Unit Testing

Test ID	USER STORY	FEATURE	TEST DESCRIPTION	PRIORITY	TEST STEPS	Expected Results	Actual Results	Pass/Fail - Comments
1	As an Admin user, I want to login to the system, so that I can perform admin features	Admin login	Admin login, to verify if admin is able to login using admin credentials	Critical	1. Admin opens login page 2. Admin inputs given credentials 3. Admin is redirected to admin page	Admin successfully redirected to admin page after inputting the right credentials	Admin successfully redirected to admin page after inputting the right credentials	Pass
2	As an Admin user, I want to login to the system, so that I can perform admin features	Admin login	Admin login, to verify if admin/user inputs the wrong credentials, denies access	Critical	1. Admin opens login page 2. Admin inputs wrong credentials 3. Admin is prompted with "Invalid Username" and "Invalid Password" 4. Redirected to admin login page to try again	"Invalid Username" "Invalid Password"	"Invalid Username" "Invalid Password"	Pass
3	As a User, I want to register for an account, so that I can buy, sell and borrow books.	Register account	User registration, to verify if user able to register for an account to use website features	Critical	1. User opens registrations page 2. User inputs credentials and details 3. Submits details and prompted registered successfully 4. Redirected to web page.	User inputs credentials and details, creates account successfully and then gets redirected to login page	User inputs credentials and details, creates account successfully and then gets redirected to login page	Pass
4	As a User, I want to register for an account, so that I can buy, sell and borrow books.	Register account	User registration, to verify if user input details that matches database, will prompt found in database	Critical	1. User opens registrations page 2. User inputs credentials and details 3. Submits details and	"Username '[Insert inputted username here]' already exists"	"Username '[Insert inputted username here]' already exists"	Pass

					prompted username already exists. 4. User is redirected to login page and is prompted with "Username '[Insert inputted username here]' already exists" in order to try again.			
5	As a User, I want to login to access website, so that I can sell/view/buy books	User login	User login, to verify if user is able to login to an account that they already created	Critical	1.User opens login page 2. User inputs given credentials 3. User successfully login and is redirected to home webpage	User successfully redirected to homepage after inputting the right credentials	User successfully redirected to homepage after inputting the right credentials	Pass
6	As a User, I want to login to access website, so that I can sell/view/buy books	User login	User login, to verify if user inputs the wrong credentials, access denied to logged in webpage	Critical	1. User opens login page 2. User inputs wrong credentials 3. User is prompted with "Invalid Username" and "Invalid Password" 4. Redirected to user login page to try again	“Invalid Username” “Invalid Password”	“Invalid Username” “Invalid Password”	Pass
7	As a Shop Owner, I want to edit a book's detail, so that I can keep that book's detail up to date	Update details	Shop owner post page, to verify if shop owner is able to update book details that has been posted	Low	1. Shop owner logs in to webpage 2. Shop owner click on on "Post page"	User input book details and is able to update book details	User input book details and is able to update book details	Pass

					3. Shop owner clicks on post he wants to edit 4. Shop owners submit new book details.			
8	As a Shop Owner, I want to sell books, so that I can get profit	Post books	Post books, to verify if shop owner is able to publish the book they want to sell on the webpage	Med	1.User logs into webpage 2. User clicks to "[Inputted user's display name] page" 3. User clicks change password 4. User inputs old password, and new password and submits 5. A message is prompted to notify that password has changed successfully	Shop owner is able to post their products with the right details	Shop owner is able to post their products with the right details	Pass
9	As a User, I want to change my password, so that I can have my new password	Change password	User page, to verify if customer is able to change password	Low	1.User logs into webpage 2. User clicks to "User profile page" 3. User clicks change password 4. User inputs old password, and new password and submits 5. A message is prompted to notify that password has changed successfully	User is able to change their password when everything is inputted correctly	User is able to change their password when everything is inputted correctly	Pass
10	As a User, I want to change my password, so that I can	Change password	User page, to verify if customer is not able to change	Low	1.User logs into webpage	"Invalid Password"	"Invalid Password"	Pass

	have my new password		password if input wrong old password		2. User clicks to "User profile page" 3. User clicks change password 4. User inputs old password, and new password and submits 5. A message "Invalid Password" is prompted to notify that old password is not correct 6. Redirects to try again			
11	As a User, I want to edit my profile detail, so that I can provide sufficient details about myself	Edit profile	User profile, to verify if user is able to edit their profile	Low	1.User logs into webpage 2. User clicks to "User profile page" 3. User clicks "edit details" 4. User inputs new details and submit 5. A message is prompted to notify that it has changed successfully	User is able to change their profile details	User is able to change their profile details	Pass
12	As a Customer, I want to search books, so that I can get information about the books	Book search	User book search, to verify if customer is able to search for desired books on the website	Med	1.User logs into webpage 2. User clicks on "Search" 3. User inputs book name of interest 4. Books are displayed for the user.	User is able to search for the book they are looking for with the details of the book being shown	User is able to search for the book they are looking for with the details of the book being shown	Pass

In our assignment code, there is a user validation code in the Backend implemented as unit testing in the form of validating code that checks if the password is at least 6 characters and rejects the inputted password if it isn't, along with rejecting the inputted password that does not matched the

inputted confirm password field. There is also a helper method used to the validate the function for the user changing their password and reject the user's inputted new password if their password is less than 6 characters, their new password doesn't match the confirm password field or that their new inputted password matches any of the old passwords they have used before.

```java
import com.rmit.sept.usermicroservices.model.User;
import com.rmit.sept.usermicroservices.payload.ChangePasswordRequest;
import org.springframework.stereotype.Component;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;

@Component
public class UserValidator implements Validator {

    @Override
    public boolean supports(Class<?> aClass) {
        return User.class.equals(aClass);
    }

    @Override
    public void validate(Object object, Errors errors) {

        User user = (User) object;

        if(user.getPassword().length() <6){
            errors.rejectValue("password","Length", "Password must be at least 6 characters");
        }

        if(!user.getPassword().equals(user.getConfirmPassword())){
            errors.rejectValue("confirmPassword","Match", "Passwords must match");
        }
    }

    public void changePasswordValidate(ChangePasswordRequest request, Errors errors) {
        if (request.getNewPassword().length() < 6) {
            errors.rejectValue("newPassword", "Length", "Password must be at least 6 characters");
        }
        if (!request.getNewPassword().equals(request.getConfirmPassword())) {
            errors.rejectValue("confirmPassword", "Match", "Passwords must match");
        }
        if (request.getNewPassword().equals(request.getPassword())) {
            errors.rejectValue("newPassword", "Match", "New and Old passwords should be different");
        }
    }
}
```

And so, if the login attempt is invalid, then the inputted username and password would also be set to invalid as a result.

```java
public class InvalidLoginResponse {

        private String username;
        private String password;

        public InvalidLoginResponse() {
            this.username = "Invalid Username";
            this.password = "Invalid Password";
        }

        public String getUsername() {
            return username;
        }

        public void setUsername(String username) {
            this.username = username;
        }

        public String getPassword() {
            return password;
        }

        public void setPassword(String password) {
            this.password = password;
        }
    }
```

Additionally, there should be a unit testing code that also throws an Exception called "UserAlreadyExistsException" upon the system finding out that the inputted username already exists within the database.

```java
public class UsernameAlreadyExistsResponse {

    private String username;

    public UsernameAlreadyExistsResponse(String username) {
        this.username = username;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }
}
```

```java
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(HttpStatus.BAD_REQUEST)
public class UsernameAlreadyExistsException extends RuntimeException {

    public UsernameAlreadyExistsException(String message) {
        super(message);
    }
}
```

```java
import com.rmit.sept.usermicroservices.Repositories.UserRepository;
import com.rmit.sept.usermicroservices.exceptions.UsernameAlreadyExistsException;
import com.rmit.sept.usermicroservices.model.User;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private BCryptPasswordEncoder bCryptPasswordEncoder;

    public User saveUser (User newUser){

      /*  newUser.setPassword(bCryptPasswordEncoder.encode(newUser.getPassword()));
        //Username has to be unique (exception)
        // Make sure that password and confirmPassword match
        // We don't persist or show the confirmPassword
        return userRepository.save(newUser);
       */
        try{
            newUser.setPassword(bCryptPasswordEncoder.encode(newUser.getPassword()));
            //Username has to be unique (exception)
            newUser.setUsername(newUser.getUsername());
            // Make sure that password and confirmPassword match
            // We don't persist or show the confirmPassword
            newUser.setConfirmPassword("");
            return userRepository.save(newUser);

        }catch (Exception e){
            throw new UsernameAlreadyExistsException("Username '"+newUser.getUsername()+"' already exists");
        }

    }

    public User getUser(String username)
    {
        return userRepository.getUser(username);
    }
    public void changePassword(String username, String password)
    {
        userRepository.changePassword(username, bCryptPasswordEncoder.encode(password));
    }
}
```

There is also a code in the controller section of the backend that is used to validate the code and determine whether functions such as register, change password and login succeeds or fails based on the authentication functions.

```
@RestController
@CrossOrigin(origins = "http://localhost:3000")
@RequestMapping("/api/users")
public class UserController {

    @Autowired
    private MapValidationErrorService mapValidationErrorService;

    @Autowired
    private CustomUserDetailsService customUserDetailsService;

    @Autowired
    private UserService userService;

    @Autowired
    private UserValidator userValidator;

    @PostMapping("/register")
    public ResponseEntity<?> registerUser(@Valid @RequestBody User user, BindingResult result){
        userValidator.validate(user,result);
        ResponseEntity<?> errorMap = mapValidationErrorService.MapValidationService(result);
        if(errorMap != null)return errorMap;
        User newUser = userService.saveUser(user);
        return  new ResponseEntity<User>(newUser, HttpStatus.CREATED);
    }
    @PostMapping("/changePassword")
    public ResponseEntity<?> changePassword(@Valid @RequestBody ChangePasswordRequest request, BindingResult result)
    {
        userValidator.changePasswordValidate(request,result);
        ResponseEntity<?> errorMap = mapValidationErrorService.MapValidationService(result);
        if(errorMap != null)
        {
            return errorMap;
        }
        Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                    request.getUsername(),
                    request.getPassword()
            )
        );
        userService.changePassword(request.getUsername(), request.getNewPassword());
        return ResponseEntity.ok("Change Password Successfully!");
    }

    @Autowired
    private JwtTokenProvider tokenProvider;
```

```java
@Autowired
private AuthenticationManager authenticationManager;

@PostMapping("/login")
public ResponseEntity<?> authenticateUser(@Valid @RequestBody LoginRequest loginRequest, BindingResult result){
    ResponseEntity<?> errorMap = mapValidationErrorService.MapValidationService(result);
    if(errorMap != null) return errorMap;
    Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                    loginRequest.getUsername(),
                    loginRequest.getPassword()
            )
    );
    SecurityContextHolder.getContext().setAuthentication(authentication);
    String jwt = TOKEN_PREFIX +  tokenProvider.generateToken(authentication);
    User loginUser = userService.getUser(loginRequest.getUsername());
    LoginResponse loginResponse = new LoginResponse();
    loginResponse.setDisplayname(loginUser.getDisplayName());
    loginResponse.setJWTLoginSucessReponse(new JWTLoginSucessReponse(true, jwt));
    return ResponseEntity.ok(loginResponse);
}
```

Similarly, there should be a unit testing code implemented that occurs when a login attempt has blank (or null) username or password fields, which would ensure that login attempts should only work when the username and password aren't blank as intended. There is also a unit testing code implemented in the form of validation that checks if the user's attempt to change their passwords has their username, current password, new password or confirm password fields being blank (or null), and should prevent the change password function from being executed if those fields are blank/null.

```java
import javax.validation.constraints.NotBlank;

public class LoginRequest {

    @NotBlank(message = "Username cannot be blank")
    private String username;
    @NotBlank(message = "Password cannot be blank")
    private String password;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

```java
import javax.validation.constraints.NotBlank;

public class ChangePasswordRequest {

    @NotBlank(message = "Username cannot be blank")
    private String username;
    @NotBlank(message = "Password cannot be blank")
    private String password;
    @NotBlank(message = "Password cannot be blank")
    private String newPassword;
    @NotBlank(message = "Password cannot be blank")
    private String confirmPassword;
    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getNewPassword() {
        return newPassword;
    }

    public void setNewPassword(String newPassword) {
        this.newPassword = newPassword;
    }

    public String getConfirmPassword() {
        return confirmPassword;
    }

    public void setConfirmPassword(String confirmPassword) {
        this.confirmPassword = confirmPassword;
    }
}
```

In the front-end page, all these features/functions are as shown below:

Register page:

Bookeroo

Account Cart (0 items)

Sign Up

Create your Account

Fullname

Please fill in this field.

Display Name

Email Address

Password

Confirm Password

Normal Customer

If you already have an account, please login here! Other types of user beside Normal Customer will require admin to approve!

Submit

Bookeroo

Account Cart (0 items)

Sign Up

Create your Account

Quoc Pham

Display Name

Please fill in this field.

Email Address

Password

Confirm Password

Normal Customer

If you already have an account, please login here! Other types of user beside Normal Customer will require admin to approve!

Submit

Bookeroo

Account Cart (0 items)

Sign Up

Create your Account

Quoc Pham

Quoc123456

Email Address

Please fill in this field.

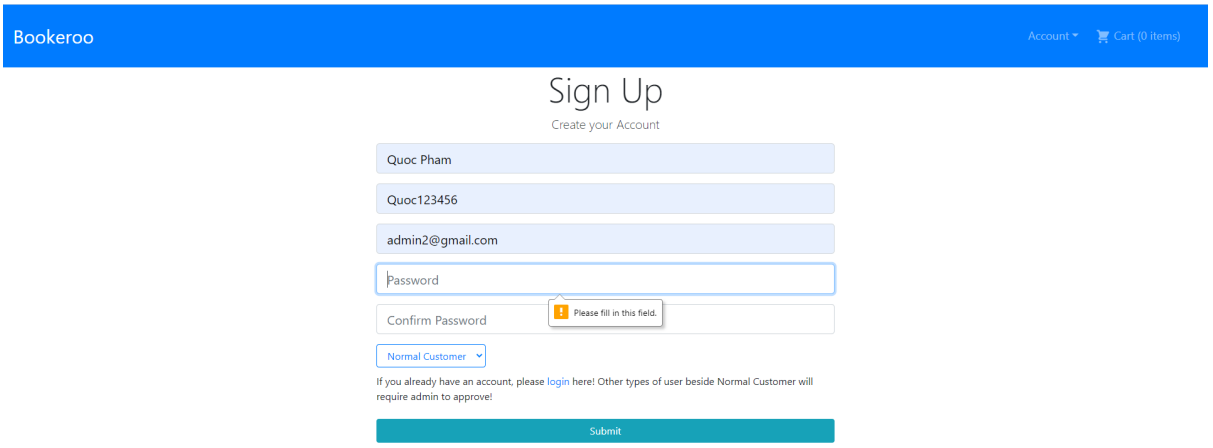
Password

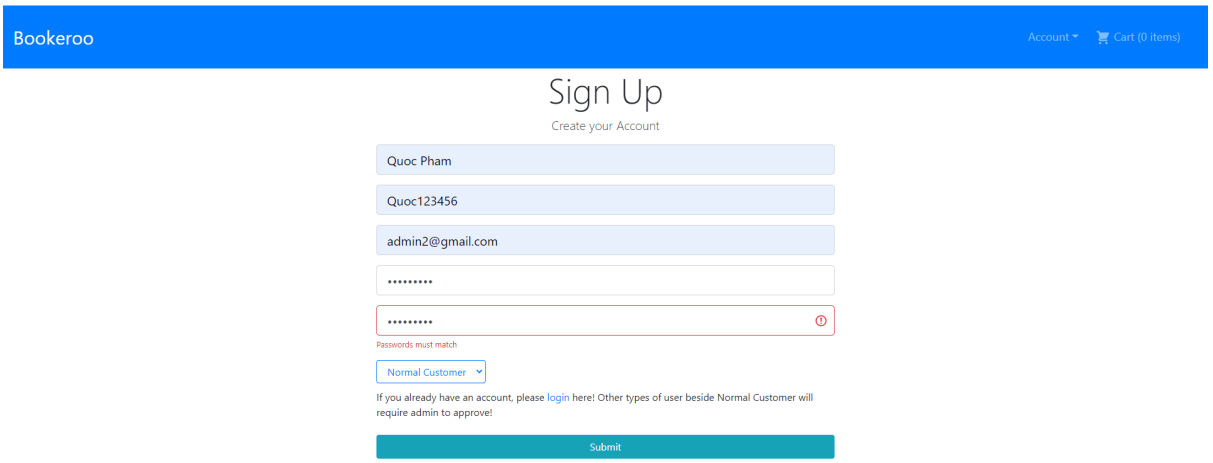
Confirm Password

Normal Customer

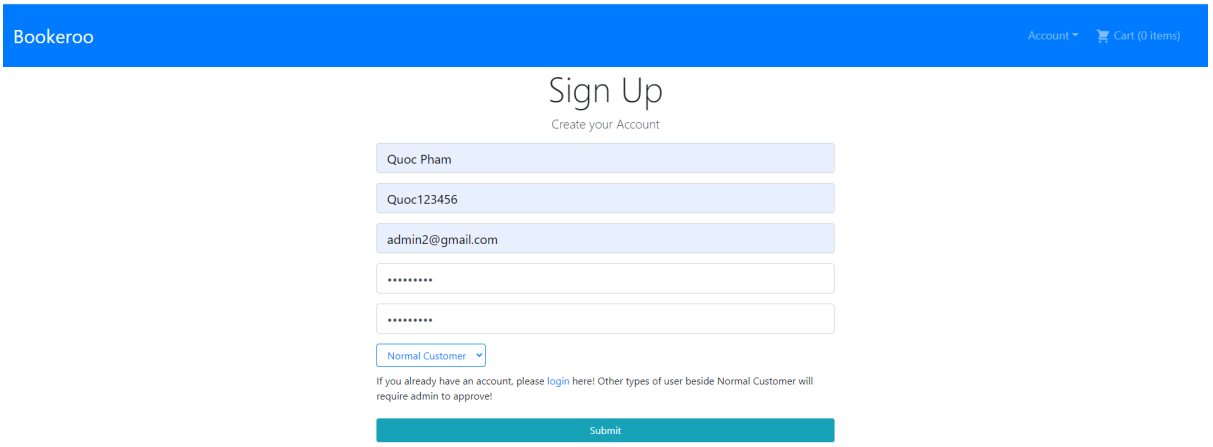
If you already have an account, please login here! Other types of user beside Normal Customer will require admin to approve!

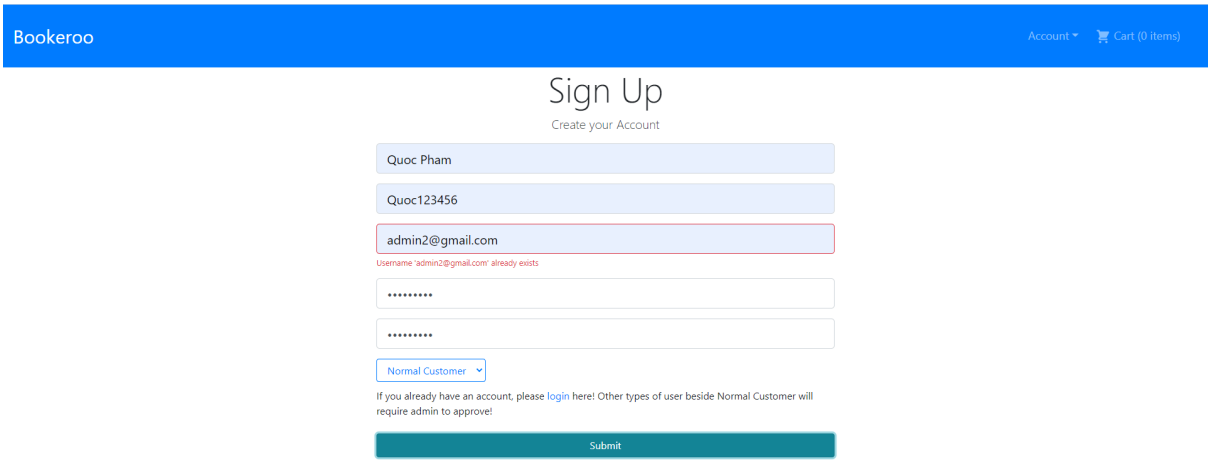
Submit



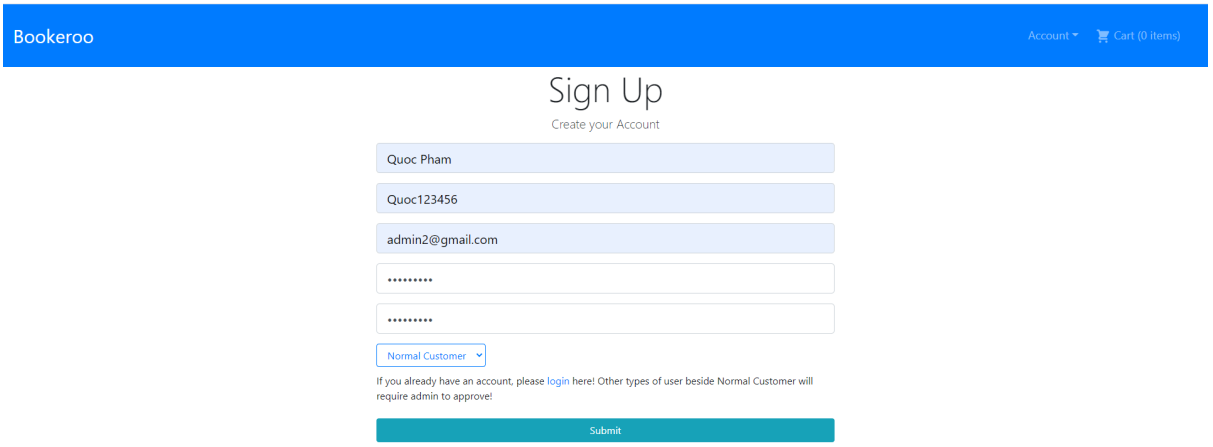


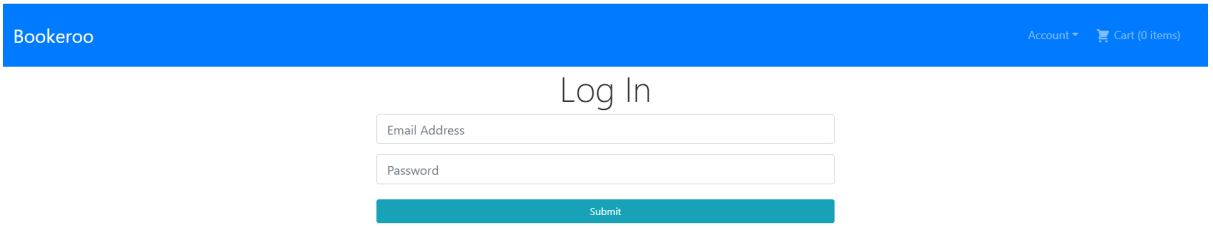
If the account already exists based on the validated username:



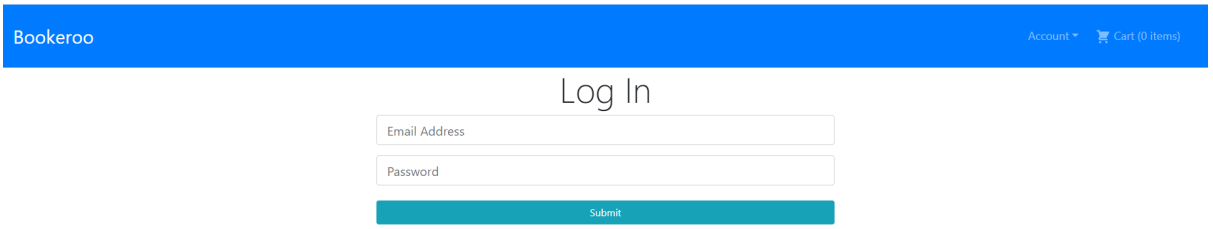


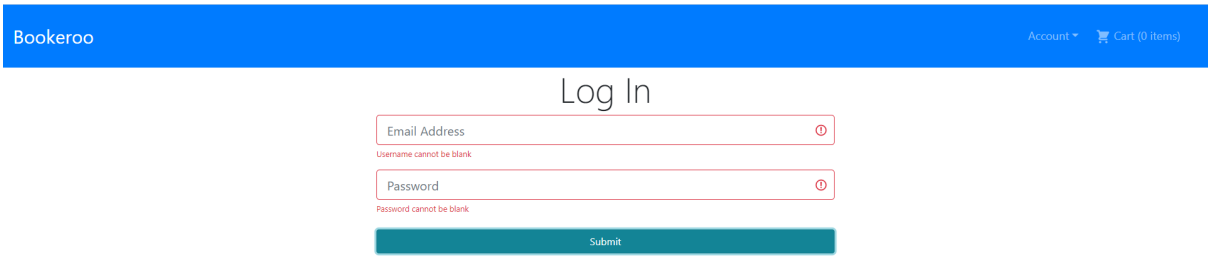
If account doesn't already exist in the database based on validated username, which would redirect the user to the login page:

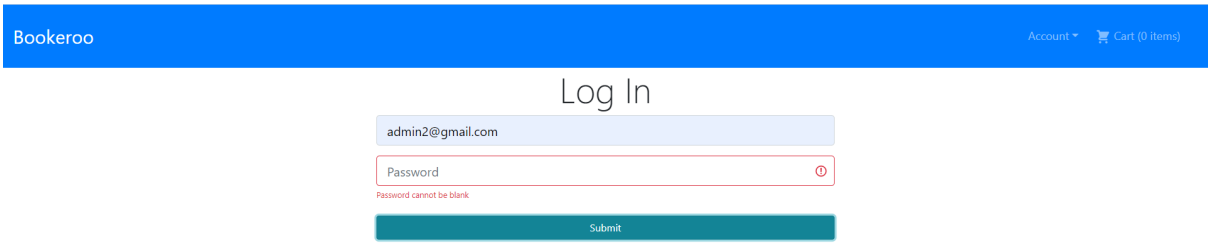


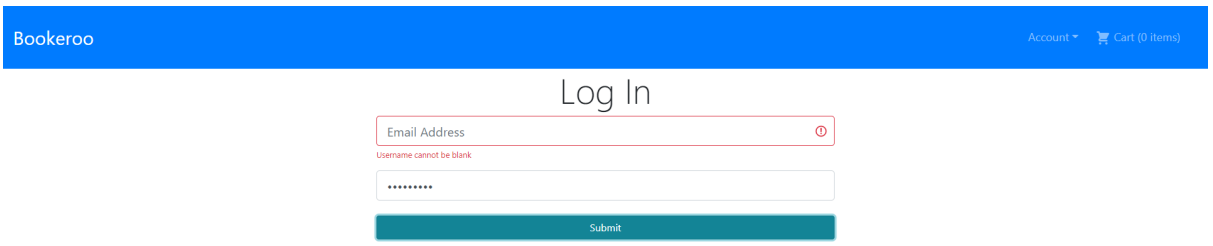


Login page:

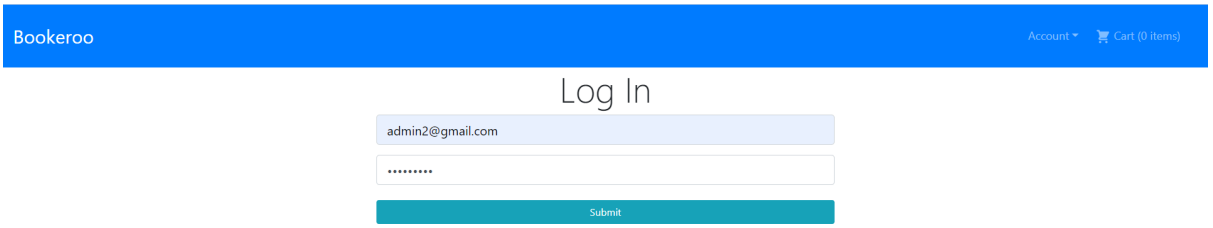


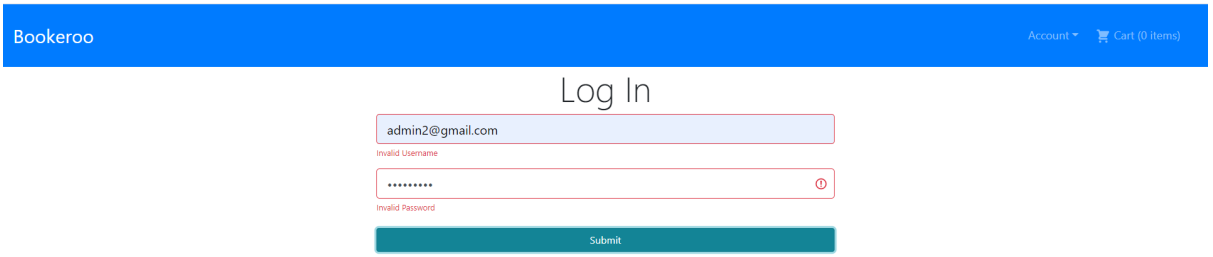




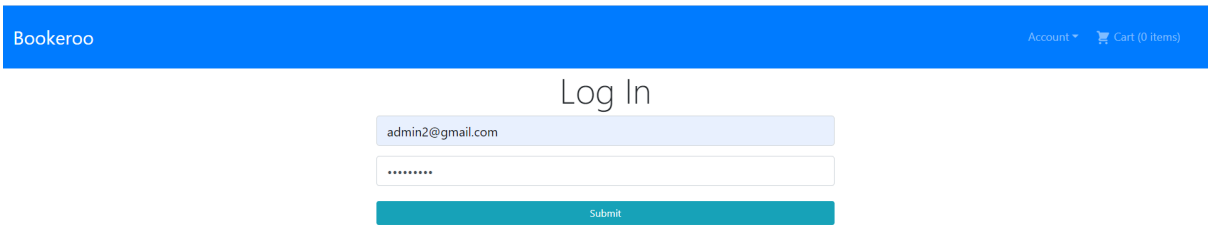


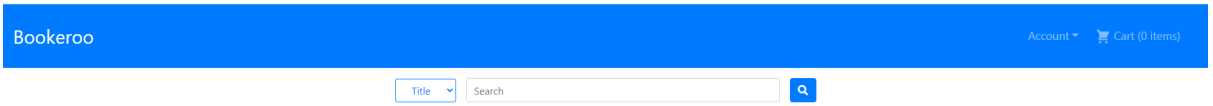
Inputting incorrect password:

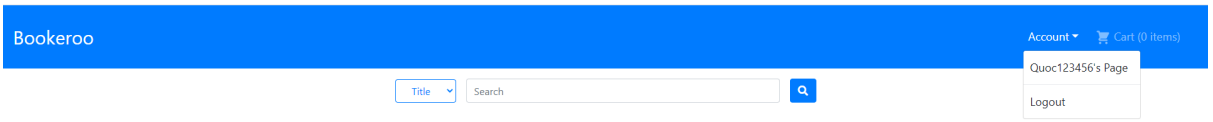




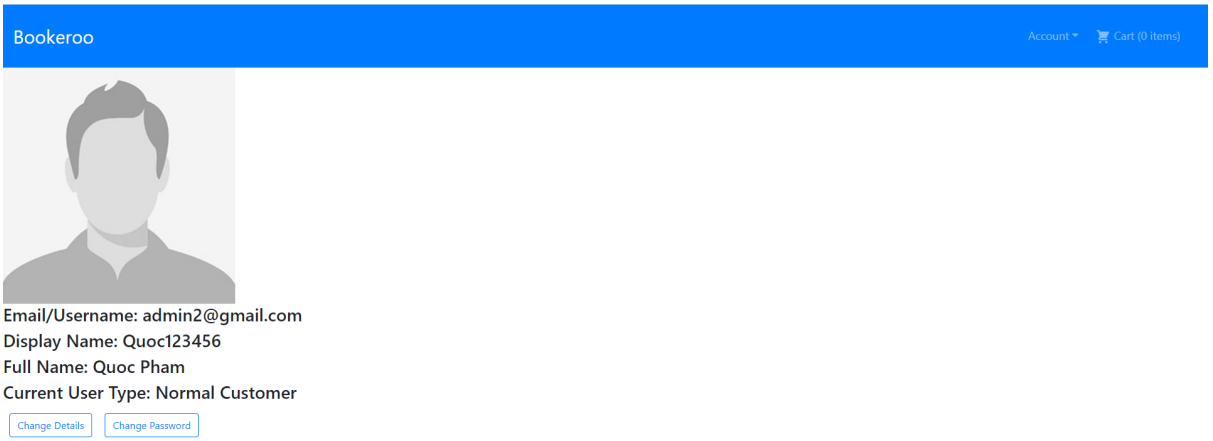
Inputting correct password:



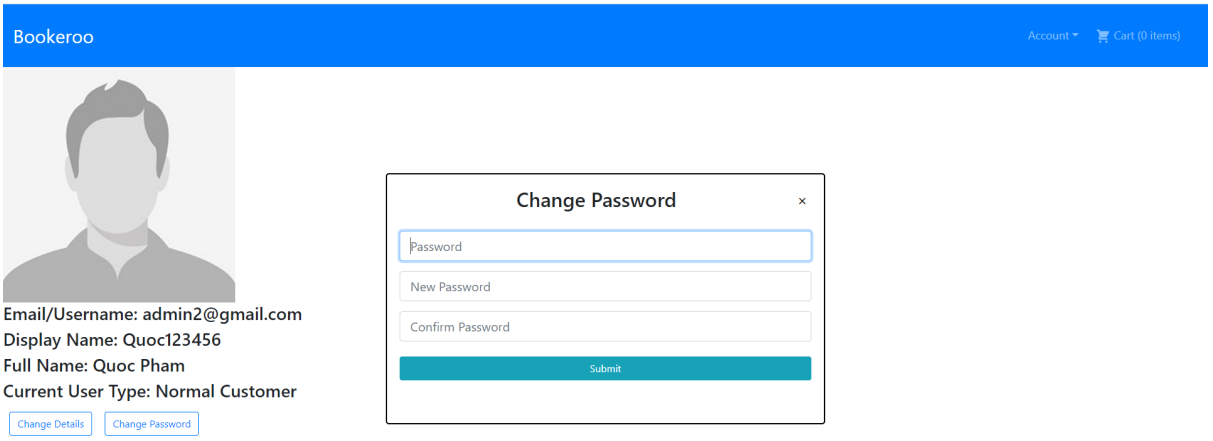




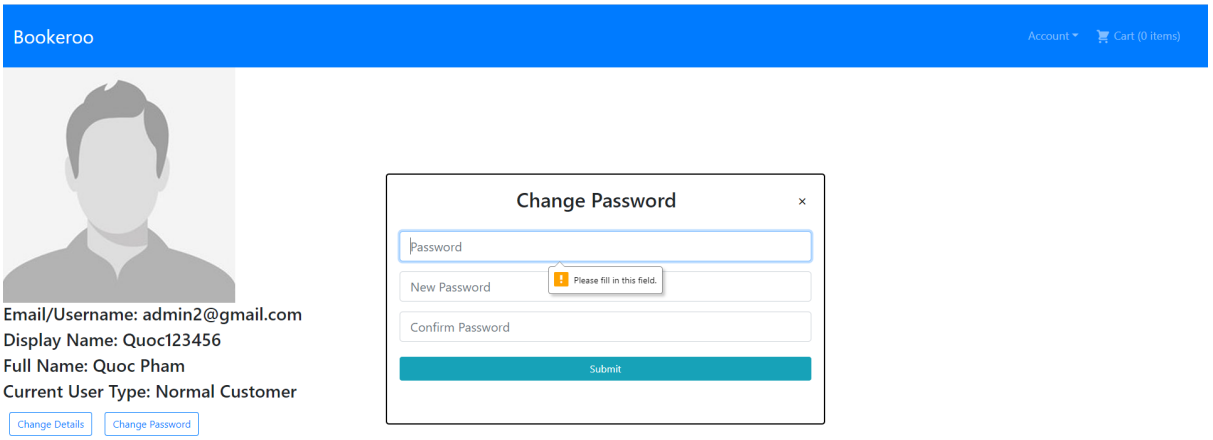
Clicking on the user's page (Quoc123456's page):

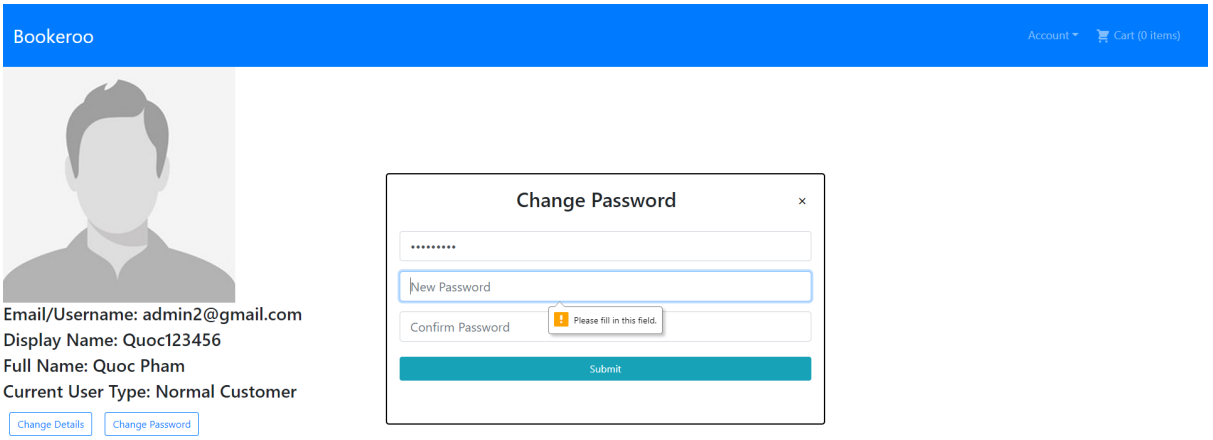


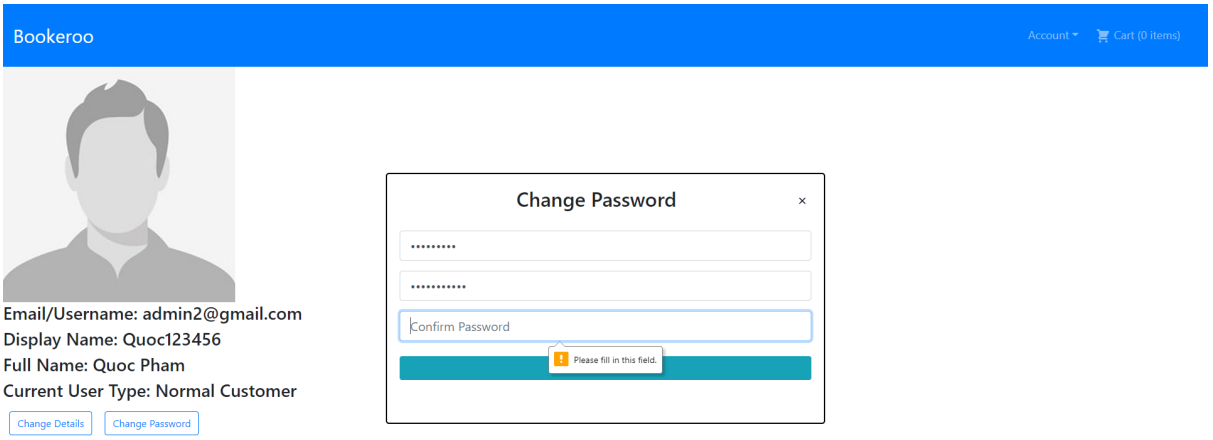
Change Password functions from the popup menu after clicking on "Change Password" button:



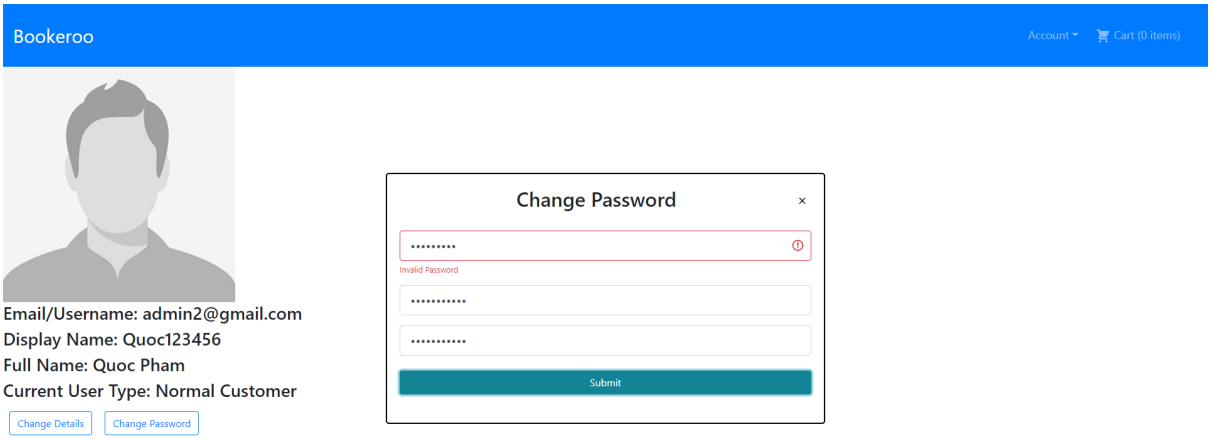
Change password functions when clicking Submit button without filling in the required fields:



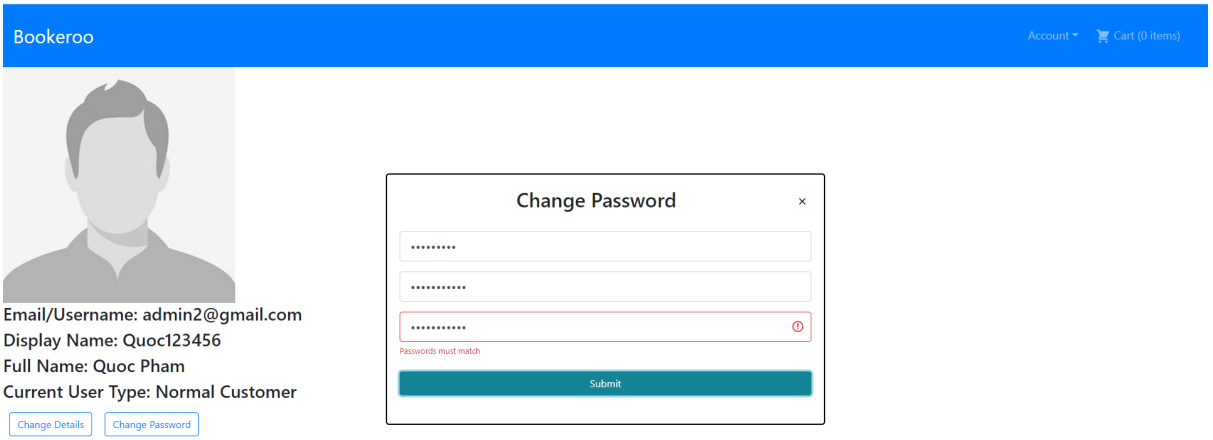




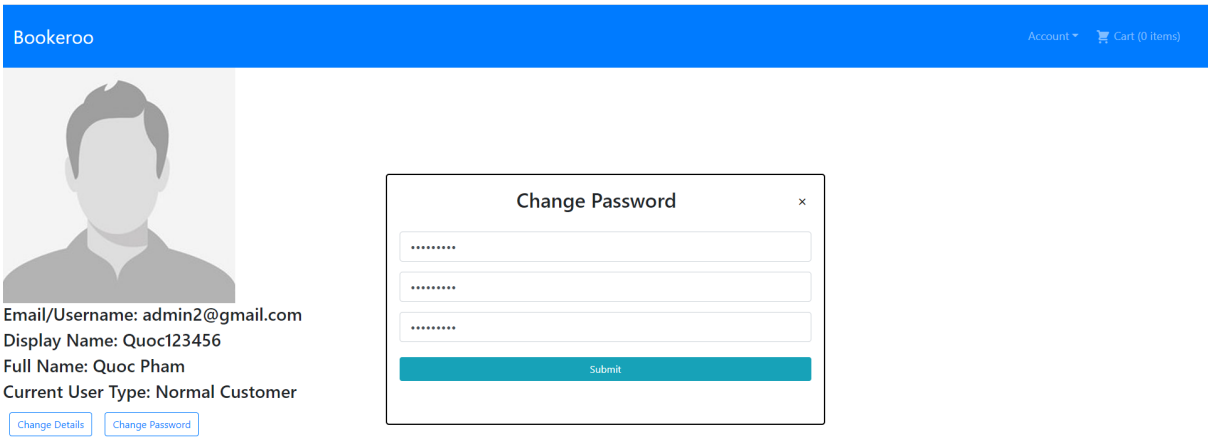
Change password functions after inputting wrong old password:

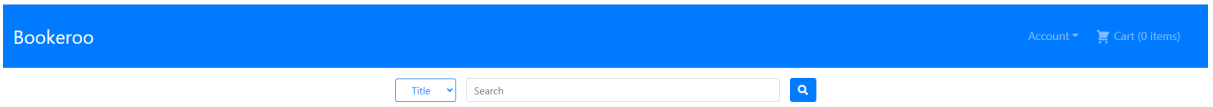


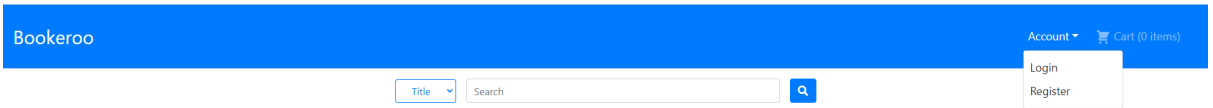
Change password functions after inputting password that does not match confirm password:



Change password functions after inputting correct old password with new password matching new confirm password, which redirects user to the home page with their account logged out:







There is also code for the features involving books such as book add, book remove, and book search, but there is currently no validation code for such functions as the project code we're working on is still in progress and aren't yet close to completed. The snippets of code within the Back-end for such is shown below:

```
import javax.persistence.*;
import java.util.Date;
@Entity
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    private String author;
    private int quality;
    private int isShareBook;
    private int price;
    private Date postDate;
    private float rate;

    public Book()
    {
    }

    public void setTitle(String title)
    {
        this.title = title;
    }
    public String getTitle()
    {
        return this.title;
    }

    public Long getId() {
        return id;
    }

    public String getAuthor() {
        return author;
    }
```

```java
package com.rmit.sept.bookmicroservices.repositories;
import com.rmit.sept.bookmicroservices.model.Book;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import java.util.Collection;

@Repository
public interface BookRepository extends  CrudRepository<Book, Long>
{
    @Query(value = "SELECT * FROM BOOK WHERE LOWER(TITLE) LIKE %?1%", nativeQuery = true)
    Collection<Book> searchBooks(String searchString);
    @Query(value="SELECT * FROM BOOK", nativeQuery = true)
    Collection<Book> getAllBooks();
    @Query(value = "SELECT * FROM BOOK WHERE LOWER(AUTHOR) LIKE %?1%", nativeQuery = true)
    Collection<Book> searchBooksByAuthor(String author);
}
```

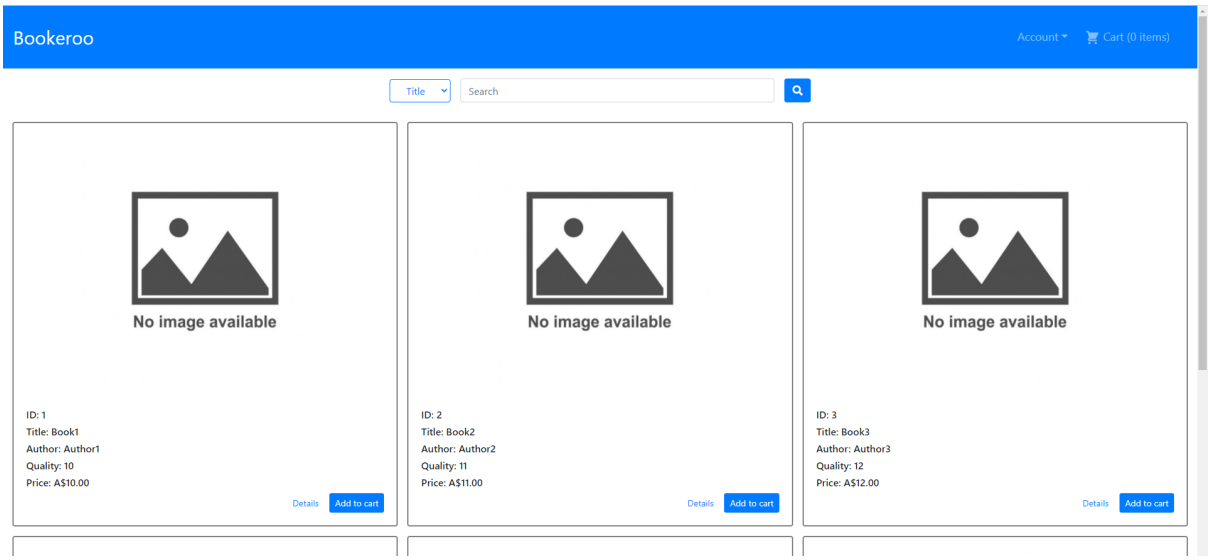
```java
import org.springframework.stereotype.Service;

import java.util.Collection;

@Service
public class BookService
{
    @Autowired
    private BookRepository bookRepository;

    public Book addBook(Book newBook)
    {
        return null;
    }

    public Collection<Book> searchBooks(String searchString)
    {
        return bookRepository.searchBooks(searchString);
    }

    public Collection<Book> getAllBooks()
    {
        return bookRepository.getAllBooks();
    }

    public Collection<Book> searchBooksByAuthor(String author)
    {
        return bookRepository.searchBooksByAuthor(author);
    }

    public Book createBook(Book newBook)
    {
        return bookRepository.save(newBook);
    }
}
```

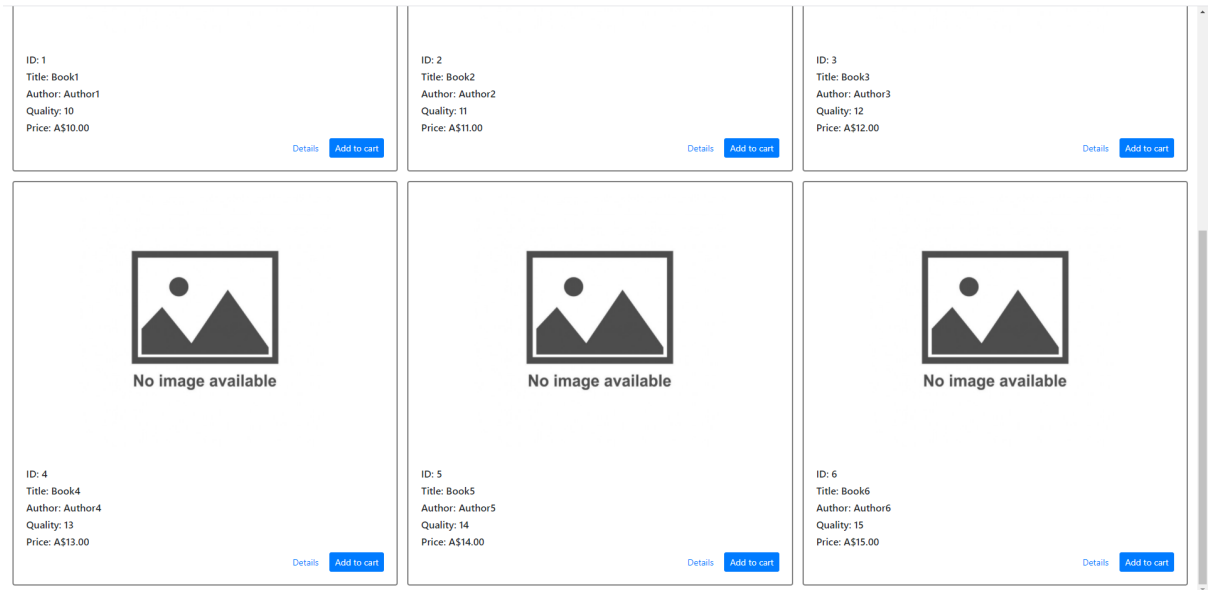
```java
@RestController
@CrossOrigin(origins = "http://localhost:3000")
@RequestMapping("/api/books")
public class BookController {

    @Autowired
    private BookService bookService;
    @GetMapping("/all")
    public @ResponseBody Collection<Book> getAllBooks()
    {
        Collection<Book> books = bookService.getAllBooks();
        return books;
    }

    @GetMapping("/search")
    public @ResponseBody Collection<Book> searchBooks(@RequestParam("searchString") String searchString)
    {
        searchString = searchString.toLowerCase();
        return bookService.searchBooks(searchString);
    }

    @GetMapping("/searchByAuthor")
    public @ResponseBody Collection<Book> searchBooksByAuthor(@RequestParam("searchString") String searchString)
    {
        searchString = searchString.toLowerCase();
        return bookService.searchBooksByAuthor(searchString);
    }
    @PostMapping("/create")
    public ResponseEntity<?> createBook(@RequestBody Book book)
    {
        Book newBook = bookService.createBook(book);
        return new ResponseEntity<Book>(newBook, HttpStatus.CREATED);
    }

}
```

```java
package com.rmit.sept.bookmicroservices;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class BookmicroservicesApplication {

    Run | Debug
    public static void main(String[] args) {
        SpringApplication.run(BookmicroservicesApplication.class, args);
    }

}
```

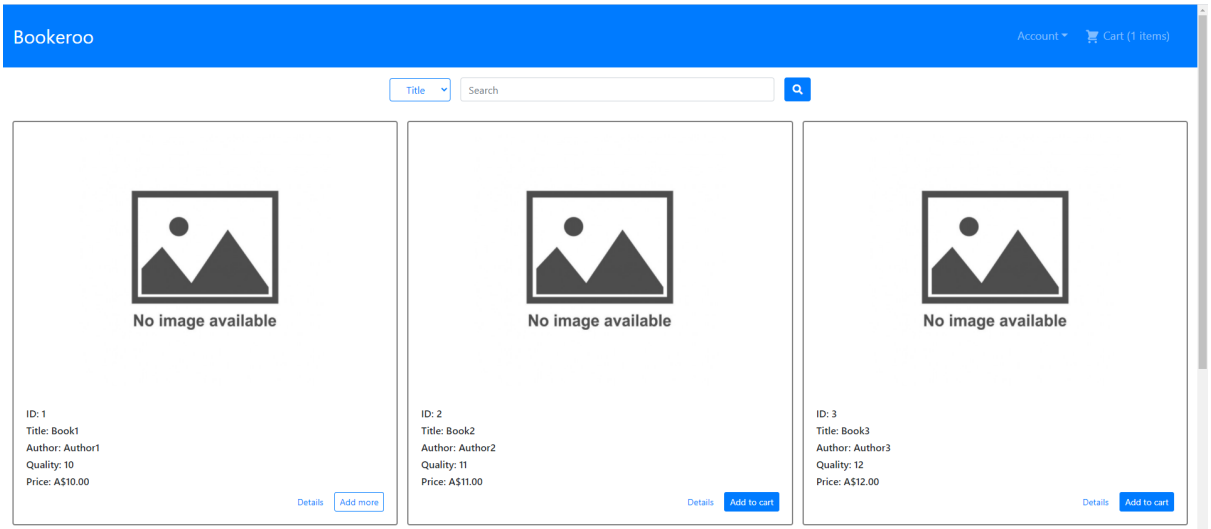
```java
package com.rmit.sept.bookmicroservices;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class BookmicroservicesApplicationTests {

    @Test
    void contextLoads() {
    }

}
```

The snippets of code within the Front-end for such is shown below:

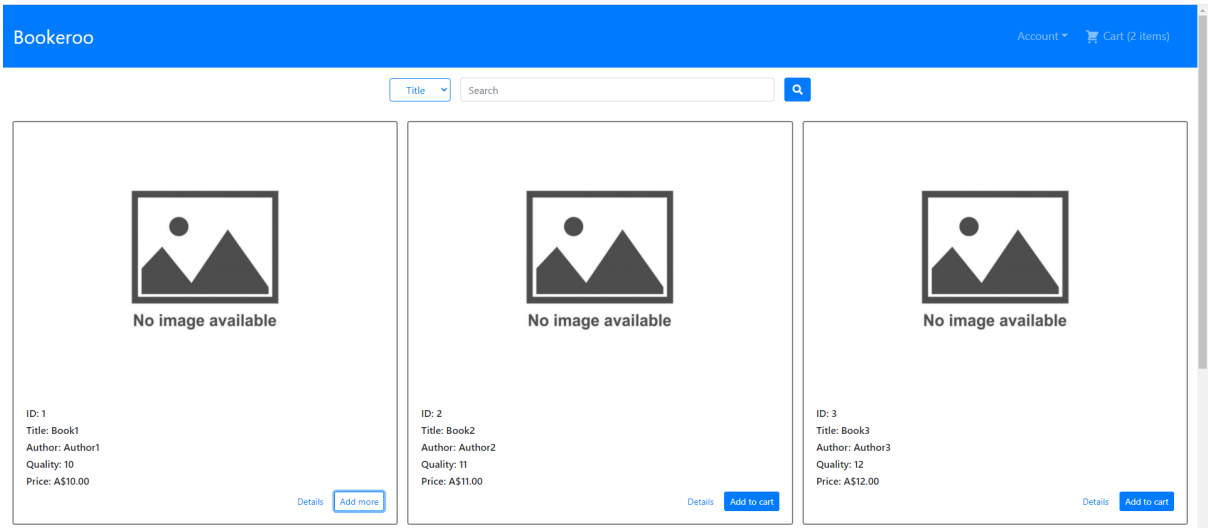




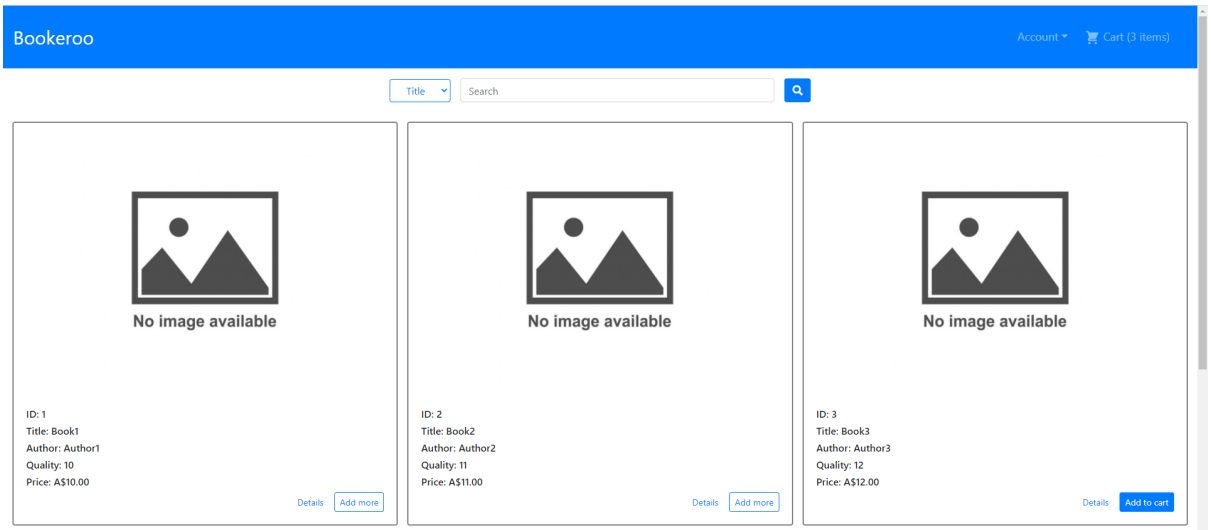
Clicking “Add to cart” button for Book1 is as shown below:



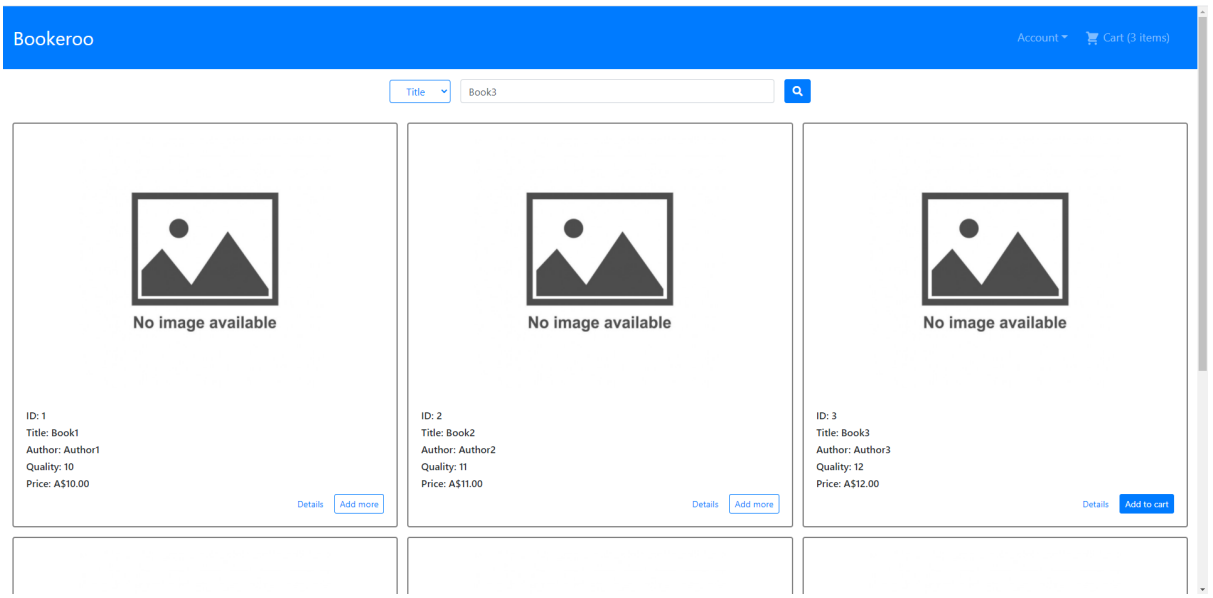
Clicking “Add more” button for Book1 is as shown below, which will add that same book again to the cart, thus increasing the amount of that book the user wants:

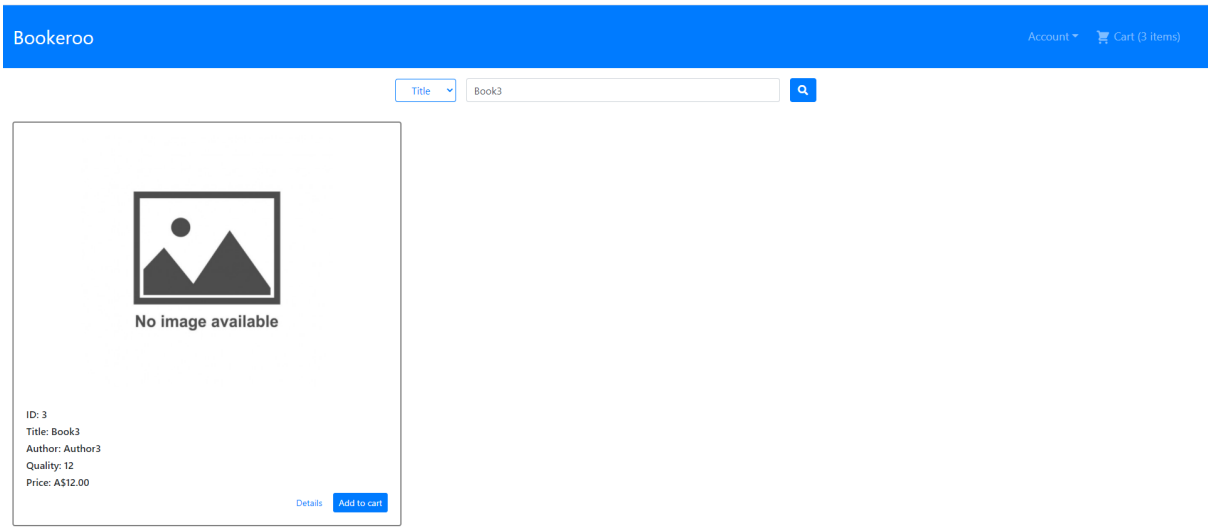


Clicking “Add to cart” button for Book2 is as shown below:

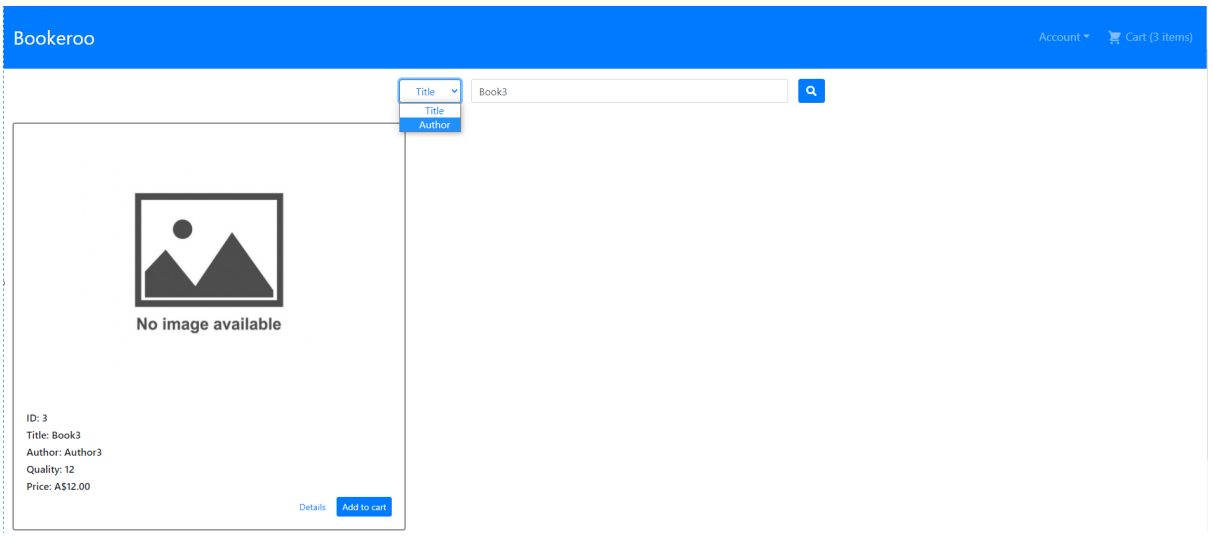


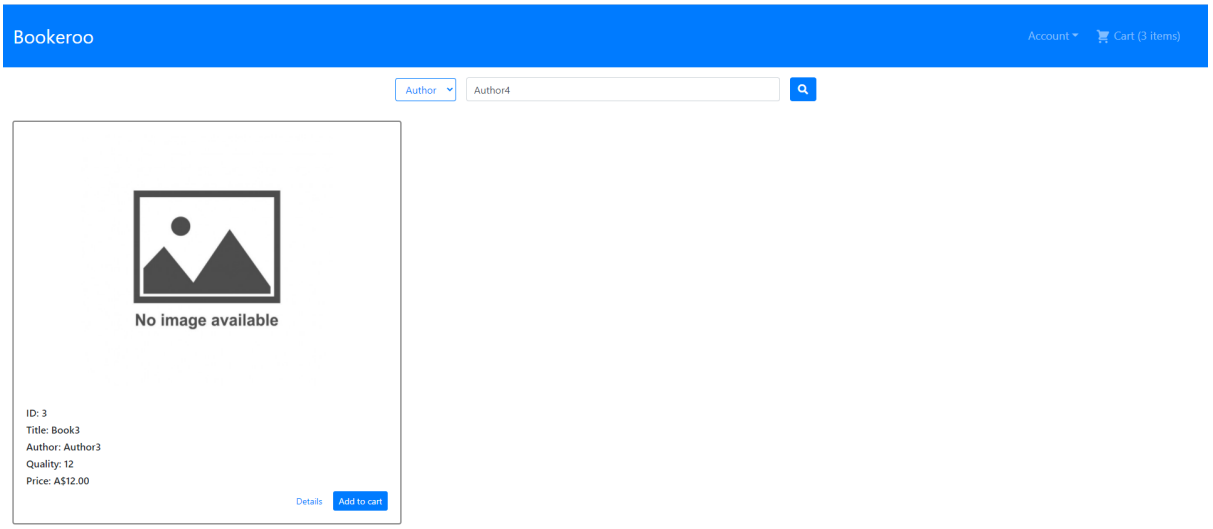
In the search bar for the book search functions, keeping the Dropdown menu as “Title” while entering the title of a book such as “Book3” in the search bar, and then clicking the search icon will then display only that book the user has searched:

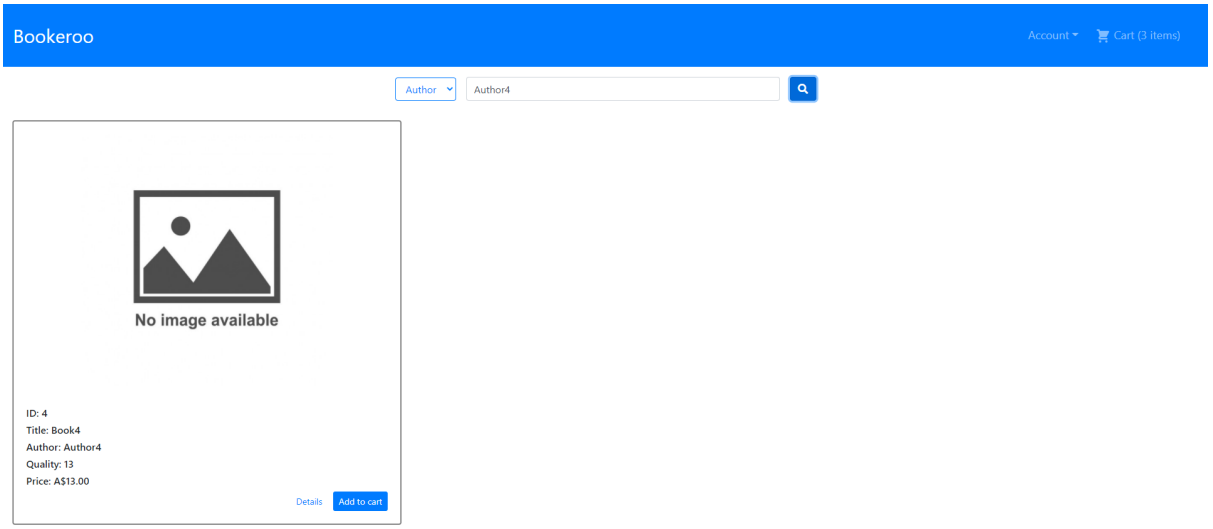




In the search bar for the book search functions, switch the Dropdown menu to “Author” while entering the title of a book such as “Author4” in the search bar, and then clicking the search icon will then display only the books from that author the user has searched.


Bookeroo
Account
Cart (3 items)
Title
Title
Author
Book3
No image available
ID: 3
Title: Book3
Author: Author3
Quality: 12
Price: A$12.00
Details
Add to cart


Bookeroo
Account
Cart (3 items)
Author
Author4
No image available
ID: 3
Title: Book3
Author: Author3
Quality: 12
Price: A$12.00
Details
Add to cart


Bookeroo
Account
Cart (3 items)
Author
Author4
No image available
ID: 4
Title: Book4
Author: Author4
Quality: 13
Price: A$13.00
Details
Add to cart