



1DV600

## Test Plan



13 Mar, 2017

*Semester:* Spring 2017  
*Course:* Software Technology  
*Author:* Sarpreet Singh Buttar

**Contents**

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Purpose . . . . .	1
1.2	System Background . . . . .	1
1.3	Stakeholder’s Goals . . . . .	1
1.4	Scope . . . . .	1
<b>2</b>	<b>Requirements for Test</b>	<b>2</b>
2.1	Function Testing . . . . .	2
2.2	Unit Testing . . . . .	2
2.3	API Testing . . . . .	2
<b>3</b>	<b>Test Strategy</b>	<b>3</b>
3.1	Function Testing . . . . .	3
3.2	Unit Testing . . . . .	3
3.3	API Testing . . . . .	3
<b>4</b>	<b>Resources</b>	<b>4</b>
4.1	Man Power . . . . .	4
4.2	Tools . . . . .	4
<b>5</b>	<b>Milestones</b>	<b>4</b>
<b>6</b>	<b>Test Plan</b>	<b>5</b>
<b>7</b>	<b>Test Design/Cases</b>	<b>5</b>
<b>8</b>	<b>Test Evaluation/Result</b>	<b>19</b>

# 1 Introduction

## 1.1 Purpose

The purpose of this test plan document is to support the followings objectives:

- Identifying requirements
- Creating test cases for all the use cases
- Implementation of the unit tests for the model classes
- Implementation of the unit tests for the test cases
- Implementation of API tests
- Illustrated the tests result

## 1.2 System Background

The given web application is a book library system and its purpose is to create, read, update and delete (CRUD) a book. It is a half-finished web application consists of two modules, the first module is a fully functional front end (client) which we will use to verify the code that we will write during the development of the partially implemented back end (server). The application stack is developed in a modern way which means these two modules will communicate via HTTP request specified in a given API. In result, no rendering is performed on server side because front end is implemented using Single Page Application (SPA) architecture and the only way to know how to communicate is to follow the API. Furthermore, for executing the application server, a virtual machine is required that is managed by Vagrant whereas the client side is executed in web browser.

## 1.3 Stakeholder's Goals

Customers	Developers
Fully working and tested CRUD functionality of the system	All the functionality should work according to the API as well as fully tested

## 1.4 Scope

This test plan applies to Unit Testing, Function Testing and API Testing. All these tests will be implemented and executed in this iteration. Information regarding the aim of these tests can be find in **Section 2** (Requirements for Test).

## 2 Requirements for Test

The listing below (uses cases, functional-requirements and non-functional-requirements) identifies all items that have been identified as targets for testing. It represents what will be tested.

### 2.1 Function Testing

Requirement #	Use Case
2.1.1	View a book
2.1.2	View a list of books
2.1.3	Add a book
2.1.4	Edit a book
2.1.5	Delete a book
2.1.6	Search a book by author or title

### 2.2 Unit Testing

Requirement #	Class
2.2.1	bookTest
2.2.2	catalogTest
2.2.3	booksDAOTest

### 2.3 API Testing

Requirement #	Class
2.3.1	AddBookResource
2.3.2	EditBookResource
2.3.3	GetBookResource
2.3.4	GetBooksResource
2.3.5	RemoveBookResource

### 3 Test Strategy

The Test Strategy presents a recommended approach of testing the identified testing-targets. The previous section described what are the testing-targets. This section will describe how it will be tested. The main considerations for the test strategy are the techniques to be used and the criterion for knowing when the testing is completed.

#### 3.1 Function Testing

<b>Test objectives</b>	Ensure proper functioning of the application
<b>Techniques</b>	Execute each use case flow and identify test target using valid and invalid inputs data to verify the expected result. In addition, mock the dependencies
<b>Completion criteria</b>	All the tests should be executed and failure tests should have to be reported
<b>Special consideration</b>	The server must be running before the execution of tests

#### 3.2 Unit Testing

<b>Test objectives</b>	Verify the proper functioning of the classes
<b>Techniques</b>	Create and run the unit tests. In addition, mock the dependencies
<b>Completion criteria</b>	All the tests should be executed and failure tests should have to be reported
<b>Special consideration</b>	The server must be running before the execution of tests

#### 3.3 API Testing

<b>Test objectives</b>	Ensure the application response according to the API
<b>Techniques</b>	Create automotive test and mock the dependencies
<b>Completion criteria</b>	System must response according to API and failure tests should have to be reported
<b>Special consideration</b>	The server must be running before the execution of tests

## 4 Resources

Since it is individual assignment, so all the roles will be played by myself.

### 4.1 Man Power

Role	Resources	Responsibilities
Test Designer	1	Identify and implement test cases
Tester	1	Executes tests
Programmer	1	Implement some missing functionality

### 4.2 Tools

Name	Type	Purpose
VS Code	Editor	Implementation
Vagrant	Virtual Machine	Code executor
Postman	API Tester	Function Testing

## 5 Milestones

Planning and working will be done in an agile-way as defined in the test-plan.

Milestone task	Start data	End date
Test Plan	13 Mar 2017	13 Mar 2017
Test Design	13 Mar 2017	13 Mar 2017
Test Implementation	13 Mar 2017	13 Mar 2017
Test Execution	13 Mar 2017	13 Mar 2017
Test Evaluation	13 Mar 2017	13 Mar 2017

## 6 Test Plan

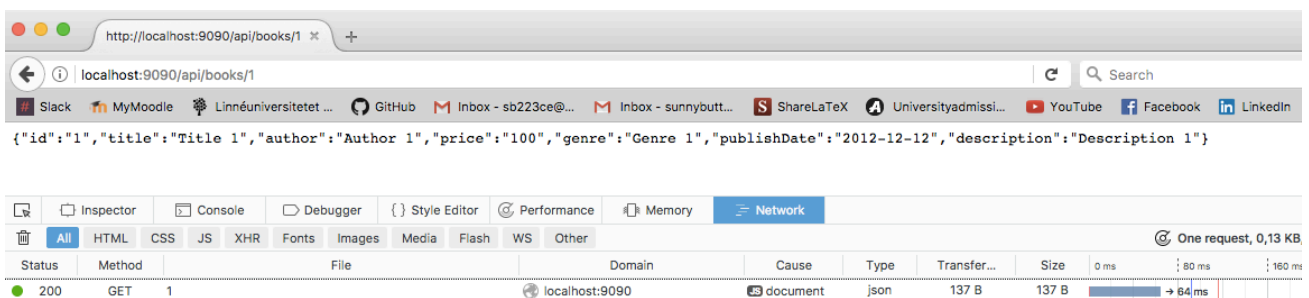
Test Type	Goals	Requirement #	Start data	End date
Function Testing	Create test cases and implement unit tests of all the use cases	<ul style="list-style-type: none"><li>• 2.1.1</li><li>• 2.1.2</li><li>• 2.1.3</li><li>• 2.1.4</li><li>• 2.1.5</li><li>• 2.1.6</li></ul>	13 Mar 2017	13 Mar 2017
Unit Testing	Implementing unit tests for all the classes	<ul style="list-style-type: none"><li>• 2.2.1</li><li>• 2.2.2</li><li>• 2.2.3</li></ul>	13 Mar 2017	13 Mar 2017
API Testing	Test the resources	<ul style="list-style-type: none"><li>• 2.3.1</li><li>• 2.3.2</li><li>• 2.3.3</li><li>• 2.3.4</li><li>• 2.3.5</li></ul>	13 Mar 2017	13 Mar 2017

## 7 Test Design/Cases

Following are the test cases related to all of the use cases which has been identified in the system

<b>ID</b>	1
<b>Name</b>	Should view a book
<b>Date</b>	13 Mar 2017
<b>Use case covered</b>	View a book
<b>Requirement Covered</b>	2.1.1
<b>Scenario</b>	User send GET request URI (/api/books/{book_id})
<b>Pre conditions</b>	System must have requested book
<b>Post conditions</b>	User should view a book
<b>Notes</b>	Server should be up on port 9090
<b>Test Data</b>	localhost:9090/api/books/1

S.No	Steps	Expected Result
1	Open any browser and provide 'Test Data' in the search tab	User should view a book in JSON format whose 'id' is same as requested and server should response 200 OK



Status	Method	File	Domain	Cause	Type	Transfer...	Size	0 ms	80 ms	160 ms
200	GET	1	localhost:9090	document	json	137 B	137 B	→ 64 ms		



ID	2	
Name	Should not view a book	
Date	13 Mar 2017	
Use case covered	View a book	
Requirement Covered	2.1.1	
Scenario	User send GET request URI (/api/books/{book_id})	
Pre conditions	System should not have requested book	
Post conditions	User should not view a book	
Notes	Server should be up on port 9090	
Test Data	localhost:9090/api/books/100	
S.No	Steps	Expected Result
1	Open any browser and provide ‘Test Data’ in the search tab	User should not view a book and server should response 404 Not Found

**HTTP ERROR 404**

Problem accessing /api/books/100. Reason:

Not Found

Powered by Jetty://

Status	Method	File	Domain	Cause	Type
404	GET	100	localhost:9090	document	html

<b>ID</b>	3
<b>Name</b>	Should view a list of books
<b>Date</b>	13 Mar 2017
<b>Use case covered</b>	View a list of books
<b>Requirement covered</b>	2.1.2
<b>Scenario</b>	User send GET request URI (/api/books)
<b>Pre conditions</b>	System must have at least one book
<b>Post conditions</b>	User should view a list of books
<b>Notes</b>	Server should be up on port 9090
<b>Test Data</b>	localhost:9090/api/books

S.No	Steps	Expected Result
1	Open any browser and provide 'Test Data' in the search tab	User should view a list of books in JSON array format and server should response 200 OK

The screenshot shows a web browser displaying a JSON array of book objects. The JSON data is as follows:

```
[{"id":1,"title":"Foundation and Empire","author":"Isaac Asimov","price":79,"genre":"Science Fiction","publishDate":"1952-10-12","description":"Foundation and Empire is a novel written by Isaac Asimov that was published by Gnome Press in 1952. It is the second book published in the Foundation Series, and the fourth in the in-universe chronology. It takes place in two halves, originally published as separate novellas: The second part, The Mule, won a Hugo Award."},{id:3,"title":"Second Foundation","author":"Isaac Asimov","price":79,"genre":"Science Fiction","publishDate":"1953-05-10","description":"Second Foundation consists of two previously published novellas originally published in Astounding Magazine (with different titles) between 1948 and 1950, making this the third volume in Asimovs Foundation series. Decades later, Asimov wrote two further sequel novels and two prequels. Later writers have added authorized tales to the series. The Foundation series is often regarded as one of Isaac Asimovs best works, along with his Robot series."},{id:4,"title":"Design Patterns: Elements of Reusable Object-Oriented Software","author":"Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John","price":350,"genre":"Computer Science","publishDate":"1994-10-21","description":"Design Patterns: Elements of Reusable Object-Oriented Software is a software engineering book describing recurring solutions to common problems in software design. The books authors are Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides with a foreword by Grady Booch. The book is divided into two parts, with the first two chapters exploring the capabilities and pitfalls of object-oriented programming, and the remaining chapters describing 23 classic software design patterns. The book includes examples in C++ and Smalltalk."},{id:5,"title":"Lewis Carroll","author":"Alice in Wonderland","price":99,"genre":"Literary nonsense","publishDate":"1865-11-26","description":"Alice in Wonderland tells of a girl named Alice falling through a rabbit hole into a fantasy world populated by peculiar, anthropomorphic creatures. The tale plays with logic, giving the story lasting popularity with adults as well as with children. It is considered to be one of the best examples of the literary nonsense genre. Its narrative course and structure, characters and imagery have been enormously influential in both popular culture and literature, especially in the fantasy genre."},{id:6,"title":"Ronia the Robbers Daughter","author":"Astrid Lindgren","price":79,"genre":"Fantasy","publishDate":"1981-04-23","description":"Ronia is a girl growing up among a clan of robbers living in a castle in the woodlands of early-Medieval Scandinavia. As the only child of Matt, the chief, she is expected to become the leader of the clan someday. Their castle, Matts Fort, is split in two parts by a lightning bolt on the day of Ronias birth. Ronia grows up with Matts clan of robbers as her only company, until a rival robber group led by Borka moves into the other half of the castle, exacerbating the longstanding rivalry between the two bands."},{id:7,"title":"The Da Vinci Code","author":"Dan Brown","price":139,"genre":"Mystery","publishDate":"2003-04-02","description":"The Da Vinci Code is a 2003 mystery-detective novel by Dan Brown. It follows symbologist Robert Langdon and cryptologist Sophie Neveu after a murder in the Louvre Museum in Paris, when they become involved in a battle between the Priory of Sion and Opus Dei over the possibility of Jesus Christ having been married to Mary Magdalene. The title of the novel refers, among other things, to the finding of the first murder victim in the Grand Gallery of the Louvre, naked and posed like Leonardo da Vincis famous drawing, the Vitruvian Man, with a cryptic message written beside his body and a pentagram drawn on his chest in his own blood."},{id:8,"title":"Gone with the Wind","author":"Margaret Mitchell","price":93,"genre":"Romance","publishDate":"1936-06-10","description":"Gone with the Wind is a novel written by Margaret Mitchell, first published in 1936. The story is set in Clayton County, Georgia, and Atlanta during the American Civil War and Reconstruction era. It depicts the struggles of young Scarlett O'Hara, the spoiled daughter of a well-to-do plantation owner, who must use every means at her disposal to claw her way out of the poverty she finds herself in after Shermans March to the Sea. A historical novel, the story is a Bildungsroman or coming-of-age story, with the title taken from a poem written by Ernest Dowson."},{id:9,"title":"Think and Grow Rich","author":"Napoleon Hill","price":124,"genre":"Personal Development","publishDate":"1927-11-12","description":""},{id:10,"title":"A Brief History of Time","author":"Stephen Hawking","price":199,"genre":"Science","publishDate":"1988-09-01","description":"Hawking attempts to explain a range of subjects in cosmology, including the big bang, black holes and light comes, to the nonspecialist reader. His main goal is to give an overview of the subject, but he also attempts to explain some complex mathematics. In the 1996 edition of the book and subsequent editions, Hawking discusses the possibility of time travel and wormholes and explores the possibility of having a universe without a quantum singularity at the beginning of time."},{id:11,"title":"Sarpreet","author":"","price":100,"genre":"","publishDate":"12-12-12","description":"des"}, {"id":12,"title":"Title 1","author":"","price":100,"genre":"Genre 1","publishDate":"2012-12-12","description":"Description 1"}, {"id":13,"title":"","author":"","price":1000,"genre":"","publishDate":"100-100-100","description":"" dd}]
```

Below the browser window, the network tab shows the details of the request:

Status	Method	File	Domain	Cause	Type	Transfer...	Size	0 ms	160 ms	320 ms	480 ms	640 ms	799 ms
200	GET	books	localhost:9090	document	json	2,47 KB	5,81 KB						828 ms

ID	4	
Name	Should view an empty list of books	
Date	13 Mar 2017	
Use case covered	View a list of books	
Requirement covered	2.1.2	
Scenario	User send GET request URI (/api/books)	
Pre conditions	System must not have any book	
Post conditions	User should view an empty list of books	
Notes	Server should be up on port 9090	
Test Data	localhost:9090/api/books	
S.No	Steps	Expected Result
1	Open any browser and provide ‘Test Data’ in the search tab	User should view an empty list of books and server should response 200 OK

http://localhost:9090/api/books						
localhost:9090/api/books						
Slack MyMoodle Linnéuniversitetet ... GitHub Inbox - sb223ce@... Inbox - sunnybutt... ShareLaTeX Univers						
[ ]						
Inspector	Console	Debugger	Style Editor	Performance	Memory	Network
All	HTML	CSS	JS	XHR	Fonts	Images
Media	Flash	WS	Other			
Status	Method	File	Domain	Cause	Type	
200	GET	books	localhost:9090	document	json	

<b>ID</b>	5
<b>Name</b>	Should add a book
<b>Date</b>	13 Mar 2017
<b>Use case covered</b>	Add a book
<b>Requirement covered</b>	2.1.3
<b>Scenario</b>	User send PUT request URI (/api/books)
<b>Pre conditions</b>	None
<b>Post conditions</b>	User should get Response 200 OK
<b>Notes</b>	Server should be up on port 9090
<b>Test Data</b>	<p><i>URL</i> = localhost:9090/api/books</p> <p><i>BODY</i> = {"id": "12", "title": "Title", "author": "Author", "genre": "Genre", "price": "100", "publishDate": "2008-05-01", "description": "Description"}</p> <p>Content-Type: application/json</p>
<b>Recommended tool</b>	Postman

S.No	Steps	Expected Result
1	Open 'Postman', select request type PUT, enter 'Test Data'. Raw should be selected as body type.	The book should be added and server should response 200 OK

PUT

localhost:9090/api/books

Params

Send

Save

Authorization

Headers (1)

Body

Pre-request Script

Tests

Cookies

Code

form-data

x-www-form-urlencoded

raw

binary

JSON (application/json)

1

{ "id": "12", "title": "Title", "author": "Author", "genre": "Genre", "price": "100", "publishDate": "2008-05-01", "description": "Description" }

Body

Cookies

Headers (2)

Tests

Status: 200 OK

Time: 146 ms

Size: 75 B

ID	6	
Name	Should not add a book	
Date	13 Mar 2017	
Use case covered	Add a book	
Requirement covered	2.1.3	
Scenario	User send PUT request URI (/api/books)	
Pre conditions	None	
Post conditions	User should get Response 404 Not Found	
Notes	Server should be up on port 9090	
Test Data	localhost:9090/api/books	
Recommended tool	Postman	
S.No	Steps	Expected Result
1	Open “Postman”, select request type PUT, enter ‘Test Data’ in URL tab.	The book should not be added and server should response 404 Not Found

PUT

localhost:9090/api/books

Params

Send

Save

Authorization

Headers

Body

Pre-request Script

Tests

Cookies

Code

key

value

Bulk Edit

Presets

Body

Cookies

Headers (2)

Tests

Status: 404 Not Found

Time: 86 ms

Size: 82 B

<b>ID</b>	7
<b>Name</b>	Should edit a book
<b>Date</b>	13 Mar 2017
<b>Use case covered</b>	Edit a book
<b>Requirement covered</b>	2.1.4
<b>Scenario</b>	User send POST request URI (/api/books/{book_id})
<b>Pre conditions</b>	System should have requested book
<b>Post conditions</b>	User should get Response 200 OK and book should be updated
<b>Notes</b>	Server should be up on port 9090
<b>Test Data</b>	<p><i>URL</i> = localhost:9090/api/books</p> <p><i>BODY</i> = {"id": "11", "title": "Title", "author": "Author", "genre": "Genre", "price": "100", "publishDate": "2008-05-01", "description": "Description"}</p> <p>Content-Type: application/json</p>
<b>Recommended tool</b>	Postman

S.No	Steps	Expected Result
1	Open 'Postman', select request type POST, enter 'Test Data'. Raw should be selected as body type.	The book should be updated and server should response 200 OK

POST

localhost:9090/api/books/11

Params

Send

Save

Authorization

Headers (1)

Body

Pre-request Script

Tests

Cookies

Code

form-data

x-www-form-urlencoded

raw

binary

JSON (application/json)

1

{ "id": "11", "title": "Title", "author": "Author", "genre": "Genre", "price": "100", "publishDate": "2008-05-01", "description": "Description" }

Body

Cookies

Headers (2)

Tests

Status: 200 OK

Time: 176 ms

Size: 75 B

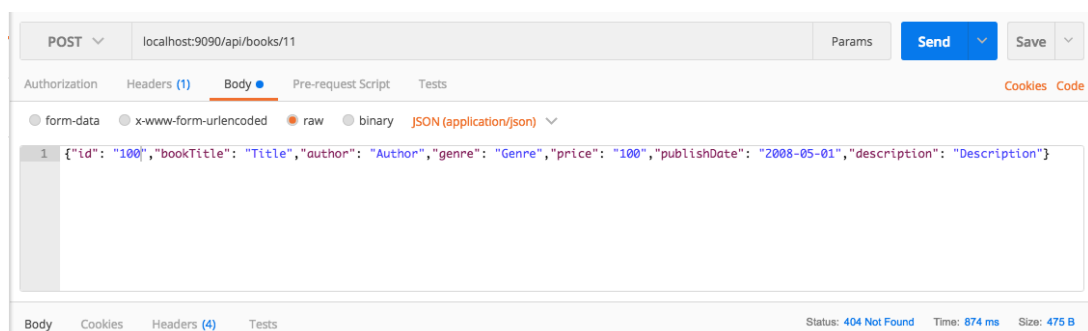
<b>ID</b>	8
<b>Name</b>	Should not edit a book when request body is invalid
<b>Date</b>	13 Mar 2017
<b>Use case covered</b>	Edit a book
<b>Requirement covered</b>	2.1.4
<b>Scenario</b>	User send POST request URI (/api/books/{book_id})
<b>Pre conditions</b>	System should have requested book
<b>Post conditions</b>	User should get Response 404 Not Found and book should not be updated
<b>Notes</b>	Server should be up on port 9090
<b>Test Data</b>	<p><i>URL</i> = localhost:9090/api/books</p> <p><i>BODY</i> = {"id": "11", "bookTitle": "Title", "author": "Author", "genre": "Genre", "price": "100", "publishDate": "2008-05-01", "description": "Description"}</p> <p>Content-Type: application/json</p>
<b>Recommended tool</b>	Postman

S.No	Steps	Expected Result
1	Open 'Postman', select request type POST, enter 'Test Data'. Raw should be selected as body type.	The book should not be updated and server should response 404 Not Found



<b>ID</b>	9
<b>Name</b>	Should not edit a book
<b>Date</b>	13 Mar 2017
<b>Use case covered</b>	Edit a book
<b>Requirement covered</b>	2.1.4
<b>Scenario</b>	User send POST request URI (/api/books/{book_id})
<b>Pre conditions</b>	System should not have requested book
<b>Post conditions</b>	User should get Response 404 Not Found
<b>Notes</b>	Server should be up on port 9090
<b>Test Data</b>	<p><i>URL</i> = localhost:9090/api/books</p> <p><i>BODY</i> = {"id": "100", "title": "Title", "author": "Author", "genre": "Genre", "price": "100", "publishDate": "2008-05-01", "description": "Description"}</p> <p>Content-Type: application/json</p>
<b>Recommended tool</b>	Postman

S.No	Steps	Expected Result
1	Open 'Postman', select request type POST, enter 'Test Data'. Raw should be selected as body type.	The server should response 404 Not Found





ID	10
Name	Should delete a book
Date	13 Mar 2017
Use case covered	Delete a book
Requirement covered	2.1.5
Scenario	User send DELETE request URI (/api/books/{book_id})
Pre conditions	System should have requested book
Post conditions	User should get Response 200 OK and book should be deleted
Notes	Server should be up on port 9090
Test Data	localhost:9090/api/books/1
Recommended tool	Postman

S.No	Steps	Expected Result
1	Open "Postman", select request type DELETE, enter 'Test Data'.	The book should be deleted and server should response 200 OK

DELETE
localhost:9090/api/books/1
Params
Send
Save

Authorization
Headers
Body
Pre-request Script
Tests
Cookies
Code

key
value
Bulk Edit
Presets

Body
Cookies
Headers (2)
Tests
Status: 200 OK
Time: 996 ms
Size: 75 B

ID	11	
Name	Should not delete a book	
Date	13 Mar 2017	
Use case covered	Delete a book	
Requirement covered	2.1.5	
Scenario	User send DELETE request URI (/api/books/{book_id})	
Pre conditions	System should not have requested book	
Post conditions	User should get Response 404 Not Found	
Notes	Server should be up on port 9090	
Test Data	localhost:9090/api/books/100	
Recommended tool	Postman	
S.No	Steps	Expected Result
1	Open "Postman", select request type DELETE, enter 'Test Data'.	The server should response 404 Not Found

DELETE localhost:9090/api/books/100 Params Send Save

Authorization Headers Body Pre-request Script Tests Cookies Code

key	value

Body Cookies Headers (2) Tests Bulk Edit Presets

Status: 404 Not Found Time: 138 ms Size: 82 B

<b>ID</b>	12
<b>Name</b>	Should search books
<b>Date</b>	13 Mar 2017
<b>Use case covered</b>	Search a book by author or title
<b>Requirement covered</b>	2.1.6
<b>Scenario</b>	User send GET request URI (/api/books/{title})
<b>Pre conditions</b>	System should have requested books
<b>Post conditions</b>	User should get a JSON array of books and response 200 OK
<b>Notes</b>	Server should be up on port 9090
<b>Test Data</b>	localhost:9090/api/books/?title=title

S.No	Steps	Expected Result
1	Open any browser and enter 'Test Data'	The user should get a JSON array of books and server should response 200 OK

The screenshot shows a web browser window with the address bar displaying 'localhost:9090/api/books/?title=title'. The page content shows a JSON array of books. Below the browser window, the Chrome DevTools Network tab is open, showing a single request to 'localhost:9090' with a status of 200, method of GET, and a response size of 142 B. The response is a JSON document.

Status	Method	File	Domain	Cause	Type	Transfer...	Size	0 ms	80 ms	160 ms	240 ms	320 ms
200	GET	books?title=title	localhost:9090	document	json	142 B	288 B		→ 52 ms			

<b>ID</b>	13
<b>Name</b>	Should get no search result
<b>Date</b>	13 Mar 2017
<b>Use case covered</b>	Search a book by author or title
<b>Requirement covered</b>	2.1.6
<b>Scenario</b>	User send GET request URI (/api/books/{title})
<b>Pre conditions</b>	System should not have requested books
<b>Post conditions</b>	User should get response 404 Not Found
<b>Notes</b>	Server should be up on port 9090
<b>Test Data</b>	localhost:9090/api/books/?title=abc

S.No	Steps	Expected Result
1	Open any browser and enter 'Test Data'	The server should response 404 Not Found

Error 404 Not Found

localhost:9090/api/books?title=abc

Slack MyMoodle Linnéuniversitetet ... GitHub Inbox - sb223ce@... Inbox - sunnybutt... ShareLaTeX Universityadmissi... YouTube

## HTTP ERROR 404

Problem accessing /api/books. Reason:

Not Found

Powered by Jetty://

Status	Method	File	Domain	Cause	Type	Transfer...	Size
404	GET	books?title=abc	localhost:9090	document	html	293 B	293 B

## **8 Test Evaluation/Result**

All the tests(Function, Unit and API) test have been successfully passed.