

ICE4016 데이터 베이스 설계

<Term Project>

보고서 작성 서약서

1. 나는 타학생의 보고서를 베끼거나 여러 보고서의 내용을 짜집기하지 않겠습니다.

2. 나는 보고서의 주요 내용을 인터넷사이트 등을 통해 얻지 않겠습니다.

3. 나는 보고서의 내용을 조작하지 않겠습니다.

4. 나는 보고서 작성에 참고한 문헌의 출처를 밝히겠습니다.

5. 나는 나의 보고서를 제출 전에 타학생에게 보여주지 않겠습니다.

나는 보고서 작성시 윤리에 어긋난 행동을 하지 않고 정보통신공학인으로서 나의 명예를 지킬 것을 맹세합니다.

2024년 12 월 15 일

학부 정보통신공학

학년 4

성명 이주용

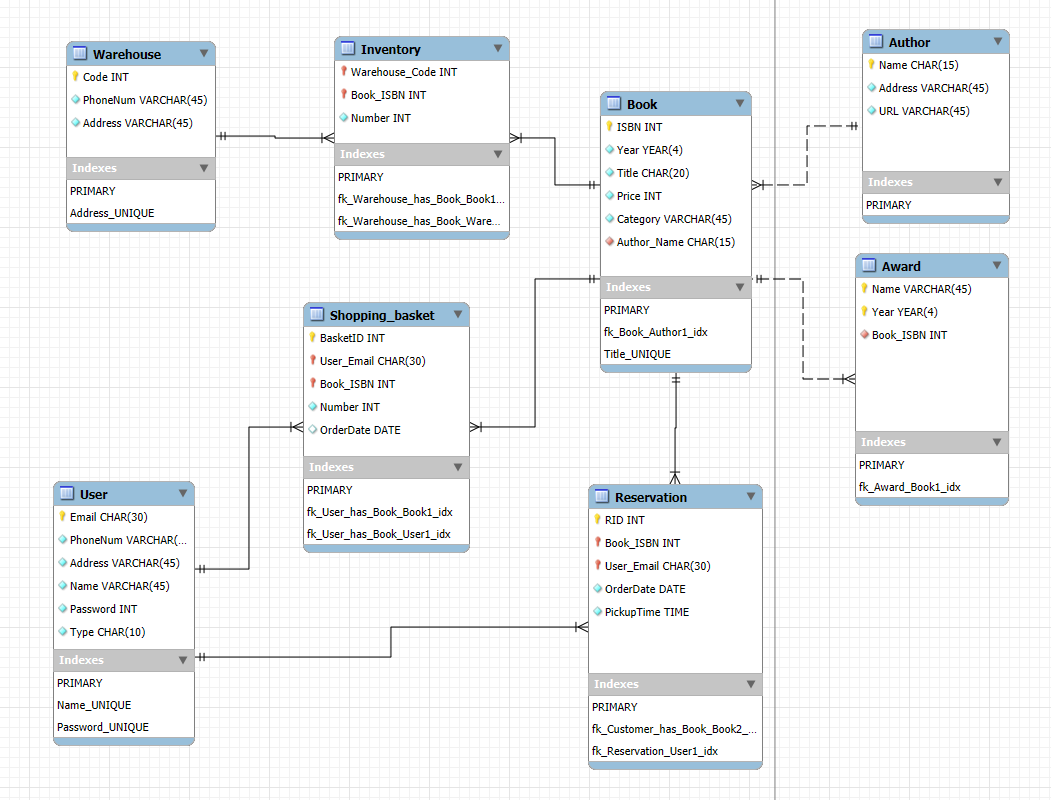
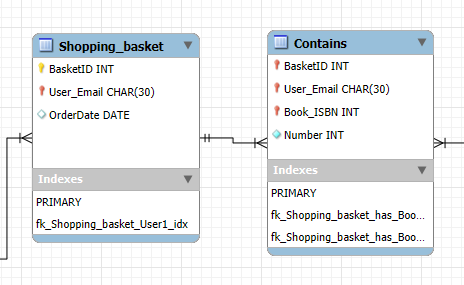
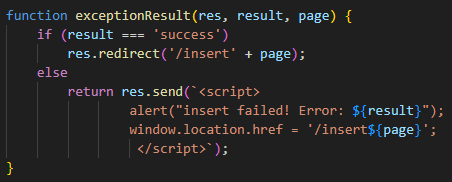
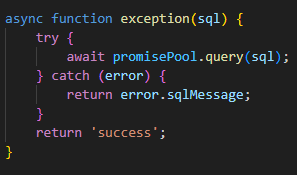
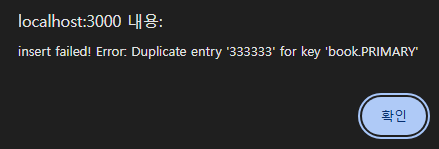
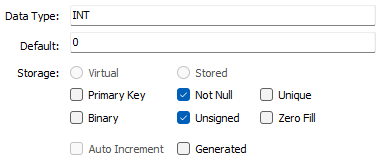
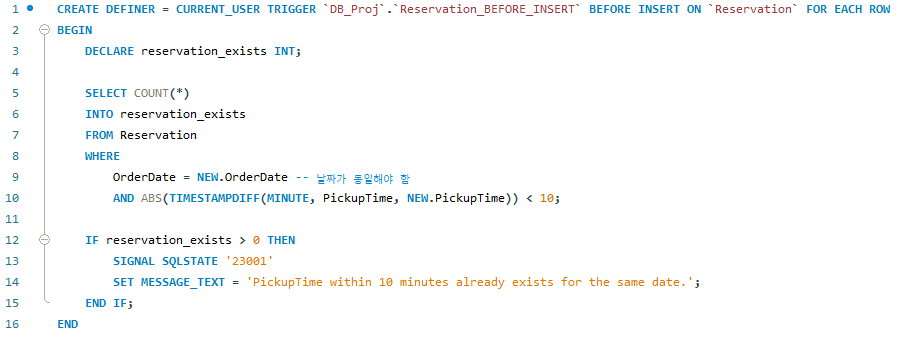
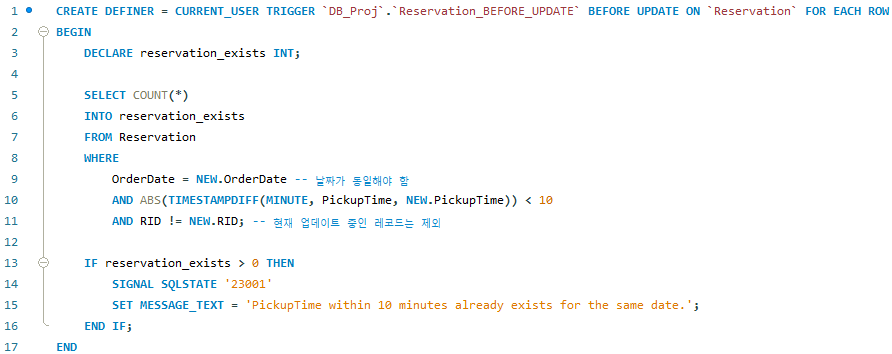
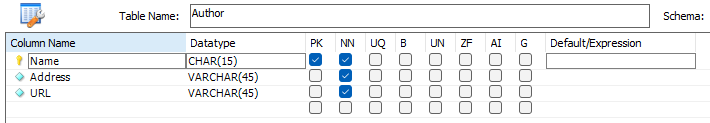
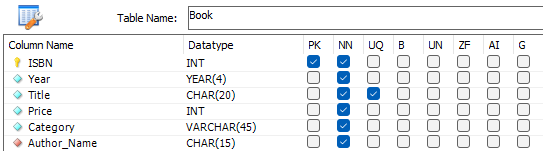
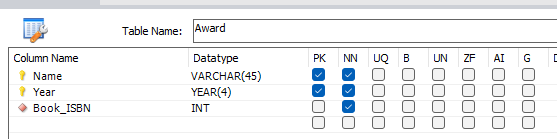
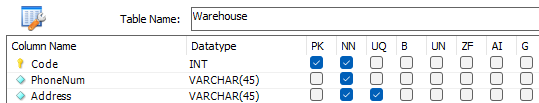
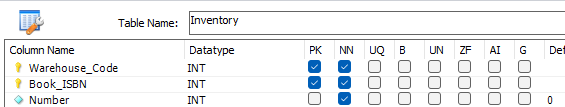
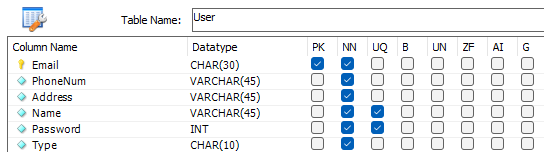
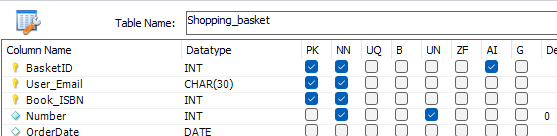
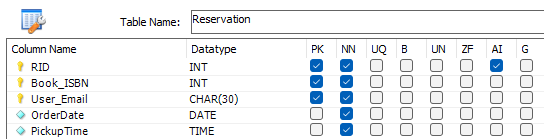
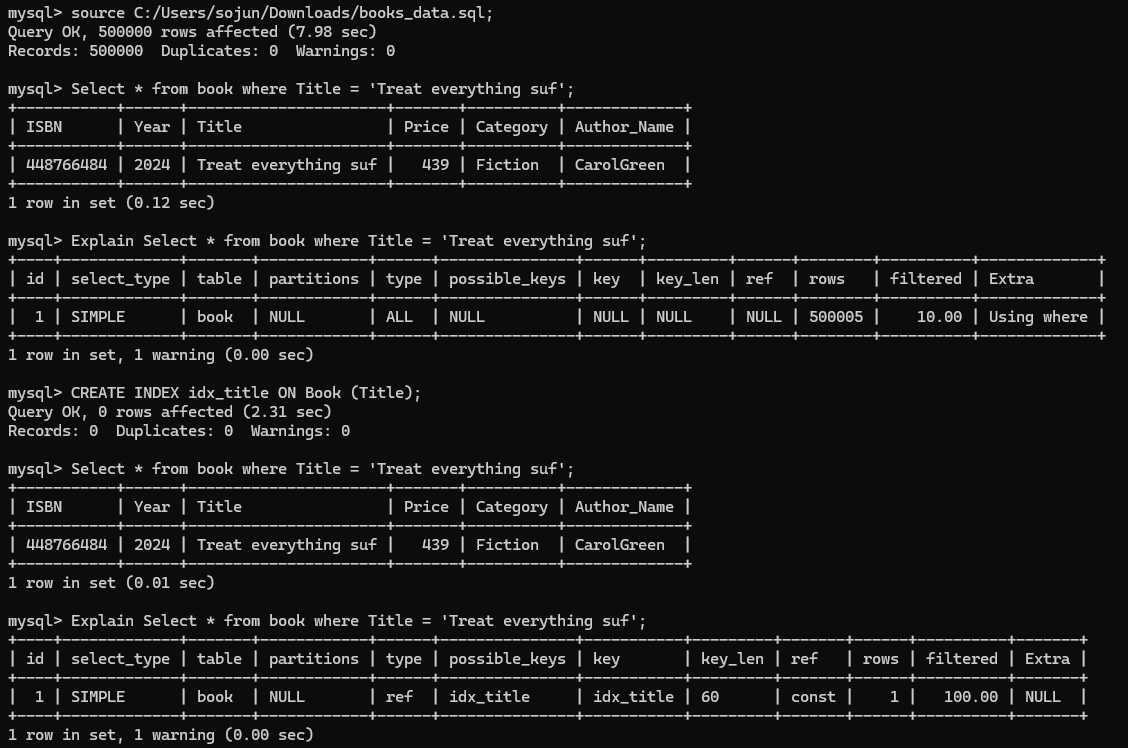
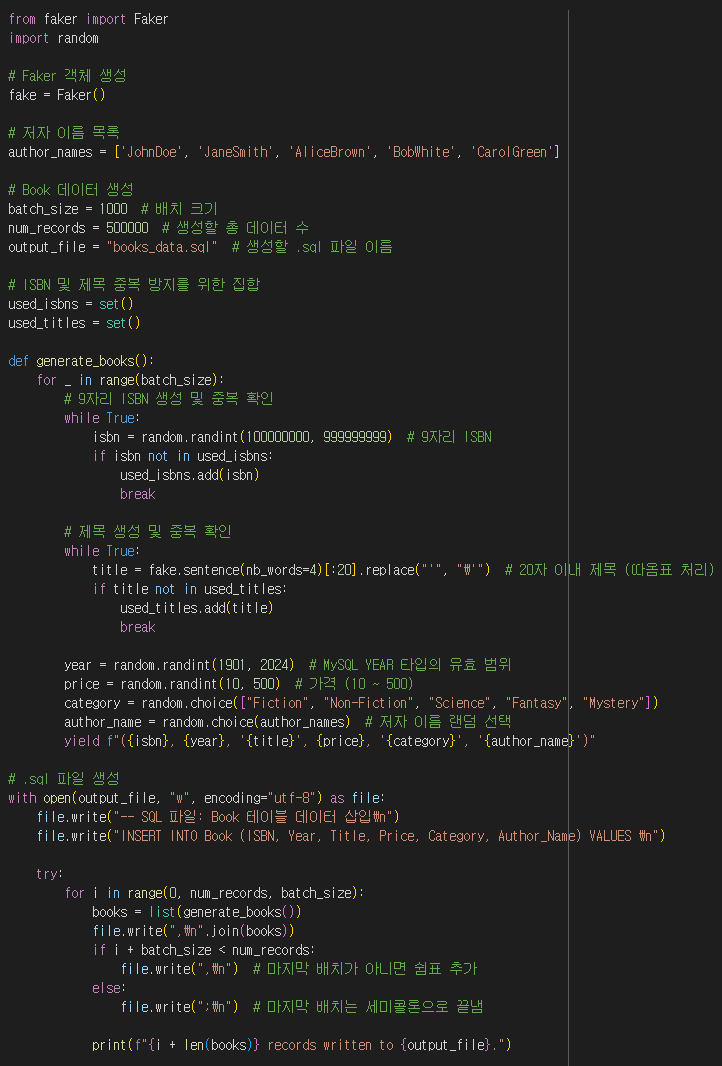
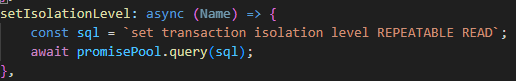
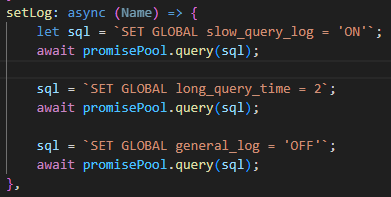
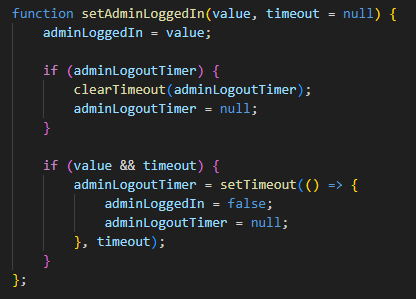
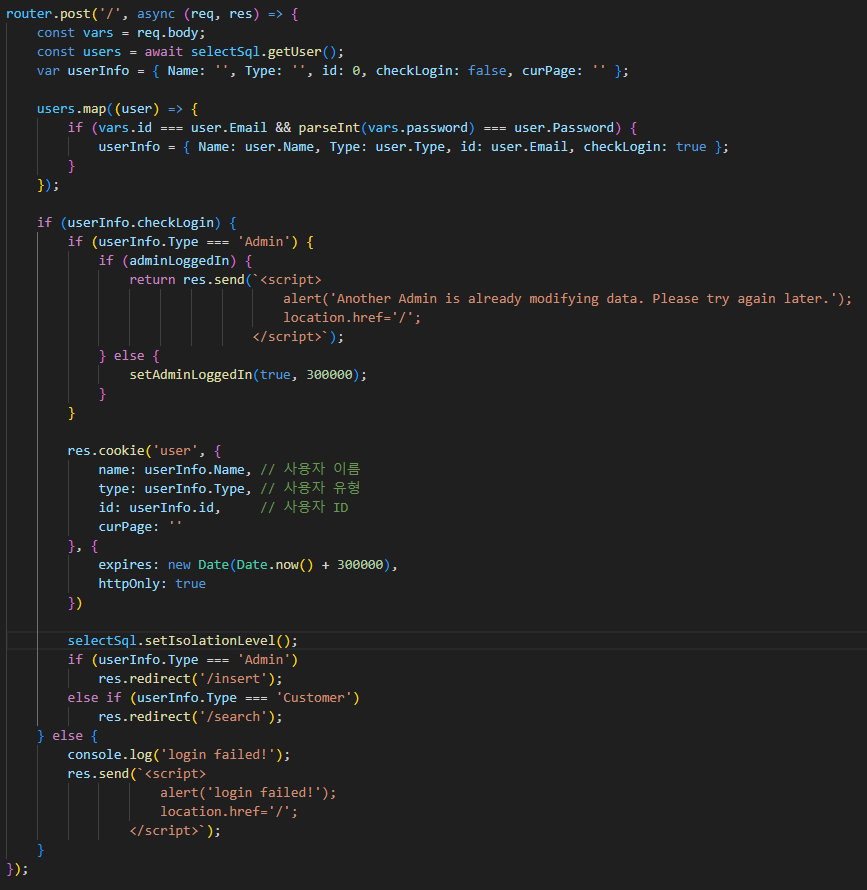
학번 12191806

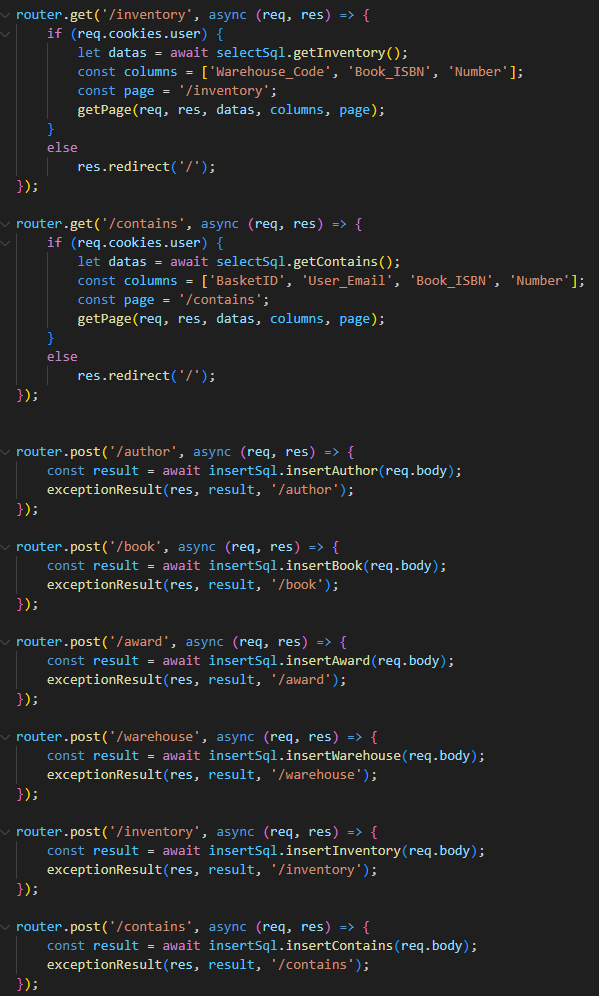
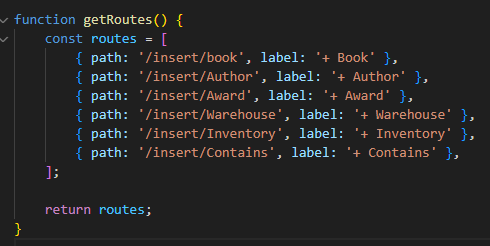
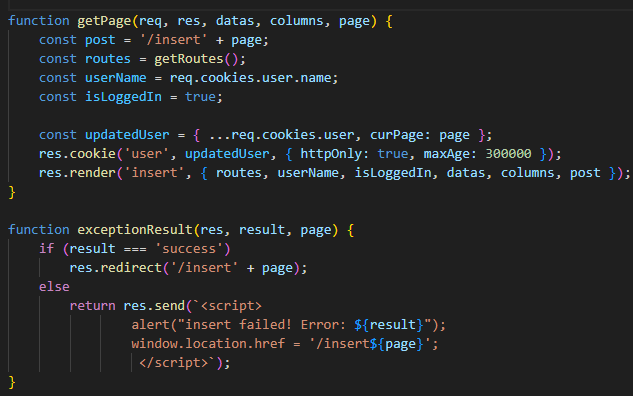


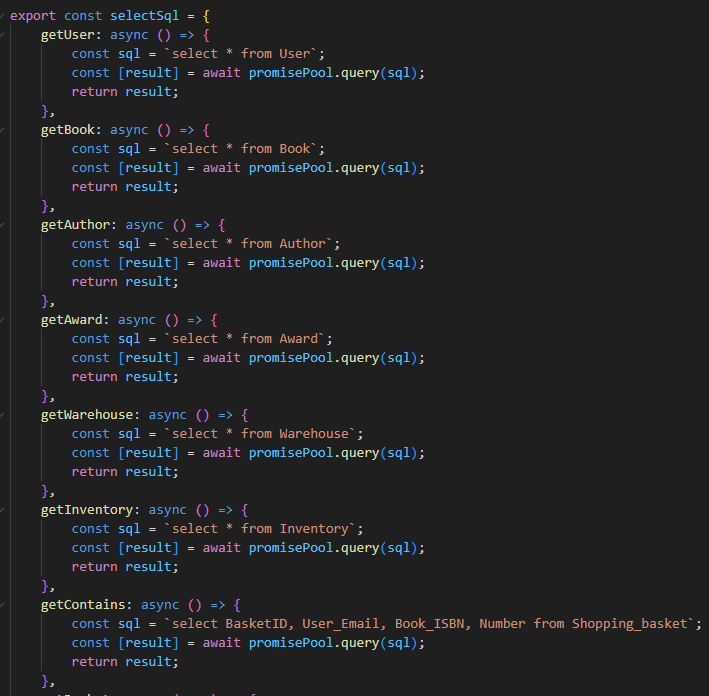
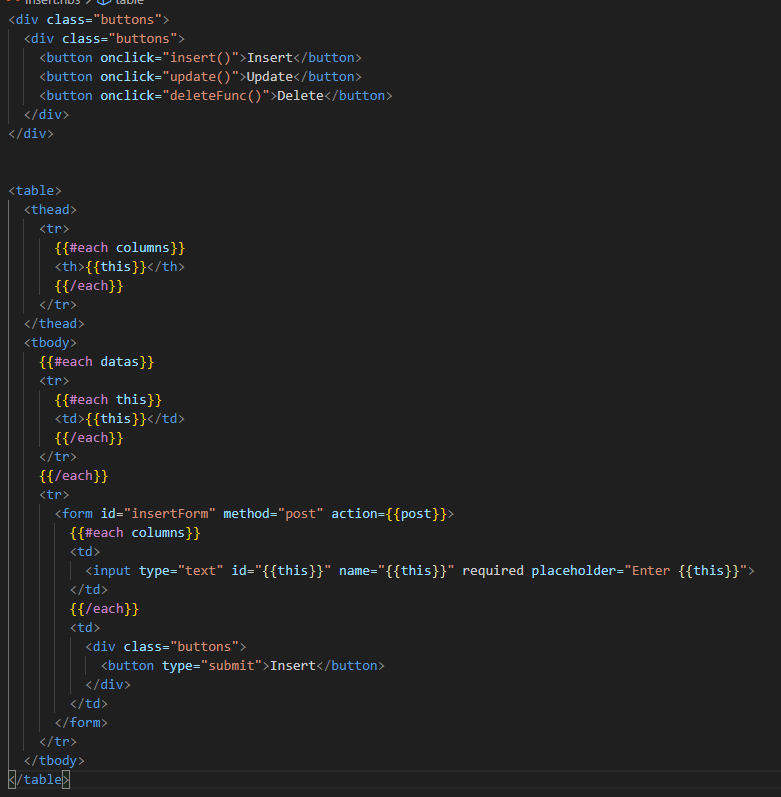
1. 개요

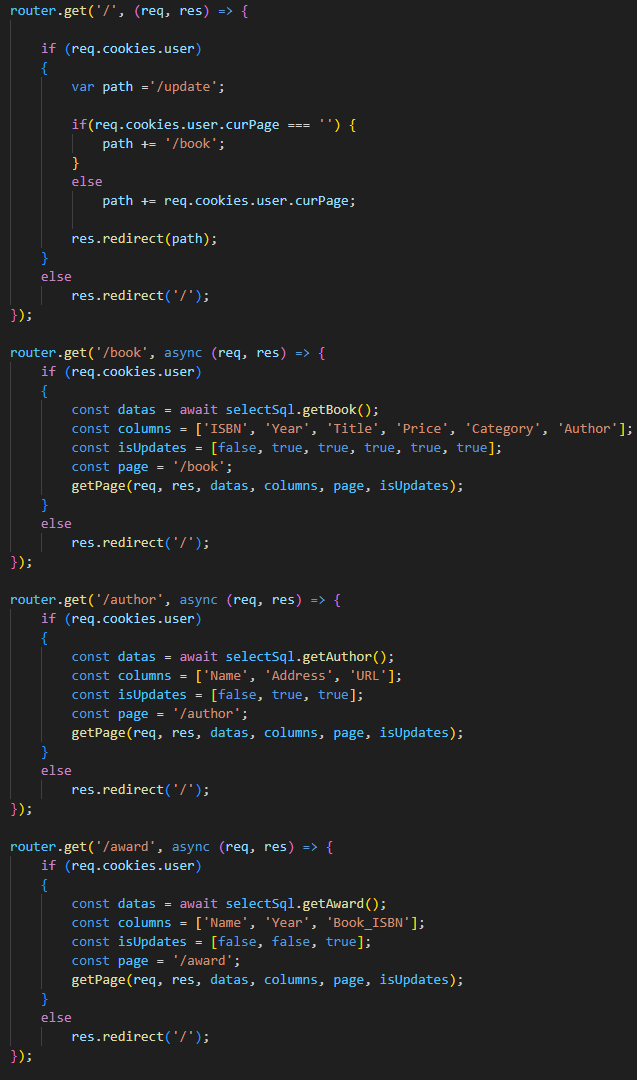
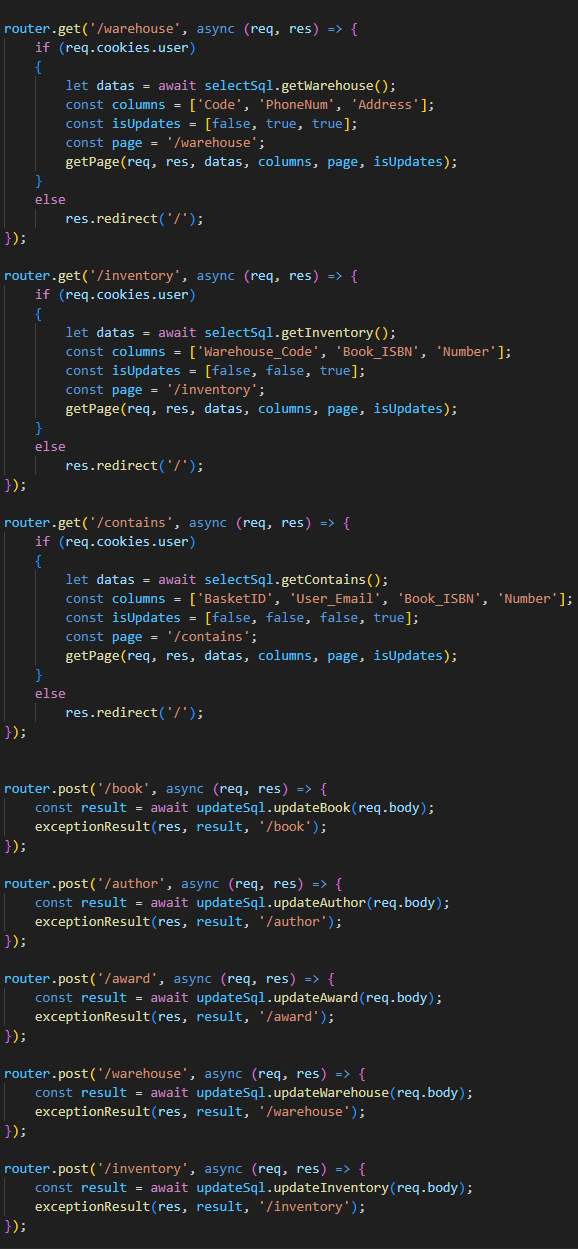
* 이번 Term Project는 주어진 EER 다이어그램을 바탕으로 Book Store 웹페이지를 설계하는 과제이다. 이때 구현해야 하는 기능은 각 Table의 데이터를 입력, 수정, 삭제가 가능한 관리자 페이지를 만들고 Book을 검색하는 Customer 페이지도 만드는 것이다. 이때, Customer의 경우 장바구니와 예약 시스템까지 활용할 수 있다. 이때, 이런 기능들을 구현하기 위해 제약조건, 정규화, Index, 트랜잭션, 로그 등을 설계하면 된다.

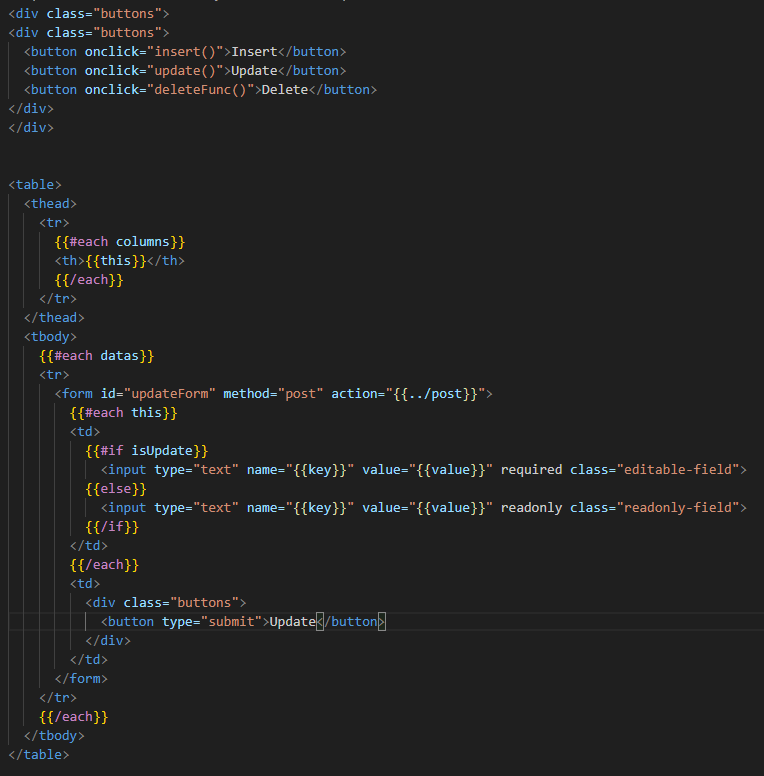
1. 상세 설계 내용

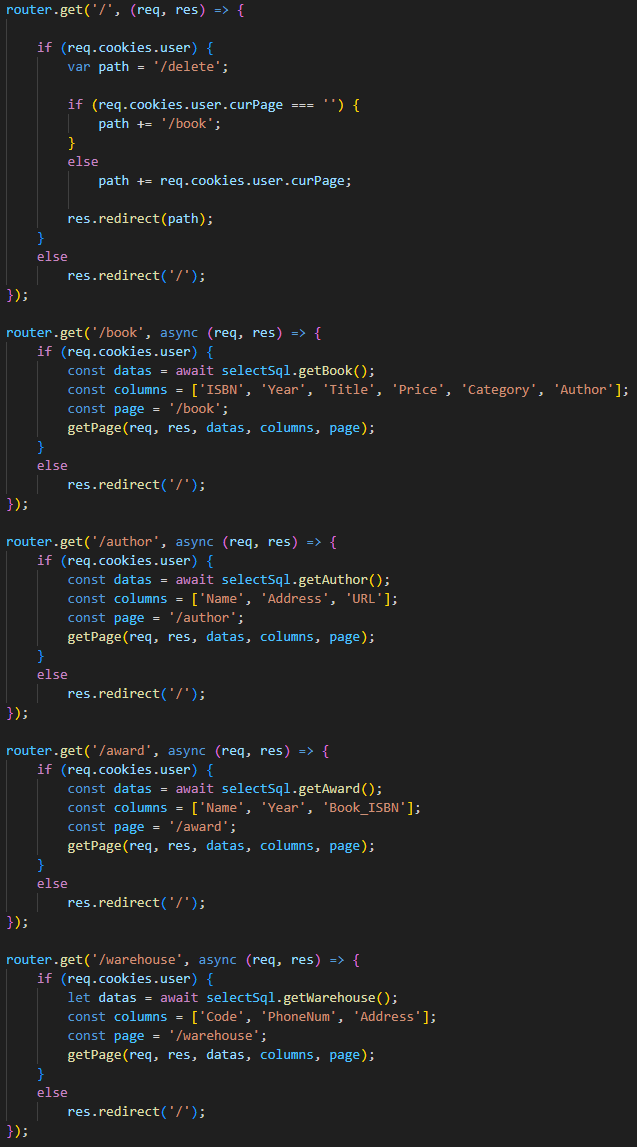
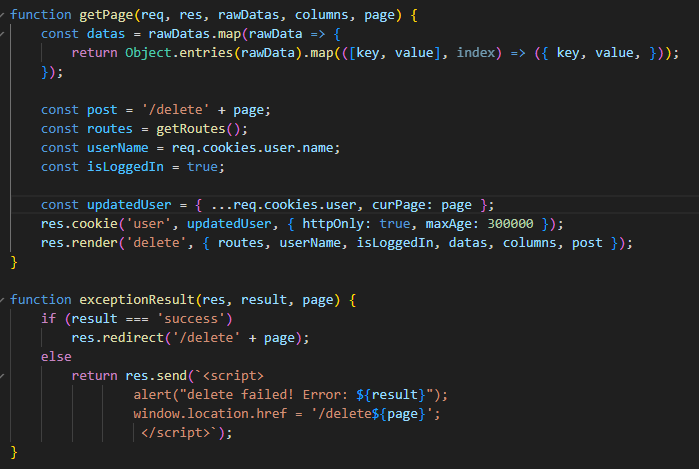
* 1. ERD 설계  
  
* 본 과제에서의 ERD는 다음과 같이 설계하였다. 우선 FK가 없는 가장 부모 테이블을 Author로 기준을 잡았다 Book과 Author는 서로 1:N 관계를 가진다. 우선 여기서 요구사항이 없기 때문에 하나의 책에 공동저자가 있는 경우는 제외하고 무조건 1개의 책에는 저자가 한명이라는 가정을 통해 1:N이라는 관계를 가정하였다. 따라서 Author에는 FK가 없고 Name을 PK로 설정하였다.
* 다음으로 모든 Table의 중심이 되는 Book이다. Book은 ISBN을 PK로 가지고 Author와 1:N 관계이기 때문에 Author\_Name을 FK로 가진다. 다른 테이블과의 관계를 설명하자면 Award와는 하나의 책은 여러 상을 받을 수 있지만 하나의 상으로 여러 책이 공동 수상을 하는 경우 역시 요구사항이 없기 때문에 제외하였다. 따라서 하나의 상은 하나의 책이 수상하고 책의 저자 역시 1명이기 때문에 Award에는 Book의 ISBN만 FK로 가진다. 그리고 하나의 상이 같은 년도에는 하나의 책을 기준으로 했기 때문에 Award의 경우 Name과 Year를 PK로 가진다. 주어진 EER에서는 Award와 Author사이의 Received\_by라는 관계가 있지만 이는 Award가 Book을 FK로 가지고 Book이 Author를 FK를 가짐으로써 만족할 수 있다. 이렇게 설계한 이유는 정규화 부분에서 설명하겠다.
* 그리고 Book은 주어진 EER 상에서 Customer와 2개의 관계, Warehouse와 1개의 관계를 가지는데 이는 전부 M:N 관계를 가지며, 각 관계 테이블을 Inventory, Shopping Basket, Reservation이라 하였다. 가장 먼저 Warehouse의 경우는 창고에 대한 정보이고 A라는 창고에 B라는 책이 몇권 있는가에 대한 정보를 Inventory를 통해 알 수 있다. Warehouse는 창고 Code를 PK로 가지고 Inventory의 경우는 관계 테이블이기 때문에 창고의 코드와 책의 ISBN이 책의 수량을 결정한다.
* Shopping Basket의 경우 주어진 EER에서는 Shopping Basket이 하나의 객체이고, 각각 Book과 Customer와 Contains와 Basket\_of라는 관계를 가지고 있어서 원래는 Contains라는 관계테이블과 Shopping Basket 테이블로 나눠서 만들었다. 이는 정규화를 고려했다고 생각했고 다음과 같이 설계하였었다.  
    
  하지만 이러한 설계는 다양한 문제가 있었는데 Customer가 Shopping Basket에 대한 리스트를 조회할 때, 조인해야 할 테이블이 많아진다는 점과 중복되는 데이터가 너무 많아진다는 문제점이었다. 물론 이렇게 구현을 한다면 구매내역과 Shopping Basket을 분리하여 작성하기 쉬워지지만 과제의 요구사항에 이러한 내용이 없었고 Shopping Basket의 구매여부만 조회하면 되므로 두 테이블을 합쳐서 하나의 테이블로 만들었다. 정규화에 대한 문제도 확인해봤는데 복합키지만 Basket ID, User Email, Order Date 이 세가지 정보가 결정되지 않는다면 Shopping Basket에 대한 정보는 결정될 수 없고 결정된다면 Unique한 값을 가지기 때문에 테이블을 합쳐도 문제가 없다고 생각하였다. 따라서 Shopping Basket을 User와 Book 사이의 M:N 관계 테이블로 정의하고 Basket ID와 두개의 FK인 User Email과 Book ISBN을 PK로 설정하였다.
* 다음으로 Reservation은 주어진 EER에서는 관계없이 이어져만 있지만 나는 Reservation 또한 M:N의 관계 테이블이라고 생각하였고 왜냐하면 양쪽의 테이블의 PK를 FK임과 동시에 PK로 가지기 때문이다. Reservation도 Shopping Basket과 마찬가지로 RID, Book ISBN, User Email을 PK로 가진다. 이렇게 설계를 함으로써 Shopping Basket과 Reservation이 비슷한 구조를 가지게 되어 코드를 작성함에 있어서 좀 더 편하게 작성할 수 있었다.
* 2. 제약조건
* 제약조건에 대해 설명하기 전에 추후 코드를 보면서 더 자세히 설명하겠지만, 나는 대부분의 예외사항에 대해서 try catch문을 활용해서 시스템이 다운되지 않도록 하고 SQL에서 에러 메시지를 출력하면 이를 그대로 사용자가 볼 수 있도록 하였다. 이런 에러 메시지를 통해서 허용되지 않는 접근이라는 것을 사용자에게 인식시켜주었다.  
    
  
* 제약조건 중에 가장 간단한 제약조건은 FK에 대한 제약조건이 있다. Workbench에서 FK를 등록하면 On Update, On Delete 즉, FK에 연결된 PK가 Update되거나 Delete 되면 어떻게 동작할 것이냐는 옵션이 있는데 이러한 제약조건은 전부 No Action으로 등록하였다. 그 이유는 Update 동작에서는 애초에 PK를 수정하는 것을 허용하지 않았다. 왜냐면 PK가 수정되면서 발생될 예외가 너무 많고 Insert 기능도 제공하기 때문에 Insert를 통해 PK를 수정할 수 있기 때문이다. 그리고 만약 FK로 연결되어 있는 데이터를 삭제하려고 하면 위와 같이 에러 메시지를 띄우는 식으로 해결하였다.
* PK에 대한 제약조건은 NULL값을 허용하지 않고 중복을 허용하지 않는 조건이 있는데 이는 Workbench에서 설정할 수 있는 NOT NULL과 UNIQUE가 있다. PK는 기본적으로 이 두가지 조건을 만족해야 한다. 따라서 PK는 이 두가지 제약조건이 자동으로 설정되어 있고 모든 Column에는 NOT NULL을 설정하였다. 하지만 Shopping Date의 Order Date는 아직 구매하지 않았다는 의미로 NULL 값을 활용하기 때문에 이 Column만 NOT NULL을 적용하지 않았다. 그리고 UNIQUE는 서비스적으로 중복을 허용하지 않는 Column에 대해서 적용하였다. 이는 Book의 Title, Ware house의 Address, User의 Name, Password가 있다.
* 다음으로는 허용되지 않는 값에 대한 제약조건이다. 여기서 허용되지 않는 값에는 수량, 가격 등에 음수를 Insert하는 행위가 있는데 이는 Workbench에서 Column의 옵션들 중에서 Unsigned를 켜주면 적용할 수 있다.  
  
* 다음으로 요구사항에 대한 제약조건이다. 과제에서 제시한 요구사항 중에는 설정하려는 픽업시간 전후 10분 이내에 다른 예약이 있다면 예약할 수 없다는 요구사항이 있다. 이 요구사항에 대한 제약조건을 만들기 위해 Trigger를 사용하였고 Workbench에서 Trigger를 추가할 수 있어서 Before Update와 Before Insert에 추가하였다.  
    
    
  코드를 살펴보면 같은 날짜의 PickupTime이 10분 내외면 제한하도록 했고 Error SIGNAL을 추가하여 try catch 문을 통하여 Error Message가 출력되도록 하였다.
* 3. 정규화, 테이블 분석
* 각 테이블들의 정규화 단계와 함께 테이블의 각 Column을 어떻게 구성했는지에 대한 분석을 함께 진행할 것이다.
* Author  
    
  정규화를 먼저 살펴보면 모든 Column이 원자성을 띄기 때문에 1NF를 만족하고 작가의 이름이 Address와 URL을 결정하므로 부분 함수 종속도 없다. 또한 Address와 URL은 이행 함수 종속이 없어서 3NF도 만족한다. 그리고 Column 들 중에서 Name만 결정자인데 Name이 Key 이므로 BCNF 또한 만족한다고 볼 수 있다. 이때, 원래는 동명이인을 고려하여 Name을 후보키로 생각하고 ID라는 PK를 사용했다가 Name이 PK가 되는 것이 좀 더 명시적이라고 생각하여 Name을 PK로 하고 동명이인이 존재하는 경우는 제외하였다.
* Book  
    
  Book 또한 모든 Column이 원자성을 띄기 때문에 1NF를 만족하고 PK는 ISBN이다. Author Name도 PK가 될 수 있지 않을까 했지만 1명의 작가가 책은 1개만 쓰는게 아니라 여러 권의 책을 쓸 수 있기 때문에 Author Name은 FK이다. 또한 PK가 하나이기 때문에 2NF를 만족한다. 그리고 다른 Column들 중에는 결정자가 없기 때문에 3NF과 BCNF를 만족한다. 그리고 Title의 경우 같은 이름의 책은 서비스 적인 관점에서 이상하다고 생각해서 Unique 옵션을 통해 이를 금지하였다.
* Award  
    
  Award 또한 모든 Column이 원자성을 띄기 때문에 1NF를 만족한다. 처음에는 Book ISBN과 Author Name을 복합키로 설정했으나 그렇게 설정했을 때의 문제는 입력을 할 때, Author가 작가가 아닌 Book ISBN을 입력했을 때의 이상이 발생할 수 있기 때문에 Author Name은 제외하였고 정규화 관점에서 본다면 Book ISBN이 Author Name을 결정하기 때문에 발생하는 문제이다. 따라서 이는 이행 종속이고 3NF에 위반되기 때문에 Author Name을 제외하였다. 그리고 Book ISBN 또한 Unique한 값이 아니기 때문에 PK가 될 수 없다. 오직 Name과 Year가 복합키 일 때 Unique를 만족하므로 PK로 선정하였다. 또한 앞에서 말한 문제들을 해결하여 3NF와 BCNF를 만족한다.
* Warehouse  
    
  Warehouse 또한 모든 Column이 원자성을 띄기 때문에 1NF를 만족한다. 이는 EER에 있는 구조를 그대로 활용하였고 PK가 하나이기 때문에 2NF를 만족한다. 그리고 같은 주소에 두개의 Warehouse가 있을 순 없으므로 Addres에 Unique를 적용하였다. 또한 Code가 유일한 결정자이므로 3NF와 BCNF를 만족한다.
* Inventory  
    
  Inventory 또한 모든 Column이 원자성을 띄기 때문에 1NF를 만족한다. Inventory는 Warehouse와 Book의 관계 테이블 이므로 Warehouse Code와 Book ISBN을 복합 PK로 가진다. 그리고 책의 수량을 Number로 표기한다. 이때, Number는 결정자가 아니기 때문에 2NF, 3NF, BCNF를 만족한다.
* User  
    
  User 또한 모든 Column이 원자성을 띄기 때문에 1NF를 만족한다. User는 본인의 로그인을 위한 ID인 Email을 PK로 사용한다. PK가 1개이기 때문에 2NF를 만족하고 유저의 이름과 Password는 유일해야 한다고 생각하여 Unique로 지정하였다. 이 둘도 Key가 될 수 있지만 Email이 로그인을 검사하기 때문에 Email을 PK로 설정하였다. 또한 Name과 Password가 결정하는 Column이 이행 종속이라고 보기는 힘들기 때문에 3NF와 BCNF 역시 만족한다. 그리고 Type은 유저가 Admin인지 Customer인지에 대한 Type이고 로그인 시 이를 검사한다.
* Shopping Basket & Reservation  
     
  Shopping Basket과 Reservation은 구조가 비슷하여 같이 설명하겠다. 이 둘 또한 모든 Column이 원자성을 띄기 때문에 1NF를 만족한다. 이 둘 모두 User와 Book의 관계 테이블 이므로 User Email과 Book ISBN을 PK로 가진다. 하지만 이때, 이 두개의 키를 복합키로 설정하면 한명의 유저가 하나의 책에 대해 하나의 장바구니 또는 하나의 예약만을 가질 수 있기 때문에 이를 해결하기 위해 각 Table에 ID Key를 추가하였다. 그리고 이는 데이터를 추가할 때 자동으로 생성될 수 있도록 Auto Increment를 켜주었다. 그리고 OrderDate는 Date 자료형이고 PickupTime은 Time 자료형으로 설정하여 사용하기 편하게 하였다. 그리고 Shopping Basket의 OrderDate는 NULL인 경우 구매하지 않았음을 판단하기 위하여 NOT NULL을 설정하지 않았다.
* 4. Index 최적화
* Index는 사용자가 직접 설정할 수도 있고 자동으로 Index가 적용되는 경우가 있다. 자동으로 Index가 적용되는 경우는 PK, FK, Unique인 경우이다. 원래 Index는 Select에서 굉장히 높은 성능 향상을 보이고 Insert나 Update에서는 오히려 성능이 하락하는 경우도 있다. 따라서 Select를 통해 검색이 자주 이뤄지는 부분에 Index를 추가하면 좋다. 현재 PK를 제외하고 검색이 자주 이뤄지는 부분은 책의 제목을 이용하여 책을 검색하는 페이지이다. 하지만 Title에는 이미 Unique가 적용되어 있기 때문에 Index 설정은 따로 하지 않아도 된다. 따라서 Title을 Index로 설정했을 때와 안 했을 때를 비교해서 최적화 차이를 비교해보겠다.  
    
  50만개의 Data를 집어넣어 테스트를 진행해보았고 Index를 적용하자 0.12초에서 0.01초로 쿼리 응답 시간이 개선된 것을 볼 수 있었다. 밑에는 50만개의 Data를 생성하는 Python 코드이다.  
  
* 5. 트랜잭션 설정
* 트랜잭션의 경우 여러 유저가 쿼리를 한번에 수행할 때, 다른 유저와의 충돌을 컨트롤하기 위해 사용하는데 따라서 쿼리를 Exception 하는 부분에 트랜잭션에 대한 내용을 추가하였다. 원래는 여러 개의 쿼리를 복합으로 Input으로 넣어야 하는데 과제를 구현하다 보니 SQL에서 제약조건 등의 예외처리와 프런트 코드에서의 예외처리를 통해 복합으로 쿼리를 구현하는 경우가 없었다. 따라서 하나의 함수로 트랜잭션을 구현할 수 있었다.  
    
  코드를 간단하게 설명하면 쿼리를 Input으로 받아서 트랜잭션을 시작하고 쿼리를 요청한 다음에 성공 시 commit과 함께 success를 return하고 실패 시 rollback과 함께 error message를 return 한다.
* 또한 isolation level을 설정할 수 있는데 이는 최대한 안전한 동작을 하기 위해 Dirty Read, Phantom Read, Repeatable Read가 발생하지 않는 Unrepeatable Read로 설정하였다.  
  
* 6. 로그 설정  
    
  로그설정은 기본적으로 mysql 설치파일에서 할 수 있지만 활성화 비활성화와 일부 설정은 런타임에서 수정할 수 있다. 나의 목표는 최대한 성능을 위해서 디버깅을 위한 general\_log은 비활성화하고 느리게 실행되는 쿼리를 식별하여 최적화에 도움이 되는 slow query log는 활성화 해주었다.
* 웹페이지 상세 구현
* Login 기능  
    
    
  로그인의 경우 Cookie를 이용하여 로그인 하였고 user의 Email과 Password를 이용하여 로그인 검사를 하였다. 그리고 user의 Type을 검사하여 Admin인지 Customer인지 판단하고 Admin이라면 Admin 기본 페이지인 /insert로 이동하고 Customer인 경우 /Search로 이동하였다. 또한 이미 로그인 된 상태에서 페이지에 접속하면 각자의 기본페이지로 이동하게 하였다. 그리고 과제의 요구사항 중에 하나인 관리자가 데이터를 수정 중이면 수정하지 못하게 하기 위하여 Admin이 로그인 한 상태면 다른 Admin이 로그인하지 못하도록 하였다. 또한 타이머를 작동시켜 Admin이 로그아웃을 누르지 않고 페이지를 나가더라도 자동으로 초기화될 수 있도록 하였다.
* 관리자 페이지 (Insert.js)

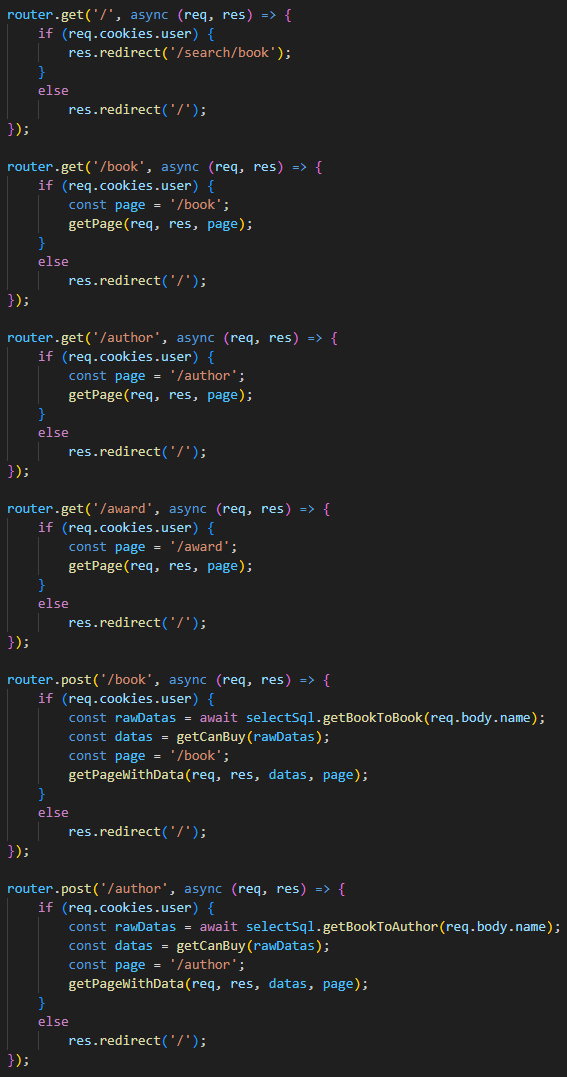
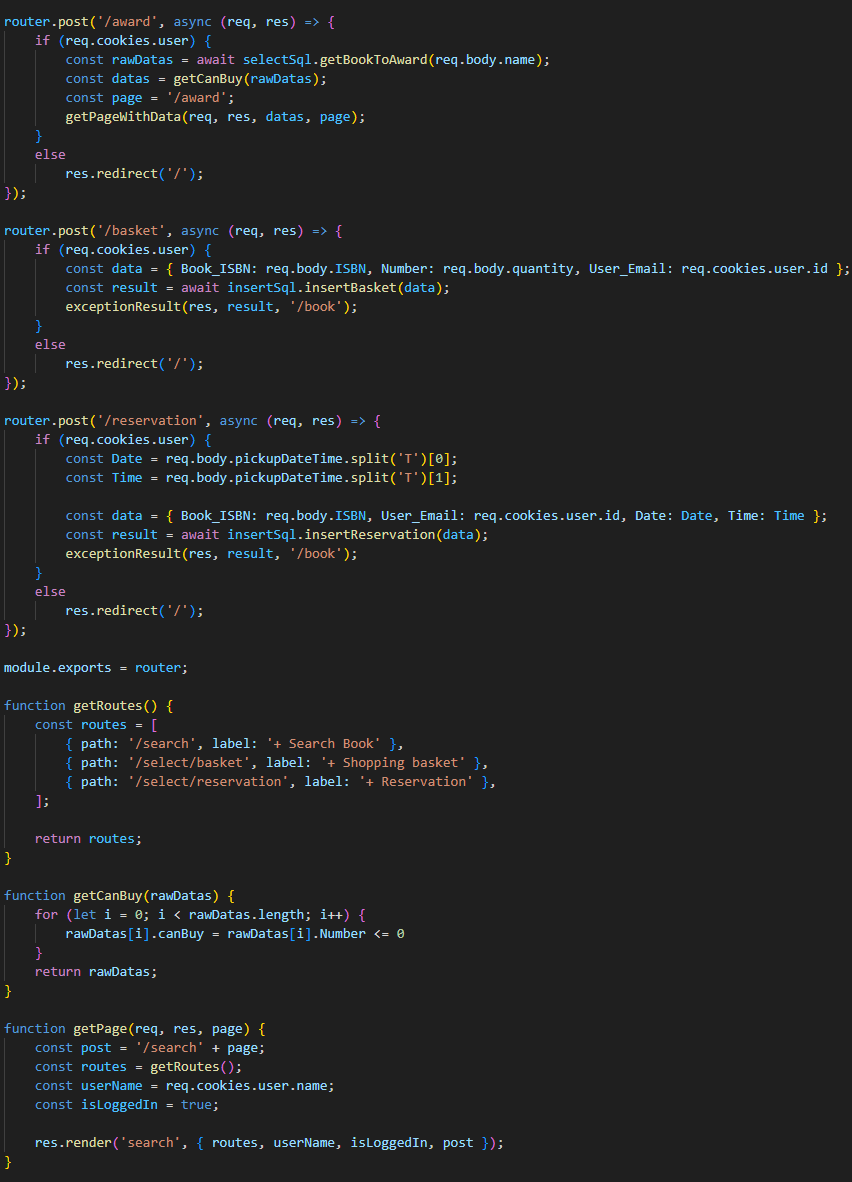
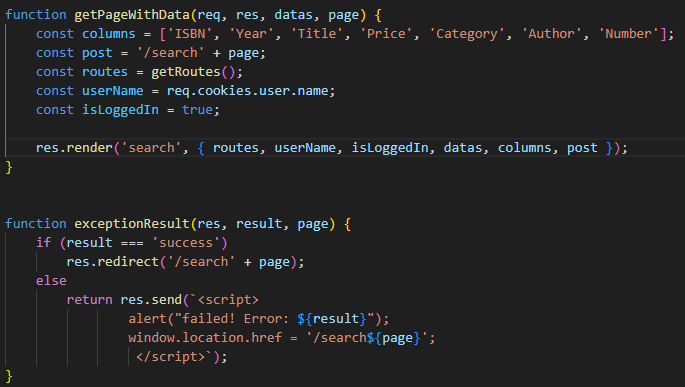
* 관리자로 로그인을 하게 되면 가장 먼저 들어오게 되는 Insert 페이지이다. Get과 Post 함수는 거의 비슷하게 생겨서 getPage와 ExceptionResult 함수가 핵심인데 GetPage에서는 데이터 들을 받아와 render함수를 통해 이를 hbs에 넘겨주는데 이때, columns는 각 열의 이름들이고 post는 버튼의 post 주소 그리고 data는 sql의 쿼리를 통해 받은 데이터이다. 그리고 중간에 cookie를 갱신해주는 이유는 curPage변수를 수정하기 위함인데 curPage 변수는 유저의 편의성을 위해서 Insert에서 Update페이지로 넘어가더라도 현재 수정 중이던 테이블 페이지를 유지하기 위함이다. 그리고 ExceptionResult 함수는 쿼리를 실행하고 난 이후 결과 메시지가 성공이면 redirect를 하고 아니라면 에러 메시지와 함께 페이지를 redirect 해준다. 관리자가 Insert, Update, Delete를 하며 발생하는 SQL에서의 예외나 에러는 DB를 설계하며 이미 SQL에서 에러를 전부 생성하게 하였고 여기서 생성된 에러는 시스템을 다운 시키는 것이 아닌 프런트로 에러 메세지를 보내서 이를 띄워준다.
* Insert Query 
* Get Query  
  
* Insert와 get 쿼리는 굉장히 간단하게 작성하였고 예외가 있다면 나는 Contains 테이블을 만들지 않고 Shopping Basket 테이블로 합쳤기 때문에 Shopping Basket에 접근할 때만 OrderDate를 제외하고 접근하였다. 만약 더 자세하게 구현한다면 Shopping Basket에서 Contains라는 View를 추가할 것 같다. 그리고 Insert 할 때는 아직 구매하지 않은 물건임을 가정하고 null을 입력하였다.
* Insert 페이지  
    
  Insert의 hbs 파일이다. 대부분 비슷한 구조를 공유하고 있는데 우선 모든 Table의 데이터를 출력하는 페이지를 만들기 위하여 원래는 data를 for문으로 각 데이터의 이름을 적어주었지만 여기서는 2중 for문을 이용하였다. 또한 Insert Update Delete 버튼은 공통된 버튼이라 Layout.hbs에서 함수를 다루고 있다.
* 관리자 페이지(Update.js)

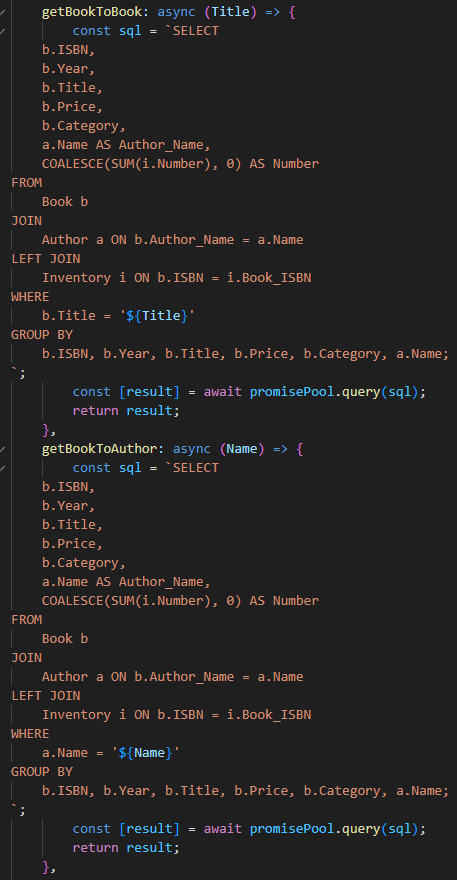
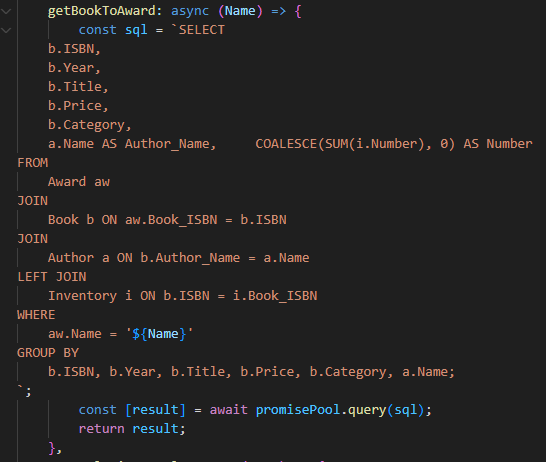
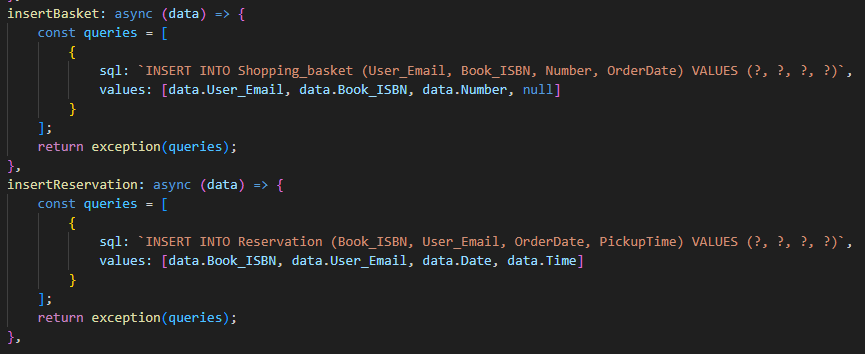
   
 

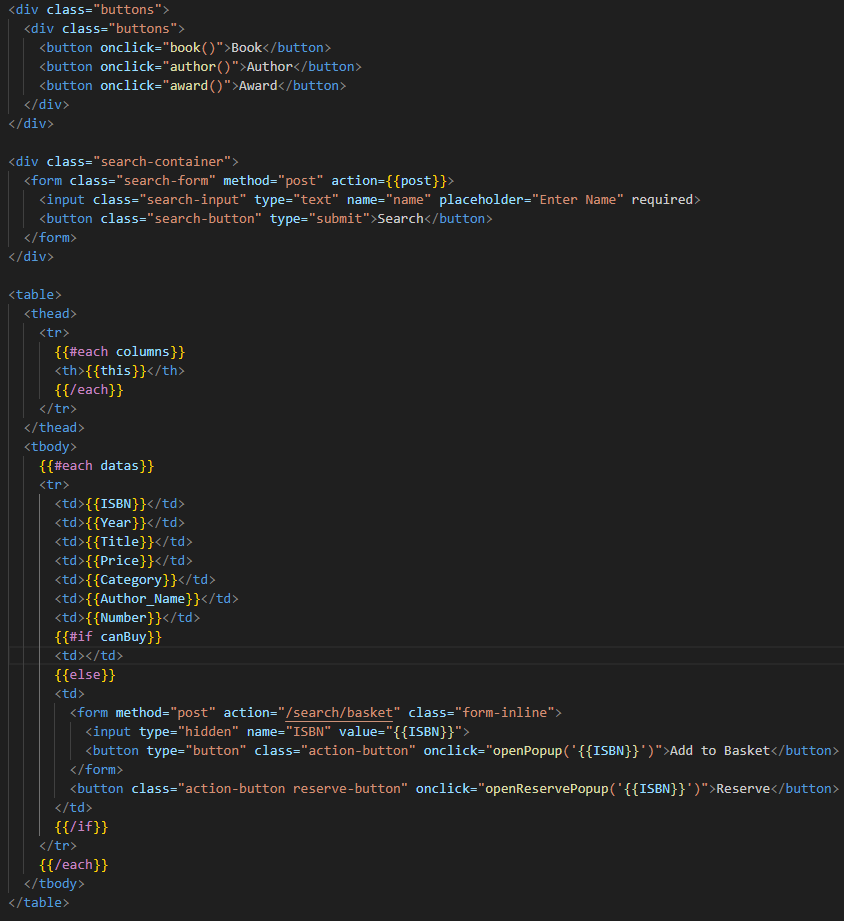
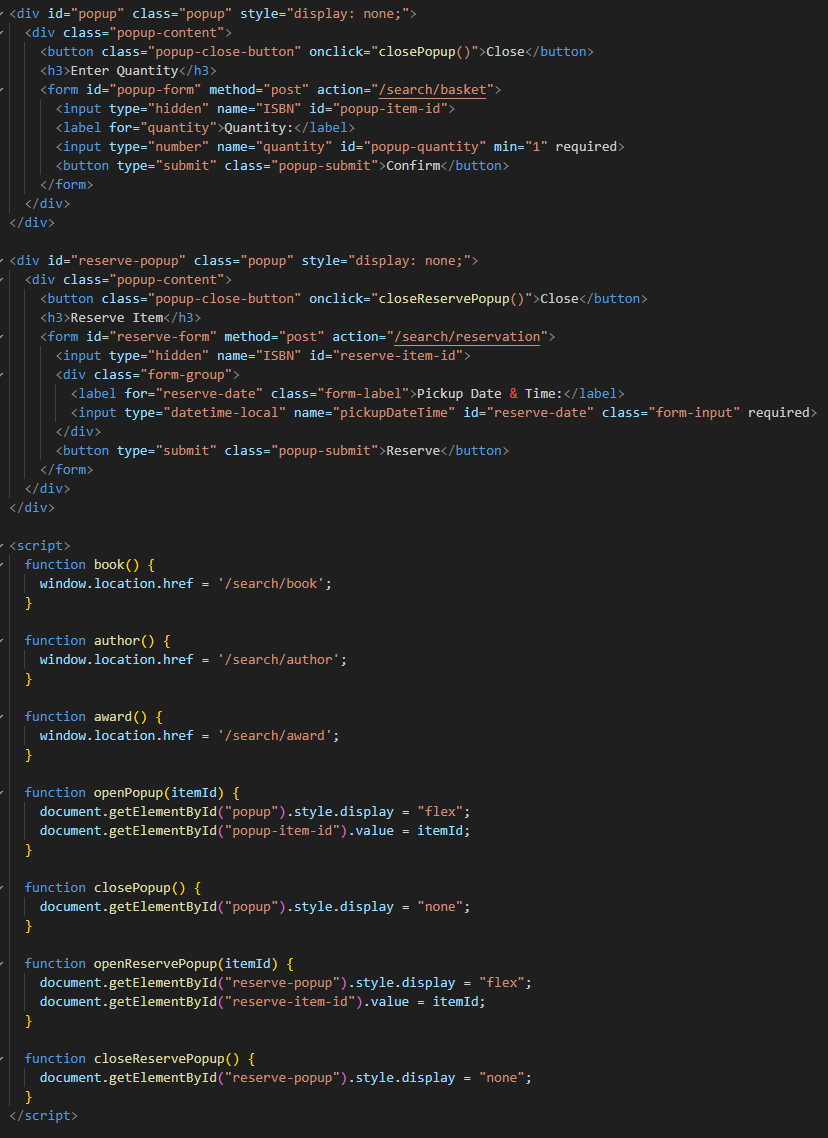
* Update의 경우 Insert와 거의 비슷하지만 조금 다르다. 다른건 전부 똑같지만 getPage에서 업데이트가 가능한지에 대한 여부를 data에 추가해준다. 추가하는 이유는 PK의 경우 업데이트가 불가능해야 하기 때문에 이를 방지하기 위하여 PK들은 수정이 불가능 하도록 hbs에서 업데이트가 불가능 한지를 체크한다.
* Update Query   
    
  Update 쿼리의 경우에도 특별한 것은 없고 Post를 통해 가져온 Data를 이용하는데 PK를 통해 검색을 하고 검색된 데이터에 가져온 데이터를 넣어주는 쿼리이다.
* Update 페이지  
    
  Update 페이지의 경우에는 아까 data에 추가해주었던 isUpdate를 통해서 만약 업데이트 불가능 하다면 readonly로 input field를 바꿔서 처리한다. 그리고 이 input field의 name과 value를 각각 Key와 Value로 설정하였는데 이렇게 하기 위해서 아까 isUpdate를 data에 추가할 때 data의 Key와 Value를 ISBN: 1010에서 Key: “ISBN”, Value: 1010 이런 식으로 나누어 주었다. 이렇게 한 이유는 마찬가지로 모든 테이블에 대응하는 hbs를 짜기 위함이다.
* 관리자 페이지(Delete.js)

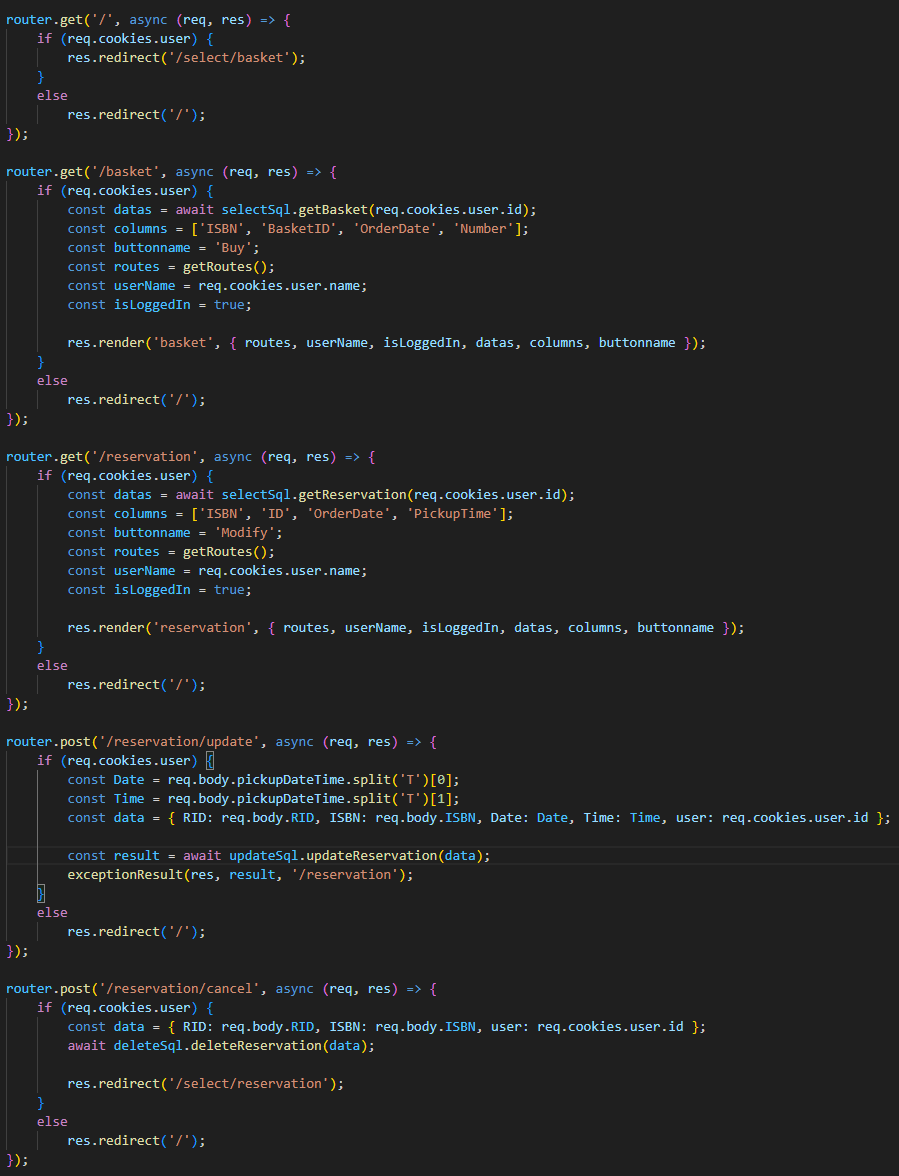
* Delete.js의 경우 insert와 구조가 거의 동일하다. 하지만 Update에서의 이유와 마찬가지로 data를 Key, Value로 분리해 주었다.
* Delete Query  
    
  Delete에 대한 쿼리도 Update와 비슷하게 Post를 통해 받아온 Data에서 PK를 통해 검색을 하고 해당 데이터를 삭제하였다.
* Delete 페이지  
    
  Delete 페이지 에서는 Update는 모든 칸에 form을 넣었다면 Delete 는 각 행마다 Delete 버튼을 넣었다. 그리고 버튼에 데이터를 담기 위해 input을 hidden으로 사용하였고 아까 분리했던 Key와 Value를 넣어주었다.
* Customer 페이지(Search.js)

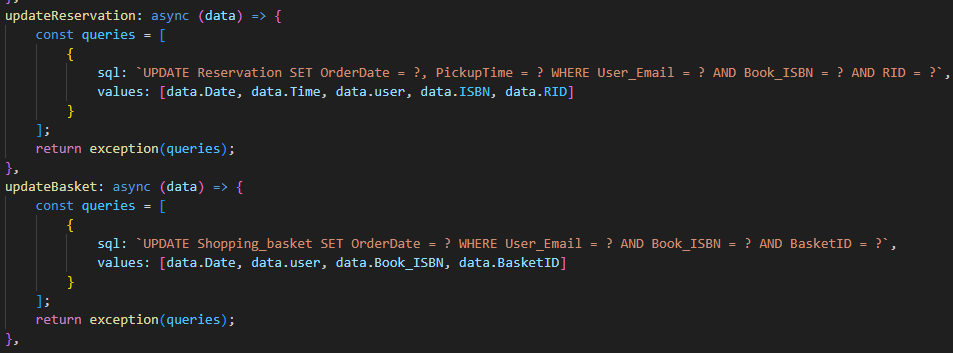
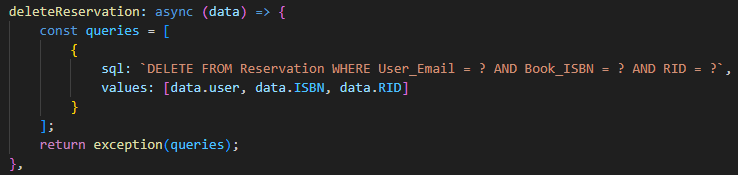
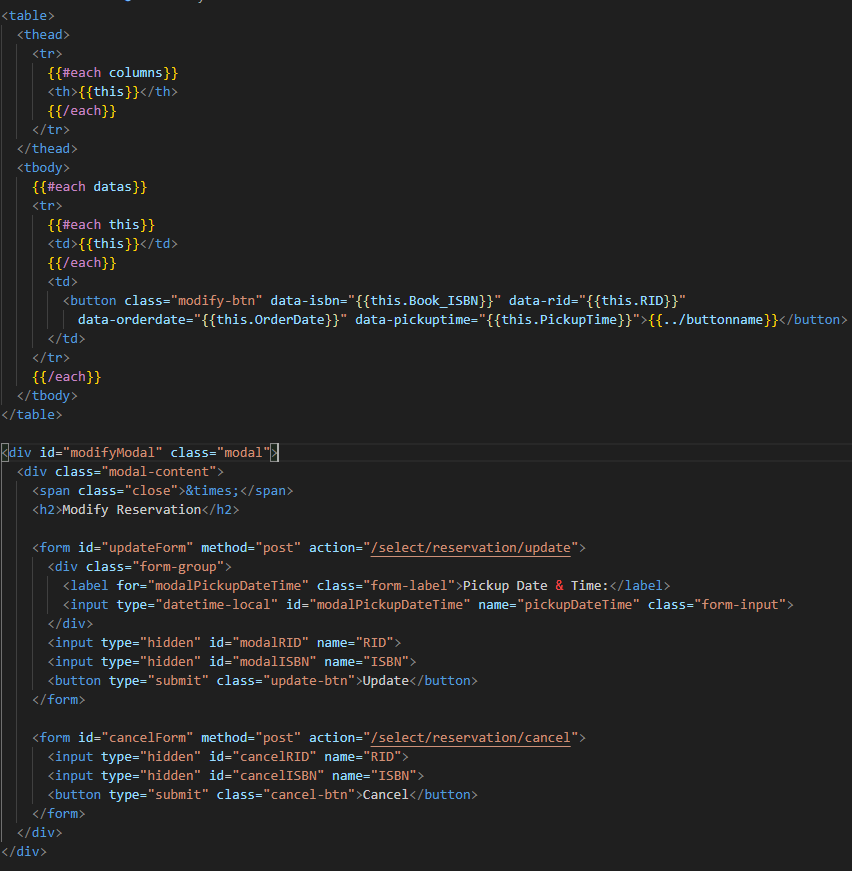
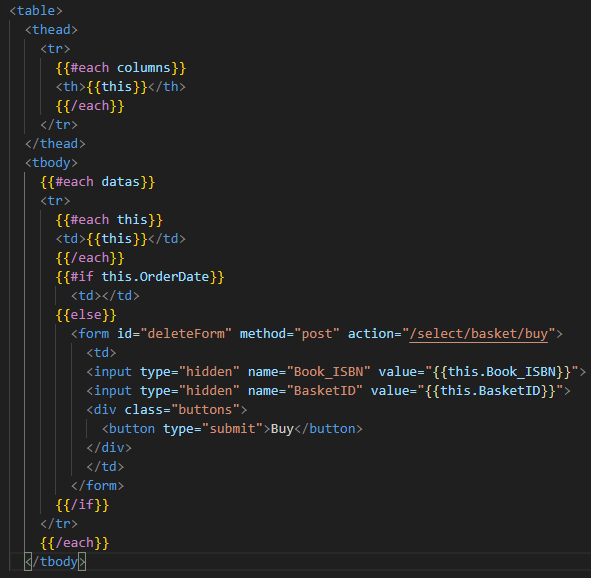
   


* Customer 페이지의 기본 상태인 Search 페이지이다. Search는 과제의 요구사항인 책을 검색하고 책을 장바구니에 담거나 예약할 수 있는 페이지이다. 이때, 책을 책 제목, 작가 이름, 상 이름으로 검색할 수 있어야 하고 각각을 검색할 수 있는 페이지를 만들었다. getPage 함수를 이용해 render를 하는데 기본상태는 아무런 데이터가 없고, 이 함수는 Insert 페이지의 getPage 함수와 거의 유사하지만 curPage를 유지하지는 않는다.
* Post를 하는 경우에는 hbs에서 받아온 제목을 이용하여 쿼리를 실행하고 이 결과를 render에 이용하는데 이때, 쿼리의 결과 중 책의 총 수량인 Number가 0보다 작거나 같으면 CanBuy를 라는 Key를 추가해 이를 False로 하여 hbs에서 장바구니 담기와 예약 버튼을 비활성화 하도록 하였다.
* 그리고 밑의 basket과 reservation은 장바구니 담기와 예약 기능인데 이를 통해 Shopping Basket과 Reservation의 Insert를 실행할 수 있고 데이터는 Post를 통해 온 req를 활용한다. 이때, Date와 Time은 한번에 가져와 지므로 이를 Split하여 쿼리를 실행한다.
* Get Book Query  
     
  책을 검색하는 z쿼리의 경우에는 Join과 Group By를 활용하였다. 이를 활용하여 해당 제목에 해당하는 책을 검색하고 여기서 어려운 점은 Inventory에 있는 책의 총 수량을 가져오는 작업인데 이때, Inventory에 해당 ISBN의 책이 아예 존재하지 않는다면 NULL 값을 받고 EmptySet이 나오기 때문에 이를 방지하기 위해서 Left Join과 COALESCE를 활용하여 Empty Set과 NULL을 방지하였다.
* Insert Shopping Basket Reservation Query  
  ****  
  이 쿼리는 Post를 통해 정제된 데이터를 각각 Basket과 Reservation에 Insert 하는 쿼리인데 이때, Basket은 아직 책을 구매한 것이 아니라 책을 담은 것이기 때문에 OrderDate는 null로 설정하였다.
* Search 페이지

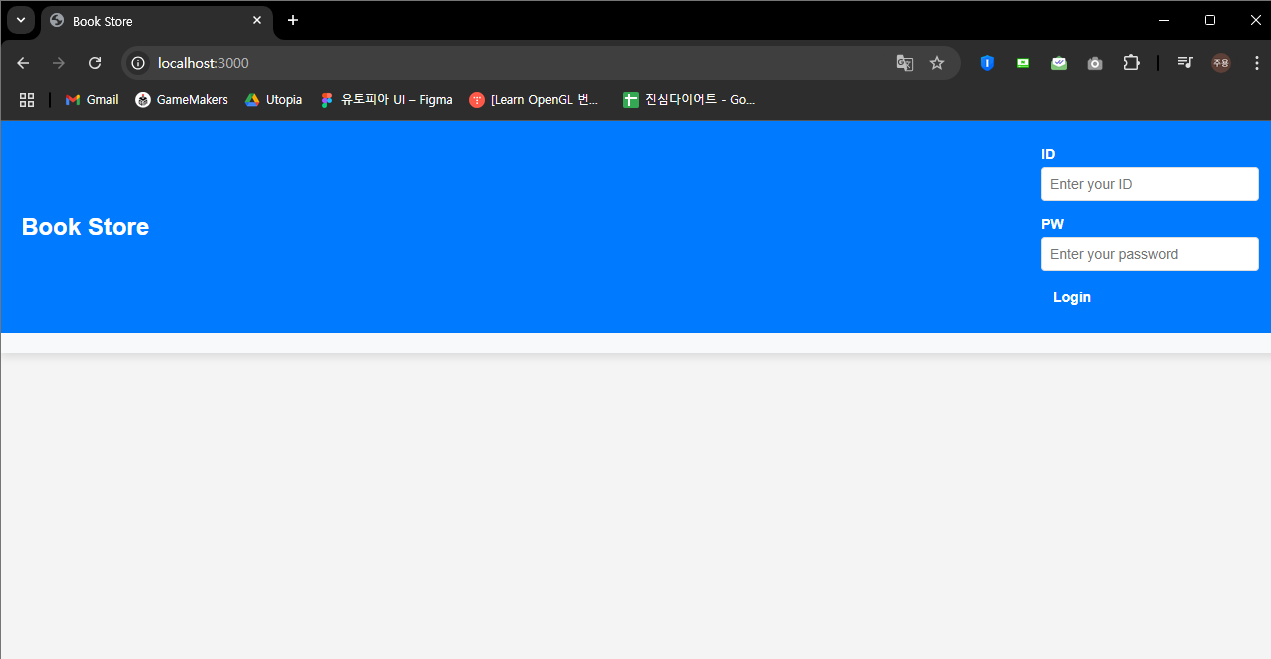
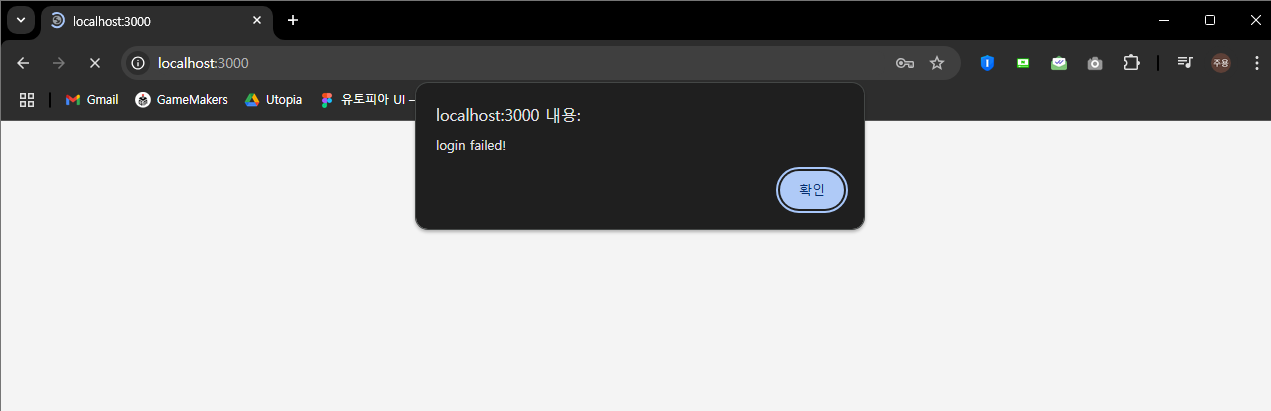
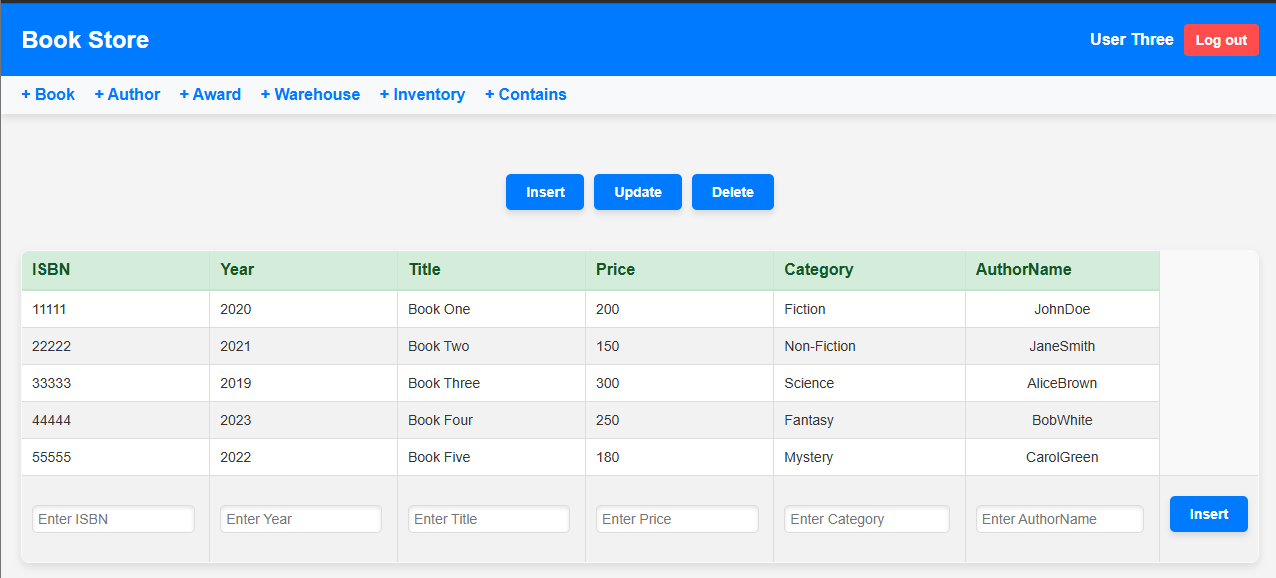
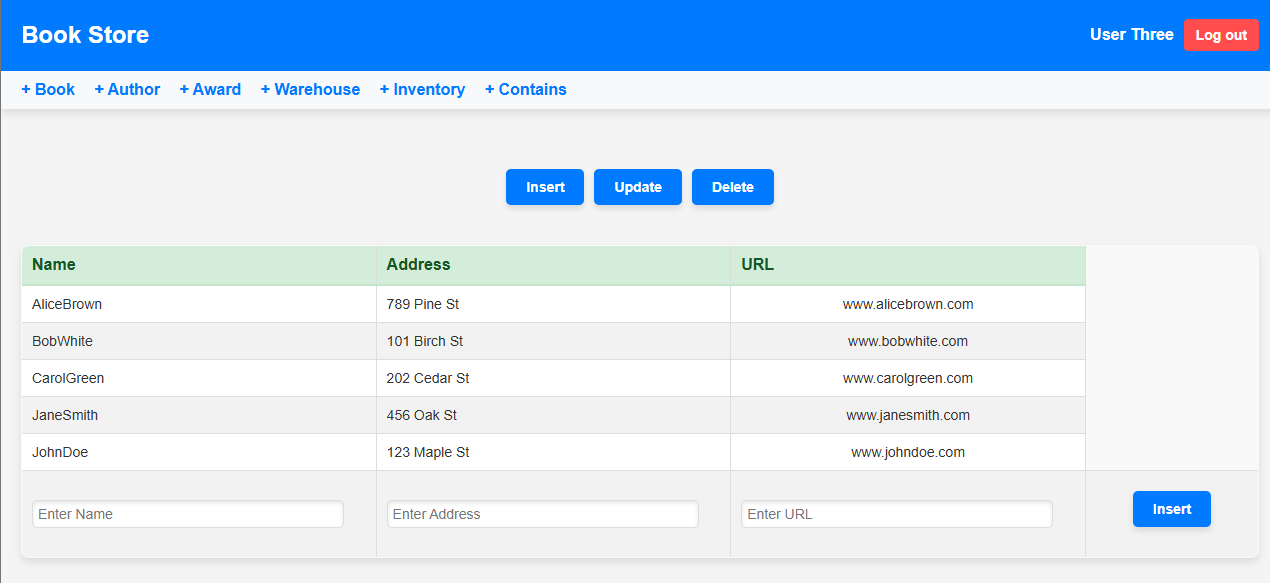
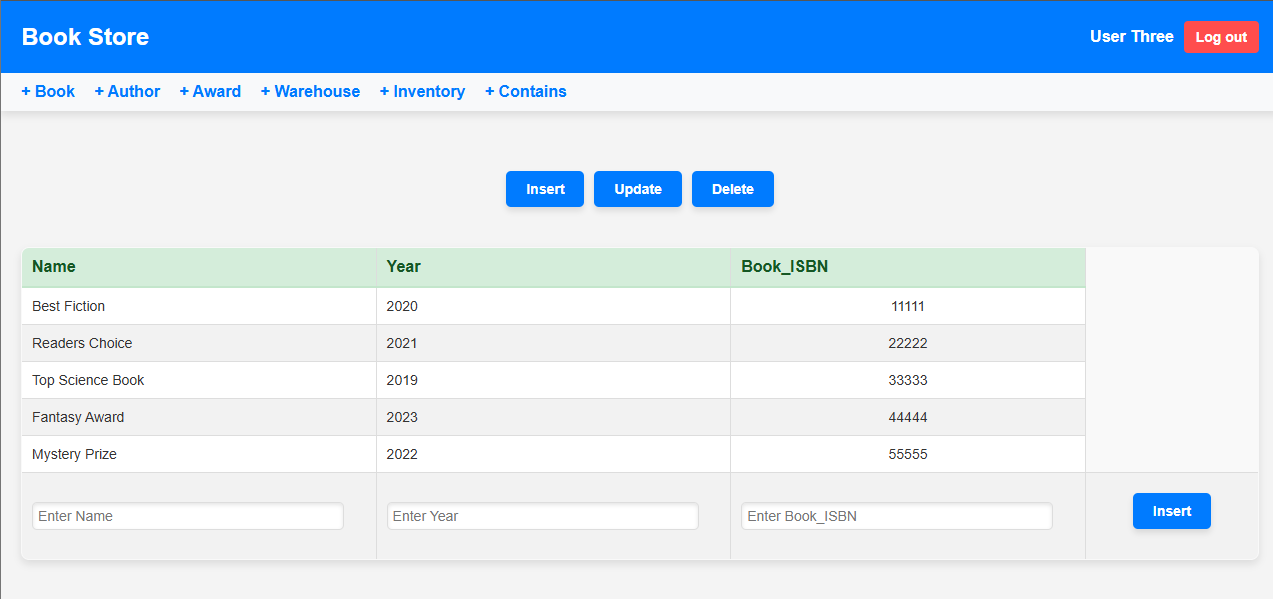
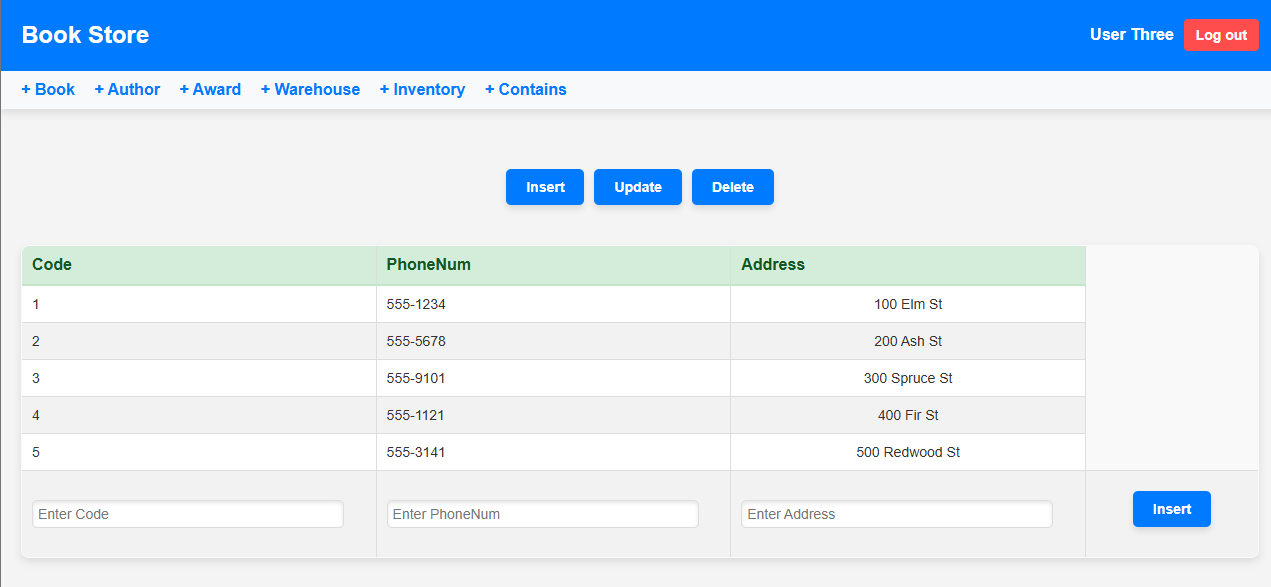
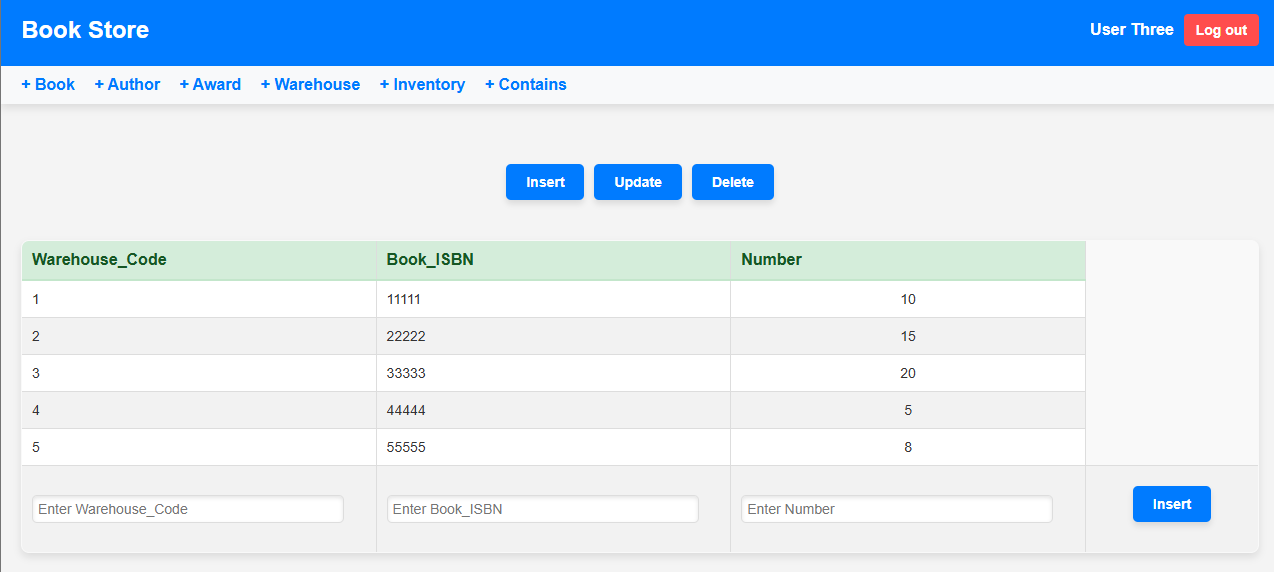
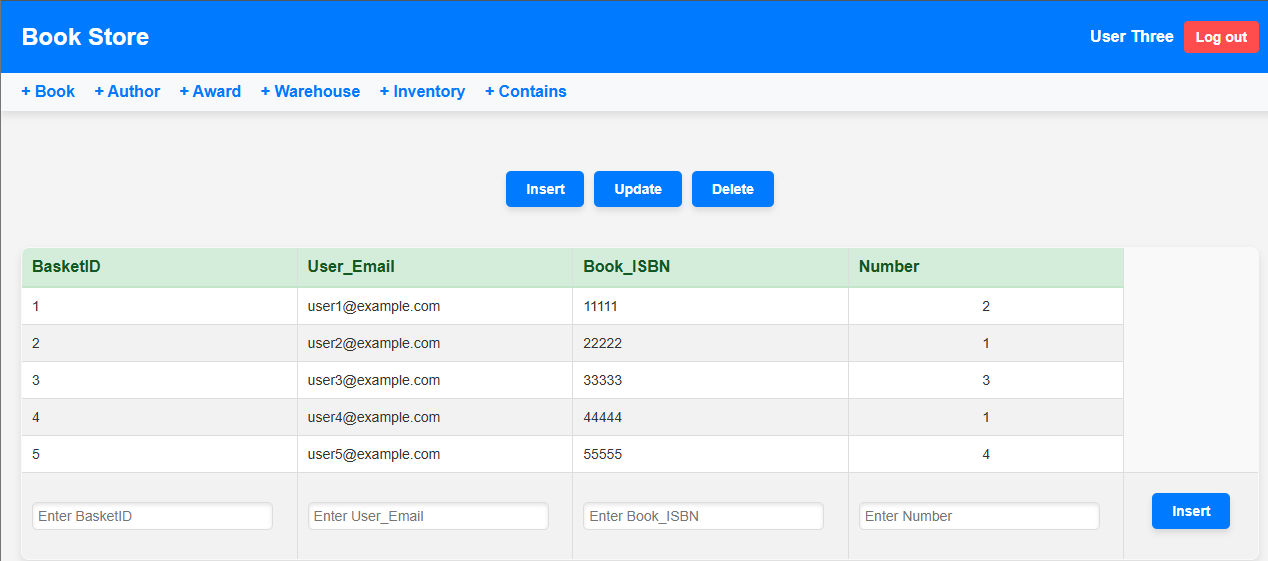
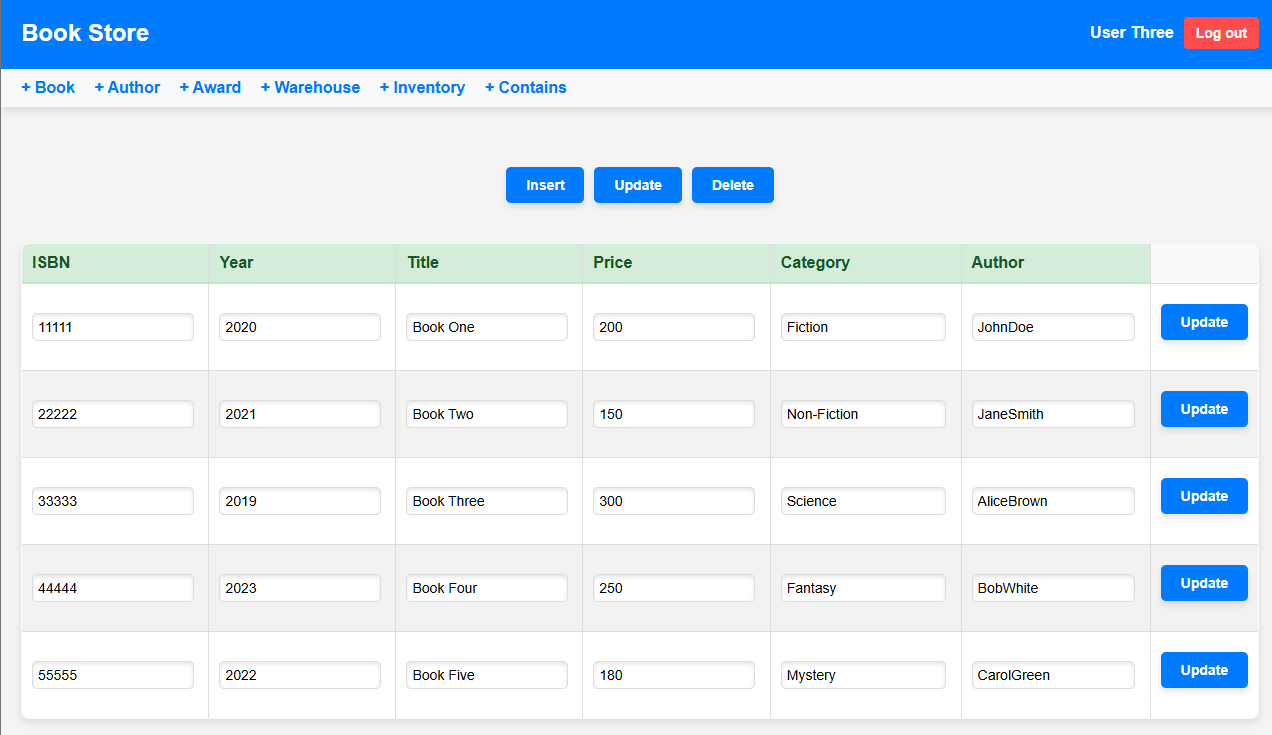
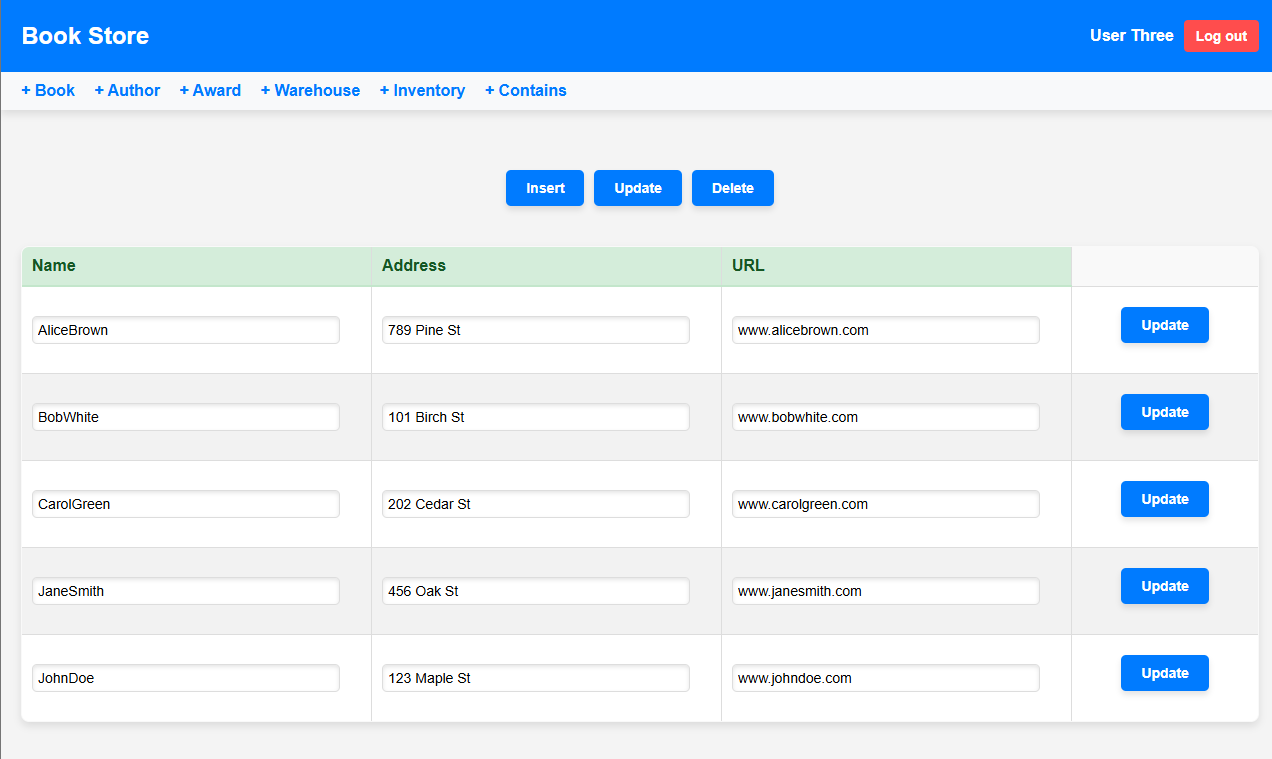
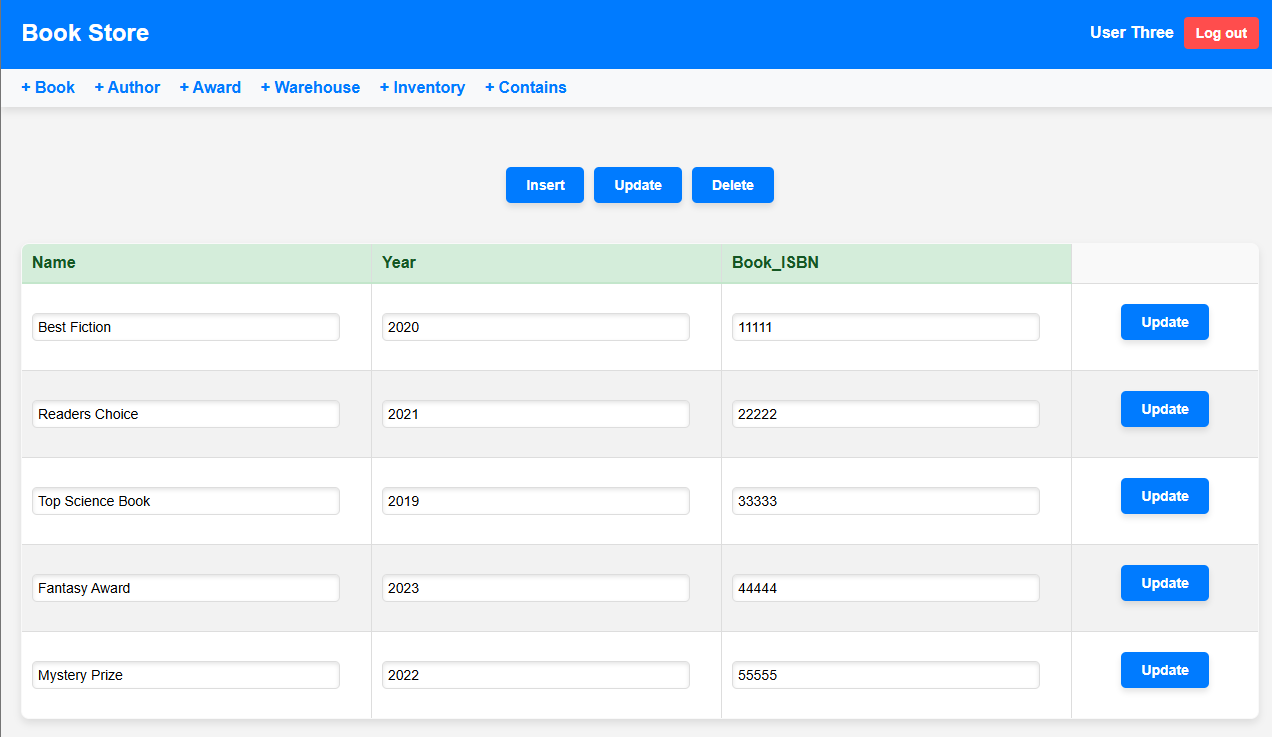
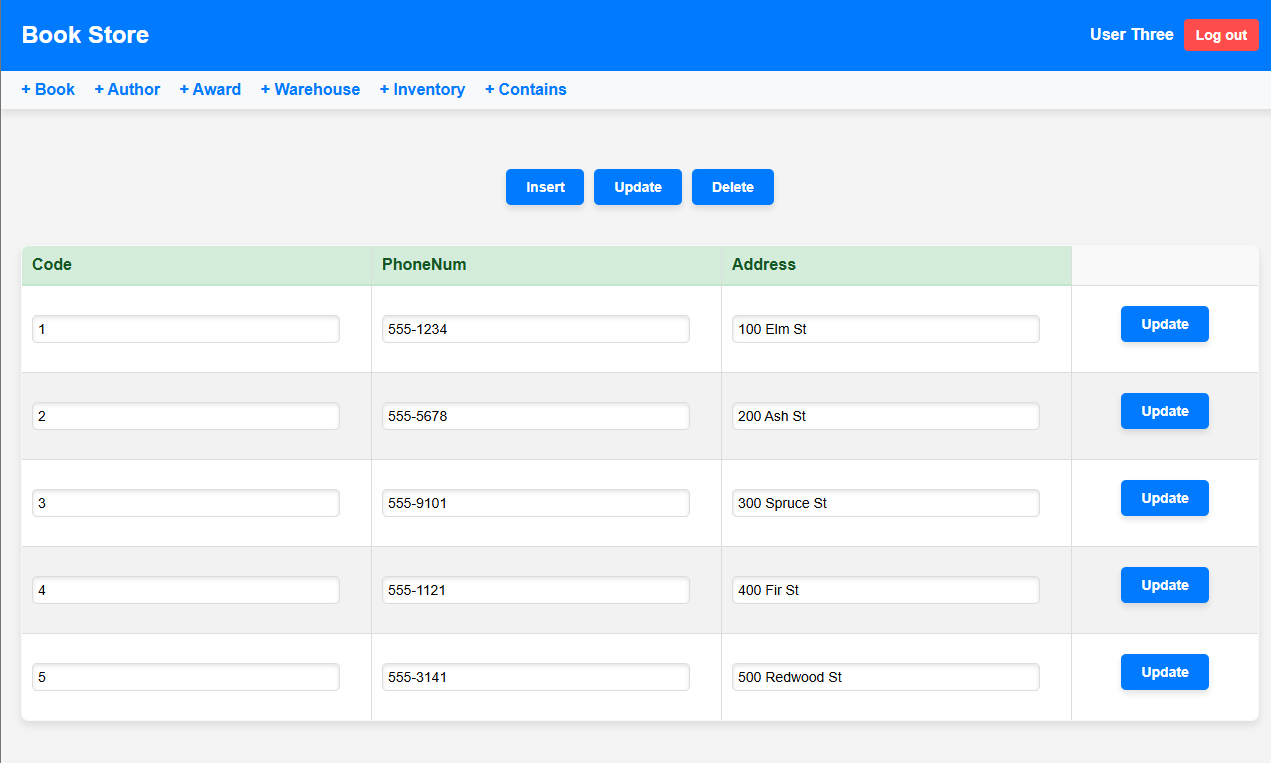
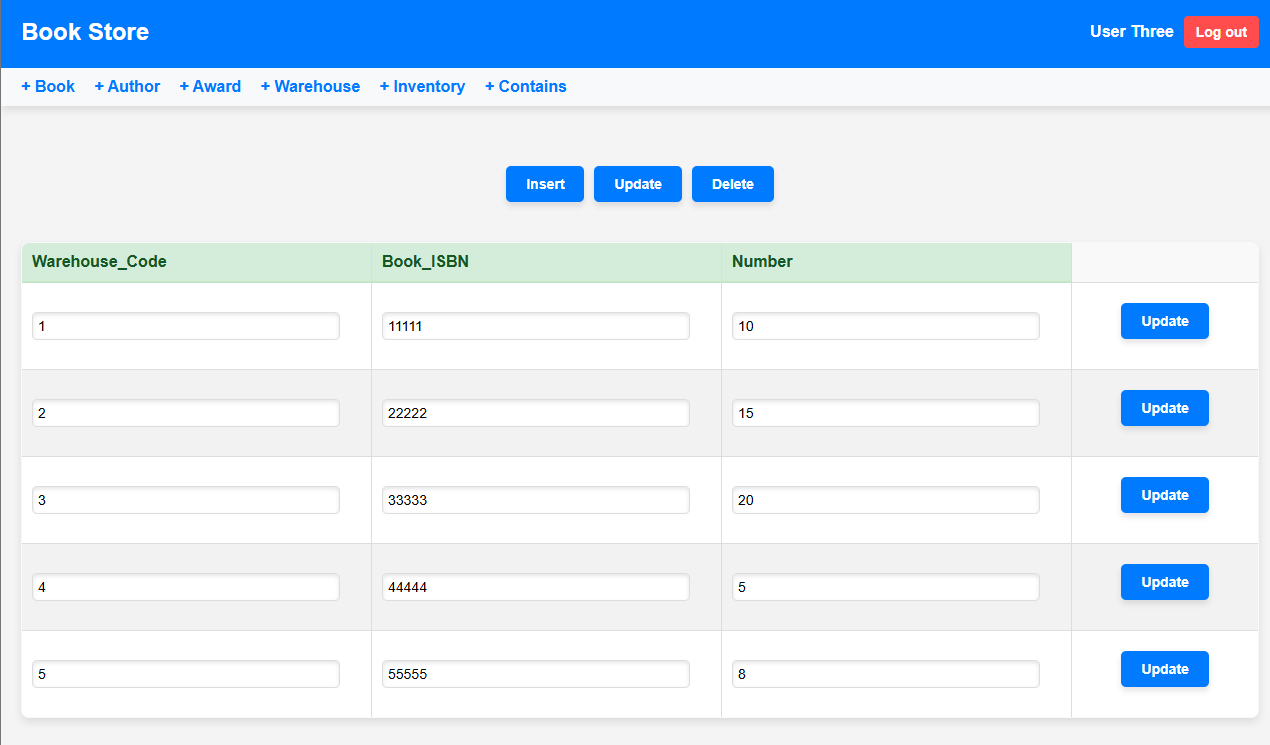
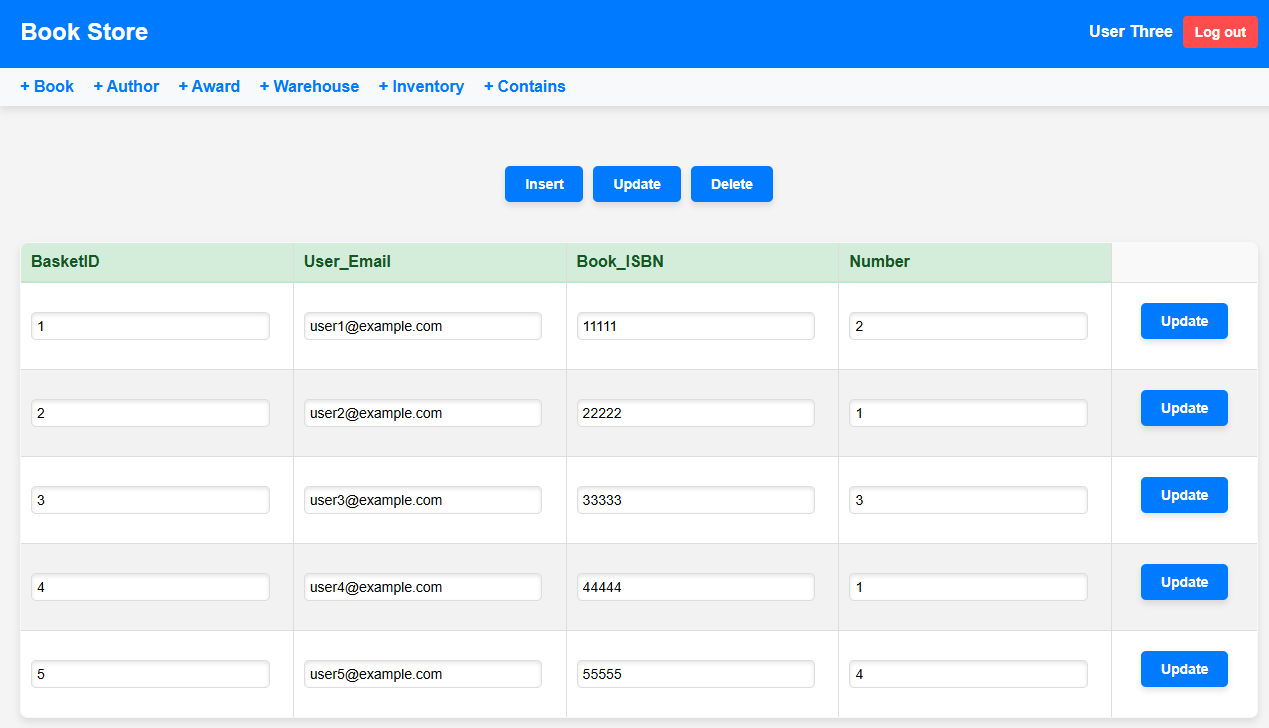
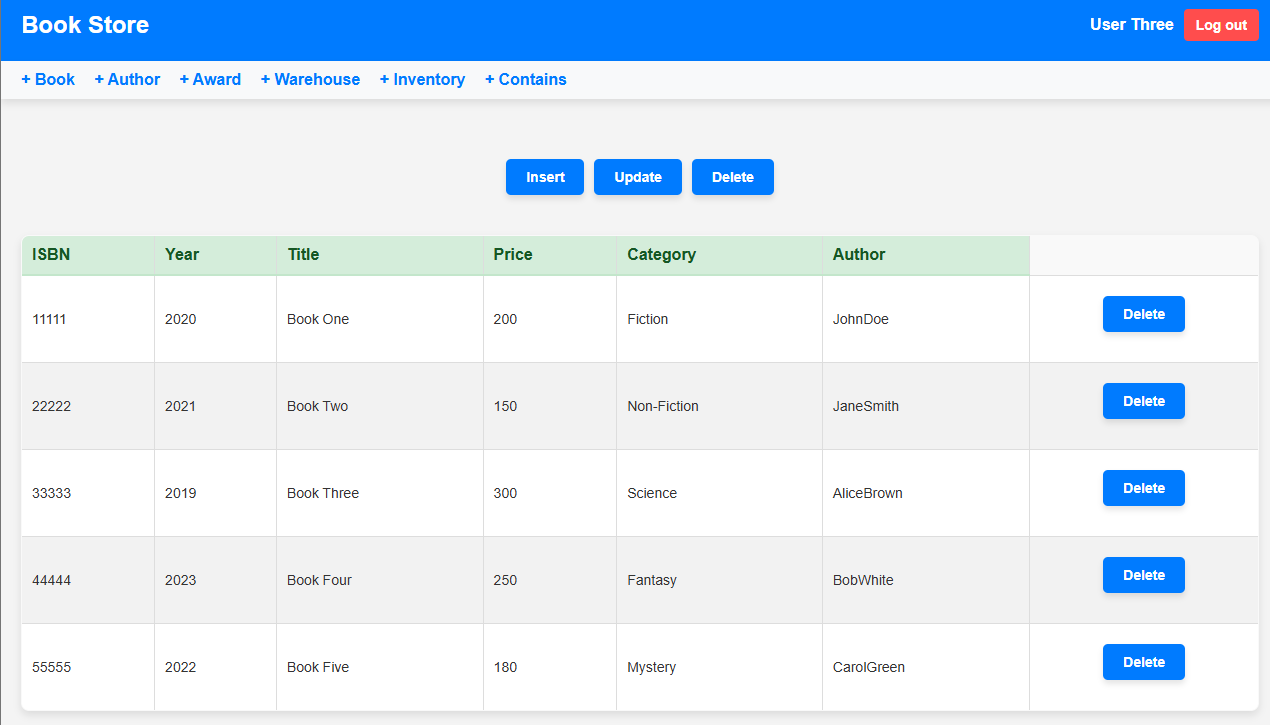
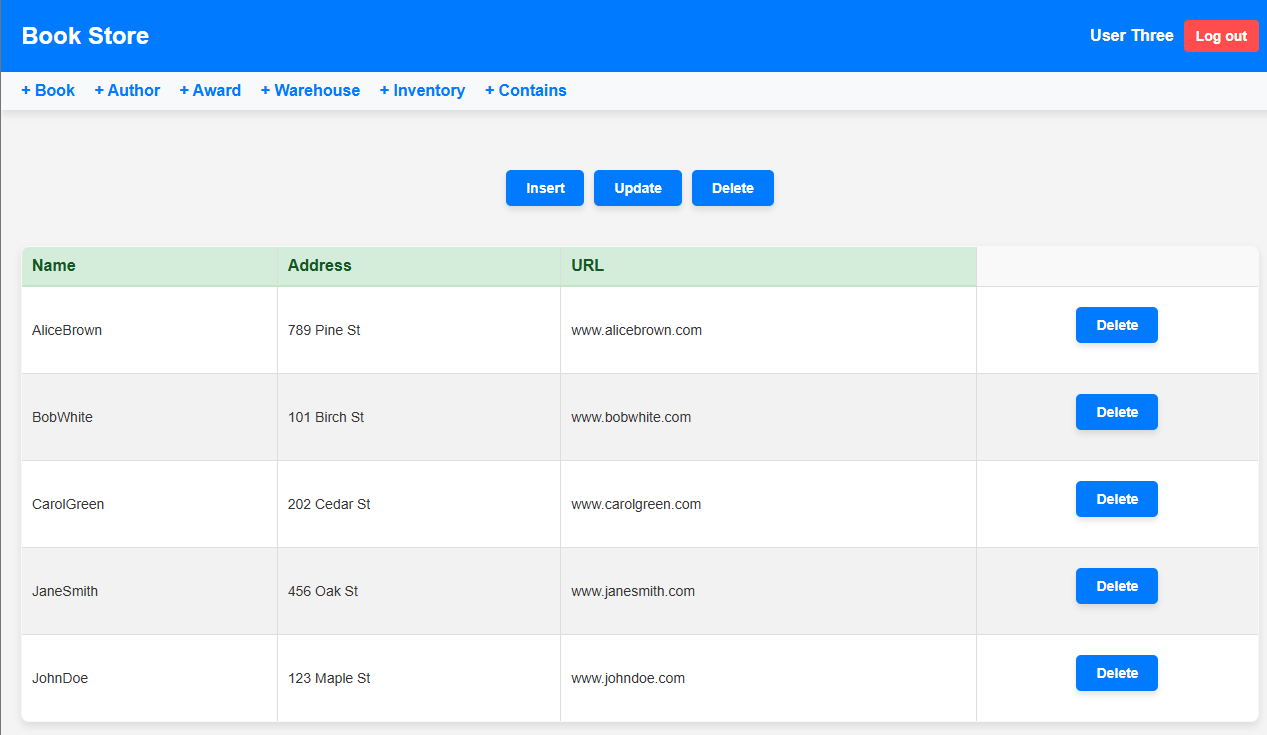
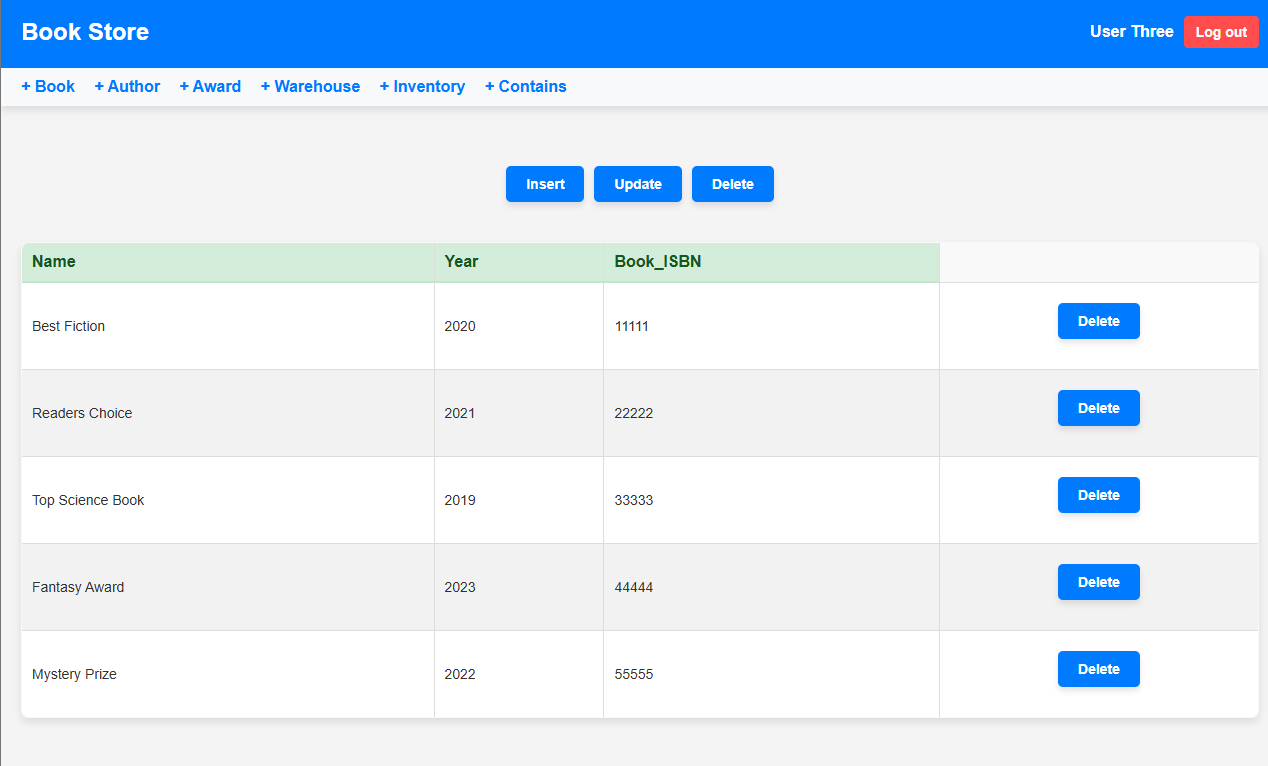
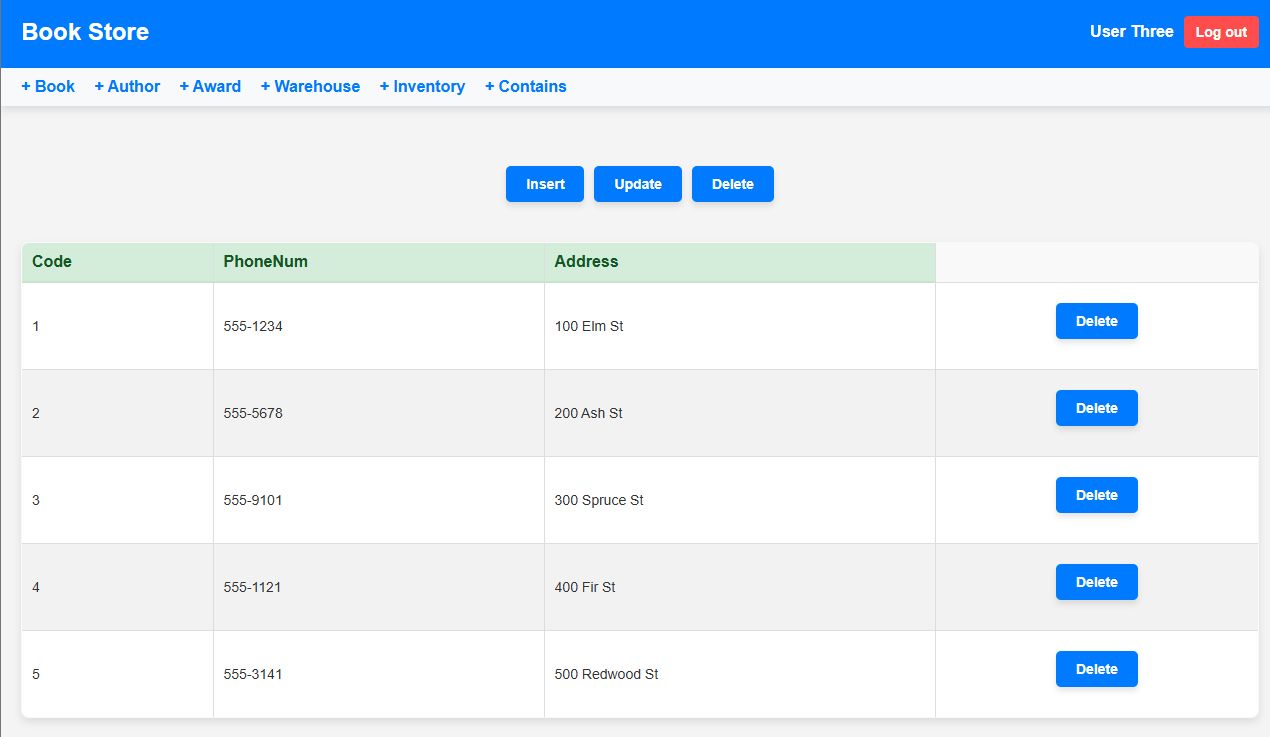
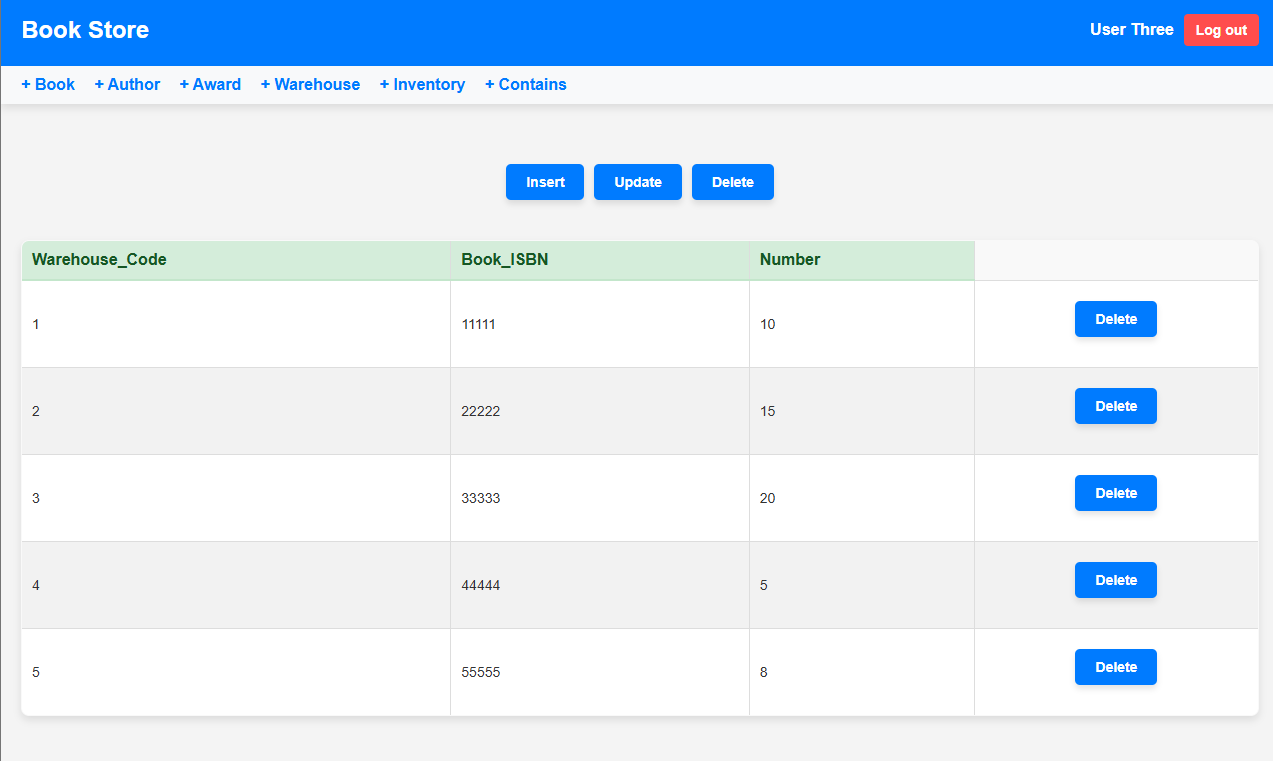
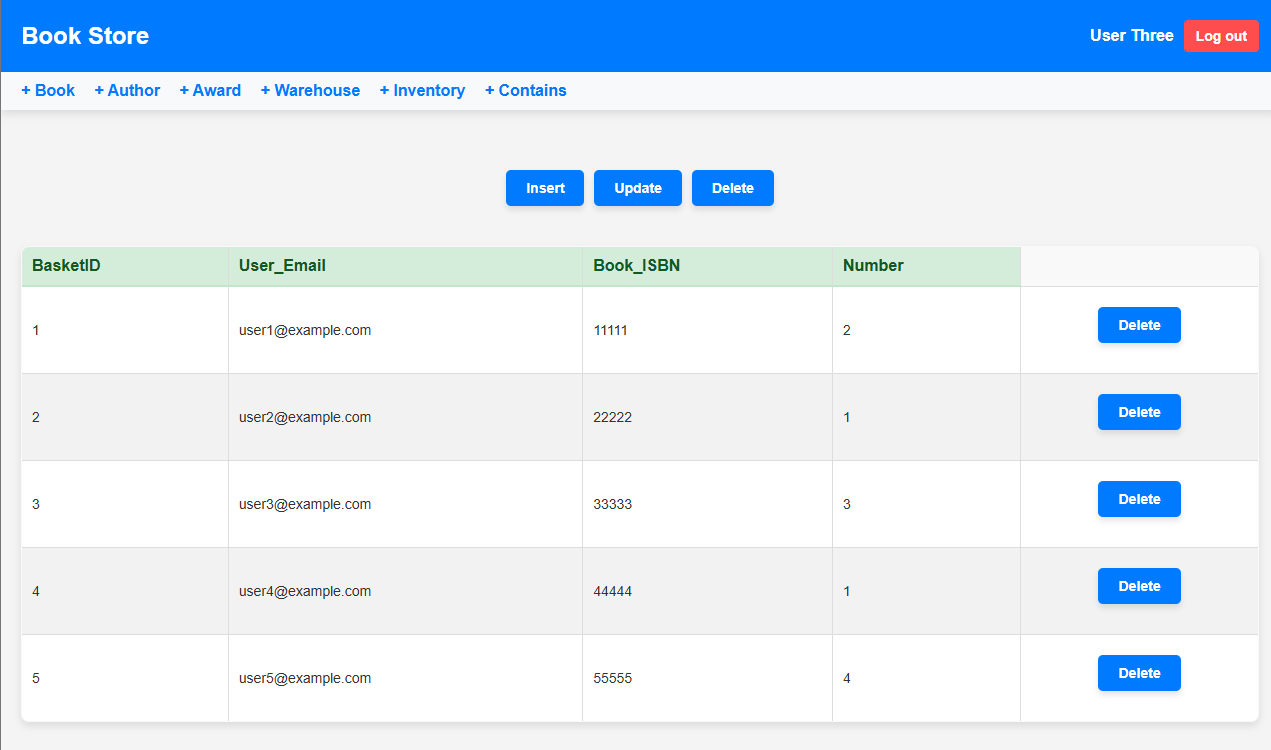
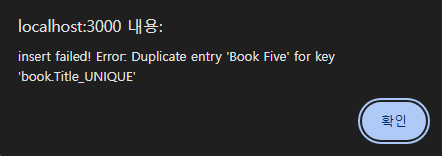
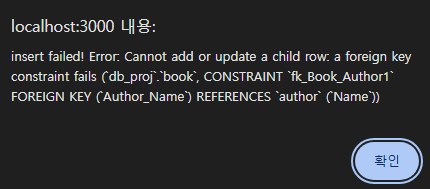
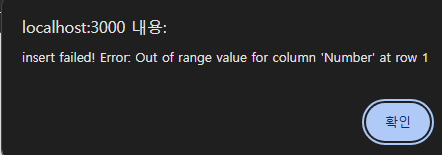
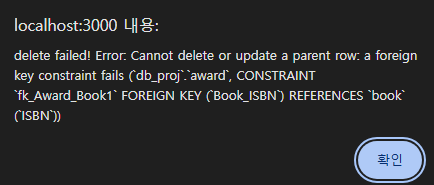
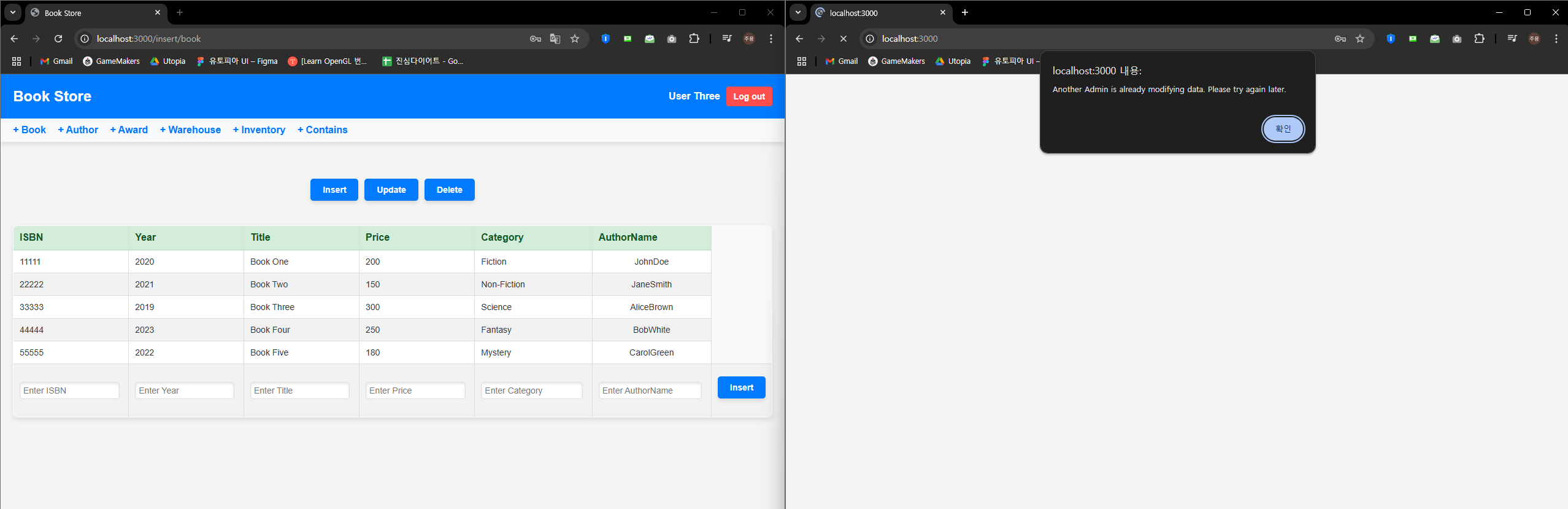
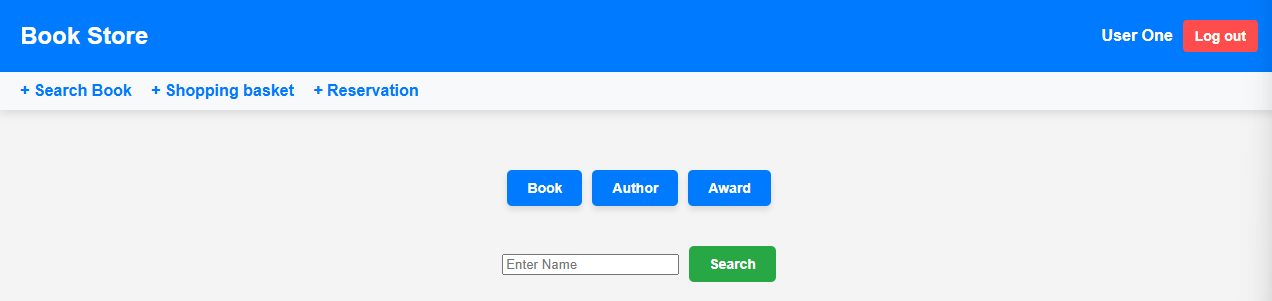
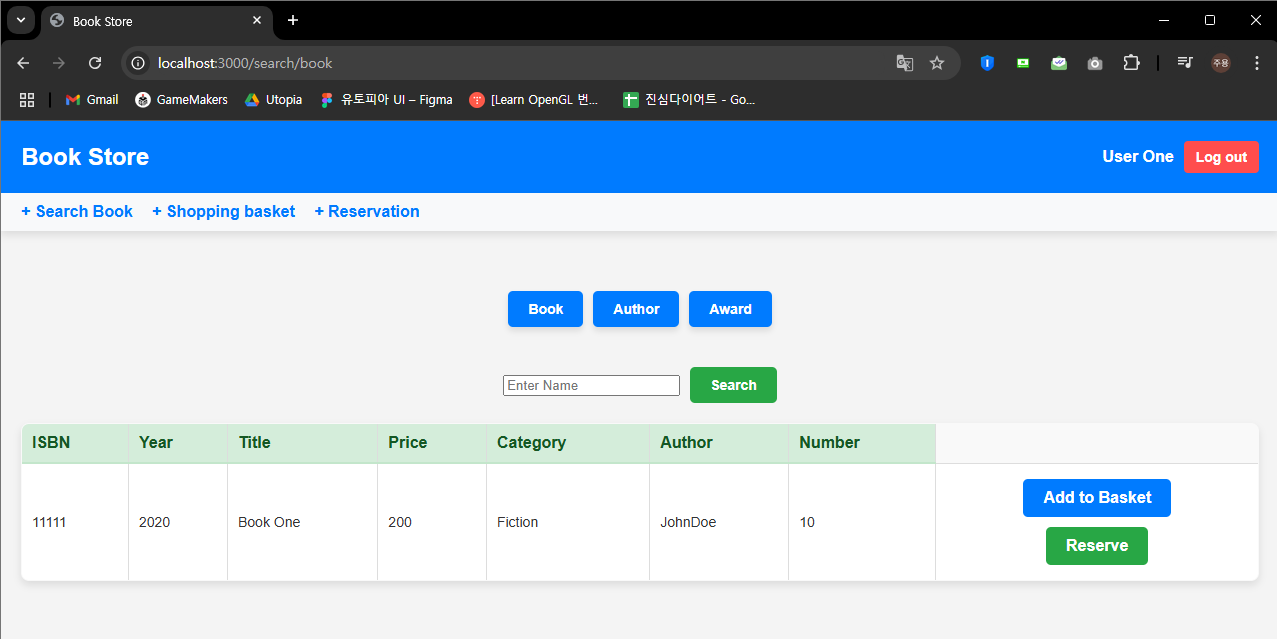
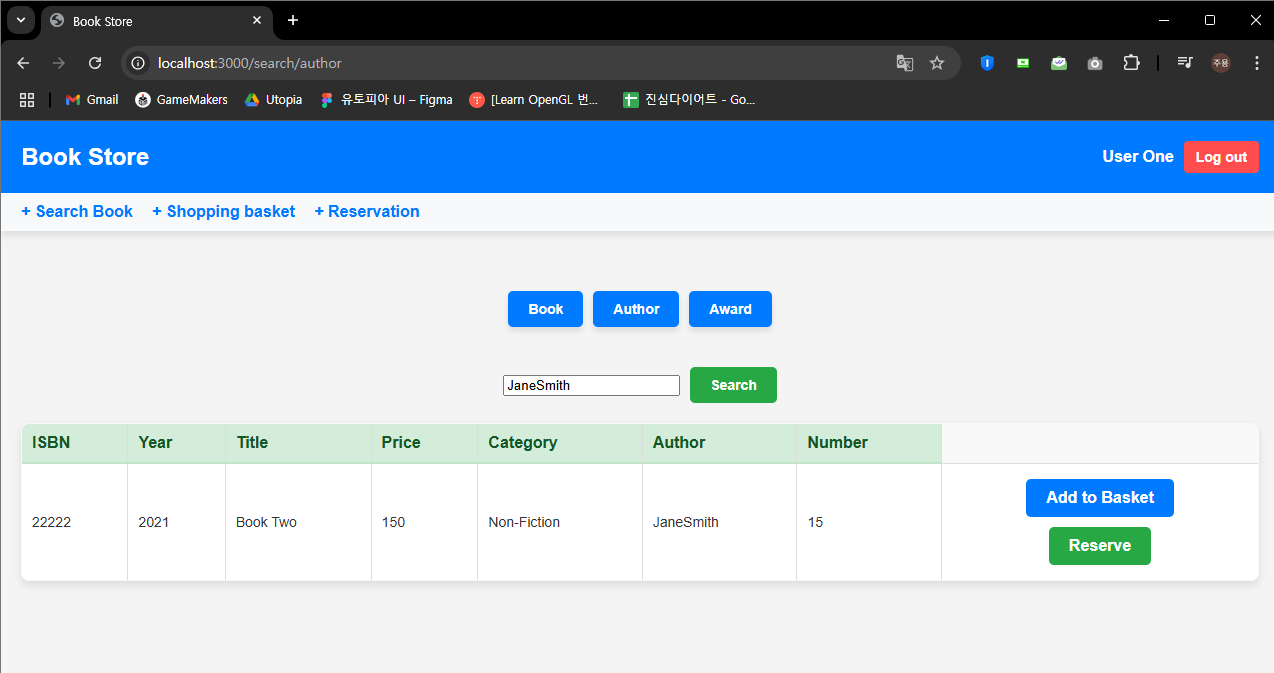
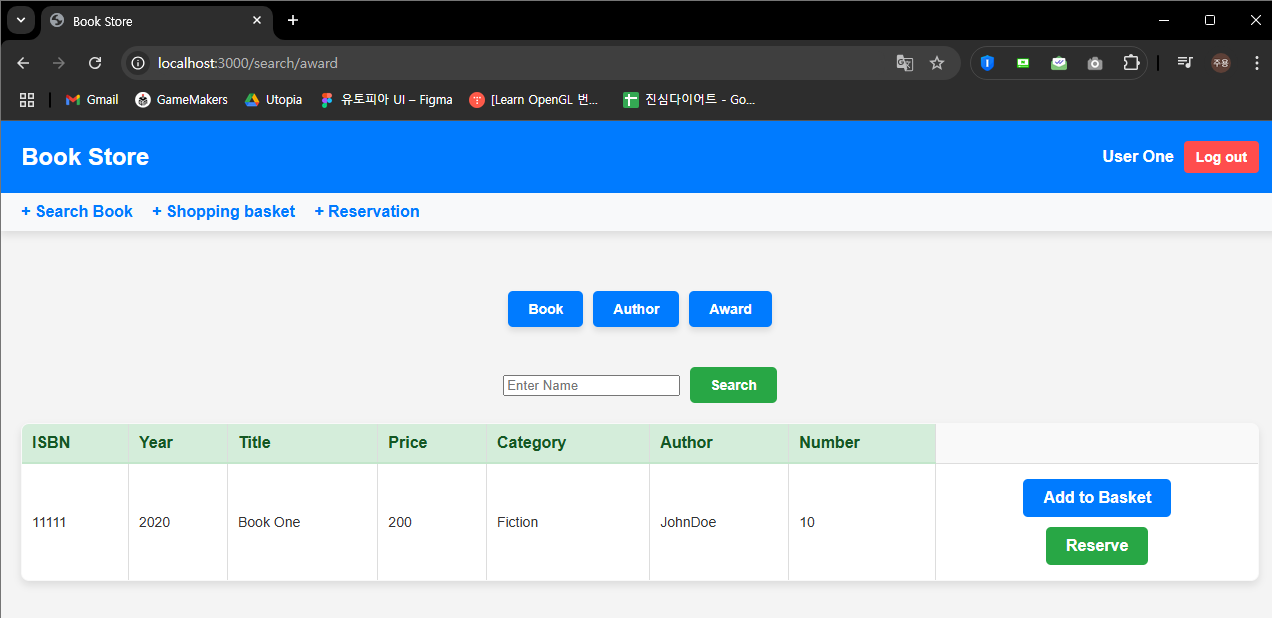
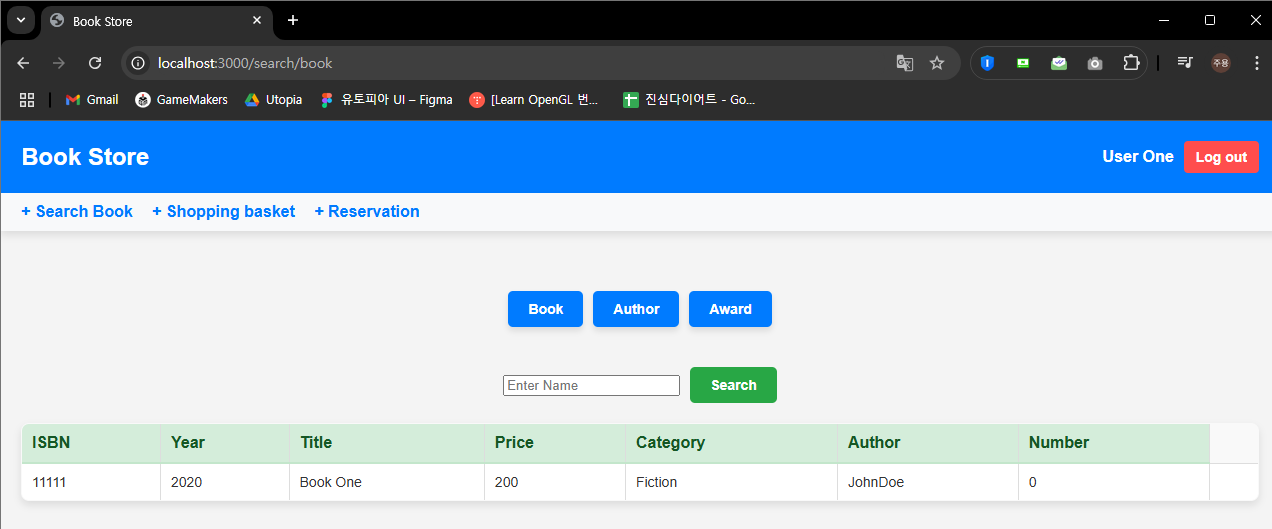
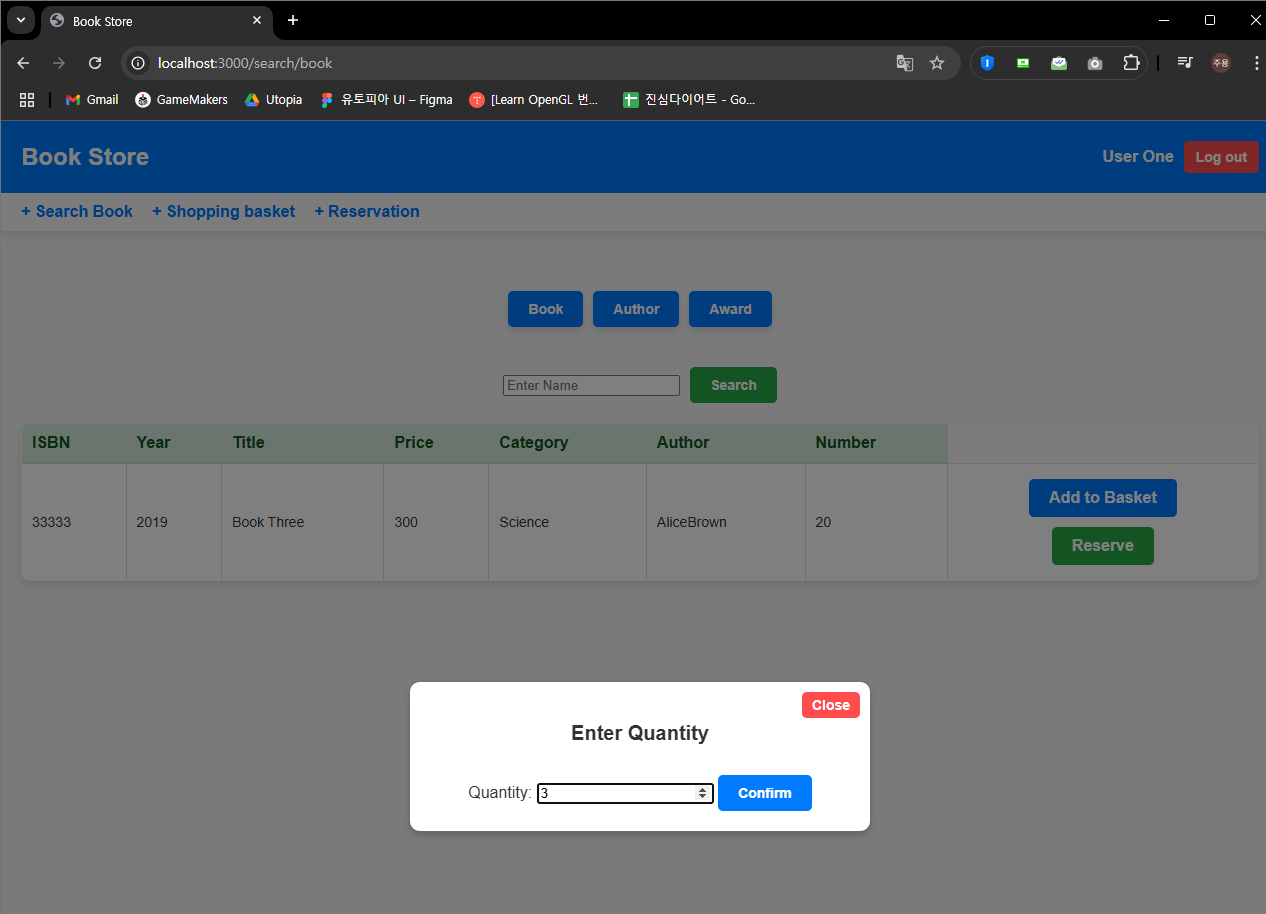
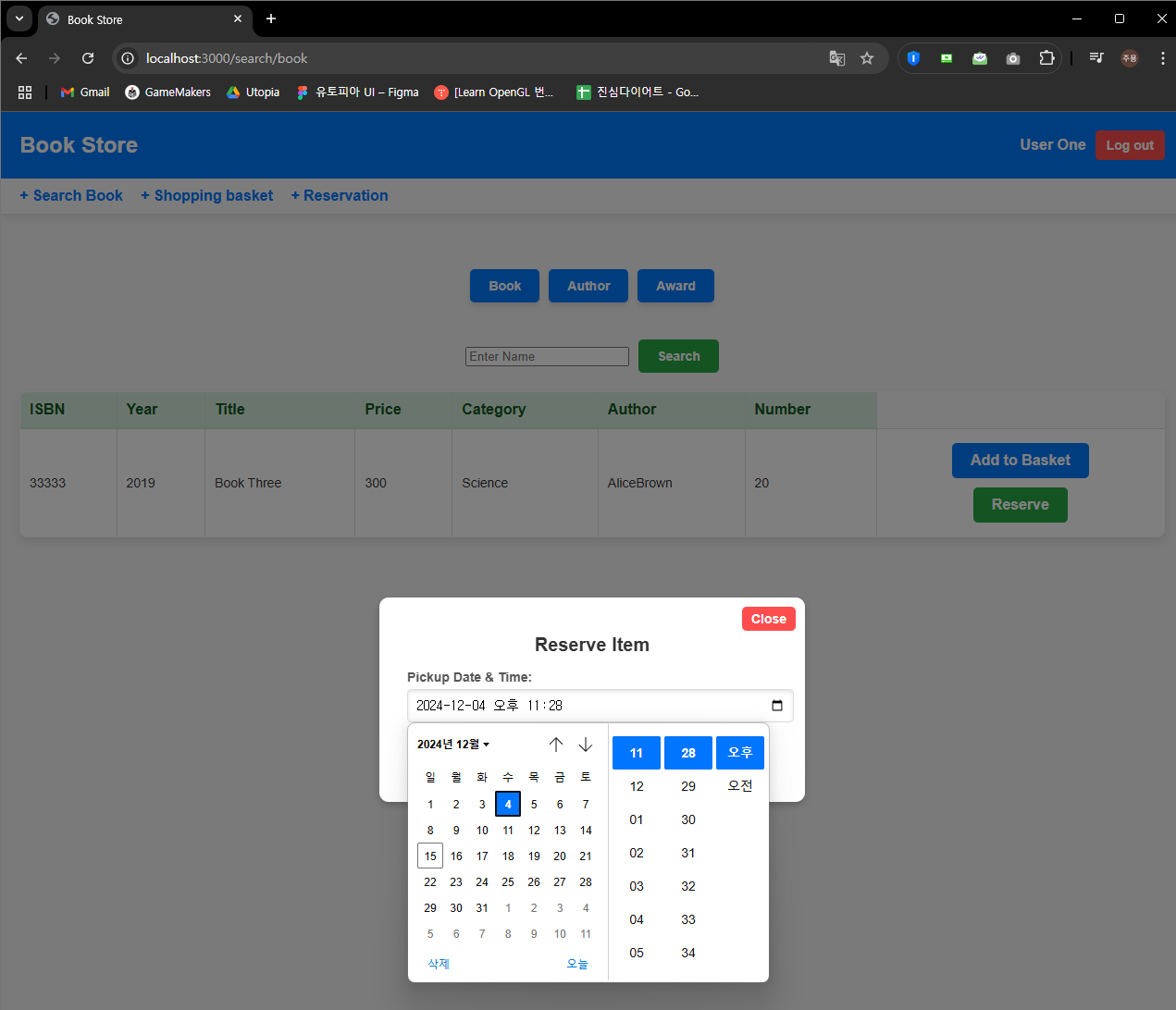
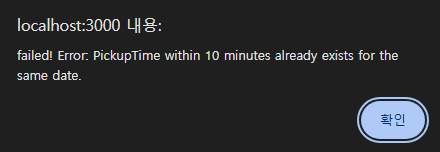
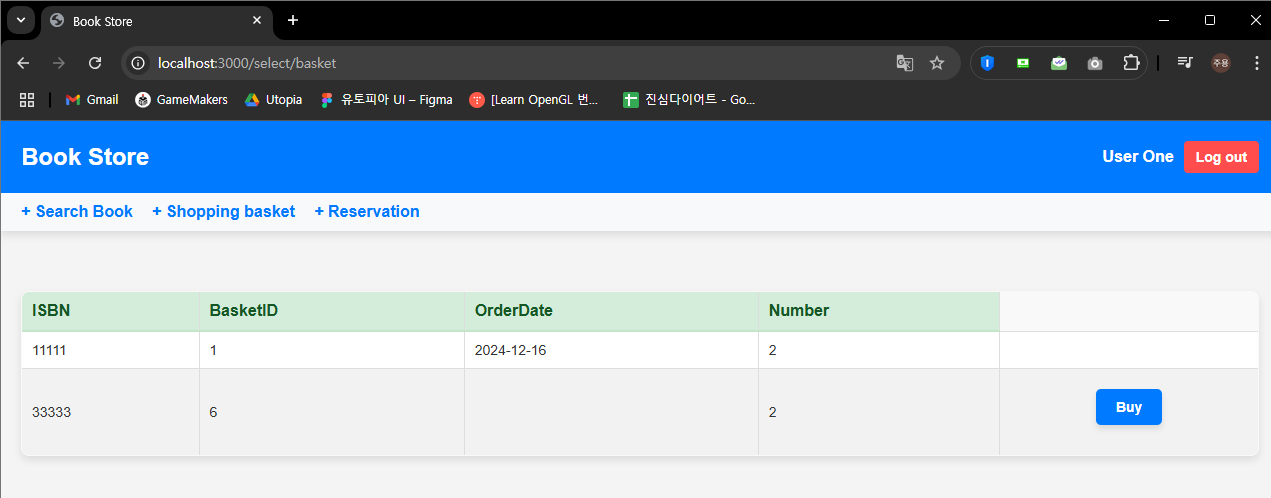
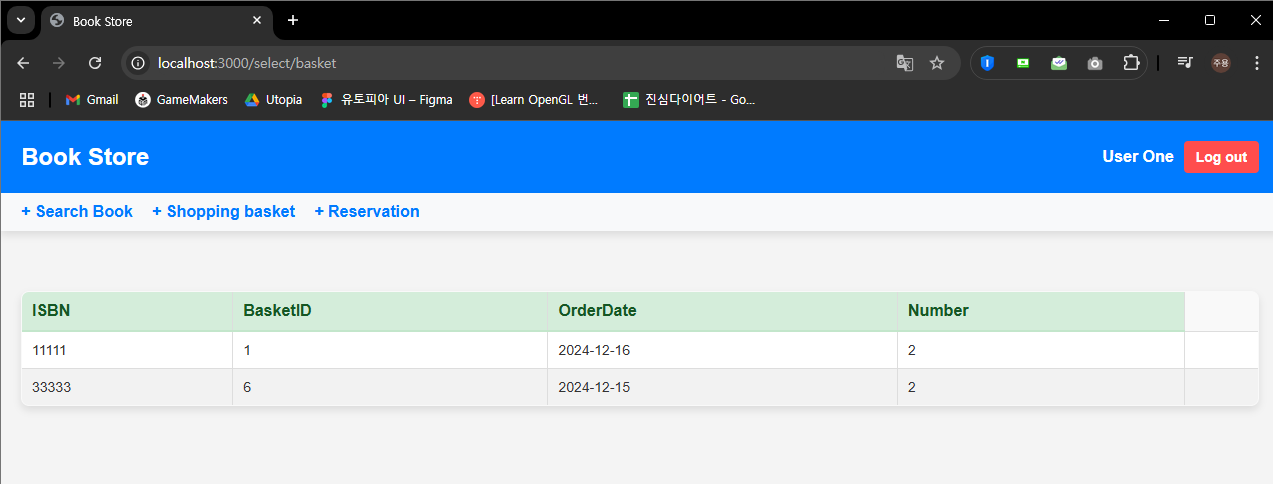
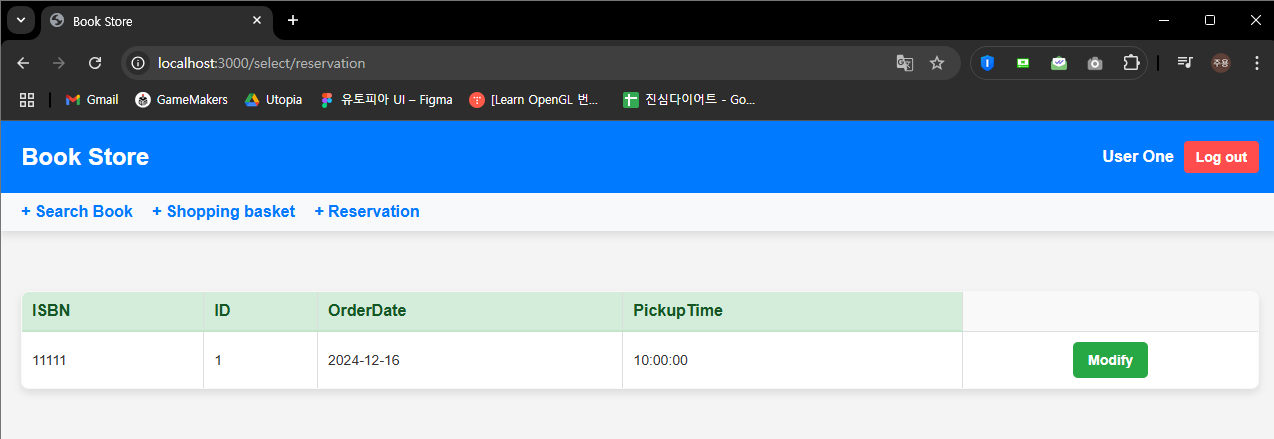
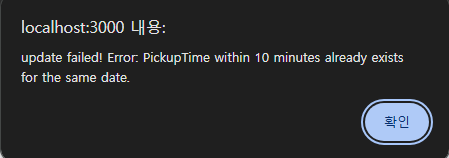
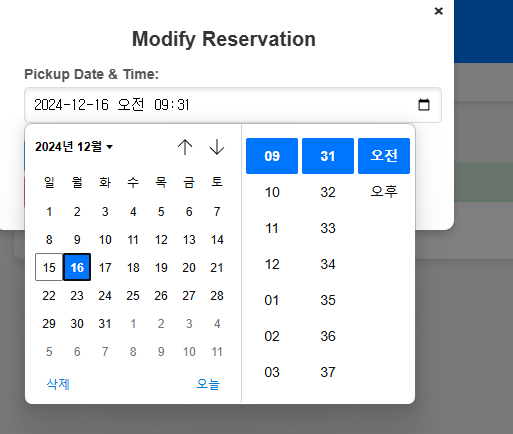
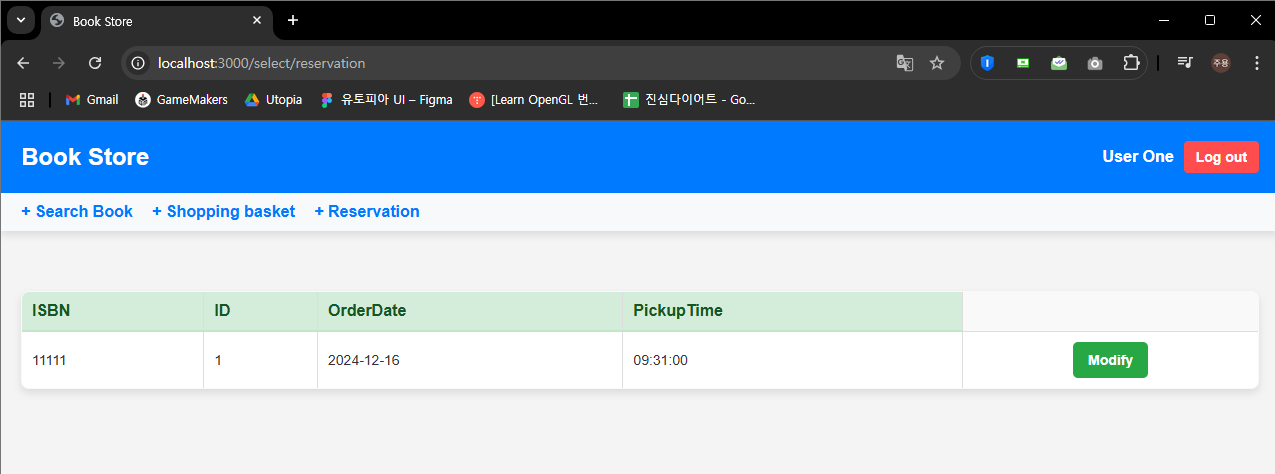
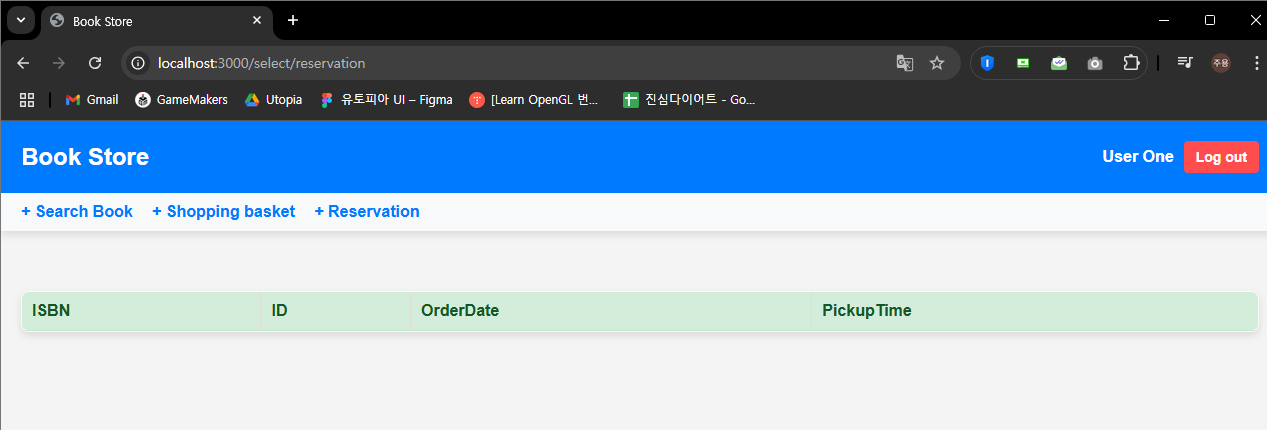
**  
**

* Search 페이지의 경우에는 맨 위의 페이지를 관리하는 것은 관리자 페이지와 동일하고 다른 점은 각 행의 맨 끝에 장바구니에 담기와 예약하기 버튼을 추가한 것이다. 이때, 장바구니 담기의 경우 수량을 적을 수 있는 Input을 추가하고 Confirm 버튼을 누를 시 Insert가 되도록 하였다. 예약의 경우 datetime-local을 통하여 pickupDateTime을 입력받고 Reserve를 통해 Insert가 되도록하였다. 밑의 OpenPopup관련 함수들은 수량이나 픽업 날짜, 시간 데이터를 Inset할 때 필요한 책의 ISBN을 팝업이 열렸을 때 저장해놓기 위한 함수이다.
* Customer 페이지(Select.js)

****

* Select.js의 경우 Shopping Basket과 Reservation의 리스트를 조회하고 장바구니의 경우 구매를 요청하고 예약의 경우 픽업 날짜, 시간 변경을 위한 페이지이다. 기본적으로 render를 위한 로직은 다른 페이지들과 거의 동일하고 post 함수를 살펴보면 예약의 경우에는 받은 DateTime을 Split하여 쿼리를 요청하는데 이때, User Email이 필요하기 때문에 쿠키를 생성할 때 넣어둔 user.id를 활용한다. 그리고 Update 쿼리를 요청하는데 과제의 요구사항이었던 픽업시간 10분내의 요청도 이때 Exception Result를 통해 출력된다. Insert 도 마찬가지이다. 그리고 Reservation Cancel의 경우 Delete 쿼리를 요청하는 점만 다르다.
* 장바구니 구매의 경우 우선 오늘 구매한 것이기 때문에 오늘 날짜를 js에서 받아오는 로직을 작성하였고 구매했다는 것은 OrderDate를 Update 해주면 되는 것이기 때문에 OrderDate를 현재 날짜로 Update하는 쿼리를 요청한다.
* Update Basket & Reservation / Delete Reservation  
    
    
  Update와 Delete 쿼리 역시 앞의 관리자 페이지에서 봤던 Update와 Delete 쿼리와 비슷하게 작성하였다.
* Reservation 페이지  
    
  Reservation의 경우 다른 리스트 들과 비슷한 render 후 해당 행의 정보를 담고 있는 button을 만든다. 이 버튼을 누르면 밑의 Modal이 나오게 된다. 그리고 Update 버튼을 누르게 되면 픽업시간이 Update 되고 Cancel 버튼을 누르게 되면 예약이 취소 된다. 그리고 밑의 script를 통해 Modal을 컨트롤하는 코드를 집어 넣었고 이때, Post 요청을 통해 전달할 데이터를 관리하였다.
* Basket 페이지  
    
  Basket 페이지는 비교적 간단하다. 관리자가 Delete 하던 페이지와 거의 동일하고 버튼에 정보를 담기 위해 Input hidden을 활용하였다. 그리고 만약 OrderDate가 NULL이 아니라면 if this.OrderDate가 true가 되면서 구매버튼이 없고 NULL이라면 false가 되어서 버튼이 출력된다.

1. 실행 화면

* 로그인 화면  
  
* 로그인 실패 시  
  
* Admin 로그인 시  
  
* 다른 Table로 이동시  
    
    
    
    
  
* Update 페이지  
    
    
    
    
    
  
* Delete 페이지  
    
    
    
    
    
  
* Insert, Update, Delete 시 에러가 발생했을 때(중복, 없는 FK, Unsigned 등등)  
   중복된 Data  
   존재하지 않는 Data FK  
  Unsigned 변수에 음수 할당  
  아직 FK가 존재하는 데이터 삭제
* 다른 관리자가 로그인 되어 있는 경우 관리자로 로그인 시도 시  
  
* Customer 로그인 시  
  
* Book을 이용한 검색  
  
* Author를 이용한 검색  
  
* Award를 이용한 검색  
  
* 창고의 총 수량이 0일 때  
  
* 장바구니 담기 버튼 클릭 시  
  
* 예약 버튼 클릭 시  
  
* 10분 이내의 예약이 존재하는 예약을 진행할 경우  
  
* 장바구니 리스트  
  
* 구매버튼 클릭 시  
  
* 예약 수정 페이지  
  
* 10분 이내의 예약이 존재하는 예약으로 수정할 경우  
  
* 예약을 수정한 경우  
    
  
* 예약을 취소한 경우  
  

1. 결론

* 이번 Term Project는 Book Store라는 웹페이지를 제작하고 이를 위한 DB를 설계하는 과제였다. 이번 과제를 하고 느낀 것은 SQL을 관리할 때 Error에 대한 Control이 매우 중요하고 이런 Error 보다도 버그 성의 데이터 이상이 발생하지 않도록 요구사항에 맞춰 Constraint를 설정하는 것이 매우 중요하다는 것이다. 이번 Project를 설계할 때 가장 먼저 고려한 것이 페이지의 통일성이다. 페이지를 최대한 비슷하게 만들다 보니 굉장히 코드를 재활용할 수 있는 부분이 많았다. 그리고 이번 수업이 데이터 베이스 설계다 보니 예외처리나 에러 관리를 최대한 Mysql에서 처리하고 js에서는 에러 메세지만 띄워주는 형식으로 하니 개발도 편하고 훨씬 직관적인 구현이 가능했다. 이 부분이 이번 프로젝트를 구현하면서 가장 뿌듯했던 점이다. 그리고 mySQL에서 할 수 없는 예외는 최대한 프론트에서 버튼을 제공하지 않는 등의 Control을 통해서 해주어서 쿼리도 최대한 간단해지도록 노력하였다. 트랜잭션의 경우에도 쿼리가 단순해서 오류가 거의 발생하지 않았고 Index의 경우 직접 Index를 설정할 수도 있지만 진짜 Index가 필요한 부분들은 이미 PK, FK, Unique 등을 통해 Index로 설정되어 이런 옵션도 많이 활용해보면 좋을 것 같다. 그리고 Constraint 대신 Trigger를 사용해 제약조건을 생성해보았는데, 훨씬 직관적이고 구현이 편리했던 것 같다. 제출물에 참고로 Index 생성은 Create.sql에서 Unique 변수들에 대해 모두 작성되어 있어서 첨부하지 않았고 Insert의 경우 각 테이블 당 5개씩 테스트를 위한 Insert.sql과 Index 최적화 측정에 사용한 books\_500K\_data.sql이 있다. 그리고 트랙잭션과 log에 대한 setting의 경우 유저가 로그인할 때, 쿼리를 요청하였다. 그리고 login.hbs는 비어있는데 이는 layout.hbs의 마지막에 구현되어 있다. 이는 모든 페이지에서 로그인과 아웃을 출력하기 위함이다.