3-2019

Applets vs JavaScript

Smitha Katta
skatta@stcloudstate.edu

Applets vs JavaScript

by

Smitha Katta

A Starred Paper

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree

Master of Science in

Information Assurance

March, 2019

Abstract

Client-side scripting languages are used to design the interactive web pages used by consumers to view web applications. With the tremendous growth in technology, numerous client-side scripting languages are being introduced into daily life. However, applets were the first technology introduced to create dynamic web pages and were the popular programming for over a decade. Later, JavaScript was introduced for the same purpose and is the most popular scripting language used by the developers today.

Both the applets and JavaScript have their advantages and disadvantages. The usage of these two technologies is dependent on the performance and other general requirements of the web application that is being developed. Hence, this paper compares JavaScript and applets concerning network, memory, performance, and security to help developers find the most efficient use between these two technologies based on the requirement of the application. This paper will also discuss other features that need to be considered while developing the application.

Table of Contents

List of Tables

List of Figures

Figure Page

Chapter I: Introduction

Introduction

With the massive growth in usage of web applications in day to day life, many technologies are introduced to design web applications such as HTML, JavaScript, applets, etc. The web pages that we see daily come from introducing static web pages using scripting languages. However, these markup languages instructed the web browser to view the web application in a certain way and failed to perform few operations on the web application, after which dynamic scripting languages such as JavaScript and applets were introduced.

The basic idea of creating interactive web pages is a few decades old and was introduced with the help of Java applets. An applet is a small program that can be embedded in an HTML page to make the browser interactive with the servers. Applets offer many capabilities, such as interaction with the server, gaming, etc. Later after few years, JavaScript was utilized to create dynamic web pages. With the advantage of lightweight technology and other key features in JavaScript, applets were depreciated, and the use of JavaScript has since increased. However, applets were many times faster than JavaScript until 2011, and applets have access to 3D hardware acceleration, making them well-suited for visualizations. Applets require a Java Runtime environment to execute Java in the browser, due to which applets have been depreciated and used mostly in a few specific applications like gaming.

This study gives an overview of creating web applications using the traditional framework such as applets and the advanced framework such as JavaScript. The web application that is being developed to compare is a Book Store Management System Application which allows the users to log in, view the books based on the category and availability, and then order the books available. The first web application this paper considers was created with Spring MVC architecture, with JavaScript providing client-side scripting and the Spring framework providing the server-side scripting. The second web application considered was created with Applets providing the client-side scripting in addition to utilizing Servlets for interaction with front-end and database. This study will compare these two applications in terms of network, security, performance, memory, etc. and discusses the advantages and disadvantages of using either application. This project also discusses the possible scenarios and business requirements to which these two applications can be applied, i.e., when to use Applets and when to use JavaScript while creating the web application.

Problem Statement

Choosing a client-side technology for designing and developing a web application depends on multiple factors, such as:

- What type of web application is being developed (i.e., consumer-facing application or business application)?
- What environments will the application run? For example, will the application run on mobile or is it only a web application?
- How much of the application is client-side?
- What is the number of application users?
- Will the application need to be visually displayed?

However, when any web application is being developed, JavaScript is most often used even though applets still have many advantages and can be used in a variety of applications. Usage of Applets is drastically reduced and almost ignored by most of the developers and architects after JavaScript is introduced. This is because there is no specific source available that compares or discusses the best architecture to be used based on the application requirement.

Hence, comparison of Applets and JavaScript concerning the above categories is needed to find out which client-side technology is to be utilized while developing a web application. This comparison individually addresses each aspect mentioned above, which is likely to help any developer or architect to choose the most appropriate option based on the requirements of an application.

Nature and Significance of the Problem

This study is designed to assist developers and architects in finding the best framework to be used between Applets and JavaScript to provide better performance while developing a web application or even a stand-alone application.

Significance of the Factors Chosen for Comparison of the Technologies

This study compares these technologies in terms of network, memory, security, performance and other factors, i.e., challenges faced while developing the application, general features are also taken into consideration.

Why security, memory, network, and performance are considered.

***Significance of security*:** Most web applications store significant data from users like personal information, payment details, etc. Therefore, it is very important to consider and analyze the security architecture, framework, or the security techniques that are used in each technology.

Significance of memory**:** The goal of any architect or developer is to create web applications with less heap memory usage and fewer memory leaks, as these issues impact the application response time.

Significance of network**:** Analyzing the network is also one of the significant factors, as the web applications are accessed by users over the network with many varying factors. For example, the time required to load each web element may fluctuate while accessing an application over the network. Consequently, comparing the application over the network is crucial.

Significance of performance**:** Performance analysis includes analyzing the general availability of the web page and responsiveness of the application to provide the users with a high-quality experience.

Objective of the Study

The primary objective of the project is to compare Applets and JavaScript in terms of network, performance, memory, and security. To do this, two web applications have been created with the same web design and database design wherein one application is developed with JavaScript and the other application with Applets.

Study Questions/Hypotheses

- How to develop a web application using Applets?
- How to develop a web application using JavaScript?
- What is the web application that is being created?
- What are the web pages that are being compared?
- What is the tool used to compare the web application?

- What are the factors that are considered to compare the web pages?
- Why are these factors taken into consideration for comparison?

Limitations of the Study

This study compared only the home page of applications created using JavaScript and Applets. This study compares only JavaScript and Applets; any other client-side technologies will be ignored, although many JavaScript frameworks are also increasing slowly and can be further taken into consideration in the future to elaborate on this study.

Since the applications are deployed on localhost, the number of users considered for the application is generally one. They can further be analyzed by deploying the application onto the network and increasing the number of users.

Definition of Terms

Applets: Applets are small Java applications which run on the browser with Java enabled which can be downloaded by any computer. (Wikipedia, n.d.)

JSP: Java Server Pages (JSP) is a technology that helps software developers create dynamically generated web pages based on HTML, XML or other document types. (Wikipedia, n.d.)

JavaScript: JavaScript is a scripting language that is commonly used to create dynamic web pages within web browsers (Mario Callegaro, 2015).

MVC: Model–view–controller (MVC) is a common software architectural pattern to develop any web applications or to implement user interfaces on computers (Wikipedia, n.d.).

Summary

The Book Store Management System allows users to log in, view the books available, and to order the books online. This web application has been created twice with the same database design and application design, but each application utilizes a different client-side technology to compare those technologies with the goal of finding the best technology between the above two based on the factors mentioned in the problem statement. The first application is created using Applets as the client-side programming, whereas the second application uses JavaScript technology as the client-side programming. The home page for each respective application is compared further.

Chapter II: Background and Review of Literature

Introduction

In this chapter, a brief description of different types of front-end technologies and also the history of these technologies on why they are introduced has been explained. This chapter also explains the review of literature that helps to compare applets and JavaScript.

Background Related to the Problem

Web applications consist of three components: client, server, and database. The client is the front-end or the browser that users would use to view the web pages. The database stores the data related to the web application. Servers are usually located in a remote location and run the website by sending the requests to the browser, receiving requests from a browser, etc. The web server executes the server-side scripting, and web browser executes the client-side script. There are many client-side and server-side scripting technologies currently growing to develop the website.

Initially, HTML was introduced to provide the static web pages with which the contents of the web page cannot be changed. Later, applets were introduced in 1995 to provide interactive web pages, which capture the input data and display the output from the server dynamically. Applets are small Java applications that run in the web browser to provide a number of features, such as providing interactive web pages, gaming, providing 3D images, etc. The applet was the most popular language for more than a decade until JavaScript was introduced (Byrne, 2016).

As per the surveys on articles from 2000 to 2005, applets are used in all web applications, camera-phones, visualizations, CCTV setups, etc. JavaScript was later introduced for client-side scripting, which would run in web browser. Applets run on the browser but need JVM installed on their client machines. However, most modern web browsers are built with JavaScript engines which can run JavaScript, so it does not require a separate plugin to run the script. For this reason, JavaScript became more popular. Conversely, applets were not immediately depreciated after JavaScript was introduced, as JVM offered performance advantages and high speed. (Byrne, 2016)

JavaScript version 8 was then released in 2011. With this version, JavaScript code was compiled by the browser and run on a user's machine directly instead of converting to bytecode like Java. There was a tremendous growth in the usage of JavaScript as client-side scripting with this release (Byrne, 2016).

In 2015, applets were officially depreciated as the latest versions of web browsers stopped supporting these plugins, although applets are more secure, fast and have many features comparing with JavaScript. As a result, this study compares both technologies to find which is the most efficient to be used while creating a web application based on their requirements (Byrne, 2016).

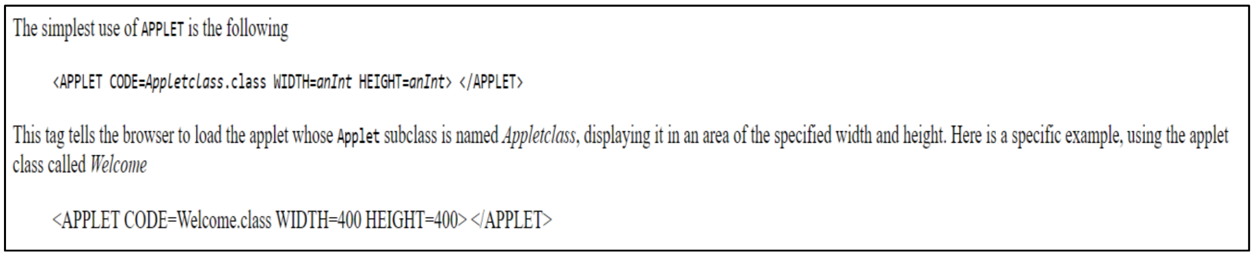
Literature Review

What are applets? Applets are small Java applications which run on the browser with Java enabled. Applets can be embedded in HTML documents and run on the browser. Applet code resides on the web server and is downloaded into a browser whenever the browser requests for the web page containing the applet. Applets are used to provide a more graphical interactive user experience on Java. When the applet is accessed, it is downloaded within the browser from the web server and the code is then embedded to a web page code that browser loads. Basically, the webpage will remain as an HTML code while the applet will be embedded as a Java code (Java applet, 2010).

Advantages of using applets: The main advantage of using applets is that they are platform independent. The applet can be run on all operating systems, and all web browsers support applets. Applets also run quickly, and there is no delay in loading the applets as they cache quickly. The execution speed of applets is also high when compared to other programming languages, such as C++. These may be also be used as a real-time stand-alone application running in local machines (Java applet, 2010).

Java applets use three main functions: *Init*, *Start*, and *Paint*. The Init method runs when the applet loads; the Start method runs the applet when the Paint displays the output to the screen; the Paint method is used to display the output to the screen. Start and Paint can run many times (Java applet, 2010).

How to create dynamic web pages using java applets: Applets can be embedded in an HTML page using an applet tag. It starts with <Applet> and ends with </Applet>. Inside this tag, the instructions needed to run the applet are given. For example, the applet tag within the HTML page is shown below.


The simplest use of APPLET is the following

```
<APPLET CODE=Appletclass.class WIDTH=anInt HEIGHT=anInt> </APPLET>
```

This tag tells the browser to load the applet whose Applet subclass is named *Appletclass*, displaying it in an area of the specified width and height. Here is a specific example, using the applet class called *Welcome*

```
<APPLET CODE=Welcome.class WIDTH=400 HEIGHT=400> </APPLET>
```

Figure 1. Screenshot of applet tag within HTML page.

In the above screenshot, the code attribute within the applet tag identifies the class file of the applet. The codebase attribute has the directory location of the applet *jar* file. The web application creates the class files of applets within the *build* folder when the HTML file is built. Height and width attributes describe the size of the applet as rendered by the browser.

JavaScript: JavaScript is a lightweight programming language used to create dynamic web pages and is integrated with HTML and Java. It is used to write client-side scripts that interact with users and create dynamic web pages. JavaScript was originally known as LiveScript, but the name was later changed to JavaScript. It is supported in web browsers such as Netscape, Internet Explorer, Google Chrome, etc.

Client-Side JavaScript: This is the most common and simple language. This script is used to interact with the user, and it controls the browser to create the content of a web page dynamically. Client-side scripting has many advantages over server-side scripting; for example, client-side validations can verify if a user has created a valid email address in the email identification text field.

Overview of spring framework: The web application examined in this paper was developed using the Spring Framework, which is an opensource Java platform that provides comprehensive infrastructure support to develop Java web applications easily. This framework was developed by Rod Johnson in 2003 under the web application server Apache Tomcat and is the most popular framework among Java frameworks for various features such as lightweight programming, which can create a high performing, reusable, and easily tested code (Spring Framework—Architecture, n.d.).

The architecture of the Spring Framework is explained below:

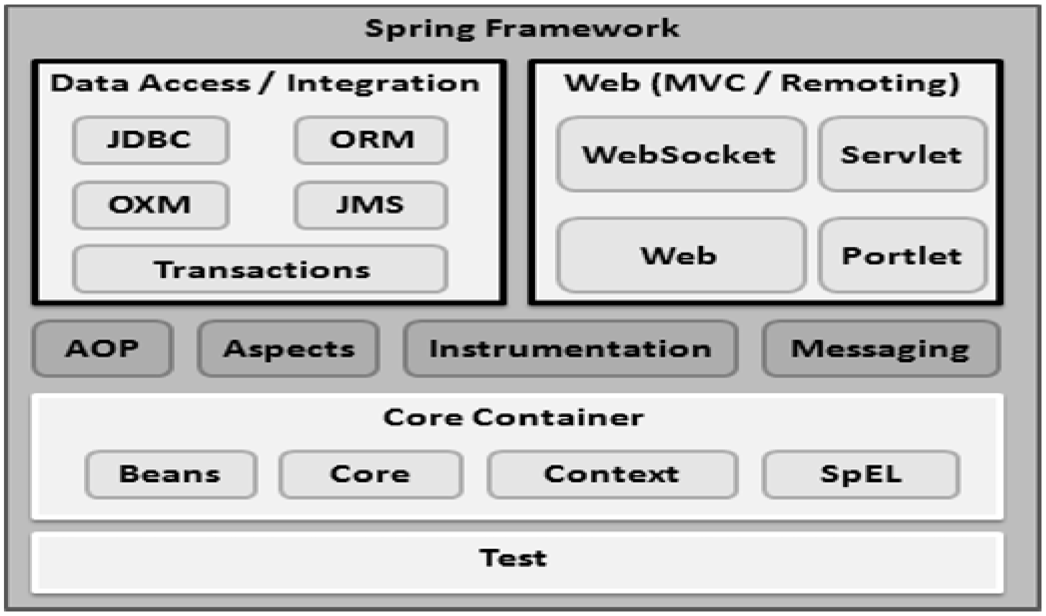


Figure 2. Spring framework architecture (spring Framework—Architecture, n.d.).

Spring Framework provides various modules which can be used according to our requirements. Various sections of this framework are Core Container, Test, Data Access/Integration, Web (MVC/Remoting), etc. where each section has different modules created to provide certain features (Dandan Zhang, 2013).

Core container: The core module within the Core Container is used to provide the two main features of the Spring Framework: IoC and Dependency Injection. The Bean module is used to provide BeanFactory. This module is used to access the objects that are defined and configured. SpEL is a querying language used to alter object graph at run time (Dandan Zhang, 2013).

Data access/integration: The JDBC module within this module is used to access the data from the database. ORM module is an integration layer for object-oriented mapping APIs (Dandan Zhang, 2013).

Web: The Web layer is used to provide web features (e.g., implementing MVC framework using MVC module), other web-oriented functionalities (e.g., file uploading by using Web container), and the two-way communication between the client and server provided by using Web-Socket module, etc. (Rohan Khanna, 2014). In this project, MVC module within the Web layer and the JDBC layer has been used to develop the application (Dandan Zhang, 2013).

Spring MVC: The Spring MVC framework is used to create Model-View-Controller architecture to develop a web application. This architecture is mainly used to separate input data, UI code, server-side code.

DispatcherServlet is used to control all the HTTP requests and responses which is as a front controller for Spring MVC Framework. HTTP requests are received by DispatcherServlet and forwards the request to Controller classes, which then calls the appropriate service methods which has the business model written inside before sending the response to DispatcherServlet. DispatcherServlet forwards this response to the view resolver, which displays the response (Dandan Zhang, 2013).

Model: The Model is the application data.

View: The view part is used to display the data. Generally, this is a JSP or JavaScript page that is run in a client’s web browser (Dandan Zhang, 2013).

Controller: The Controller is used to handle and send requests to appropriate service methods which have the business logic written inside them. It also receives the response from the method and redirects it to the view page to display the response.

The Web.Xml file within WebContent/WEB-INF folder is used to give the URL mapping of a file that we want DispatcherServlet to handle, which then calls the XML file with the name

<Dispatcher Servlet Name-servlet>.xml. This file is the most important in Spring Framework, as it indicates all the URL mappings and classes that need to be handled upon any request (Dandan Zhang, 2013).

Summary

The history of applets and the rise of JavaScript have been outlined in this chapter. A brief discussion of applets, JavaScript and the implementation of web application using Spring MVC, JavaScript and applets have also been explained.

Chapter III: Methodology

Introduction

This chapter details the application design and the database design of the Book Store Management application that is being evaluated, in addition to explaining how the application is developed using Spring MVC architecture, applets, and how to embed applets in web applications. This chapter also describes the tools and techniques that are used to develop the application. Developer Tools is used to compare the applets and JavaScript applications concerning network, memory, performance, security, etc. A brief introduction of Developer Tools and how the above analysis is completed will be discussed in this chapter.

Design of the Study

Development of Book Store Management has been implemented using Agile methodology, so the design of the system could be changed later. One such change was removing the payment page, as it includes storing sensitive information, integrates with third parties, and is difficult to implement as part of this project. The application was also designed in numerous phases, as part of the Agile methodology, and the testing of the application has been done as part of each phase to avoid any possible additional failures.

Comparison of Home Pages created using applet and JavaScript uses both the quantitative approach and qualitative approach. Using quantitative analysis, data would be analyzed based on the network, memory, performance, and provides a statistical analysis of data. To analyze the security and other factors, the existing studies have been taken into consideration. Thus, this study uses both quantitative and qualitative approach.

Data Collection

Application design: The Design of the Book Store Management System is given as follows:

- **Login Page:** Registered users will log in to the system. Users are given a drop-down box to select the username under which they would like to log in. Clicking on the sign-in button redirects users to the main page.
- **Home Page:** Users are redirected to the main page where they are given options to select the category of the books they would like to buy.
- Once the category is selected, it will display the books available for the designated category and users will be able to select the book and the quantity they would like to order.
- **Add to Cart:** Users can add the selected books to their Cart.
- **Cart:** The cart will display the list of items selected and the price for each item.
- **Checkout:** This page will allow the user to check out the selected items in the cart and will place the order. Once the user clicks the button, they will be redirected to the "Payment Successful" page.
- **Payment Successful:** This page will show the success message mentioning that the user has placed the order and payment was successful.

Database design.

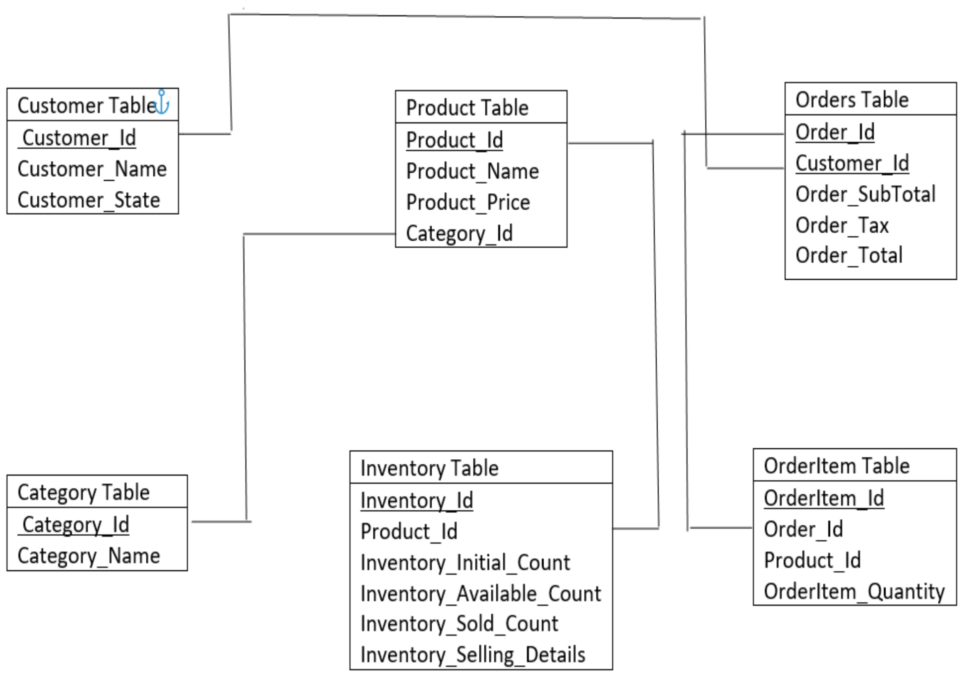


Figure 3. Database design of book store management system.

Design of book store management system using applets as client-side scripting. The Book Store Management System using applets is a web application which is run inside a web browser and allows the users to access the MySQL database over the network. This application uses a client-server paradigm. Client-side application is developed using Java applets, and the server-side application uses servlet technology and is used to interact with the MySQL database and applets.

This application uses three-tier architecture consisting of a graphical user interface, business logic, and database access. The graphical user interface is implemented using the Java applet, which runs inside the web browser. Accessing the user interface through a browser guarantees portability across all platforms that support web browsers. The applets interact with web servers and send the HTTP requests over the network. Apache Tomcat Server has been used as the middle tier and is used to interact with the MySQL database through a JDBC connection and sends the response to the browser (Parekh, 2006).

There are several advantages of using three-tier architecture:

- Applications are modular, making it easier to modify or replace User Interface which can be separated from Business Logic and Storage.
- Achieving thin clients is possible. Subsequently, almost all of the processing takes place at the server. The client is not required to perform complex processing.
- All that is required on the client side is a web browser capable of running Java applets (Parekh, 2006).

The underlying architecture of 3-tier architecture is given as follows:

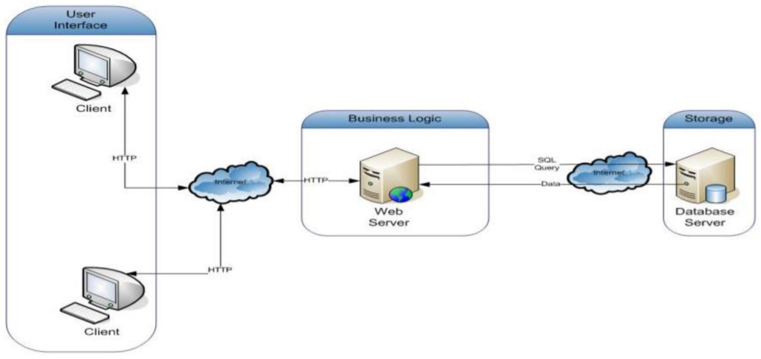


Figure 4. Basic architecture of three-tier architecture (Parekh, 2006).

Home page of book management system created using applets: The below screenshot shows the Login Page of the Book Store Management System that is being created by applets. Selecting the user dropdown retrieves all registered customers of the applications from the Customer Table and displays the customers in the drop-down. When a user selects the desired customer and clicks on the Sign-In Button, the user will be redirected to the CategoryList applet embedded in another HTML page, which displays the list of all categories available from the database.

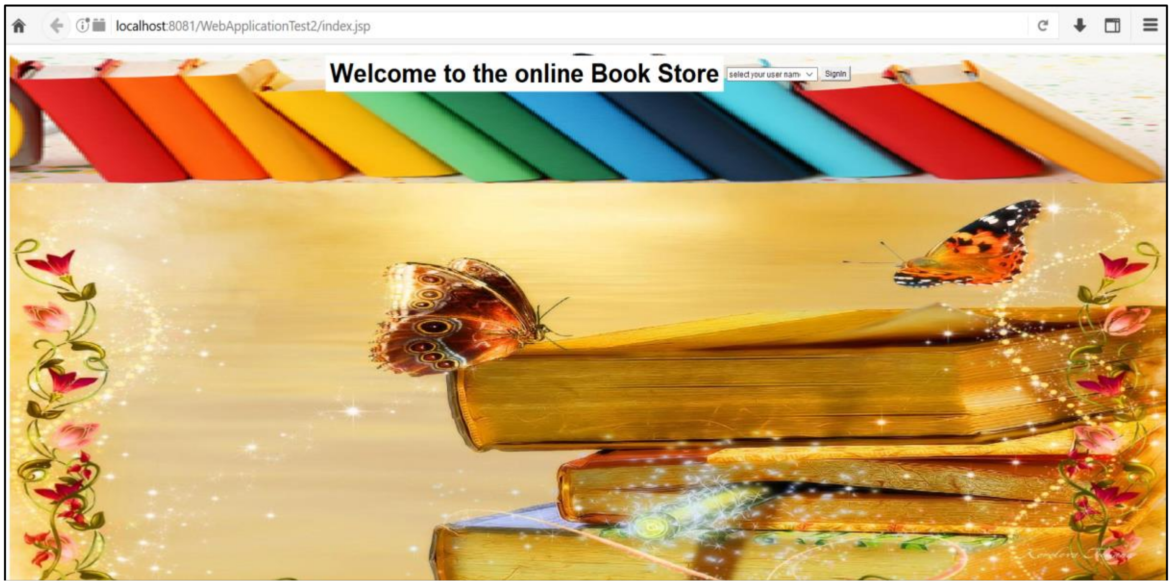

localhost:8081/WebApplicationTest2/index.jsp
Welcome to the online Book Store
select your user name
Signin

Figure 5. Home page of book store management system created using applets.

BookStore management system using javaScript and spring MVC framework: This application was developed using Spring Framework where JavaScript has been used as the client-side scripting language, and Controllers are used to access/manage the data from MySQL database.

Home page of book management system created using javaScript: The below screenshot shows the Login Page of Book Store Management System created by JavaScript.

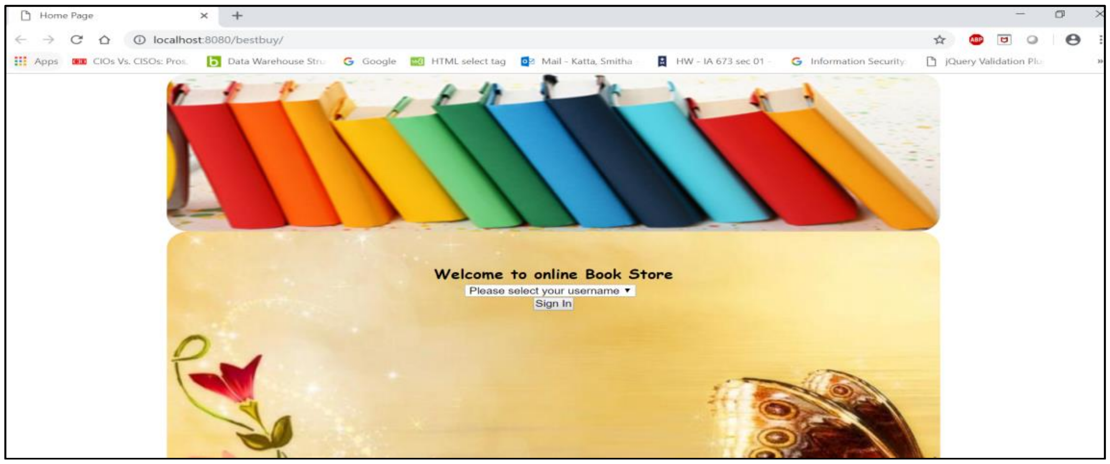


Figure 6. Home page of book store management system created using JavaScript.

Comparison of network, performance, memory tabs using mozilla firefox devTools.

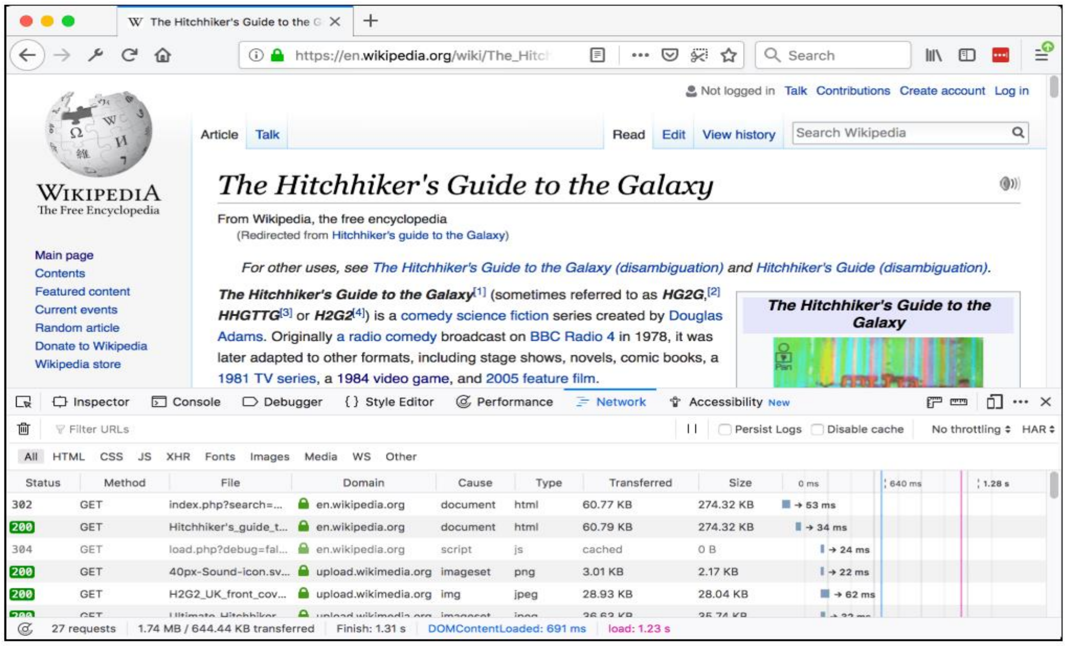
Applets and JavaScript are being compared in terms of the below aspects:

1. Network
2. Memory
3. Performance
4. Security

1. *Steps to Capture Network of Applets and JavaScript:* The Network tab within DevTools is used to compare the network performance of Applets and the JavaScript web page. This tab loads all the network requests such as GET, POST requests of the web page, and its corresponding details. The appropriate steps to utilize this tool are as follows:

- Open the web page needed to capture the network details.
- Select the Tools menu and choose the web developer menu.
- Click the ( ) icon in the main toolbar and choose "Network".

The Network tab would appear at the bottom of the web page. Network requests would be displayed after the page is reloaded (Network Monitor, n.d.).



The Hitchhiker's Guide to the G
https://en.wikipedia.org/wiki/The_Hitc
Search
Not logged in Talk Contributions Create account Log in
Article Talk Read Edit View history Search Wikipedia
WIKIPEDIA
The Free Encyclopedia
Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store
Inspector Console Debugger Style Editor Performance Network Accessibility New
Filter URLs
Persist Logs Disable cache No throttling HAR
All HTML CSS JS XHR Fonts Images Media WS Other

The Hitchhiker's Guide to the Galaxy

From Wikipedia, the free encyclopedia
(Redirected from Hitchhiker's guide to the Galaxy)

For other uses, see The Hitchhiker's Guide to the Galaxy (disambiguation) and Hitchhiker's Guide (disambiguation).

The Hitchhiker's Guide to the Galaxy[1] (sometimes referred to as HG2G,[2] HHGTTG[3] or H2G2[4]) is a comedy science fiction series created by Douglas Adams. Originally a radio comedy broadcast on BBC Radio 4 in 1978, it was later adapted to other formats, including stage shows, novels, comic books, a 1981 TV series, a 1984 video game, and 2005 feature film.

Status	Method	File	Domain	Cause	Type	Transferred	Size	0 ms
302	GET	index.php?search=...	en.wikipedia.org	document	html	60.77 KB	274.32 KB	→ 53 ms
200	GET	Hitchhiker's_guide_t...	en.wikipedia.org	document	html	60.79 KB	274.32 KB	→ 34 ms
304	GET	load.php?debug=fal...	en.wikipedia.org	script	js	cached	0 B	→ 24 ms
200	GET	40px-Sound-icon.sv...	upload.wikimedia.org	imageset	png	3.01 KB	2.17 KB	→ 22 ms
200	GET	H2G2_UK_front_cov...	upload.wikimedia.org	img	jpeg	28.93 KB	28.04 KB	→ 62 ms

27 requests 1.74 MB / 644.44 KB transferred Finish: 1.31 s DOMContentLoaded: 691 ms load: 1.23 s

Figure 7. Screenshot of the network tab in developer tools.

This Network tab contains three sections: the toolbar, network request list, and network request pane. The toolbar is placed at the top of the Network tab. The following figure displays the toolbar of the network tab.

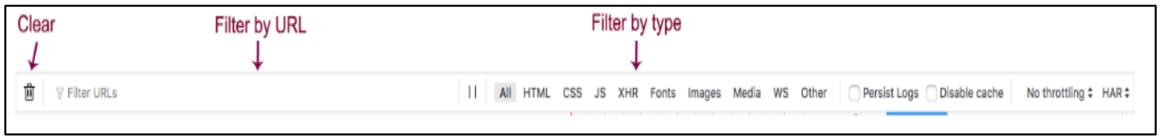


Figure 8. Screenshot of the toolbar section within network tab.

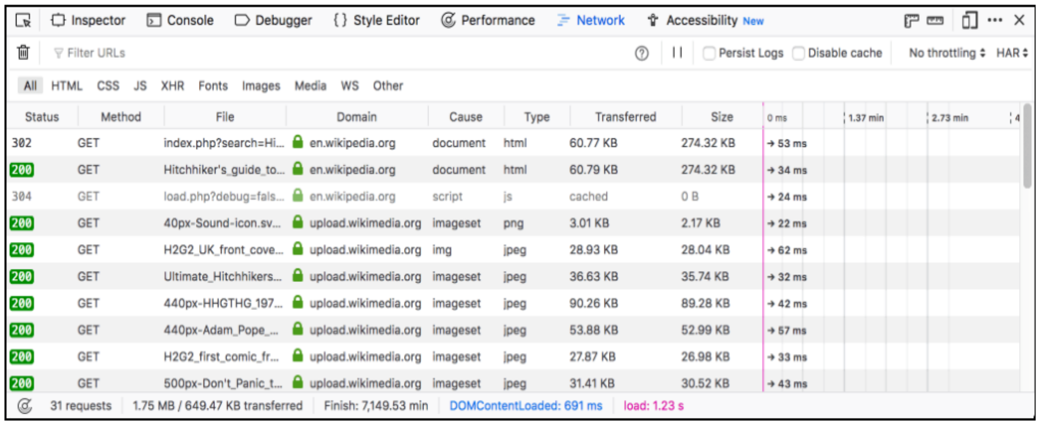
This toolbar provides:

- An icon to search the request
- A trash icon to clear the list of all network requests within Network Tab
- An array of icons which filters the list of network requests by type or by the response content type
- XHR requests

- WebSocket upgrades (labeled WS)
- Network Monitor is cleared each time when navigated to a new page and displays the new page network requests when the new web page is loaded.
- Throttling
- HAR
- A checkbox option to disable cache

A second toolbar is located at the bottom of the network monitor which provides an icon to launch performance analysis, and a summary of this page including the number of requests, total size, and total time (Network monitor toolbar, n.d.).

Network Request List: This section within Network tab displays all the network requests made while loading the page. The following figure displays the Network Requests section within Network Tab.



Status	Method	File	Domain	Cause	Type	Transferred	Size	
302	GET	index.php?search=Hi...	en.wikipedia.org	document	html	60.77 KB	274.32 KB	→ 53 ms
200	GET	Hitchhiker's_guide_to...	en.wikipedia.org	document	html	60.79 KB	274.32 KB	→ 34 ms
304	GET	load.php?debug=fals...	en.wikipedia.org	script	js	cached	0 B	→ 24 ms
200	GET	40px-Sound-icon.sv...	upload.wikimedia.org	imageset	png	3.01 KB	2.17 KB	→ 22 ms
200	GET	H2G2_UK_front_cove...	upload.wikimedia.org	img	jpeg	28.93 KB	28.04 KB	→ 62 ms
200	GET	Ultimate_Hitchhikers...	upload.wikimedia.org	imageset	jpeg	36.63 KB	35.74 KB	→ 32 ms
200	GET	440px-HHGTHG_197...	upload.wikimedia.org	imageset	jpeg	90.26 KB	89.28 KB	→ 42 ms
200	GET	440px-Adam_Pope_...	upload.wikimedia.org	imageset	jpeg	53.88 KB	52.99 KB	→ 57 ms
200	GET	H2G2_first_comic_fr...	upload.wikimedia.org	imageset	jpeg	27.87 KB	26.98 KB	→ 33 ms
200	GET	500px-Don't_Panic_t...	upload.wikimedia.org	imageset	jpeg	31.41 KB	30.52 KB	→ 43 ms

31 requests | 1.75 MB / 649.47 KB transferred | Finish: 7,149.53 min | DOMContentLoaded: 691 ms | load: 1.23 s

Figure 9. Screenshot of the network requests section within the network tab.

This section includes the following columns:

- Status: returns the HTTP code status. The status is displayed as a color-coded icon based on the status of the request.

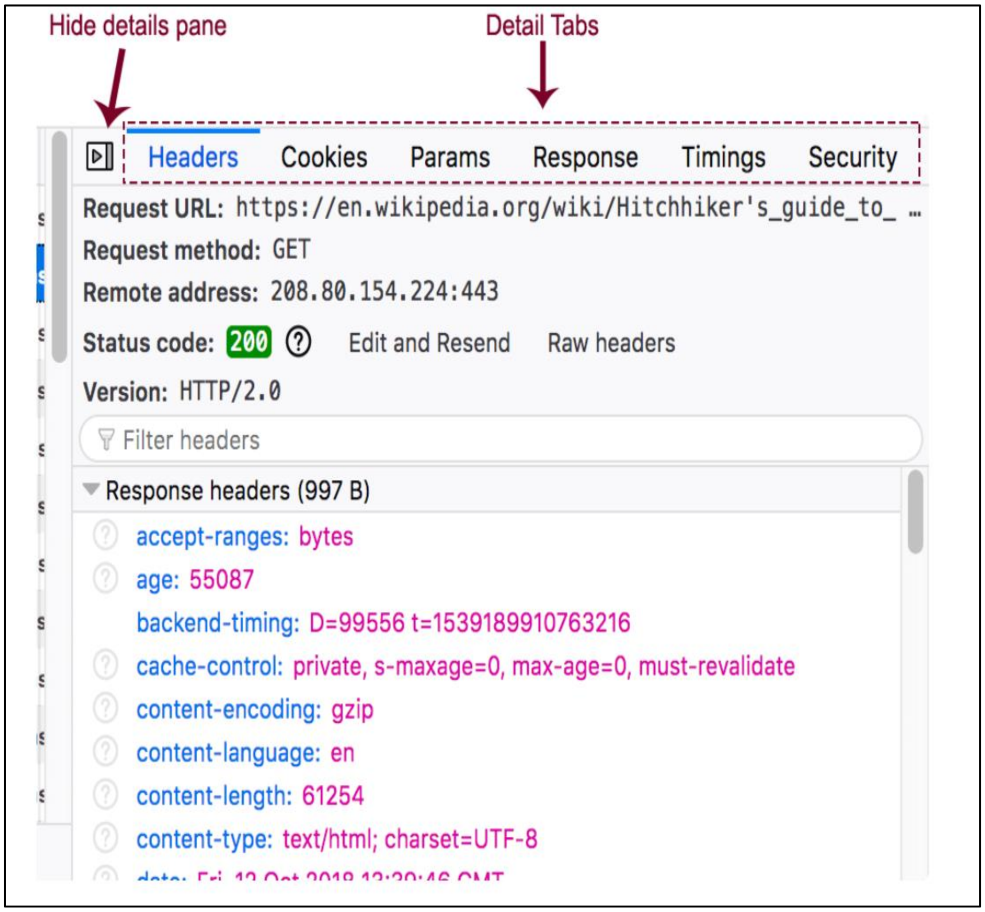
- Method: displays the HTTP method used, such as GET/POST, etc.
- File: basename of the file requested.
- Protocol: displays the network protocol used to get/post the data.
- Scheme: scheme of the path requested.
- Domain: displays the requested path domain.

There is also an icon next to Domain that displays extra information about the request security status.

- Remote IP: displays the server IP address answering the request.
- Cause: reason for the network request occurrence.
- Type: displays the content-type of the response.
- Cookies: displays the number of request cookies associated with the request.
- Set-cookies: displays the number of request cookies associated with the request.
- Transferred: number of bytes transferred while loading the resource.
- Size: size of the transferred resource. (Network request list, n.d.).

Network request details section: This section is displayed when any request is selected within the Network tab. It will display the details of the request selected.

The following figure shows the Network Details section.



```
Request URL: https://en.wikipedia.org/wiki/Hitchhiker's_guide_to_ …
Request method: GET
Remote address: 208.80.154.224:443
Status code: 200
Version: HTTP/2.0
Response headers (997 B)
accept-ranges: bytes
age: 55087
backend-timing: D=99556 t=1539189910763216
cache-control: private, s-maxage=0, max-age=0, must-revalidate
content-encoding: gzip
content-language: en
content-length: 61254
content-type: text/html; charset=UTF-8
```

Figure 10. Screenshot of the network details section within network tab.

This section displays the following tabs:

- **Headers**: shows the basic information about the requests such as URL, request Method, Remote IP address, status code, HTTP request and response headers, etc.
- **Cookies**: displays the list of cookies that are sent with the request/response.
- **Params**: displays the parameters that are sent in the GET request and the POST data of a request.
- **Response**: displays the response of the request in a text format.
- **Cache**: displays the details about the cached resource.

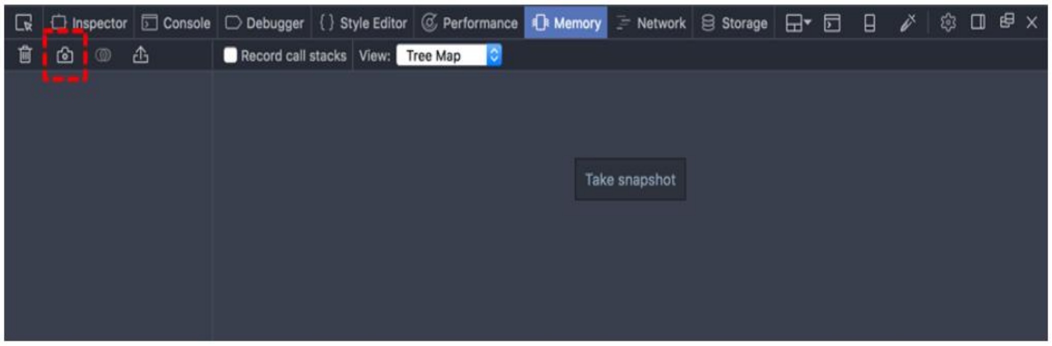
- **Timings:** this tab breaks down the network request and shows the time spent in various categories such as blocked time, DNS resolution, connecting time, waiting time, sending time, receiving time etc.
- **Security**: this tab is shown only when the request is sent over HTTPS and contains details about the secure connection used including the protocol, the cipher suite, and certificate details.
- **Stack Trace:** displays responses that possess a stack trace (Network request list, n.d.).

2. *Steps to capture memory management of a web page.* The Memory tool is used to take a snapshot of the current tab's memory. It provides a detailed view of all the object's memory usage and where exactly the memory is allocated in the code (Memory, n.d.).

Steps to Open Memory Tool: The Memory tab is enabled by default for Mozilla Firefox from Firefox 50 onwards. So, Memory tab is viewed by default when opened the Developer Tools (Memory, n.d.).

To take a heap snapshot: Click on the Take Snapshot Button in the Memory tab, which will display all objects using memory and each entry for the new snapshot (e.g., memory, size, etc.). The snapshot can be saved to the desktop by clicking the Save button (Memory, n.d.).

Comparing Snapshots: The Venn diagram icon in the toolbar is used to compare two snapshots. The comparison shows where the memory was allocated or freed between two snapshots (Memory, n.d.).



+

Figure 11. Snapshot button in memory tab.

Memory analysis can be viewed in three different ways: treemap view, aggregate view, and dominators view.

TreeMap view: This view provides a visual representation of the snapshot, which displays the objects that use the most memory (Memory, n.d.).

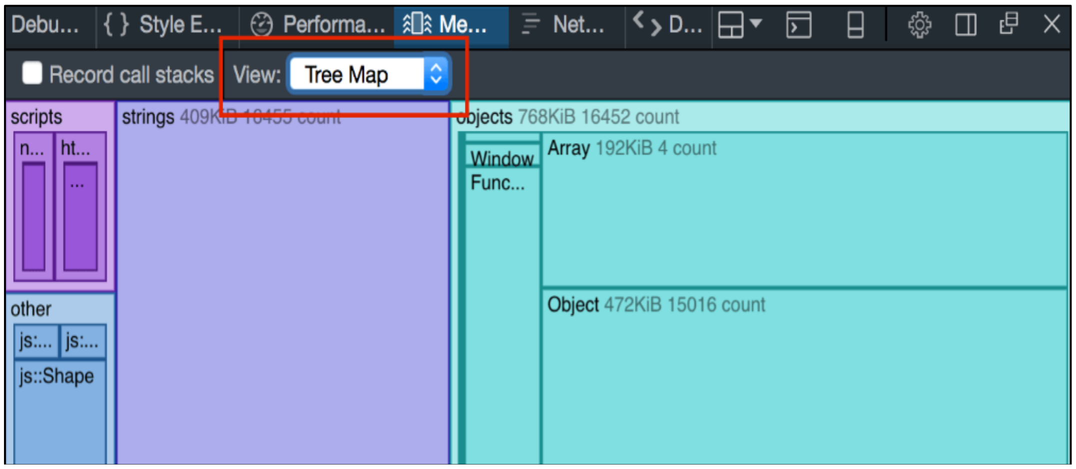


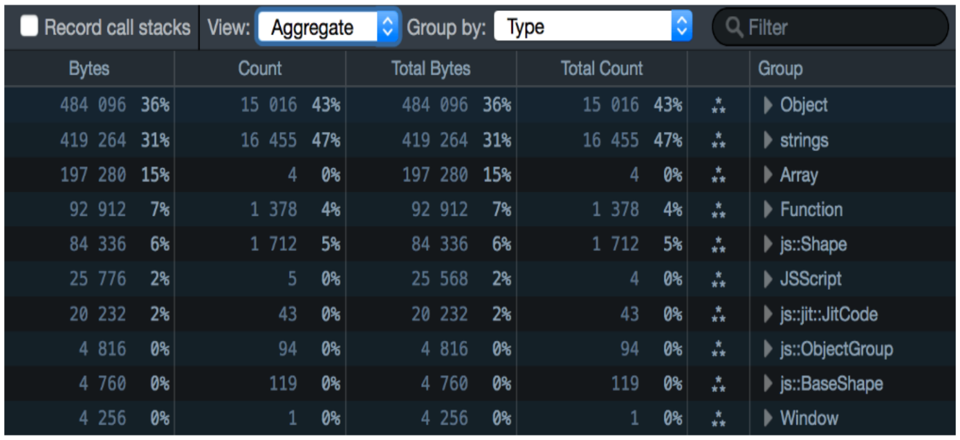
Figure 12. TreeMap view of the memory tab.

The treemap view displays the following sections:

- Objects: displays the memory usage of JavaScript and DOM objects such as Function, Object, or Array.
- Scripts: displays the JavaScript sources loaded by the page.

- Strings
- Other: includes all other internal objects (Memory, n.d.).

Aggregate view: This view displays the memory usage as a table of allocated types. The view captured in the below screenshot displays the breakdown of heap's contents. The memory usage can be grouped by Type, Call Stack, and Inverted Call Stack.



Record call stacks View: Aggregate Group by: Type Filter

Bytes	Count	Total Bytes	Total Count	Group
484 096 36%	15 016 43%	484 096 36%	15 016 43%	Object
419 264 31%	16 455 47%	419 264 31%	16 455 47%	strings
197 280 15%	4 0%	197 280 15%	4 0%	Array
92 912 7%	1 378 4%	92 912 7%	1 378 4%	Function
84 336 6%	1 712 5%	84 336 6%	1 712 5%	js::Shape
25 776 2%	5 0%	25 568 2%	4 0%	JSScript
20 232 2%	43 0%	20 232 2%	43 0%	js::jit::JitCode
4 816 0%	94 0%	4 816 0%	94 0%	js::ObjectGroup
4 760 0%	119 0%	4 760 0%	119 0%	js::BaseShape
4 256 0%	1 0%	4 256 0%	1 0%	Window

Figure 13. Screenshot of aggregate view of the memory tab.

The *Group By Type* drop-down menu can group the heap into types, including:

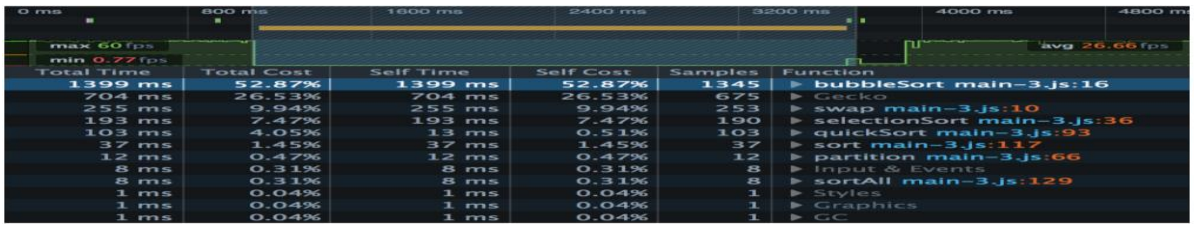
- JavaScript objects
- DOM elements
- Strings
- JavaScript resources
- Internal objects

The *Group By Call* stack displays where exactly heap allocations are made in the code. The *Group by Inverted Call* stack displays the bottom-up view of the program, showing the exact places where allocations are happening ranked by the size of the allocation at each place (Memory, n.d.).

Dominators View: This view displays the retained size of objects. The dominators' view is useful for understanding the size of objects allocated to the site which is the object's size and the object's size which keep alive through references (Memory, n.d.).

Steps to capture performance of a web page. The performance tool is used to analyze the responsiveness of a page, JavaScript, and overall layout performance. With this tool, a user can record or profile a web page over a period of time and then displays the overview of what the browser was doing to render the site and the frame rate of the profile. There are four sections in the Performance tab that refine the performance analysis of a web page in more detail:

- The Waterfall provides a comprehensive analysis of the operations that the web browser was performing such as executing layout, JavaScript, garbage collection, and repaints.
- The Call Tree provides the analysis of JavaScript functions for which the browser spent more time to load.
- The Frame Chart provides the call stack of JavaScript functions while recording the performance.
- The Allocations view displays the heap allocations made by the code while recording. This view is not displayed by default and can be enabled by selecting the Record Allocations checkbox in Performance Tool Settings.



Total Time	Total Cost	Self Time	Self Cost	Samples	Function
1399 ms	52.87%	1399 ms	52.87%	1345	▶ bubbleSort main-3.js:16
704 ms	26.53%	704 ms	26.53%	675	▶ Gecko
255 ms	9.94%	255 ms	9.94%	253	▶ swap main-3.js:10
193 ms	7.47%	193 ms	7.47%	190	▶ selectionSort main-3.js:36
103 ms	4.05%	13 ms	0.51%	103	▶ quickSort main-3.js:93
37 ms	1.45%	37 ms	1.45%	37	▶ sort main-3.js:117
12 ms	0.47%	12 ms	0.47%	12	▶ partition main-3.js:66
8 ms	0.31%	8 ms	0.31%	8	▶ Input & Events
8 ms	0.31%	8 ms	0.31%	8	▶ sortAll main-3.js:129
1 ms	0.04%	1 ms	0.04%	1	▶ Styles
1 ms	0.04%	1 ms	0.04%	1	▶ Graphics
1 ms	0.04%	1 ms	0.04%	1	▶ GC

Figure 14. Sample screenshot of the calltree view of performance tab.

This allows the user to recognize exactly which function was executed at any point during the recording, how long the function ran for, and where the function was called from (Performance, n.d.).

Tools and Technology

Tools used.

- Spring Tool Suite (STS) IDE is used to create the Book Store Management application using Spring Framework; Java NetBeans IDE is used to create the application using applets.
- Apache Tomcat is used as the web server to deploy the web application.
- Workbench GUI is used to access the data from the MySQL database.
- Mozilla Firefox browser is used as the web browser to view the web application.
- Firefox DevTools is used to compare the network, performance, and memory of the web application.

Technology used

- Spring Model View Controller Framework: used to develop the web application where JavaScript and HTML code is embedded in Java Servlet pages as the View part of the model view controller (MVC). DAO Implementation classes act as the controller classes, which apply the business logic implemented in these Java classes. Model classes are the POJO classes which have the getters and setters.
- Restful Webservices: used to integrate the application with the database.
- Applets and Servlet Technology: used to create the login page of the second Book Store Management System application.

Hardware and Software Requirements

Hardware requirements.

Table 1

Hardware Requirements

Item	**Size**
Memory	4.0 GB
Processor Type	32/64 bit
Processor Speed	1Ghz
OS	Windows

Software requirements.

Table 2

Software Requirements

Technology	HTML, JavaScript, JSP, J2EE, CSS, Applets, Restful Web Services
Database	My SQL
Server	Apache Tomcat
Framework	Spring
Operating System	Windows 8
Web Browser	Mozilla Firefox
IDE	Spring Tool Suite(STS), Java Net Beans

Chapter IV: Data Presentation and Analysis

Introduction

In this chapter, the data presentation section will exhibit screenshots of each Book Store Management System web page that is created using JavaScript, as well as the home page of the web application created using applets. This section will also describe the functionality of the Book Store Management System.

In the data analysis section, the home page of both applications created using JavaScript and applets will be compared. These two web pages have been evaluated in terms of network, performance, memory, and security in order to determine which is the better technology that can be used by consumers. The two web pages have also been analyzed based on their general features, advantages, and disadvantages of both applets and JavaScript.

Data Presentation

The UI screenshots of the Book Store Management System created using applets and JavaScript are displayed in this section. Below figure displays, the screenshot of the home page created using JavaScript.

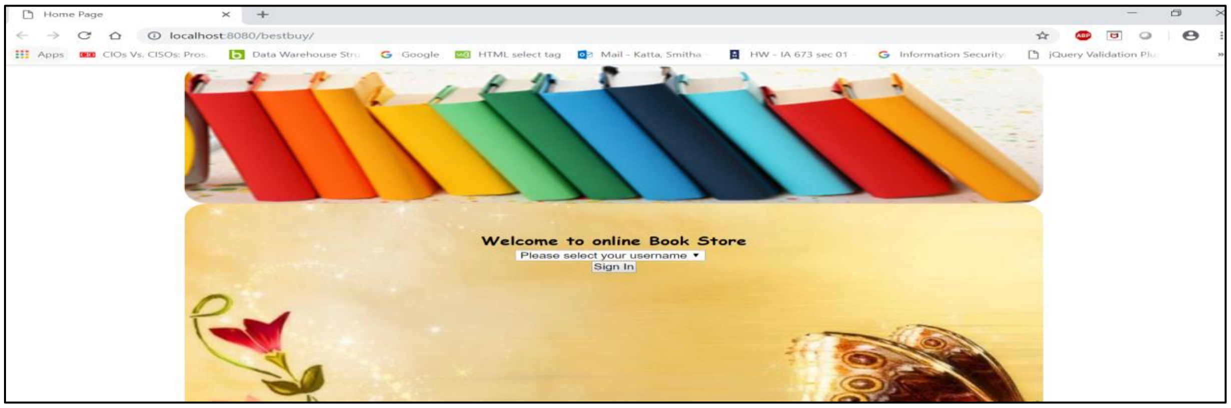
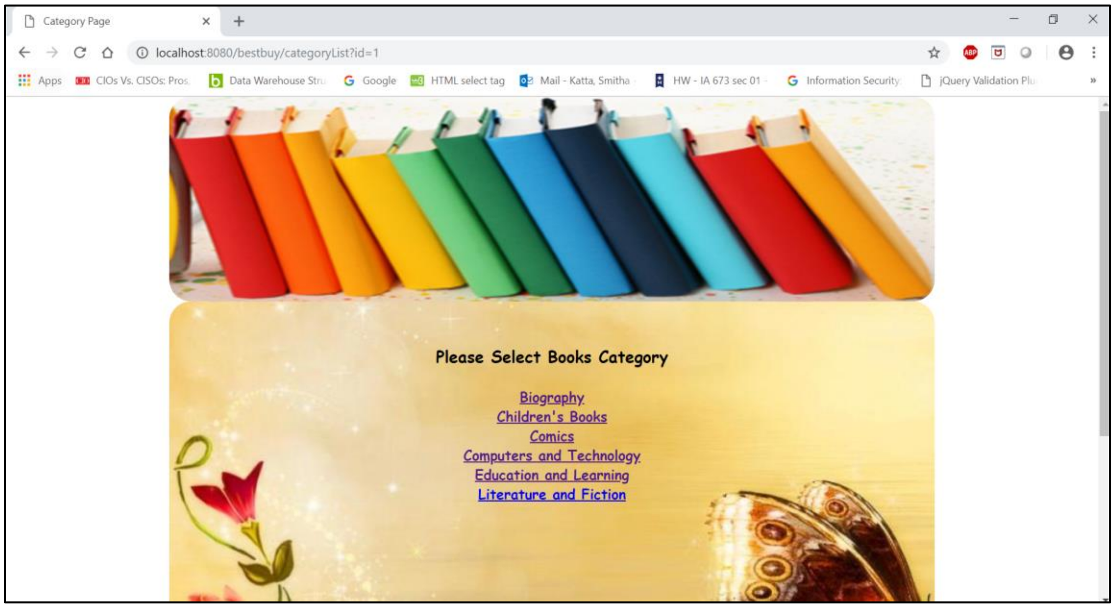


Figure 15. Home page of online book store.

After selecting the username and clicking on the Sign In button, the site is redirected to the Main Page, which displays the list of available categories as shown in the figure below.

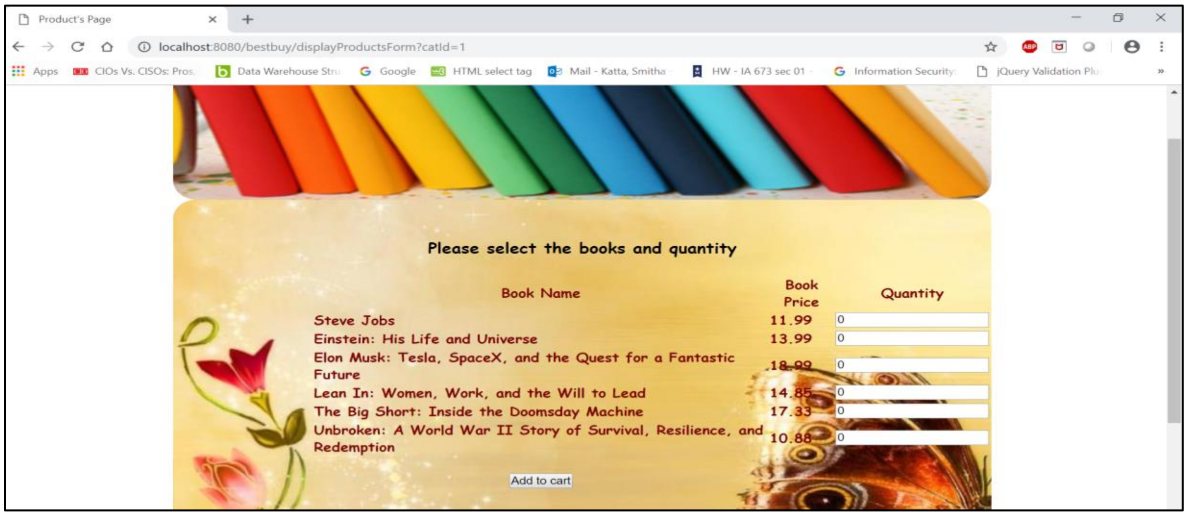


Please Select Books Category

Biography
Children's Books
Comics
Computers and Technology
Education and Learning
Literature and Fiction

Figure 16. Category List page of book store created using JavaScript.

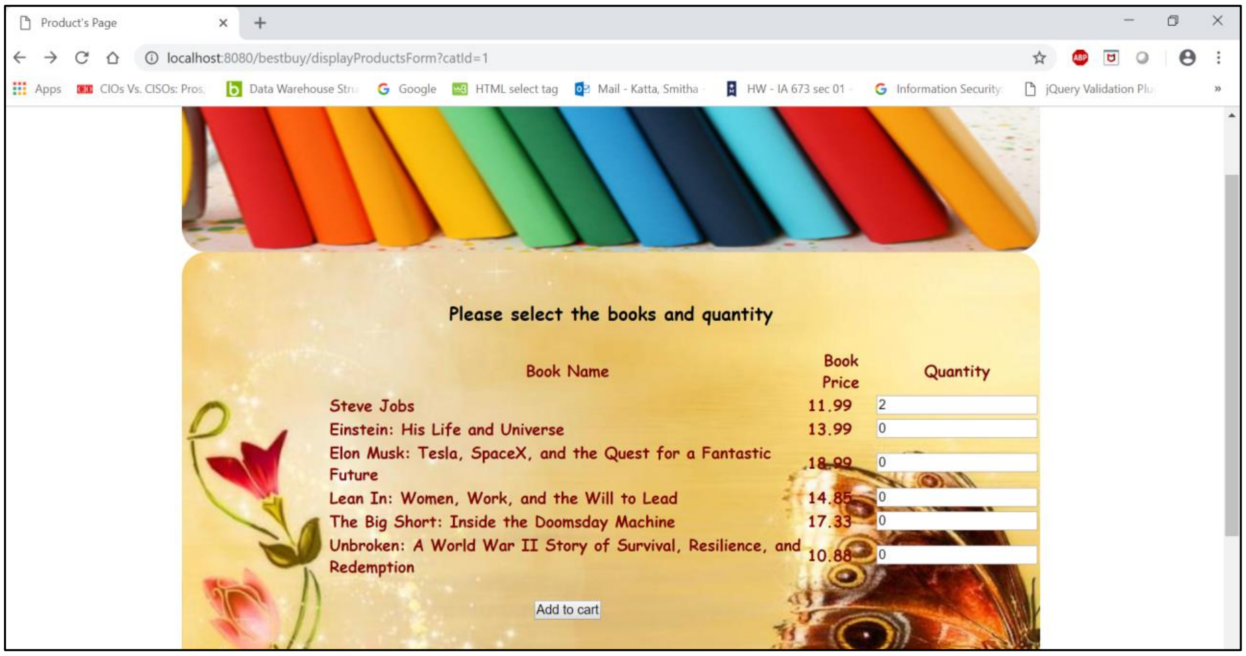
If the user chooses to select the Biography category, the following page will display the list of available books in that category.



Please select the books and quantity

Book Name	Book Price	Quantity
Steve Jobs	11.99	0
Einstein: His Life and Universe	13.99	0
Elon Musk: Tesla, SpaceX, and the Quest for a Fantastic Future	18.99	0
Lean In: Women, Work, and the Will to Lead	14.85	0
The Big Short: Inside the Doomsday Machine	17.33	0
Unbroken: A World War II Story of Survival, Resilience, and Redemption	10.88	0

Figure 17. Products form page of book store created using JavaScript.

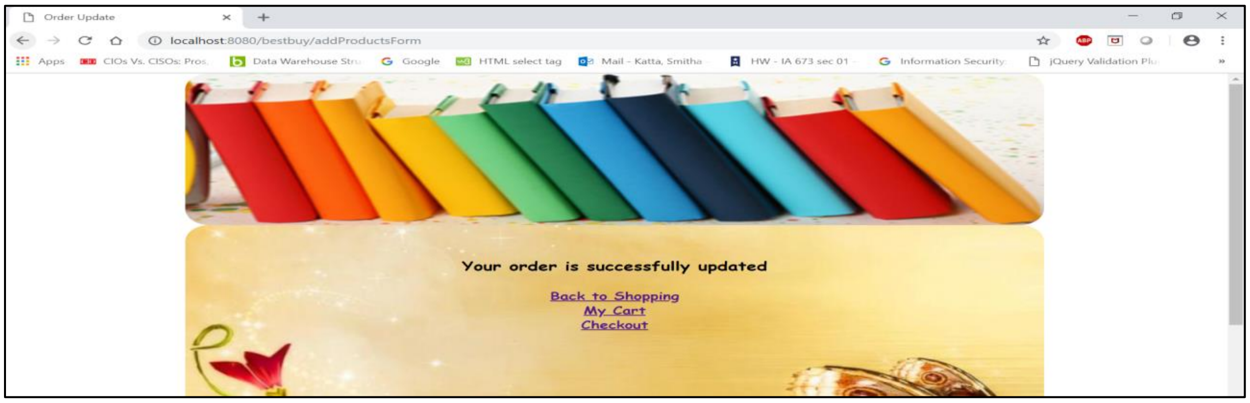


Please select the books and quantity

Book Name	Book Price	Quantity
Steve Jobs	11.99	2
Einstein: His Life and Universe	13.99	0
Elon Musk: Tesla, SpaceX, and the Quest for a Fantastic Future	18.99	0
Lean In: Women, Work, and the Will to Lead	14.85	0
The Big Short: Inside the Doomsday Machine	17.33	0
Unbroken: A World War II Story of Survival, Resilience, and Redemption	10.88	0

Figure 18. Updating quantity for the book in products form page of book store created using JavaScript.

When the user selects the quantity for a book they would like to order and click on the *Add to Cart* button, the order details will be added to the cart, and the page will be redirected to the *Add Products* page.



Your order is successfully updated

Back to Shopping
My Cart
Checkout

Figure 19. Add products page of book store created using JavaScript.

When the user clicks on the *Back to Shopping* link from the above page, the user will be redirected to the main page with the initial list of categories displayed.

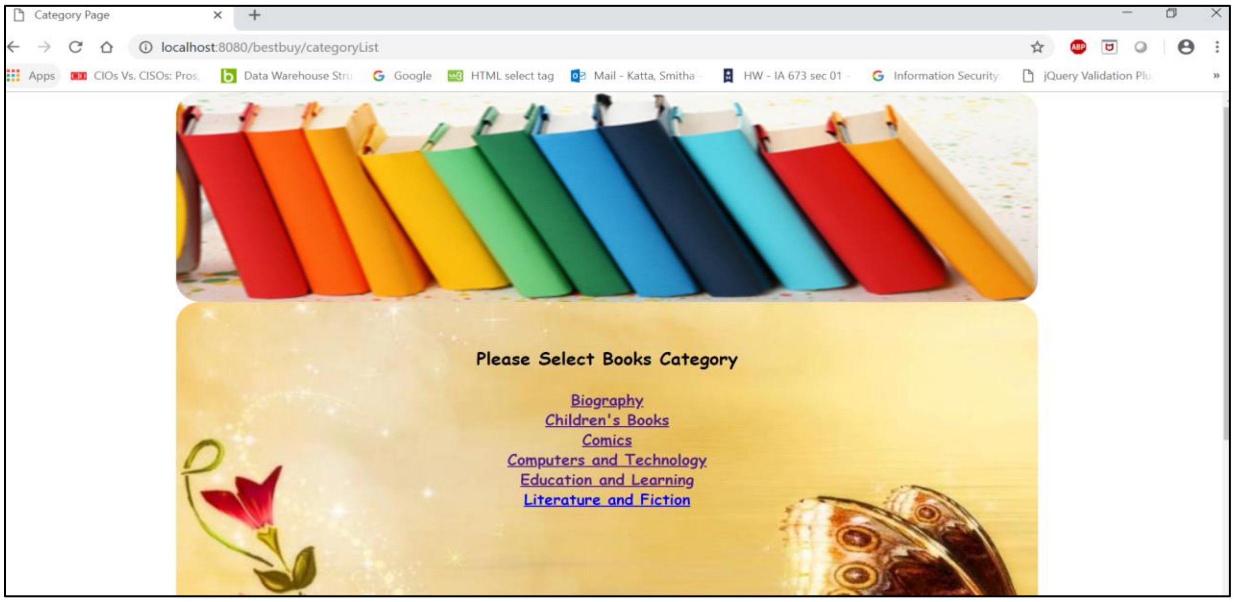
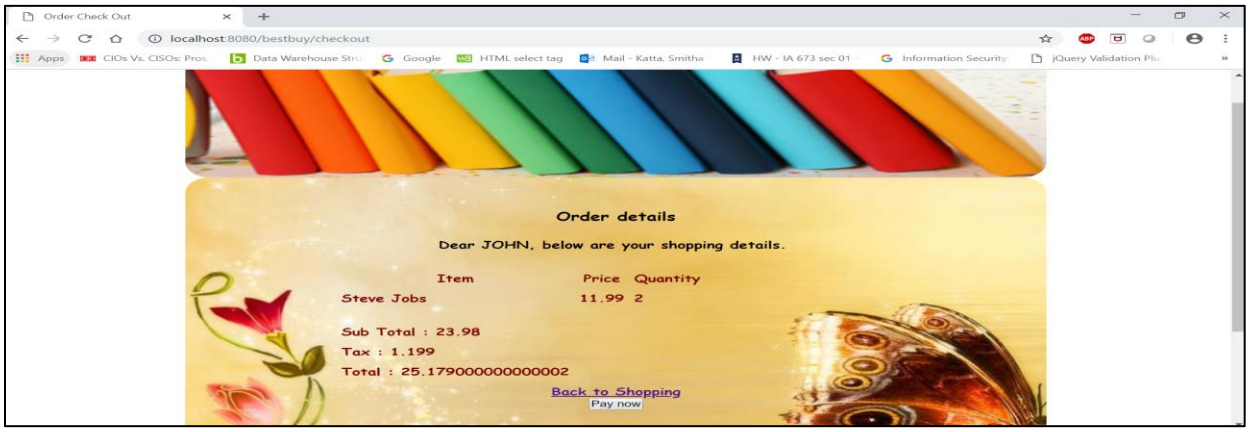


Figure 20. Category list page of book store created using JavaScript.

When the user clicks on the *My Cart* link on the Add Products page, they will be redirected to the Cart page with the list of books that are added to the user's cart.



Order details

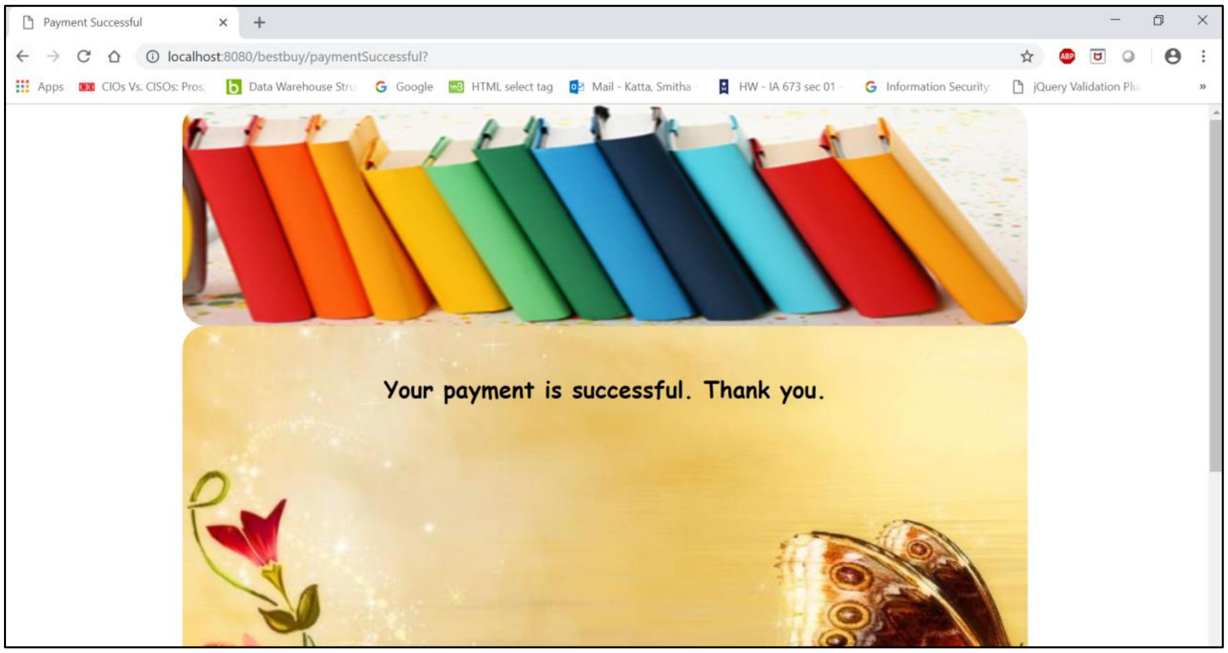
Dear JOHN, below are your shopping details.

Item	Price	Quantity
Steve Jobs	11.99	2

Sub Total : 23.98
Tax : 1.199
Total : 25.179000000000002

Figure 21. Checkout page of book store created using JavaScript.

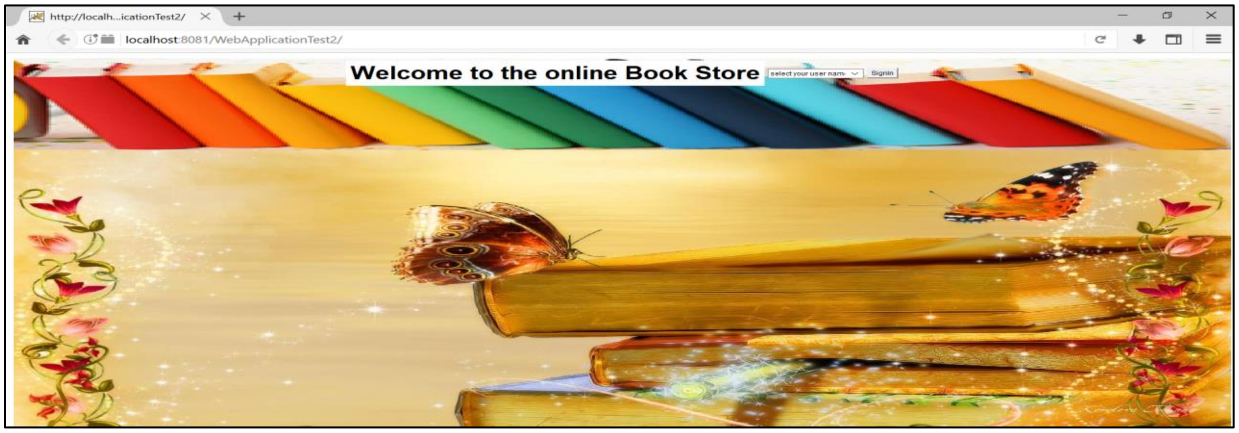
Payment Successful page is displayed when the user clicks on the *Pay Now* button.



Your payment is successful. Thank you.

Figure 22. Product success page of book store created using JavaScript.

The home page of the Book Store Management page that was created using applets is shown in the below figure.


http://localh...icationTest2/
localhost:8081/WebApplicationTest2/
select your user name
Signin

Welcome to the online Book Store

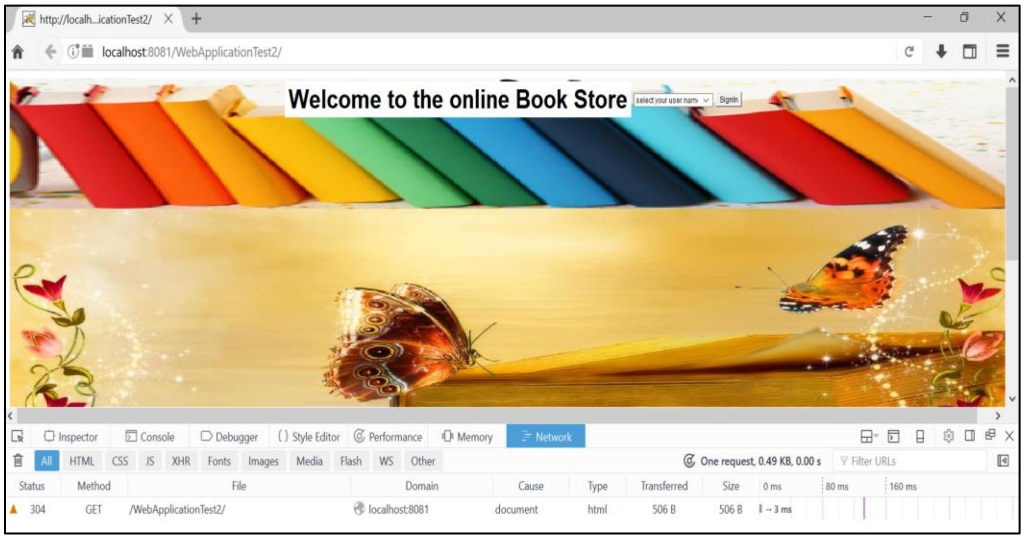
*Figure 23. H*ome page of book store created using applets.

Data Analysis

Network, memory, and performance analysis of the home pages for both the JavaScript and applet applications are compared in this section.

Network tab comparison.

Network tab analysis of home page created using applet: The network tab analysis of the home page created using applets is shown below.

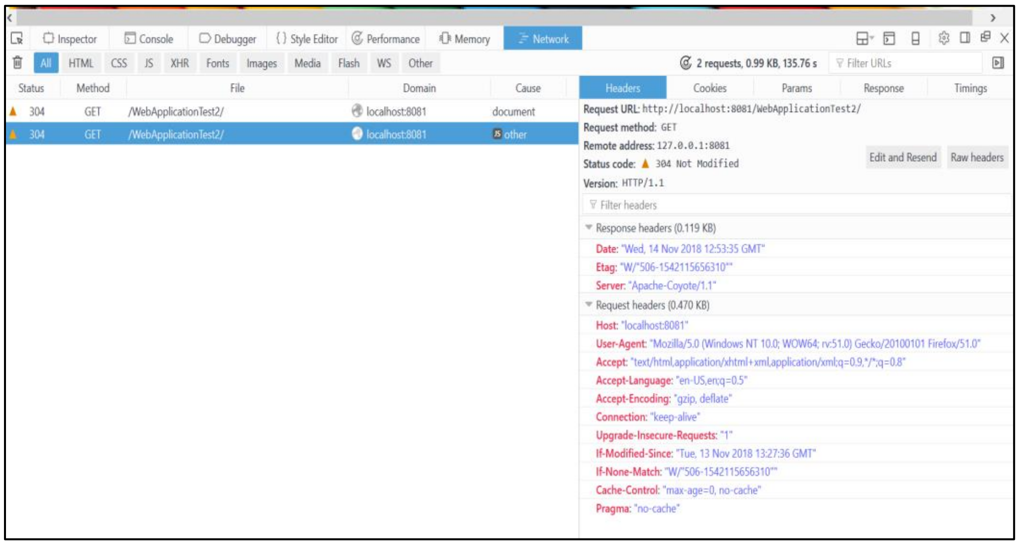


Status	Method	File	Domain	Cause	Type	Transferred	Size	0 ms
304	GET	/WebApplicationTest2/	localhost:8081	document	html	506 B	506 B	→ 3 ms

Figure 24. Network tab of home page created using applet.

The above screenshot displays the Network tab, which has the Get method that loads the index or home page of the application. The index page calls the applet defined in the *index.jsp* file, which has the business logic to display all the elements of the web page, the styling of each element, and the logic to retrieve the customer list.

From the above Network tab, when clicked on the GET method, the details of the concerned request are shown. To analyze further, the headers, response, and timings sections are taken into consideration. The screenshots of those three tabs are shown below.



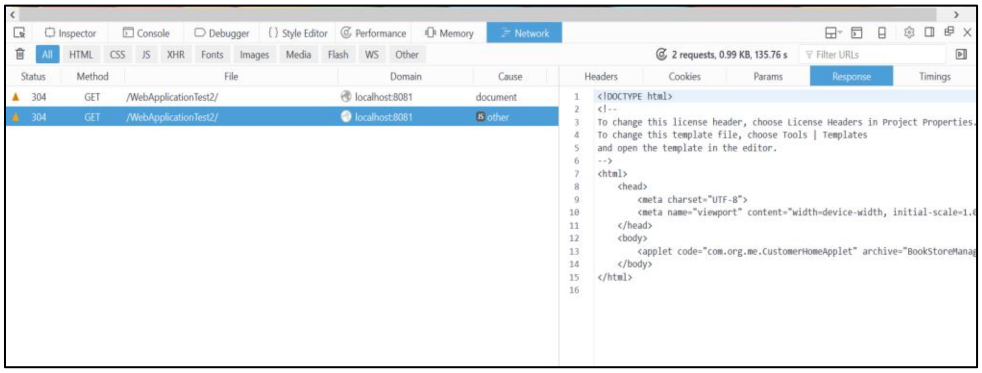
2 requests, 0.99 KB, 135.76 s

Status	Method	File	Domain	Cause
304	GET	/WebApplicationTest2/	localhost:8081	document
304	GET	/WebApplicationTest2/	localhost:8081	other

```
Request URL: http://localhost:8081/WebApplicationTest2/
Request method: GET
Remote address: 127.0.0.1:8081
Status code: 304 Not Modified
Version: HTTP/1.1
Response headers (0.119 KB)
  Date: "Wed, 14 Nov 2018 12:53:35 GMT"
  Etag: "W/"506-1542115656310""
  Server: "Apache-Coyote/1.1"
Request headers (0.470 KB)
  Host: "localhost:8081"
  User-Agent: "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:51.0) Gecko/20100101 Firefox/51.0"
  Accept: "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
  Accept-Language: "en-US,en;q=0.5"
  Accept-Encoding: "gzip, deflate"
  Connection: "keep-alive"
  Upgrade-Insecure-Requests: "1"
  If-Modified-Since: "Tue, 13 Nov 2018 13:27:36 GMT"
  If-None-Match: "W/"506-1542115656310""
  Cache-Control: "max-age=0, no-cache"
  Pragma: "no-cache"
```

Figure 25. Headers tab of network tab for the home page created using applets.

The above header page shows the details related to the URL or the network request that has been selected. The applet web page has only one network request to load all the elements, and the above tab displays the type of request (i.e., GET method). The above tab also displays other details such as request URL, type of request, the request status, response headers info, etc.

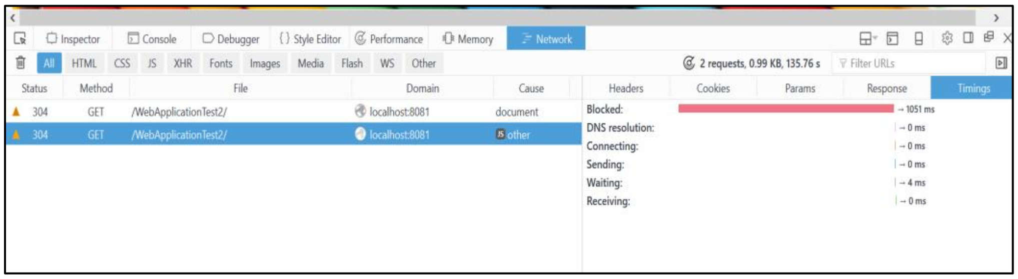


2 requests, 0.99 KB, 135.76 s

Status	Method	File	Domain	Cause
304	GET	/WebApplicationTest2/	localhost:8081	document
304	GET	/WebApplicationTest2/	localhost:8081	other

```
<!DOCTYPE html>
<!--
To change this license header, choose License Headers in Project Properties.
To change this template file, choose Tools | Templates
and open the template in the editor.
-->
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0
    </head>
    <body>
        <applet code="com.org.me.CustomerHomeApplet" archive="BookStoreManag
    </body>
</html>
```

Figure 26. Response tab of network tab for the home page created using applets.



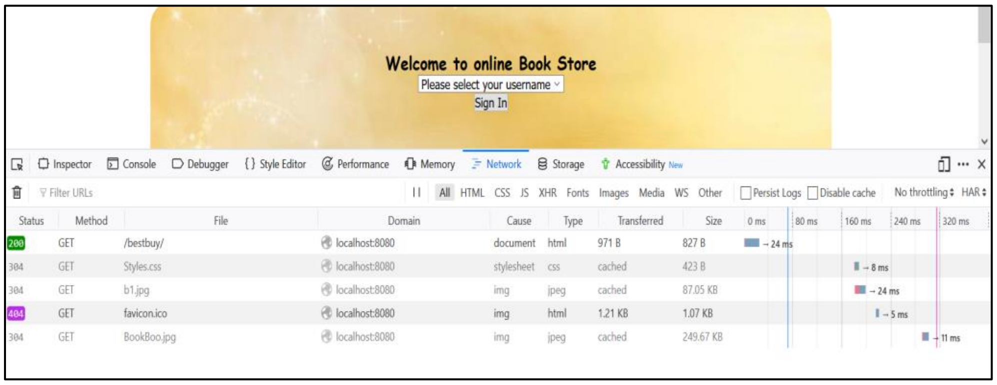
Status	Method	File	Domain	Cause
304	GET	/WebApplicationTest2/	localhost:8081	document
304	GET	/WebApplicationTest2/	localhost:8081	other

Blocked:	→ 1051 ms
DNS resolution:	→ 0 ms
Connecting:	→ 0 ms
Sending:	→ 0 ms
Waiting:	→ 4 ms
Receiving:	→ 0 ms

Figure 27. Timings tab of network tab for the home page created using applets.

The above screenshot displays the timings section of applets home page, which displays the total time that is taken to load the request in the request, i.e., the waiting time, blocked time, receiving time and the time taken to connect to the network.

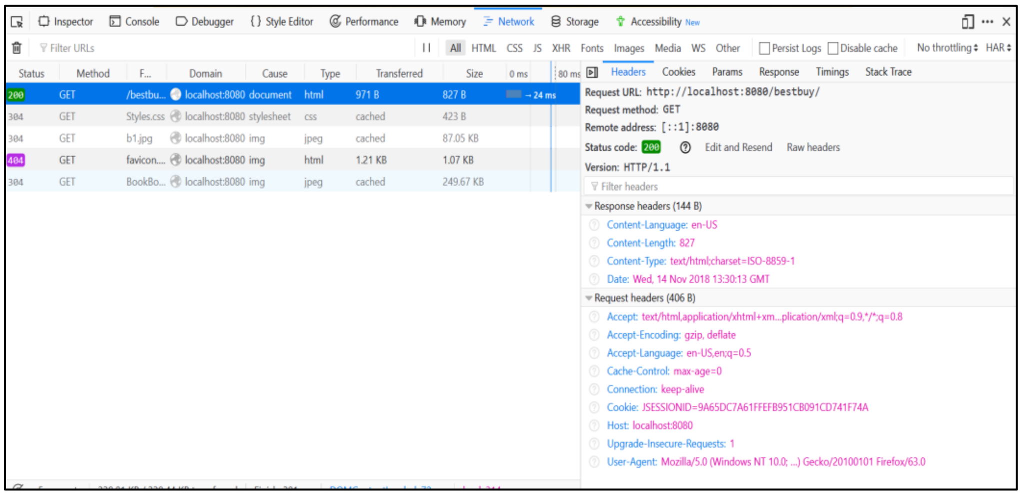
Network tab of javascript home page: The Network Tab of Home page created using JavaScript is shown in the below screenshot.



Status	Method	File	Domain	Cause	Type	Transferred	Size	0 ms 80 ms 160 ms 240 ms 320 ms
200	GET	/bestbuy/	localhost:8080	document	html	971 B	827 B	→ 24 ms
304	GET	Styles.css	localhost:8080	stylesheet	css	cached	423 B	→ 8 ms
304	GET	b1.jpg	localhost:8080	img	jpeg	cached	87.05 KB	→ 24 ms
404	GET	favicon.ico	localhost:8080	img	html	1.21 KB	1.07 KB	→ 5 ms
304	GET	BookBoo.jpg	localhost:8080	img	jpeg	cached	249.67 KB	→ 11 ms

Figure 28. Network tab for the home page created using JavaScript.

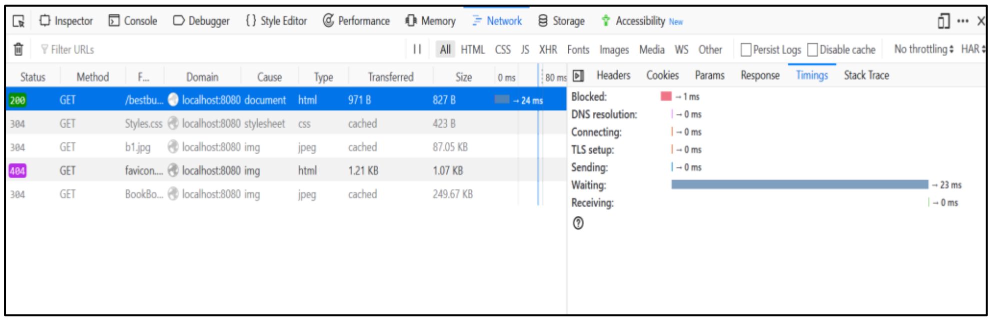
The above screenshot shows all the requests of the Home Page that are needed to load the Home Page which includes the GET method to load the index/home jsp file and the css styles page, the images, etc. along with the time taken to load each request. The headers, timings, response tabs of all the above requests are shown in the below screenshots.



```
Status  Method  F...         Domain          Cause       Type  Transferred  Size       0 ms
200     GET     /bestbu...   localhost:8080  document    html  971 B        827 B      → 24 ms
304     GET     Styles.css   localhost:8080  stylesheet  css   cached       423 B
304     GET     b1.jpg       localhost:8080  img         jpeg  cached       87.05 KB
404     GET     favicon....  localhost:8080  img         html  1.21 KB      1.07 KB
304     GET     BookBo...    localhost:8080  img         jpeg  cached       249.67 KB
```

```
Request URL: http://localhost:8080/bestbuy/
Request method: GET
Remote address: [::1]:8080
Status code: 200
Version: HTTP/1.1
Response headers (144 B)
  Content-Language: en-US
  Content-Length: 827
  Content-Type: text/html;charset=ISO-8859-1
  Date: Wed, 14 Nov 2018 13:30:13 GMT
Request headers (406 B)
  Accept: text/html,application/xhtml+xm...plication/xml;q=0.9,*/*;q=0.8
  Accept-Encoding: gzip, deflate
  Accept-Language: en-US,en;q=0.5
  Cache-Control: max-age=0
  Connection: keep-alive
  Cookie: JSESSIONID=9A65DC7A61FFEFB951CB091CD741F74A
  Host: localhost:8080
  Upgrade-Insecure-Requests: 1
  User-Agent: Mozilla/5.0 (Windows NT 10.0; ...) Gecko/20100101 Firefox/63.0
```

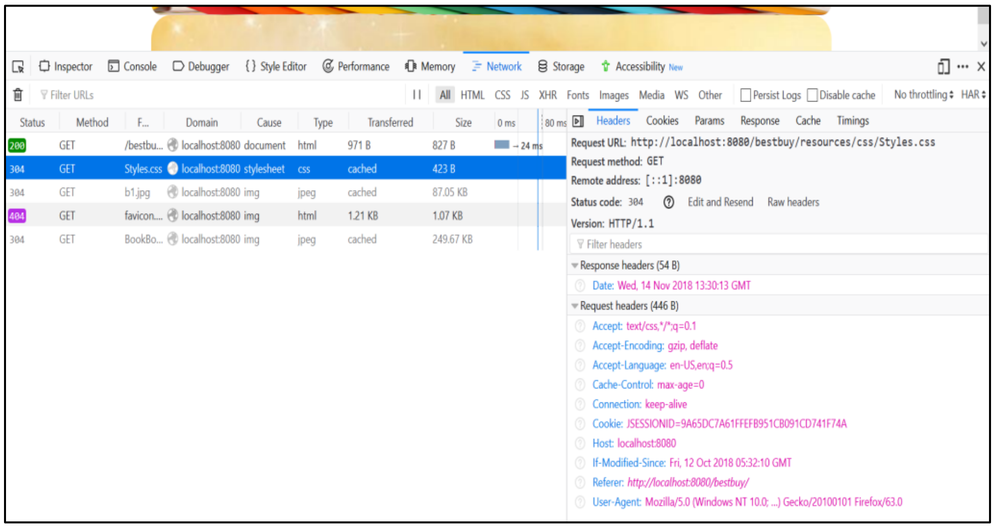
Figure 29. Headers tab details of the GET method created using JavaScript.



```
Status  Method  F...         Domain          Cause       Type  Transferred  Size       0 ms
200     GET     /bestbu...   localhost:8080  document    html  971 B        827 B      → 24 ms
304     GET     Styles.css   localhost:8080  stylesheet  css   cached       423 B
304     GET     b1.jpg       localhost:8080  img         jpeg  cached       87.05 KB
404     GET     favicon....  localhost:8080  img         html  1.21 KB      1.07 KB
304     GET     BookBo...    localhost:8080  img         jpeg  cached       249.67 KB
```

```
Blocked:         → 1 ms
DNS resolution:  → 0 ms
Connecting:      → 0 ms
TLS setup:       → 0 ms
Sending:         → 0 ms
Waiting:         → 23 ms
Receiving:       → 0 ms
```

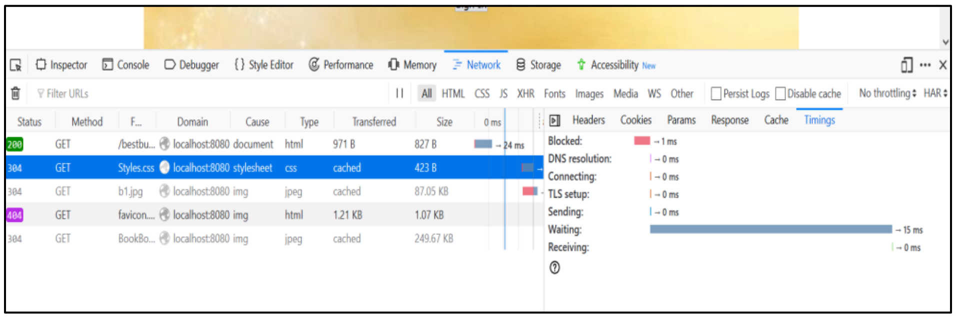
Figure 30. Timings details for the GET method created using JavaScript.



```
Status  Method  F...         Domain          Cause       Type  Transferred  Size       0 ms
200     GET     /bestbu...   localhost:8080  img         html  971 B        827 B      → 24 ms
304     GET     Styles.css   localhost:8080  stylesheet  css   cached       423 B
304     GET     b1.jpg       localhost:8080  img         jpeg  cached       87.05 KB
404     GET     favicon....  localhost:8080  img         html  1.21 KB      1.07 KB
304     GET     BookBo...    localhost:8080  img         jpeg  cached       249.67 KB
```

```
Request URL: http://localhost:8080/bestbuy/resources/css/Styles.css
Request method: GET
Remote address: [::1]:8080
Status code: 304
Version: HTTP/1.1
Response headers (54 B)
  Date: Wed, 14 Nov 2018 13:30:13 GMT
Request headers (446 B)
  Accept: text/css,*/*;q=0.1
  Accept-Encoding: gzip, deflate
  Accept-Language: en-US,en;q=0.5
  Cache-Control: max-age=0
  Connection: keep-alive
  Cookie: JSESSIONID=9A65DC7A61FFEFB951CB091CD741F74A
  Host: localhost:8080
  If-Modified-Since: Fri, 12 Oct 2018 05:32:10 GMT
  Referer: http://localhost:8080/bestbuy/
  User-Agent: Mozilla/5.0 (Windows NT 10.0; ...) Gecko/20100101 Firefox/63.0
```

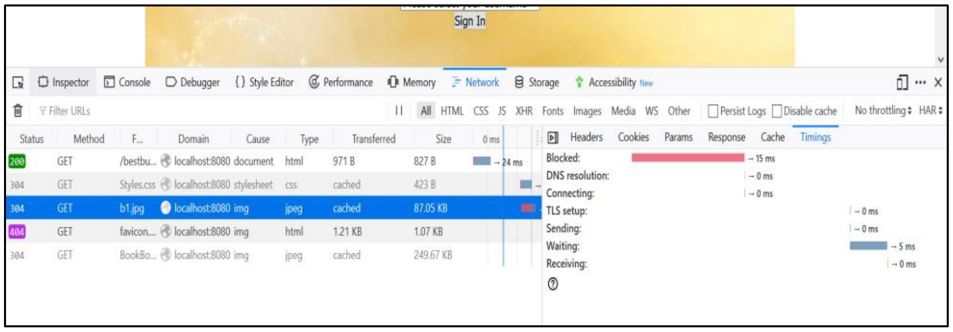
Figure 31. Headers details of styles.css in home page created using JavaScript.



Status	Method	F...	Domain	Cause	Type	Transferred	Size	0 ms
200	GET	/bestbu...	localhost:8080	document	html	971 B	827 B	→ 24 ms
304	GET	Styles.css	localhost:8080	stylesheet	css	cached	423 B	
304	GET	b1.jpg	localhost:8080	img	jpeg	cached	87.05 KB	
404	GET	favicon...	localhost:8080	img	html	1.21 KB	1.07 KB	
304	GET	BookBo...	localhost:8080	img	jpeg	cached	249.67 KB	

```
Blocked:          → 1 ms
DNS resolution:   → 0 ms
Connecting:       → 0 ms
TLS setup:        → 0 ms
Sending:          → 0 ms
Waiting:          → 15 ms
Receiving:        → 0 ms
```

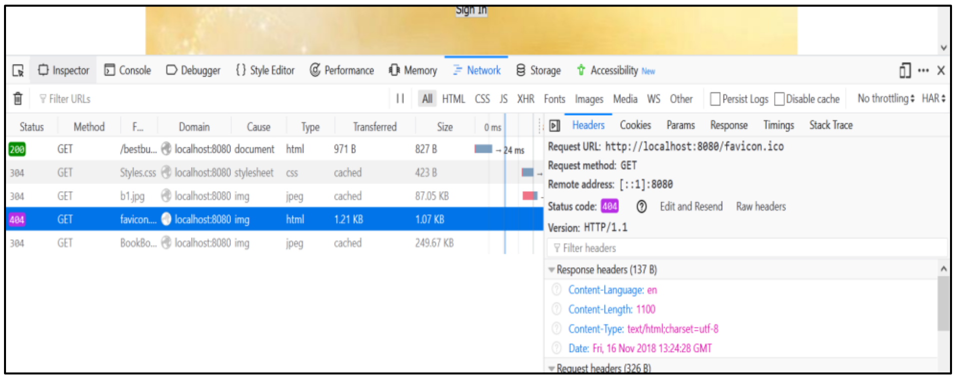
Figure 32. Timings tab of styles.css in home page created using JavaScript.



Status	Method	F...	Domain	Cause	Type	Transferred	Size	0 ms
200	GET	/bestbu...	localhost:8080	document	html	971 B	827 B	→ 24 ms
304	GET	Styles.css	localhost:8080	stylesheet	css	cached	423 B	
304	GET	b1.jpg	localhost:8080	img	jpeg	cached	87.05 KB	
404	GET	favicon...	localhost:8080	img	html	1.21 KB	1.07 KB	
304	GET	BookBo...	localhost:8080	img	jpeg	cached	249.67 KB	

```
Blocked:          → 15 ms
DNS resolution:   → 0 ms
Connecting:       → 0 ms
TLS setup:        → 0 ms
Sending:          → 0 ms
Waiting:          → 5 ms
Receiving:        → 0 ms
```

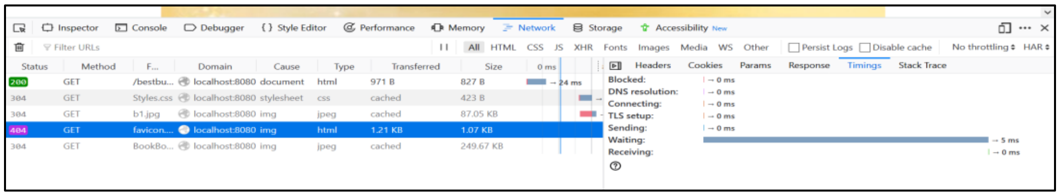
Figure 33. Headers details of the image1 in home page created using JavaScript



Status	Method	F...	Domain	Cause	Type	Transferred	Size	0 ms
200	GET	/bestbu...	localhost:8080	document	html	971 B	827 B	→ 24 ms
304	GET	Styles.css	localhost:8080	stylesheet	css	cached	423 B	
304	GET	b1.jpg	localhost:8080	img	jpeg	cached	87.05 KB	
404	GET	favicon...	localhost:8080	img	html	1.21 KB	1.07 KB	
304	GET	BookBo...	localhost:8080	img	jpeg	cached	249.67 KB	

```
Request URL: http://localhost:8080/favicon.ico
Request method: GET
Remote address: [::1]:8080
Status code: 404
Version: HTTP/1.1
Response headers (137 B)
  Content-Language: en
  Content-Length: 1100
  Content-Type: text/html;charset=utf-8
  Date: Fri, 16 Nov 2018 13:24:28 GMT
Request headers (326 B)
```

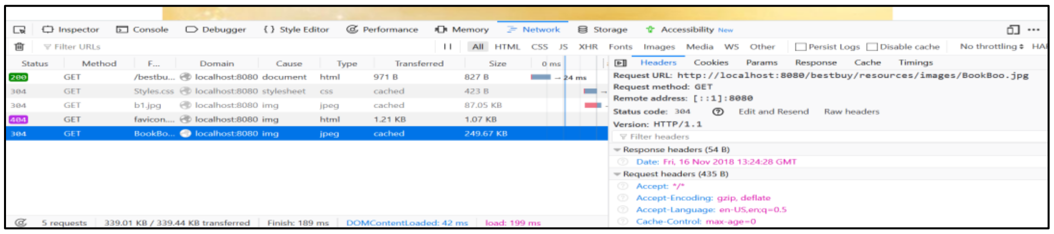
Figure 34. Headers details of the image2 in home page created using JavaScript.



Status	Method	F...	Domain	Cause	Type	Transferred	Size	0 ms
200	GET	/bestbu...	localhost:8080	document	html	971 B	827 B	→ 24 ms
304	GET	Styles.css	localhost:8080	stylesheet	css	cached	423 B	
304	GET	b1.jpg	localhost:8080	img	jpeg	cached	87.05 KB	
404	GET	favicon...	localhost:8080	img	html	1.21 KB	1.07 KB	
304	GET	BookBo...	localhost:8080	img	jpeg	cached	249.67 KB	

```
Blocked:          → 0 ms
DNS resolution:   → 0 ms
Connecting:       → 0 ms
TLS setup:        → 0 ms
Sending:          → 0 ms
Waiting:          → 5 ms
Receiving:        → 0 ms
```

Figure 35. Timings details of the image2 in home page created using JavaScript.



Status	Method	F...	Domain	Cause	Type	Transferred	Size
200	GET	/bestbu...	localhost:8080	document	html	971 B	827 B
304	GET	Styles.css	localhost:8080	stylesheet	css	cached	423 B
304	GET	b1.jpg	localhost:8080	img	jpeg	cached	87.05 KB
404	GET	favicon....	localhost:8080	img	html	1.21 KB	1.07 KB
304	GET	BookBo...	localhost:8080	img	jpeg	cached	249.67 KB

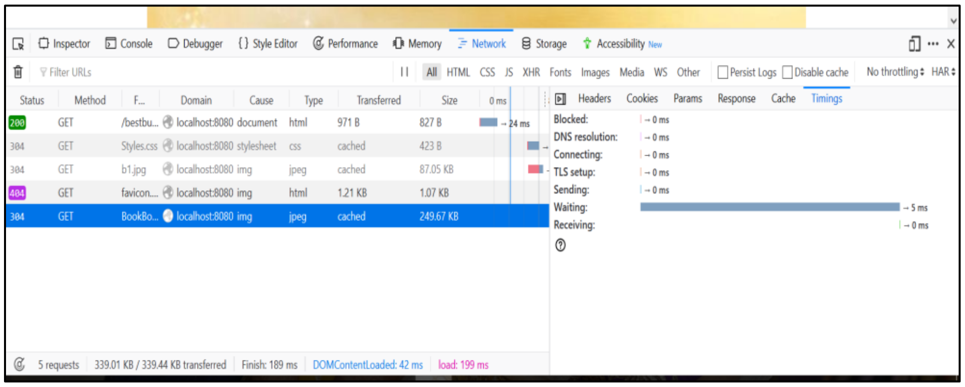
5 requests | 339.01 KB / 339.44 KB transferred | Finish: 189 ms | DOMContentLoaded: 42 ms | load: 199 ms

Request URL: http://localhost:8080/bestbuy/resources/images/BookBoo.jpg
Request method: GET
Remote address: [::1]:8080
Status code: 304
Version: HTTP/1.1

Response headers (54 B)
Date: Fri, 16 Nov 2018 13:24:28 GMT

Request headers (435 B)
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.5
Cache-Control: max-age=0

Figure 36. Headers details of the image3 in home page created using JavaScript.



Status	Method	F...	Domain	Cause	Type	Transferred	Size
200	GET	/bestbu...	localhost:8080	document	html	971 B	827 B
304	GET	Styles.css	localhost:8080	stylesheet	css	cached	423 B
304	GET	b1.jpg	localhost:8080	img	jpeg	cached	87.05 KB
404	GET	favicon....	localhost:8080	img	html	1.21 KB	1.07 KB
304	GET	BookBo...	localhost:8080	img	jpeg	cached	249.67 KB

5 requests | 339.01 KB / 339.44 KB transferred | Finish: 189 ms | DOMContentLoaded: 42 ms | load: 199 ms

Blocked:	→ 0 ms
DNS resolution:	→ 0 ms
Connecting:	→ 0 ms
TLS setup:	→ 0 ms
Sending:	→ 0 ms
Waiting:	→ 5 ms
Receiving:	→ 0 ms

Figure 37. Timings details of the image3 in home page created using JavaScript

From the screenshot of Network tab of an applet, it is noticed that the time taken to load the request over the network is, 2ms whereas the time taken to load the JavaScript page with the same number of elements are, 60ms including the index page and the elements such as CSS styling request, images, etc. From this, it is noticed that the applets comparatively take less time to load the same elements over network compared to JavaScript. However, the difference between the two pages is comparatively less (in milliseconds).

Network analysis of JavaScript web page.

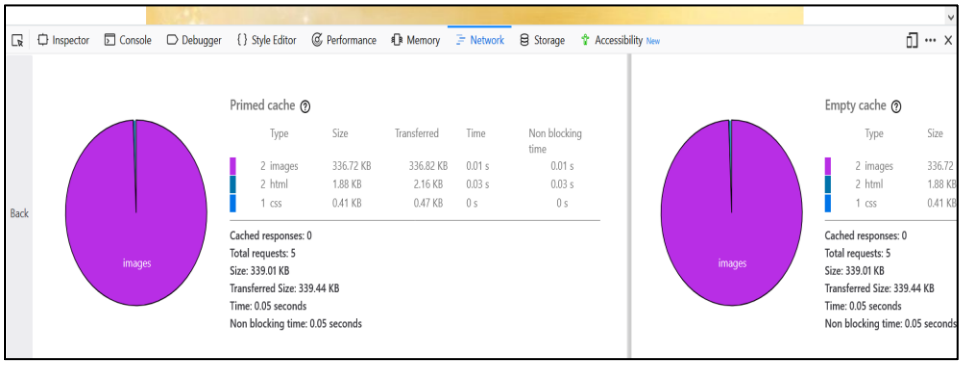
Comparison.

Table 3

Network Analysis of Applet and JavaScript

	Applets	**JavaScript**
No. of requests needed to load the web page over Network	1	5
No. of web elements that are added in the home page	5	5
Type of Request	GET	GET
Overall time taken to load all elements	2 ms	60 ms
Time taken to load elements when loading the page for the second time	1 ms	60 ms

Performance analysis section within network tab of home page created using javaScript.



Primed cache

	Type	Size	Transferred	Time	Non blocking time
🟪	2 images	336.72 KB	336.82 KB	0.01 s	0.01 s
🟦	2 html	1.88 KB	2.16 KB	0.03 s	0.03 s
🟦	1 css	0.41 KB	0.47 KB	0 s	0 s

Cached responses: 0
Total requests: 5
Size: 339.01 KB
Transferred Size: 339.44 KB
Time: 0.05 seconds
Non blocking time: 0.05 seconds

Empty cache

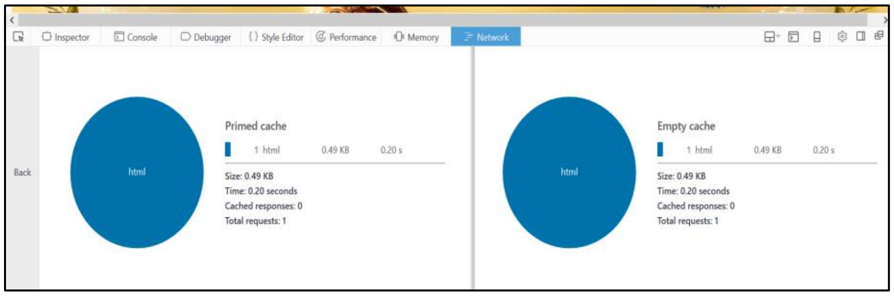
	Type	Size
🟪	2 images	336.72
🟦	2 html	1.88 KB
🟦	1 css	0.41 KB

Cached responses: 0
Total requests: 5
Size: 339.01 KB
Transferred Size: 339.44 KB
Time: 0.05 seconds
Non blocking time: 0.05 seconds

Figure 38. Performance analysis of JavaScript page.

The above performance analysis section displays the number of requests, the time taken, and size required to load the web page over the network. This section is the same as the above Network analysis, but the results are instead shown as a pie chart so that users may obtain a better overview of the above results.

Performance analysis of home page created using applet.s.



Primed cache

▮	1 html	0.49 KB	0.20 s

Size: 0.49 KB
Time: 0.20 seconds
Cached responses: 0
Total requests: 1

Empty cache

▮	1 html	0.49 KB	0.20 s

Size: 0.49 KB
Time: 0.20 seconds
Cached responses: 0
Total requests: 1

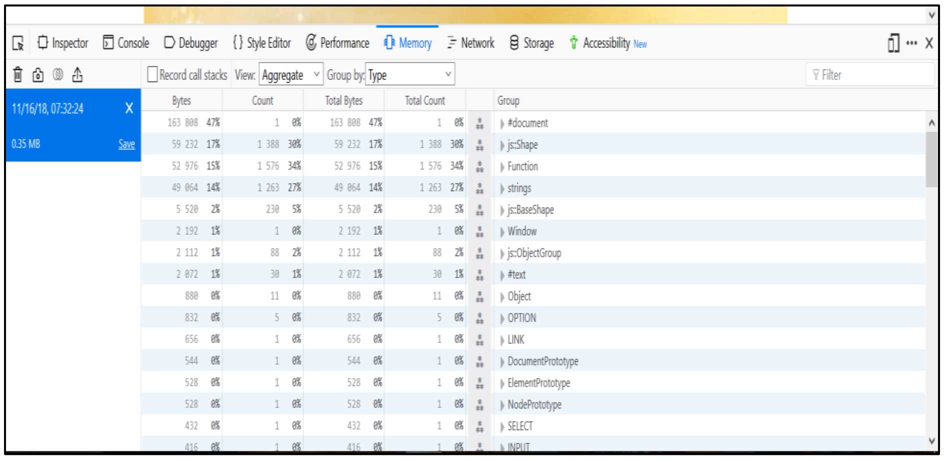
Figure 39. Performance analysis within network tab of applet page.

The above performance tab shows detailed information on the time taken for the browser to download the elements of the web page.

Comparison. The analysis results are the same as those obtained in the above table that compared the Network tab of both the JavaScript and applet-driven applications.

Memory tab analysis of JavaScript and applets: The screenshot of the Memory tab of the home page created using JavaScript is shown below. The Memory tab displays the heap memory overview taken to store the elements that are needed to load the page. In order to analyze the memory of these two pages, the web pages are opened in aggregate view to display the memory overview in a table format.

Memory overview of JavaScript home page.



Record call stacks View: Aggregate Group by: Type

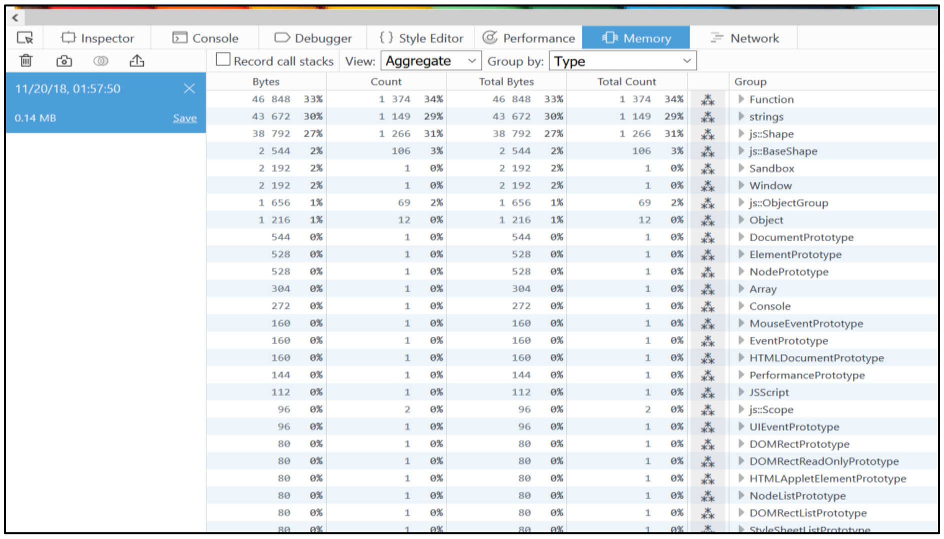
11/16/18, 07:32:24
0.35 MB Save

Bytes		Count		Total Bytes		Total Count		Group
163 808	47%	1	0%	163 808	47%	1	0%	#document
59 232	17%	1 388	30%	59 232	17%	1 388	30%	js::Shape
52 976	15%	1 576	34%	52 976	15%	1 576	34%	Function
49 064	14%	1 263	27%	49 064	14%	1 263	27%	strings
5 520	2%	230	5%	5 520	2%	230	5%	js::BaseShape
2 192	1%	1	0%	2 192	1%	1	0%	Window
2 112	1%	88	2%	2 112	1%	88	2%	js::ObjectGroup
2 072	1%	30	1%	2 072	1%	30	1%	#text
880	0%	11	0%	880	0%	11	0%	Object
832	0%	5	0%	832	0%	5	0%	OPTION
656	0%	1	0%	656	0%	1	0%	LINK
544	0%	1	0%	544	0%	1	0%	DocumentPrototype
528	0%	1	0%	528	0%	1	0%	ElementPrototype
528	0%	1	0%	528	0%	1	0%	NodePrototype
432	0%	1	0%	432	0%	1	0%	SELECT
416	0%	1	0%	416	0%	1	0%	INPUT

Figure 40. Memory analysis of JavaScript page.

The above figure displays the list of all the elements and its memory consumption of each element.

Memory tab of the home page created using applets.



Record call stacks View: Aggregate Group by: Type

11/20/18, 01:57:50
0.14 MB Save

Bytes		Count		Total Bytes		Total Count		Group
46 848	33%	1 374	34%	46 848	33%	1 374	34%	Function
43 672	30%	1 149	29%	43 672	30%	1 149	29%	strings
38 792	27%	1 266	31%	38 792	27%	1 266	31%	js::Shape
2 544	2%	106	3%	2 544	2%	106	3%	js::BaseShape
2 192	2%	1	0%	2 192	2%	1	0%	Sandbox
2 192	2%	1	0%	2 192	2%	1	0%	Window
1 656	1%	69	2%	1 656	1%	69	2%	js::ObjectGroup
1 216	1%	12	0%	1 216	1%	12	0%	Object
544	0%	1	0%	544	0%	1	0%	DocumentPrototype
528	0%	1	0%	528	0%	1	0%	ElementPrototype
528	0%	1	0%	528	0%	1	0%	NodePrototype
304	0%	1	0%	304	0%	1	0%	Array
272	0%	1	0%	272	0%	1	0%	Console
160	0%	1	0%	160	0%	1	0%	MouseEventPrototype
160	0%	1	0%	160	0%	1	0%	EventPrototype
160	0%	1	0%	160	0%	1	0%	HTMLDocumentPrototype
144	0%	1	0%	144	0%	1	0%	PerformancePrototype
112	0%	1	0%	112	0%	1	0%	JSScript
96	0%	2	0%	96	0%	2	0%	js::Scope
96	0%	1	0%	96	0%	1	0%	UIEventPrototype
80	0%	1	0%	80	0%	1	0%	DOMRectPrototype
80	0%	1	0%	80	0%	1	0%	DOMRectReadOnlyPrototype
80	0%	1	0%	80	0%	1	0%	HTMLAppletElementPrototype
80	0%	1	0%	80	0%	1	0%	NodeListPrototype
80	0%	1	0%	80	0%	1	0%	DOMRectListPrototype
80	0%	1	0%	80	0%	1	0%	StyleSheetListPrototype

Figure 41. Memory analysis of applet page.

The above screenshot shows the Memory analysis of home page created using the applet and its heap memory usage.

Memory comparison of applet and JavaScript.

Table 4

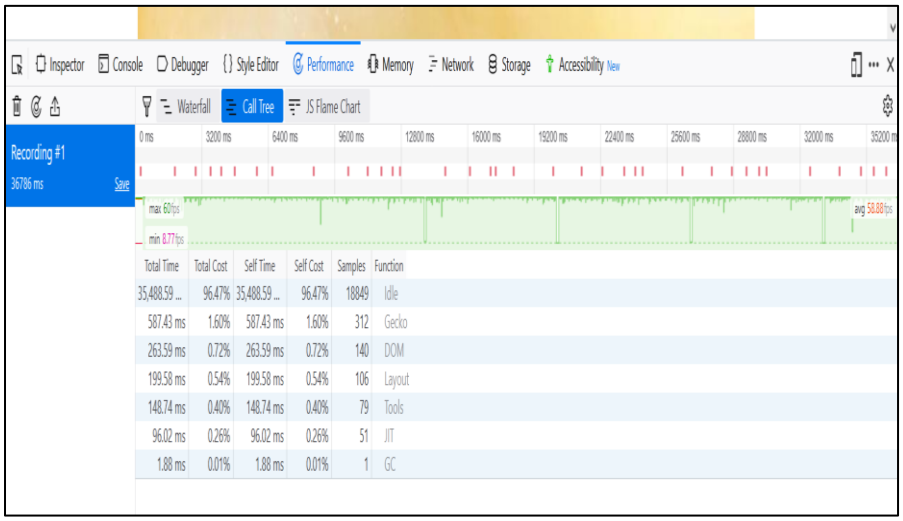
Memory comparison of Applet and JavaScript

	Applet	**JavaScript**
function	33%	22%
strings	30%	27%
array	0%	1%
js::Shape	27%	19%
JSScript	0%	9%
Object	1%	8%
js::Scope	0%	4%
js::ObjectGroup	1%	3%
Js::BaseShape	2%	1%

From the above table, it is noticed that applets consume more memory than JavaScript.

Performance analysis: Performance Analysis gives the overview of general responsiveness of the web page, JavaScript and layout performance. In order to compare the performance overview of applet and JavaScript, the Call Tree tab is taken into consideration as it gives the overview of JavaScript functions in which the browser spends most of its time.

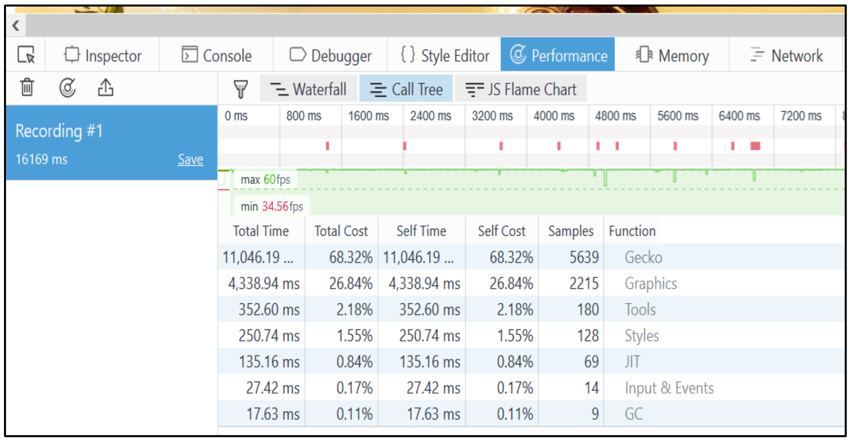
Performance analysis of JavaScript: The performance analysis screenshot of the home page created using JavaScript is shown in the following figure. In order to analyze the performance further, the Call Tree tab is being considered and is shown in the below picture.



Total Time	Total Cost	Self Time	Self Cost	Samples	Function
35,488.59 ...	96.47%	35,488.59 ...	96.47%	18849	Idle
587.43 ms	1.60%	587.43 ms	1.60%	312	Gecko
263.59 ms	0.72%	263.59 ms	0.72%	140	DOM
199.58 ms	0.54%	199.58 ms	0.54%	106	Layout
148.74 ms	0.40%	148.74 ms	0.40%	79	Tools
96.02 ms	0.26%	96.02 ms	0.26%	51	JIT
1.88 ms	0.01%	1.88 ms	0.01%	1	GC

Figure 42. CallTree view of performance analysis for JavaScript page.

Performance analysis of applets: The performance tab of the home page using applets is shown in the following screenshot. To analyze the performance, the Call Tree tab is considered created in the same way as the performance analysis of JavaScript.



Total Time	Total Cost	Self Time	Self Cost	Samples	Function
11,046.19 ...	68.32%	11,046.19 ...	68.32%	5639	Gecko
4,338.94 ms	26.84%	4,338.94 ms	26.84%	2215	Graphics
352.60 ms	2.18%	352.60 ms	2.18%	180	Tools
250.74 ms	1.55%	250.74 ms	1.55%	128	Styles
135.16 ms	0.84%	135.16 ms	0.84%	69	JIT
27.42 ms	0.17%	27.42 ms	0.17%	14	Input & Events
17.63 ms	0.11%	17.63 ms	0.11%	9	GC

Figure 43. CallTree view of performance analysis for applet page.

Performance analysis comparison.

Table 5

Performance Analysis Comparison of Applet and JavaScript

	Applet (Time Taken to load)	**JavaScript (Time Taken to load)**
Gecko	11,046 ms	587.43 ms
Tools	352.60 ms	148.74 ms
JIT	135.16 ms	96.02 ms
GC	17.63 ms	1.88 ms

To compare the performance of applets and JavaScript, the common elements while loading the page and the time taken to load these elements are considered. From the above table, it is noticed that the time taken to load these elements for applets is relatively longer compared to JavaScript. It is therefore concluded that the performance for JavaScript is better when compared with applets.

Security comparison of applets and javaScript: The different types of security models that are incorporated in each technology are explained as follows.

Security of applets: Applets provide two types of security models: signed applets and unsigned applets. As of Java SE 7 Update 21 (April 2013), applets are encouraged to allow signed applets. However, unsigned applets are still allowed with a warning message displayed on the web page when run. Later with the Java 7 Update 51, unsigned applets were blocked by default and these applets now must to be run by adding the site as a Trusted Site in the Java Control Panel.

Unsigned: Unsigned applets are assigned by default when an applet is run. Unsigned applets have many security restrictions. For instance, they cannot execute external commands on a local system or cannot access all system properties. The reason for these security restrictions is that the web browser starts the applet automatically and not on a user's initiative; therefore, the user cannot take responsibility for the application's behavior on their machine. These unsigned applets are also called untrusted applets or sandbox applets.

In order to further eliminate the security restrictions on the local machine, signed applets have been introduced, of which there are two types: self-signed applets and signed applets.

Signed: A signed applet will generate a digital signature that the browser should verify before running the applet, by which the applet will then obtain complete access to the local machine. By digital signature verification and the user accepting the signature, applets become trusted applets and will gain complete access to the local machine. These applets are also called privileged applets. User authentication is achieved with the generation of digital signatures with user approval. Thus, the signed applets provide security.

Self-signed: Self-signed applets are almost the same as signed applets, except that these applets are signed by the developers instead of a third-party. This security model is also used to provide user authentication. However, these applets are usually used during development of the application and before the release of the application.

Security model of JavaScript: JavaScript also has its own security model, but the basic difference between applets JavaScript security model is that the security model for JavaScript was designed to protect users from malicious web sites. Conversely, the applet security model protects the owner of the web site and shields the data that is being passed over the network between browser and server.

However, security can be accomplished in JavaScript by using Google's two-factor authentication. A Java applet is not appreciably more secure than the two-factor authentication without Java. The multi-factor authentication may be implemented on both the form and the applet. The encryption process is handled by the web browser and the web server for the information on transit via SSL/TLS. This process is independent of the underlying technology (applet, or HTML form).

Security comparison of applets and JavaScript: JavaScript is open source, and any user can view the script that is being used to accomplish a technical objective. This is a plus for security and a minus for protecting intellectual property, whereas the code of the applet is not visible to the users when it is embedded in HTML. The security model in applets also provides the security for the data passed between browser and server and authenticates the user. Accessing the user's machine can also be restricted with unsigned applets, whereas the JavaScript security model is used to protect the user from other malicious browsers.

Comparison.

Table 6

Security Comparison of Applet and JavaScript

	Applets	**JavaScript**
Is the code accessible to the user?	No	Yes
Security Model	Provides unsigned, signed and self-signed applets to authenticates the user, and to protect the data that is passed from server to browser.	Provides the security model to restrict the user from other malicious web browsers by restricting them.
User Authentication	Is provided by using signed and self-signed applets. However, these are not as secure as external multi-factor authentication techniques.	JavaScript does not provide any signatures by default. Google Multi-factor authentication can be used to provide security which is an external tool and is common to both Applets and JavaScript.
Access to user's local machine	Access to the user's local machine can be restricted with the use of unsigned applets and can be accessed with the use of signed applets and self-signed applets.	JavaScript can access the user's local machine by default, and there are no restrictions on this usage.

General features, advantages and disadvantages of javaScript and applets. For further analysis on these two technologies, along with the above comparison, the general features, advantages, and disadvantages of both JavaScript and applets are taken into consideration from the web and are explained as below:

Advantages ofusing JavaScript.

- Less server interaction
- Immediate feedback to the visitors
- Increased interactivity
- Richer interfaces
- Simplicity. It is relatively simple to implement web pages using JavaScript.
- Versatility. Javascript can be used in many applications for which back end scripting is written in other languages such as PHP.
- Server Load. JavaScript reduces the server load, which makes it lightweight (Advantages and Disadvantages of Applets, n.d.).

Advantages of using applets.

- Applets are supported on all operating systems such as Linux or MS Windows and are simple to make an application cross-platform. Applets are supported by most web browsers.
- Applets are cached in most of the web browsers which will make the web page quick while returning or reloading the page. Also, applets are relatively fast with usage in the web application. If a web page uses the applet, JVM starts and is in running condition. When navigating to another web page which uses another web page, applets will take less time to load as the JVM is already running. However, from JRE version 1.5 and greater, when a user navigates from one web page to another, it would stop the JVM in the first web page and restarts the JVM on the next, which would take the same time to load each page containing applets.

- Untrusted applets can not access the local machine in which an applet is running and can only have access to the server from which the applet came from. This makes applets more secure. However, signed applets can have full access to the local machine in which an applet is running.
- Applets are relatively faster and can even have similar performance to natively installed software (Advantages and Disadvantages of Applets, n.d.).

Disadvantages of using JavaScript.

- Security. As the code is accessible to the user and executes on the user's machine, the code is vulnerable to security attacks, such as cross-site scripting.
- JavaScript code sometimes changes for different browsers, whereas server-side scripting output is the same for all the browsers (Advantages and Disadvantages of Applets, n.d.).

Disadvantages of using applets.

- Applets require separate Java plug-in.
- Mobile browsers that run Apple iOS and Android applications do not support applets.
- Google's Chrome browser is phasing out all plugins related to applets, will effectively kill both Java and Flash as programming platforms in the browser.
- With the security restriction in Java untrusted applets, it is sometimes impossible to achieve desired goals; for example, a web application may require the client-side program to access a local machine. However, this can still be accessed in applets by editing the java.policy file within JAVA JRE installation (Advantages and Disadvantages of Applets, n.d.).

Chapter V: Results, Conclusion, and Recommendations

Introduction

In this chapter, the results obtained after comparing applets and JavaScript are examined. Also, this chapter discusses the future work and limitations of this comparison.

Results

From the above analysis of network, memory, and performance of both the applications created using applets and JavaScript, it is understood that applets are comparatively faster than JavaScript. When comparing the study of applets and JavaScript, it is observed that applets are more secure than JavaScript. However, applets require a separate plugin whenever they are running on the browser, as consumers do not often use applets. The following table explains the results obtained after the comparison in terms of each factor that can be considered while developing any web application.

Table 7

Overall Analysis of Applet and JavaScript

	Applets	**JavaScript**
Speed	Faster than JavaScript	Slower than applets
Security	More Secure than JavaScript	Less secured than applets
Network	Faster to load elements over network	Slower to load elements over the network
Memory	Consumes more memory than JavaScript	Consumes less memory than applets
Performance	Applets have lower performance over JavaScript	JavaScript has better performance over applets
Environments supported	Supports all operating systems except mobile applications	Supports all operating systems including mobile applications
Type of applications that can be developed	Stand-alone applications, web applications can be developed using applets	Web, mobile applications can be developed using JavaScript.
External Plugins Required	Yes. It requires a Java Virtual Machine installed on the client’s machine.	No external plugin is required to view the web page.
Development environments supported	Applets can be used only when the server-side programming is developed using Java.	JavaScript can be used for most of the server-side programmings such as Java, Python, Ruby, etc.
Is the application client-side?	The business logic will reside on the server side and within the web application or web page.	The business logic will reside on the web page. Server integration is not needed to drive the application, which makes the application client-side.
User-interactions that the application delivers	User cannot view/edit the applet code from the web browser.	Users can view/update the javaScript code from the web browser, which makes it readable to the user.

Conclusion

Based on the above results, it is concluded that applets can still be a feasible option in many environments and can not be ignored as it has advantages such as its speed, security, etc. Applets can be used when the application is not needed to be supported by mobile devices and when the consumers of the application have JVM installed on their machines. Also, applets are best used in gaming, graphical interfaces, etc. Applets also work better in stand-alone applications.

JavaScript can be used when the application must be supported on mobile devices. Additionally, JavaScript needs to be used when the application has numerous users who do not have JVM already installed in their machines. JavaScript can also be used when the server-side programming is not Java, as it supports multiple server-side programming languages.

With applets, the usual concurrency issues apply and is not safe to share the thread data. With JavaScript, there are no such issues because the code runs inside a single thread (David Balme, 2017).

Future Work

This study compares only the home page of both the applet and JavaScript applications, which receive the response from the server and display the user details from the database. This study can be investigated further in the future by comparing other pages (e.g., the Order page, which sends the request details to the server) and updating some data in the database. Multi-threading and thread safety can also be compared in the future to evaluate further the best technology to use.

References

Advantages & Disadvantages of Applets. (n.d.). saRetrieved from Study.com: mcatee@stcloudstate.edu*Advantages and Disavntages of Applets*.

Byrne, M. (2016, February). *The rise and fall of the Java Applet: Creative coding's awkward little square*. Retrieved from Motherboard: https://motherboard.vice.com/en_us/article/8q8n3k/a-brief-history-of-the-java-applet.

Dandan Zhang, Z. W. (2013). *Research on lightweight MVC framework based on spring MVC*. Research on Lightweight MVC Framework Based on Spring MVC.

David Balme, S. D. (2017). *If Java applets are outdated, what should i use?* Retrieved from Quora: https://www.quora.com/If-java-applets-are-outdated-what-should-i-use.

Java applet. (2010). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Java_applet.

Mario Callegaro, K. L. (2015). Web survey methodology. In K. L. Mario Callegaro, *Web survey methodology* (p. 344).

Memory. (n.d.). Retrieved from MDN Web Tools: https://developer.mozilla.org/en-US/docs/Tools/Memory.

Network monitor. (n.d.). Retrieved from MDN Web Tools: https://developer.mozilla.org/en-US/docs/Tools/Network_Monitor.

Network monitor toolbar. (n.d.). Retrieved from MDN Web Tools: https://developer.mozilla.org/en-US/docs/Tools/Network_Monitor/Toolbar.

Network request list. (n.d.). Retrieved from MDN Web Tools: https://developer.mozilla.org/en-US/docs/Tools/Network_Monitor/request_list.

Parekh, P. O. (2006). *Java applet based database management interface.* Retrieved from https://www.semanticscholar.org/paper/Java-Applet-Based-Database-Management-Interface-Parekh/053c80016cf8f1e3d31de868be32c29c8e8625b0?navId=citing-papers.

Performance. (n.d.). Retrieved from MDN Web Tools: https://developer.mozilla.org/en-US/docs/Tools/Performance.

Rohan Khanna, P. P. (2014). *Study of spring framework.* Study of Spring Framework.

Spring Framework—Architecture. (n.d.). *Tutorial point.* Retrieved from tutorialspoint: https://www.tutorialspoint.com/spring/spring_architecture.htm

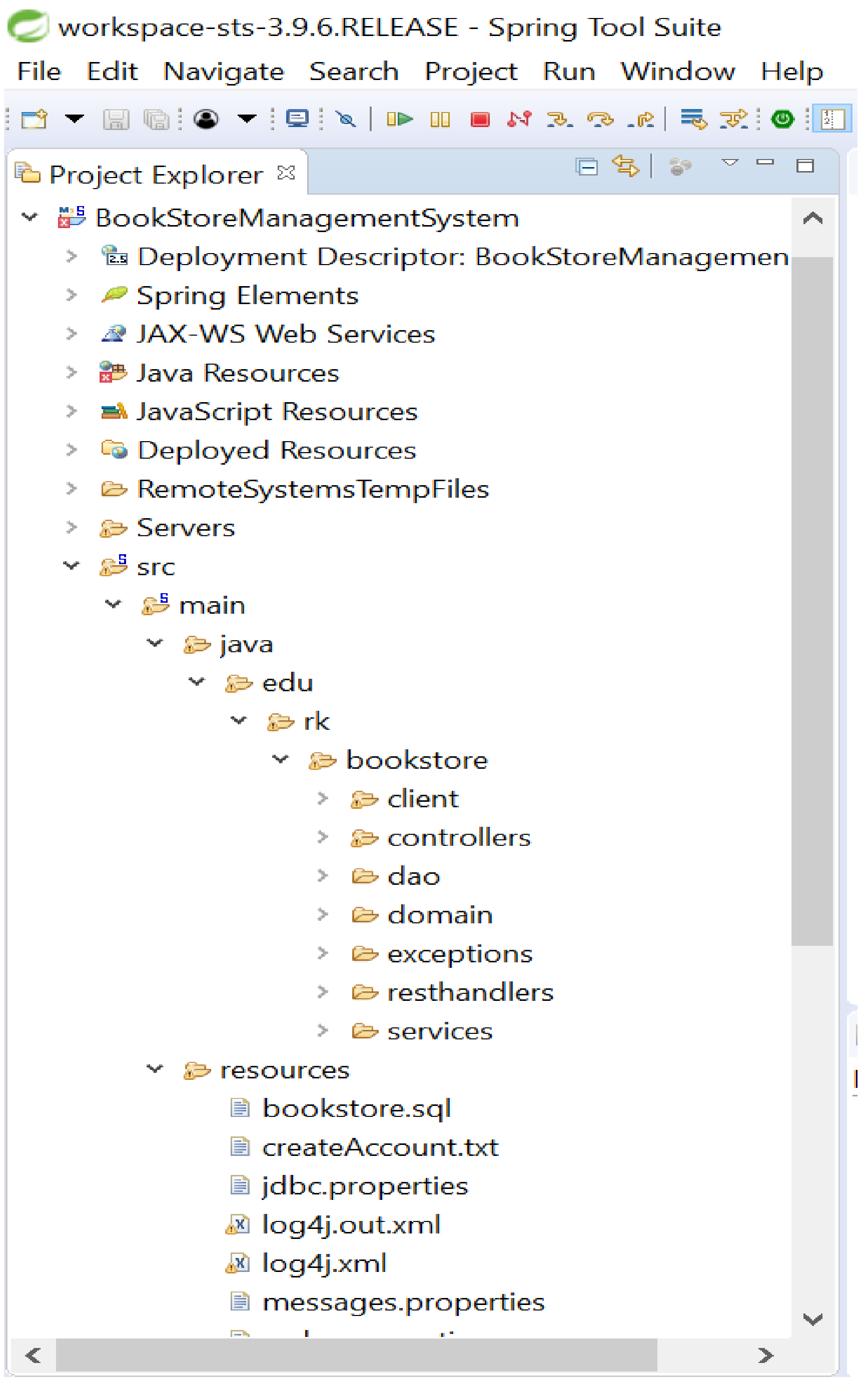
Wikipedia. (n.d.). *MVC.* Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller.

Wikipedia. (n.d.). *JSP*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/JavaServer_Pages.

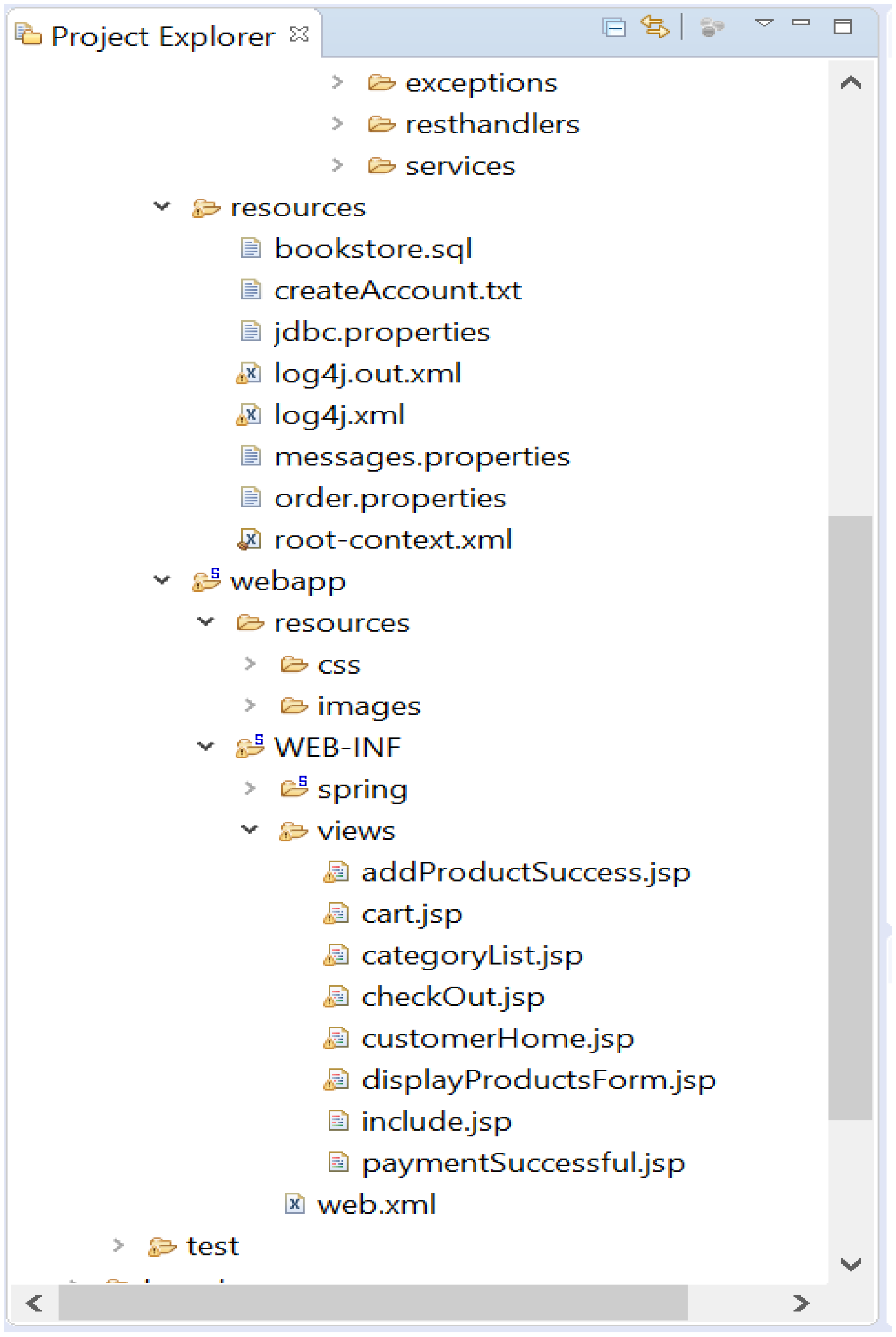
Wkipedia. (n.d.). *Applet*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Applet.

Appendix

Project Structure of Book Store Management System


workspace-sts-3.9.6.RELEASE - Spring Tool Suite
File Edit Navigate Search Project Run Window Help
Project Explorer

```
BookStoreManagementSystem
  Deployment Descriptor: BookStoreManagemen
  Spring Elements
  JAX-WS Web Services
  Java Resources
  JavaScript Resources
  Deployed Resources
  RemoteSystemsTempFiles
  Servers
  src
    main
      java
        edu
          rk
            bookstore
              client
              controllers
              dao
              domain
              exceptions
              resthandlers
              services
      resources
        bookstore.sql
        createAccount.txt
        jdbc.properties
        log4j.out.xml
        log4j.xml
        messages.properties
```



```
      > exceptions
      > resthandlers
      > services
  v resources
      bookstore.sql
      createAccount.txt
      jdbc.properties
      log4j.out.xml
      log4j.xml
      messages.properties
      order.properties
      root-context.xml
  v webapp
    v resources
      > css
      > images
    v WEB-INF
      > spring
      v views
          addProductSuccess.jsp
          cart.jsp
          categoryList.jsp
          checkOut.jsp
          customerHome.jsp
          displayProductsForm.jsp
          include.jsp
          paymentSuccessful.jsp
        web.xml
> test
```

DAO Pages:

BillingSystem.java

```
package edu.rk.bookstore.client;

import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

import edu.rk.bookstore.domain.Category;
import edu.rk.bookstore.domain.Customer;
import edu.rk.bookstore.domain.Order;
import edu.rk.bookstore.domain.OrderItem;
import edu.rk.bookstore.domain.Product;
import edu.rk.bookstore.services.BillingService;
import edu.rk.bookstore.services.ProductService;

public class BillingSystem {
	public static void main(String[] args) {
		List<Product> prodList;
		List<OrderItem> oiList = new ArrayList<OrderItem>();
		OrderItem orderItem;
		Order order = new Order();
		Scanner scanner = new Scanner(System.in);
		int option;
		double subTotal = 0.0;
		AbstractApplicationContext container = new 
ClassPathXmlApplicationContext("root-context.xml");
		container.registerShutdownHook();
		ProductService productService = 
(ProductService)container.getBean("productService");
		List<Category> categoryList  = productService.getCategories();
		do{

	System.out.println("\n*****************************************\n"
);
			System.out.println("Please select a Category : ");
			for(Category c: categoryList){
				System.out.println(c.getCatId()+" : 
"+c.getCatName());
			}
			System.out.println("Enter 10 to quit!");

	System.out.println("\n*****************************************\n"
);
```

```
                option = scanner.nextInt();

                if(option >= 1 && option <= 6){
                        prodList = productService.getProducts(option);
                        int prodOption;
                        do{
                                int serialNum = 1;
                                System.out.println("\nPlease select a product 
to order");
                                for(Product p: prodList){
                                        System.out.println(serialNum+" : 
"+p.getName()+" : "+p.getPrice());
                                        serialNum++;
                                }
                                System.out.println("Enter 30 to quit!\n");
                                prodOption = scanner.nextInt();
                                orderItem = new OrderItem();
                                if(prodOption >= 1 && prodOption < 
serialNum){

        orderItem.setProduct(prodList.get(prodOption-1));
                                        System.out.print("Please enter quantity 
: ");

        orderItem.setQuantity(scanner.nextInt());
                                        subTotal += 
orderItem.getProduct().getPrice()*orderItem.getQuantity();
                                        oiList.add(orderItem);
                                }else if(prodOption == 30){
                                        System.out.println("\nGOING BACK TO 
CATEGORY LIST\n");
                                }else{
                                        System.out.print("\nPLEASE SELECT 
CORRECT PRODUCT\n");
                                }
                        }while(prodOption != 30);
                }else if(option == 10){
                        if(!oiList.isEmpty()){
                                order.setOiList(oiList);
                                Customer customer = new Customer();
                                customer.setId(1);
                                customer.setName("John");
                                customer.setState("California");
                                order.setCustomer(customer);
                                order.setSubtotal(subTotal);
                                BillingService billingService  = 
(BillingService)container.getBean("billingService");
                                billingService.computeTotalPrice(order);
                                System.out.println("************** YOUR ORDER 
IS **************");
                                for(OrderItem oi : oiList){
```

```
      System.out.println(oi.getProduct().getName()+" : "+
oi.getProduct().getPrice() +" : "+oi.getQuantity());
                              }
                              System.out.println("SubTotal : "+
order.getSubtotal());
                              System.out.println("Tax : "+ order.getTax());
                              System.out.println("Total : "+
order.getTotal());
                              try{

      billingService.processCustomerPurchase(order);
                              }catch(Exception e){
                                    e.printStackTrace();
                              }
                        }
                        System.out.println("\nTHANK YOU FOR SHOPPING WITH
US\n");
                  }else{
                        System.out.print("\nPLEASE SELECT CORRECT
CATEGORY\n");
                  }
            } while(option != 10);
      }
}
```

BookStoreController.java (under controllers package)

```
package edu.rk.bookstore.controllers;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;

import javax.servlet.http.HttpSession;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;
import edu.rk.bookstore.domain.Category;
import edu.rk.bookstore.domain.Customer;
import edu.rk.bookstore.domain.Order;
import edu.rk.bookstore.domain.OrderForm;
import edu.rk.bookstore.domain.OrderItem;
import edu.rk.bookstore.domain.Product;
import edu.rk.bookstore.services.BillingService;
import edu.rk.bookstore.services.CustomerService;
import edu.rk.bookstore.services.ProductService;

@Controller
public class BookStoreController {
      @Autowired
      private ProductService productService;
      @Autowired
      private BillingService billingService;
      @Autowired
      private CustomerService customerService;
      private static final Logger logger =
LoggerFactory.getLogger(BookStoreController.class);

      // Display customer details home page
            @RequestMapping(value = {"/", "home"}, method =
RequestMethod.GET)
            public ModelAndView home(HttpSession session) {
                  session.setAttribute("orderform", new OrderForm());
                  List<Customer> custList = customerService.getCustomers();
                  Map<Integer, Customer> custMap = new HashMap<Integer,
Customer>();
                  for(Customer c:custList){
                        custMap.put(c.getId(), c);
                  }
                  session.setAttribute("custMap", custMap);
                  ModelAndView modelView;
                  modelView = new ModelAndView("customerHome");
                  modelView.addObject("custList", custList);
                  modelView.addObject("customer", new Customer());
                  return modelView;
            }

// Displays all the categories
      @RequestMapping(value = "/categoryList", method = RequestMethod.GET)
      public ModelAndView categoryList(@ModelAttribute("customer")
Customer customer, HttpSession session) {
            OrderForm sesOrderForm =
(OrderForm)session.getAttribute("orderform");
            if(null == sesOrderForm.getCustomer() ||
(customer.getId()!=sesOrderForm.getCustomer().getId()&&customer.getId()>0)
){
                  Map<Integer, Customer> custMap = (Map<Integer, Customer>)
session.getAttribute("custMap");
                  Set<Integer> cusIds = (Set<Integer>) custMap.keySet();
                  Customer ctemp = new Customer();
                  for(Integer id:cusIds){
                        if(customer.getId() == id){
                              ctemp = custMap.get(id);
                        }
                  }
                  OrderForm orderform =
(OrderForm)session.getAttribute("orderform");
                  orderform.setCustomer(ctemp);
                  session.setAttribute("orderform", orderform);
            }
            ModelAndView modelView;
            List<Category> catList = productService.getCategories();
            modelView = new ModelAndView("categoryList");
            modelView.addObject("catList", catList);
            return modelView;
      }

      // Displays list of products for the selected category in the home
page
      @RequestMapping(value = "/displayProductsForm", method =
RequestMethod.GET)
      public ModelAndView displayProductsForm(@RequestParam(value="catId",
required = true) int catId, HttpSession session) {
            ModelAndView modelView;
            OrderForm sessOrderform = (OrderForm)
session.getAttribute("orderform");
            List<Product> prodList = productService.getProducts(catId);

            OrderForm orderform = new OrderForm();
            List<OrderItem> oiList = new ArrayList<OrderItem>();
            OrderItem oi;
            for(Product p: prodList){
                  oi = new OrderItem();
                  oi.setProduct(p);
                  oi.setQuantity(0);
                  oiList.add(oi);
            }
            orderform.setOiList(oiList);
            if(null != sessOrderform && null !=
sessOrderform.getOiList()){
                  boolean exist = false;
                  for(OrderItem sesOi : sessOrderform.getOiList()){
                        exist = false;
                        for(OrderItem oi1 : oiList){
                              if(!exist &&
oi1.getProduct().getName().equalsIgnoreCase(sesOi.getProduct().getName()))
{
                                    oi1.setQuantity(sesOi.getQuantity());
                                    exist = true;
                              }
                        }
```

```
				}
			}
			modelView = new ModelAndView("displayProductsForm");
			modelView.addObject("orderform", orderform);
			return modelView;
		}

		// Takes the list of products selected, their quantities and update
the Order object in session
		@RequestMapping(value = "/addProductsForm", method =
RequestMethod.POST)
		public ModelAndView
addNewProduct(@ModelAttribute("orderform")OrderForm orderform, HttpSession
session) {

			OrderForm sesOrderform =
(OrderForm)session.getAttribute("orderform");
			Iterator<OrderItem> itr = orderform.getOiList().iterator();
			OrderItem oi1;
			while(itr.hasNext()){
				oi1 = (OrderItem) itr.next();
				if(oi1.getQuantity()==0)
					itr.remove();
			}
			List<OrderItem> oiList = sesOrderform.getOiList();
			if(null == oiList) {
				sesOrderform.setOiList(orderform.getOiList());
			}else{
				for(OrderItem newOi: orderform.getOiList()){
					boolean exist = false;
					for(OrderItem sesOi: oiList){
						if(!exist &&
sesOi.getProduct().getName().equalsIgnoreCase(newOi.getProduct().getName()
)){
							sesOi.setQuantity(newOi.getQuantity());
							exist = true;
						}
					}
					if(!exist){
						oiList.add(newOi);
					}
				}
			}
			session.setAttribute("orderform", sesOrderform);
			return new ModelAndView("addProductSuccess");
		}

		//Display your cart details
		@RequestMapping(value = "/cart", method = RequestMethod.GET)
		public ModelAndView showCart(HttpSession session) {
			return new ModelAndView("cart");
		}
```

```
      //Your order checkout page
      @RequestMapping(value = "/checkout", method = RequestMethod.GET)
      public ModelAndView checkOut(HttpSession session) {
            OrderForm orderform =
(OrderForm)session.getAttribute("orderform");
            Order order = new Order();
            double subtotal = 0.0;
            order.setOiList(orderform.getOiList());
            order.setCustomer(orderform.getCustomer());
            for(OrderItem oi: orderform.getOiList()){
                  subtotal += oi.getProduct().getPrice()*oi.getQuantity();
            }
            order.setSubtotal(subtotal);
            billingService.computeTotalPrice(order);
            ModelAndView modelview = new ModelAndView("checkOut");
            modelview.addObject("order", order);
            return modelview;
      }

      //Payment successful page
      @RequestMapping(value = "/paymentSuccessful", method =
RequestMethod.GET)
      public ModelAndView paymentSuccessful(HttpSession session) {
            session.setAttribute("orderform",null);
            return new ModelAndView("paymentSuccessful");
      }
}
```

bookstore → dao → Impl

CategoryRowMapper.java

```
package edu.rk.bookstore.dao.impl;

import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

import edu.rk.bookstore.domain.Category;

public class CategoryRowMapper implements RowMapper<Category>{

      public Category mapRow(ResultSet rs, int row) throws SQLException {
            Category category = new Category();
            category.setCatId(rs.getInt(1));
            category.setCatName(rs.getString(2));
            return category;
      }

}
```

CustomerDaoImpl.java

```
package edu.rk.bookstore.dao.impl;

import java.util.List;

import javax.annotation.PostConstruct;
import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import edu.rk.bookstore.dao.CustomerDao;
import edu.rk.bookstore.domain.Customer;

@Repository("customerDao")
public class CustomerDaoImpl implements CustomerDao {
      @Autowired
      private DataSource dataSource;
      private JdbcTemplate jdbcTemplate;
      private CustomerRowMapper customerRowMapper;

      @PostConstruct
      public void setup(){
            jdbcTemplate = new JdbcTemplate(dataSource);
            customerRowMapper = new CustomerRowMapper();
      }


      public List<Customer> getCustomers(){
            String sql = "SELECT * FROM CUSTOMER";
            return jdbcTemplate.query(sql, customerRowMapper);
      }
}
```

CustomerRowMapper.java

```
package edu.rk.bookstore.dao.impl;

import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

import edu.rk.bookstore.domain.Customer;

public class CustomerRowMapper implements RowMapper<Customer> {
	@Override
	public Customer mapRow(ResultSet rs, int row) throws SQLException {
		Customer customer = new Customer();
		customer.setId(rs.getInt(1));
		customer.setName(rs.getString(2));
		customer.setState(rs.getString(3));
		return customer;
	}

}
```

InventoryDaoImpl.java

```
package edu.rk.bookstore.dao.impl;

import java.util.List;

import javax.annotation.PostConstruct;
import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
import
org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
import org.springframework.stereotype.Repository;

import edu.rk.bookstore.dao.InventoryDao;
import edu.rk.bookstore.domain.Inventory;
import edu.rk.bookstore.domain.OrderItem;
import edu.rk.bookstore.exceptions.ProductOutOfStockException;

@Repository("inventoryDao")
public class InventoryDaoImpl implements InventoryDao {
	@Autowired
	private DataSource dataSource;
	private JdbcTemplate jdbcTemplate;
	private InventoryRowMapper inventoryRowMapper;
	private NamedParameterJdbcTemplate namedTemplate;

	@PostConstruct
	public void setup(){
		jdbcTemplate = new JdbcTemplate(dataSource);
		inventoryRowMapper = new InventoryRowMapper();
		namedTemplate = new NamedParameterJdbcTemplate(dataSource);
	}

	public List<Inventory> getInventory() {
		String sql = "SELECT * FROM INVENTORY";
		return jdbcTemplate.query(sql, inventoryRowMapper);
	}

	public void updateInventory(List<OrderItem> oiList) throws
ProductOutOfStockException {
		String sql = "UPDATE INVENTORY SET availableCount =
:newAvailableCount, soldCount = :newSoldCount WHERE prodId = :prodId";
		int newAvailableCount, newSoldCount, prodId;
		MapSqlParameterSource params;
		int rowsAffected = 0;
		List<Inventory> invList = getInventory();
		for(OrderItem oi : oiList){
			prodId = oi.getProduct().getProdId();
			for(Inventory inv : invList){
				newAvailableCount=0;
				newSoldCount=0;
				if(prodId == inv.getProdId()){
					newSoldCount = inv.getSoldCount() +
oi.getQuantity();
					newAvailableCount = inv.getAvailableCount() -
oi.getQuantity();
					if(newAvailableCount < 0)
						throw new
ProductOutOfStockException("You cannot order more than
"+inv.getAvailableCount()+" "+oi.getProduct().getName());
					params = new MapSqlParameterSource("prodId",
prodId);
					params.addValue("newAvailableCount",
newAvailableCount);
					params.addValue("newSoldCount",
newSoldCount);
					rowsAffected += namedTemplate.update(sql,
params);
				}
			}
		}
		System.out.println(rowsAffected);
	}
}
```

InventoryRowMapper.java

```
package edu.rk.bookstore.dao.impl;

import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

import edu.rk.bookstore.domain.Inventory;

public class InventoryRowMapper implements RowMapper<Inventory>{
	public Inventory mapRow(ResultSet rs, int row) throws SQLException {
		Inventory inventory = new Inventory();
		inventory.setInvId(rs.getInt(1));
		inventory.setProdId(rs.getInt(2));
		inventory.setInitialCount(rs.getInt(3));
		inventory.setAvailableCount(rs.getInt(4));
		inventory.setSoldCount(rs.getInt(5));
		inventory.setSellingDetails(rs.getString(6));
		return inventory;
	}
}
```

OrderDaoImpl.java

```
package edu.rk.bookstore.dao.impl;

import java.util.List;

import javax.annotation.PostConstruct;
import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.jdbc.core.JdbcTemplate;
//import
org.springframework.jdbc.core.namedparam.BeanPropertySqlParameterSource;
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
//import
org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
//import org.springframework.jdbc.core.namedparam.SqlParameterSource;
import org.springframework.jdbc.core.simple.SimpleJdbcInsert;
import org.springframework.stereotype.Repository;

import edu.rk.bookstore.dao.OrderDao;
import edu.rk.bookstore.domain.Order;
import edu.rk.bookstore.domain.OrderItem;
import edu.rk.bookstore.exceptions.ProductRuntimeException;

@Repository("orderDao")
public class OrderDaoImpl implements OrderDao {
      @Autowired
      @Qualifier("dataSource")
      private DataSource dataSource;
      private JdbcTemplate jdbcTemplate;
//    private NamedParameterJdbcTemplate dbTemplate;
      private SimpleJdbcInsert orderJdbcInsert;
      private SimpleJdbcInsert orderItemJdbcInsert;

      @PostConstruct
      public void setup() {
            jdbcTemplate = new JdbcTemplate(dataSource);
//          dbTemplate = new NamedParameterJdbcTemplate(dataSource);
            orderJdbcInsert = new SimpleJdbcInsert(dataSource)
                              .withTableName("orders")
                              .usingGeneratedKeyColumns("orderId");
            orderItemJdbcInsert = new SimpleJdbcInsert(dataSource)
                 .withTableName("orderitem")
                 .usingGeneratedKeyColumns("orderItemId");
      }

      public void saveOrder(Order order) {
            MapSqlParameterSource params = new MapSqlParameterSource();
            params.addValue("custId", order.getCustomer().getId());
            params.addValue("subtotal", order.getSubtotal());
            params.addValue("tax", order.getTax());
            params.addValue("total", order.getTotal());
          Number newId = orderJdbcInsert.executeAndReturnKey(params);
          order.setCode(newId.intValue());

          List<OrderItem> oiList = order.getOiList();

          for(OrderItem oi : oiList){
            if(oi.getQuantity() <= 0){
                  throw new ProductRuntimeException("Product quantity
cannot be less than or equal to ZERO!!");
            }
            params = new MapSqlParameterSource();
                  params.addValue("orderId", order.getCode());
                  params.addValue("prodId", oi.getProduct().getProdId());
                  params.addValue("quantity", oi.getQuantity());
            newId = orderItemJdbcInsert.executeAndReturnKey(params);
          }
      }

      public int getOrdersCount(){
            String sql = "select count(*) FROM orders";
            return jdbcTemplate.queryForObject(sql, Integer.class);
```

```
	}

}
```

ProductDaoImpl.java

```
package edu.rk.bookstore.dao.impl;

import java.util.List;

import javax.annotation.PostConstruct;
import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
import
org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;
import org.springframework.jdbc.core.simple.SimpleJdbcInsert;
import org.springframework.stereotype.Repository;

import edu.rk.bookstore.dao.ProductDao;
import edu.rk.bookstore.domain.Category;
import edu.rk.bookstore.domain.Product;

@Repository("productDao")
public class ProductDaoImpl implements ProductDao {

	@Autowired
	private DataSource dataSource;
	private JdbcTemplate jdbcTemplate;
	private CategoryRowMapper categoryRowMapper;
	private ProductRowMapper productRowMapper;
	private NamedParameterJdbcTemplate dbTemplate;
	private SimpleJdbcInsert prodJdbcInsert;

	@PostConstruct
	public void setup(){
		jdbcTemplate = new JdbcTemplate(dataSource);
		dbTemplate = new NamedParameterJdbcTemplate(dataSource);
		categoryRowMapper = new CategoryRowMapper();
		productRowMapper = new ProductRowMapper();
		prodJdbcInsert = new SimpleJdbcInsert(dataSource)
		     .withTableName("product")
		     .usingGeneratedKeyColumns("prodId")
		     .usingColumns("catId", "name", "price");
	}
	public List<Category> getCategories(){
```

```
            String sql = "SELECT * FROM CATEGORY";
            return jdbcTemplate.query(sql, categoryRowMapper);
      }

      public List<Product> getProducts(int catId){
            String sql = "SELECT * FROM PRODUCT WHERE CATID = ?";
            return jdbcTemplate.query(sql, productRowMapper, catId);
      }
      public void updatePrice(int prodId, double newPrice){
            String sql = "UPDATE PRODUCT SET price=:price WHERE 
prodId=:prodId";
            MapSqlParameterSource params = new 
MapSqlParameterSource("price", newPrice);
            params.addValue("prodId", prodId);
            dbTemplate.update(sql, params);
      }
      public Product getProduct(int prodId){
            String sql = "SELECT * FROM PRODUCT WHERE PRODID = ?";
            return jdbcTemplate.queryForObject(sql, productRowMapper, 
prodId);
      }
      public void addNewProduct(Product product){
            MapSqlParameterSource params = new MapSqlParameterSource();
            params.addValue("catId", product.getCatId());
            params.addValue("name", product.getName());
            params.addValue("price", product.getPrice());
          Number newId = prodJdbcInsert.executeAndReturnKey(params);
          product.setProdId(newId.intValue());
      }
      public void deleteProduct(int prodId){
            String sql = "DELETE FROM PRODUCT WHERE PRODID = ?";
            jdbcTemplate.update(sql, prodId);
      }
}
```

ProductRowMapper.java

```
package edu.rk.bookstore.dao.impl;

import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

import edu.rk.bookstore.domain.Product;

public class ProductRowMapper implements RowMapper<Product> {
      public Product mapRow(ResultSet rs, int row) throws SQLException {
            Product product = new Product();
            product.setProdId(rs.getInt(1));
            product.setName(rs.getString(2));
            product.setPrice(rs.getDouble(3));
            product.setCatId(rs.getInt(4));
            return product;
      }
}
```

CustomerDao.java

```
package edu.rk.bookstore.dao;

import java.util.List;

import edu.rk.bookstore.domain.Customer;

public interface CustomerDao {
      public List<Customer> getCustomers();
}
```

InventoryDao.java

```
package edu.rk.bookstore.dao;

import java.util.List;

import edu.rk.bookstore.domain.Inventory;
import edu.rk.bookstore.domain.OrderItem;
import edu.rk.bookstore.exceptions.ProductOutOfStockException;

public interface InventoryDao {
      public List<Inventory> getInventory();
      public void updateInventory(List<OrderItem> oiList) throws
ProductOutOfStockException;

}
```

OrderDao.java

```
package edu.rk.bookstore.dao;
```

```
import edu.rk.bookstore.domain.Order;

public interface OrderDao {

      public void saveOrder(Order order);
      public int getOrdersCount();
}
```

ProductDao.java

```
package edu.rk.bookstore.dao;

import java.util.List;

import edu.rk.bookstore.domain.Category;
import edu.rk.bookstore.domain.Product;

public interface ProductDao {
      public List<Category> getCategories();
      public List<Product> getProducts(int catId);
      public void updatePrice(int prodId, double newPrice);
      public Product getProduct(int prodId);
      public void addNewProduct(Product product);
      public void deleteProduct(int prodId);
}
```

Domain package

Category.java

```
package edu.rk.bookstore.domain;

import java.util.List;

public class Category {

      private int catId;
      private String catName;
      private List<Product> prodList;
      public List<Product> getProdList() {
            return prodList;
```

```
	}
	public void setProdList(List<Product> prodList) {
		this.prodList = prodList;
	}
	public int getCatId() {
		return catId;
	}
	public void setCatId(int catId) {
		this.catId = catId;
	}
	public String getCatName() {
		return catName;
	}
	public void setCatName(String catName) {
		this.catName = catName;
	}
}
```

CategoryList.java

```
package edu.rk.bookstore.domain;

import java.io.Serializable;
import java.util.List;

import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class CategoryList implements Serializable{
	private static final long serialVersionUID = 1L;
	@XmlElement(name = "category")
	private List<Category> catList;
	public List<Category> getCatList() {
		return catList;
	}

	//set'CategoryList'() must match the 'class name'
	public void setCategoryList(List<Category> catList) {
		this.catList = catList;
	}
	public int numEntries(){
		if(catList == null) return 0;
		return catList.size();
	}
	public Category getCategory(int index){
		return catList.get(index);
	}
	public String toString(){
		String listStr;

		listStr = "CategoryList{";
		for (Category entry: catList) {
			listStr = listStr + "\n\t" + entry;
		}

		listStr = listStr + "\n}";
		return listStr;
	}
}
```

Customer.java

```
package edu.rk.bookstore.domain;

public class Customer {
	private String name;
	private String state;
	private int id;
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getState() {
		return state;
	}
	public void setState(String state) {
		this.state = state;
	}
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
}
```

Inventory.java

```
package edu.rk.bookstore.domain;
```

```
public class Inventory {
      private int invId;
      private int prodId;
      private int initialCount;
      private int availableCount;
      private int soldCount;
      private String sellingDetails;
      public int getInvId() {
            return invId;
      }
      public void setInvId(int invId) {
            this.invId = invId;
      }
      public int getProdId() {
            return prodId;
      }
      public void setProdId(int prodId) {
            this.prodId = prodId;
      }
      public int getInitialCount() {
            return initialCount;
      }
      public void setInitialCount(int initialCount) {
            this.initialCount = initialCount;
      }
      public int getAvailableCount() {
            return availableCount;
      }
      public void setAvailableCount(int availableCount) {
            this.availableCount = availableCount;
      }
      public int getSoldCount() {
            return soldCount;
      }
      public void setSoldCount(int soldCount) {
            this.soldCount = soldCount;
      }
      public String getSellingDetails() {
            return sellingDetails;
      }
      public void setSellingDetails(String sellingDetails) {
            this.sellingDetails = sellingDetails;
      }
}
```

Order.java

```
package edu.rk.bookstore.domain;

import java.util.Iterator;
import java.util.List;

public class Order {
      private int code;
      private Customer customer;
      private List<OrderItem> oiList;
      private double subtotal;
      private double tax;
      private double total;
      public Order(){  }
      public Order(int newOrderCode) {
            code = newOrderCode;
      }
      public int getCode() {
            return code;
      }
      public void setCode(int code) {
            this.code = code;
      }
      public Customer getCustomer() {
            return customer;
      }
      public void setCustomer(Customer customer) {
            this.customer = customer;
      }
      public List<OrderItem> getOiList() {
            return oiList;
      }
      public void setOiList(List<OrderItem> oiList) {
            this.oiList = oiList;
      }
      public double getSubtotal() {
            return subtotal;
      }
      public void setSubtotal(double subtotal) {
            this.subtotal = subtotal;
      }
      public double getTax() {
            return tax;
      }
      public void setTax(double tax) {
            this.tax = tax;
      }
      public double getTotal() {
            return total;
      }
      public void setTotal(double total) {
            this.total = total;
      }
      public void addItem(OrderItem item){
            boolean prodExists = false;
            for(OrderItem oi: this.oiList){
```

```
      if(oi.getProduct().getName().equalsIgnoreCase(item.getProduct().getN
ame())){

      oi.setQuantity(oi.getQuantity()+item.getQuantity());
                        prodExists = true;
                  }
            }
            if(!prodExists)
                  this.oiList.add(item);
      }
      public void removeProduct(Product prod){
            Iterator<OrderItem> oiIter = this.oiList.iterator();
            while(oiIter.hasNext()){

      if(oiIter.next().getProduct().getName().equalsIgnoreCase(prod.getNam
e())){
                        oiIter.remove();
                  }
            }
      }
}
```

OrderForm.java

```
package edu.rk.bookstore.domain;

import java.util.List;

public class OrderForm {
      private Customer customer;
      private List<OrderItem> oiList;

      public List<OrderItem> getOiList() {
            return oiList;
      }
      public void setOiList(List<OrderItem> oiList) {
            this.oiList = oiList;
      }
      public Customer getCustomer() {
            return customer;
      }

      public void setCustomer(Customer customer) {
            this.customer = customer;
      }

}
```

OrderItem.java

```
package edu.rk.bookstore.domain;

public class OrderItem {
	private int orderId;
	public int getOrderId() {
		return orderId;
	}
	public void setOrderId(int orderId) {
		this.orderId = orderId;
	}
	private Product product;
	private int quantity;
	public Product getProduct() {
		return product;
	}
	public void setProduct(Product product) {
		this.product = product;
	}
	public int getQuantity() {
		return quantity;
	}
	public void setQuantity(int quantity) {
		this.quantity = quantity;
	}
}
```

Product.java

```
package edu.rk.bookstore.domain;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "product")
public class Product {
	private int prodId;
	private String name;
	private double price;
	private int catId;
	public int getProdId() {
		return prodId;
	}
	public void setProdId(int prodId) {
		this.prodId = prodId;
	}
	public int getCatId() {
		return catId;
	}
	public void setCatId(int catId) {
		this.catId = catId;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public double getPrice() {
		return price;
	}
	public void setPrice(double price) {
		this.price = price;
	}
}
```

ProductList.java

```
package edu.rk.bookstore.domain;

import java.io.Serializable;
import java.util.List;

import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class ProductList implements Serializable{
	private static final long serialVersionUID = 1L;

	@XmlElement(name="product")
	private List<Product> prodList;

	public List<Product> getProdList() {
		return prodList;
	}
	public void setProductList(List<Product> prodList) {
		this.prodList = prodList;
	}
	public int numEntries(){
		if(prodList == null) return 0;
		return prodList.size();
	}
	public Product getProduct(int index){
```

```
            return prodList.get(index);
      }
      public String toString(){
            String listStr;
            listStr = "ProductList{";
            for (Product entry: prodList) {
                  listStr = listStr + "\n\t" + entry;
            }
            listStr = listStr + "\n}";
            return listStr;
      }
}
```

Bookstore→ exceptions

ProductOutOfStockException.java

```
package edu.rk.bookstore.exceptions;

public class ProductOutOfStockException extends Exception{
      private static final long serialVersionUID = 1L;
      public ProductOutOfStockException(String msg){
            super(msg);
      }
}
```

ProductRuntimeException.java

```
package edu.rk.bookstore.exceptions;

public class ProductRuntimeException extends RuntimeException{
      private static final long serialVersionUID = 1L;
      public ProductRuntimeException(String msg){
            super(msg);
      }
}
```

Bookstore → resthandlers

BookStoreRestHandler.java

```
package edu.rk.bookstore.resthandlers;

import java.util.ArrayList;
import java.util.List;

import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.WebApplicationException;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.ResponseBuilder;
import javax.ws.rs.core.Response.Status;

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

import edu.rk.bookstore.domain.Category;
import edu.rk.bookstore.domain.CategoryList;
import edu.rk.bookstore.domain.Product;
import edu.rk.bookstore.domain.ProductList;
import edu.rk.bookstore.services.ProductService;

@Path("/bookstorerestapp")
public class BookStoreRestHandler {

      @Autowired
      private ProductService productService;

      //http://localhost:8080/bookstore/webservices/bookstorerestapp/categ
ory
      // Get all Categories - GET
      @GET
      @Path("/category")
      @Produces("application/xml")
      public CategoryList getCategories(){
            List<Category> catList = productService.getCategories();
            CategoryList cList = new CategoryList();
            cList.setCategoryList(catList);
            return cList;
      }

      //http://localhost:8080/bookstore/webservices/bookstorerestapp/categ
ory/1/product
      // Get Products for a Category - GET
      @GET
      @Path("/category/{catId}/product")
      @Produces("application/xml")
```

```
	public ProductList getAllProducts(@PathParam(value="catId") int 
catId ){
		List<Product> prodList = productService.getProducts(catId);
		ProductList pList = new ProductList();
		pList.setProductList(prodList);
		return pList;
	}


	//http://localhost:8080/bookstore/webservices/bookstorerestapp/produ
ct/1
	// Get details of a Product - GET
	@GET
	@Path("/product/{prodId}")
	@Produces("application/xml")
	public ProductList getProduct(@PathParam(value="prodId") int prodId 
){
		Product prod = productService.getProduct(prodId);
		ProductList pList = new ProductList();
		List<Product> prodList = new ArrayList<Product>();
		prodList.add(prod);
		pList.setProductList(prodList);
		return pList;
	}

	//http://localhost:8080/bookstore/webservices/bookstorerestapp/produ
ct/1/price/100
	// Update Price of a Product - PUT
	@PUT
	@Path("/product/{prodId}/price/{newPrice}")
	public void updateProductPrice(@PathParam(value="prodId") int 
prodId, @PathParam(value="newPrice") double newPrice ){
		productService.updatePrice(prodId, newPrice);
	}


	// Create a new Product - POST
	// 
http://localhost:8080/bookstore/webservices/bookstorerestapp/product
	//<product><catId>1</catId><name>newProduct</name><price>10</price><
/product>
	@POST
	@Path("/product")
	@Produces("application/xml")
	@Consumes("application/xml")
	public Response addNewProduct(Product newProduct){
		ResponseBuilder respBuilder;
		try {
			productService.addNewProduct(newProduct);
		} catch (Exception ex) {
			throw new WebApplicationException(ex.getMessage(), 
Status.INTERNAL_SERVER_ERROR);
```

```
            }
            respBuilder = Response.status(Status.CREATED);
            respBuilder.entity(newProduct);
            return respBuilder.build();
      }

      // Delete a Product - DELETE
      //
http://localhost:8080/bookstore/webservices/bookstorerestapp/product/26
      @DELETE
      @Path("/product/{prodId}")
      public void deleteProduct(@PathParam(value="prodId") int prodId){
            productService.deleteProduct(prodId);
      }


}
```

RestApplicationConfig.java

```
package edu.rk.bookstore.resthandlers;

import java.util.HashSet;
import java.util.Set;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

import org.glassfish.jersey.jackson.JacksonFeature;

@ApplicationPath("/")
public class RestApplicationConfig extends Application{
      private Set<Class<?>> restClassSet = new HashSet<Class<?>>();

      public RestApplicationConfig(){
            restClassSet.add(JacksonFeature.class);
            restClassSet.add(BookStoreRestHandler.class);
      }

      public Set<Class<?>> getClasses(){
            return restClassSet;
      }
}
```

Bookstore→services →impl

BillingServiceImpl.java

```
package edu.rk.bookstore.services.impl;

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

import edu.rk.bookstore.domain.Order;
import edu.rk.bookstore.exceptions.ProductOutOfStockException;
import edu.rk.bookstore.services.BillingService;
import edu.rk.bookstore.services.InventoryService;
import edu.rk.bookstore.services.OrderService;
import edu.rk.bookstore.services.TaxService;

@Service("billingService")
public class BillingServiceImpl implements BillingService {

      @Autowired
      private TaxService taxService;
      @Autowired
      private OrderService orderService;
      @Autowired
      private InventoryService inventoryService;
      public void setOrderService(OrderService orderService) {
            this.orderService = orderService;
      }
      public void setInventoryService(InventoryService inventoryService) {
            this.inventoryService = inventoryService;
      }
      public void setTaxService(TaxService taxService) {
            this.taxService = taxService;
      }
      public void computeTotalPrice(Order order){
            double totalTax = order.getSubtotal() *
taxService.computeTax(order)/100;
            order.setTax(totalTax);
            order.setTotal(order.getSubtotal() + totalTax);
      }
      @Transactional(readOnly=false,
rollbackForClassName="ProductOutOfStockException")
      public void processCustomerPurchase(Order order) throws
ProductOutOfStockException{
            orderService.saveOrder(order);
            inventoryService.updateInventory(order.getOiList());
      }
}
```

CustomerServiceImpl.java

```
package edu.rk.bookstore.services.impl;

import java.util.List;

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

import edu.rk.bookstore.dao.CustomerDao;
import edu.rk.bookstore.domain.Customer;
import edu.rk.bookstore.services.CustomerService;

@Service("customerService")
public class CustomerServiceImpl implements CustomerService {
      @Autowired
      private CustomerDao customerDao;
      public List<Customer> getCustomers(){
            return customerDao.getCustomers();
      }

}
```

InventoryServiceImpl.java

```
package edu.rk.bookstore.services.impl;

import java.util.List;

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

import edu.rk.bookstore.dao.InventoryDao;
import edu.rk.bookstore.domain.Inventory;
import edu.rk.bookstore.domain.OrderItem;
import edu.rk.bookstore.exceptions.ProductOutOfStockException;
import edu.rk.bookstore.services.InventoryService;

@Service("inventoryService")
public class InventoryServiceImpl implements InventoryService {
      @Autowired
      InventoryDao inventoryDao;

      public void setInventoryDao(InventoryDao inventoryDao) {
            this.inventoryDao = inventoryDao;
      }
```

```
	public List<Inventory> getInventory() {
		return inventoryDao.getInventory();
	}

	public void updateInventory(List<OrderItem> oiList) throws 
ProductOutOfStockException {
		inventoryDao.updateInventory(oiList);
	}
}
```

OrderServiceImpl.java

```
package edu.rk.bookstore.services.impl;

import java.util.List;

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

import edu.rk.bookstore.dao.OrderDao;
import edu.rk.bookstore.domain.Order;
import edu.rk.bookstore.domain.OrderItem;
import edu.rk.bookstore.domain.Product;
import edu.rk.bookstore.services.OrderService;

@Service("orderService")
public class OrderServiceImpl implements OrderService {

	@Autowired
	private OrderDao orderDao;
	public void setOrderDao(OrderDao orderDao) {
		this.orderDao = orderDao;
	}
	public void saveOrder(Order order){
		orderDao.saveOrder(order);
	}
	public int getOrdersCount(){
		return orderDao.getOrdersCount();
	}
	public void addProduct(Order order, Product product, int quantity){
		List<OrderItem> oiList = order.getOiList();
		boolean exist = false;
		for(OrderItem oi: oiList){
			if(!exist && 
oi.getProduct().getName().equalsIgnoreCase(product.getName())){
				oi.setQuantity(quantity);
				exist = true;
			}
```

```
            }
            if(!exist){
                  OrderItem oi = new OrderItem();
                  oi.setOrderId(order.getCode());
                  oi.setProduct(product);
                  oi.setQuantity(quantity);
                  oiList.add(oi);
            }
      }
}
```

ProductServiceImpl.java

```
package edu.rk.bookstore.services.impl;

import java.util.List;

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

import edu.rk.bookstore.dao.ProductDao;
import edu.rk.bookstore.domain.Category;
import edu.rk.bookstore.domain.Product;
import edu.rk.bookstore.services.ProductService;

@Service("productService")
public class ProductServiceImpl implements ProductService {
      @Autowired
      private ProductDao productDao;
      public void setProductDao(ProductDao productDao) {
            this.productDao = productDao;
      }
      public List<Category> getCategories(){
            return productDao.getCategories();
      }
      public List<Product> getProducts(int catId){
            return productDao.getProducts(catId);
      }
      public void updatePrice(int prodId, double newPrice){
            productDao.updatePrice(prodId, newPrice);
      }
      public Product getProduct(int prodId){
            return productDao.getProduct(prodId);
      }
      public void addNewProduct(Product product){
            productDao.addNewProduct(product);
      }
      public void deleteProduct(int prodId){
            productDao.deleteProduct(prodId);
      }
}
```

TaxServiceImpl.java

```
package edu.rk.bookstore.services.impl;

import java.util.HashMap;
import java.util.Map;
import java.util.StringTokenizer;

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

import edu.rk.bookstore.domain.Order;
import edu.rk.bookstore.services.TaxService;

@Service("taxService")
public class TaxServiceImpl implements TaxService {

      Map<String,Integer> stateSalesTaxPercentageMap;

      @Autowired
      public TaxServiceImpl(@Value("${stateSalesTaxPercentage.map}")
String stateSalesTaxPercentageMap){
            this.stateSalesTaxPercentageMap =
getMap(stateSalesTaxPercentageMap);
      }

      public double computeTax(Order order){
            return
stateSalesTaxPercentageMap.get(order.getCustomer().getState());
      }

      public Map<String,Integer> getMap(String str){
            Map<String,Integer> map = new HashMap<String,Integer>();
            String key = null;
            String value = null;
            StringTokenizer st = new StringTokenizer(str, ",");
            while (st.hasMoreElements()) {
                  if(null == key)
                        key = st.nextToken();
                  else if(null != key)
                        value = st.nextToken();
                  if(null != key && null != value){
                        map.put(key, Integer.parseInt(value));
                        key = null;
                        value = null;
                  }
            }
            return map;
      }

}
```

Bookstore → services

BillingService.java

```
package edu.rk.bookstore.services;

import edu.rk.bookstore.domain.Order;
import edu.rk.bookstore.exceptions.ProductOutOfStockException;

public interface BillingService {
      public void computeTotalPrice(Order order);
      public void processCustomerPurchase(Order order) throws
ProductOutOfStockException;
}
```

CustomerService.java

```
package edu.rk.bookstore.services;

import java.util.List;

import edu.rk.bookstore.domain.Customer;

public interface CustomerService {
      public List<Customer> getCustomers();

}
```

InventoryService.java

```
package edu.rk.bookstore.services;
```

```
import java.util.List;

import edu.rk.bookstore.domain.Inventory;
import edu.rk.bookstore.domain.OrderItem;
import edu.rk.bookstore.exceptions.ProductOutOfStockException;

public interface InventoryService {
	public List<Inventory> getInventory();
	public void updateInventory(List<OrderItem> oiList) throws
ProductOutOfStockException;
}
```

OrderService.java

```
package edu.rk.bookstore.services;

import edu.rk.bookstore.domain.Order;
import edu.rk.bookstore.domain.Product;

public interface OrderService {
	public void saveOrder(Order order);
	public int getOrdersCount();
	public void addProduct(Order order, Product product, int quantity);
}
```

ProductService.java

```
package edu.rk.bookstore.services;

import java.util.List;

import edu.rk.bookstore.domain.Category;
import edu.rk.bookstore.domain.Product;

public interface ProductService {
	public List<Category> getCategories();
	public List<Product> getProducts(int catId);
	public void updatePrice(int prodId, double newPrice);
	public Product getProduct(int prodId);
	public void addNewProduct(Product product);
	public void deleteProduct(int prodId);
}
```

TaxService.java

```
package edu.rk.bookstore.services;

import edu.rk.bookstore.domain.Order;

public interface TaxService {
      public double computeTax(Order order);
}
```

Main →webapp → resources → css

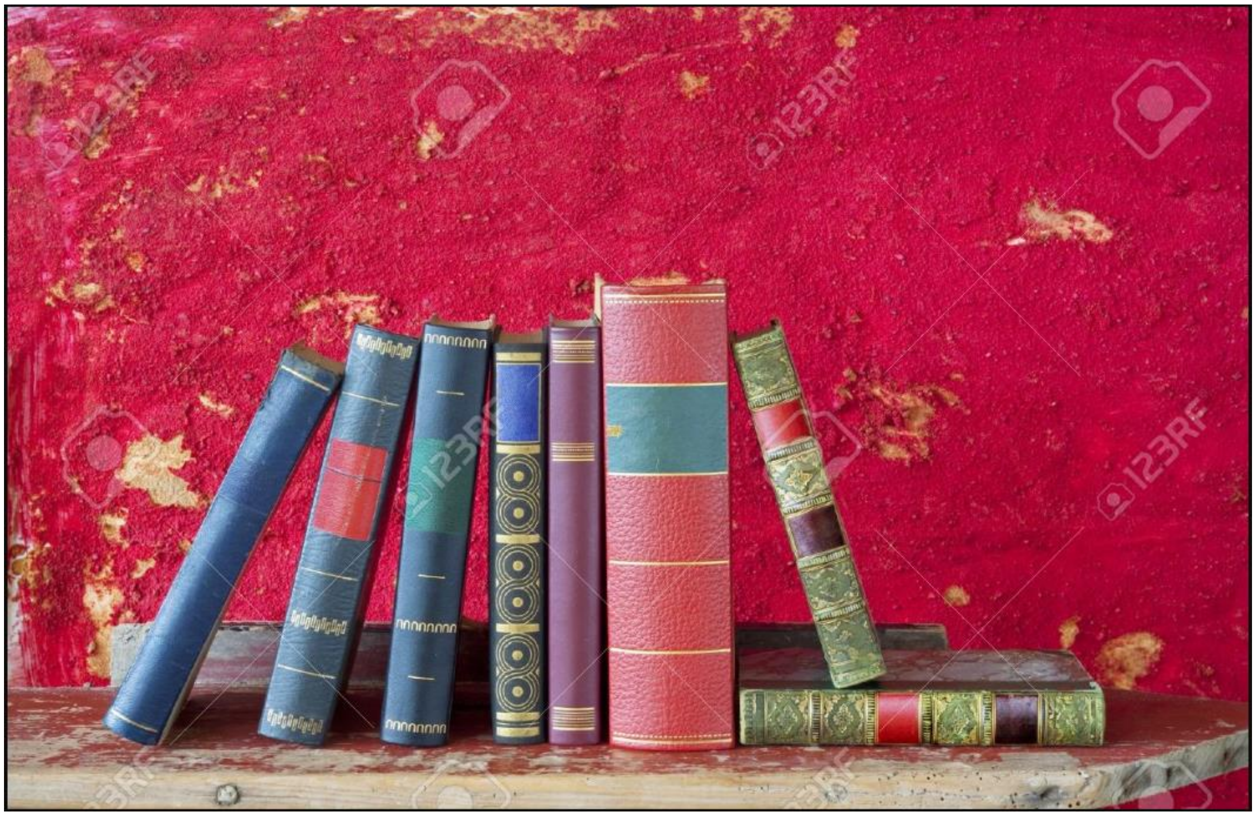
Styles.css

```
@CHARSET "UTF-8";
*{
    margin: 0px;
    padding: 0px;
}
.container{
    width:70%;
    margin :0px auto;
    font-family: cursive;
    font-weight: bold;
}
table{
      color:maroon;
      padding-left: 17%;
      font-weight: bold;
}
section{
text-align: center;
padding-top: 5.5%;
background-image: url('resources/images/BookBoo.jpg');
width:100%;
border-radius : 25px;
height:100%;
}
img{
      border-radius : 25px;
}
ul{
      align:"left"
}
```

Webapp → resources→ images

b1.jpg

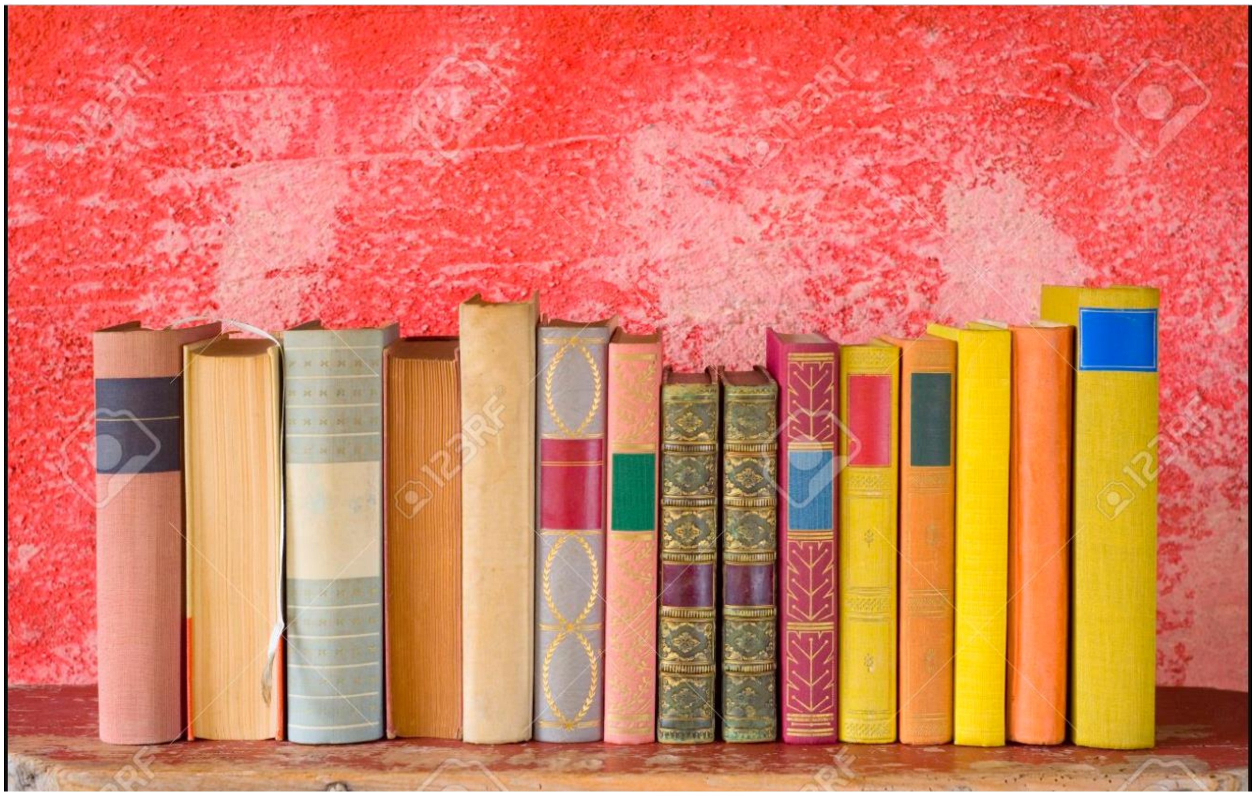
b2.jpg


123RF

b3.jpg

b4.jpg

b5.jpg



BookBoo.jpg

Main-> webapp → WEB-INF

Spring→ appServlet

servlet-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:beans="http://www.springframework.org/schema/beans"
	xmlns:context="http://www.springframework.org/schema/context"
	xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
		http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
		http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

	<!-- DispatcherServlet Context: defines this servlet's request-processing
infrastructure -->

	<!-- Enables the Spring MVC @Controller programming model -->
	<annotation-driven />

	<!-- Handles HTTP GET requests for /resources/** by efficiently serving up
static resources in the ${webappRoot}/resources directory -->
	<resources mapping="/resources/**" location="/resources/" />

	<!-- Resolves views selected for rendering by @Controllers to .jsp resources
in the /WEB-INF/views directory -->
	<beans:bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
		<beans:property name="prefix" value="/WEB-INF/views/" />
		<beans:property name="suffix" value=".jsp" />
	</beans:bean>
	
	<context:component-scan base-package="edu.rk.bookstore.controllers" />
	
	
	
</beans:beans>
```

Webapp → WEB-INF → spring

Root-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:aop="http://www.springframework.org/schema/aop"
	xmlns:context="http://www.springframework.org/schema/context"
	xmlns:lang="http://www.springframework.org/schema/lang"
	xmlns:util="http://www.springframework.org/schema/util"
	xmlns:tx="http://www.springframework.org/schema/tx"
	xsi:schemaLocation="http://www.springframework.org/schema/beans 
http://www.springframework.org/schema/beans/spring-beans.xsd
		http://www.springframework.org/schema/aop 
http://www.springframework.org/schema/aop/spring-aop.xsd
		http://www.springframework.org/schema/context 
http://www.springframework.org/schema/context/spring-context.xsd
		http://www.springframework.org/schema/lang 
http://www.springframework.org/schema/lang/spring-lang.xsd
		http://www.springframework.org/schema/tx 
http://www.springframework.org/schema/tx/spring-tx.xsd
		http://www.springframework.org/schema/util 
http://www.springframework.org/schema/util/spring-util.xsd">

	<!-- <context:annotation-config/> -->
	<context:component-scan base-package="edu.rk.bookstore.services, 
edu.rk.bookstore.dao"/>
	<context:property-placeholder location="classpath:jdbc.properties, 
classpath:order.properties"/>
	<!-- <context:property-placeholder location="order.properties"/>
	<aop:aspectj-autoproxy proxy-target-class="true"/>
	<bean id="inventoryAspect" class="edu.npu.inv.aspects.InventoryAspects"/> -->
```

```
        <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
              <property name="driverClassName" value="${jdbc.driver_class}" />
              <property name="url" value="${jdbc.url}" />
              <property name="username" value="${jdbc.username}" />
         <property name="password" value="${jdbc.password}" />
              <property name="initialSize" value="5" />
        </bean>

        <bean id="txManager"

        class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
              <property name="dataSource" ref="dataSource" />
        </bean>
        <tx:annotation-driven transaction-manager="txManager" />

        <bean id="messageSource"
class="org.springframework.context.support.ResourceBundleMessageSource">
      <property name="basename" value="messages"/>
    </bean>
</beans>
```

webapp → WEB-INF → views

addProductSuccess.jsp

```
<%@ include file="./include.jsp"%>
<html>
<head>
<title>Order Update</title>
<link rel="stylesheet" type="text/css" href="${context}/resources/css/Styles.css" />

</head>
<body>
<div class ="container">
<header>
<img src='resources/images/b1.jpg' width="100%" height="40%"/>
</header>
<section style="background-image: url('resources/images/BookBoo.jpg');">

<h3>Your order is successfully updated</h3>
<br>

<a href="${context}/categoryList">Back to Shopping</a></br>

<a href="${context}/cart">My Cart</a></br>

<a href="${context}/checkout">Checkout</a></br>
```

```
</section>
</div>
</body>
</html>
```

cart.jsp

```
<%@ include file="./include.jsp"%>

<html><head>
<link rel="stylesheet" type="text/css" href="${context}/resources/css/Styles.css" />

<title>Cart</title>
</head>
<body>
<div class ="container">
<header>
<img src='resources/images/b1.jpg' width="100%" height="40%"/>
</header>
<section style="background-image: url('resources/images/BookBoo.jpg');">

<h3>You have below items in the cart</h3>
<br>

<table cellspacing="10">
<tr><td align="center">Book Name<br></td><td align="center">Book Price<br></td><td
align="center">Quantity<br></td></tr>
<c:forEach var = "oi" items = "${sessionScope.orderform.oiList}" varStatus="status">
       <tr><td>
        <c:out value = "${oi.product.name}" /> </td><td>
        <c:out value = "${oi.product.price}" /> </td><td>
        <c:out value = "${oi.quantity}"/>  </td></tr>
</c:forEach>
</table>
<br>

<a href="${context}/categoryList"><fmt:message key="back" /></a></br>
<a href="${context}/checkout"><fmt:message key="checkout" /></a>

</section>
</div>

</body>
</html>
```

categoryList.jsp

```
<%@ include file="./include.jsp"%>

<html><head>
<title>Category Page</title>
<link rel="stylesheet" type="text/css" href="${context}/resources/css/Styles.css" />

</head><body>
<div class ="container">
<header>
<img src='resources/images/b1.jpg' width="100%" height="40%"/>
</header>
<section style="background-image: url('resources/images/BookBoo.jpg');">
<h3>Please Select Books Category</h3></br>
<c:forEach var="category" items="${catList}" >
<a
href="${context}/displayProductsForm?catId=${category.catId}"><l1>${category.catName}
</a></br>
</c:forEach>
</section>
</div>
</body></html>
```

checkOut.jsp

```
<%@ include file="./include.jsp"%>


<html><head>
<title><fmt:message key="checkout.title" /></title>
<link rel="stylesheet" type="text/css" href="${context}/resources/css/Styles.css" />

</head>
<body>
<div class ="container">
<header>
<img src='resources/images/b1.jpg' width="100%" height="40%"/>
</header>
<section style="background-image: url('resources/images/BookBoo.jpg');">

<h3>Order details</h3>
<br>

<form:form action="./paymentSuccessful" method="GET" commandName="order">
Dear <c:out value = "${order.customer.name}" />, below are your shopping details.
<br><br>
```

```
<table cellspacing="10">
<tr><td align="center"><b>Item</b><br></td><td
align="center"><b>Price</b><br></td><td align="center"><b>Quantity</b><br></td></tr>
<c:forEach var = "oi" items = "${order.oiList}" varStatus="status">
             <tr><td>
        <c:out value = "${oi.product.name}" /> </td><td>
        <c:out value = "${oi.product.price}" /> </td><td>
        <c:out value ="${oi.quantity}"/> </td></tr>
</c:forEach>
<tr><td> <br>
<b>Sub Total :</b> <c:out value = "${order.subtotal}" /><br></td></tr><tr><td>
<b>Tax : </b><c:out value = "${order.tax}" /><br></td></tr><tr><td>
<b>Total :</b> <c:out value = "${order.total}" /><br></td></tr>
</table>

<a href="${context}/categoryList"><fmt:message key="back" /></a></br>
<input type="submit" value="Pay now">

</form:form>

</section>
</div>
</body>
</html>
```

customerHome.jsp

```
<%@ include file="./include.jsp"%>
<html>
<head>
      <title>Home Page</title>
      <link rel="stylesheet" type="text/css"
href="${context}/resources/css/Styles.css" />
</head>
<body>
<div class ="container">
<header>
<img src='resources/images/b1.jpg' width="100%" height="40%"/>
</header>
<section style="background-image: url('resources/images/BookBoo.jpg')">
<form:form action="./categoryList" method="GET" commandName="customer">

<div align="center">

<h3>Welcome to online Book Store</h3>
      <form:select path="id">
              <form:option value="0" label="Please select your username"/>
              <form:options items="${custList}" itemValue="id" itemLabel="name"/>
      </form:select>
      </br>
      <input type="submit" value="Sign In" >

        </div>
  </form:form>
  </section>
 </div>
</body>
</html>
```

displayProductsForm.jsp

```
<%@ include file="./include.jsp"%>
<html>
<head>
<link rel="stylesheet" type="text/css" href="${context}/resources/css/Styles.css" />

<title>Product's Page</title>
</head>
<body>
<div class ="container">
<header>
<img src='resources/images/b1.jpg' width="100%" height="40%"/>
</header>
<section style="background-image: url('resources/images/BookBoo.jpg');">
<form:form action="./addProductsForm" method="POST" commandName="orderform">

<h3>Please select the books and quantity</h3></br>
<table>
<tr><td align="center">Book Name<br></td><td align="center">Book Price<br></td><td
align="center">Quantity<br></td></tr>
      <c:forEach var = "oi" items = "${orderform.oiList}" varStatus="status">
      <tr><td>
        <c:out value = "${oi.product.name}" /> </td>
        <td>
        <c:out value = "${oi.product.price}" /> </td>
        <td>
        <form:input path="oiList[${status.index}].quantity" type="text" />
<br></td></tr>
        <form:hidden path="oiList[${status.index}].product.name"/>
        <form:hidden path="oiList[${status.index}].product.price"/>
    </c:forEach>
    <tr><td align="center"><br>
<input type="submit" value="Add to cart">
</td></tr></table>
</form:form>
</section>
</div>
```

```
</body>
</html>
```

include.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<c:set var="context" scope="request" value="<%= request.getContextPath()%>" />
```

paymentSuccess.jsp

```
<%@ include file="./include.jsp"%>

<html><head>
<link rel="stylesheet" type="text/css" href="${context}/resources/css/Styles.css" />

<title><fmt:message key="success.title" /></title>
</head><body>
<div class ="container">
<header>
<img src='resources/images/b1.jpg' width="100%" height="40%"/>
</header>
<section style="background-image: url('resources/images/BookBoo.jpg');">
<h2>
<fmt:message key="paySuccess" />
</h2>
</section>
</div>
</body>
</html>
```

webapp → WEB-INF

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id= "WebApp_9" version="2.5" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/root-context.xml</param-value>
  </context-param>
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
  </listener>
  <servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
  <servlet>
    <servlet-name>JerseyWebApplication</servlet-name>
    <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
    <init-param>
      <param-name>javax.ws.rs.Application</param-name>
      <param-value>edu.rk.bookstore.resthandlers.RestApplicationConfig</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>JerseyWebApplication</servlet-name>
    <url-pattern>/webservices/*</url-pattern>
  </servlet-mapping>
</web-app>
```

InventoryDAOTest.java

```
package edu.rk.bookstore.test.dao;

import static org.junit.Assert.*;

import java.util.ArrayList;
import java.util.List;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import edu.rk.bookstore.dao.InventoryDao;
import edu.rk.bookstore.domain.Inventory;
import edu.rk.bookstore.domain.OrderItem;
import edu.rk.bookstore.domain.Product;

@ContextConfiguration("classpath:bookstore-test-context.xml")
@RunWith(SpringJUnit4ClassRunner.class)
public class InventoryDAOTest {
      @Autowired
      private InventoryDao inventoryDao;

      @Test
      public void testOrderCount() {
            List<Inventory> invList = inventoryDao.getInventory();
            System.out.println(invList.size());
      }

      /* Verifies if the Inventory table is updated properly when a
purchase is done */
      @Test
      public void testUpdateInventory(){
            OrderItem orderItem = new OrderItem();
            List<OrderItem> oiList = new ArrayList<OrderItem>();
            Product product = new Product();
            product.setCatId(1);
            product.setPrice(19.25);
            product.setProdId(1);
            orderItem.setProduct(product);
            orderItem.setQuantity(10);
            oiList.add(orderItem);
            int initialAvailable = 0;
            int initialSold = 0;
            List<Inventory> invList = inventoryDao.getInventory();
            for(Inventory inv : invList){
                  if(inv.getProdId() == product.getProdId()){
                        initialAvailable = inv.getAvailableCount();
                        initialSold = inv.getSoldCount();
                  }
            }
            try{
                  inventoryDao.updateInventory(oiList);
            }catch(Exception e){
                  System.out.println("Out of Stock");
            }
            int finalAvailable = 0;
            int finalSold = 0;
            invList = inventoryDao.getInventory();
            for(Inventory inv : invList){
                  if(inv.getProdId() == product.getProdId()){
                        finalAvailable = inv.getAvailableCount();
                        finalSold = inv.getSoldCount();
                  }
            }
            assertEquals(initialAvailable-10, finalAvailable);
            assertEquals(initialSold+10, finalSold);
      }

}
```

OrderDAOTest.java

```
package edu.rk.bookstore.test.dao;

import static org.junit.Assert.*;

import java.util.ArrayList;
import java.util.List;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import edu.rk.bookstore.dao.OrderDao;
import edu.rk.bookstore.domain.Customer;
import edu.rk.bookstore.domain.Order;
import edu.rk.bookstore.domain.OrderItem;
import edu.rk.bookstore.domain.Product;

@ContextConfiguration("classpath:bookstore-test-context.xml")
@RunWith(SpringJUnit4ClassRunner.class)
public class OrderDAOTest {
      @Autowired
      private OrderDao orderDao;

      @Test
      public void testOrderCount() {
            int cnt = orderDao.getOrdersCount();
            System.out.println(cnt);
      }

      /* verify that an order and corresponding orderItems are saved
properly to database */
      @Test
      public void testSaveOrder(){
            Order order;
            Product product;
            OrderItem orderItem;
            List<OrderItem> oiList;
            product = new Product();
            Customer cust;
            orderItem = new OrderItem();
            oiList = new ArrayList<OrderItem>();
            product.setCatId(1);
            product.setPrice(19.25);
            product.setProdId(1);
            orderItem.setProduct(product);
            orderItem.setQuantity(10);
            oiList.add(orderItem);
            order = new Order();
            order.setOiList(oiList);
            cust = new Customer();
            cust.setId(1);
            cust.setName("JOHN");
            cust.setState("CA");
            order.setCustomer(cust);
            order.setSubtotal(202.54);
            order.setTax(22.5);
            order.setTotal(1200.98);

            int oldCnt = orderDao.getOrdersCount();
            orderDao.saveOrder(order);
            int newCnt = orderDao.getOrdersCount();
            assertEquals(oldCnt+1 , newCnt);
      }

}
```

BillingServiceTest.java

```
package edu.rk.bookstore.test.services;

import static org.junit.Assert.*;

import java.util.ArrayList;
import java.util.List;

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import edu.rk.bookstore.domain.Customer;
import edu.rk.bookstore.domain.Order;
import edu.rk.bookstore.domain.OrderItem;
import edu.rk.bookstore.domain.Product;
import edu.rk.bookstore.exceptions.ProductOutOfStockException;
import edu.rk.bookstore.exceptions.ProductRuntimeException;
import edu.rk.bookstore.services.BillingService;

@ContextConfiguration("classpath:bookstore-test-context.xml")
@RunWith(SpringJUnit4ClassRunner.class)
public class BillingServiceTest {
      @Autowired
      private BillingService billingService;
      private Order order;
      private OrderItem orderItem;
      private Product product;
      private List<OrderItem> oiList;
      private Customer cust;

      @Before
      public void init() {
            product = new Product();
            product.setCatId(1);
            product.setPrice(19.25);
            product.setProdId(1);
            orderItem = new OrderItem();
            orderItem.setProduct(product);
            orderItem.setQuantity(0);
            oiList = new ArrayList<OrderItem>();
            oiList.add(orderItem);
            product.setCatId(1);
            product.setPrice(99.50);
            product.setProdId(2);
            orderItem = new OrderItem();
            orderItem.setProduct(product);
            orderItem.setQuantity(10);
            oiList.add(orderItem);
            order = new Order();
            order.setOiList(oiList);
            cust = new Customer();
            cust.setId(1);
            cust.setName("JOHN");
            cust.setState("CA");
            order.setCustomer(cust);
            order.setSubtotal(202.54);
            order.setTax(22.5);
            order.setTotal(1200.98);
      }

      /* Verifies that ZERO product quantity in an order is not acceptable
*/
      @Test(expected=ProductRuntimeException.class)
      public void testZeroQuantityProduct() throws
ProductRuntimeException, ProductOutOfStockException {
            billingService.processCustomerPurchase(order);
      }

      /* Verifies that a product cannot be ordered more than its
availability  */
      @Test(expected=ProductOutOfStockException.class)
      public void testOrderMoreThanProductAvailability() throws
ProductOutOfStockException {
            order.getOiList().get(0).setQuantity(70);
            billingService.processCustomerPurchase(order);
      }

}
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <groupId>edu.npu.cs548</groupId>
      <artifactId>courseapp</artifactId>
      <name>courseapp</name>
      <packaging>war</packaging>
      <version>1.0.0-SNAPSHOT</version>

      <properties>
            <org.springframework.version>4.1.0.RELEASE</org.springframework.version>
            <org.slf4j.version>1.7.7</org.slf4j.version>
            <org.hibernate.version>4.3.6.Final</org.hibernate.version>
            <junit.version>4.11</junit.version>
      </properties>
      <dependencies>

            <!-- Spring -->
            <dependency>
                  <groupId>org.springframework</groupId>
                  <artifactId>spring-beans</artifactId>
                  <version>${org.springframework.version}</version>
            </dependency>
            <dependency>
                  <groupId>org.springframework</groupId>
                  <artifactId>spring-aop</artifactId>
                  <version>${org.springframework.version}</version>
            </dependency>
```

```
<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-context-support</artifactId>
	<version>${org.springframework.version}</version>
</dependency>
<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-expression</artifactId>
	<version>${org.springframework.version}</version>
</dependency>


<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-tx</artifactId>
	<version>${org.springframework.version}</version>
</dependency>
<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-jdbc</artifactId>
	<version>${org.springframework.version}</version>
</dependency>
<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-oxm</artifactId>
	<version>${org.springframework.version}</version>
</dependency>
<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-orm</artifactId>
	<version>${org.springframework.version}</version>
</dependency>
<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-context</artifactId>
	<version>${org.springframework.version}</version>
</dependency>
<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-web</artifactId>
	<version>${org.springframework.version}</version>
</dependency>
<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-webmvc</artifactId>
	<version>${org.springframework.version}</version>
</dependency>
<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-test</artifactId>
	<version>${org.springframework.version}</version>
</dependency>
```

```
		<dependency>
			<groupId>org.apache.httpcomponents</groupId>
			<artifactId>httpclient</artifactId>
			<version>4.1.1</version>
			<exclusions>
				<!-- Exclude Commons Logging in favor of SLF4j -->
				<exclusion>
					<groupId>commons-logging</groupId>
					<artifactId>commons-logging</artifactId>
				</exclusion>
			</exclusions>
		</dependency>
		<!-- @Inject -->
		<dependency>
			<groupId>javax.inject</groupId>
			<artifactId>javax.inject</artifactId>
			<version>1</version>
		</dependency>

		<dependency>
			<groupId>org.codehaus.jackson</groupId>
			<artifactId>jackson-core-asl</artifactId>
			<version>1.9.7</version>
		</dependency>
		<dependency>
			<groupId>org.codehaus.jackson</groupId>
			<artifactId>jackson-mapper-asl</artifactId>
			<version>1.9.7</version>
		</dependency>
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
			<version>8.0.13</version>
		</dependency>
		<dependency>
			<groupId>commons-dbcp</groupId>
			<artifactId>commons-dbcp</artifactId>
			<version>1.4</version>
		</dependency>

<!--
		<dependency>
			<groupId>com.google.guava</groupId>
			<artifactId>guava</artifactId>
			<version>r03</version>
		</dependency>
-->
		<dependency>
			<groupId>org.hibernate</groupId>
			<artifactId>hibernate-core</artifactId>
			<version>${org.hibernate.version}</version>
```

```xml
</dependency>
<dependency>
	<groupId>org.hibernate</groupId>
	<artifactId>hibernate-entitymanager</artifactId>
	<version>${org.hibernate.version}</version>
</dependency>

<!-- Logging -->
<dependency>
	<groupId>org.slf4j</groupId>
	<artifactId>slf4j-api</artifactId>
	<version>${org.slf4j.version}</version>
</dependency>
<dependency>
	<groupId>org.slf4j</groupId>
	<artifactId>jcl-over-slf4j</artifactId>
	<version>${org.slf4j.version}</version>
	<scope>runtime</scope>
</dependency>
<dependency>
	<groupId>org.slf4j</groupId>
	<artifactId>slf4j-log4j12</artifactId>
	<version>${org.slf4j.version}</version>
	<scope>runtime</scope>
</dependency>
<dependency>
	<groupId>log4j</groupId>
	<artifactId>log4j</artifactId>
	<version>1.2.16</version>
	<exclusions>
		<exclusion>
			<groupId>javax.mail</groupId>
			<artifactId>mail</artifactId>
		</exclusion>
		<exclusion>
			<groupId>javax.jms</groupId>
			<artifactId>jms</artifactId>
		</exclusion>
		<exclusion>
			<groupId>com.sun.jdmk</groupId>
			<artifactId>jmxtools</artifactId>
		</exclusion>
		<exclusion>
			<groupId>com.sun.jmx</groupId>
			<artifactId>jmxri</artifactId>
		</exclusion>
	</exclusions>
	<scope>compile</scope>
</dependency>
<!-- JSR 303 with Hibernate Validator -->
<dependency>
	<groupId>javax.validation</groupId>
```

```
        <artifactId>validation-api</artifactId>
        <version>1.0.0.GA</version>
</dependency>
<dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-validator</artifactId>
        <version>4.1.0.Final</version>
</dependency>

<!-- Servlet -->
<dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>servlet-api</artifactId>
        <version>2.5</version>
        <scope>provided</scope>
</dependency>
<dependency>
        <groupId>javax.servlet.jsp</groupId>
        <artifactId>jsp-api</artifactId>
        <version>2.1</version>
        <scope>provided</scope>
</dependency>
<dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
</dependency>
<dependency>
        <groupId>taglibs</groupId>
        <artifactId>standard</artifactId>
        <version>1.1.2</version>
        <scope>compile</scope>
</dependency>

<!-- for file upload -->
<dependency>
        <groupId>commons-fileupload</groupId>
        <artifactId>commons-fileupload</artifactId>
        <version>1.2.1</version>
</dependency>
<dependency>
        <groupId>commons-io</groupId>
        <artifactId>commons-io</artifactId>
        <version>1.4</version>
</dependency>

<!-- Test -->
<dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
```

```
		</dependency>

		<!-- easy mock -->
		<dependency>
			<groupId>org.easymock</groupId>
			<artifactId>easymock</artifactId>
			<version>2.4</version>
			<scope>test</scope>
		</dependency>

        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.2.4</version>
        </dependency>

		 <!-- JAX-RS 2.0  -->
		 <dependency>
            <groupId>org.glassfish.jersey.core</groupId>
            <artifactId>jersey-server</artifactId>
            <version>2.7</version>
        </dependency>
        <dependency>
            <groupId>org.glassfish.jersey.containers</groupId>
            <artifactId>jersey-container-servlet-core</artifactId>
            <version>2.7</version>
        </dependency>
        <dependency>
			<groupId>org.glassfish.jersey.containers</groupId>
			<artifactId>jersey-container-jdk-http</artifactId>
			<version>2.2</version>
		 </dependency>
        <dependency>
            <groupId>org.glassfish.jersey.media</groupId>
            <artifactId>jersey-media-json-jackson</artifactId>
            <version>2.7</version>
        </dependency>
        <dependency>
			<groupId>org.glassfish.jersey.media</groupId>
			<artifactId>jersey-media-multipart</artifactId>
			<version>2.7</version>
		 </dependency>
        <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-web-api</artifactId>
            <version>6.0</version>
            <scope>provided</scope>
        </dependency>

        <!-- Needed to integrate JAX-RS 2.0 with Spring.  Without this you can't
inject Spring beans into Jersey objects -->
        <dependency>
		<groupId>org.glassfish.jersey.ext</groupId>
			<artifactId>jersey-spring3</artifactId>
		<version>2.7</version>
		</dependency>

		<dependency>
			<groupId>com.fasterxml.jackson.module</groupId>
			<artifactId>jackson-module-jaxb-annotations</artifactId>
			<version>2.4.1</version>
		</dependency>

		<dependency>
		<groupId>org.apache.httpcomponents</groupId>
			<artifactId>httpclient</artifactId>
			<version>4.3.3</version>
		</dependency>

	</dependencies>

	<build>
		<plugins>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-compiler-plugin</artifactId>
				<configuration>
					<source>1.6</source>
					<target>1.6</target>
				</configuration>
			</plugin>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-dependency-plugin</artifactId>
				<executions>
					<execution>
						<id>install</id>
						<phase>install</phase>
						<goals>
							<goal>sources</goal>
						</goals>
					</execution>
				</executions>
			</plugin>
			<plugin>
		<groupId>org.apache.maven.plugins</groupId>
			<artifactId>maven-surefire-plugin</artifactId>
				<version>2.19.1</version>
				<configuration>
					<testFailureIgnore>true</testFailureIgnore>
				</configuration>
			</plugin>

		</plugins>
```

```
      </build>
</project>
```

Code for Applet Pages:

CustomerHomeApplet:

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.org.me;

import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.Button;
import java.awt.Choice;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.GridLayout;
import java.awt.Image;
import java.awt.Label;
import java.awt.MediaTracker;
import java.awt.Panel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.URL;
import java.net.URLConnection;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.imageio.ImageIO;
import javax.swing.JApplet;
import javax.swing.JComboBox;

/**
 *
 * @author Smitha
 */
public class CustomerHomeApplet extends Applet implements ActionListener {
    String customer1, customer2, customer3, customer4, customer5;
    Button submit, b2;
    Choice customerList = new Choice();
    String name;
    Applet second;
    Label n = new Label("please select your username", Label.CENTER);

    private JComboBox codes1;
    Image backGround;
    Image bgImage = null;
    BufferedImage img = null;
    Image picture, booImage;

    public void init() {
        try {
            picture = ImageIO.read(getClass().getResource("b1.jpg"));
            booImage =
ImageIO.read(getClass().getResource("BookBoo.jpg"));
        } catch (Exception e) {
            System.out.println(e.toString());
        }
        getComboBox2();
    }

    public void getComboBox2() {
        Label n2 = new Label("Welcome to the online Book Store",
Label.CENTER);
        n2.setFont(new java.awt.Font("Arial", Font.BOLD, 40));
        System.out.println("adding items");
        add(n2);
        n2.setBounds(280, 100, 90, 20);
        customerList.add("select your user name");
        add(customerList);

        try {
            customerList.setBounds(280, 900, 90, 120);

            URL link = new
URL("http://localhost:8081/WebApplicationTest2/CustomerLoginServlet");

            URLConnection urlconnection = (URLConnection)
link.openConnection();

            urlconnection.setDoOutput(true);
            urlconnection.setDoInput(true);
            urlconnection.setUseCaches(false);
            urlconnection.setDefaultUseCaches(false);

            // Specify the content type that we will send binary data
            urlconnection.setRequestProperty("Content-Type",
"application/x-java-serialized-object");
```

```
            String lastName = "test1";

            ObjectOutputStream oos = new
ObjectOutputStream(urlconnection.getOutputStream());
            oos.writeObject(lastName);

            ObjectInputStream ois = new
ObjectInputStream(urlconnection.getInputStream());
            List<String> test1 = (List<String>) ois.readObject();
            for (String s : test1) {
                customerList.add(s);
                add(customerList);
            }
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }

        b2 = new Button("SignIn");
        add(b2);
        b2.setBounds(280, 1600, 90, 120);
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.drawImage(picture, 0, 0,
                (int) ((getBounds().getWidth())), (int)
getBounds().getHeight() / 4, this);
        g.drawImage(booImage, 0, 200,
                (int) ((getBounds().getWidth())), ((int)
getBounds().getHeight()) * 3 / 4, this);

    }

    public void actionPerformed(ActionEvent e) {

        destroy();
        Panel container = new Panel();
        container.setLayout(new GridLayout(1, 0));
        submit.setVisible(false);
        n.setVisible(false);
        customerList.setVisible(false);

        this.submit.setVisible(false);
        this.n.setVisible(false);
        this.customerList.setVisible(false);
        destroy();
    }

}
```

CustomerLoginServlet.java

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

import java.io.IOException;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 *
 * @author Smitha
 */
@WebServlet(urlPatterns = {"/CustomerLoginServlet"})
public class CustomerLoginServlet extends HttpServlet {

    private String driverName;
    private String userName;
    private String password;
    private String url;
    private String databaseName;

    private Connection conn;
    private Statement st;

    /**
     * Processes requests for both HTTP <code>GET</code> and
<code>POST</code>
     * methods.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("application/octet-stream");
        InputStream in = request.getInputStream();
        ObjectInputStream inputFromApplet = new ObjectInputStream(in);
        OutputStream outstr = response.getOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(outstr);

        try {
            response.setContentType("application/x-java-serialized-
object");
            String lastName = (String) inputFromApplet.readObject();

            driverName = "com.mysql.jdbc.Driver";
            databaseName = "bookstore";
            userName = "root";
            password = "bookstoremgmt";
            url = "jdbc:mysql://localhost:3306/bookstore";

            Class.forName(driverName);
            System.out.println("init(): getting connection");
            conn = DriverManager.getConnection(url, userName, password);
            st = conn.createStatement();
            ResultSet rs = st.executeQuery("SELECT * FROM CUSTOMER");
            ArrayList<String> usernames = new ArrayList<String>();
            while (rs.next()) {
                Customer c = new Customer(rs.getInt(1), rs.getString(2),
rs.getString(3));
                usernames.add(c.getName());
            }
            oos.writeObject(usernames);

            rs.close();
            st.close();
        } catch (Exception ex) {
            System.out.println(ex.toString());
        } finally {
            oos.flush();
        }

    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods.
Click on the + sign on the left to edit the code.">
```

```
    /**
     * Handles the HTTP <code>GET</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Handles the HTTP <code>POST</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Returns a short description of the servlet.
     *
     * @return a String containing servlet description
     */
    @Override
    public String getServletInfo() {
        return "Short description";
    }// </editor-fold>

}
```

Customer.java

```
import java.io.Serializable;

/*
```

```
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
/**
 *
 * @author Smitha
 */
public class Customer implements Serializable {

    private String name;
    private String state;
    private int id;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getState() {
        return state;
    }

    public void setState(String state) {
        this.state = state;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    /**
     *
     * Default constructor
     */
    public Customer() {

    }

    /**
     * working constructor
     *
     * @param i ID
     * @param l last name
     * @param f first name
     * @param a1 address 1
     * @param a2 address 2
     * @param c city
     * @param s state
     * @param z zip
     * @param p phone
     */
    public Customer(int i, String n, String s) {
        id = i;
        name = n;
        state = s;
    }

    /**
     * Converts object to a string
     *
     * @return String
     */

    @Override
    public String toString() {
        return id + " : " + name + ", " + state;
    }

}
```

Index.jsp

```
<!DOCTYPE html>
<!--
To change this license header, choose License Headers in Project
Properties.
To change this template file, choose Tools | Templates
and open the template in the editor.
-->
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    </head>
    <body>
    <applet code="com.org.me.CustomerHomeApplet"
archive="BookStoreManagementSystemApplet.jar" width="1270"
height="540"></applet>
</body>
</html>
```