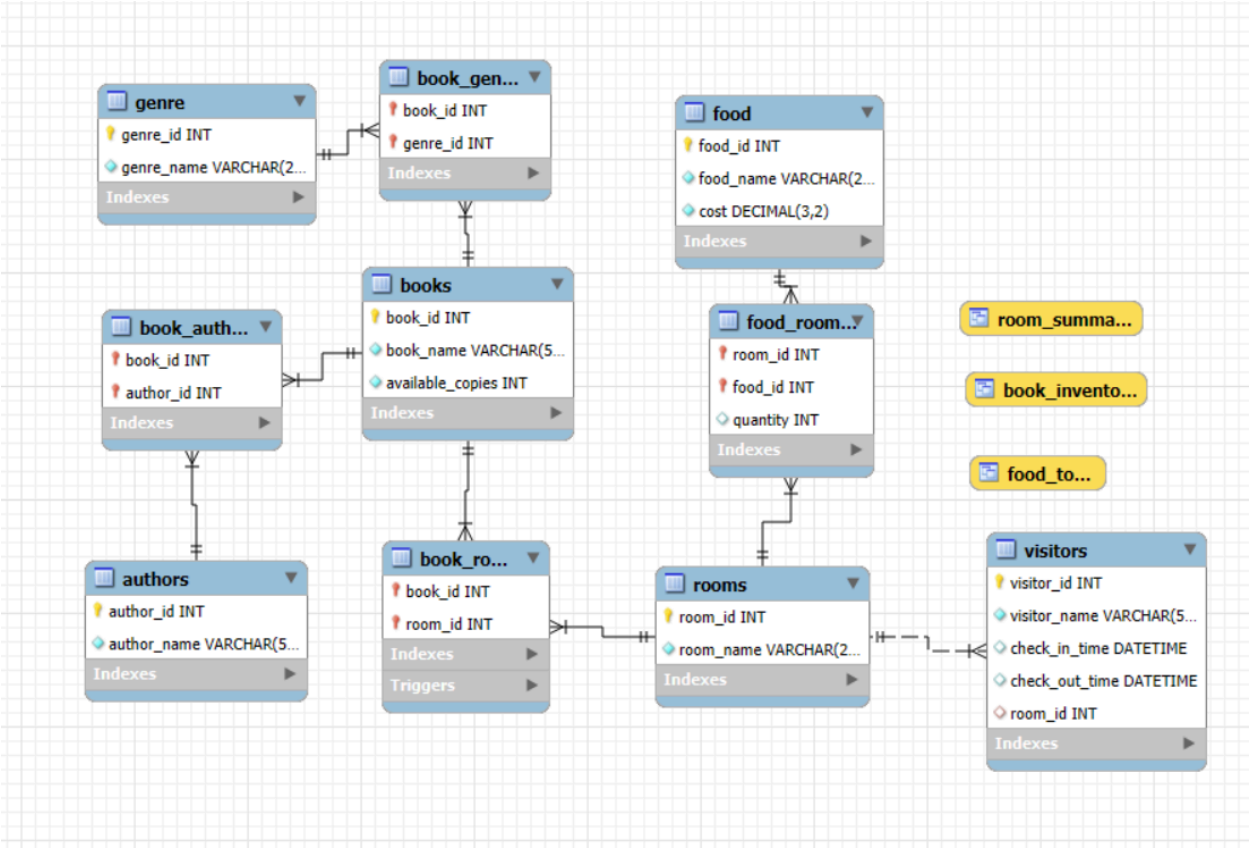
Overview Of The System:

The system that I decided to work on was a Manga Cafe POS System. A brief description of a Manga Cafe is a store in which you would typically rent an isolated room for a period of time, and borrow the manga books they have available. Within my database I also decided to include a food ordering service they customers can charge to the room. One of the screens is the summary page. Where the employee will be able to view the inventory summary of each room, and of the current stock of books. There is no stock functionality for the food tables though. My second screen is the POS screen of the system. In this screen you have three functions. To check out a book to a room, order food to a room, or to check out of a room altogether, having the customer pay their final bill. You can only check out one book at a time, food is restricted to one food type per purchase.

EER Diagram:



Normalization:

1NF: While making my outline for the tables I ensured that the first form of normalization was taken care of. Each table has atomic fields.

2NF: There are no functional dependencies within the Database as all of the function dependencies have been removed with the use of join tables. The book_room, food_room, book_genre all use composite primary keys, making it so that you cannot find the relationship of a record without knowing the primary keys.

3NF: Non-key fields depend only on the primary keys, such as the available copies field only relies on the primary key, and not something such as the name of the book.

Stored Procedures, Triggers and Views:

Stored Procedures

- check_out: This procedure checks out a room having the room_id set to null of the visitor who has left, and removing all food and books from that room.
- food_room_total: this procedure calls the 'food_total' view for a given room_id getting the total amount of money that room has spent on food. The reason for this is due to the 'quantity' field within the food_room table. This field is a non aggregate field meaning there is no proper way to use a group by clause on it. So instead this procedure just uses a where clause, using the id given as input. It will then select only the totals from that given id adding them up getting the total spent on food.
- order_book/order_food: Both functions act the same. Given a room id and the id of the book or food it will be added to the join tables associated with each id.
- room_total_cost: This function calls the previously mentioned food_room_total and adds this number with the static $10 charge for a room, alongside a $1 fee for any book borrowed from the store.

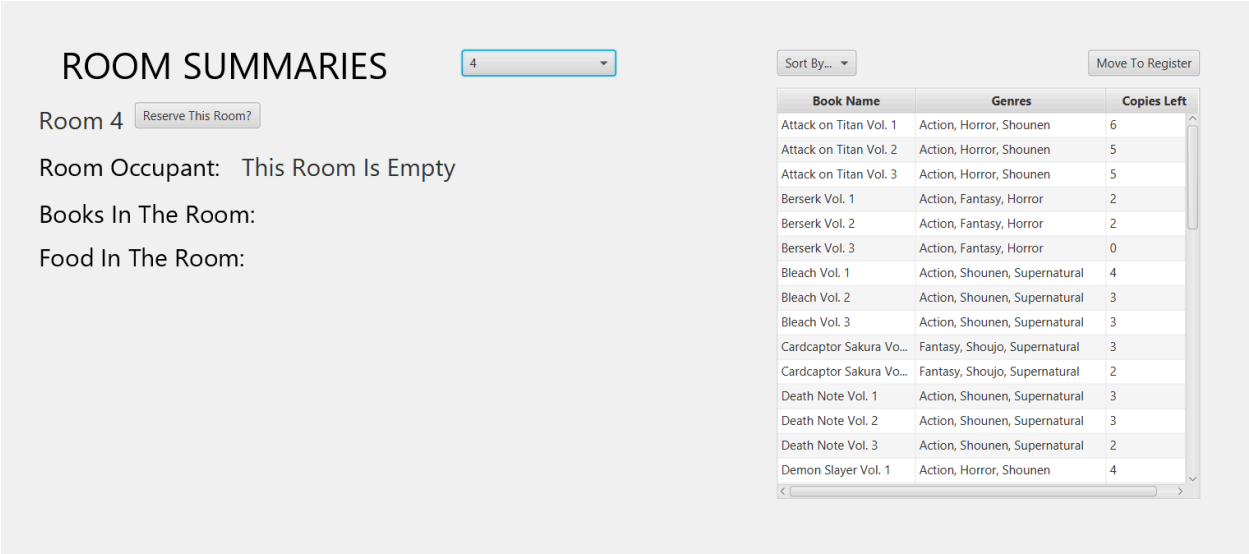
Triggers

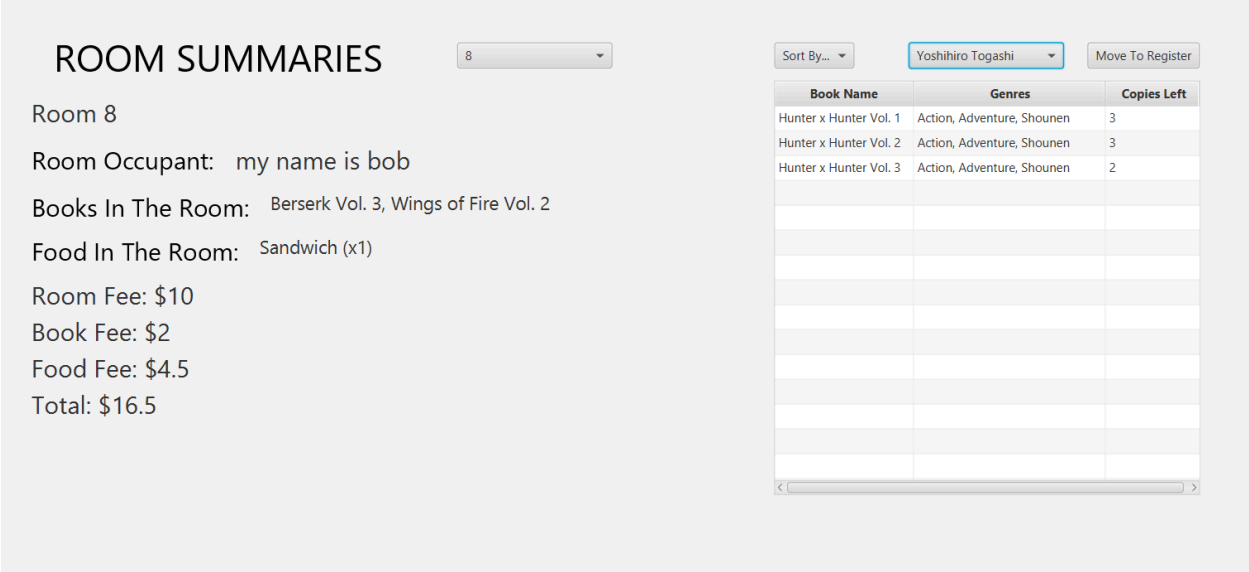
- auto_dec_book_copies/auto_inc_book_copies: Both of these triggers act as one anothers counterparts. When checking out a book it will decrease the available quantity by one. While the other one will increase the book quantity by one when a room is checked out of.
- prevent_zero_available_copies: This trigger will block the check out of a book which has 0 copies. So if there are 0 copies left, it will not allow you to insert a book into the book_room table.

Views

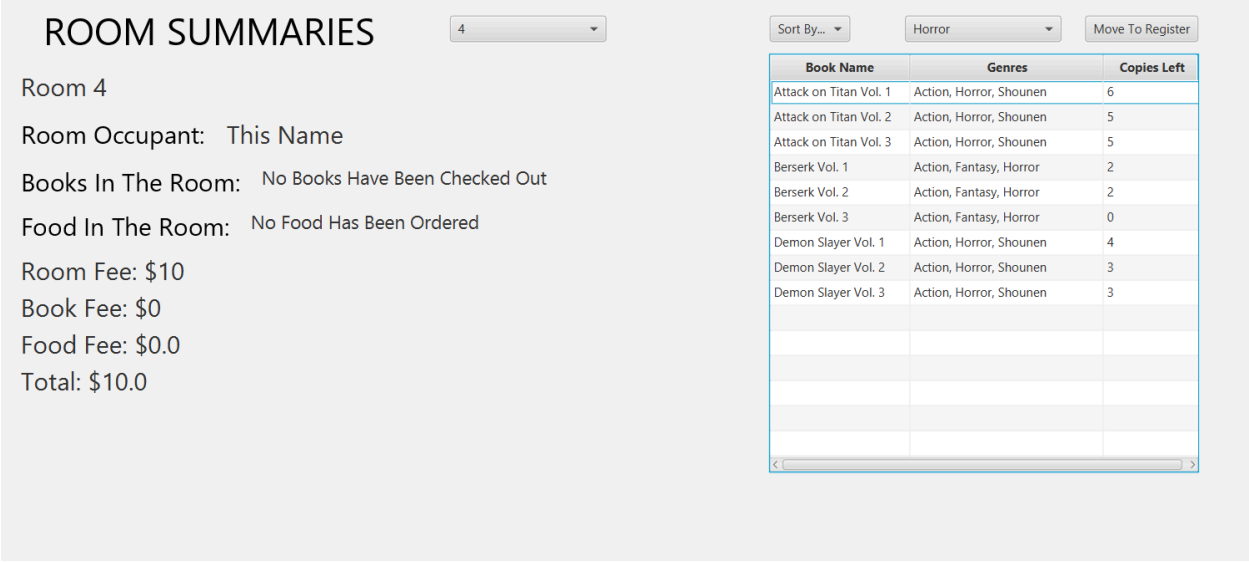
- book_inventory: This view will just show all of the books and their current availability. This view also joins the genre table showing a more accurate description of the book.
    (If I were to redo this view I would have also added the author into it to make filtering more easy as I did within my FE)
- food_total: This view just shows each food in each room with their respective quantity. The one downside of this view is due to the quantity field not being aggregated you cannot find out the total cost with a Group By Clause.
- room_summary: This view will return the summary of the room (only if it's in use). Showing who is in it, what books they currently have, and what food they have ordered

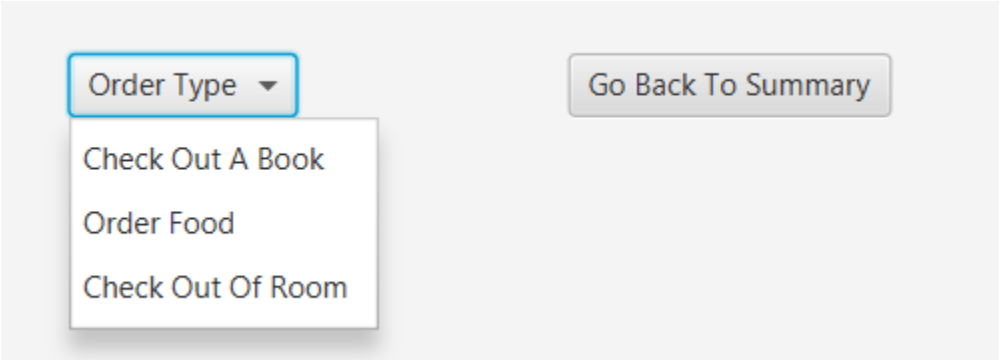
Front End SS's:

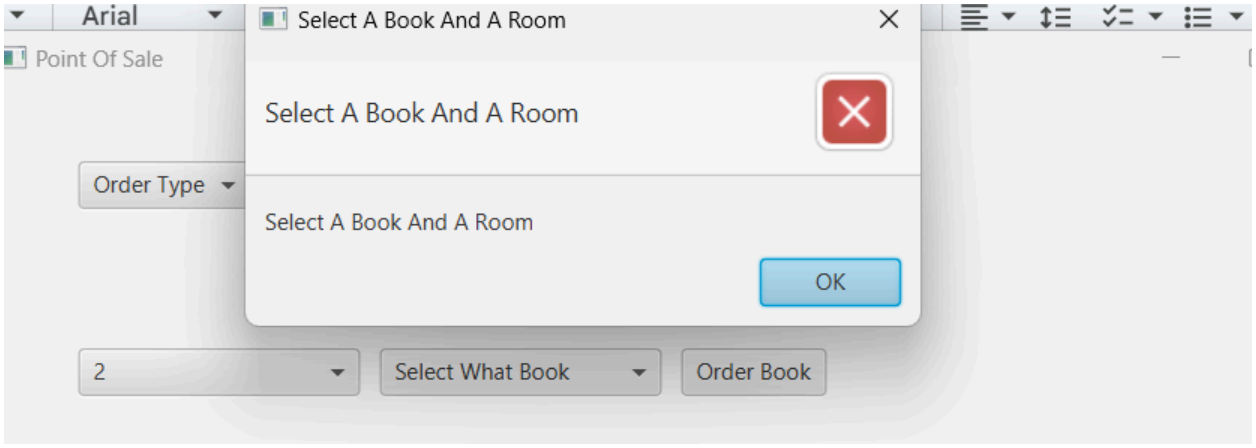


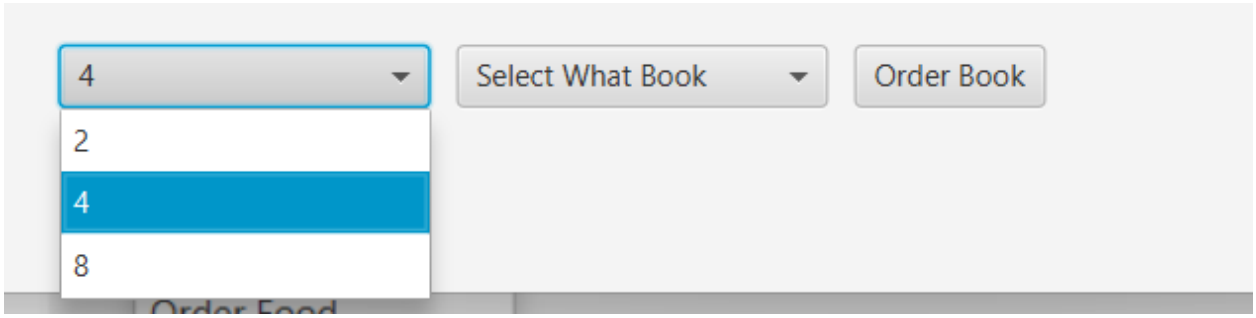



Room Number: 4
Enter Occupant Name
This Name
Reserve The Room

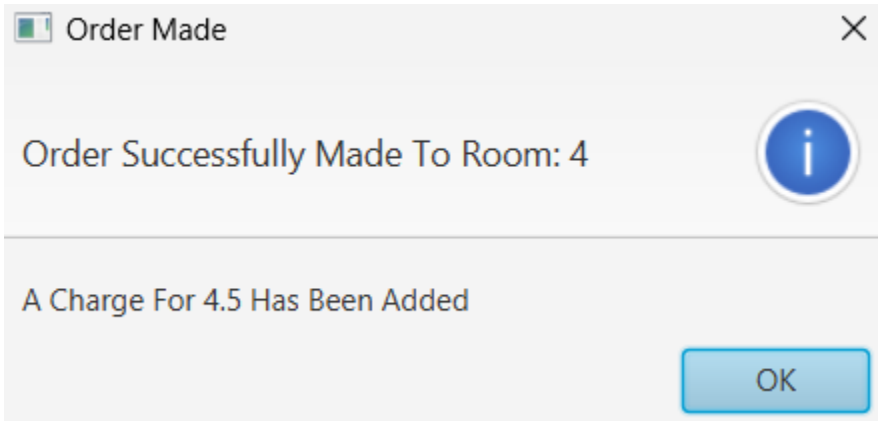

ROOM SUMMARIES
4
Room 4
Room Occupant: This Name
Books In The Room: No Books Have Been Checked Out
Food In The Room: No Food Has Been Ordered
Room Fee: $10
Book Fee: $0
Food Fee: $0.0
Total: $10.0
Sort By...
Horror
Move To Register
Book Name
Genres
Copies Left
Attack on Titan Vol. 1
Action, Horror, Shounen
6
Attack on Titan Vol. 2
Action, Horror, Shounen
5
Attack on Titan Vol. 3
Action, Horror, Shounen
5
Berserk Vol. 1
Action, Fantasy, Horror
2
Berserk Vol. 2
Action, Fantasy, Horror
2
Berserk Vol. 3
Action, Fantasy, Horror
0
Demon Slayer Vol. 1
Action, Horror, Shounen
4
Demon Slayer Vol. 2
Action, Horror, Shounen
3
Demon Slayer Vol. 3
Action, Horror, Shounen
3


Order Type
Check Out A Book
Order Food
Check Out Of Room
Go Back To Summary


Arial
Point Of Sale
Select A Book And A Room
Select A Book And A Room
Select A Book And A Room
OK
Order Type
2
Select What Book
Order Book


4
Select What Book
Order Book
2
4
8


Order Type
Go Back To Summary
Sandwich
4
+1
-1
Sandwich Quantity: 1x Subtotal: 4.5
Order


Order Made
Order Successfully Made To Room: 4
A Charge For 4.5 Has Been Added
OK

ROOM SUMMARIES

4

Room 4

Room Occupant: This Name

Books In The Room: Berserk Vol. 2

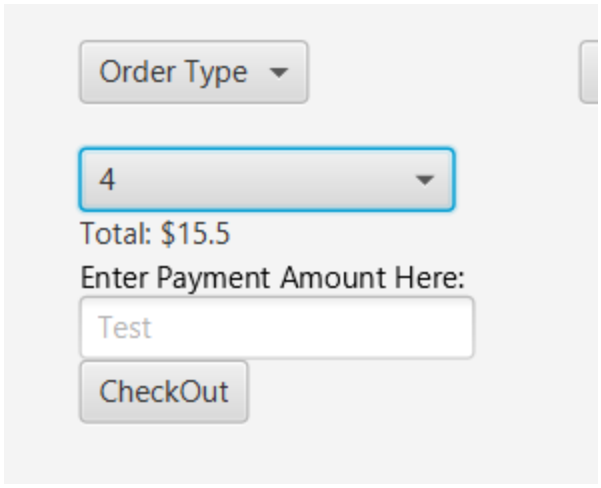
Food In The Room: Sandwich (x1)

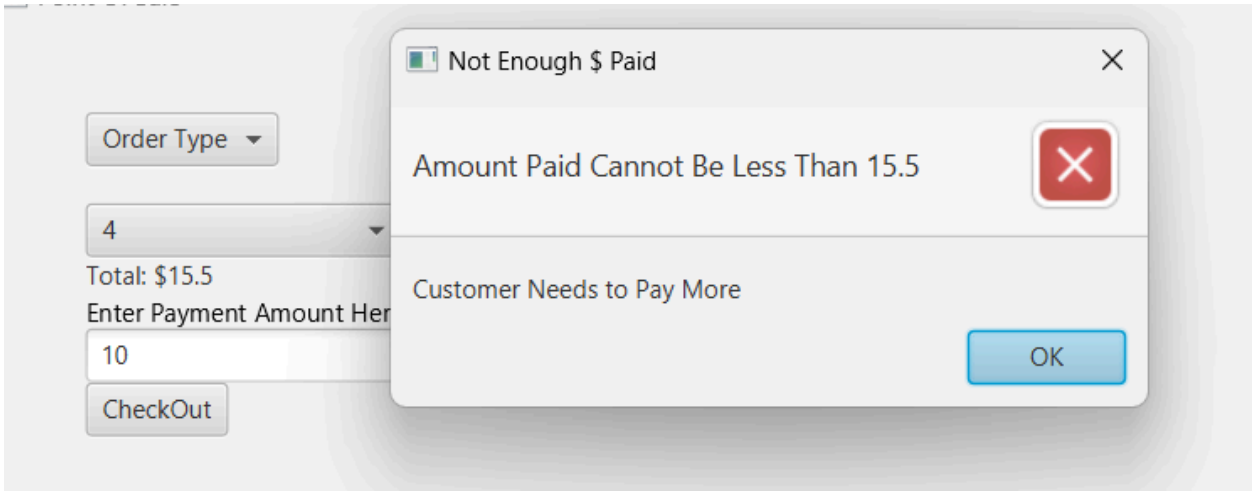
Room Fee: $10

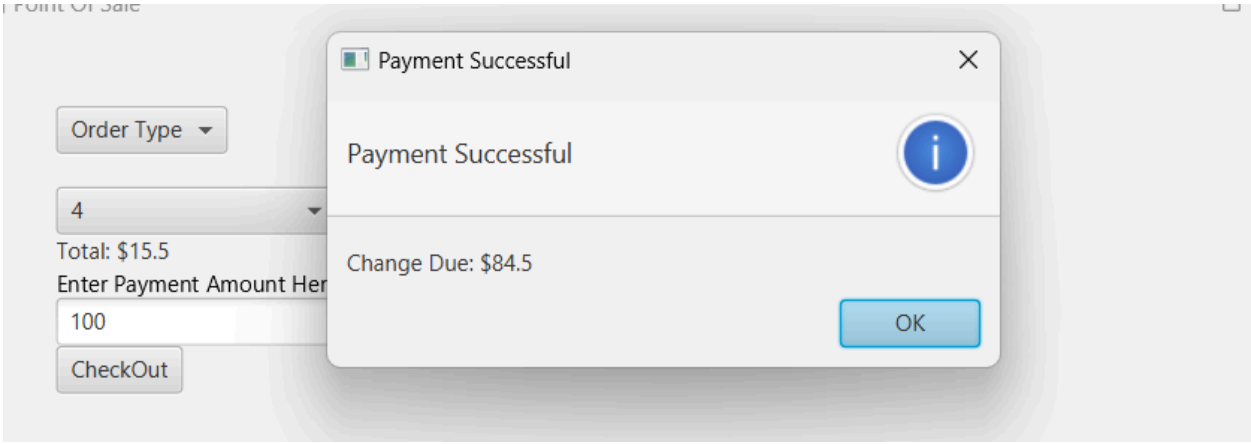
Book Fee: $1

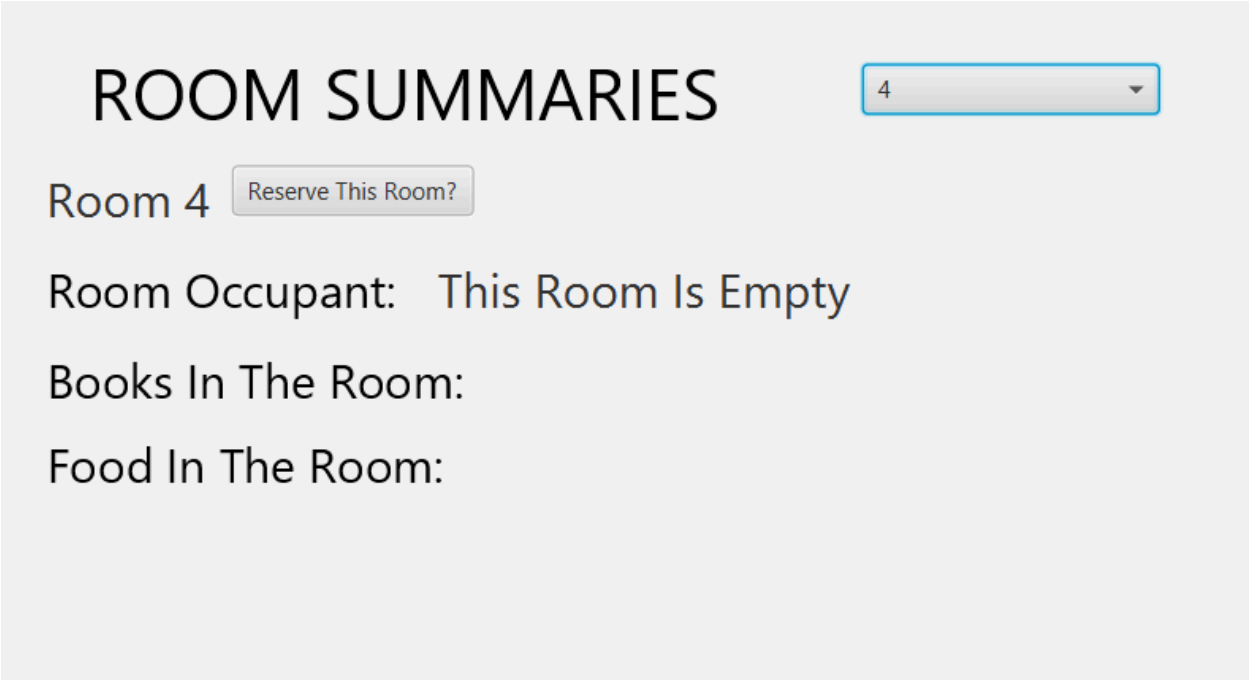
Food Fee: $4.5

Total: $15.5


Order Type
4
Total: $15.5
Enter Payment Amount Here:
Test
CheckOut


Order Type
4
Total: $15.5
Enter Payment Amount Her
10
CheckOut
Not Enough $ Paid
Amount Paid Cannot Be Less Than 15.5
Customer Needs to Pay More
OK


Order Type
4
Total: $15.5
Enter Payment Amount Her
100
CheckOut
Payment Successful
Payment Successful
Change Due: $84.5
OK


ROOM SUMMARIES
4
Room 4
Reserve This Room?
Room Occupant: This Room Is Empty
Books In The Room:
Food In The Room:

Challenges Faced/Issues/Future:

The first issue I have with my own design is how I set up the GUI interface on the POS screen. Despite knowing the redundancy of having three comboBoxes that stored the rooms that are currently in use (all with nearly identical functionality). I implemented this way to save time for myself. If I had planned the GUI out more in advance then I most likely would have been able to come up with a better design that would accommodate only having one of these comboBoxes.

The last table I implemented was the Visitors table. I am not the happiest about how I designed this table in particular. Currently it just works as a log of all visitors who have ever visited, storing the check in date time and checkout date time. If I were to expand upon this project in the future this is one of the first changes I would make. I change the visitors table into two different tables. One being current visitors, a list of every visitor who is currently in the store, and another that shows all invoices. This second table would be an audit table for when a customer leaves. Storing all of the books they had checked out, check in time, check out time, foods they ordered and the total price they paid. This would be a nice feature as businesses that use this system could then keep track of how well certain items are performing and if they are constantly running low on what manga etc.

Another change I would have made is the ordering system within the Java side of my system. Reflecting upon how I designed it, I don't like how you can only have one book, and one food type being ordered at the same time. If I were able to redo this section I would make it so that instead of having the books and foods being stored within a combobox, that they are stored within a gridview. This gridview would store buttons of every type of food there is to offer, and books as well. Upon the employee clicking one of these buttons that item would be added to a list, and finally upon finishing the order one function would be called that would query both lists into the database.

I don’t recall any large challenges I had faced with this project. If I were to name anything directly it would be time management. While I may have had my database planned and flushed out sooner than most I had completely ignored the front end aspect as well. Much of my frontend code is sloppily written and poorly designed. If I were to ever have a full stack project to do in the future, I will now know to make sure to give both the front and back end as much attention as one another.

LINK TO GITHUB (READ ME HAS THE LINK TO THE VIDEO)

https://github.com/NateC-F/DataBase-Final