API Testing: Postman

Test ID	USER STORY	FEATURE	TEST DESCRIPTION	PRIORITY	TEST STEPS	Expected Results	Actual Results	Pass/ Fail - Comments
1	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 the user's account is already created in the database.	Critical	1. Open postman app 2. Input “http://localhost:8080/api/users/getUser?username=admin2@gmail.com” in HTTP Request 3. Click “Send”	"id": 2, "username": "admin2@gmail.com", "fullName": "Quoc Pham", "password": "$2a$10$rzyjWtvhQHnjmV1dTKwqZOOXu9qZFlqSw1pD02AuXJodEVw56gvii", "displayName": "Quoc123456", "confirmPassword": null, "create_At": "2021-09-18T03:10:52.221+00:00", "update_At": null, "userType": "Normal Customer", "userTypeRequest": ""	"id": 2, "username": "admin2@gmail.com", "fullName": "Quoc Pham", "password": "$2a$10$rzyjWtvhQHnjmV1dTKwqZOOXu9qZFlqSw1pD02AuXJodEVw56gvii", "displayName": "Quoc123456", "confirmPassword": null, "create_At": "2021-09-18T03:10:52.221+00:00", "update_At": null, "userType": "Normal	Pass

							Customer", "userType Request": ""	
2	As a Customer, I want to search books, so that I can get information about the books	User book search, to verify if customer is able to search for desired books on the website	User must be able to search for books	Med	1. Open postman app 2. Input "http://localhost:8080/api/books/search?searchString=Book2" in HTTP Request 3. Click "Send"	"id": 2, "title": "Book2", "author": "Author2", "quality": 11, "price": 11, "postDate": "2021-09-17T14:07:30.862+00:00", "rate": 11.0	"id": 2, "title": "Book2", "author": "Author2", "quality": 11, "price": 11, "postDate": "2021-09-17T14:07:30.862+00:00", "rate": 11.0	Pass
3	As a Customer, I want to search books, so that I can get information about the books	User book search, to verify if customer is able to search for desired books on the website	User must be able to search for books	Med	1. Open postman app 2. Input "http://localhost:8080/api/books/searchByAuthor?searchString=Author1" in HTTP Request 3. Click "Send"	"id": 1, "title": "Book1", "author": "Author1", "quality": 10, "price": 10, "postDate": "2021-09-17T14:05:56.984+00:00", "rate": 10.0	"id": 1, "title": "Book1", "author": "Author1", "quality": 10, "price": 10, "postDate": "2021-09-17T14:05:56.984+00:00", "rate": 10.0	Pass

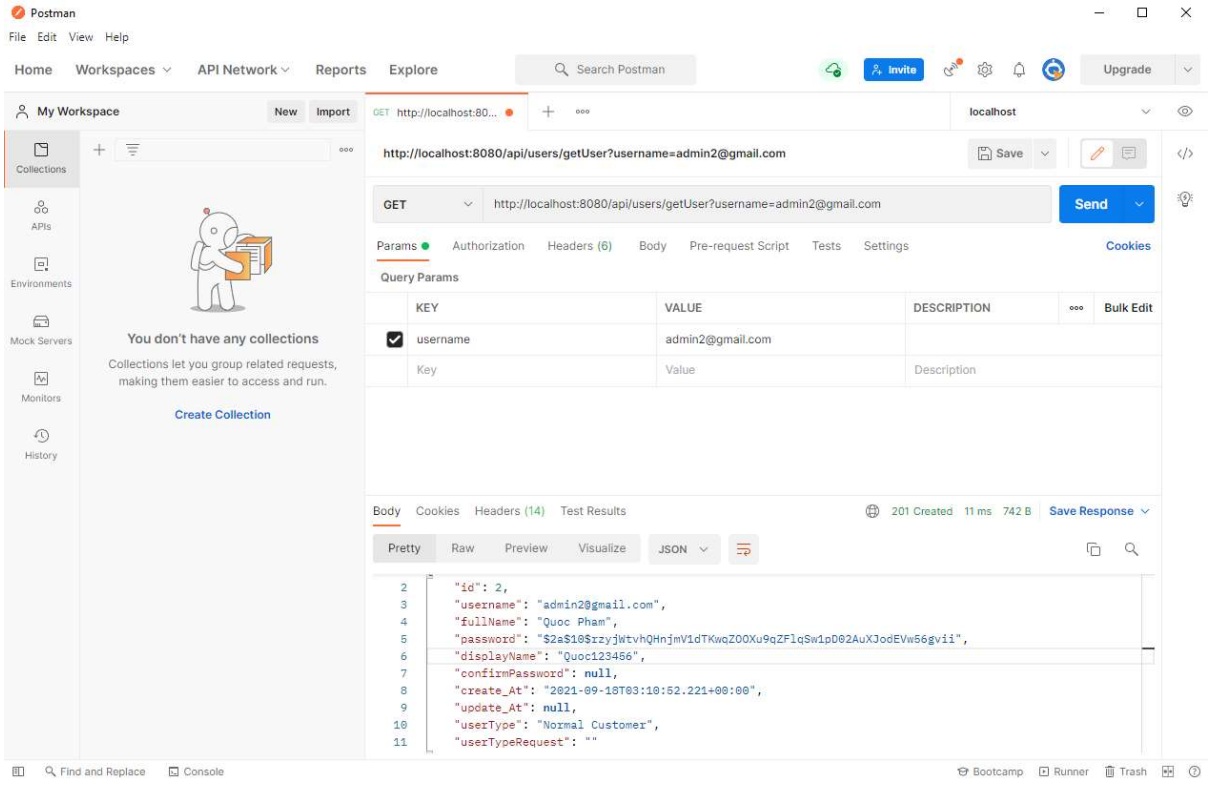
In our assignment code, there is a GetMapping code in the Backend implemented as a get method code that returns all the details of a certain user based on their inputted username (or email) within

the user controller code, which are derived from the executed query located within the user repository code. This will then return an entire collection of users as a GET request in the Postman application.

```
@GetMapping("/getUser")
public ResponseEntity<?> getUser(@RequestParam("username") String username){
    User newUser = userService.getUser(username);
    return  new ResponseEntity<User>(newUser, HttpStatus.CREATED);
}
```

```
@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);
    }
```

```java
import com.rmit.sept.usermicroservices.model.User;


import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

@Repository
public interface UserRepository extends CrudRepository<User, Long> {

    User findByUsername(String username);
    User getById(Long id);

    @Query(value = "SELECT * FROM USER WHERE USERNAME = ?1", nativeQuery = true)
    User getUser(String username);
```



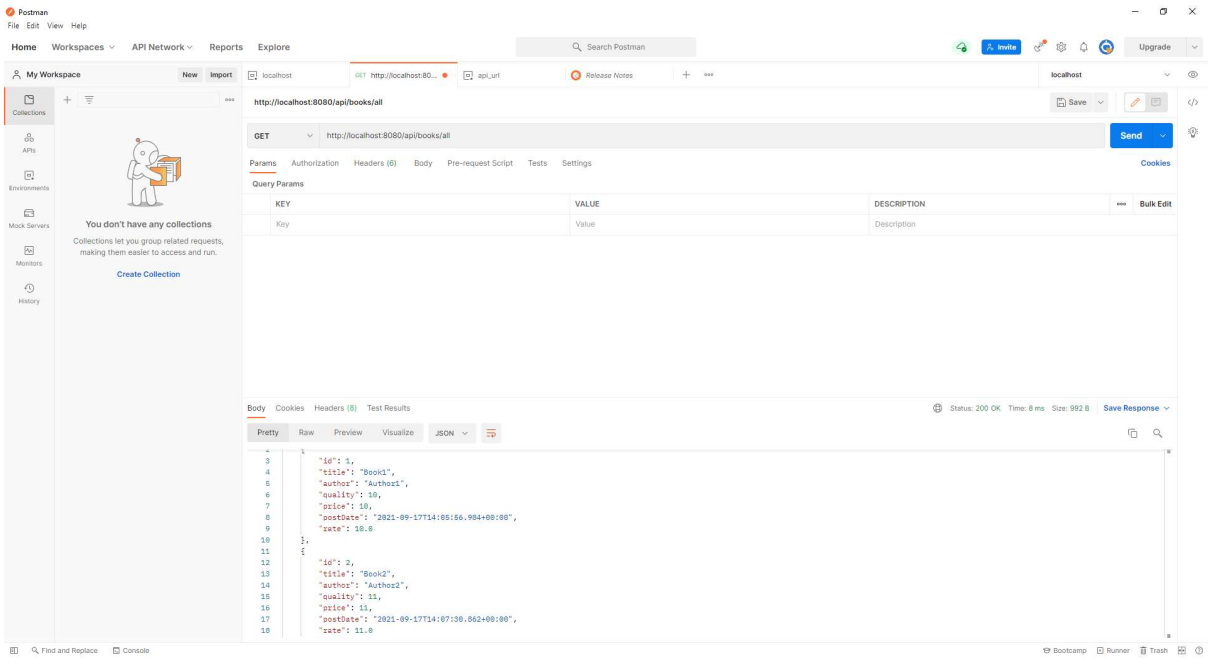
Similarly, there is a GetMapping code in the Backend implemented as a get method code that returns all the details of all books per collection based on the /all GetMapping, although there are /search and /searchByAuthor GetMappings that can be used to return a collection of book details of books that the user have searched based on the inputted title or author within the book controller code respectively, which are derived from the executed query located within the book repository code. This will then return an entire collection of users as a GET request in the Postman application.

```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);
    }
```

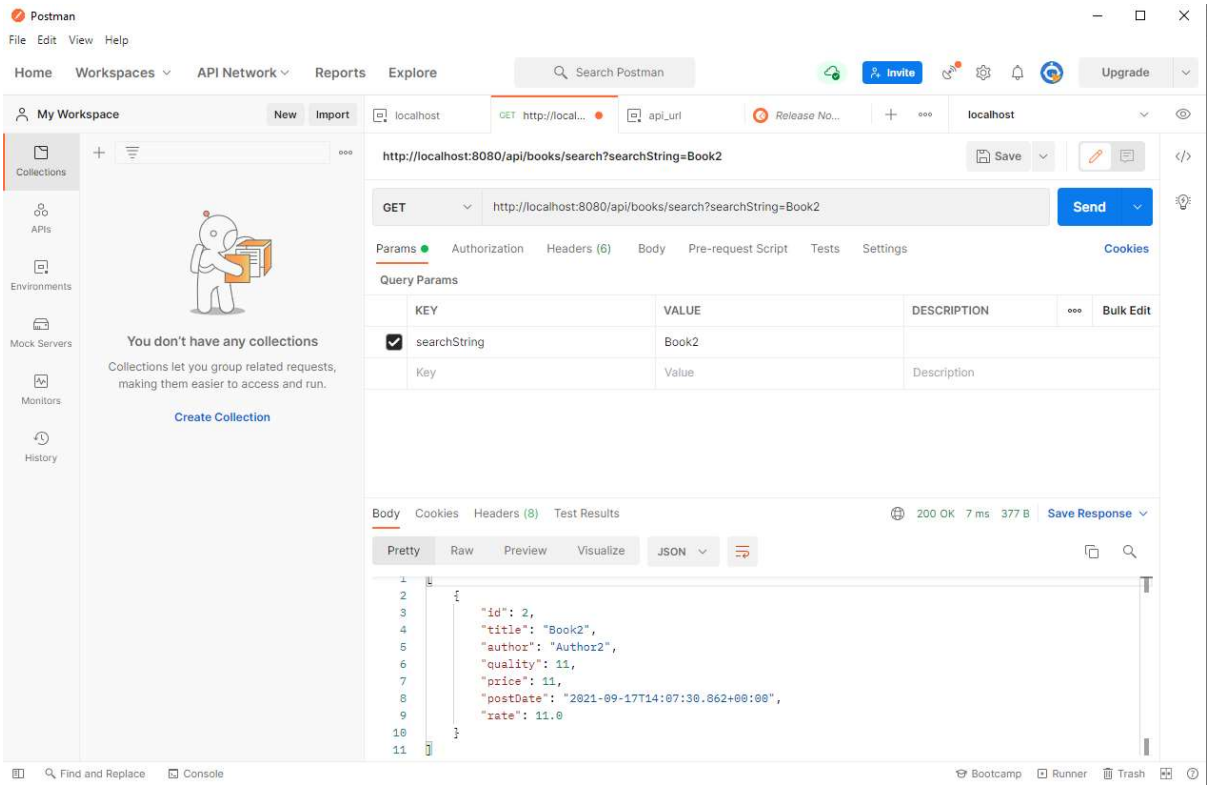
```java
import org.springframework.stereotype.Service;

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
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);
}
```

/all:



/search:



/searchByAuthor: