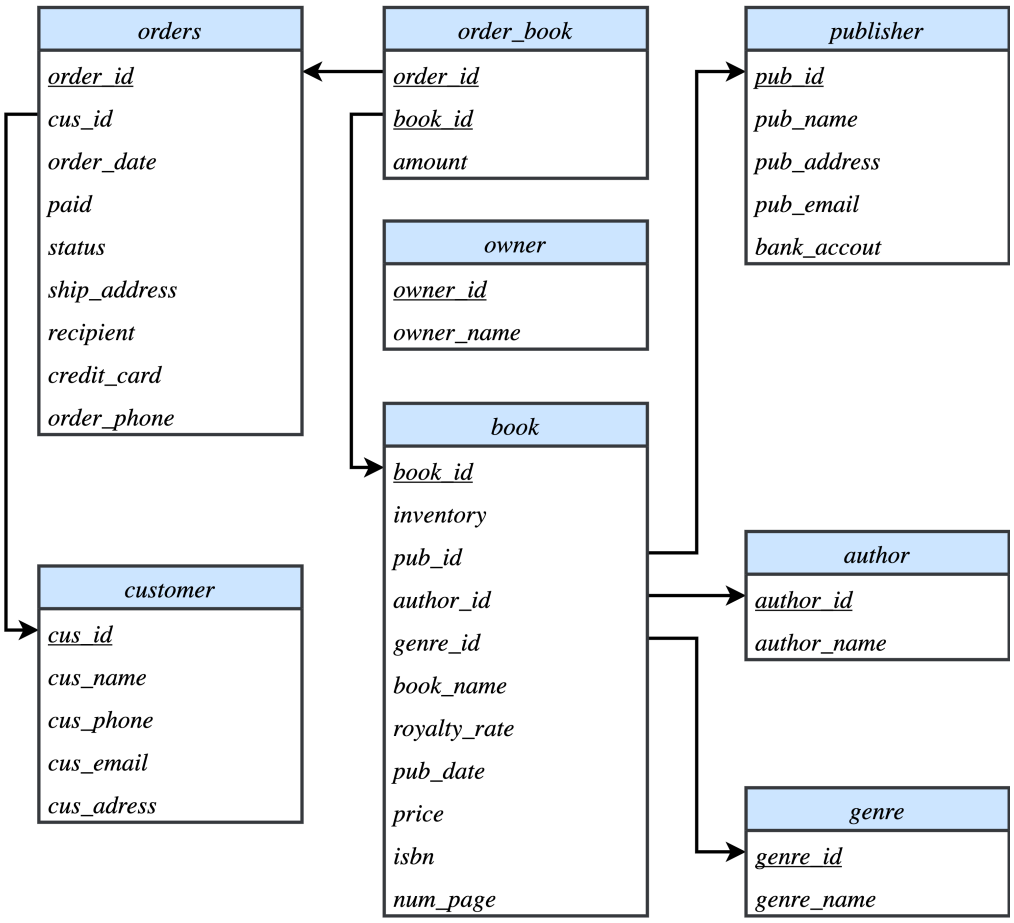

Figure 2: Databas Schema Diagram

5 Implementation

5.1 Overview of the APP

The app is implemented by Javascript+Pyhon. With a web-based client site and a RESTful API implemented by python flask Library [3] and psycopg2 Library [4]. The architecture of the our online book store app is shown in Figure 3.

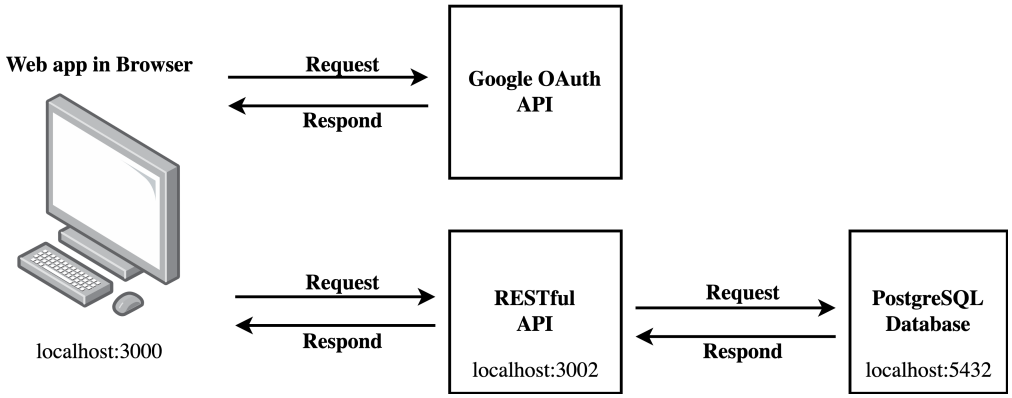

Figure 3: Architecture of the App

The web app is built with React/Redux [1] in Javascript. It used the Axios to interact with the RESTful API(which interact with the PostgreSQL Database) and Google OAuth API [2]. It uses Redux Store to manage users' states and the books added to users' basket, and Google OAuth API to register/login, which means that users and admins can login with their google accounts. Users can login and add books into the basket, after checking the total amount to pay, users can place orders by entering shipping and billing information. After placing the order, the user will receive a confirmation number. Users can also check their order information by using confirmation number to search in the User Center page. Users who have admin privilege can manage the bookstore by adding and deleting books. They can also check the selling report on the Admin Center page.

5.2 Login with Google Account

Figure 4 shows the main page of our app.

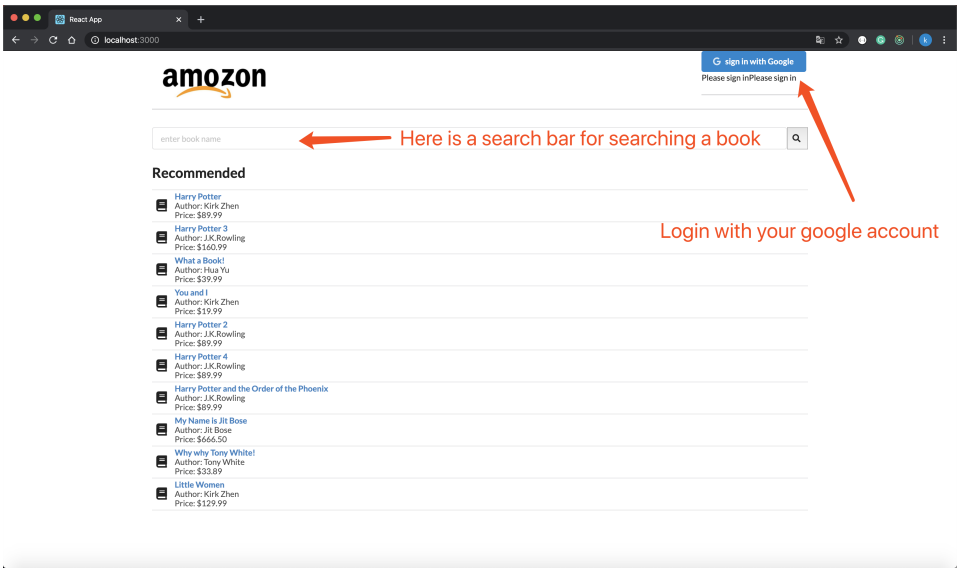

Figure 4: Main Page

After clicking the Google login button, you would be asked to login with your google account, as shown in Figure 5. Only two accounts are authorized as admin (owners of the book store). In the following subsections, we would show the interface for customers and admin.

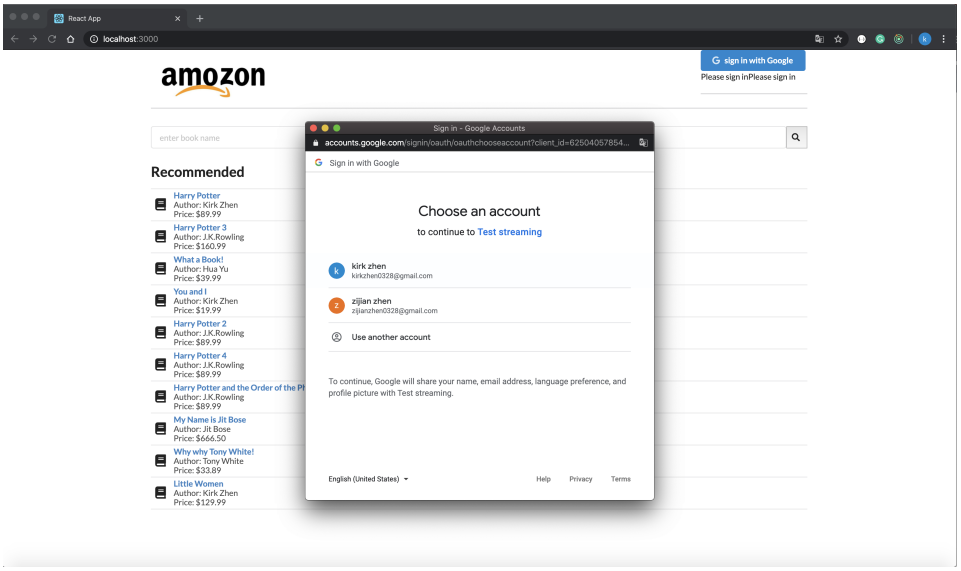

Figure 5: Login with Google Account

5.3 Customer Interface

After login in as a customer, you will see a list of recommended books.

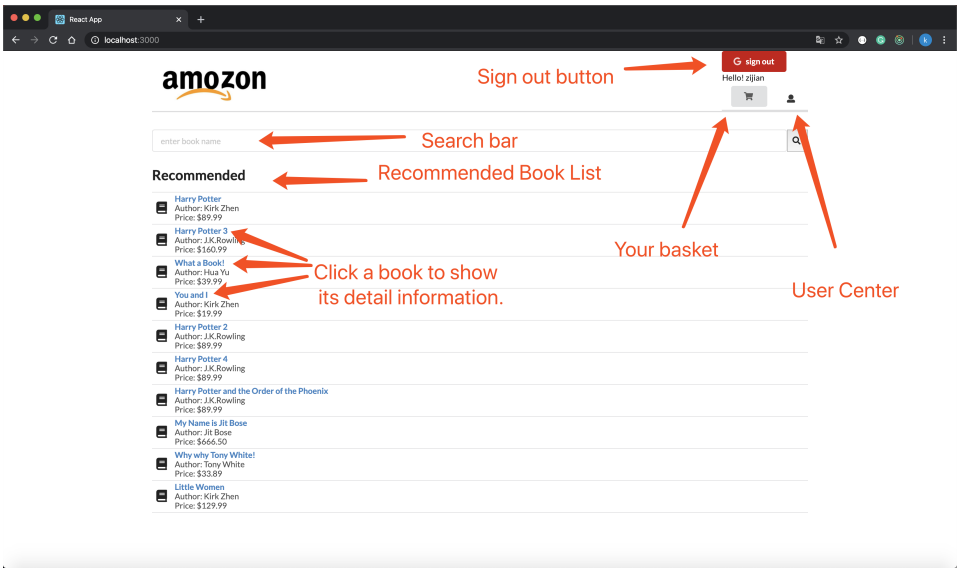

Figure 6: Customer Main Page

Clicking into a book, for example *"Harry Potter and the Order of the Phoenix"*, you will see the detail information about this book, as shown in Figur 7. And now, you can choose to add it into your shopping basket.

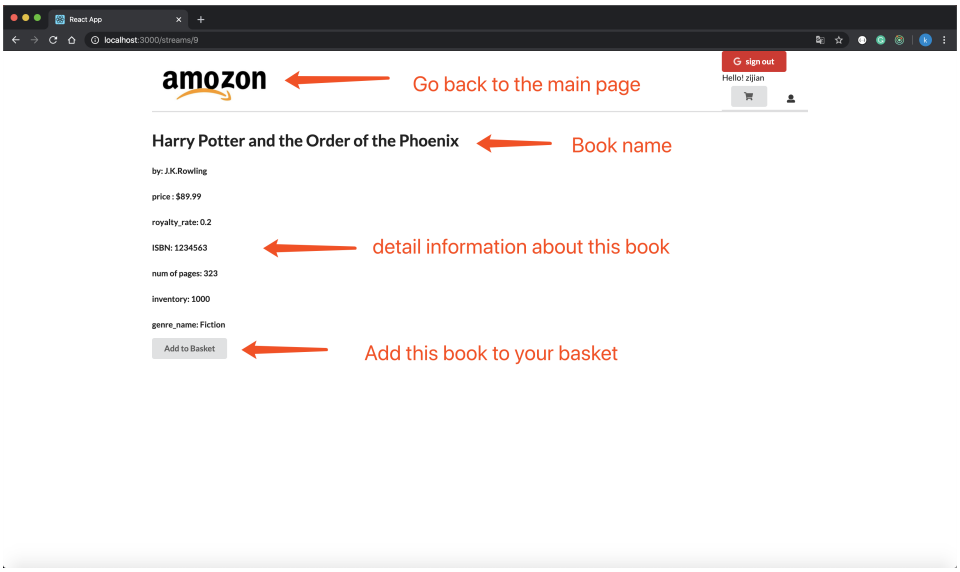

Figure 7: Book Information

After adding some books into your basket, you can choose to check out by clicking the check out Button.(Figure 8)

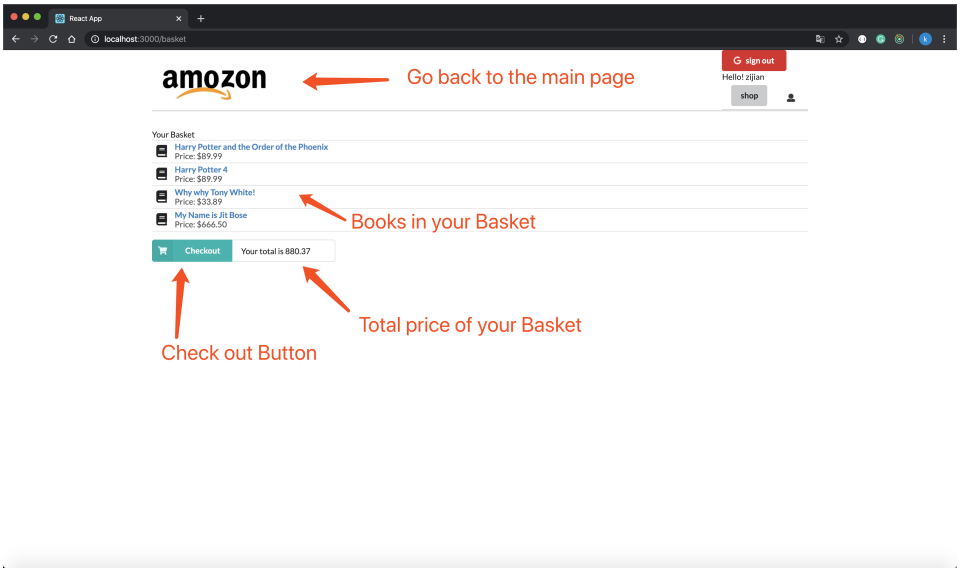

Figure 8: Shopping Basket

Then input some necessary information for our order (for shipping and payment), and click the pay Button to pay, as shown in Figure 9

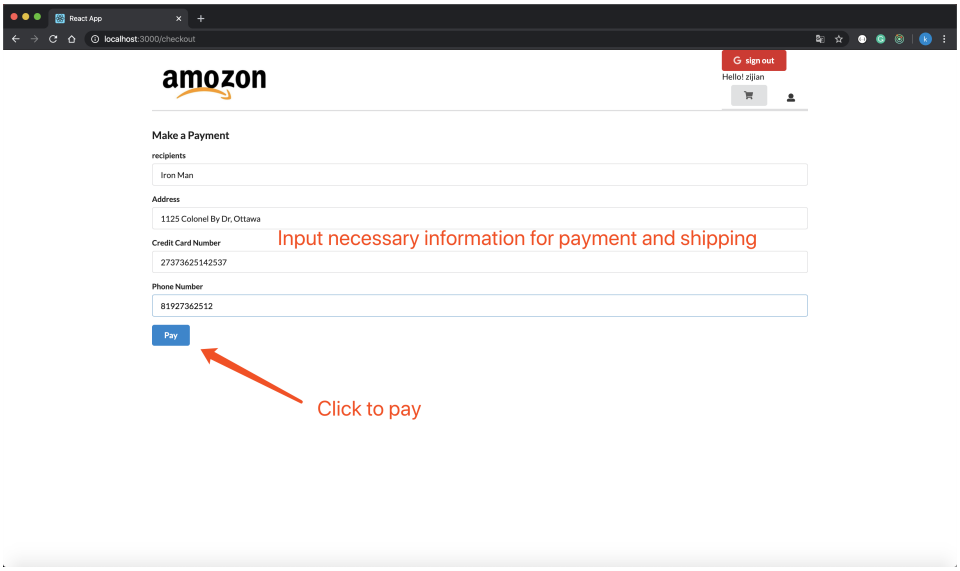

Figure 9: Make payment

After paying, the User Center page would be shown, and you can see your order number. You can also track the information about your orders by inputting the order number in the input box, as shown in Figure 10
After paying, the system would check the inventory(remaining amount) of the books in the order, if the inventory is less than 10, then the system would send an email to the publisher. **This functionality is implemented in api.py, and not shown in the interface.**

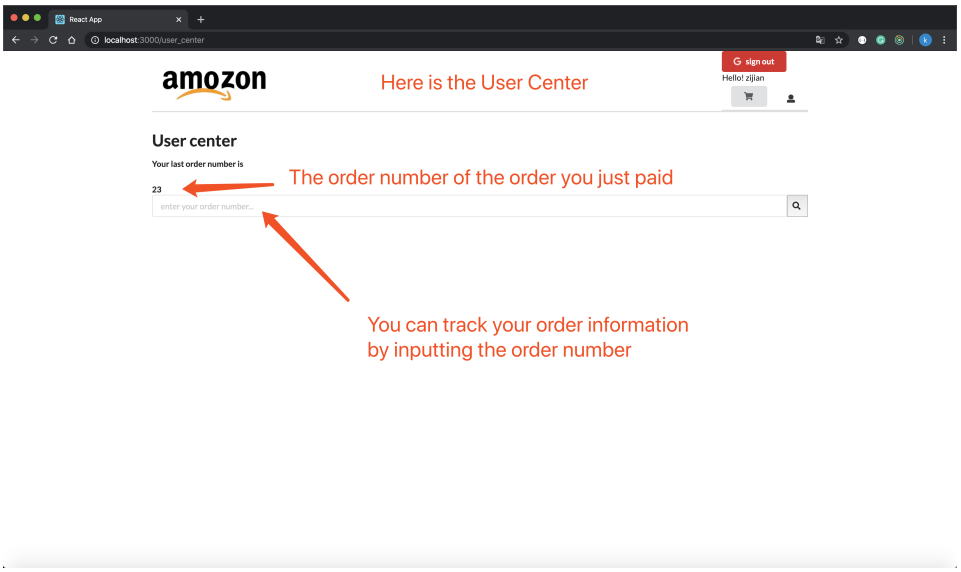

Figure 10: After Paying

For example, you would like to track the order you just created, you can input the order number "23", and click the search button on the left, then the information would be shown.(Figure 11)

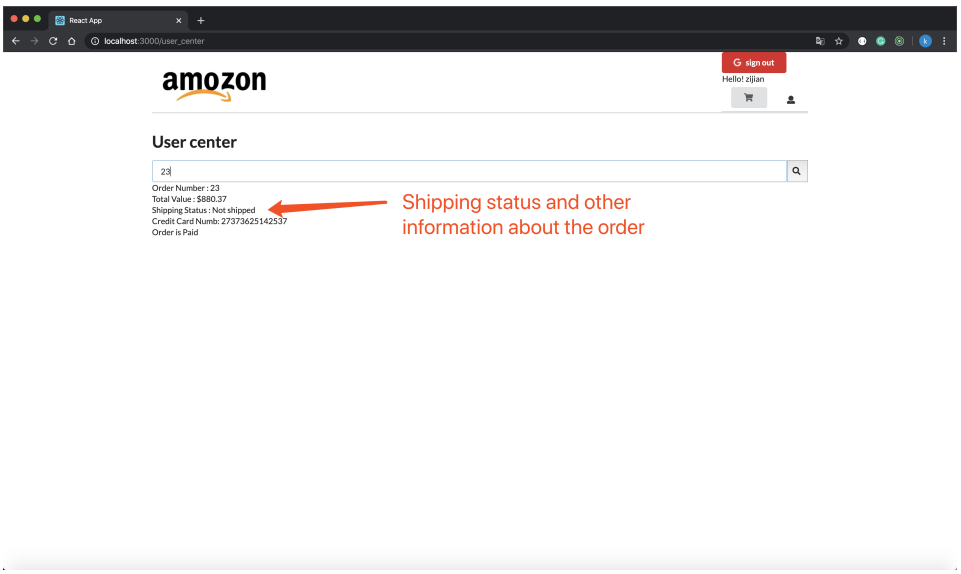

Figure 11: Track Order

5.4 Admin Interface

After your account is authorized as a admin, you can log into a Admin main page, shown in Figure 12

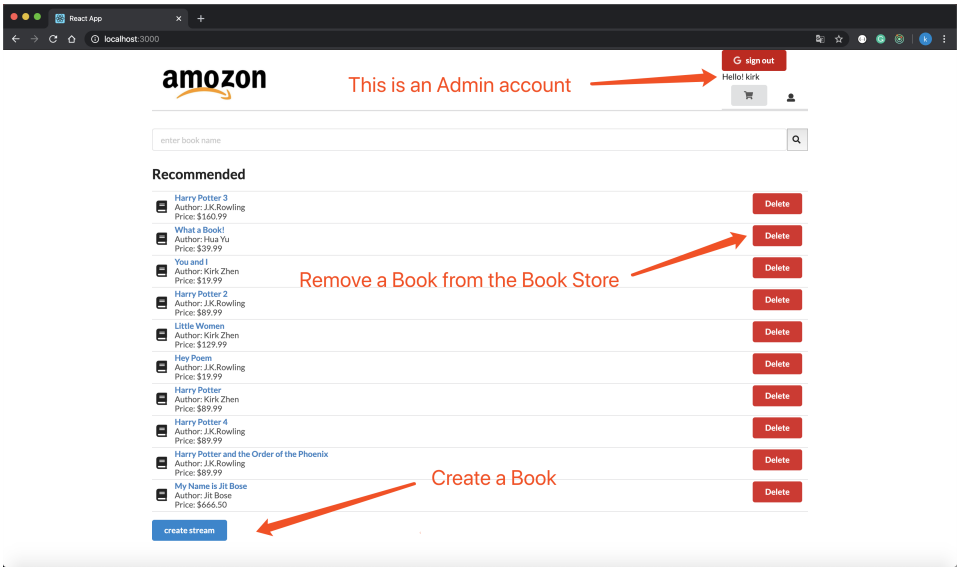

Figure 12: Admin Main Page

Click the Create book button, you can get into a page for creating a book, as shown in Figure 13. You can input some information about the book and click submit button to create the book.

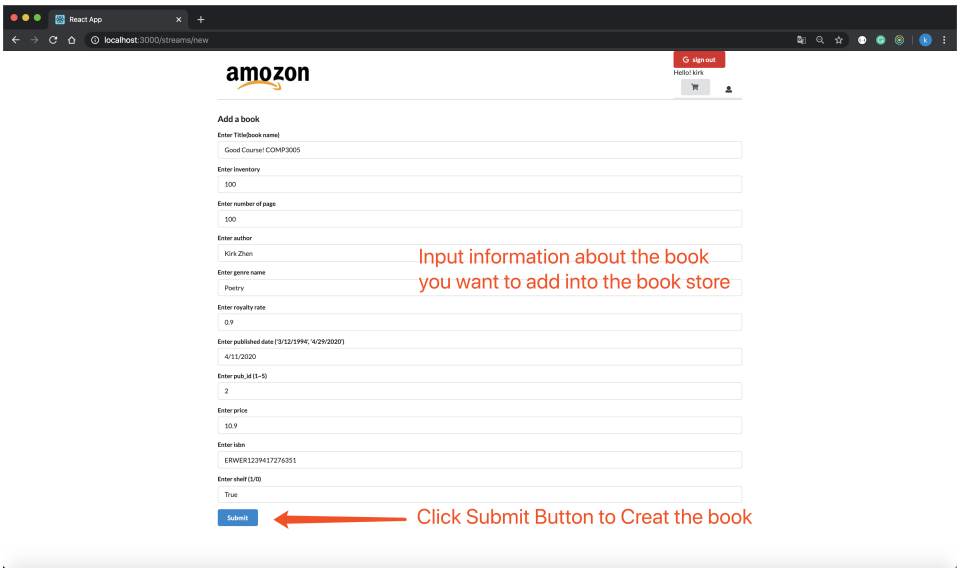

Figure 13: Create a Book

After Clicking the submit button, you have successfully create a book. Search the book by inputting keyword, we can see that the book in created in the store. (Figure 14).
Now, we would see how to remove a book from the store. We can click the delete button to remove a book.

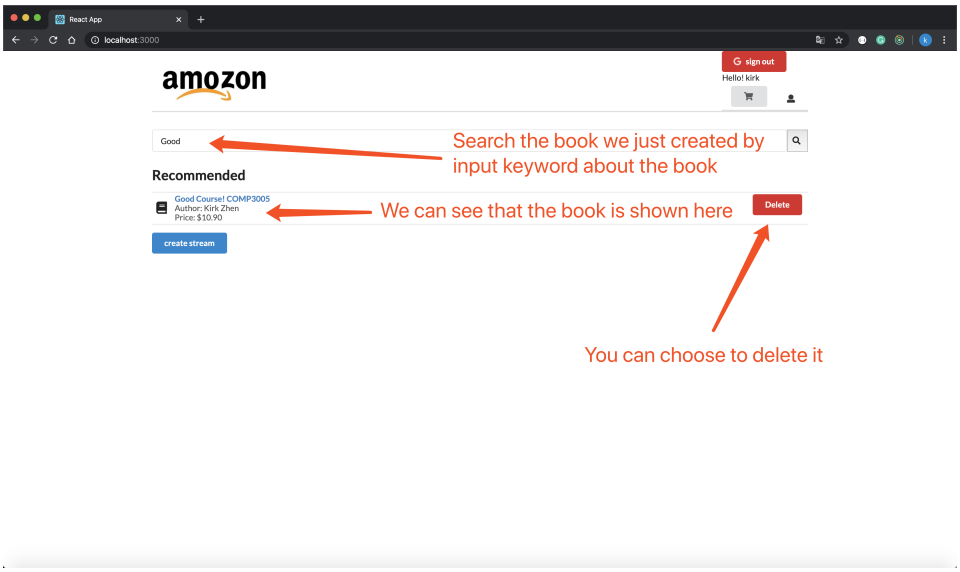

Figure 14: Successfully Create a Book

After clicking the delete button, a pop up window would appear to make sure whether you want to remove the book. (Figure 15)

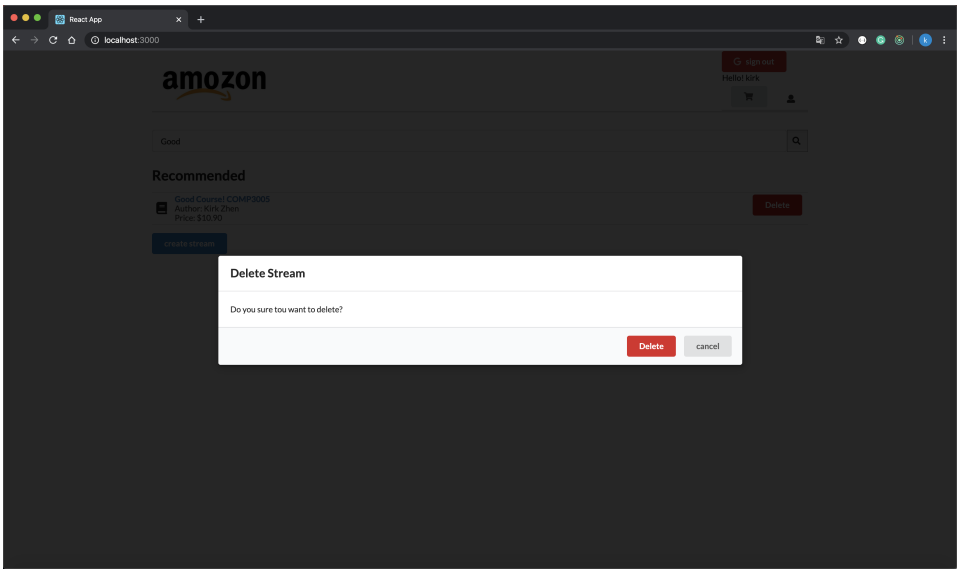

Figure 15: Delete a Book

Click Delete, and the book would be removed. Search the book by inputting keyword, we can see that the book in no longer on the shelf.

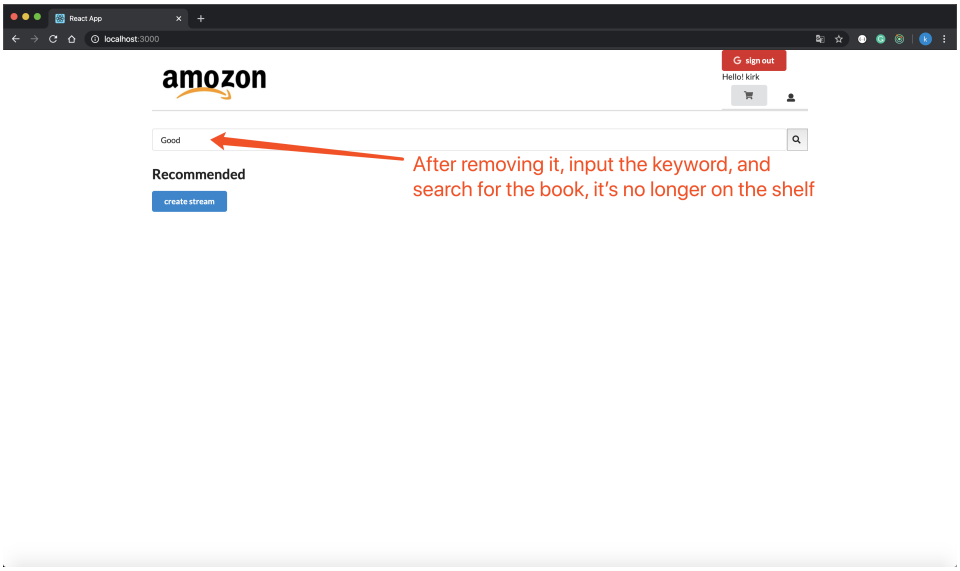

Figure 16: Successfully Remove a Book

Admin can also access the Sales Report in the Admin Center. You can click the little man on the top right to get into the Admin Center (Figure 17). You can input a genre name and author name to access the report for a specific genre and a specific author.

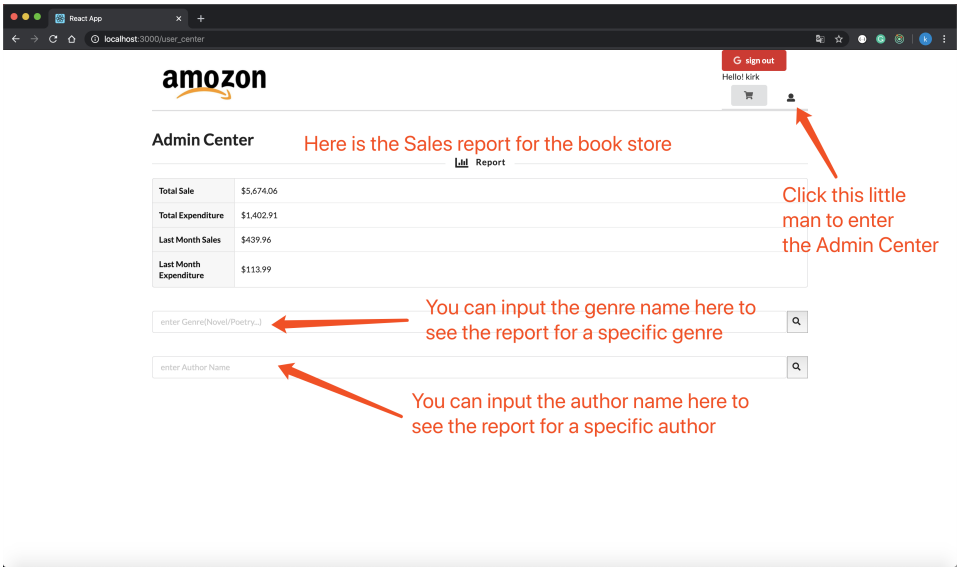

Figure 17: Admin Center(Sales Report)

After input a genre name and author name and click the search button. We can access the report for them, as shown in Figure 18.

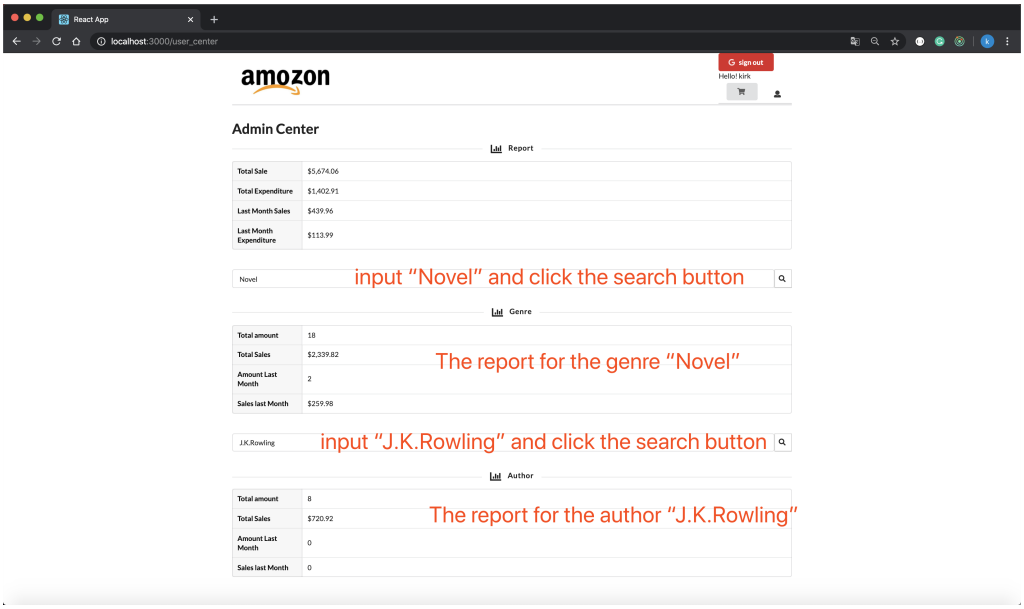

Figure 18: Genre Report and Admin Report

6 Searching Functionality(Bonus Features)

Our app is capable to search for approximate matching. In this subsection, we will show how this works.
In our database, there are multiple books about "Harry Potter", for example, "Harry Potter and the Order of the Phoenix", "Harry Potter 3", "Harry Potter 4", etc. If we search for "Harry Potter", all these book would be shown in the searching results.(Figure 19)

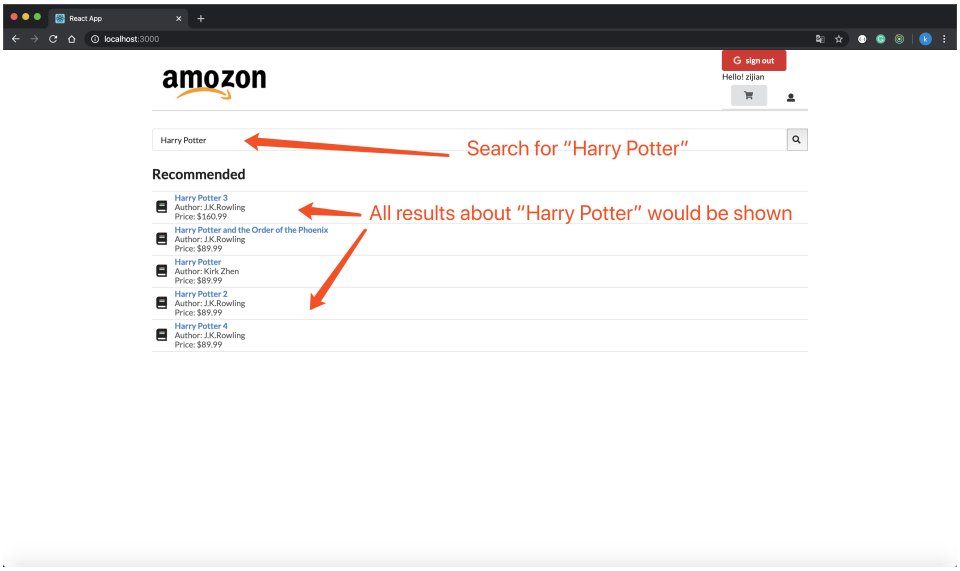

Figure 19: Search for "Harry Potter"

You can also search books for a specific genre. For example "Fiction", "Novel", "Poetry". Then books in this genre would been shown. (Figure 20)

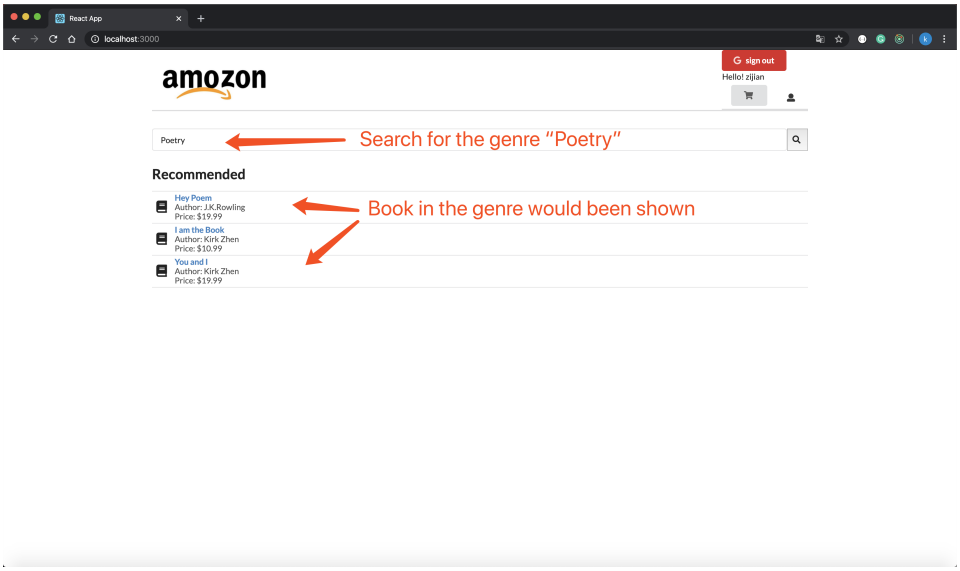

Figure 20: Search for the genre "Poetry"

The system is also capable to handle typos when the user input some keyword to search. For example, the user want to search books by "Kirk Zhen", but he input "Kirk Zhems", here is a typo. Our system is able to correct this typo, and show the book written by "Kirk Zhenn".(Figure 21)

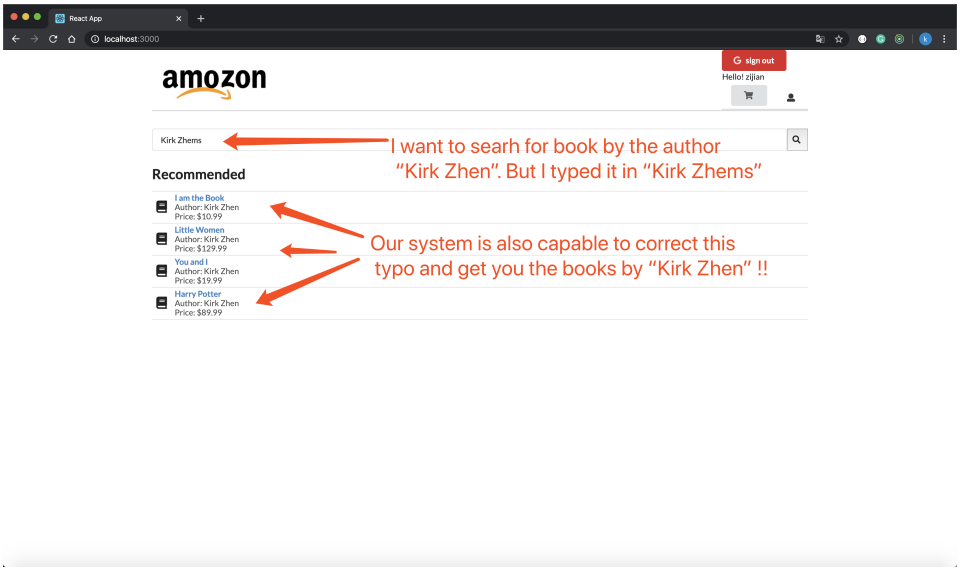

Figure 21: Search for the author "Kirk Zhen" with typo

7 GitHub Repository

All the code for this project can be accessed on: `https://github.com/moxxxx/Amozon`

References

[1] "Redux - A Predictable State Container for JavaScript Apps.: Redux." A predictable state container for JavaScript apps. Accessed April 13, 2020. https://redux.js.org/.

[2] "OAuth 2.0 for Client-Side Web Applications — Google Identity Platform." Google. Google. Accessed April 13, 2020. https://developers.google.com/identity/protocols/oauth2/javascript-implicit-flow.

[3] "Welcome to Flask." Welcome to Flask - Flask Documentation (1.1.x). Accessed April 13, 2020. https://flask.palletsprojects.com/en/1.1.x/.

[4] "Psycopg 2.8.5 Documentation." Psycopg. Accessed April 13, 2020. https://www.psycopg.org/docs/index.html.