COMP 3005 Final Project

Book Store Application
December 19, 2021

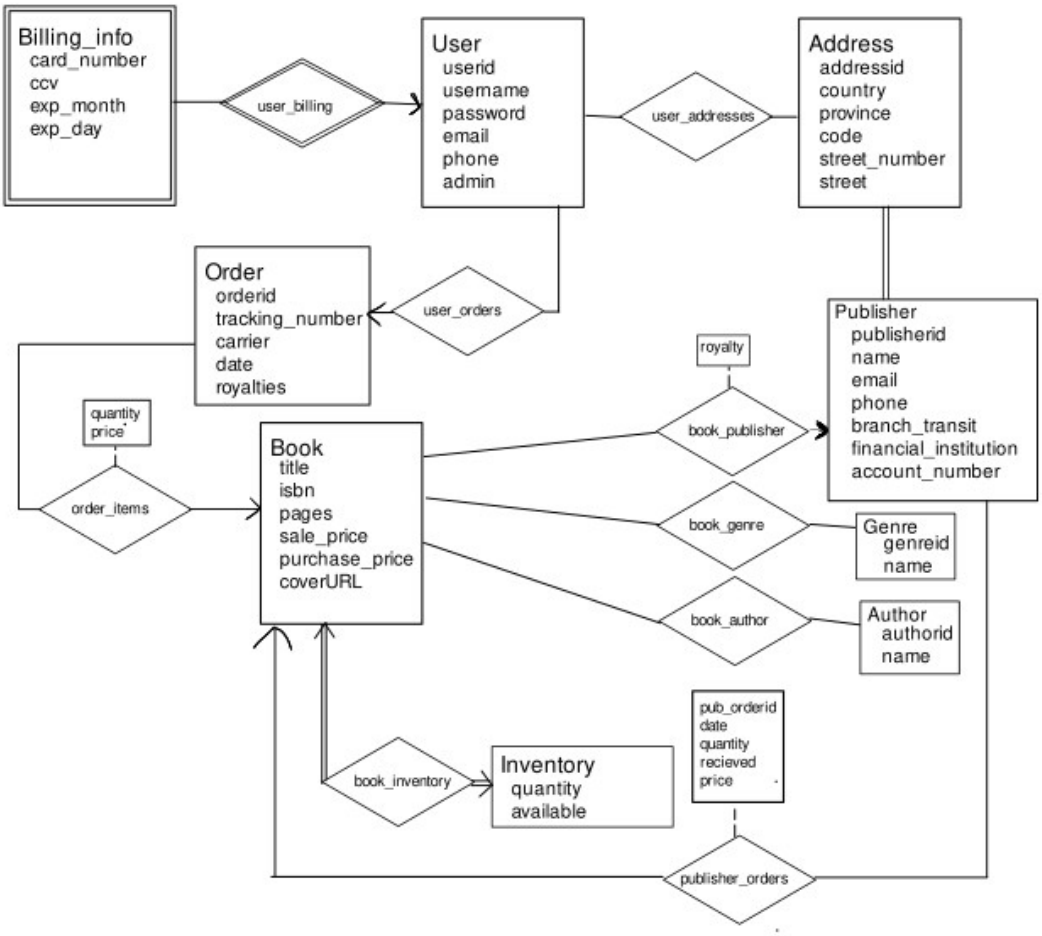
Emma Chadwick (100993911) + Clark Bains (101149052)

2.1 – Conceptual Design

Below are some assumptions we made during the design phase, and the ER diagram we generated based off of them

- Billing
  - Billing info cannot be deleted/edited,
  - Billing info must belong to a user.
- Users
  - Users must have unique usernames, as they login with this
  - A user can have multiple addresses
  - A user can have multiple orders
  - A user can only be an admin or a user. There are no finer grained access controls
- Addresses
  - Two different copies of the same address can exist (2 users who live together)
  - Address could be shared between Publisher and Users, so if publishers are ordering books, they can be shipped to the current address of a user.
- Orders
  - A "Order" is only the kind of order that a user makes, not the store ordering more books from the Publisher
  - Order can have 1 quantity of a specific item. So maybe 4 x book1 and 3 x book2, but not 2 x book1 and 3 x book1 and 3 x book2
  - Tracking state (like location) is not stored. In the real world this would be delagated to the third part carrier, and the retailer would only keep a carrier and tracking number
- Books
  - Books always have an isbn, that is an integer
  - Books can only have one publisher
  - Books can have 0 to multiple authors (anonymously published) and genres
  - Deleted books are only ever marked as unavailable. This allows valid reports to be generated
  - Books can be repriced without retroactively changing order data
- Genres
  - Whole genres can be deleted (and all books will update) (for example, if a genre becomes no longer socially acceptable)
  - Genres do not need to have a book that includes them.
- Authors
  - Two authors with the same name can be treated the same, there is no additional information for an author

    - Authors do not need to have a book that includes them. (application has search functionality for both, so imagine importing a large list, to make searching easier when adding books.)
- Publisher
    - Publisher orders can only be for one book at once.
    - Publishers can only have one bank account
    - Publisher orders are not fufilled automatically (This is more realistic)
    - Publisher orders may not all be recieved at once (but always are in this system)
    - In addition to royalty, there is a purchase price that must be paid to the publisher every time a new order is made
- Other
    - The min inventory for a book is a static constant throughout the system, not made to be easily changed through the application itself
    - Carts do not need to be stored server-side
    - Sensitive Information (CC numbers, passwords) can be stored in plain-text



2.2 – Reduction to Relation Schemas

Many sets require an id to nicely identify them to the user when editing. For example, if editing a Billing_info entry, to uniquely identify it, you need all attributes, which would typically lead to frontend urls such as /ui/billing/edit?card_number=XXXXXXXXXXXX&ccv=XXX&exp_month=XX&exp_year=XX. This is even worse for addresses, so for relations where we want to allow editing, we have to add a id. This is not done for the sole purpose of making normalization easier byy almost always having a superkey.

Billing_info(userid, card_number, ccv, exp_month, exp_year)
Is a weak entity set, so it needs userid.

Users(userid, username, password, email, phone, admin)

User_Address(userid, addressid)
Need this relation so that publishers can also use a given address

Address(addressid, country, province, code, street_number, street)

Orders(orderid, userid, tracking_number, carrier, date, price, royalties)
Total Participation with inventory, so it can be integrated into this relation.

OrderIterms(orderid, isbn, quantity)

Books(isbn, title, papges, sale_price, purchase_price, coverURL, available, quantity, royalty, publisherid, royalty)

Publisher(publisherid, addressid, name, email, phone, addressid, branch_transit, financial_institution, account_number)

Genres(genreid, name)

Authors(authorid, name)

Book_genre(isbn, genreid)

Book_author(isbn, authorid)

Publisher_orders(pub_orderid, isbn, publisherid, date, quantity, recieved, price)

2.3 – Normalization

Billing_info(userid, card_number, ccv, exp_month, exp_year)
userid, card_number, ccv, exp_month, exp_year-> userid, card_number, ccv, exp_month, exp_year
All attributes form the candidate key, so this is in 3NF

Users(userid, username, password, email, phone, admin)
userid->username, password, email, phone, admin
username->password, email, phone, admin
userid and username are both super keys, so this is in 3NF

User_Address(userid, addressid)
userid, addressid -> userid, addressid
candidate key is userid, addressid
In FD, X->Y, where y is userid, addressid, both elements are part of the candidate key, so this is 3NF

Address(addressid, country, province, code, street_number, street)
addressid->country, province, code, street_number, street
code -> province, city
candidate key is (addressid, country, province, code, street_number, street)
addressid is a superkey, and province,city is in a candidate key, so this is in 3NF

Orders(orderid, userid, tracking_number, carrier, date, price, royalties)
orderid->userid, tracking_number, carrier, date, price, royalties
Orderid is a superkey, so this is in 3NF

OrderIterms(orderid, isbn, quantity)
orderid, isbn -> quantity
Orderid,isbn form a superkey, so this is in 3NF

Books(isbn, title, papges, sale_price, purchase_price, coverURL, available, quantity, royalty, publisherid)
isbn-> title, pages, sale_price, purchase_price, coverURL, available, quantity, royalty, publisherid
Isbn is a superkey, so this is in 3NF

Publisher(publisherid, name, email, phone, addressid, branch_transit, financial_institution, account_number)
publisherid -> name, email, phone, addressid, branch_transit, financial_institution, account_number
publisherid is a superkey, so this is in 3NF

Genres(genreid, name)
genreid->name
genreid is a superkey, so this is in 3NF

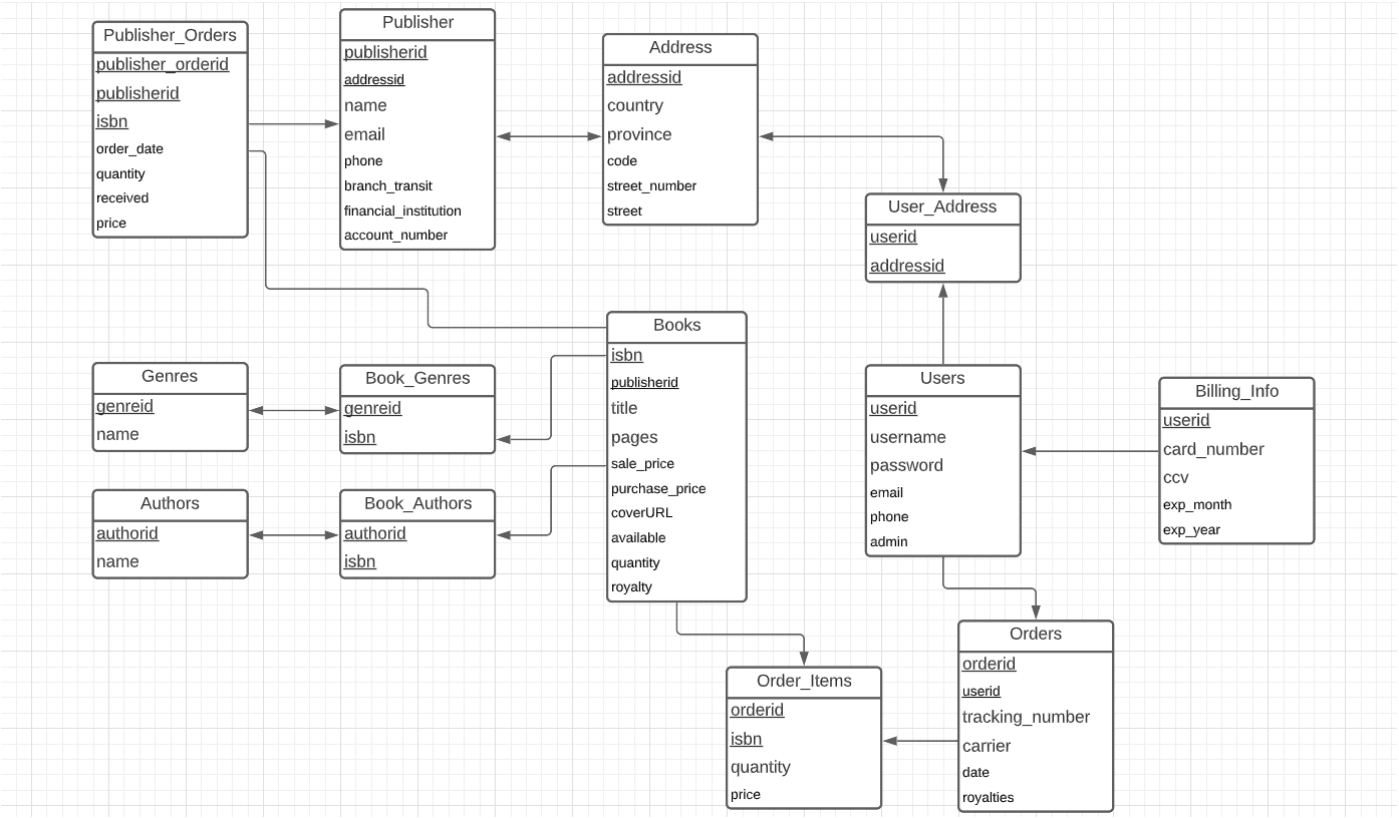
Authors(authorid, name)
authorid->name
authorid is a superkey, so this is in 3NF

Book_genre(isbn, genreid)
isbn, genreid-> isbn, genreid
isbn, genreid is a candidate key, so this is in 3NF

Book_author(isbn, authorid)
isbn, authorid-> isbn, authorid
isbn, author is a candidate key, so this is in 3NF

Publisher_orders(publisher_orderid, isbn, publisherid, date, quantity, recieved, price)
publisher_orderid-> isbn, publisherid, date, quantity, recieved, price
publisher_orderid is a superkey, so this is in 3NF

2.4 – Database Schema Diagram



2.5 – Implementation

Backend

The backend is a nodejs server paired with a better-sqlite3-pool libaray. With the event loop architecture of nodejs, this allows for a fairly responsive application, that can process multiple requests concurrently.

It uses the callback architecture so often found in javascript projects to effectively process data, for example, the pagination, where a utility accepts two functions, getData(limitStr) and getCount(), to generate a new function, that when called with information from the http request, allows it to construct, execute and return data in the form of a paginated response easily.

As it was not the focus of the project, the backend lacks a significant amount of security. By messing with request headers, it is easily possible to emulate another user. This was done due to Cross Origin Request Sharing issues, as the project runs with the frontend on a different domain than the backend. Beyond that, it is missing anything more than basic checks on most endpoints. For example, as long as a user is signed in, they can request any billing information, which would leak credit card numbers. It additionally omits most validation, relying entirely on the database, which has little validation beyond not null and foreign key constraints.

Frontend

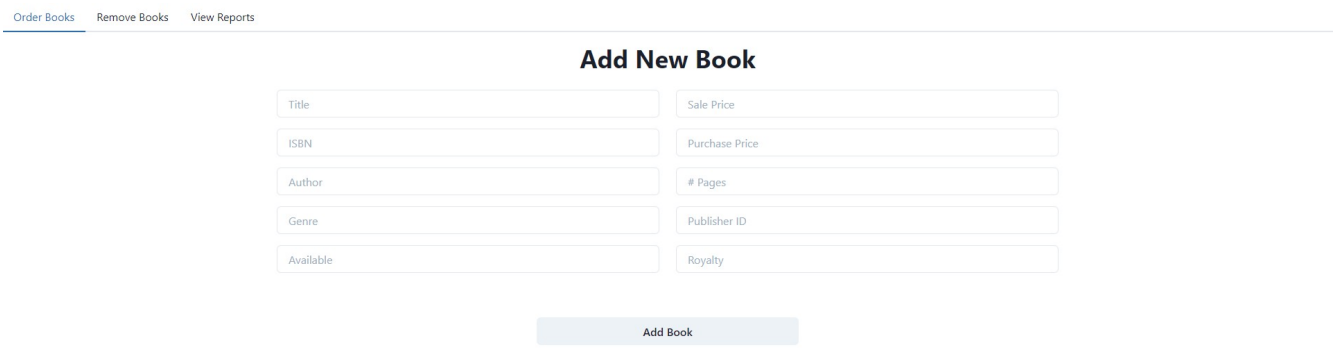


Figure 1: Adding a new book as an admin

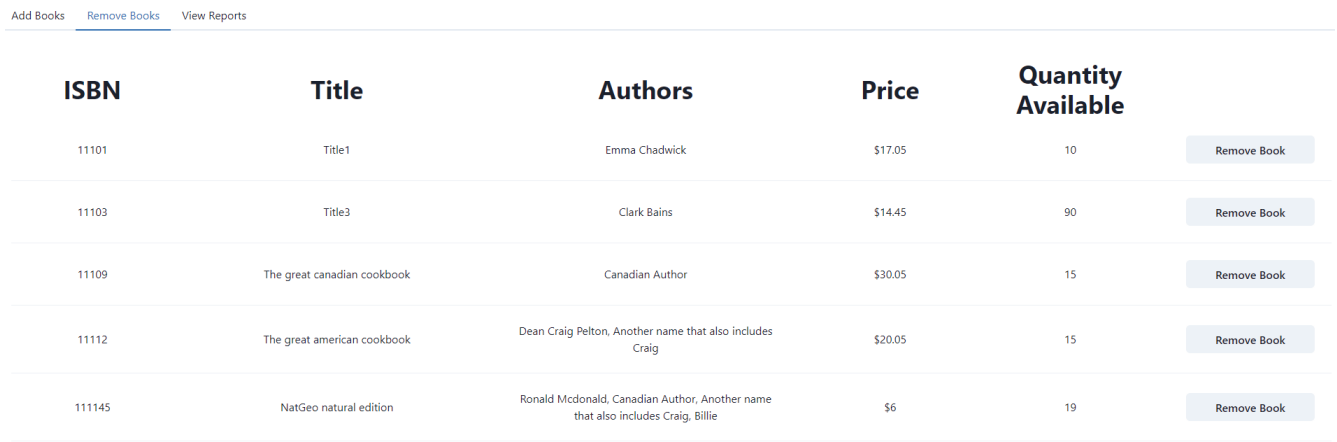


Figure 2: Menu to remove books as an admin

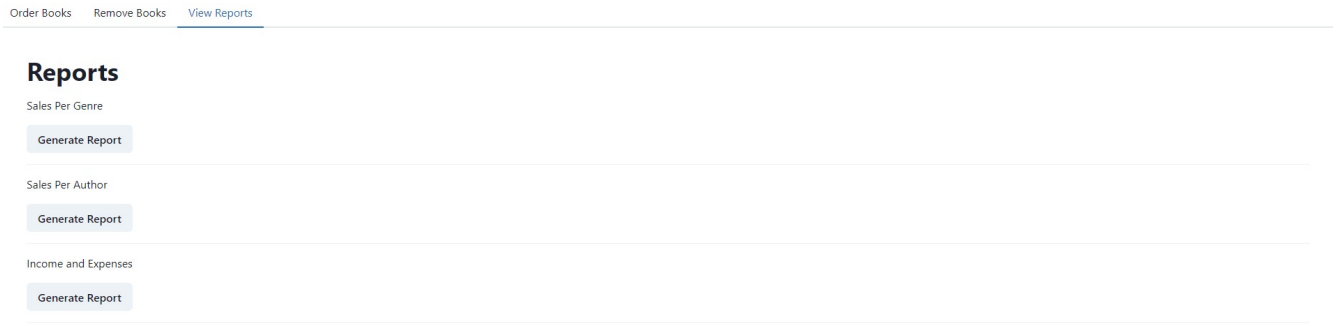


Figure 3: Report Generation as an Admin

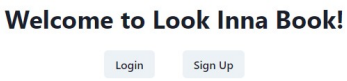


Figure 4: Home Page

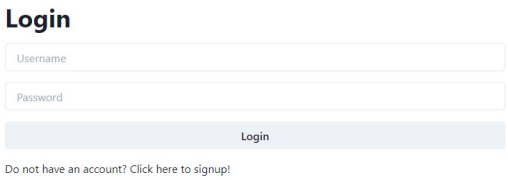
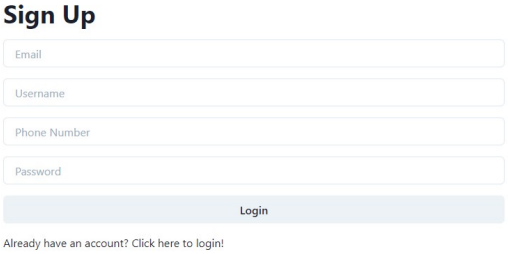


Figure 5: Login Page



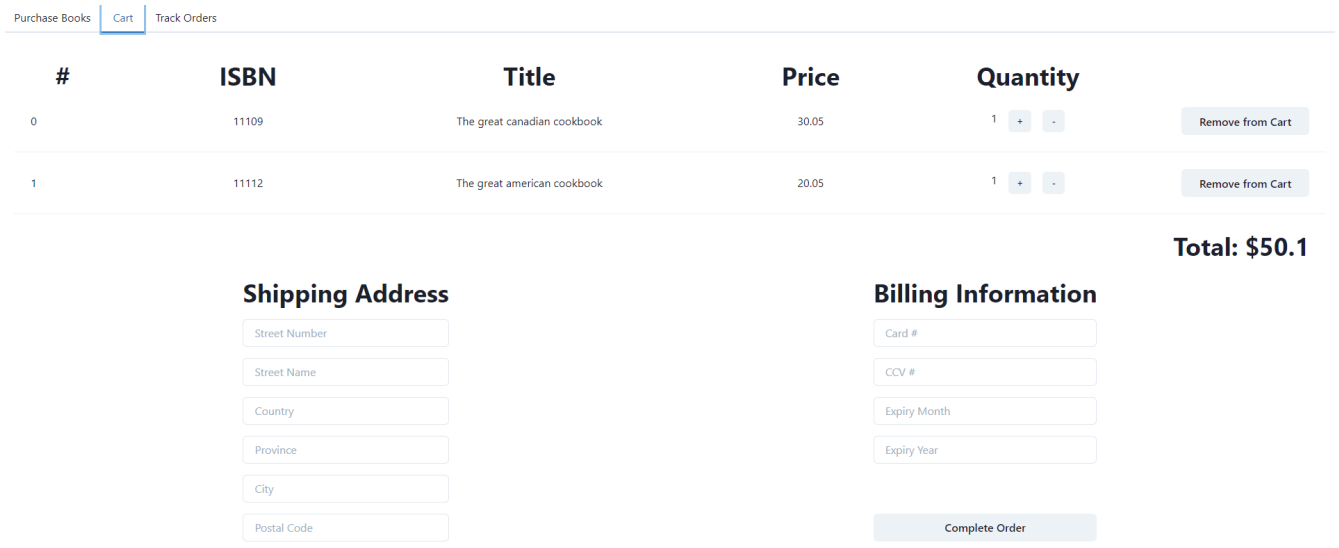


Figure 7: User Cart View

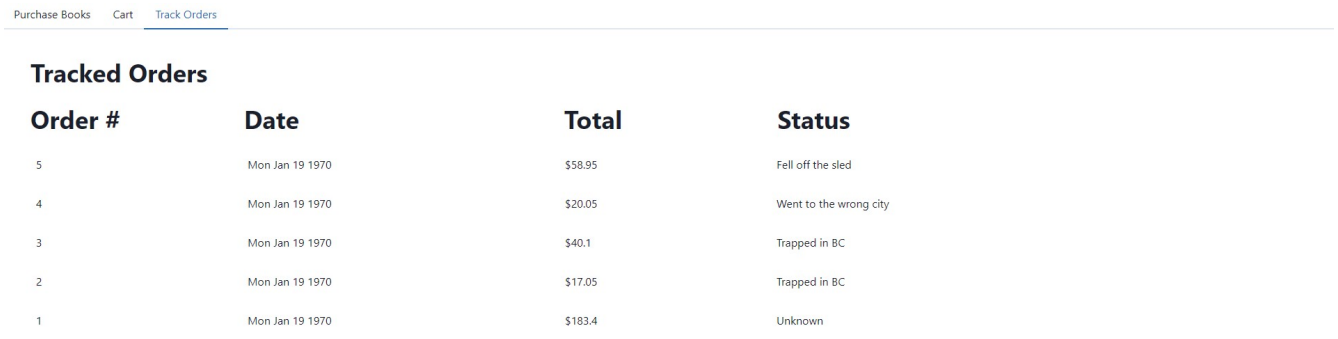


Figure 8: User Order View

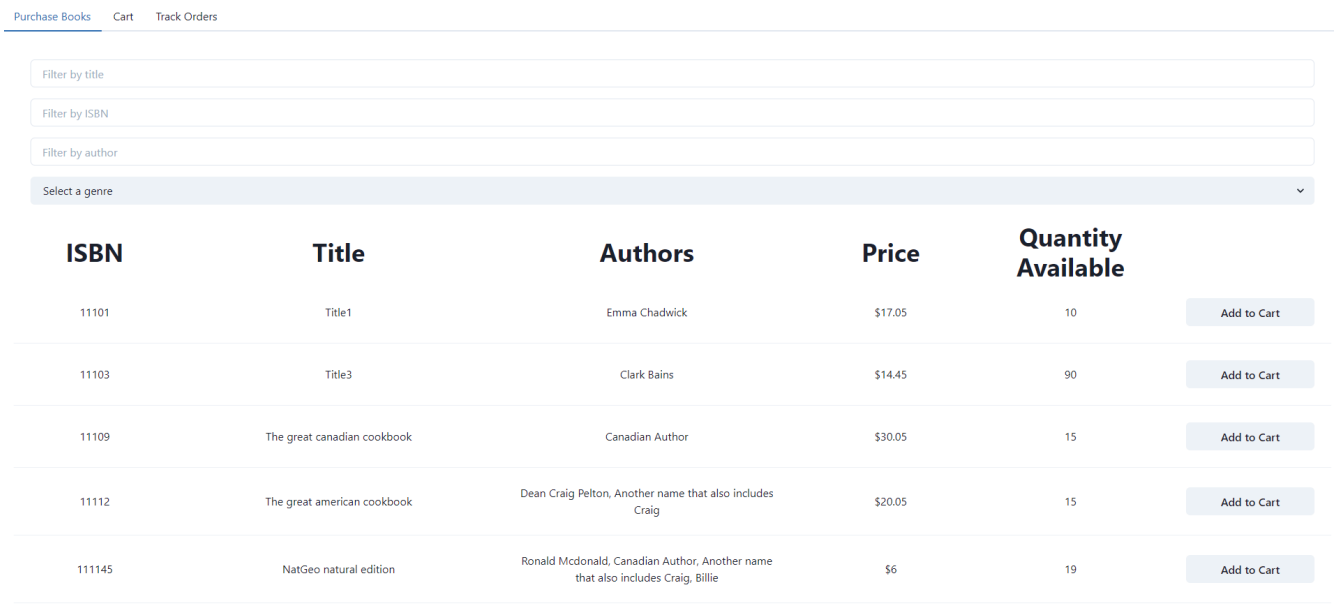


Figure 9: User Search View

2.6 – Bonus Features

Docker

This package is built as a set of 2 docker containers, one handling front end, and one handling back end. This enables easy deployment on most Linux servers, where docker is increasingly deployed in the world of containerized micro services. Also included is a docker-compose file, which contains all the configuration needed for the docker deployment, such as which data should be persisted across container restarts, and what ports should be forwarded from the container's network name space.

Continuous Delivery

This project includes my standard Continuous Delivery script, a small nodejs server that listens on a port I forward to the outside world, and then it listens for web requests from a Github project configured to deliver on a push. This enables it to update the project, and then rebuild using docker, and redeploy, all automatically. This process happens nearly instantaneously, taking only a second or two to deploy, if the build process can effectively leverage docker's cache. The script is also able to send Discord notifications when a commit is made, which is helpful for keeping your working tree up to date when working in a collaborative environment, such as with a partner.

Fuzzy Searching

The search endpoint, allows for a CSV list of possible titles to be submitted. This is also true for the author and genre endpoints, with the name attribute. Internally, these are split into a list, and then have wildcards appended to each end of each of the split values, and then are pushed into an array, along with the appropriate clause for the query itself, for example "(name like ?)" The query clauses are all joined with either an AND or OR, depending on the situation, and are appended into the query string, which the modified split values are passed as parameter values to the query.

Dynamic + Pretty

The front end is created with react, and is dynamic, which creates a far smoother user experience, as there are minimal reloads required as you navigate the site. It also uses premade CSS libraries, to provide a very visually appealing experience, compared to what could be achieved manually in the same time period.

2.7 – GitHub Repository

The GitHub repository can be found here, at https://github.com/clarkbains/3005Final.

2.8 – Appendix I

Availability of both members is 9am to 12am