BOOK STORE MANAGEMENT SYSTEM USING SPRING BOOT, THYMELEAF AND MYSQL DB

OBJECTIVE

To demonstrate the basic concepts of Spring Boot, Thymeleaf along with Book Store Management System using MySQL DB

ABSTRACT

Backend

- Spring Boot Application
  - Model
  - Repository
  - Controller
  - Configuration

Frontend

- Thymeleaf

Database

- MySQL

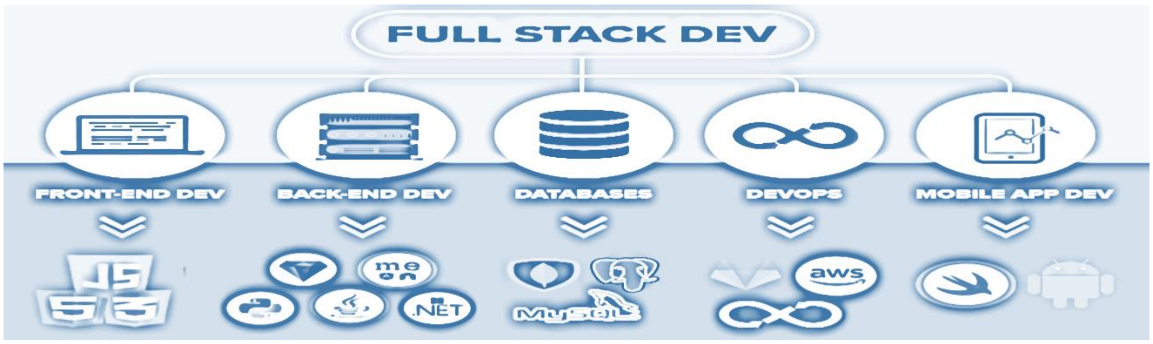
Authors:
Mr.B.Muthukrishna Vinayagam,
AP-CSE

Contents

Web Application Development

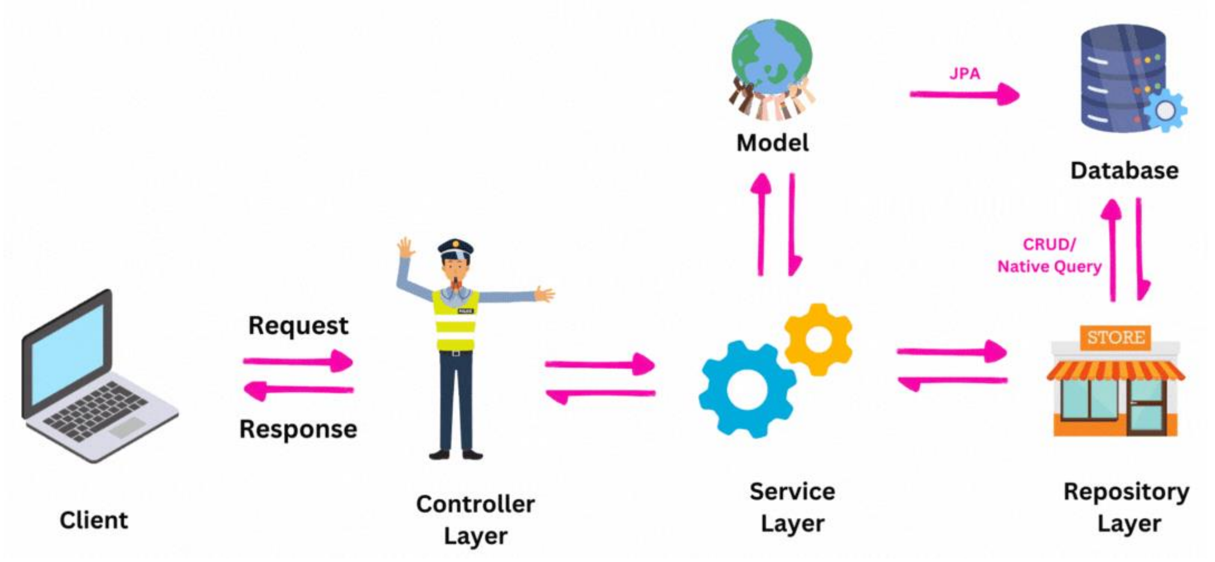
- Web application development is the creation of application programs that reside on remote/local servers and are delivered to the user's device over the Internet / Intranet.

- Web technologies like HTML (Hypertext Markup Language), CSS (Cascading Style Sheets), and JavaScript, which are executed by the browser, are used to create web applications.
- A full-stack developer needs to be proficient in both front-end and back-end of a website



Spring Boot

- Java Spring Framework (Spring Framework) is a popular, open source, enterprise-level framework for creating standalone, production-grade applications that run on the Java Virtual Machine (JVM).
- Java Spring Boot (Spring Boot) is a tool that makes developing web application and microservices with Spring Framework faster and easier through three core capabilities:
    1. Autoconfiguration
    2. An opinionated approach to configuration
    3. The ability to create standalone applications
- These features work together to provide you with a tool that allows you to set up a Spring-based application with minimal configuration and setup. Spring Boot applications can also be optimized and run with the Open Liberty runtime."

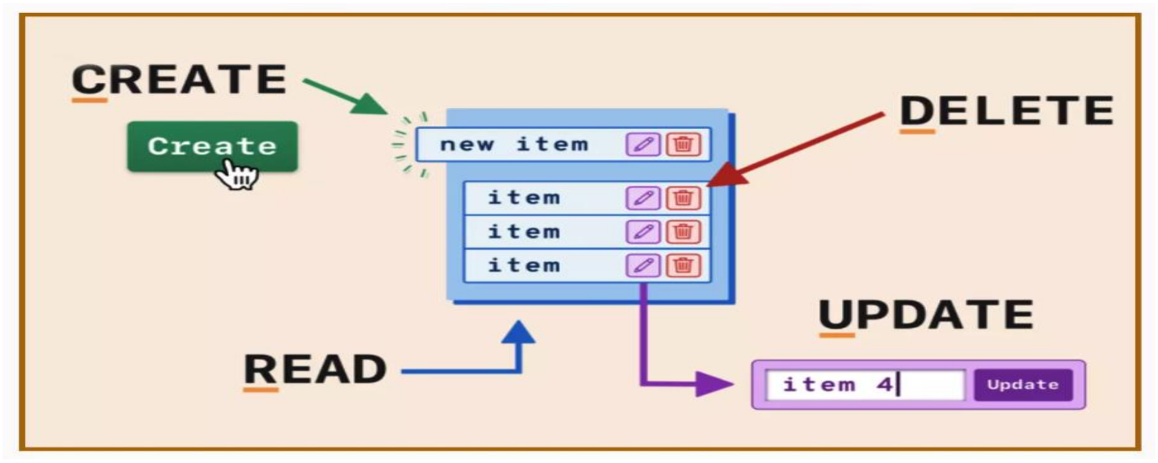


Thymeleaf

- The Thymeleaf is an open-source Java library that is licensed under the Apache License 2.0. It is a HTML5/XHTML/XML template engine. It is a server-side Java template engine for both web (servlet-based) and non-web (offline) environments. It is perfect for modern-day HTML5 JVM web development. It provides full integration with Spring Framework.
- It applies a set of transformations to template files in order to display data or text produced by the application. It is appropriate for serving XHTML/HTML5 in web applications.
- The goal of Thymeleaf is to provide a stylish and well-formed way of creating templates. It is based on XML tags and attributes. These XML tags define the execution of predefined logic on the DOM (Document Object Model) instead of explicitly writing that logic as code inside the template. It is a substitute for JSP.
- The architecture of Thymeleaf allows the fast processing of templates that depends on the caching of parsed files. It uses the least possible amount of I/O operations during execution.

CRUD

- CRUD is an acronym for **C**reate, **R**ead, **U**pdate, and **D**elete. CRUD operations are basic data manipulation for the database
    - Create (C): The create operation involves adding new data to the storage system. i.e., Inserting a new record or document into a table or collection.
    - Read (R): The read operation retrieves existing data from the storage system. It allows you to fetch and view the data.
    - Update (U): The update operation modifies existing data in the storage system. It allows you to make changes to specific records or documents
    - Delete (D): The delete operation removes data from the storage system. It involves the permanent deletion of records or documents.



Project Summary

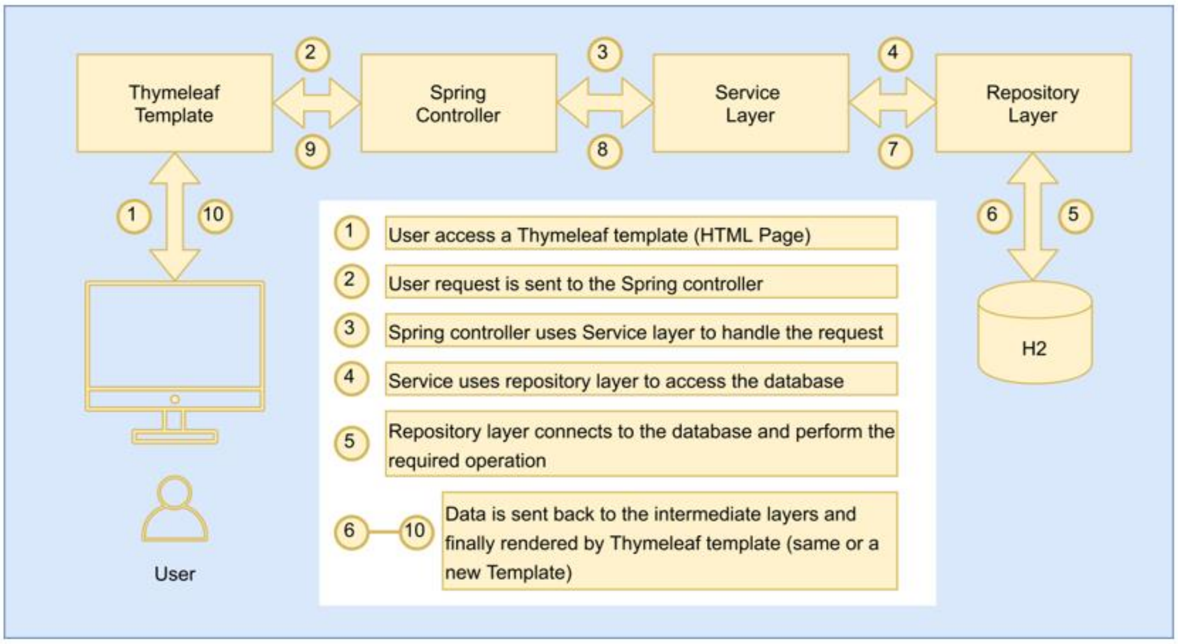
Problem Statements

- The traditional methods of managing a book store involve manual processes that are time-consuming, error-prone, and lack efficiency. In order to overcome these challenges and enhance the overall management of the book store, there is a need for a comprehensive Book Store Management System (BSMS) that integrates modern technology to streamline operations, improve accuracy, and provide a better customer experience.

Requirements

- Software Requirements
    - Eclipse IDE
    - Spring Boot Framework
    - Hibernate Framework
    - Thymeleaf

Proposed Architecture


Thymeleaf Template
Spring Controller
Service Layer
Repository Layer
H2
User
1 User access a Thymeleaf template (HTML Page)
2 User request is sent to the Spring controller
3 Spring controller uses Service layer to handle the request
4 Service uses repository layer to access the database
5 Repository layer connects to the database and perform the required operation
6—10 Data is sent back to the intermediate layers and finally rendered by Thymeleaf template (same or a new Template)

Methodology

- Requirement Gathering
- System Design
- Development
- Testing
- Deployment

Setting Up New Spring Boot Project, Dependency, Development and Deployment

- Create new Spring Boot project in Eclipse IDE
  - New→others →Spring Web Project
  - Add Dependency
    - JPA
    - H2
    - Thymeleaf
    - Spring Boot
  - In Project Folder, Right click → RunAs → Spring → Spring Boot App (i.e DemoApplication.java)

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

@SpringBootApplication
public class DemoApplication {

	public static void main(String[] args) {
		SpringApplication.run(DemoApplication.class, args);
	}

}
```

Working with Model

- Create the package as "Model" and store Book.java
- In Book.java, create new book model with the following fields: id (auto increment), title, author

#Book.java

```
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Book {
```

```java
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    private String author;

    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }
    public String getAuthor() {
        return author;
    }
    public void setAuthor(String author) {
        this.author = author;
    }

}
```

Working with Repository

- Create the package as “Repository” and store BookRepository.java
- In BookRepository.java, create the interface that extends from JpaRepository **<Book, Long>** (i.e) Template that receives Book Object and Id from Book.java

```java
import org.springframework.data.jpa.repository.JpaRepository;

import com.example.demo.model.Book;

public interface BookRepository extends JpaRepository<Book, Long> {

}
```

Working with Service

- Create the package as “Service” and store BookService.java
- In BookService.java, create the modules for CRUD operations using BookRepository Interface

```java
import java.util.List;

import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.example.demo.model.Book;
import com.example.demo.repository.BookRepository;


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

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

    public Book getBookById(Long id) {
        return bookRepository.findById(id).orElse(null);
    }

    public void saveBook(Book book) {
        bookRepository.save(book);
    }

    public void deleteBook(Long id) {
        bookRepository.deleteById(id);
    }

}
```

Working with Controller

- Create the package as “Controller” and store BookController.java
- In BookController.java, create the REST API end points for getting the BookService

```java
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;

import com.example.demo.model.Book;
import com.example.demo.service.BookService;

@Controller
public class BookController {
    @Autowired
    private BookService bookService;

    @GetMapping("/")
    public String showIndex() {
        return "index";
    }
    @GetMapping("/books")
    public String getAllBooks(Model model) {
        List<Book> books = bookService.getAllBooks();
        model.addAttribute("books", books);
        return "book-list";
    }

    @GetMapping("/books/{id}")
    public String getBookById(@PathVariable Long id, Model model) {
        Book book = bookService.getBookById(id);
        model.addAttribute("book", book);
        return "book-detail";
    }
    @GetMapping("/books/new")
    public String showNewBookForm(Model model) {
        model.addAttribute("book", new Book());
        return "new-book";
    }

    @PostMapping("/books/new")
    public String saveBook(@ModelAttribute("book") Book book) {
        bookService.saveBook(book);
        return "redirect:/books";
    }

    @GetMapping("/books/delete/{id}")
    public String deleteBook(@PathVariable Long id) {
        bookService.deleteBook(id);
        return "redirect:/books";
    }

    @GetMapping("/books/edit/{id}")
    public String showEditForm(@PathVariable Long id, Model model) {
        Book book = bookService.getBookById(id);
        model.addAttribute("book", book);
        return "edit-book";
    }

    @PostMapping("/books/edit/{id}")
    public String editBook(@PathVariable Long id,
@ModelAttribute("book") Book updatedBook) {
        Book existingBook = bookService.getBookById(id);
        if (existingBook != null) {
            existingBook.setTitle(updatedBook.getTitle());
          existingBook.setAuthor(updatedBook.getAuthor());
            bookService.saveBook(existingBook);
        }
        return "redirect:/books";
    }
}
```

Working with Configuration

- Create the package as "Config" and store ThymeleafConfig.java
- In ThymeleafConfig.java, setting up the thymeleaf resolver for templates (html files) and static (CSS, JS and media files) folder

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import
org.thymeleaf.spring6.templateresolver.SpringResourceTemplateResolver;


@Configuration
public class ThymeleafConfig implements WebMvcConfigurer {

    @Bean
    public SpringResourceTemplateResolver templateResolver() {
        SpringResourceTemplateResolver templateResolver = new
SpringResourceTemplateResolver();
        templateResolver.setPrefix("classpath:/templates/");
        templateResolver.setSuffix(".html");
        templateResolver.setTemplateMode("HTML");
        templateResolver.setCharacterEncoding("UTF-8");
        templateResolver.setCacheable(false); // Set to true for
production
        return templateResolver;
    }

    // Additional configuration if needed

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {

registry.addResourceHandler("/static/**").addResourceLocations("classpath:
/static/");
    }
}
```

Configuring Backend DBs, Hihernate and Thymeleaf

- In application.properties, configure DB, hibernate, thymeleaf,

```
# Database Configuration
spring.datasource.url=jdbc:mysql://localhost:3306/bookdb
spring.datasource.username=root
spring.datasource.password=1985$univ
spring.datasource.driver-class-
name=com.mysql.cj.jdbc.Driver

# Hibernate Configuration
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

# Thymeleaf Configuration
spring.thymeleaf.cache=false
spring.thymeleaf.enabled=true
spring.thymeleaf.check-template-location=false
spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html
```

Working with Templates using Thymeleaf

- In templates folder, include the html files for requesting and rendering the results.

#index.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Spring Boot CRUD App</title>
    <link rel="stylesheet" th:href="@{css/style.css}" />
</head>
<body>
    <h1>Welcome to the BOOK CRUD Application</h1>
    <a th:href="@{/books}">Go to Book List</a>
</body>
</html>
```

#book-list.html

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Book List</title>
    <link rel="stylesheet" th:href="@{css/style.css}" />
     <script th:src="@{js/jquery.min.js}"></script>
</head>

<body>
    <h2>Book List</h2>
    <script>
      $(document).ready(function(){
      $("#myInput").on("keyup", function() {
        var value = $(this).val().toLowerCase();
        $("#myTable tr").filter(function() {
          $(this).toggle($(this).text().toLowerCase().indexOf(value) > -1)
        });
      });
    });
    </script>
    <center>
    Search Here <input type="search" id="myInput"/>
    <table border="1" id="myTable" >
        <thead>
            <tr>
                <th>ID</th>
                <th>Title</th>
                <th>Author</th>
                <th>Action</th>
            </tr>
        </thead>
        <tbody>
            <tr th:each="book : ${books}">
                <td th:text="${book.id}"></td>
                <td th:text="${book.title}"></td>
                <td th:text="${book.author}"></td>
                <td>
                     <a th:href="@{'/books/edit/' + ${book.id}}">Edit</a>
                    <a th:href="@{'/books/delete/' + ${book.id}}">Delete</a>
                </td>
            </tr>
        </tbody>
    </table>
    </center>
    <a href="/books/new">Add New Book</a>
</body>
</html>
```

#new-book.html

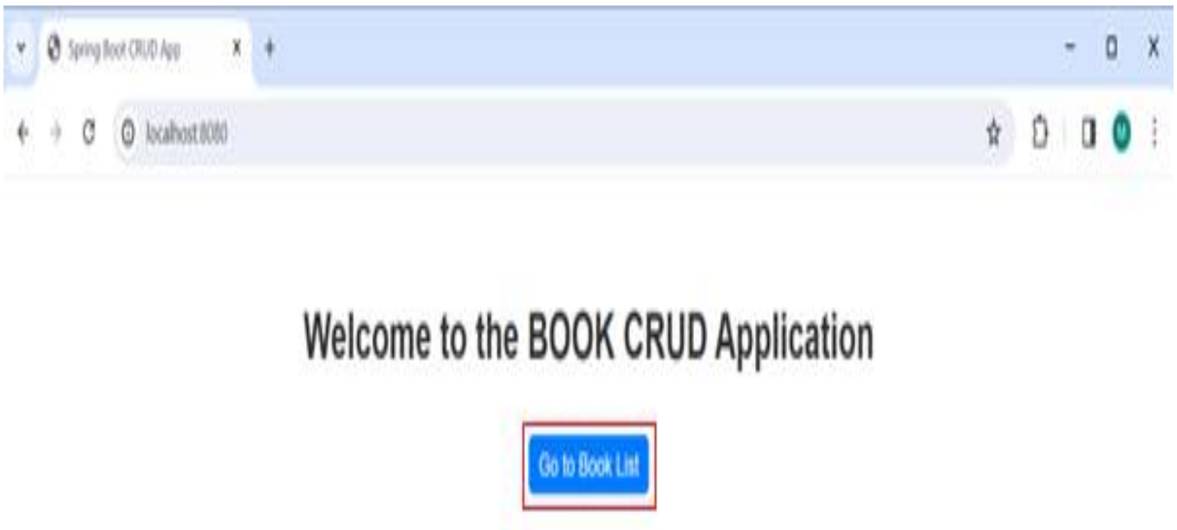
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Add New Book</title>
<link rel="stylesheet" th:href="@{css/style.css}" />
<link rel="stylesheet" th:href="@{css/bootstrap.min.css}" />
</head>
<body>
    <h2>Add New Book</h2>
    <form th:action="@{/books/new}" th:object="${book}" method="post" >
        <label for="title">Title:</label>
    <input type="text" id="title" name="title" th:field="*{title}" required />
        <br/>
        <label for="author">Author:</label>
 <input type="text" id="author" name="author" th:field="*{author}" required />
        <br/>
        <button  type="submit">Save</button>
    </form>
    <a href="/books">Cancel</a>
</body>
</html>
```

#edit-book.html

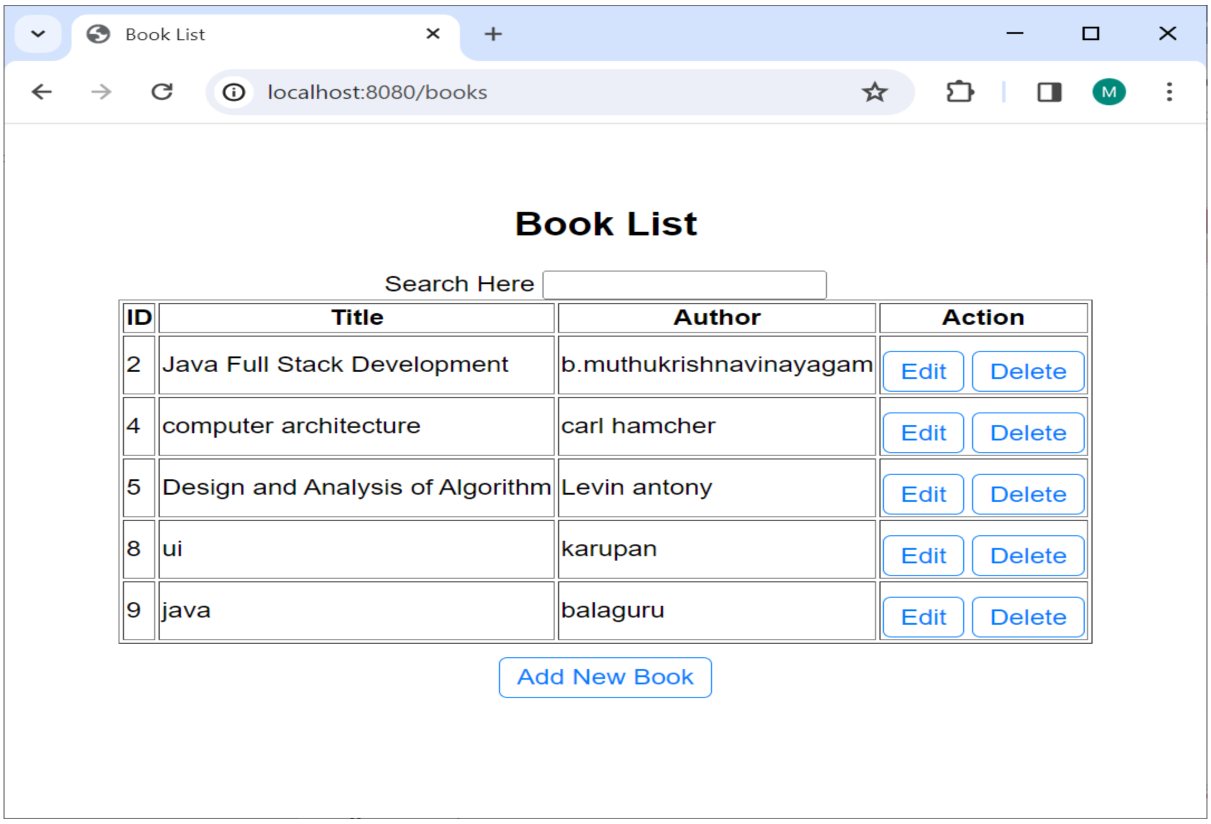
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Edit Book</title>
    <link rel="stylesheet" th:href="@{css/style.css}" />

</head>
<body>
    <h2>Edit Book</h2>
<form th:action="@{'/books/edit/' + ${book.id}}" th:object="${book}" method="post">
        <input type="hidden" th:field="*{id}" />
        <label for="title">Title:</label>
   <input type="text"  id="title" name="title" th:field="*{title}" required />
        <br/>
        <label for="author">Author:</label>
<input type="text"  id="author" name="author" th:field="*{author}" required />
        <br/>
        <button  type="submit">Save Changes</button>
    </form>
    <a href="/books">Cancel</a>
</body>
</html>
```

Result and Discussion

- By clicking the Go to Book List button displays all the book details



- List all the book details like id, title and author along with action (Edit & Delete)



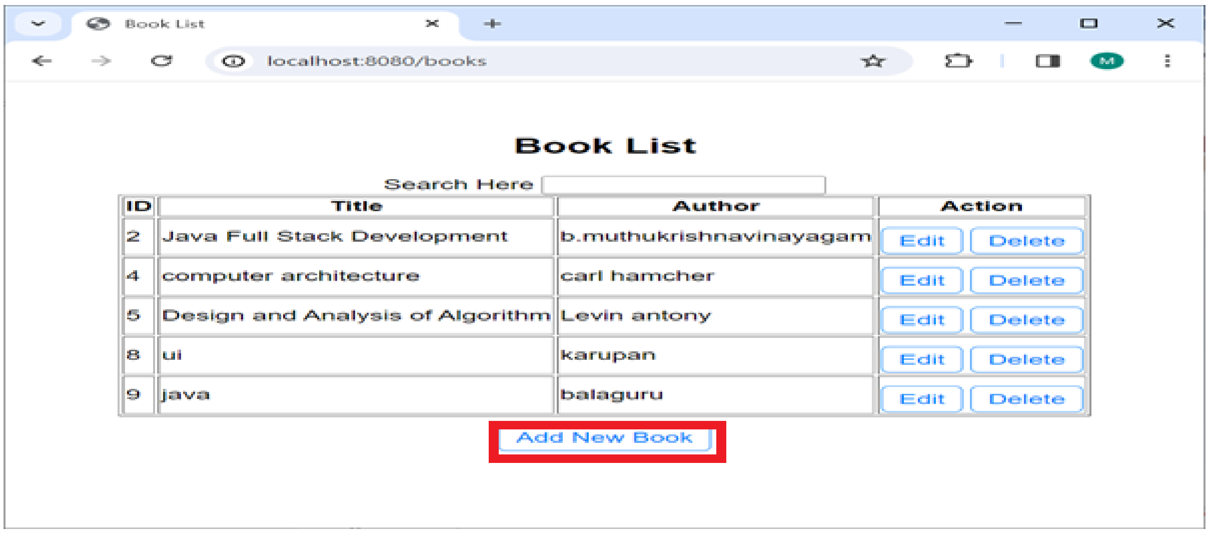
Book List

Search Here

ID	Title	Author	Action
2	Java Full Stack Development	b.muthukrishnavinayagam	Edit Delete
4	computer architecture	carl hamcher	Edit Delete
5	Design and Analysis of Algorithm	Levin antony	Edit Delete
8	ui	karupan	Edit Delete
9	java	balaguru	Edit Delete

Add New Book

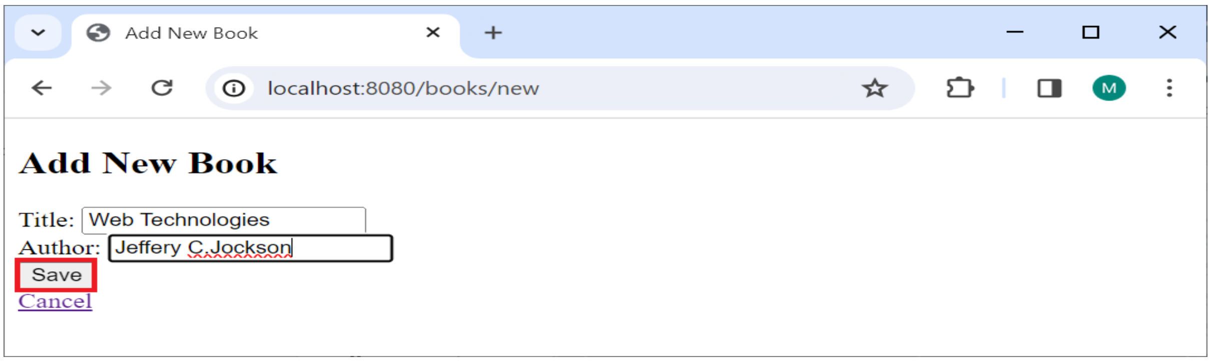
- To insert a new book detail by clicking Add New Book button



Book List

Search Here

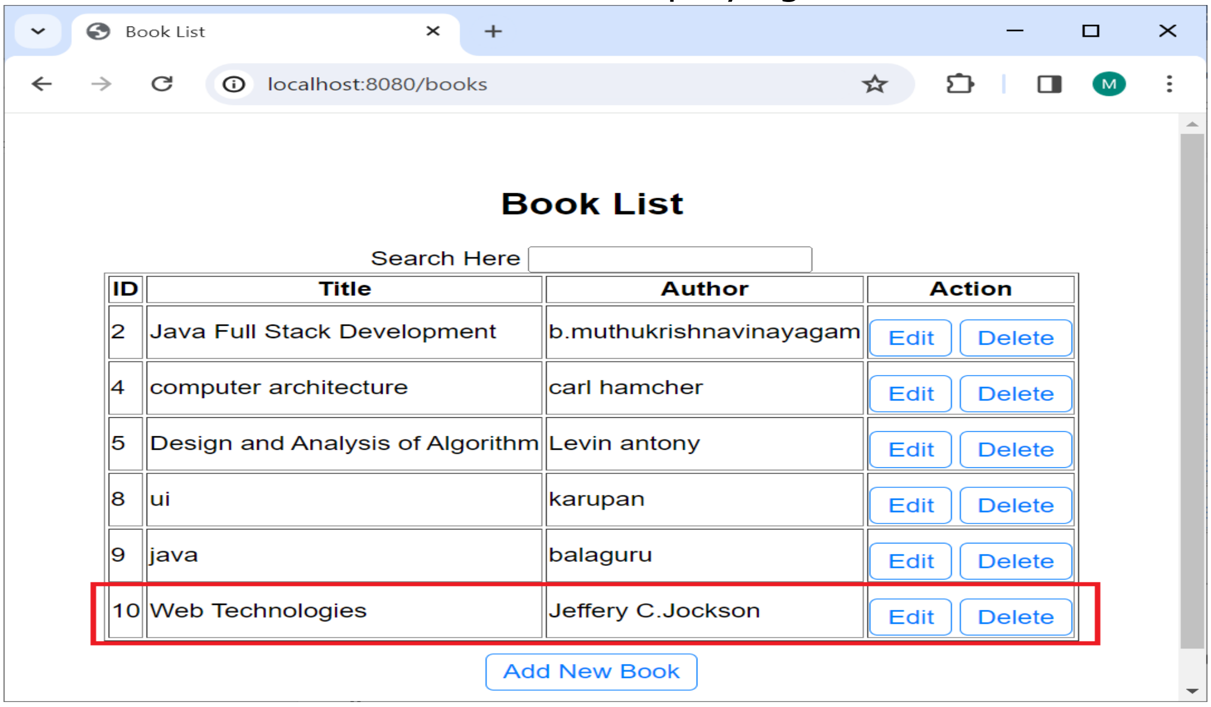
ID	Title	Author	Action
2	Java Full Stack Development	b.muthukrishnavinayagam	Edit Delete
4	computer architecture	carl hamcher	Edit Delete
5	Design and Analysis of Algorithm	Levin antony	Edit Delete
8	ui	karupan	Edit Delete
9	java	balaguru	Edit Delete



Add New Book

Title: Web Technologies
Author: Jeffery C.Jockson

- Then new book details are inserted successfully and it redirects to book-list for displaying result like.

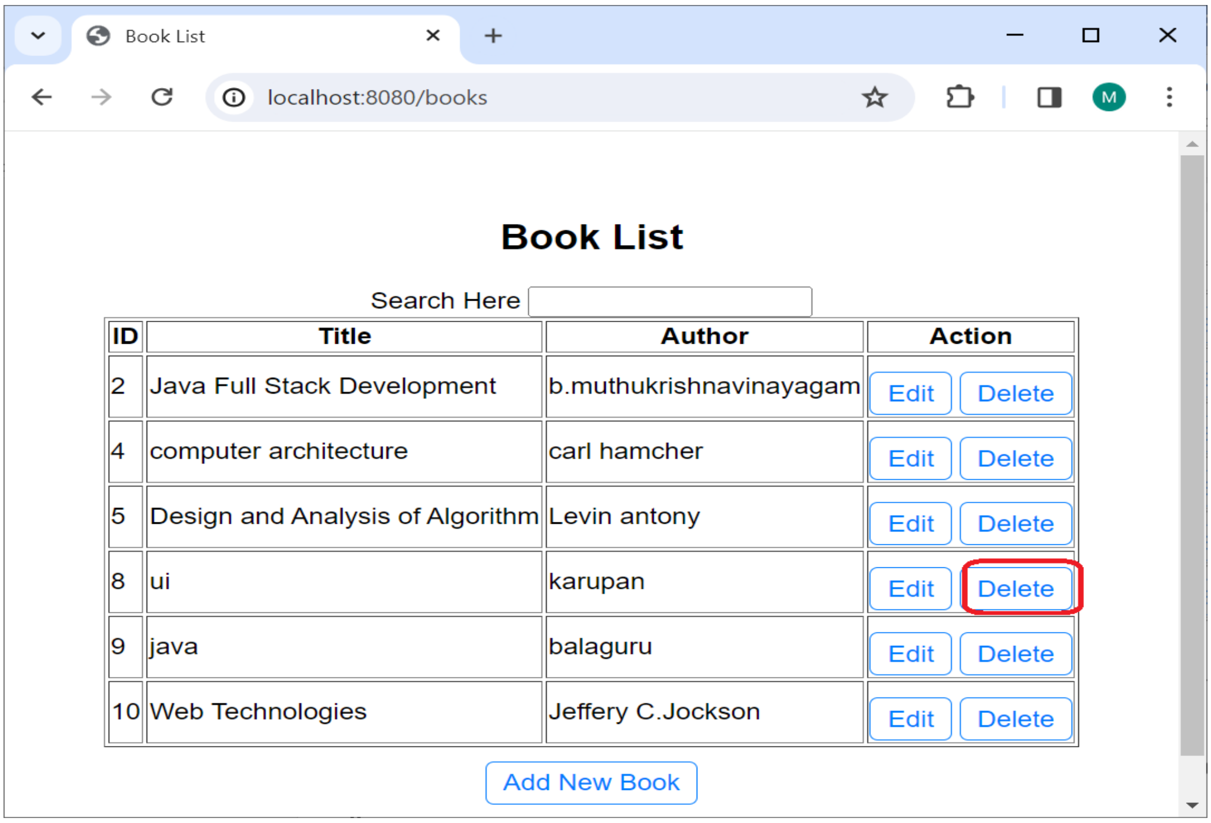


Book List

Search Here

ID	Title	Author	Action
2	Java Full Stack Development	b.muthukrishnavinayagam	Edit Delete
4	computer architecture	carl hamcher	Edit Delete
5	Design and Analysis of Algorithm	Levin antony	Edit Delete
8	ui	karupan	Edit Delete
9	java	balaguru	Edit Delete
10	Web Technologies	Jeffery C.Jockson	Edit Delete

- On clicking delete hyperlink, to delete the record from MySQL DB

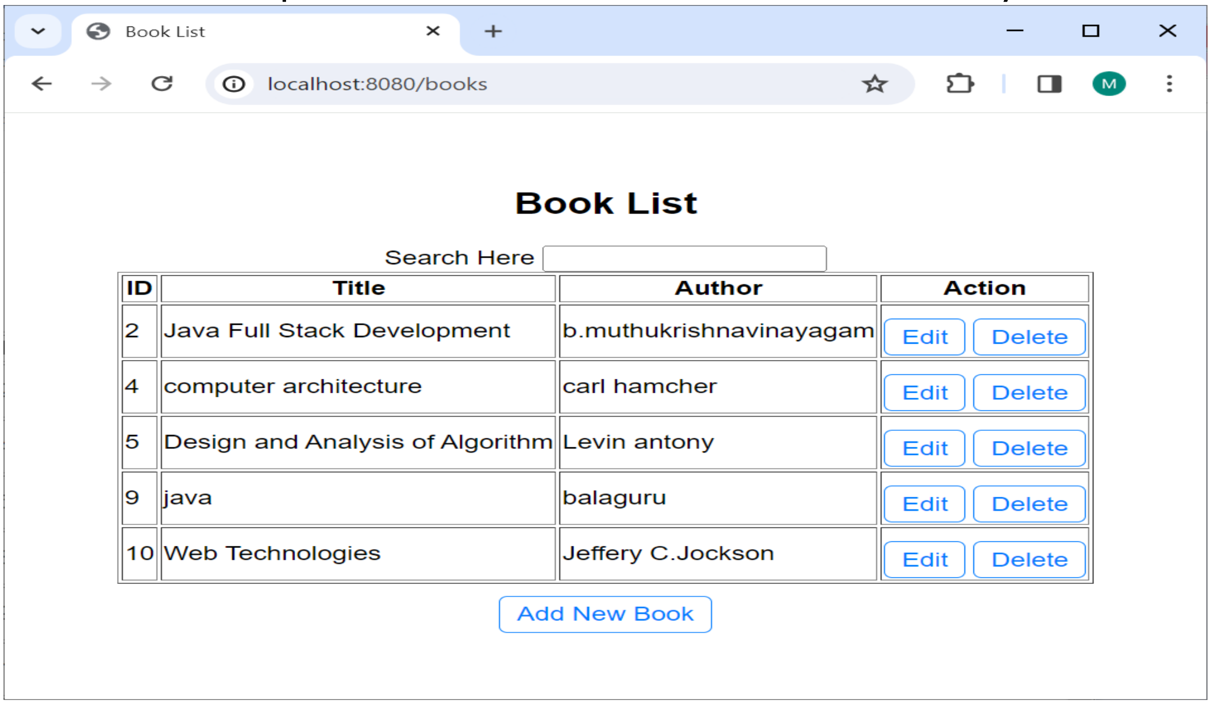


Book List

Search Here

ID	Title	Author	Action
2	Java Full Stack Development	b.muthukrishnavinayagam	Edit Delete
4	computer architecture	carl hamcher	Edit Delete
5	Design and Analysis of Algorithm	Levin antony	Edit Delete
8	ui	karupan	Edit Delete
9	java	balaguru	Edit Delete
10	Web Technologies	Jeffery C.Jockson	Edit Delete

- Then the particular record is deleted successfully

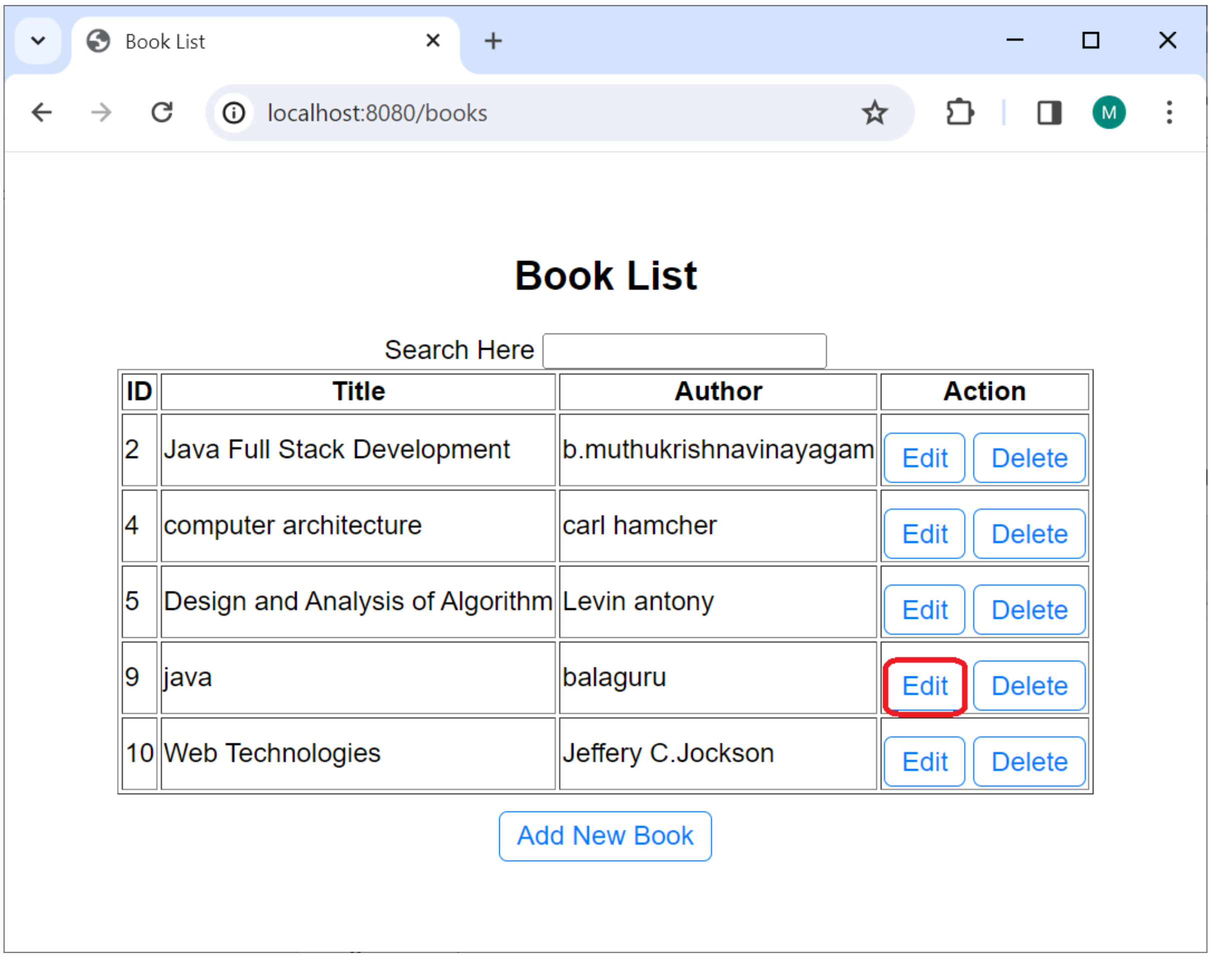


Book List

Search Here

ID	Title	Author	Action
2	Java Full Stack Development	b.muthukrishnavinayagam	Edit Delete
4	computer architecture	carl hamcher	Edit Delete
5	Design and Analysis of Algorithm	Levin antony	Edit Delete
9	java	balaguru	Edit Delete
10	Web Technologies	Jeffery C.Jockson	Edit Delete

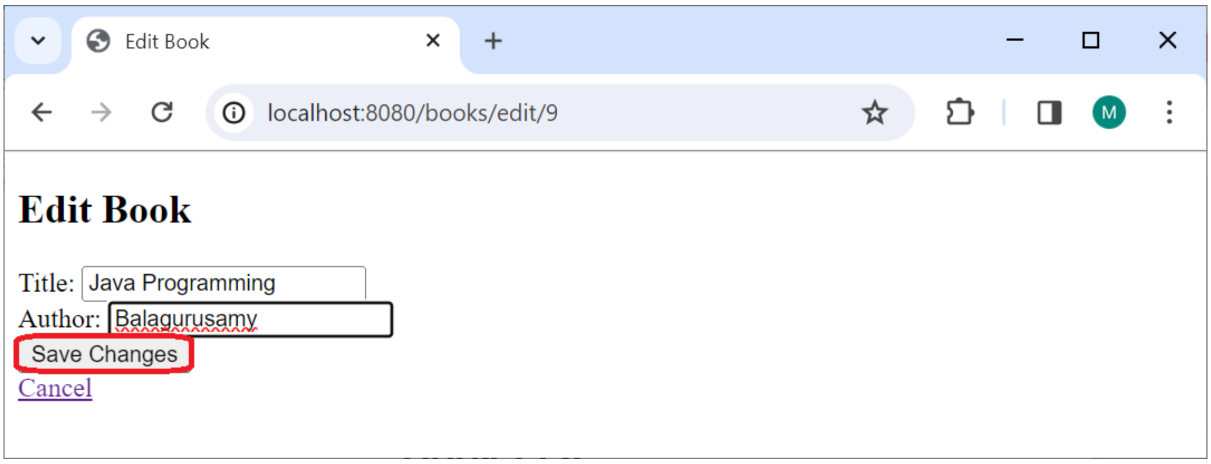
- On clicking edit hyperlink, The content of current record is loaded on edit.html and updating the current record by clicking save button.



Book List

Search Here

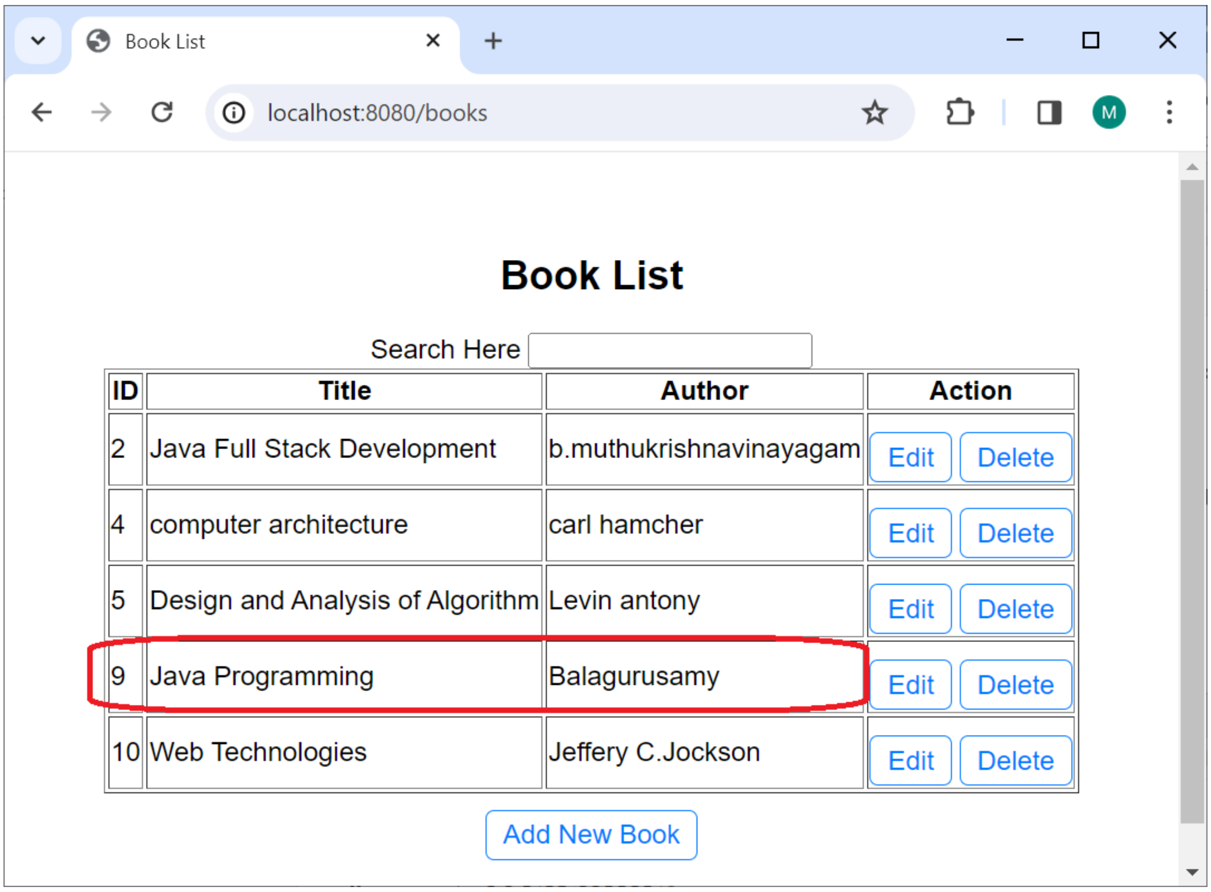
ID	Title	Author	Action
2	Java Full Stack Development	b.muthukrishnavinayagam	Edit Delete
4	computer architecture	carl hamcher	Edit Delete
5	Design and Analysis of Algorithm	Levin antony	Edit Delete
9	java	balaguru	Edit Delete
10	Web Technologies	Jeffery C.Jockson	Edit Delete



Edit Book

Title: Java Programming
Author: Balagurusamy

- Then particular record is updated successfully.



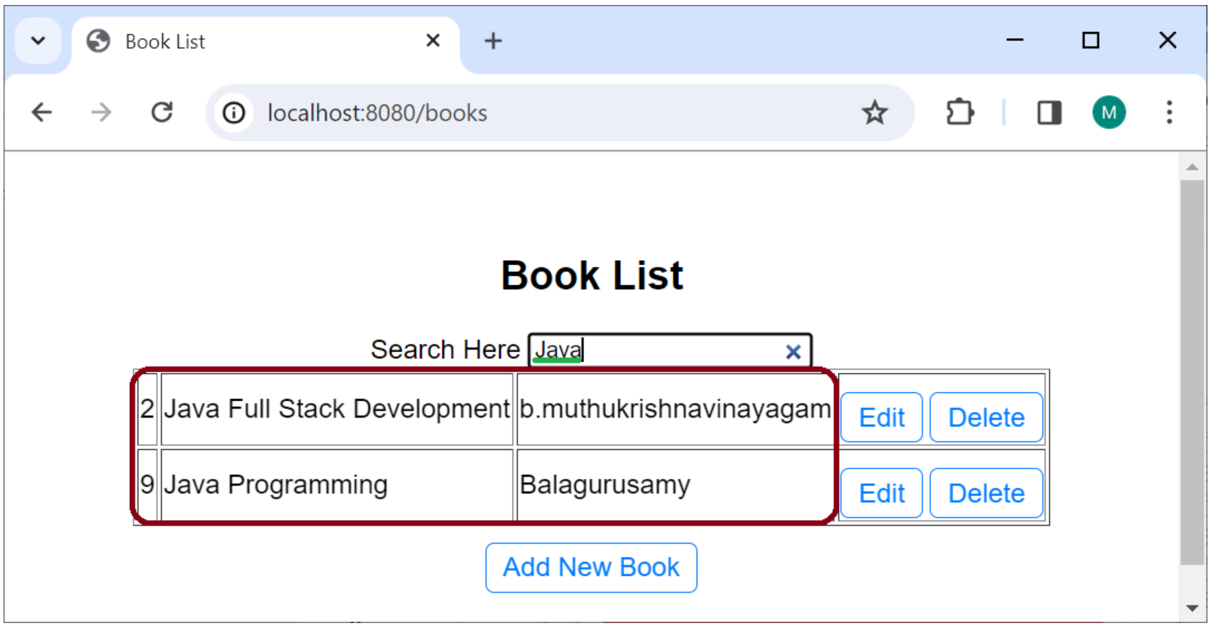
Book List

Search Here

ID	Title	Author	Action
2	Java Full Stack Development	b.muthukrishnavinayagam	Edit Delete
4	computer architecture	carl hamcher	Edit Delete
5	Design and Analysis of Algorithm	Levin antony	Edit Delete
9	Java Programming	Balagurusamy	Edit Delete
10	Web Technologies	Jeffery C.Jockson	Edit Delete

Add New Book

- Searc Here option to filter the records and display using JQuery



Book List

Search Here Java

2	Java Full Stack Development	b.muthukrishnavinayagam	Edit Delete
9	Java Programming	Balagurusamy	Edit Delete

Add New Book