COMP 3005: Database Management Systems **(Due: Friday, Dec 17, 2021 (11:59 PM))**

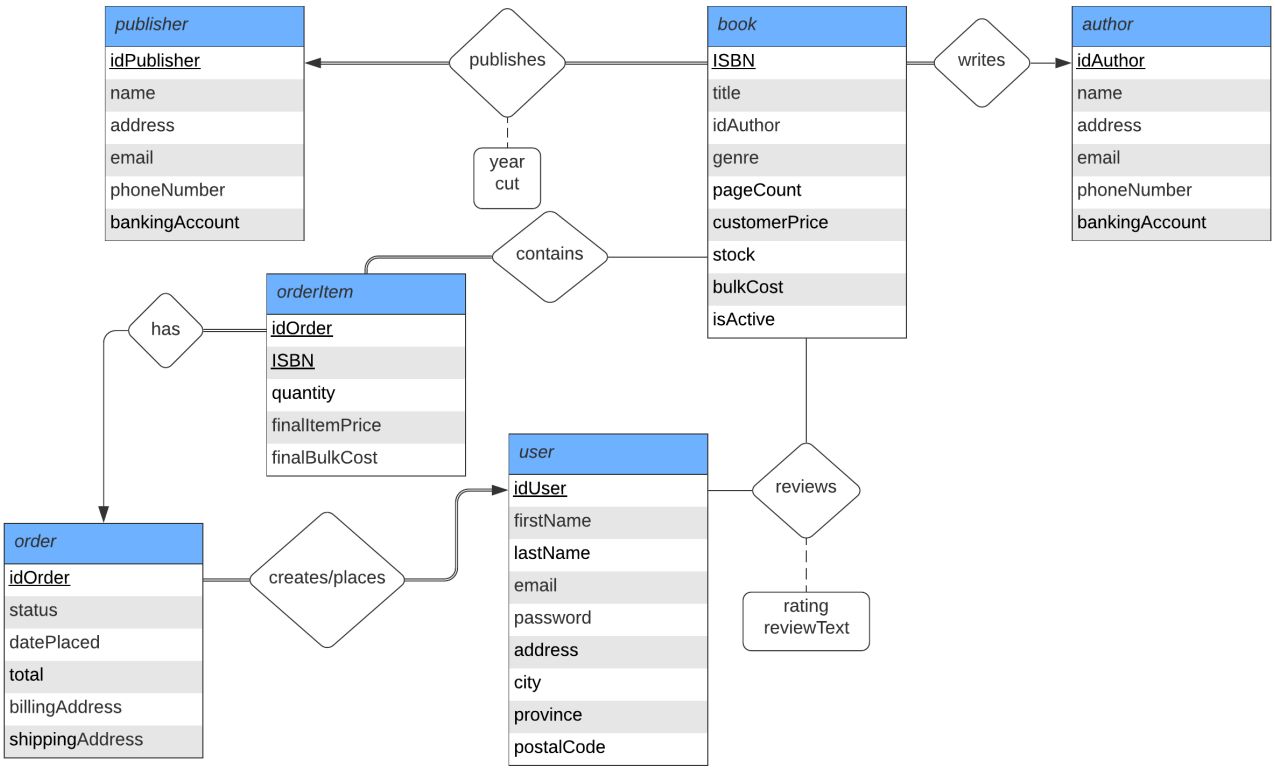
COMP 3005 Project (Fall 2021)

Instructor: Ahmed El-Roby *Name:* Matthew Hobbs / Adam Koziak, *ID:* 101145836 / 101140761

Design and implement an application for an online bookstore (Look Inna Book). This application lets users browse a collection of books that are available in the bookstore. A user can search the bookstore by book name, author name, ISBN, genre, etc.. When a book is selected, information on the author(s), genre, publisher, number of pages, price, etc. can be viewed. A user can select as many books as she likes to be added to the checkout basket. A user needs to be registered in the bookstore to be able to checkout. When checking out, the user inserts billing and shipping information (can be different than those used in registration), and completes the order. The bookstore has the feature of tracking an order via an order number. A user can use this order number to track where the order is currently. Although shipping is carried out by a third-party shipping service, the online bookstore should have the tracking information available for when the user inquires about an order using the order number. Assume all books are shipped from only one warehouse (no multiple order numbers for multiple books shipped from multiple warehouses). The bookstore owners can add new books to their collections, or remove books from their store. They also need to store information on the publishers of books such as name, address, email address, phone number(s), banking account, etc.. The banking account for publishers is used to transfer a percentage of the sales of books published by these publishers. This percentage is variable and changes from one book to another. The owners should have access to reports that show sales vs. expenditures, sales per genres, sales per author, etc.. The application should also be able to automatically place orders for new books if the remaining quantity is less than a given threshold (e.g., 10 books). This is done by sending an email to the publisher of the limited books to order a number of books equal to how many books were sold in the previous month (you do not have to implement the email sending component).

---

1 Conceptual Design



Entities:

book represents a single book offered at the store. It contains all pertinent information about the book like the required details to display and sort by in the catalogue, stock count for the restocking triggers, and pricing/availability information required for the store stats.

author represents the author of a book, containing personal information to display to a user, and banking information to take a cut of the author's book sales.

publisher represents a single publisher. Similar to *author* in that it contains basic information and banking details.

user represents a single user registered with the store. It contains all of the user's information from logins to address.

order represents a single order created or placed by a user. It contains billing and shipping addresses, as well as an order status which varies from incomplete orders (user's basket) to the shipping status of completed orders. Everything except the *idOrder* and *status* attributes will remain null while the order status is incomplete, and upon being placed will be filled with the appropriate values.

orderItem represents a single item within an order. It has information about the quantity of a specific book within an order, as well as a snapshot of the price and cost of that book at the time of purchase in order to ensure accurate store stats if prices happen to change. The *finalItemPrice* and *finalBulkCost* attributes will remain null until the associated order is finalized, at which point they will be filled with the appropriate values.

Relationships:

writes is a is many-to-one relationship from *book* to *author* because an author may write many books, but it is assumed that each book may only have one author. *book* has total participation because every single book must have an author.

publishes is a many-to-one relationship from *book* to *publisher* because a publisher can publish many books, but a book only has one publisher. *book* has total participation because every single book in the bookstore must be published. *publishes* has attributes *year* and *cut*, which respectively represent the year the book was published in and the publisher's cut of that book's sales.

reviews is a many-to-many relationship between *user* and *book* because a user may review many books, and a book may be reviewed by many users. Neither entity has total participation since there is no requirement for users to review books. *reviews* has attributes *rating* and *reviewText*. *rating* is a rating out of 5 for the book, and *reviewText* is a written review by the user about the book. *reviewText* is optional and may remain null.

creates/places is a many-to-one relationship from *order* to *user*, since a user can place many orders but each order only has one user that created it. Both *order* and *user* have total participation, since every order must be placed by a user, and every user by default always has an incomplete order representing their basket.

contains is a many-to-many relationship from *orderItem* to *book*, as there can be any number of orders containing a certain book, and a book can be in many orders. *orderItem* has total participation because there cannot be an order item without a corresponding book.

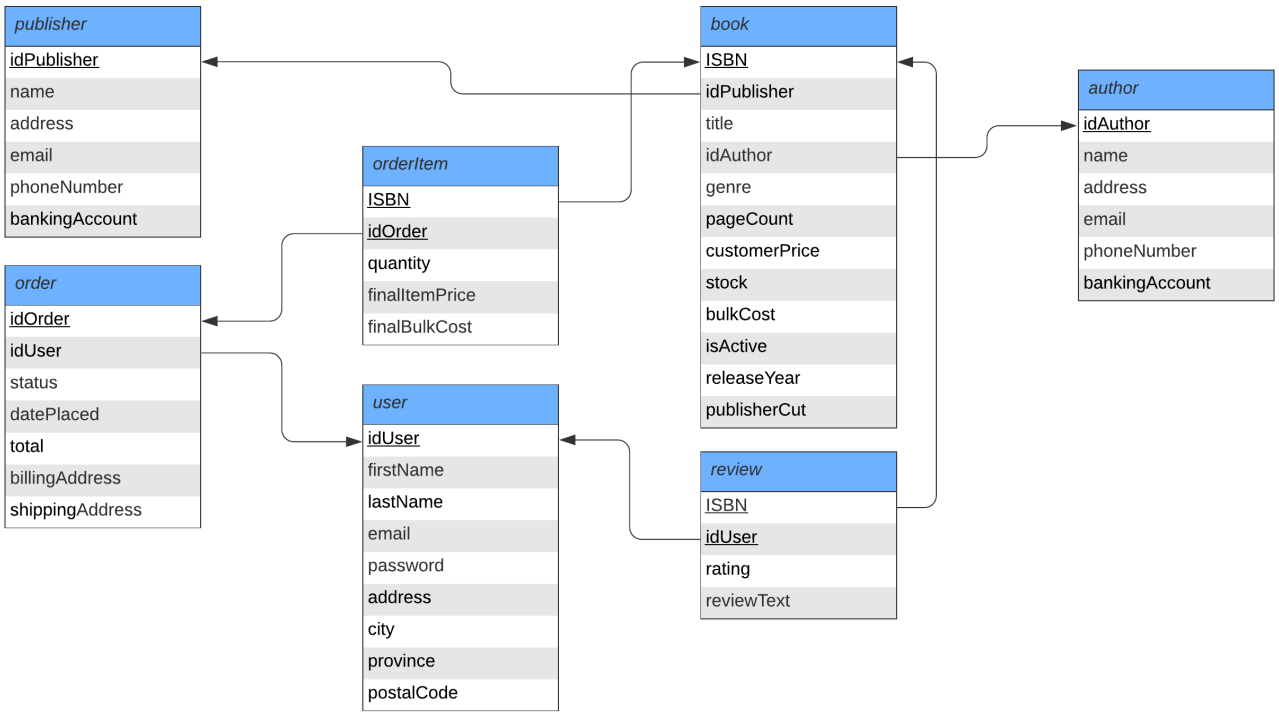
has is a many-to-one relationship from *orderItem* to *order*, since an order may have many items but an item only has one order. *orderItem* has total participation because every order item must have an associated order, but an order can be empty.

This design is intended to be as initially normalized as possible, accounting for all required features of the system as well as being modular enough to allow for simple addition of bonus features. There are going to be null values in *order* and *orderItem*, but this will be accounted for during queries involving those entities.

2 Reduction to Relation Schemas

Reducing to relation schemas yields the following:

publisher(idPublisher, name, address, email, phoneNumber, bankingAccount)
book(ISBN, idPublisher, title, idAuthor, genre, pageCount, customerPrice, stock, bulkCost, isActive, releaseYear, publisherCut)
author(idAuthor, name, address, email, phoneNumber, bankingAccount)
order(idOrder, idUser, status, datePlaced, total, billingAddress, shippingAddress)
orderItem(ISBN, idOrder, quantity, finalItemPrice, finalBulkCost)
user(idUser, firstName, lastName, email, password, address, city, province, postalCode)
review(ISBN, idUser, rating, reviewText)



3 Normalization of Relation Schemas

Let $R_1 = \{A, B, C, D, E, F\}$ represent *publisher(idPublisher, name, address, email, phoneNumber, bankingAccount)*
$F_1 = \{A \rightarrow BCDEF\}$
Computing A^+ :
$AA \rightarrow ABCDEF$ using Augmentation
$A \rightarrow ABCDEF$ using Decomposition, therefore *idPublisher* is a superkey and thus *publisher* is in 3NF.

Let $R_2 = \{A, B, C, D, E, F, G, H, I, J, K, L\}$ represent *book(ISBN, idPublisher, title, idAuthor, genre, pageCount, customerPrice, stock, bulkCost, isActive, releaseYear, publisherCut)*
$F_2 = \{A \rightarrow BCDEFGHIJKL\}$
Computing A^+ :
$AA \rightarrow ABCDEFGHIJKL$ using Augmentation
$A \rightarrow ABCDEFGHIJKL$ using Decomposition, therefore *ISBN* is a superkey and thus *book* is in 3NF.

Let $R_3 = \{A, B, C, D, E, F\}$ represent *author(idAuthor, name, address, email, phoneNumber, bankingAccount)*
$F_3 = \{A \rightarrow BCDEF\}$
Computing A^+ :
$AA \rightarrow ABCDEF$ using Augmentation
$A \rightarrow ABCDEF$ using Decomposition, therefore *idAuthor* is a superkey and thus *author* is in 3NF.

Let $R_4 = \{A, B, C, D, E, F, G\}$ represent *order(idOrder, idUser, status, datePlaced, total, billingAddress, shippingAddress)*
$F_4 = \{A \rightarrow BCDEFG\}$
Computing A^+ :
$AA \rightarrow ABCDEFG$ using Augmentation
$A \rightarrow ABCDEFG$ using Decomposition, therefore *idOrder* is a superkey and thus *order* is in 3NF.

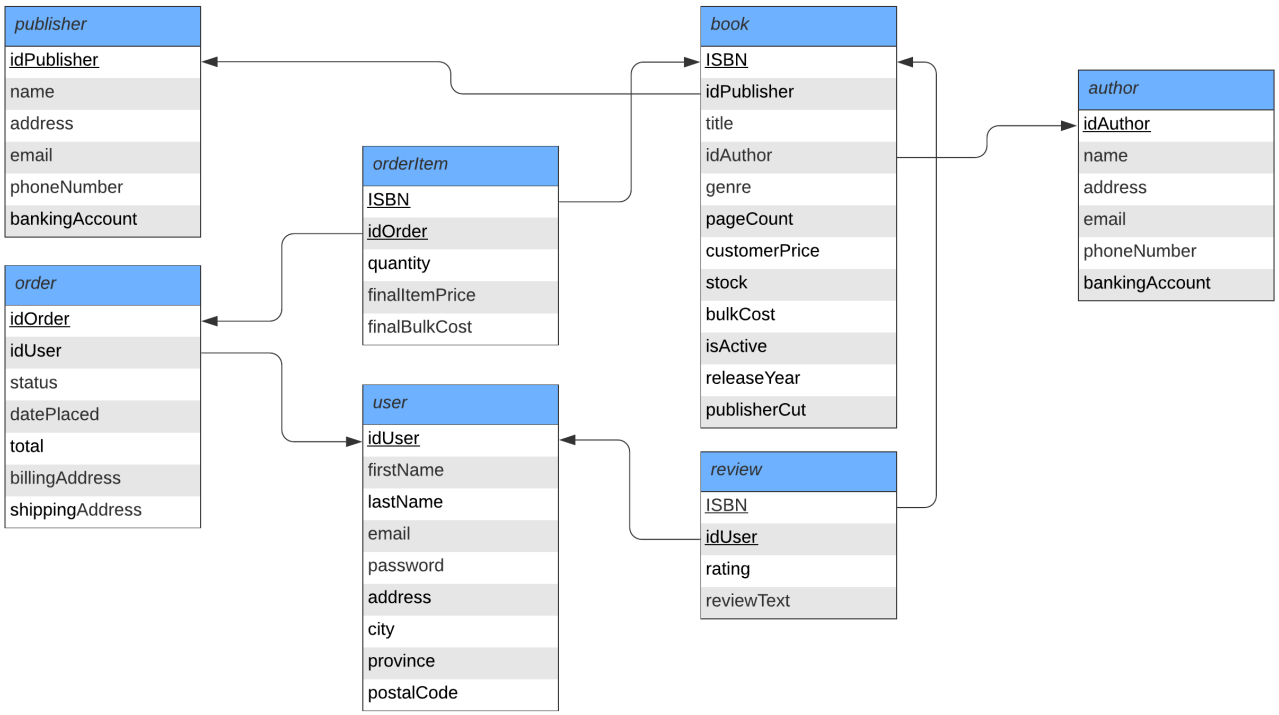
Let $R_5 = \{A, B, C, D, E, F, G, H, I\}$ represent *user(idUser, firstName, lastName, email, password, address, city, province, postalCode)*
$F_5 = \{A \rightarrow BCDEFGHI\}$
Computing A^+ :
$AA \rightarrow ABCDEFGHI$ using Augmentation
$A \rightarrow ABCDEFGHI$ using Decomposition, therefore *idUser* is a superkey and thus *user* is in 3NF.

Let $R_6 = \{A, B, C, D\}$ represent *review(ISBN, idUser, rating, reviewText)*
$F_6 = \{AB \rightarrow CD\}$
Computing AB^+ :
$AABB \rightarrow ABCD$ using Augmentation
$AB \rightarrow ABCD$ using Decomposition, therefore $\{ISBN, idUser\}$ is a superkey and thus *review* is in 3NF.

Let $R_7 = \{A, B, C, D, E\}$ represent *orderItem(ISBN, idOrder, quantity, finalItemPrice, finalBulkCost)*
$F_7 = \{AB \rightarrow CDE\}$
Computing AB^+ :
$AABB \rightarrow ABCDE$ using Augmentation
$AB \rightarrow ABCDE$ using Decomposition, therefore $\{ISBN, idOrder\}$ is a superkey and thus *orderItem* is in 3NF.

Since every relation in the database is in 3NF, the database as a whole is normalized and no changes are required.

4 Database Schema Design

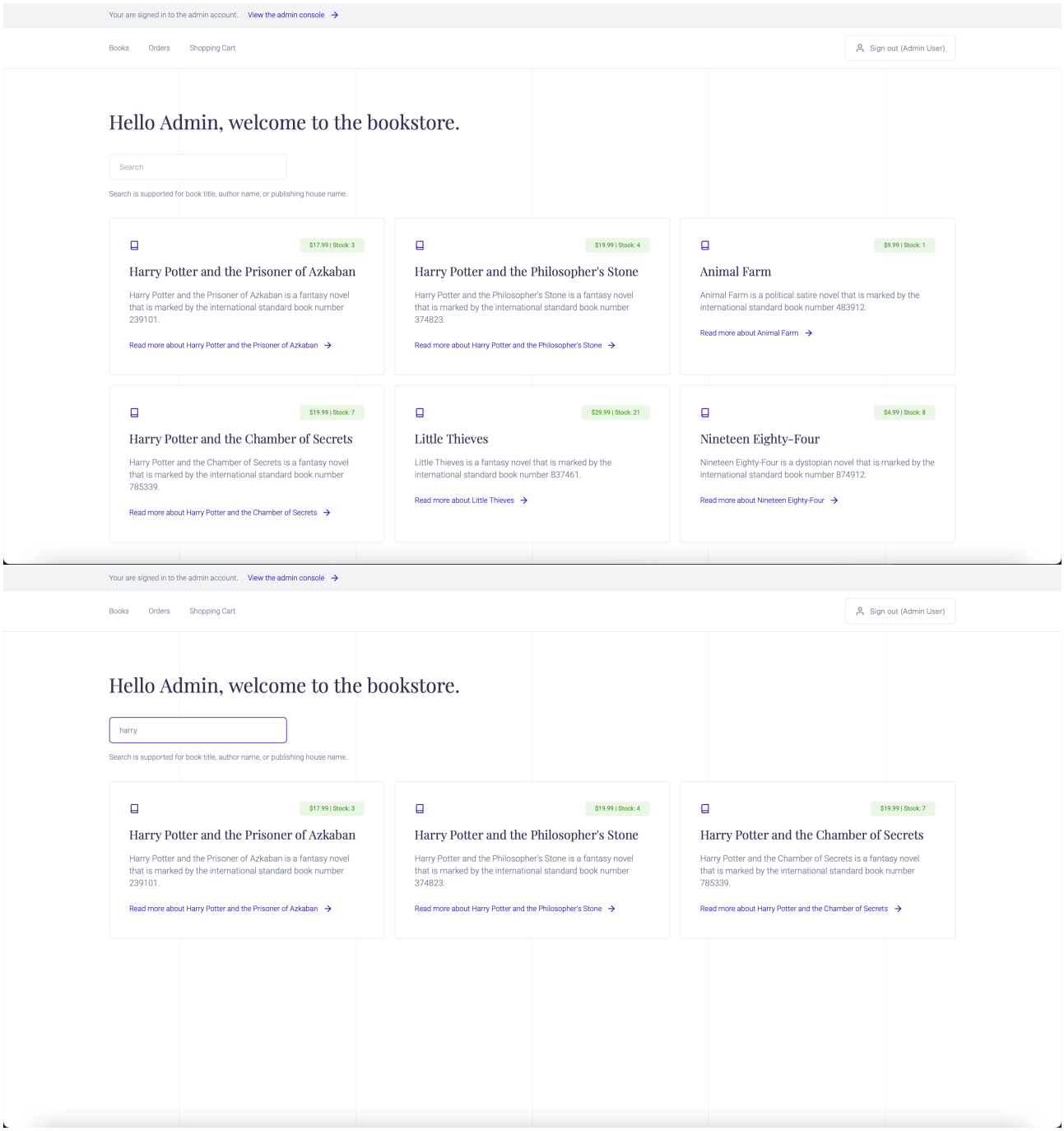


5 Implementation

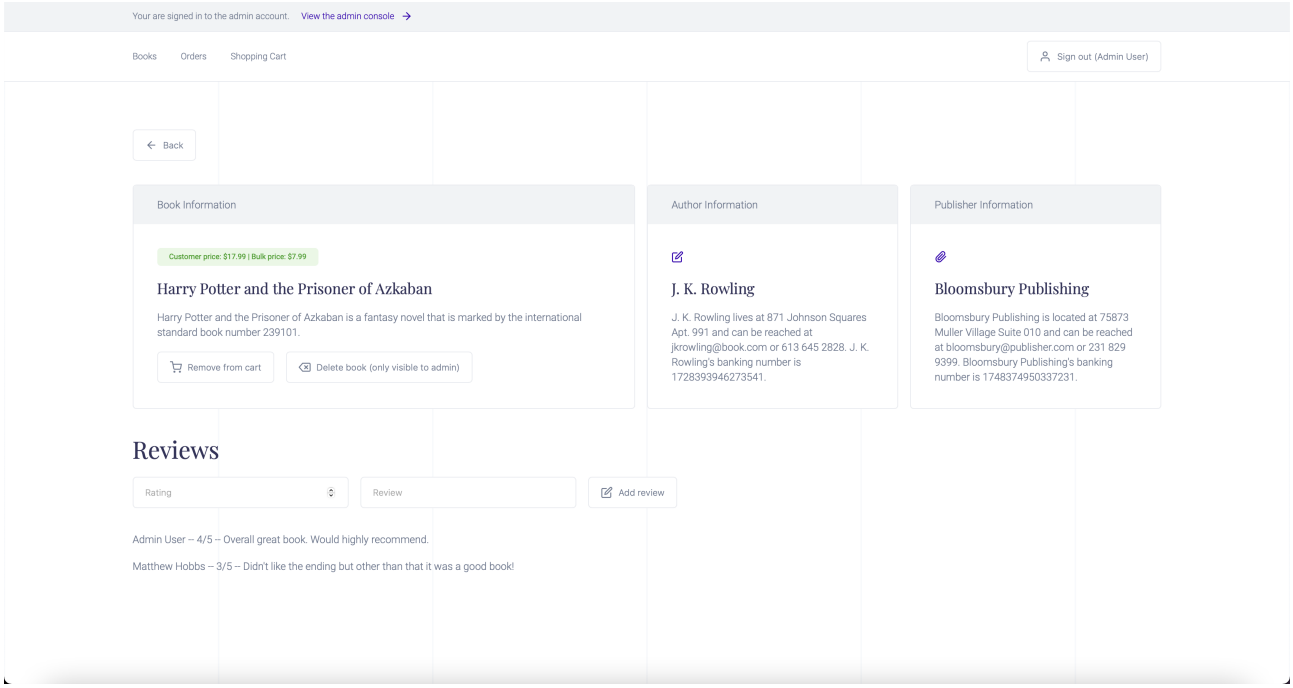
We developed a web application using Svelte as the frontend framework. The SQL commands were abstracted out using a REST API that has been implemented in Node.js and Express.

The application includes 3 unique user classes: guest, user, and admin. These user classes determine which pages and options are visible in the app. For example, in the book view a guest has no options, a user can add to cart, and an admin can add to cart or delete the book from the database. The admin account can be accessed by signing in with the email admin@admin.com and the password admin. A banner will appear to signify you are signed into the admin account.

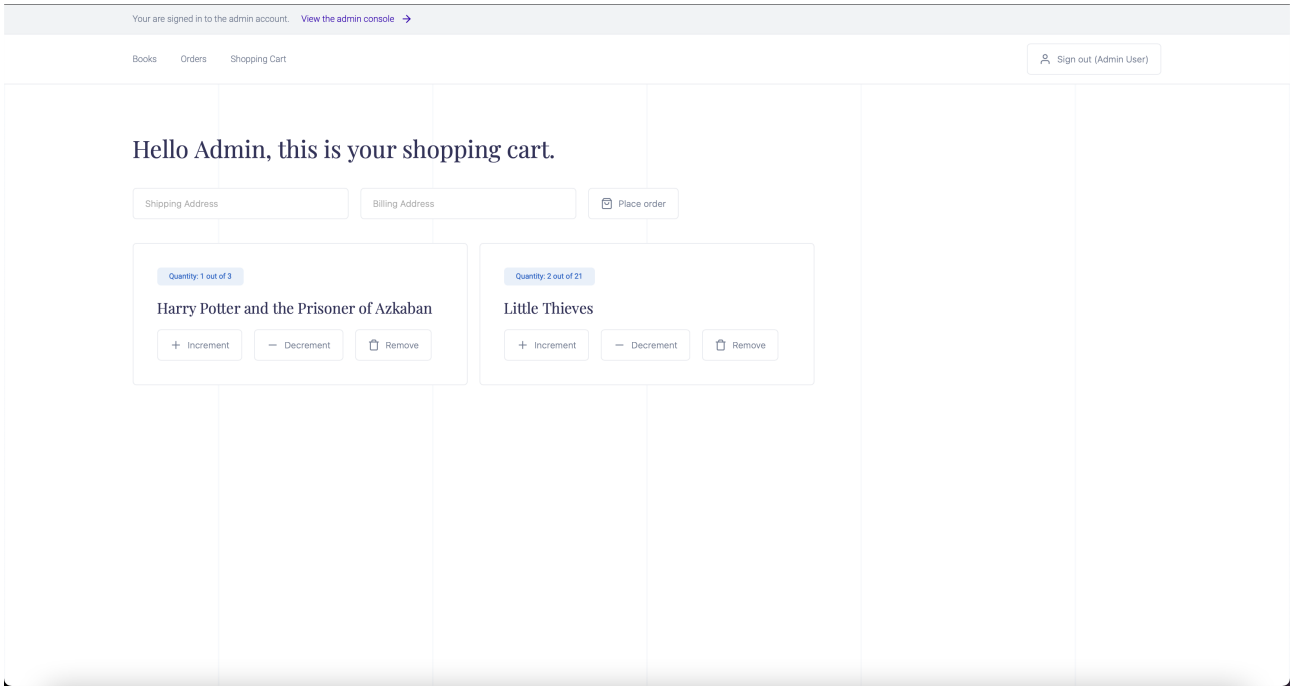
The first scenario is browsing books. This can be accessed at the route / and is visible to guests, users, and admins. This pulls all books from the database with the SQL command SELECT * FROM Book. There is also a search functionality implemented into this page. Both functionalities can be seen below.



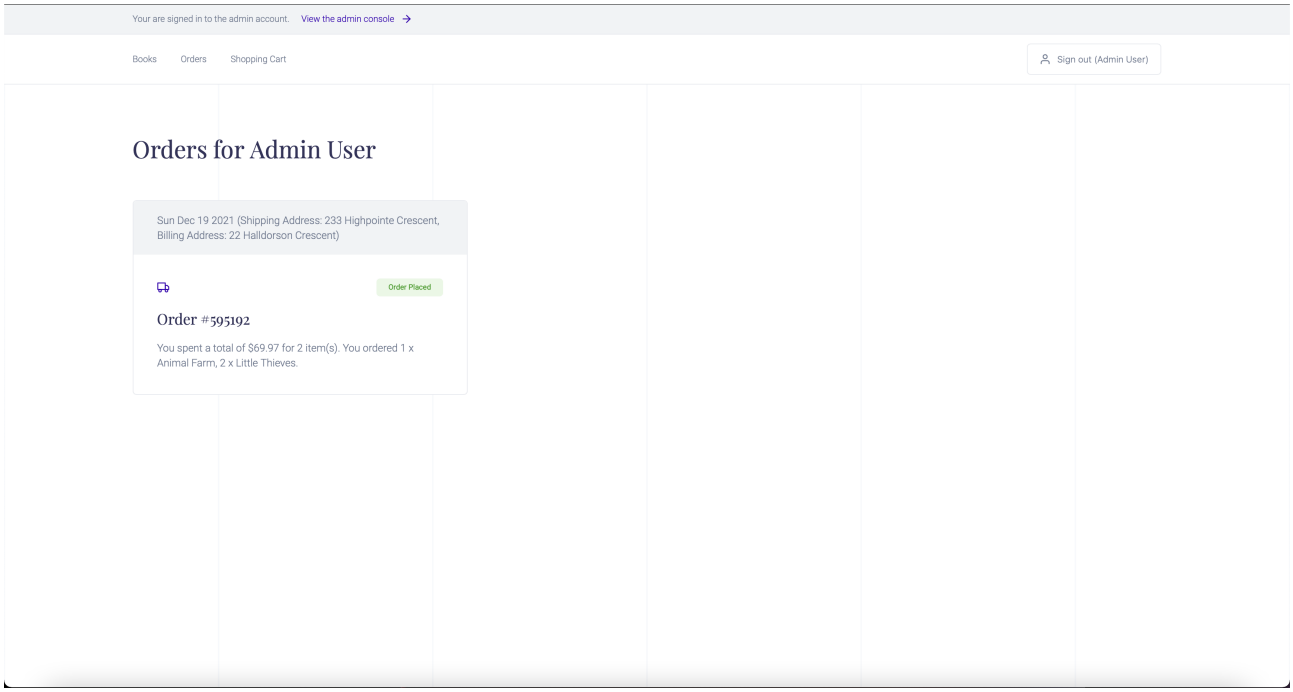
Users can also view more detailed information about a book by selecting the link. The content of this page changes depending on the user class. For example, guests will have access to none of the buttons, users will have access to the add to cart button, and admins will have access to add to cart and delete book from database. This page features the book's information, the author and publisher's details, as well as a list of the reviews for the book and an option for leaving one, available only to registered users.



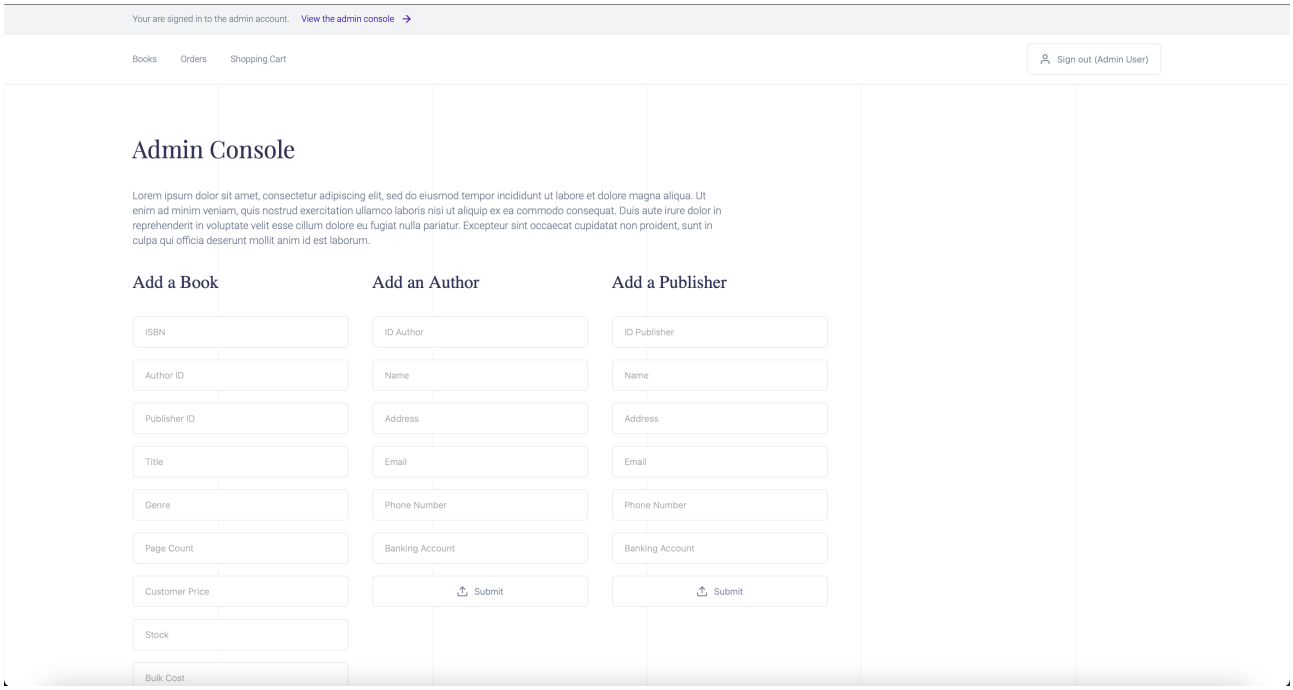
Every user other than a guest can access their shopping cart. This page will show all of the books the user has added to their cart (if any), and provide them with the options of incrementing/decrementing the quantity of a certain book, or removing it from the cart entirely. This page also allows the user to enter shipping/billing information and place the order. Upon placing the order, the *order* instance is finalized with the relevant information and the status is updated accordingly, then a new instance is created for the empty cart which will be displayed accordingly.



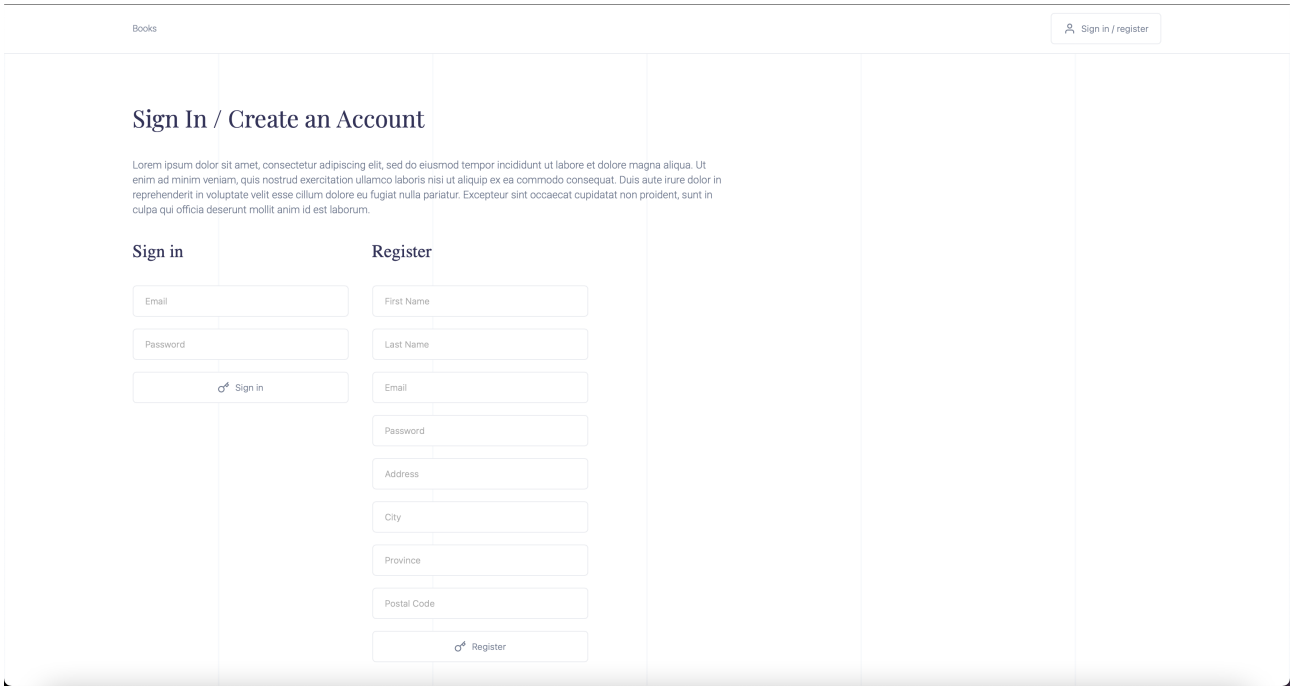
Every user other than a guest can access their orders page. This page will show all of the orders the user has placed (if any), with price information relevant to the time it was placed, a summary of all items, shipping/billing information, and the order number for tracking.



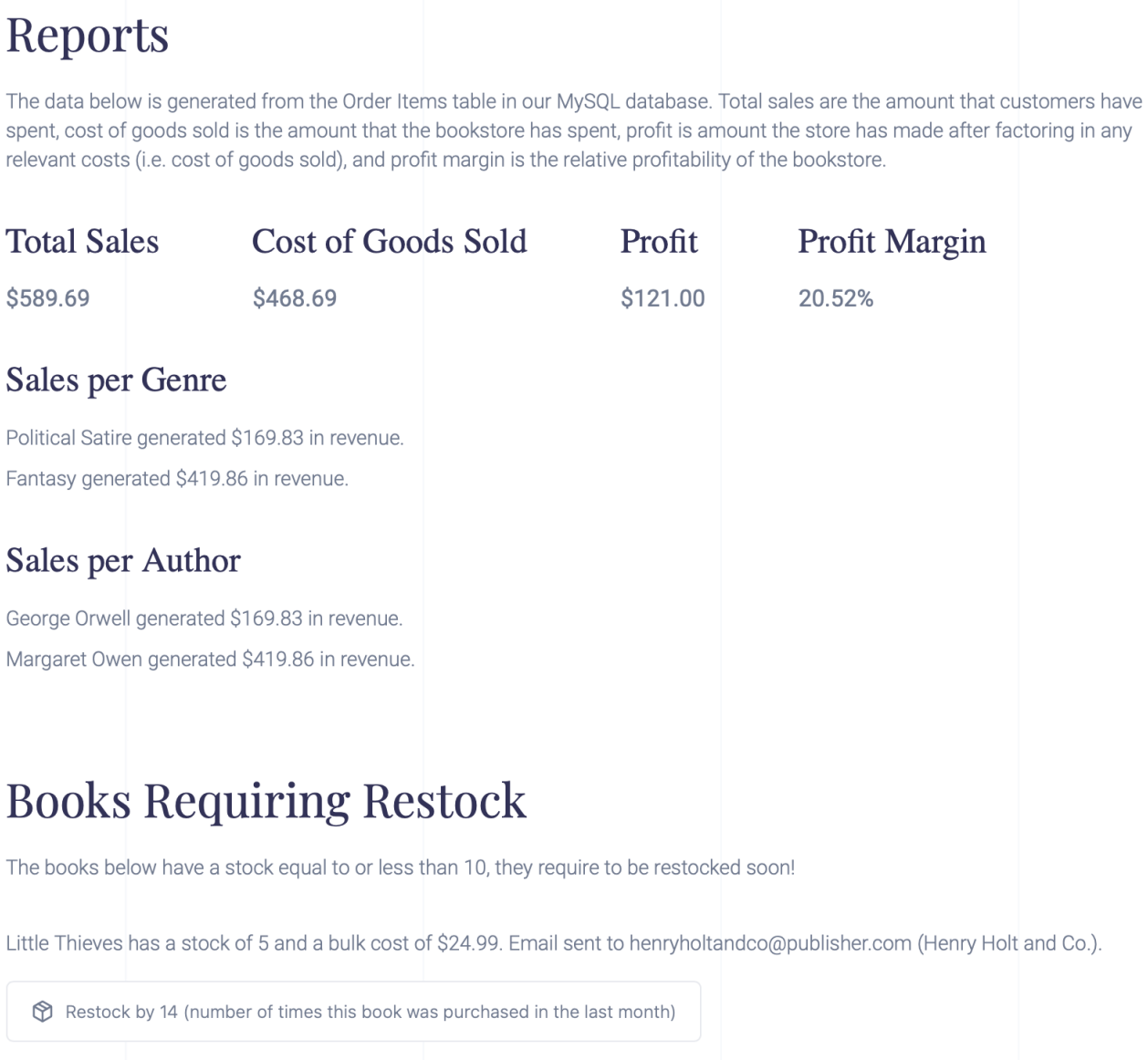
The admin console is only visible to an admin-class user. This page allows the user to add a new book, author, or publisher, along with all of their corresponding details. Upon submitting each form, the data is inserted into a new tuple in the corresponding relation.



Guest users can access the authorization page, where they can either sign in to an existing account, or create a new one.



The report page displays all relevant statistics for the store.



6 Bonus Features

Users have the ability to leave reviews for books in the store, which consist of a rating out of five and an optional written review. These reviews are displayed for all users on the page corresponding to the book.

Upon placing an order, a snapshot of the current bulk cost and customer price of each book is stored within the corresponding tuple in *orderItem*, meaning that even if those values for that book change in the future, the store's profits/expenditures and any other statistics that require those values remain accurate.

7 GitHub Repository

https://github.com/MatthewDHobbs/COMP3005-Final-Project

8 Appendix I

We are available any time on December 20th after 1pm